Санкт–Петербургский государственный университет

Белоусов Юрий Вячеславович

Выпускная квалификационная работа

Распознавание языка из аудио с помощью глубоких нейронных сетей

Уровень образования: бакалавриат
Направление 02.03.02 «Фундаментальная информатика и информационные технологии»
Основная образовательная программа СВ.5003.2015 «Программирование и информационные технологии»

Научный руководитель:
кафедра теории систем управления электрофизической аппаратурой, доцент, кандидат физ.-мат. наук
Козынченко Владимир Александрович

Рецензент:
кафедра теории управления,
доктор физ.-мат. наук
Котина Елена Дмитриевна

Санкт-Петербург
2019 г.

Содержание

Введение

Данная работа посвящена разработке алгоритма, предназначенного для определения языка, на котором произнесена речь, записанная в аудиодорожке.

Когда мы слышим речь на знакомом для нас языке, то мы практически сразу определяем язык. Перенести эту возможность на автоматические устройства — цель работы. Для данной предметной области характерно, что человеческие возможности ограничены небольшим количеством языков по сравнению с общим количеством различных языков на планете.

Основные вопросы в данной области включают в себя: какие акустические параметры помогают достичь наибольшей точности? Какие алгоритмы, методы и подходы лучше всего работают? Какая зависимость между длиной сигнала и уверенностью в ответе?

Решение этой задачи необходимо для полноценной работы систем распознавания речи, так как в настоящий момент большинство открытых и коммерческих реализаций (Google Cloud Speech-to-Text Api, Microsoft Azure Cognitive Services Speech to Text и т. п.) требуют в качестве параметра явное указание языка. Также данная задача встречается в многоязычных системах перевода, в службах экстренной поддержки, где время ответа оператора, свободно говорящего на языке позвонившего, может быть критично [1].

Работа состоит из нескольких частей. В разделе Постановка задачи задача формулируется в терминах машинного обучения, в разделе Обзор литературы содержится обзор различных подходов к исследуемой задаче, в главе Нейросети и глубокое обучение приводятся теоретические обоснования, лежащие в основе выбранного подхода, в главе Данные: анализ и обработка рассматривается важность грамотного сбора данных и выделения из них факторов, глава Результаты экспериментов включает в себе результаты вычислительных экспериментов, глава Выводы охватывает разбор и анализ результатов, а в главе Заключение подводится итог проведенной работы.

Постановка задачи

Целью выпускной квалификационной работы является решение задачи распознавания языка из аудио.

В данной работе рассматривается подход, заключающийся в постановке *классификационной задачи машинного обучения с учителем*, где в качестве входных данных поступает аудио, а на выходе — категория принадлежности к языку.

Обучение с учителем (supervised learning) — один из способов машинного обучения, в ходе которого построенная система обучается с помощью примеров. Между входными и выходными обучающими сигналами существует некоторая зависимость, но она неизвестна. Требуется найти эту зависимость, то есть построить алгоритм, который для любых входных данных выдаст достаточно точный ответ. Для измерения точности ответов определённым образом вводится функционал качества.

В **задачах классификации** алгоритм предсказывает дискретные значения, соответствующие номерам классов, к которым принадлежат объекты. В обучающем датасете с аудио каждый файл будет иметь соответствующую метку — «английский», «немецкий», «французский» и т. д. Качество алгоритма оценивается тем, насколько точно он может правильно классифицировать новые аудио, то есть по новому файлу определить язык, на котором произнесена речь.

Обзор литературы

Автоматическое распознавание языка — это одна из областей исследований, которая привлекла многих ученых. Эта область изучается достаточно давно, больше 30 лет.

Первые работы в этой области основывались на выделении и определении изменения частот некоторых базовых звуков, которые брались за основу [2]. Условием определения языка было успешное нахождение данных референсных образцов в речи. И хотя такой подход позволял получить некоторые приемлемые результаты, он требовал длительной ручной подборки параметров, а также плохо масштабировался на другие языки.

В 1977 году был предложен метод выделения фонетических факторов с последующей тренировкой скрытых марковских моделей [3]. В дальнейшем развитие этой идеи привело к появлению систем, в основе выделения факторов которых лежит вычисление *i-vector* [4]. Однако для полноценной работы этого подхода требуется достаточно большое обучающее множество.

В последние годы получили распространение подходы, базирующиеся на различных методах машинного обучения, особенно с использованием нейронных сетей [5, 6]. Эти методы способны показывать хороший результат, при этом для обучения не требуется большое количество данных. Именно обзору этих методов посвящена наша работа.

Теория по используемым в работе нейронным сетям подробна описана в [7, 8].

Глава 1. Нейросети и глубокое обучение

1.1 Модель нейрона

В 1943 году была опубликована работа Мак-Каллока и Питтса [9], в которой они представили модель искусственного нейрона.

Нейрон представляет собой объект, получающий на вход n величин $x_1, \ldots, x_n$. Значения этих величин будем трактовать как величины импульсов, поступающих на вход нейрона через n входных синапсов. Далее считается взвешенная сумма этих входных величин с весами $w_1, \ldots, w_n$. Существует также дополнительный вес b, называемый смещением. Для удобства полагают, что $w_0 = b$ и $x_0 = 1$.

Эта взвешенная сумма передается на вход функции активации $f(u)$, а ее значение является выходным значением нейрона.

Таким образом, нейрон вычисляет следующую функцию:

$$y(x) = f(\sum_{i=1}^{n} x_i w_i + b)$$

Схема нейрона изображена на Рис. 1.

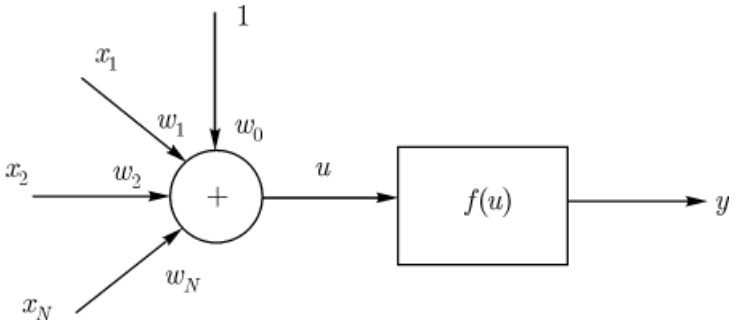


Рис. 1: Схема нейрона.

1.2 Многослойный персептрон

Как можно понять из названия, многослойный персептрон состоит из нескольких слоёв. Каждый слой состоит из нейронов, причём каждый нейрон на предыдущем уровне соединён с нейроном на следующем уровне. Та-

кую многослойную нейросеть называют *Fully-Connected*-сетью или *Dense*-сетью.

Схема примера полносвязной сети изображена на Рис. 2.

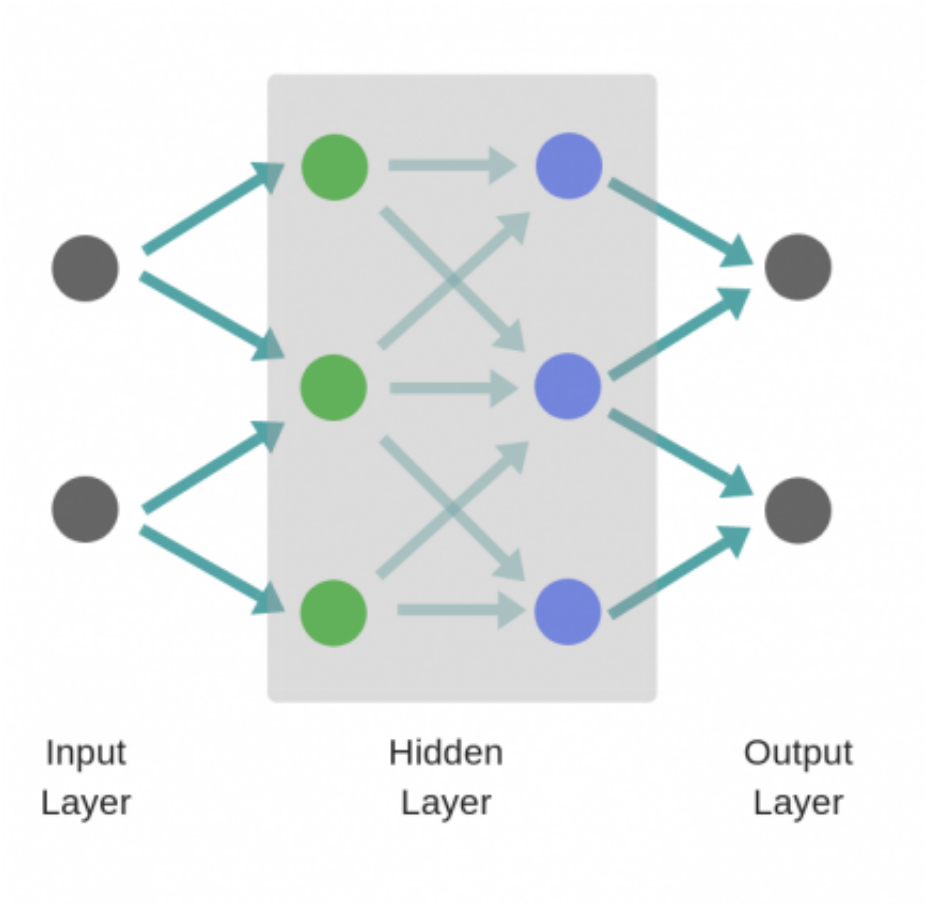


Рис. 2: Схема нейрона.

Стандартная архитектура полносвязной нейросети выглядит следующим образом:

- Входной слой состоит из n нейронов, каждый нейрон соответствует признаку во входных примерах;
- Один или несколько скрытых слоёв, каждый нейрон получает на вход все значения с предыдущего слоя;
- Выходной слой.

От количества нейронов в скрытом слое зависят интерполяционные и аппроксимирующие (обобщающие) свойства сети при решении конкрет-

ной задачи. Если нейронов в скрытом слое будет недостаточно, то сеть не сможет обучиться на обучающем множестве, и ее аппроксимирующие свойства будут плохими. Если нейронов в скрытом слое будет слишком много, то сеть может быть успешно обучена на тренировочном множестве и получит хорошие интерполяционные свойства, но при этом она будет обладать плохими аппроксимирующими свойствами и будет показывать высокую погрешность на данных контрольного множества.

При решении задачи регрессии на выходном слое обычно один нейрон, который возвращает предсказанные числа (для каждого объекта по числу).

В случае задачи K-класовой классификации на выходном слое обычно K нейронов.

1.3 Сверточный слой

Сверточный слой можно рассматривать как набор фильтров, применяемых к изображению. Каждый такой фильтр обучен находить на небольшом фрагменте определенные признаки. Признаками могут являться линии под определенным углом, дуги, кружочки и любые другие незамысловатые объекты. Мы последовательно прикладываем фильтр к участку картинки и определяем, есть ли на нем искомый признак, затем сдвигаем его дальше и повторяем операцию. Такие фильтры называют картами признаков. В сверточных нейросетях такую архитектуру воплощают с помощью *локального восприятия* и *разделяемых весов.*

Локальное восприятие заключается в том, что каждый нейрон слоя связан не со всеми нейронами предыдущего слоя (как в персептроне), а лишь с небольшим участком. Такой участок называют локальным рецептивным полем. То есть на вход каждого нейрона в сверточном слое подается область, ширина и высота которой меньше ширины и высоты оригинального изображения.

Концепция разделяемых весов. При выборе параметров сверточного слоя отдельно выбирается количество ядер, отвечающих за поиск определенного признака. Если веса позволяют найти признак на одном

участке изображения, то эти же веса подойдут для любого другого участка, при условии, что признак мы ищем один и тот же. Таким образом, у всех нейронов одного ядра одинаковый набор весов. Благодаря этому количество настраиваемых параметров существенно уменьшается.

Так как у всех нейронов одной плоскости одни и те же веса, то проход изображения через сверточный слой будет являться его сверткой. При этом ядром будет набор весов. В результате свертки мы получим карты признаков, по одной на каждое ядро свертки. Таким образом, каждое ядро фильтрует изображение, находя определенный признак. Сверточный слой позволяет сети быть устойчивой к изменению масштаба изображений, к сдвигу и повороту. Так же он улучшает обобщающие свойства сети, так как сеть ищет общие признаки, а не запоминает изображения попиксельно, как перцептрон.

Схема сверточного слоя изображена на Рис. 3.

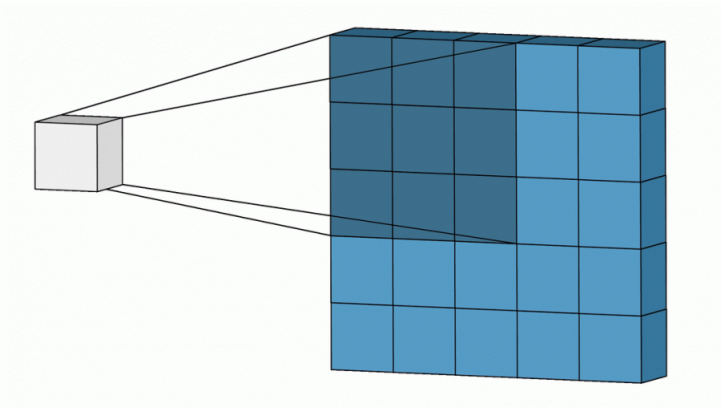


Рис. 3: Схема сверточного слоя.

Часто используемые техники:

- *Padding*. Крайние пиксели никогда не оказываются в центре ядра, потому что тогда ядру не над чем будет скользить за краем. Padding добавляет к краям поддельные пиксели (обычно нулевого значения, вследствие этого к ним применяется термин *нулевое дополнение*). Таким образом, ядро при проскальзывании позволяет неподдельным пикселям оказываться в своем центре, а затем распространяется на поддельные пиксели за пределами края;

- *Striding.* Часто бывает, что при работе со сверточным слоем нужно получить выходные данные меньшего размера, чем входные. Это обычно необходимо в сверточных нейронных сетях, где размер пространственных размеров уменьшается при увеличении количества каналов. Идея stride заключается в том, чтобы пропустить некоторые области, над которыми скользит ядро. Шаг 1 означает, что берутся пролеты через пиксель, то есть по факту каждый пролет является стандартной сверткой. Шаг 2 означает, что пролеты совершаются через каждые два пикселя, шаг 3 означает пропуск 3-х пикселей и т. д.

1.4 Рекуррентный слой

Рекуррентная нейронная сеть (RNN) — вид нейронных сетей, где связи между элементами образуют направленную последовательность.

В традиционных нейронных сетях подразумевается, что все входы и выходы независимы. Но для многих задач это не подходит. Если вы хотите предсказать следующее слово в предложении, лучше учитывать предшествующие ему слова. RNN называются рекуррентными, потому что они выполняют одну и ту же задачу для каждого элемента последовательности, причем выход зависит от предыдущих вычислений. Еще одна интерпретация RNN: это сети, у которых есть «память», которая учитывает предшествующую информацию. Теоретически RNN могут использовать информацию в произвольно длинных последовательностях, но на практике они ограничены лишь несколькими шагами. Схема рекуррентного слоя изображена на Рис. 4.

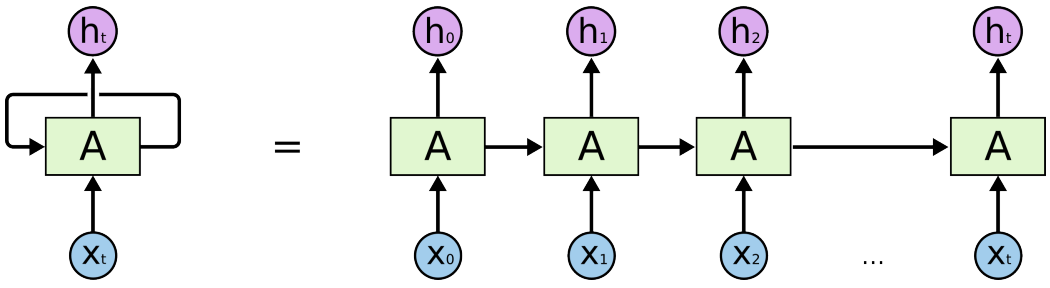


Рис. 4: Рекуррентная нейронная сеть и ее развертка [11].

Долгая краткосрочная память (Long short-term memory; LSTM) –

особая разновидность архитектуры рекуррентных нейронных сетей, способная к обучению долговременным зависимостям [10]. Схема сети долго-краткосрочной памяти изображена на Рис. 5.

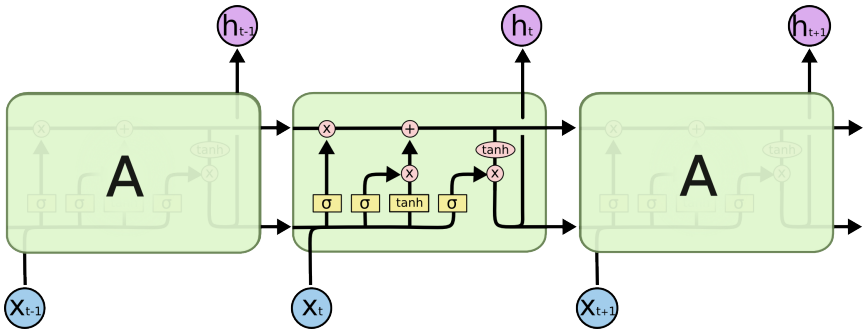
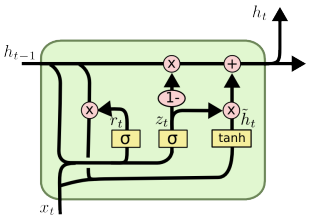


Рис. 5: Схема слоев рекуррентной сети долго-краткосрочной памяти [11].

Немного больше отличаются от стандартных LSTM управляемые рекуррентные блоки (Gated recurrent units, GRU) [12]. В них фильтры «забывания» и входа объединяют в один фильтр «обновления» (update gate). Кроме того, состояние ячейки объединяется со скрытым состоянием. Построенная в результате модель обладает меньшим количеством параметров, чем у LSTM, при этом производительность сопоставима с LSTM. Схема управляемого рекуррентного блока изображена на Рис. 6.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Рис. 6: Схема управляемого рекуррентного блока [11].

1.5 Слой дискретизации

Распространенной практикой является чередование сверточных слоев со слоями дискретизации. Эти слои нужны для уменьшения размера данных и, следовательно, количества настраиваемых параметров. Также данный слой усиливает обобщающие свойства сети.

К каждому локальному полю применяется одна и та же операция. Выбирается участок $f \times f$, и вместо всех значений этого фрагмента оставляют только одно.

В зависимости от выбранной конфигурации слоя оставляют:

- наибольшее значение (MAX-pooling);
- среднее значение (average-pooling);
- максимизированное по норме l_p (l_p-pooling) [13].

Затем выбирается следующий участок путем сдвига на s позиций. И среди значений этого участка также оставляют одно.

На Рис. 7 приведен пример применения дискретизации.

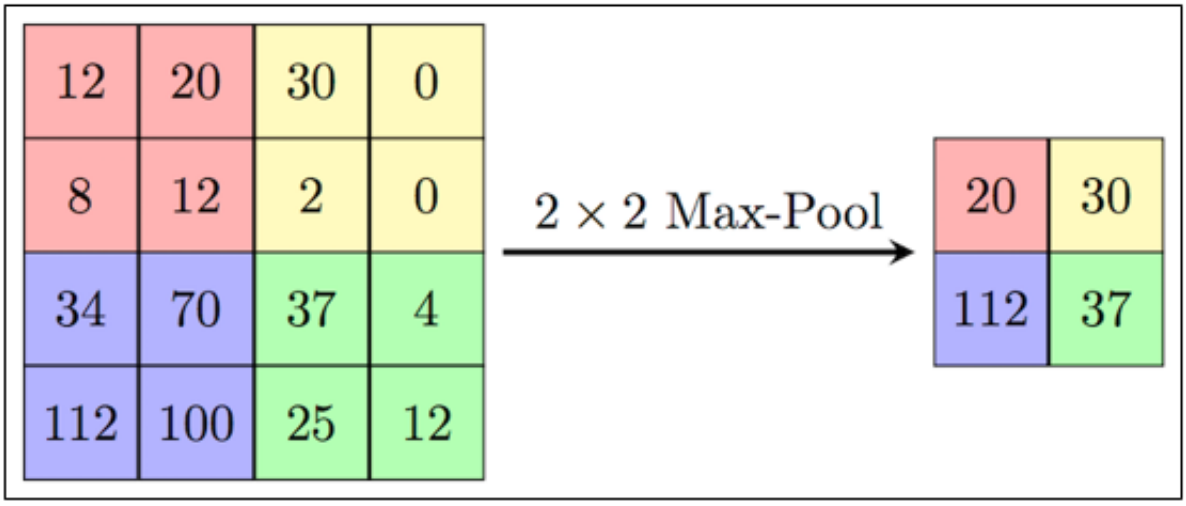


Рис. 7: Формирование новой карты подвыборочного слоя на основе предыдущей карты сверточного слоя. Операция дискретизации (Max Pooling).

1.6 Пакетная нормализация

Пакетная нормализация (batch-normalization) — метод, который позволяет повысить производительность и стабилизировать работу искусственных нейронных сетей [14]. Суть данного метода заключается в том, что некоторым слоям нейронной сети на вход подаются данные, предварительно обработанные и имеющие нулевое математическое ожидание и единичную дисперсию.

Свойства при использовании пакетной нормализации:

- достигается более быстрая сходимость моделей, несмотря на выполнение дополнительных вычислений;
- пакетная нормализация позволяет каждому слою сети обучаться более независимо от других слоев;
- становится возможным использование более высокого темпа обучения, так как пакетная нормализация гарантирует, что выходы узлов нейронной сети не будут иметь слишком больших или малых значений;
- пакетная нормализация в каком-то смысле также является механизмом регуляризации: данный метод привносит в выходы узлов скрытых слоев некоторый шум, аналогично методу dropout;
- модели становятся менее чувствительны к начальной инициализации весов.

1.7 Функция активации

Функция активации определяет выходное значение нейрона в зависимости от результата взвешенной суммы входов и порогового значения. Зачем же нужно вводить эту функцию? В сложных задачах, для решения которых используются глубокие сети, связь между входными и выходными сигналами нелинейная. Таким образом, чтобы ее смоделировать, нужно также использовать нелинейные функции. Для этого после осуществления взвешенного суммирования $\sum_{i=1}^{n} x_i w_i + b$ добавляют нелинейную активационную функцию. Для большинства обучающих алгоритмов к активационной функции предъявляется дополнительное требование непрерывной дифференцируемости на всей числовой оси. В частности, это требование необходимо для использования метода обратного распространения ошибки, применяющегося для обучения сети в данной работе.

На Рис. 8 приведены примеры распространенных функций активации.

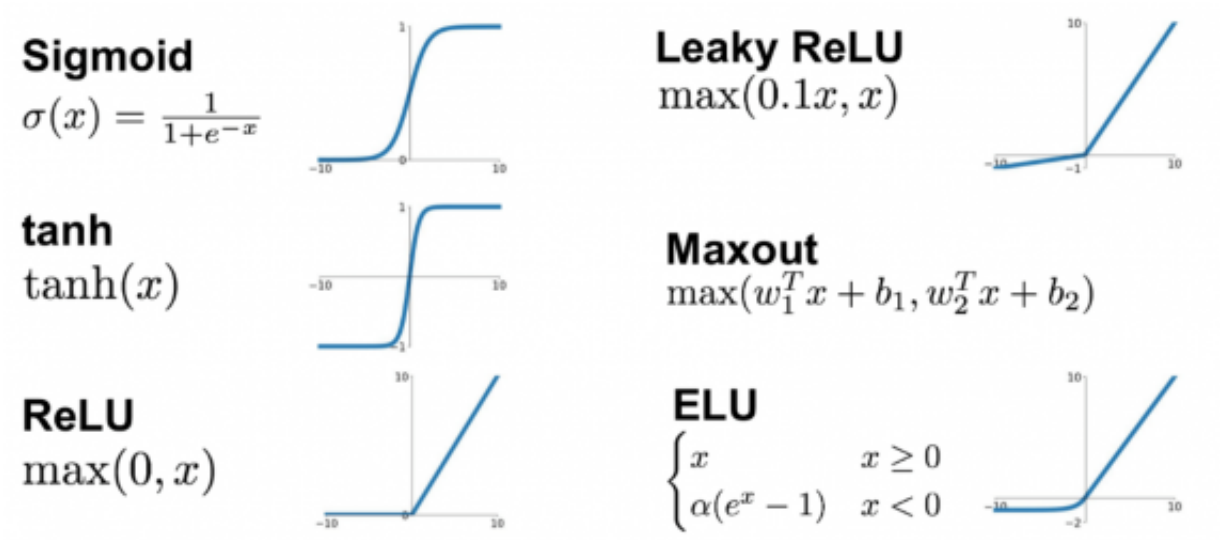


Рис. 8: Распространенные функции активации.

1.8 Функция потерь

Функция потерь — одна из важнейших составляющих любой нейросети. Она используется для расчета ошибки между реальными и полученными ответами. Наша глобальная цель — минимизировать эту ошибку. Таким образом, функция потерь эффективно приближает обучение нейронной сети к этой цели.

Функция потерь измеряет «насколько хороша» нейронная сеть в отношении данной обучающей выборки и ожидаемых ответов. Она также может зависеть от весов нейронной сети.

Некоторые известные функции потерь:

- Квадратичная (среднеквадратичное отклонение): $\sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)}$;
- Кросс-энтропия: $-\sum_x p(x)\log q(x)$;
- Кусочно-линейная функция потерь: $\frac{1}{n}\max(0, 1 - y_j\hat{y}_j)$.

Глава 2. Данные: анализ и обработка

Одним из ключевых этапов на пути решения любой задачи машинного обучения является грамотный выбор и подготовка данных. Не является исключением и решаемая нами задача. Выбор непрезентативного датасета не может гарантировать хорошо работающего в общем случае решения.

Один из примеров не самого удачного выбора данных — контест от TopCoder «SpokenLanguages2» [15]. Предоставленный датасет был основан на записях Библии, при этом для некоторых языков был только один спикер (мужчина в большинстве случаев). Что ещё хуже, так это то, что этот же спикер был и в тестовом множестве. Не исключено, что модель, обученная на таком наборе данных, настроена различать языки по особенностям спикера, а не по особенностям языка [16].

2.1 Описание датасета

В качестве датасета для поставленной задачи выбран датасет *Common Voice* от компании Mozilla [17]. Выбор обусловлен тем, что в настоящее время этот набор данных является крупнейшим мультиязычным общедоступным набором голосовых данных. К тому же при его подготовке участвовало большое количество людей, что приближает этот датасет к реальным условиям. Так, при грамотном разбиении на тренировочную и валидационную выборку, можно предположить, что алгоритм не переобучится путем определения языка по индивидуальным особенностям говорящего, таких как акцент, тембр и высота голоса или условиям записи, а сможет найти закономерности в рассматриваемой задаче.

Выбранные языки и объем использованных данных представлены в Таблице 1.

Названия языка	**Код**	**Число часов**	**Количество голосов**	**Количество дорожек для тренировочной выборки**	**Количество дорожек для валидационной выборки**
Каталанский	ca	92	1639	4000	800
Немецкий	de	140	2249	2629	800
Английский	en	582	33 541	4000	800
Французский	fr	74	2249	4000	800
Китайский	zh-TW	19	695	1240	800

Таблица 1: Характеристика данных

2.2 Разделение на выборки

Разбиение на тренировочную и валидационную выборку произведено следующим образом:

1. Оставлены только проверенные дорожки, достоверность которых подтвердили реальные пользователи.
2. Группировка по $client\_id$ произведена так, чтобы распределение по возрасту, полу, акценту было похожим между тренировочным и валидационным множествами.
3. Для тренировочный выборки отобраны не более 4000 дорожек.
4. Для валидационной выборки взято ровно 800 дорожек.

2.3 Предобработка аудио

Для работы с аудиодорожками была использована библиотека *librosa* [18], предоставляющая удобный интерфейс для работы с аудио.

Алгоритм предобработки аудио выглядит следующим образом:

- загрузка данных из mp3 файла;
- удаление «тишины» по краям дорожки;
- выделение фиксированного фрагмента (во время проведения экспериментов использовались дорожки длительностью $1, 2$, и 4 секунд);
- если исходная дорожка по длительности меньше, чем выделяемый фрагмент, то она дополняется «тишиной» до требуемой длины.

2.4 Выделение факторов из аудио

Для построения хорошо работающего классификатора аудио необходим отбор факторов, которые не зависят напрямую от произносящего, так как любой признак, который дает информацию только о говорящем (например, высота голоса), не будет полезен при классификации. Другими словами, необходимо использовать те факторы, которые отражают в большой степени характерные особенности языка, нежели особенности говорящего. Немаловажно, что эта техника должна отражать особенности человеческого восприятия речи, например, наше восприятие громкости основано на логарифмической шкале. Одним из общепринятых подходов является использование *мел-кепстральных коэффициентов* (*MFCC*) [19] и *мел-спектрограмм*.

Визуализация сигнала, спектрограммы и мел-спектрограммы изображена на Рис. 9.

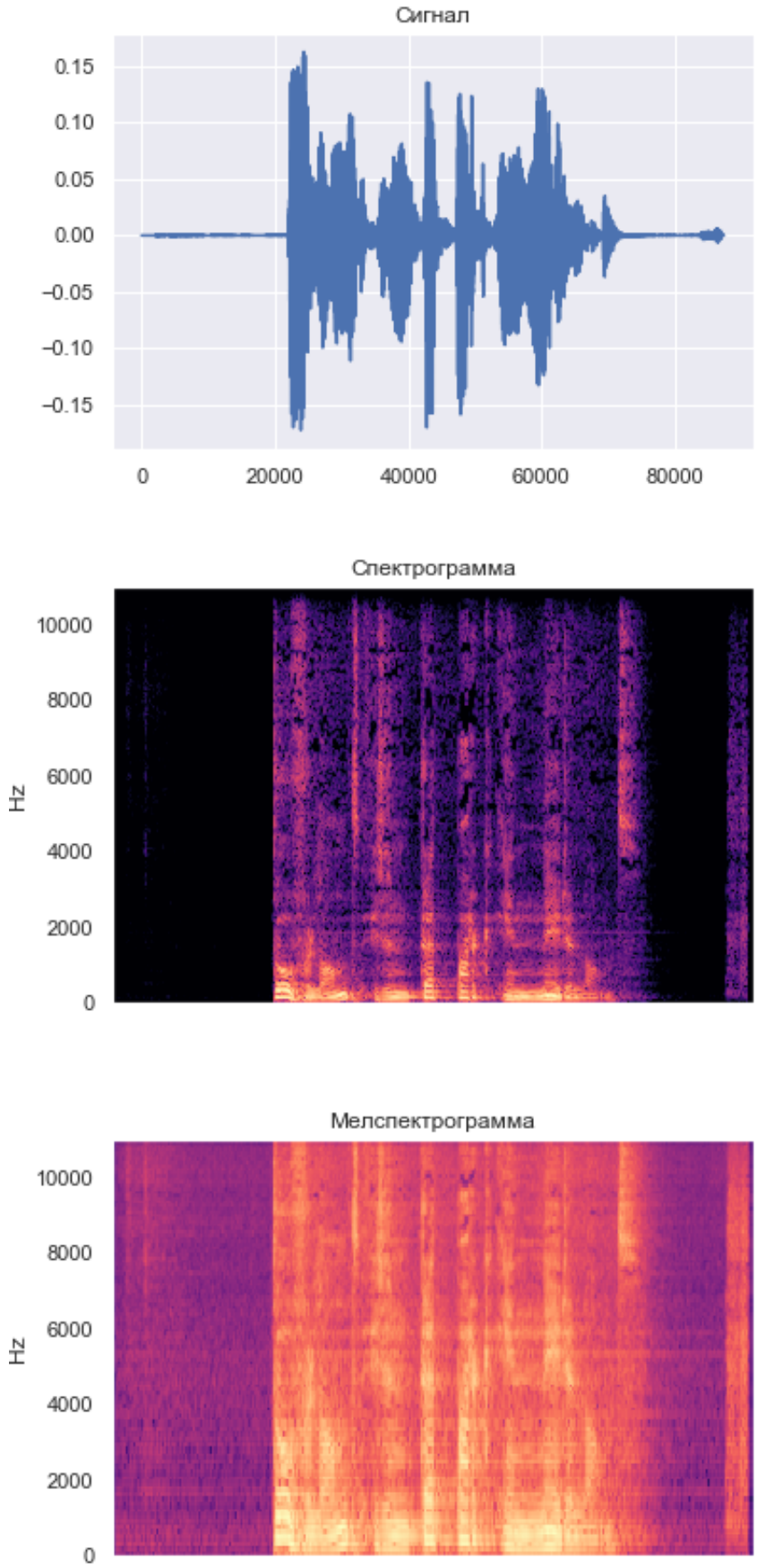


Рис. 9: (Сверху вниз) сигнал, спектрограмма и мел-спектрограмма соответственно.

Глава 3. Результаты экспериментов

3.1 Метрика

В качестве целевой выбрана функция *Accuracy* — доля правильных ответов, т. е. отношение правильно классифицированных аудио к общему количеству аудиодорожек. Однако эта метрика недостаточно устойчива к несбалансированным классам, поэтому также использовались метрики *Precision*, *Recall*, F_1 — точность, полнота и F-мера соответственно.

3.2 Выбор инструментов, имплементация

Для глубокого обучения с использованием GPU существует большое количество фреймворков, например:

- Caffe;
- Pytorch;
- CNTK;
- Keras.

Среди них по ряду причин был выбран фреймворк *Keras* [20]:

- Python интерфейс;
- Простота использования;
- Высокая скорость обучения и предсказания;
- Хорошая документация.

Также для работы с аудио была использована библиотека *Keras Audio Preprocessors* [21], а для мониторинга качества обучения и просмотра графиков ошибки и доли правильных ответов в реальном времени была применена библиотека *TensorBoard* [22].

3.3 Рассматриваемые архитектуры

В качестве решающей функции использовалась нейронная сеть. Ее архитектуру можно разделить на три основных блока:

1. Чтение и преобразование аудио в признаки.
2. Работа с факторами.
3. Слои классификации.

Несомненным преимуществом включения первого блока в модель, в сравнении с предварительным извлечением факторов и сохранением в виде картинок, является гибкость во время проведения экспериментов, так как при изменении одного из параметров нет необходимости в преобразовании данных заново, поскольку преобразование происходит «на лету» во время обучения.

Также для удобства проведения различных экспериментов и подбора различных архитектур была создана базовая модель, включающая в себя:

- Задание конфигурации (размер входных данных, размер выходного слоя и т. д.);
- Построение графика обучения в реальном времени;
- Сохранение весов, при которых достигается наилучшее качество;
- Сохранение архитектуры модели в виде схемы;
- Контроль за переобучением;
- Предсказание на новых данных.

Код базовой модели представлен ниже.

```
from time import time
import numpy as np
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
from keras.utils import plot_model
```

```
class BaseNetwork:
    def __init__(self, name="base_model", input_shape=(1, 64000),
                                          output_dim=5, sample_rate=16000):
        self.model = Sequential()
        self.name = name
        self.input_shape = input_shape
        self.output_dim = output_dim
        self.sample_rate = sample_rate

        self.compiled = False
        self.weights_filepath = f"./models/weights/{self.name}.hdf5"

    def __compile(self):
        self.model.compile(loss='categorical_crossentropy',
                           optimizer='adam',
                           metrics=['accuracy'])

    def train(self, X_train, y_train, X_test, y_test, epochs=100, verbose=
                                          1, shuffle=True, batch_size=512,
                                          patience=10):
        if not self.compiled:
            self.__compile()
            self.compiled = True

        checkpointer = ModelCheckpoint(
            self.weights_filepath, monitor='val_acc', verbose=verbose,
                                                  save_best_only=True)
        tensorboard = TensorBoard(
            log_dir=f'./logs_models/5_lang_{self.name}/{time()}',
                                                  write_graph=True)
        early_stopping = EarlyStopping(monitor='val_acc',  patience=
                                             patience)
        callbacks = [checkpointer, early_stopping, tensorboard]

        history = self.model.fit(X_train, y_train,
                                 epochs=epochs, verbose=verbose,
                                 validation_data=(X_test, y_test),
                                 shuffle=shuffle,
                                 batch_size=batch_size,
                                 callbacks=callbacks)

        return history

    def load_best_weights(self):
        self.model.load_weights(self.weights_filepath)
```

```
    def predict(self, X_test, batch_size=512):
        self.load_best_weights()
        predicted_one_hot = self.model.predict(X_test, batch_size=
                                               batch_size)
        predicted = np.argmax(predicted_one_hot, axis=1)
        return predicted

    def plot_model_arch(self):
        plot_model(
            self.model, to_file=f'model/archs/{self.name}.png',
                                                  show_layer_names=False)
```

Выделение такой базовой модели позволяет сократить код моделей наследников до минимума; например, задание модели, состоящей из слоя, вычисляющего мел-спектрограмму и полносвязных слоев выглядит следующим образом:

```
from models.base_model import BaseNetwork
from keras.layers import Dense, Flatten
from kapre.time_frequency import Melspectrogram

class Dense_MFCC_Model(BaseNetwork):
    def __init__(self):
        super().__init__(name="1D_Pure_Dense_Model")

        self.model.add(Melspectrogram(n_dft=512, input_shape=self.
                                               input_shape,
                       padding='same', sr=self.sample_rate, n_mels=32,
                       fmin=0.0, fmax=5000, power_melgram=1.0,
                       return_decibel_melgram=True, trainable_fb=False,
                       trainable_kernel=False))

        self.model.add(Flatten())
        self.model.add(Dense(128, activation='relu'))
        self.model.add(Dense(64, activation='relu'))
        self.model.add(Dense(self.output_dim, activation='softmax'))
```

3.4 Достигнутые результаты

Были рассмотрены архитектуры различного типа для решения поставленной задачи.

На обучающем множестве в результате экспериментов практически

каждый вариант архитектуры достигал качества в районе $0,9$. Поэтому так важно оценивать результаты на данных, не участвовавших в обучении, чтобы удостовериться, действительно ли модель способна обнаруживать закономерности в поставленной задаче.

На валидационных данных был достигнут результат, представленный в Таблице 2.

Описание архитектуры	**Доля правильных ответов**
2 полносвязных слоя на чистом сигнале	0,23
2 полносвязных слоя на мел-спектрограмме	0,26
1-D свертка на чистом сигнале	0,52
AlexNet на мел-спектрограмме	0,63
GRU после сверточных слоев на мел-спектрограмме	0,81

Таблица 2: Качество на валидационной выборке

Модель оценивалась на говорящих, аудио которых не было включено в тренировочную выборку.

На Рис. 10 приведена архитектура нейронной сети, показавшей наилучшее качество в поставленной задаче.

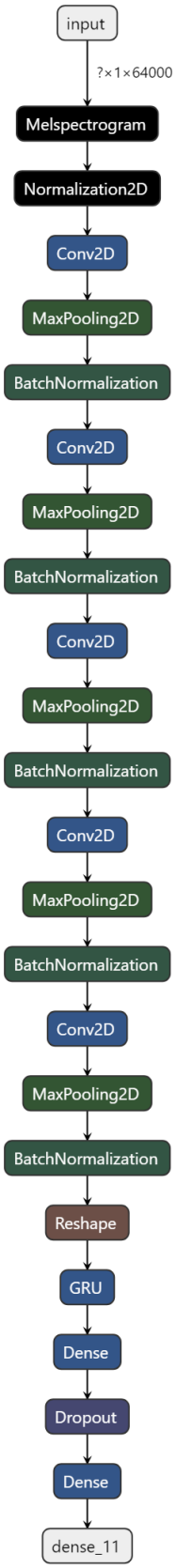


Рис. 10: Архитектура нейронной сети, показавшей наилучшее качество.

Выводы

Исследуя результаты, полученные в результате вычислительных экспериментов, можно сделать следующие выводы:

- Сети, состоящие только из полносвязных слоев, практически не способны выявлять закономерности в поставленной задаче;
- Сверточные слои могут фиксировать общие паттерны изменения частоты в спектрограмме;
- Комбинация сверточных и полносвязных слоев способна увеличивать качество распознавания;
- Грамотная работа с факторами (выделение мел-коэффициентов и т. д.) позволяет достигнуть лучших результатов;
- Добавление рекуррентных слоев к сверточным способно поднять качество ещё выше за счет улавливания переходящих паттернов частот.

Ещё раз стоит подчеркнуть необходимость грамотной работы с данными, выделение непересекающихся и достаточно репрезентативных множеств, построение факторов, способствующих решению задачи.

Заключение

В работе реализованы различные архитектуры нейронных сетей для решения задачи автоматического определения языка. Проведен вычислительный эксперимент, в результате которого было выявлено, что сети, состоящие из сверточных, рекуррентных и полносвязных слоев показывают лучший результат в данной области и более устойчивы к переобучению. Достигается качество, сравнимое с системами, основанными на i-vector [23].

Для улучшения качества распознавания языка следует рассмотреть более глубокие нейронные сети, а также провести эксперименты по классификации для большого количества языков. Возможно, в этом случае стоит использовать иерархическую классификацию.

Список литературы

[1] Muthusamy Y. K., Barnard E., Cole R. A. Reviewing automatic language identification // Signal Processing Magazine. 1994. No 11.

[2] Leonard R., Doddington G. Automatic language identification // Air Force Rome Air Development Center. 1974.

[3] House A., Neuberg E. Toward Automatic Identification of the Languages of an Utterance: Priliminary Methodological Considerations // Journal of the Acoustical Societyof America. 1997. №62(3). P. 708-713.

[4] Dehak N, Kenny P, Dehak R, Dumouchel P, Ouellet P. Front-End Factor Analysis for Speaker Verifica-tion. Audio, Speech, and Language Processing, IEEE Transactions on. 2011 February; 19(4):788–798.

[5] Lopez-Moreno I., Gonzalez-Dominguez J., Plchot O. Automatic language identification using deep neural networks // ICASSP. 2014. P. 5337–5341.

[6] Gonzalez-Dominguez J., Lopez-Moreno I. et al Automatic Language Identification using Long Short-Term Memory Recurrent Neural Networks // Interspeech. 2014.

[7] Schmidhuber J. Deep Learning in Neural Networks: An Overview P. 10-18, 2014 [Электронный ресурс] // arXiv.org. URL: `https://arxiv.org/pdf/1404.7828.pdf` (дата обращения: 10.05.2019).

[8] Christopher M. Bishop. Pattern Recognition and Machine Learning, 2006.

[9] Мак-Каллок У. С., Питтс У. Логическое исчисление идей, относящихся к нервной активности. // Автоматы, под ред. Шеннона К. Э. и Маккарти Дж. М.: ИЛ, 1956. С. 362–384

[10] Hochreiter S., Schmidhuber J. Long Short-Term Memory // Neural Computation. 1997. №8. P. 1735-1780.

[11] Understanding LSTM Networks // colah's blog URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (дата обращения: 10.05.2019).

[12] Kyunghyun C., Merrienboer B. et al Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation // CoRR. 2014. №abs/1406.1078.

[13] Bruna J., Szlam A., LeCun Y. Signal Recovery from l_p Pooling Representations. // Proceedings of the 31st International Conference on Machine Learning (ICML-14), 2014, P. 307–315 URL: http://arxiv.org/pdf/1312.6229.pdf

[14] Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift // CoRR. 2015. №abs/1502.03167.

[15] Contest: Spoken Languages 2 // topcoder. URL: https://community.topcoder.com/longcontest/?module=ViewProblemStatement&rd=16555&pm=13978 (дата обращения: 10.05.2019).

[16] spoken_language_dataset // GitHub URL: https://github.com/tomasz-oponowicz/spoken_language_dataset#background (дата обращения: 10.05.2019).

[17] Mozilla common voice [Электронный ресурс]: URL: https://voice.mozilla.org (дата обращения: 10.03.2019).

[18] McFee B., Raffel C., Liang D. et al. Librosa: audio and music signal analysis in Python // In Proceedings of the 14th Python in science conference. 2015. P. 18–25.

[19] Davis S. B., Mermelstein P. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences // IEEE Transactions on Acoustics, Speech and Signal Processing. 1980. P. 357–366.

[20] Keras: The Python deep learning library [Электронный ресурс]: URL: https://keras.io (дата обращения: 15.03.2019).

[21] Choi K., Deokjin J., Juho K. Kapre: On-GPU audio preprocessing layers for a quick implementation of deep neural network models with Keras // ICML. 2017.

[22] TensorBoard: Visualizing Learning // tensorflow URL: https://www.tensorflow.org/guide/summaries_and_tensorboard (дата обращения: 10.05.2019).

[23] Martinez D., Plchot O. et al. Language Recognition in iVectorsSpace // inINTERSPEECH. 2011, P. 861–864.