МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ
Государственное образовательное учреждение
высшего профессионального образования
«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ»
Физико-технический факультет
Кафедра радиофизики и инфокоммуникационных технологий
Направление подготовки 10.03.01 Информационная безопасность

К защите допустить:
Зав. кафедрой радиофизики и
инфокоммуникационных технологий
_____ д.т.н., проф. В.В. Данилов
«_____» _____ 2021г.

ДИПЛОМНАЯ РАБОТА

на тему: **«Распознавание речи с помощью нейронных сетей»**

Студента: **Мышкин Артем Евгеньевич**

Научный руководитель: ст. преподаватель, **Долбещенкова Н.В. _____**

ст. преподаватель, **Кожекина Е. А._____**

Работа представлена на кафедру «17» 05.2021г. рег. №Д 2021/07 _____

Донецк 2021г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Искусственные нейронные сети стали важной частью нашей жизни и активно используются во всех областях, где традиционные алгоритмические решения работают не так хорошо или же не работают вовсе. Нейронные сети отлично справляются с распознаванием текстов, им доверяют работу по фильтрации спама, игру на биржах и работу с контекстной рекламой в интернете. Одной из наиболее значимых и перспективных сфер, в которой развивается эта технология, является безопасность: здесь речь идет как об отслеживании подозрительных банковских операций, биометрической аутентификации, распознавания речи.

Данная работа посвящена распознаванию речи и преобразованию ее в текст

Цель исследования - исследование параметров рекуррентной нейронной сети для распознавания человека по голосу и преобразования в текст

Объект исследования – рекуррентная нейронная сеть для распознавания человека по голосу

Предмет исследования - модель искусственной нейронной сети для распознавания речи человека

1) Выбор метода и модели

    а) Распознавание человека по голосу

    б) Методы преобразования речевого сообщения в текст

2) Разработка автоматизированной системы обнаружения с преобразованием в текст распознанных голосов

3) Тестирование на реальных голосах, исследование качества распознавания в зависимости от параметров

РАЗДЕЛ 1. НЕЙРОННЫЕ СЕТИ

В этом разделе представлен обзор нейронных сетей, принципы их работы, основные уравнения и функции, стратегии оценки ошибок и обучения. В конце концов, исследуется полезность нейронных сетей для распознавания речи.

1.1. Математическое обоснование

Нейрон. Как и его естественный аналог, искусственный нейрон получает информацию от нескольких других нейронов и генерирует выходное значение, которое затем передается нескольким другим нейронам. Все связи между нейронами характеризуются определенным весом, а нейрон характеризуется функцией, которая выдает выходной сигнал на основе входных данных (Рис. 1.1).

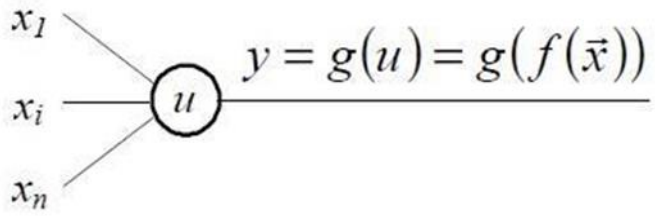

Рисунок 1.1. Искусственный нейрон

Нейронные сети могут различаться по структуре и функциям, которые используются в нейронах. Тип сети, который чаще всего используется при обработке речи, - это сеть прямого распространения с функцией непрерывной активации.

Функция активации рассчитывается так:

$$y = f\left(\sum_{i=1}^{N} w_i\, x_i - \vartheta\right), \qquad (1.1)$$

где N это количество входов, xi это ценность i-th вход, wi это вес связи между xi вход и текущим узлом, где ϑ(тета) - порог нейрона. Аргумент функции активации называется базовой функцией.

Обычно, добавляя нулевой ввод x0 = 1 с весом w0 = - ϑ, мы можем упростить это уравнение до следующего:

$$y = f\left(\sum_{i=0}^{N} w_i\, x_i\right), \qquad (1.2)$$

Некоторые функции активации принимают в качестве аргумента не взвешенное суммирование, а произведение весов и входов или расстояние между вектором входов и вектором весов. Однако эти случаи не будут здесь подробно обсуждаться, поскольку они не так широко используются при распознавании речи, как взвешенное суммирование [1].

Функция непрерывной активации принимает результат базовой функции в качестве аргумента и выдает результат, который находится в непрерывном пространстве между определенными границами. Это зависит от типа функции. Наиболее широко используемые функции непрерывной активации – сигмовидная (рис.1.2) и касательная гипербола (рис.1.3). Функции и их графическое представление следующие:

а) Сигмовидная функция:

$$f(x) = \frac{1}{1+e^{x}}, \quad (1.3)$$

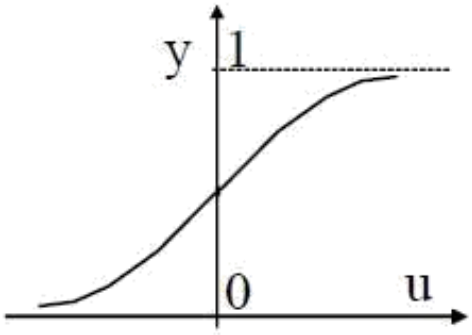

Рисунок 1.2. Сигмовидная функция

б) Касательная функция гиперболы:

$$y = \tan(u), \quad (1.4)$$

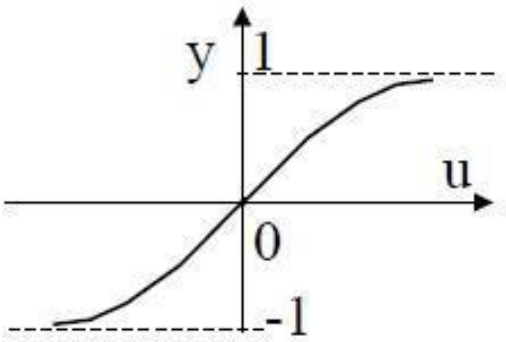

Рисунок 1.3. Касательная функция гиперболы

Как видите, сигмовидная функция может быть удобна в использовании, поскольку ее выходные данные можно интерпретировать как вероятности для классификации. Если используется многоуровневая сеть, все узлы на одном уровне обычно имеют одинаковую функцию активации.

Если мы хотим рассматривать выходы сети как вероятности принадлежности единицы определенному класса, нам нужно, чтобы выходы соответствовали следующим условиям: $0 \leq yi \leq 1$ and $\sum i\ yi = 1$. Таким образом, для выходного слоя функция softmax обычно используется для определения значения j-й вывод:

$$f_j(u_1, u_2, \dots, u_M) = \frac{e^{uj}}{\sum_{K=1}^{M} e^{uk}}, \quad (1.5)$$

где M - количество узлов в выходном слое, и uj является выходом основной функции выходного нейрона.

Для многозадачной нейронной сети softmax необходимо рассчитывать отдельно для каждого блока, поскольку мы рассматриваем каждый блок как отдельную задачу, в которой вероятности принадлежности объекта к определенному классу в этой задаче должны в сумме равняться 1 [2].

Структура сети (рис. 1.4)**.** Что касается структуры нейронных сетей, то здесь подробно будет рассмотрена только сеть прямого распространения. В этом типе сети узлы разделены на подмножества, называемые уровнями, так что никакие соединения не ведут от любого узла к узлу на предыдущем уровне. Более того, сеть прямого распространения является ациклической, что означает отсутствие связи между узлами на одном уровне. Последнее условие: соединения разрешены только между узлом в слое x к узлу в слое x + 1

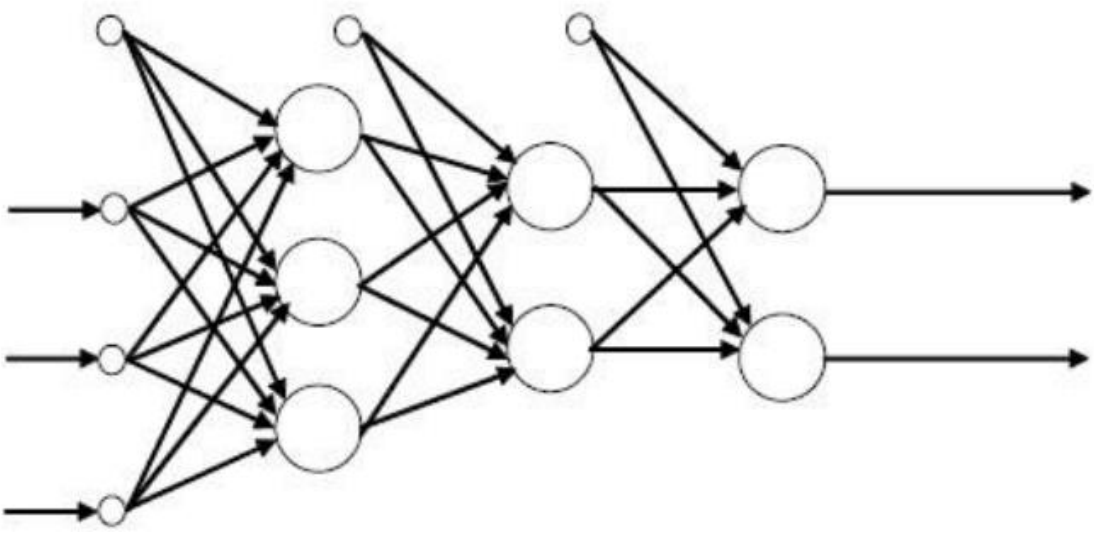

Рисунок 1.4. Сеть прямого распространения

Первый слой в такой сети называется входным слоем, последний - выходным слоем, а все остальные называются скрытыми слоями. Узлы последовательно более высоких уровней абстрагируют объекты более высокого уровня от предыдущих уровней.

Такая архитектура со многими уровнями нелинейных скрытых блоков получила название Deep Neural Network (DNN).

Функция ошибки. Для оценки производительности нужен какой-то критерий. В случае обучения с учителем этим критерием является ошибка между результатом, выданным сетью, и эталонным результатом. Затем мы можем обучить сеть минимизировать эту ошибку.

Функция ошибок для выходного слоя выглядит так:

$$E(Y, D) = \sum_{p=0}^{P} \varepsilon_p \, (y_p, d_p) \qquad (1.6)$$

где Y вектор выходов, D - вектор целей, y_p и d_p являются элементами Y и D соответственно и ε в мгновенной ошибке.

В нейронных сетях используются два типа мгновенных ошибок: среднеквадратическая ошибка и кросс-энтропия.

а) Среднеквадратичная ошибка:

$$\varepsilon_p = \frac{1}{2}\sum_{j=1}^{M}(jd_p - jy_p)^2, \qquad (1.7)$$

Среднеквадратичная ошибка обычно используется для задач регрессии.

б) Кросс-энтропия:

$$\varepsilon_p = -\sum_{j=1}^{M} jd_p \ln(jy_p), \quad (1.8)$$

Кросс-энтропия обычно используется для задач классификации.

Метод Градиентного Спуска. Градиентный спуск — это метод итеративного поиска минимума функции ошибок. Если функция ошибок представлена в многомерном пространстве, где одно из измерений является значением ошибки, а все остальные - параметрами модели, то конкретная комбинация параметров соответствует только одной точке функции. Для этой точки найдено направление, в котором ошибка уменьшается быстрее всего (рис. 1.5). Для этого мы вычисляем градиент, который показывает направление, в котором ошибка увеличивается быстрее всего, а это как раз направление, противоположное тому, которое нам нужно.

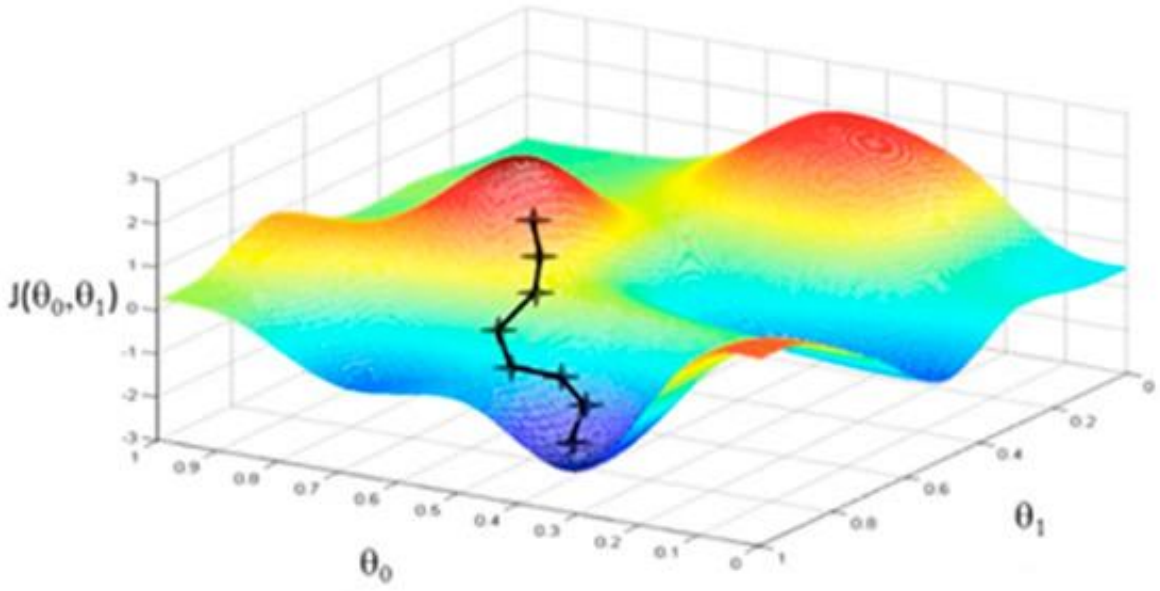

Рисунок 1.5. Градиентный спуск в 3D

Градиент — это вектор первых выводов ошибки E по пространственным измерениям:

$$\nabla E = [\frac{\partial E}{\partial W_1}, \frac{\partial E}{\partial W_2}, \dots, \frac{\partial E}{\partial W_M}]^T, \quad (1.9)$$

После вычисления градиента одна итерация градиентного спуска определяется следующим образом:

$$w(t+1) = w(t) - \mu \nabla E, \quad (1.10)$$

где μ - коэффициент обучения, определяющий, насколько быстро функция сходится к минимуму.

Если μ слишком мал, обучение занимает много времени и может вообще не сходиться. Если он слишком большой, тренировка может упустить оптимальный минимум. μ может быть постоянным на протяжении всего обучения, но более эффективный подход - постепенно уменьшать его.

С этой целью обычно используется процедура уменьшения вдвое. Идея уменьшения вдвое заключается в том, что, если точность набора перекрестной проверки улучшается за одну итерацию на значение, меньшее некоторого порога, коэффициент обучения делится на два. Это помогает в конечном итоге сделать тренировку более точной. Если на каком-то шаге точность становится хуже, чем после предыдущего шага, веса меняются местами.

Если функция ошибок не является выпуклой, спуск градиента может находить разные минимумы в зависимости от разных инициализаций.

Метод обратного распространения ошибки. После получения ошибки для определенной гипотезы, сгенерированной текущим состоянием нейронной сети, мы хотим, чтобы нейронная сеть училась и улучшала свою производительность. Обучение нейронной сети означает обновление весов связей между элементами в разных слоях. Чтобы вычислить, как нужно обновлять веса, используется алгоритм обратного распространения ошибки. Он возвращается к каждому узлу в каждом слое и вычисляет его долю в общей ошибке последнего слоя.

Для каждого узла в слое веса обновляются следующим образом:

$$\nabla w_i = -\mu \frac{\partial E}{\partial w_i}, \tag{1.11}$$

Вычислять E, необходимо вычислить мгновенную ошибку каждого узла. Следующая формула показывает вычисление ошибки для узла я в слое п-1:

$$\varepsilon_j^{n-1} = \sum_{j=1}^{M} \varepsilon_j^n f'(u_j^n) w_j^i, \tag{1.12}$$

Где Ɛ это ошибка узла j в слое п, f ' вывод функции активации узла j в слое N и является значением массы между узлом i и узлом j

Таким образом, ошибка сначала вычисляется для выходного слоя, затем для последнего скрытого слоя и вплоть до первого скрытого слоя. После того, как все ошибки известны, каждый вес сети обновляется, и следующая итерация обучения начинается с подачи новых обучающих данных в сеть для прямого распространения.

На следующем рисунке (рис. 1.6) показано, как происходит обратное распространение. Вот, δ это ошибка, g это функция активации, z это основная функция и θ это веса. В выходном слое, a - выходное значение и y это востребованная стоимость. Весь расчет записывается в векторном виде для всех значений в одном слое.

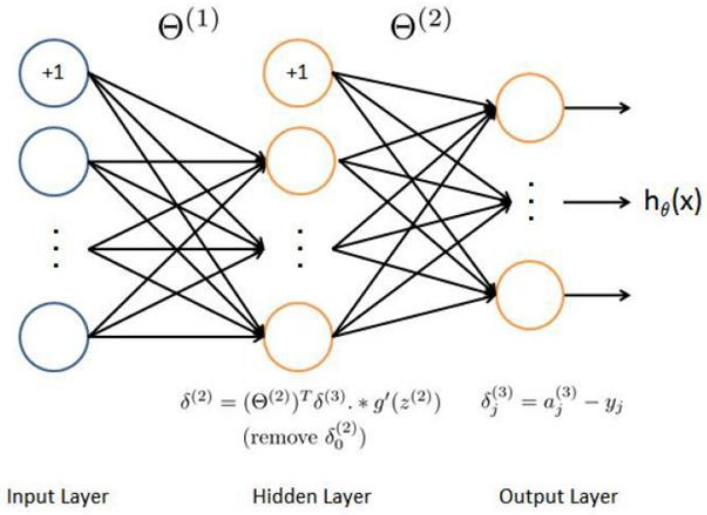

Рисунок 1.6. Обратное распространение

1.2. Нейронные сети для распознавания речи

Распознавание речи основано на концепции моделирования: акустические и языковые модели являются основными частями большинства систем распознавания речи. Акустическая модель - это статистическое представление фонемы или трифона, которая является фонемой с левым и правым контекстом. Искусственные нейронные сети, особенно глубокие нейронные сети (DNN), которые описаны в 2.2, широко используются для акустического моделирования, и за ними обычно следует система скрытых марковских моделей (HMM).

Таким образом, обычно структура DNN, используемая для распознавания речи, следующая: входной уровень сети получает функции, извлеченные из каждого кадра высказывания, а выходной уровень обычно представляет фонемы или трифоны или состояния, в зависимости от настройки. В контексте DNN-HMM

При акустическом моделировании задача сети состоит в том, чтобы вычислить вероятности, которые могут быть использованы для вероятностей эмиссии HMM.

Иногда нейронные сети также используются для предоставления векторов признаков для моделей гауссовой смеси в системе GMM-HMM. Наиболее распространенный подход состоит в том, чтобы активировать скрытые узкие места как функции, что известно, как тандемный подход. Более сложный и эффективный подход использует составную структуру узких мест для генерации функций.

DNN также оказались эффективными для обнаружения субфонетических речевых атрибутов (артикуляционных особенностей). Всего 22 атрибута, характеризующих манеру и место артикуляции, а также дополнительные признаки фонем были обнаружены с более чем 90% успехом с помощью DNN, содержащего 5 скрытых слоев по 2048 на каждом.

Многозадачные нейронные сети. Идея многозадачного обучения заключается в том, что иногда выгоднее изучать несколько вещей одновременно, чем использовать для них отдельные нейронные сети. При многозадачном обучении сеть обучается выполнять как основную задачу классификации, так и одну или несколько вторичных задач с использованием общего представления (рис. 1.7 и 1.8). Обратное распространение можно использовать для обучения искусственных нейронных сетей выполнению этих задач.

На вопрос, как задачи влияют друг на друга, может быть три ответа. Во-первых, некоррелированные задачи могут действовать как источник шума, который иногда может улучшить обобщение при добавлении к обратному распространению. Во-вторых, добавление задач изменяет динамику обновления

веса, так что обучение становится более эффективным, если задачи связаны между собой. Третья возможность - это чистая емкость; многозадачные сети разделяют скрытый слой между всеми задачами, а уменьшенная пропускная способность улучшает обобщение этих проблем.

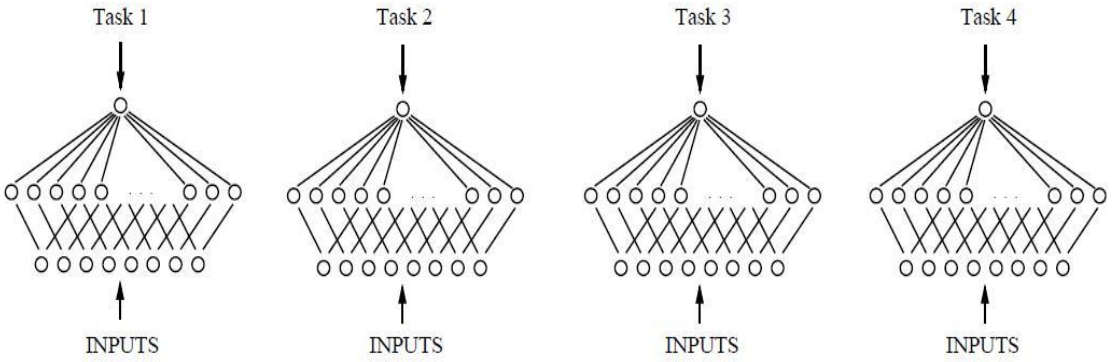

Рисунок 1.7. Отдельные сети для каждой задачи

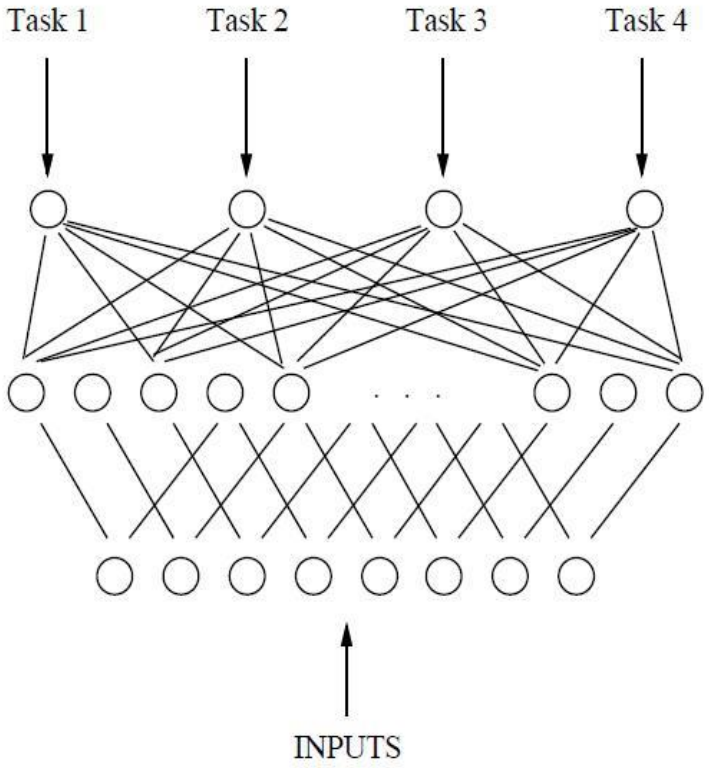

Рисунок 1.8. Многозадачная нейронная сеть

С добавлением второстепенных задач нейронная сеть не должна изменять свою структуру, за исключением размера выходного слоя. Дополнительные параметры в сети, связанные со второстепенными задачами, используются

только для помощи в обучении сети. После завершения обучения часть сети, связанная со второстепенными задачами, отбрасывается, и классификация выполняется идентично традиционному классификатору одной задачи.

Эксперименты показали, что, если задачи связаны, производительность увеличивается при добавлении большего количества задач [3]. Если задачи связаны, вполне вероятно, что вычисленные подкомпоненты, необходимые для некоторых задач, иногда будут полезны для других задач. Эти вспомогательные функции могут быть недоступны для изучения задачами, которые «подслушивают» их, если они обучаются отдельно, но могут оказаться ценными источниками информации, когда они будут изучены для других задач.

Главный недостаток многозадачных нейронных сетей - сложность выбора второстепенных задач. Термин «связанный» далек от формального определения, и хотя можно сделать разумные предположения об эффективности какой-либо другой второстепенной задачи, невозможно быть уверенным, что это окажется правдой. Так что единственный способ выбрать второстепенные задачи — это экспериментировать.

Многозадачные нейронные сети в распознавании речи для распознавания речи многозадачная структура открывает бесчисленные возможности использования, так как многие речевые характеристики взаимозависимы. Многозадачные нейронные сети не новы, и эксперименты с ними проводились с 1989 года, когда классическое приложение NETtalk использовало одну сеть для изучения фонем и их напряжений [3]. Но это только одна из множества возможных комбинаций задач, которые могут улучшить распознавание речи. Некоторые из возможных настроек могут включать, например, совместное изучение сегментных и супрасегментных характеристик (например, тонов в тональных языках), обозначений фонем и характеристик фонем или инвентаризации фонем различных языков в многоязычной задаче. В недавней статье [3] были исследованы следующие второстепенные задачи: метка телефона, контекст телефона и контекст состояния.

1.3. Артикуляционные характеристики фонемы

При работе с продуктами человеческой деятельности, которыми является речь, всегда полезно понимать, как эта деятельность выглядит. Производство речи находится в компетенции артикуляционной фонетики, которая имеет дело с физиологической природой речи.

Речь создается людьми с помощью сложной системы, называемой голосовым трактом. Эта система имеет источник звука (воздух, выдыхаемый легкими, проходит через вибрирующие голосовые связки и производит звук) и ряд резонаторов, которые можно настраивать и изменять для создания различных звуковых эффектов. Таким образом, положение различных частей речевого тракта определяет воспроизводимый звук, а зависимость между конфигурацией речевого тракта и акустическими характеристиками звука более или менее прямая. Из этого знания возникает классификация звуков речи в соответствии с конфигурацией и активностью различных частей речевого тракта при их произнесении. Рассмотрим некоторые из этих артикуляционных характеристик, поскольку они будут использоваться в дальнейших экспериментах.

Сопротивление гласных / согласных самая очевидная артикуляционная характеристика звука, которую распознают даже наивные говорящие, - гласная это или согласная. В случае согласной звук артикулируется путем полного или частичного закрытия речевого тракта, тогда как при произнесении гласного речевой тракт открывается, и нет идентифицируемого места наибольшего давления.

Гласные и согласные, будучи такими разными, также характеризуются по-разному. Согласные обычно определяются местом и манерой артикуляции и звучания. Гласные отличаются высотой, спиной и округлостью. Начнем с обсуждения артикуляционных характеристик согласных, поскольку они лежат в основе экспериментов. Затем мы переходим к артикуляционным характеристикам гласным.

Характеристики согласных характеристики согласных для формирования второстепенных задач в экспериментах, поэтому они обсуждаются здесь более подробно, чем артикуляционные характеристики гласных [4].

Место артикуляции — это место, где создается препятствие при произнесении согласного. Это препятствие находится между пассивным участком, которым является некоторая часть верхней задней части неба во рту, и активным артикулятором, которым является нижняя поверхность рта. На рисунке 1.9 перечислены возможные места сочленения, как активные, так и пассивные. На рисунке 1.10 показаны возможные взаимодействия между этими частями и звуки, которые они вызывают.

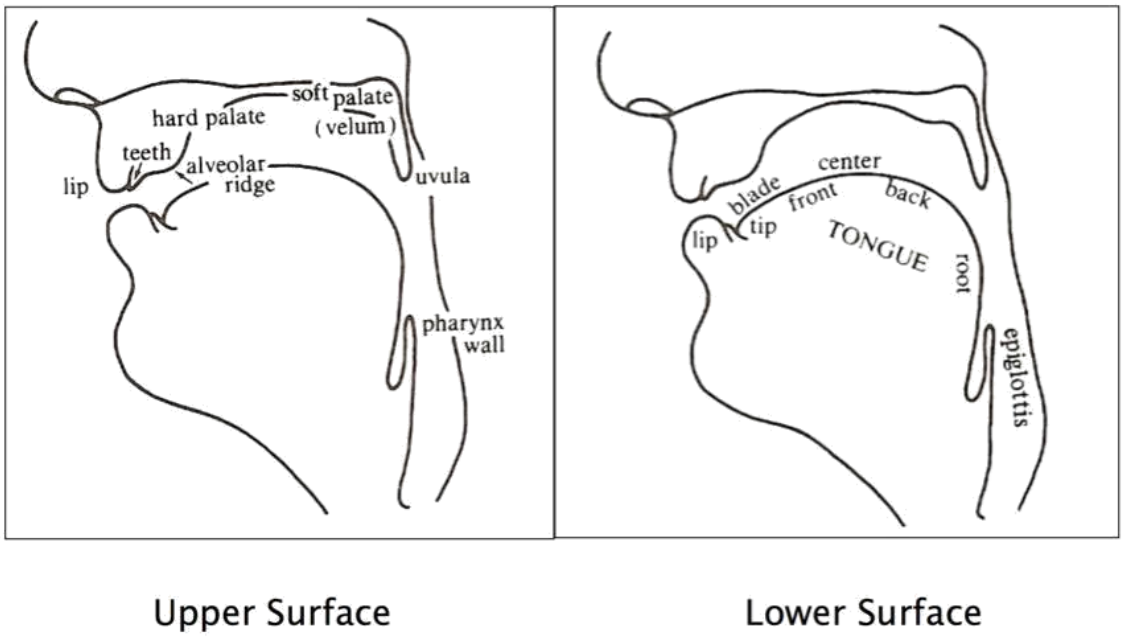

Рисунок 1.9. Места сочленения

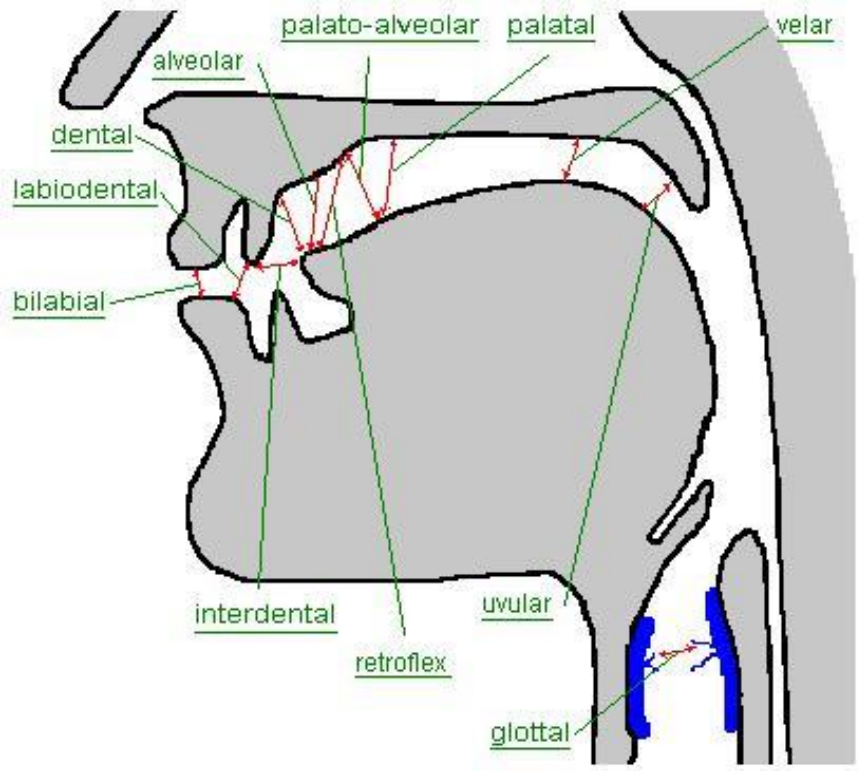

Рисунок 1.10. Типы согласных по месту артикуляции

Классификация согласных по их артикуляционным характеристикам. Артикуляционные характеристики обеспечивают отличную основу для классификации фонем, поэтому было предпринято множество попыток стандартизировать эту классификацию. Это удалось IPA (Международной фонетической ассоциации). Их классификация теперь является международным стандартом для фонетистов всего мира. Таблица согласных звуков МФА, использованная в экспериментах в данной работе, представлена на рисунке. Сюда входят только легочные согласные, но, поскольку в этой работе мы не встретим никаких внелегочных, этого будет достаточно (рис. 1.11).

Способ артикуляции		Место артикуляции											Участие голоса и шума
		Губные				Язычные							
		губно-губные		губно-зубные		переднеязычные				средне-язычные	заднее-язычные		
						зубные		передне-нёбные					
Смычные		[б]	[б']			[д]	[д']				[г]	[г']	зв.
		[п]	[п']			[т]	[т']				[к]	[к']	гл.
Щелевые				[в]	[в']	[з]	[з']	[ж]	[$\overline{ж}$']	[j]	([γ])	([γ'])	зв.
				[ф]	[ф']	[с]	[с']	[ш]	[$\overline{ш}$']		[х]	[х']	гл.
Смычно-щелевые						[ц]			[ч']				гл.
Смычно-проходные	носовые	[м]	[м']			[н]	[н']						сонорные
	боковые					[л]	[л']						
Дрожащие								[р]	[р']				
		тв.	м.	тв.	м.	тв.	м.	тв.	м.	палат.	тв.	м.	
		Отсутствие или наличие палатализации											

Рисунок 1.11 Классификация согласных звуков

1.4. Супрасегментарные характеристики

Супрасегментные характеристики — это элементы речи, которые могут сосуществовать с несколькими сегментами (фонемами) и не могут быть дискретно упорядочены ими. Некоторые примеры суперсегментных характеристик - ударение, тон и интонация [5].

Тон характеризуется использованием высоты звука для различения слов и словоформ. В тональных языках один и тот же слог с другим тоном имеет разное значение. Некоторые из экспериментов в этой работе проводятся на вьетнамском наборе данных, а вьетнамский - это тональный язык с 6 тонами: средний уровень, низкий падающий, низкий подъем, высокий сломанный, высокий подъем и низкий сломанный (рис. 1.12). Высота тона сильно влияет на акустические характеристики фонемы, поэтому второстепенная задача тона была выбрана в качестве одной из второстепенных задач на вьетнамском наборе данных.

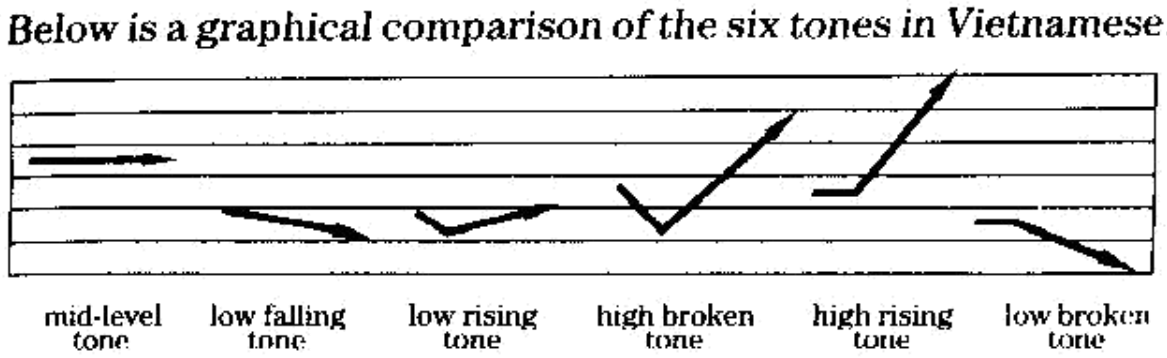

Рисунок 1.12. Вьетнамские тона

Коартикуляция — это процесс, происходящий в непрерывной речи, в котором артикуляция фонемы находится под влиянием предшествующей или последующей фонемы и становится более похожей на нее. Коартикуляция обычно затрагивает место артикуляции, так как движение голосового тракта непрерывно и в процессе переключения между двумя фонемами голосовой тракт неизбежно оказывается в каком-то среднем положении между двумя артикуляциями. Результатом эффекта коартикуляции является то, что фонемы могут различаться в зависимости от контекста, то есть от предшествующих и последующих фонем [1].

Поскольку для наших экспериментов были выбраны независимые от контекста метки фонем, что означает, что эффекты коартикуляции не были представлены в этой модели, контекстная информация была одной из второстепенных задач, выбранных для многозадачных экспериментов. Это уже было полезно в такой функции [2].

РАЗДЕЛ 2. РЕКУРРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ

Рекуррентные нейронные сети (RNN) — это тип нейронных сетей, которые специализируются на обработке последовательностей. Зачастую их используют в таких задачах, как обработка естественного языка (Natural Language Processing) из-за их эффективности в анализе текста. В данной статье мы наглядно рассмотрим рекуррентные нейронные сети, поймем принцип их работы, а также создадим одну сеть в Python, используя numpy [6].

Один из нюансов работы с нейронными сетями (а также CNN) заключается в том, что они работают с предварительно заданными параметрами. Они принимают входные данные с фиксированными размерами и выводят результат, который также является фиксированным. Плюс рекуррентных нейронных сетей, или RNN, в том, что они обеспечивают последовательности с вариативными длинами как для входа, так и для вывода. Вот несколько примеров того, как может выглядеть рекуррентная нейронная сеть (рис. 2.1)

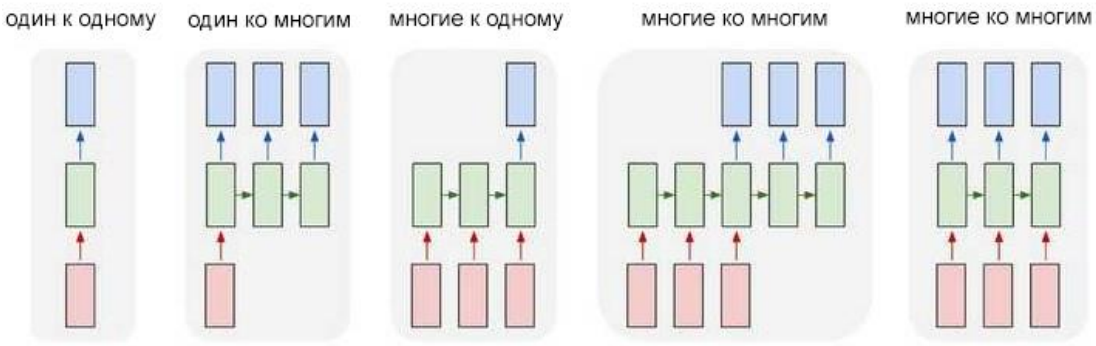

Рисунок 2.1. Виды рекуррентной нейронной сети

Входные данные отмечены красным, нейронная сеть RNN — зеленым, а вывод — синим.

Способность обрабатывать последовательности делает рекуррентные нейронные сети RNN весьма полезными. Области использования:

Машинный перевод (пример Google Translate) выполняется при помощи нейронных сетей с принципом «многие ко многим». Оригинальная

последовательность текста подается в рекуррентную нейронную сеть, которая затем создает переведенный текст в качестве результата вывода;

Анализ часто выполняется при помощи рекуррентных нейронных сетей с принципом «многие к одному». Такая постановка является одним из примеров анализа настроений. Анализируемый текст подается в нейронную сеть, которая затем создает единственную классификацию вывода. Например — Этот отзыв положительный.

Рецидивирующий нейронная сеть (РНН) является классом искусственных нейронных сетей, где соединение между узлами образуют ориентированный граф вдоль временной последовательности. Это позволяет ему демонстрировать динамическое поведение во времени. Полученные из нейронных сетей прямого распространения, RNN могут использовать свое внутреннее состояние (память) для обработки последовательностей входных данных переменной длины. Это делает их применимыми для таких задач, как несегментированное связанное распознавание рукописного ввода или распознавание речи [7].

Термин «рекуррентная нейронная сеть» используется без разбора для обозначения двух широких классов сетей с аналогичной общей структурой, где одна - конечный импульс, а другой - бесконечный импульс. Оба класса сетей демонстрируют временное динамическое поведение. Конечная импульсная рекуррентная сеть - это направленный ациклический граф, который можно развернуть и заменить нейронной сетью со строго прямой связью, в то время как бесконечная импульсная рекуррентная сеть - это направленный циклический граф, который нельзя развернуть. Как конечные импульсные, так и бесконечные импульсные рекуррентные сети могут иметь дополнительные сохраненные состояния, и хранилище может находиться под прямым контролем нейронной сети. Хранилище также может быть заменено другой сетью или графиком, если он включает временные задержки или имеет петли обратной связи. Такие контролируемые состояния называются стробированными состояниями или стробированной памятью и являются частью сетей долговременной краткосрочной памяти (LSTM) и стробированных рекуррентных единиц.

2.1. Виды архитектуры RNN

Долгая кратковременная память (LSTM) - это система глубокого обучения, которая позволяет избежать проблемы исчезающего градиента. LSTM предотвращает исчезновение или распространение ошибок с обратным распространением. Вместо этого ошибки могут течь в обратном направлении через неограниченное количество виртуальных слоев, развернутых в пространстве. То есть LSTM может изучать задачи, требующие воспоминаний о событиях, произошедших на тысячи или даже миллионы дискретных временных шагов ранее. Топологии, подобные LSTM, могут быть разработаны для конкретных задач. (рис. 2.2)

LSTM работает даже при длительных задержках между важными событиями и может обрабатывать сигналы, которые смешивают низкочастотные и высокочастотные компоненты. Многие приложения используют стеки LSTM RNN и обучают их с помощью временной классификации Connectionist (CTC), чтобы найти весовую матрицу RNN, которая максимизирует вероятность последовательностей меток в обучающем наборе с учетом соответствующих входных последовательностей. CTC добивается как согласованности, так и признания. LSTM может научиться распознавать контекстно-зависимые языки, в отличие от предыдущих моделей, основанных на скрытых марковских моделях (HMM) и аналогичных концепциях. (рис. 2.3)

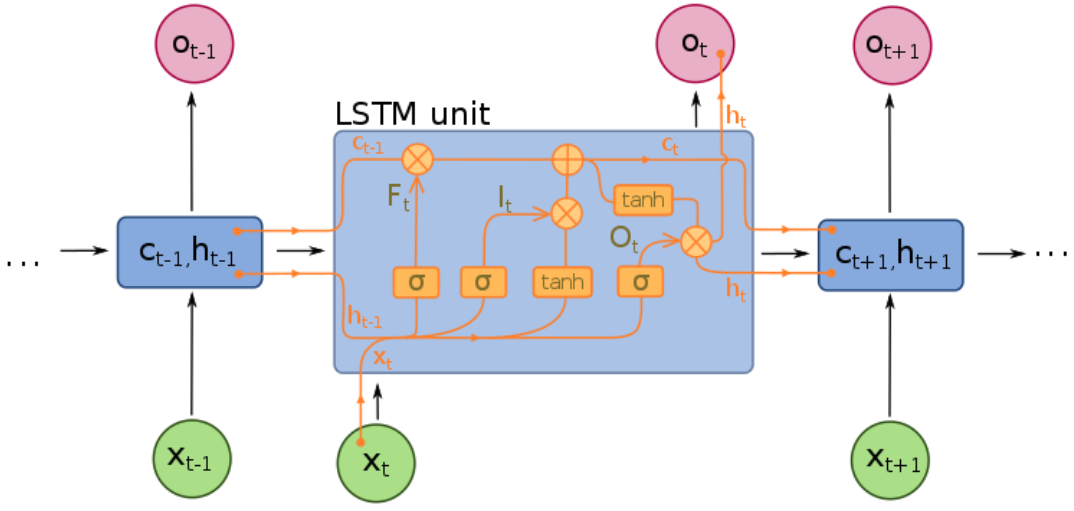

Рисунок 2.2. Блок долговременной кратковременной памяти

Полностью повторяющийся. Полностью рекуррентные нейронные сети (FRNN) соединяют выходы всех нейронов со входами всех нейронов. Это наиболее общая топология нейронной сети, потому что все другие топологии могут быть представлены путем установки весов некоторых соединений на ноль для имитации отсутствия соединений между этими нейронами.

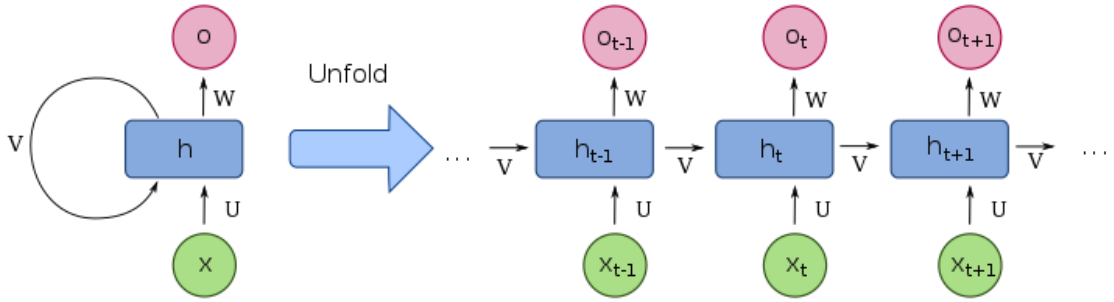

Рисунок 2.3. Развернутая базовая рекуррентная нейронная сеть

Рисунок может ввести в заблуждение многих, потому что практические топологии нейронных сетей часто организованы в «слои», и рисунок дает такой вид. Однако то, что кажется слоями, на самом деле представляет собой разные шаги во времени одной и той же полностью рекуррентной нейронной сети. Крайний левый элемент на иллюстрации показывает повторяющиеся соединения в виде дуги с надписью «v». Он «разворачивается» во времени, чтобы создать вид слоев.

Сети Элмана и Иорданские сети. Эльман сеть представляет собой трехслойную сеть (расположены горизонтально, как x, y и г на рисунке) с добавлением множества контекстов единиц (U на рисунке). Средний (скрытый) уровень связан с этими блоками контекста с весом, равным единице. На каждом временном шаге входные данные передаются вперед, и применяется правило обучения. Фиксированные обратные соединения сохраняют копию предыдущих значений скрытых единиц в единицах контекста (поскольку они распространяются по соединениям до применения правила обучения). (рис. 2.4)

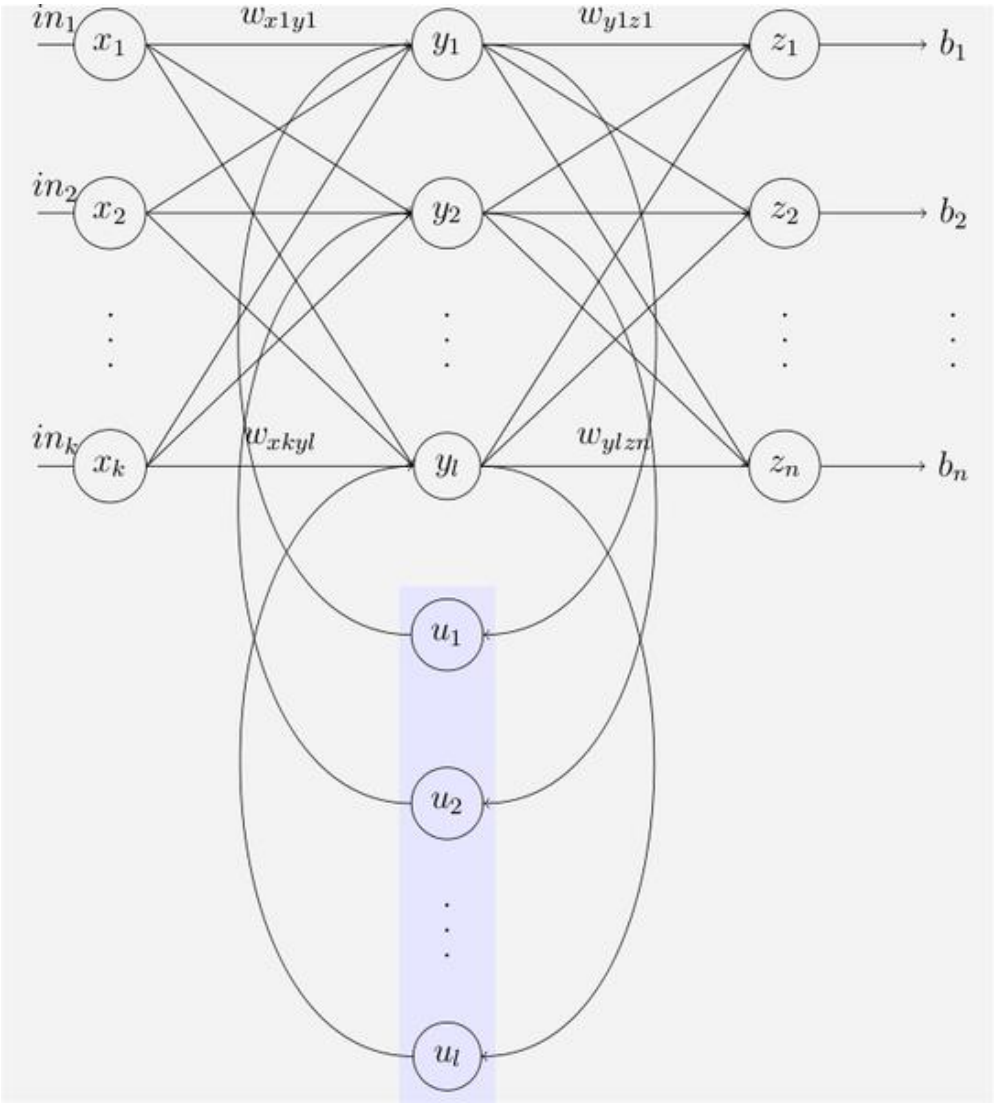

Рисунок 2.4. Сеть Эльмана

Таким образом, сеть может поддерживать своего рода состояние, позволяя ей выполнять такие задачи, как предсказание последовательности, которые выходят за рамки возможностей стандартного многослойного персептрона. Сети Иордании похожи на сети Элмана. Единицы контекста загружаются из выходного слоя, а не из скрытого слоя. Единицы контекста в сети Иордании также называются уровнем состояния. У них есть постоянная связь с самими собой. Сети Элмана и Джордана также известны как «Простые рекуррентные сети» (SRN).

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$

$$y_t = \sigma_y(W_h h_t x_t + b_y) \qquad (2.1)$$

Сеть Эльмана

$$h_t = \sigma_h(W_h x_t + U_h y_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y) \quad (2.2)$$

Иорданская сеть

Переменные и функции

xt: входной вектор

ht: вектор скрытого слоя

yt: выходной вектор

W, и: матрицы параметров и вектор Ub

σh и: Функции активации σy

Сеть Хопфилда — это RNN, в которой все соединения между уровнями имеют одинаковый размер. Он требует стационарных входов и, следовательно, не является обычной RNN, поскольку не обрабатывает последовательности шаблонов. Однако это гарантирует, что он сойдется. Если соединения обучаются с использованием обучения Hebbian, тогда сеть Hopfield может работать как надежная адресно-адресная память, устойчивая к изменению соединения.

Двунаправленная ассоциативная память (BAM). Представленная Бартом Коско сеть двунаправленной ассоциативной памяти (BAM) представляет собой вариант сети Хопфилда, в которой ассоциативные данные хранятся в виде вектора. Двунаправленность возникает из-за прохождения информации через матрицу и ее транспонирования. Обычно биполярное кодирование предпочтительнее двоичного кодирования ассоциативных пар. Недавно стохастические модели BAM с использованием марковского шага были оптимизированы для повышения стабильности сети и соответствия реальным приложениям. Сеть BAM имеет два уровня, каждый из которых может использоваться как вход для вызова ассоциации и создания выходных данных на другом уровне.

Сеть состояния эха (ESN) имеет редко связанный случайный скрытый слой. Вес выходных нейронов - единственная часть сети, которая может изменяться (обучаться). ESN хорошо воспроизводят определенные временные

ряды. Вариант для нейронов с импульсами известен как машина с жидким состоянием.

Независимая рекуррентная нейронная сеть (IndRNN) решает проблемы исчезновения и взрыва градиента в традиционной полносвязной RNN. Каждый нейрон в одном слое получает только свое собственное прошлое состояние в качестве контекстной информации (вместо полной связи со всеми другими нейронами в этом слое), и, таким образом, нейроны не зависят от истории друг друга. Обратное распространение градиента можно регулировать, чтобы избежать исчезновения и увеличения градиента, чтобы сохранить долгосрочную или краткосрочную память. Информация о кросс-нейронах исследуется на следующих уровнях. IndRNN можно надежно обучить с помощью ненасыщенных нелинейных функций, таких как ReLU. С помощью пропуска подключений можно обучать глубокие сети.

Рекурсивная нейронная сеть создается путем применения того же набора весов рекурсивны над дифференцируемым графом-подобной структурой путем обхода структуры в топологическом порядке. Такие сети обычно также обучаются с помощью обратного режима автоматического дифференцирования. Они могут обрабатывать распределенные представления структуры, такие как логические термины. Частным случаем рекурсивных нейронных сетей является РНС, структура которой соответствует линейной цепочке. Рекурсивные нейронные сети применялись для обработки естественного языка. Рекурсивная нейронная тензорная сеть использует основанную на тензоре функцию композиции для всех узлов в дереве.

2.2. Адаптивные оптимизаторы

Математическая оптимизация —это выбор лучшего элемента с учетом некоторого критерия из некоторого набора доступных альтернатив. Задачи оптимизации возникают во всех количественных дисциплинах, от информатики и инженерии до исследования операций и экономики, и разработка методов решения вызывает интерес в математике на протяжении веков. В простейшем

случае, задача оптимизации состоит из максимизации или минимизации на действительную функции путем систематического выбора входных значений внутри допустимого множества, и вычисления значения функции. Обобщение теории и методов оптимизации на другие формулировки составляет большую область прикладной математики. В более общем плане оптимизация включает в себя поиск «наилучших доступных» значений некоторой целевой функции с учетом определенной области (или входных данных), включая множество различных типов целевых функций и различные типы областей [8].

Adadelta. Алгоритм AdaDelta, как и алгоритм RMSProp, использует небольшую группу случайных градиентов g_t.Экспоненциально взвешенная переменная скользящего среднего по квадрату элементаs_tНа временном шаге 0 все его элементы инициализируются в 0. Учитывая гиперпараметры $0 \leq \rho \leq 1$ (Соответствует алгоритму RMSProp y) на временном шаге t > 0, вычисляемом аналогично алгоритму RMSProp:

$$s_t \leftarrow \rho s_{t-1} + (1-\rho) g_t \odot g_t, \tag{2.3}$$

В отличие от алгоритма RMSProp, алгоритм AdaDelta также поддерживает дополнительную переменную состоянияΔx_t, Его элементы также инициализируются значением 0 на временном шаге 0. Мы используемΔx_{t-1} Чтобы вычислить изменение независимой переменной:

$$g'_t \leftarrow \sqrt{\frac{\Delta x_{t-1}+\epsilon}{s_t+\epsilon}} \odot g_t, \tag{2.4}$$

из их ϵ Для поддержания числовой стабильности и добавления констант, таких как10^{-5}, а затем обновите аргумент (параметр):

$$x_t \leftarrow x_{t-1} - g'_t, \tag{2.5}$$

Наконец, используемΔx_t. Запишите изменение собственной переменнойg'_tЭкспоненциально взвешенное скользящее среднее по квадрату элемента:$\Delta x_t \leftarrow \rho \Delta x_{t-1} + (1-\rho) g'_t \odot g'_t$

Как видите, если не учитыватьϵРазница между алгоритмом AdaDelta и алгоритмом RMSProp заключается в использовании$\sqrt{\Delta x_{t-1}}$Вместо скорости обучения η.

Adagrad - это адаптивный метод настройки скорости обучения. Рассмотрим два сценария на рисунке 2.5.

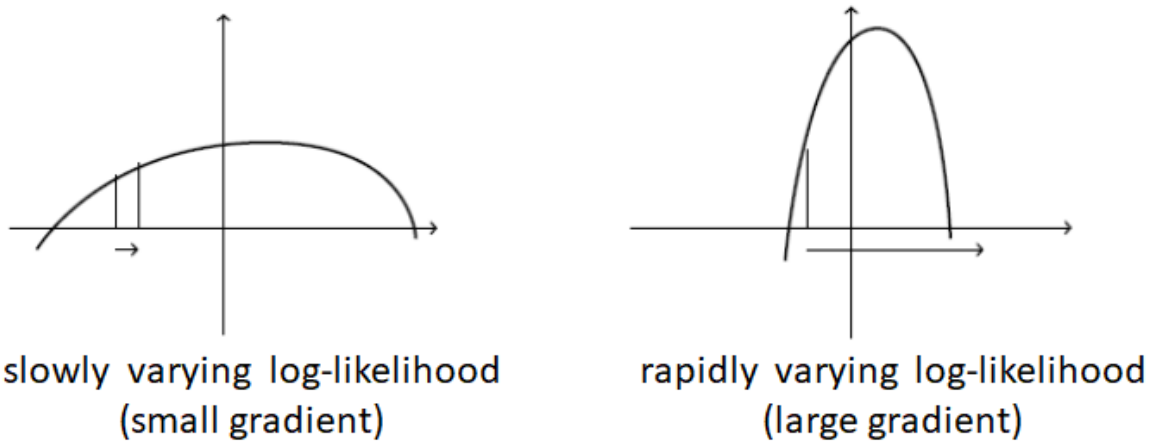

Рисунок 2.5. Сравнение малого и большого градиента

В случае медленно меняющейся цели (слева), градиент обычно (в большинстве точек) имеет небольшую величину. В результате нам потребуется большая скорость обучения, чтобы быстро достичь оптимального уровня. В случае быстро меняющейся цели (справа) градиент, как правило, будет очень большим. Использование большой скорости обучения приводит к очень большим шагам, колеблющимся вокруг, но не достигающим оптимального уровня.

Эти две ситуации возникают потому, что скорость обучения устанавливается независимо от градиента. AdaGrad решает эту проблему путем накопления квадратов норм градиентов, которые мы видели до сих пор, и деления скорости обучения на квадратный корень из этой суммы:

$$g = \frac{1}{m}\nabla_\theta \sum_i L\left(f\left(x^{(i)},\theta\right),y^{(i)}\right)$$

$$s = s + g^T g$$

$$\theta = \theta - \epsilon_k \times g/\sqrt{s + eps} \qquad (2.6)$$

В результате для параметров, которые получают высокие градиенты, будет снижена их эффективная скорость обучения, а для параметров, которые получают маленькие градиенты, будет увеличена их эффективная скорость обучения. Чистый эффект - больший прогресс в более пологих направлениях пространства параметров и более осторожное обновление при наличии больших градиентов.

Adam. Переменная момента используется в Адамеv^tИ алгоритм RMSprop в небольшой пакетной статистической градиентной экспоненциально взвешенной переменной скользящего среднего по квадрату элемента$s^t s_t$, И инициализировать каждый из них на 0 на временном шаге 0. Учитывая гиперпараметры $0 \leq \beta_1 \leq 1$ (Автор алгоритма рекомендует равным 0,9), импульсная переменная временного шага tv^tСтохастический градиент g^tЭкспоненциально взвешенная скользящая средняя:

$$v^t \leftarrow \beta_1 v_{t-1} + (1-\beta_1) g_t \tag{2.7}$$

Как и в алгоритме RMSProp, учитывая гиперпараметры $0 \leq \beta_2 \leq 1$ (Автор алгоритма рекомендует установить значение 0,999), который представляет собой элемент в квадрате небольшой партии случайных градиентов.$g_t \odot g_t$Сделайте экспоненциально взвешенное скользящее среднее, чтобы получитьs_t:

$$s_t \leftarrow \beta_2 s_{t-1} + (1-\beta_2) g_t \odot g_t \tag{2.8}$$

Элементы v_0, s_0 инициализируются значением 0, на временном шаге t мы получаем:

$$v_t = (1-\beta_1) \sum_{i=1}^{t} \beta_1^{t-i} g_i \tag{2.9}$$

Добавьте веса мини-пакетных стохастических градиентов на каждом временном шаге в прошлом, чтобы получить:

$$(1-\beta_1) \sum_{i=1}^{t} \beta_1^{t-i} = 1 - \beta_1^t \tag{2.10}$$

Следует отметить, что, когда t мало, сумма весов пакетного стохастического градиента на каждом временном шаге в прошлом будет небольшой. Например, когда$\beta_1 = 0.9$, время $v_1 = 0.1 g_1$, чтобы исключить такое влияние, для любого шага по времени t можно изменитьv_tРазделить на$1 - \beta_1^t$

(Коррекция отклонения сначала не допускается). С этого момента сумма весов случайных градиентов мини-пакета на каждом временном шаге в прошлом равна 1. Это также называется коррекцией отклонения. В алгоритме Адама есть переменная v_t, s_tВсе исправления отклонений:

$$\acute{\upsilon}_t \leftarrow \frac{v_t}{1-\beta_1^t}$$

$$\dot{\hat{S}}_t \leftarrow \frac{s_t}{1-\beta_2^t} \tag{2.11}$$

Затем алгоритм Адама использует указанные выше переменные с поправкой на отклонение $\acute{\upsilon}_t, \dot{\hat{S}}_t$.Измените скорость обучения каждого элемента в параметрах модели с помощью поэлементных операций:

$$g'_t \leftarrow \frac{\eta \acute{\upsilon}_t}{\sqrt{\dot{\hat{S}}_t} + \epsilon} \tag{2.12}$$

из их η Скорость обучения, єДля поддержания числовой стабильности и добавления констант, таких как 10^{-8}. Как и алгоритм AdaGrad, алгоритм RMSProp и алгоритм AdaDelta, каждый элемент в переменной (параметре) целевой функции имеет свою скорость обучения. Наконец, используйтеg'_t Переменные итерации (параметры):

$$x_t \leftarrow x_{t-1} - g'_t \tag{2.13}$$

AdamW. Распространенные библиотеки глубокого обучения обычно реализуют последнюю регуляризацию L2. Однако в статье показано, что эта эквивалентность справедлива только для SGD, а не для адаптивных оптимизаторов, таких как Adam.

Распад веса также нормализуется sqrt(v). Термин спада веса или регуляризации не попадает в скользящие средние, а в выходных данных он пропорционален только самому весу. На практике авторы показывают, что AdamW дает лучшие потери при обучении, что означает, что модели обобщают намного лучше, чем модели, обученные с Adam, позволяя римейку конкурировать со стохастическим градиентным спуском с импульсом. Если градиент определенного веса велик (или сильно меняется), то соответствующий

v тоже велик, и вес регуляризован меньше, чем веса с малыми и медленно меняющимися градиентами. Это означает, что регуляризация L2 работает не так, как предполагалось, и не так эффективна, как при SGD, поэтому SGD дает модели, которые лучше обобщаются и используются для большинства современных результатов (рис. 2.6).

Авторы предлагают улучшенную версию Adam под названием AdamW, где распад веса выполняется только после управления параметрическим размером шага

Авторы экспериментально показывают, что AdamW дает лучшие потери при обучении и что модели обобщаются гораздо лучше, чем модели, обученные с помощью Adam, позволяя новой версии конкурировать со стохастическим градиентным спуском с импульсом. Это означает, что в будущем исследователям и инженерам, возможно, не придется так часто переключаться между SGD и Adam. (рис 2.6).

Algorithm 2 Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1})$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

Рисунок 2.6. Алгоритм работы AdamW

Adamax - это вариант Адама, и этот метод обеспечивает более простой диапазон для верхнего предела скорости обучения. Изменения формулы, следующие:

$$n_t = \max(v * n_{t-1}, |g_t|)$$

$$\Delta x = -\frac{\acute{m}_t}{n_t + \epsilon} * \eta \qquad (2.14)$$

Адаптивный стохастический градиентный спуск SGD(ASGD). SGD обновляет параметры модели (тета) в отрицательном направлении градиента (г) взяв подмножество или мини-пакет данных размером (м):

Нейронная сеть представленаf (x (i); тета) где x(я) данные обучения иу (я)учебные метки, градиент потерьLвычисляется относительно параметров моделитета (рис 2.7), Скорость обучения (eps_k) определяет размер шага, который алгоритм выполняет вдоль градиента (в отрицательном направлении в случае минимизации и в положительном направлении в случае максимизации).

Скорость обучения является функцией итерации является единственным наиболее важным гиперпараметром. Слишком высокая скорость обучения (например,> 0,1) может привести к обновлениям параметров, которые пропускают оптимальное значение, а слишком низкая скорость обучения (например, <1e-5) приведет к излишне длительному времени обучения. Хорошая стратегия состоит в том, чтобы начать со скорости обучения 1e-3 и использовать график скорости обучения, который уменьшает скорость обучения как функцию итераций (например, планировщик шагов, который делит скорость обучения пополам каждые 4 эпохи):

```
def step_decay(epoch):
   lr_init = 0.001
   drop = 0.5
   epochs_drop = 4.0
   lr_new = lr_init * \
         math.pow(drop, math.floor((1+epoch)/epochs_drop))
   return lr_new
```

Необходимо, чтобы скорость обучения (eps_k) выполняла условия Роббинса-Монро:

Первое условие гарантирует, что алгоритм сможет найти локально оптимальное решение независимо от начальной точки, а второе управляет колебаниями.(рис. 2.7).

Algorithm 4 Adaptive SGD

$x^0 \in \mathbb{R}^d$, $\lambda_0 > 0$, $\theta_0 = +\infty$, ξ^0, $x^1 = x^0 - \lambda_0 \nabla f_{\xi^0}(x^0)$, $\alpha > 0$
for $k = 1, 2, \ldots$ **do**
  Sample ξ^k and optionally ζ^k
  Option I (biased): $L_k = \frac{\|\nabla f_{\xi^k}(x^k) - \nabla f_{\xi^k}(x^{k-1})\|}{\|x^k - x^{k-1}\|}$
  Option II (unbiased): $L_k = \frac{\|\nabla f_{\zeta^k}(x^k) - \nabla f_{\zeta^k}(x^{k-1})\|}{\|x^k - x^{k-1}\|}$
  $\lambda_k = \min\left\{\sqrt{1+\theta_{k-1}}\lambda_{k-1}, \frac{1}{\alpha L_k}\right\}$
  $x^{k+1} = x^k - \lambda_k \nabla f_{\xi^k}(x^k)$
  $\theta_k = \frac{\lambda_k}{\lambda_{k-1}}$
end for

Рисунок 2.7. Алгоритм работы ASGD

РАЗДЕЛ 3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ТЕСТИРОВАНИЕ И АНАЛИЗ РЕЗУЛЬТАТОВ

3.1. Программная реализация и тестирование программы преобразования речи в текст

В ходе выполнения работы была разработана программа для преобразования речи в текст. Для ее реализации были использованы следующие бибилотеки

Подключение библиотек:

```
import speech_recognition as sr
import sys
import webbrowser
import pyttsx3;
import speech_recognition as sr
```

SpeechRecognition: - пакет, позволяет создавать сценарии для доступа к микрофонам и обработки аудиофайлов с нуля.

Библиотека SpeechRecognition действует как оболочка для нескольких популярных речевых API и, таким образом, является чрезвычайно гибкой

Гибкость и простота использования пакета SpeechRecognition делают его отличным выбором для любого проекта Python. Тем не менее, поддержка каждой функции каждого API, который он включает, не гарантируется. Вам нужно будет потратить некоторое время на изучение доступных опций, чтобы выяснить, будет ли SpeechRecognition работать в вашем конкретном случае

import sys- за счет функции(exit) библиотеки sys, будет осуществляться закрытие программы

import webbrowser- за счет функции(open) библиотеки webbrowser, будет осуществляться открытие сайта с моим инстаграммом и яндекс-поиск

import pyttsx3- PyTTSx3 — удобная кроссплатформенная библиотека для реализации **Text To Speech** в приложениях на Python 3. Использует разные системы синтеза речи в зависимости от текущей ОС:

- в Windows — SAPI5,
- в Mac OS X — nsss,
- в Linux и на других платформах — eSpeak.

```
def talk(words):
print(words)
engine = pyttsx3.init();
engine.say(words);
engine.runAndWait();
```

Функция для воспроизведения звука из текста (переменной “talk”)

Функция command() служит для отслеживания микрофона.

Вызывая функцию, мы будет слушать что скажет пользователь, при этом для прослушивания будет использован микрофон.

Получение данные будут сконвертированы в строку и далее будет происходить их проверка.

```
def command ():
    r = sr.Recognizer()
    with sr.Microphone() as source:
        r.adjust_for_ambient_noise(source, duration=1)
        # adjust_for_ambient_noise для удаления посторонних шумов из аудио дорожки
        # Полученные данные записываем в переменную audio
        audio = r.listen(source)
        # пока мы получили лишь mp3 звук
    try: # Обрабатываем все при помощи исключений
        zadanie = r.recognize_google(audio, language="ru-RU").lower()
        # Просто отображаем текст что сказал пользователь
        talk("Вы сказали: " + zadanie)
        print("Вы сказали: " + zadanie)
    # Если не смогли распознать текст, то будет вызвана эта ошибка
    except sr.UnknownValueError:
```

```
            talk("Я вас не поняла")
            zadanie = command()
        # В конце функции возвращаем текст задания
        # или же повторный вызов функции
        return zadanie
# Данная функция служит для проверки текста,
# что сказал пользователь (zadanie - текст от пользователя)
def makeSomething(zadanie):
        # Попросту проверяем текст на соответствие
        # Если в тексте что сказал пользователь есть слова
        # "открыть сайт", то выполняем команду
        if 'что ты умеешь' in zadanie:
            talk("Смотрите")
            print('я умею несколько вещей:\n'
                  'Слушать вас.\n'
                  'Открыть инстаграм, просто скажите "открой instagram".\n'
                  'Найти что-то для вас, просто скажите "поиск".\n'
                  'Спросить мое имя, просто скажите "твое имя".\n'
                  'Рассказать о себе, просто скажите "что ты такое".\n'
                  'Остановить программу, просто скажите "стоп".\n')
        elif 'открой instagram' in zadanie:
            talk("Уже открываю")
            url = 'https://www.instagram.com/artem.my29/'
            webbrowser.open(url)
        # если было сказано "стоп", то останавливаем прогу
        elif 'поиск' in zadanie:
            talk('Что будем искать ?')
            word = command()
            webbrowser.open('https://yandex.ua/search/?text='+word)
        elif 'что ты такое' in zadanie:
```

```
            talk('Я еще очень глупый, но уже что-то умею')
    elif 'стоп' in zadanie:
            talk("Без проблем, до свидания")
            sys.exit()
    elif 'твое имя' in zadanie:
            talk("На ваше усмотрение , мой повелитель")
# Вызов функции для проверки текста будет
# осуществляться постоянно, поэтому здесь прописан бесконечный цикл while
while True:
    makeSomething(command())
```

3.2. Программная реализация и тестирование нейронной сети

from os import listdir - Нужно для открытия папок

import numpy as np - NumPy — это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами.

import librosa - Librosa — это пакет Python для анализа музыки и аудио. Он предоставляет строительные блоки для создания структур, которые помогают получать информацию о музыке.

from sklearn.model_selection import train_test_split - Для примеров будет использоваться модуль train_test_split библиотеки Scikit-learn, который очень полезен для разделения датасетов, независимо от того, будете ли вы применять Scikit-learn для выполнения других задач машинного обучения. Конечно, можно выполнить такие разбиения каким-либо другим способом (возможно, используя только Numpy). Библиотека Scikit-learn включает полезные функции, позволяющее сделать это немного проще.

import torch, import torch.nn as nn, import torch.nn.functional as F. PyTorch определяется как библиотека машинного обучения с открытым исходным кодом для Python. Он используется для приложений, таких как

обработка естественного языка. Первоначально он был разработан исследовательской группой по искусственному интеллекту Facebook и программным обеспечением Uber Pyro для вероятностного программирования, которое основано на нем.

PyTorch перепроектирует и внедряет Torch в Python, совместно используя те же основные библиотеки C для внутреннего кода. Разработчики PyTorch настроили этот внутренний код для эффективной работы с Python. Они также сохранили аппаратное ускорение на основе графического процессора, а также функции расширяемости, которые сделали Torch на базе Lua.

```
def loadSound(path):
    soundList = listdir(path)
    loadedSound = []
    for sound in soundList:
        Y, sr = librosa.load(path + sound)
        loadedSound.append(librosa.feature.mfcc(y=Y, sr=sr))
    return np.array(loadedSound)
```

Загружаются данные в переменную “loadedSound” и используется библиотека librosa и функцию mfcc

MFCC — это представление сигнала, грубо говоря, в виде особого спектра, из которого с помощью различных фильтраций и преобразований удалены незначительные для человеческого слуха компоненты. Спектр носит кратковременный характер, то есть изначально сигнал делится на пересекающиеся отрезки по 20-40 мс. Предполагается, что на таких отрезках частоты сигнала не меняются слишком сильно. И уже на этих отрезках и считаются волшебные коэффициенты.

```
one = loadSound('./voice/one/')
two = loadSound('./voice/two/')
three = loadSound('./voice/three/')
```

Загружаем наш звук в программу

```
X = np.concatenate((one, two, three), axis=0)
```

Соединяем их в один датасет

```
one_label = np.concatenate((np.ones(10), np.zeros(10), np.zeros(10)))
two_label = np.concatenate((np.zeros(10), np.ones(10), np.zeros(10)))
three_label = np.concatenate((np.zeros(10), np.zeros(10), np.ones(10)))
y = np.concatenate((np.repeat(0, 10), np.repeat(1, 10), np.repeat(2, 10)), axis=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=42, shuffle=True)
X_train = X_train.swapaxes(1, 0)
X_test = X_test.swapaxes(1, 0)
```

Создаем Label(метки) и соединяем их

```
X_train_torch = torch.from_numpy(X_train).float()
X_test_torch = torch.from_numpy(X_test).float()
y_train_torch = torch.from_numpy(y_train).long()
y_test_torch = torch.from_numpy(y_test).long()
```

Делаем разделение на обучающий и тестовый сет

```
class RNN(nn.Module):
    def __init__(self):
        super(RNN, self).__init__()
        self.lstm1 = nn.LSTM(input_size=87, hidden_size=256)
        self.lstm2 = nn.LSTM(input_size=256, hidden_size=128)
        self.lstm3 = nn.LSTM(input_size=128, hidden_size=64)
        self.lstm4 = nn.LSTM(input_size=64, hidden_size=32)
        self.fc1 = nn.Linear(in_features=32, out_features=128)
        self.fc2 = nn.Linear(in_features=128, out_features=64)
        self.fc3 = nn.Linear(in_features=64, out_features=32)
        self.fc4 = nn.Linear(in_features=32, out_features=3)
    def forward(self, x):
        x = torch.tanh(self.lstm1(x)[0])
        x = torch.tanh(self.lstm2(x)[0])
```

```
        x = torch.tanh(self.lstm3(x)[0])
        x = torch.tanh(self.lstm4(x)[0][0])
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return x
```

Создание экземпляра класса RNN и передачи необходимых аргументов

Функция 1- В первой строке инициализации класса *def __init__(self)* мы имеем требуемую *super()* функцию языка Python, которая создает объект базового класса передача необходимых аргументов и определения скелета архитектуры сети.

Функция 2- Это делается с помощью определяемого метода *forward(),* который переписывает фиктивный метод в базовом классе и требует определения для каждой сети Принципы, по которым данные будут перемещаться по ней(в данном случае, через переменную X).

```
model = RNN()
Название модели
loss_fn = torch.nn.CrossEntropyLoss()
Потери в ходе обучения
learning_rate = 0.0001
Шаг обучения
params = model.parameters()
Параметры модели
txtfile = input("Введите имя файла"+'\n')
Запись показателей в файл
optimizer = torch.optim.Adam(params, lr=learning_rate)
Оптимизатор адам
for t in range(0, 1001):
   y_pred = model(X_train_torch)
```

```
    loss = loss_fn(y_pred, y_train_torch)
    print(t, loss.item())
    f = open(txtfile+'.txt', "a")
    f.write(str(t)+'\t')
    f.close()
    f = open(txtfile+'.txt', "a")
    f.write(str(loss.item())+'\n')
    f.close()
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    with torch.no_grad():
        def accuracy_calc(X, y, type):
            correct = 0
            total_correct = 0
            outputs = model(X).detach().numpy()
            label = y.detach().numpy()
            for number in range(outputs.shape[0]):
                correct = np.argmax(outputs[number]) == label[number]
                total_correct += correct
      print(type + ' accuracy: ' + str(total_correct / outputs.shape[0] * 100) + '%')
      q = open(txtfile+'.txt', "a")
      q.write(type + ' accuracy: ' + str(total_correct / outputs.shape[0] * 100) + '\n')
      q.close()


        accuracy_calc(X_train_torch, y_train_torch, "Training")
        accuracy_calc(X_test_torch, y_test_torch, "Testing")
```

Цикл загрузки всех данных и тренировка модели

3.3. Результаты тестовых расчетов

Результат работы программы выражается в зависимости процентов успешной работы нейронов к количеству пройденных циклов. То, насколько каждый из оптимизаторов соответствует поставленной задаче, представлено в виде графиков соотношений тестовых циклов к тренировочным.

Рассмотрим графики для каждого из оптимизаторов в том же порядке, что и в предыдущем разделе.

Adadelta. Как видно на рисунке 3.1. для тестовых циклов оптимизатора, в промежутке циклов 0-50 наблюдается стремительный рост процентов успеха от 0 до 37, после чего оно стабильно равно 37 до 500 цикла, затем на протяжении оставшихся 500 циклов наблюдается рост значений, который перемежается спадами. Для тренировочных значений до 500 цикла показатель успеха равен 0, после чего виден резкий рост до 66%, далее спад до 33% и стабильная работа до завершения циклов, что может означать вероятный выброс. Анализ результатов работы процент от тренированных нейронов стабильный, но не качественный и нуждается в большем количестве циклов.

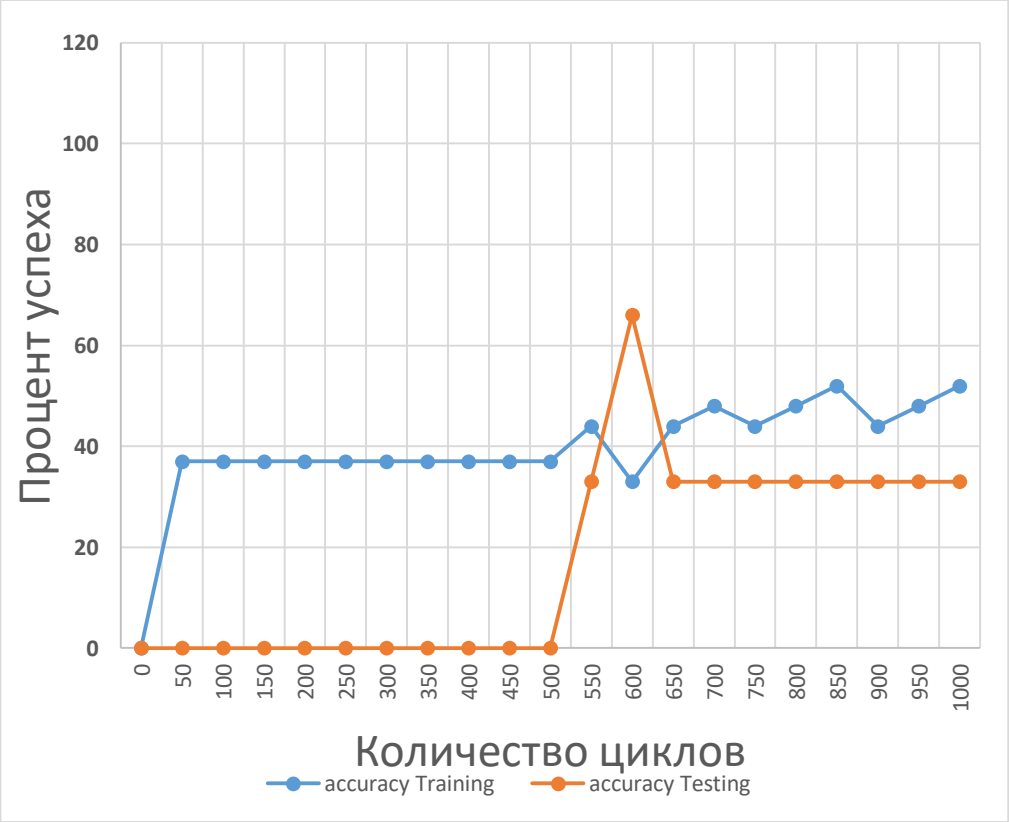

Рисунок 3.1. Adadelta

Adagrad. Для оптимизатора Adagrad ниже приведён рисунке 3.2. Из него видно, что тестовые и тренировочные значения после 50 циклов стабильны на протяжении всей работы и равны, соответственно, 33% и 41%. Полученные значения говорят о стабильной работе данного оптимизатора, однако низком проценте успеха, это означает, что Adagrad плохо подходит для распознания речи.

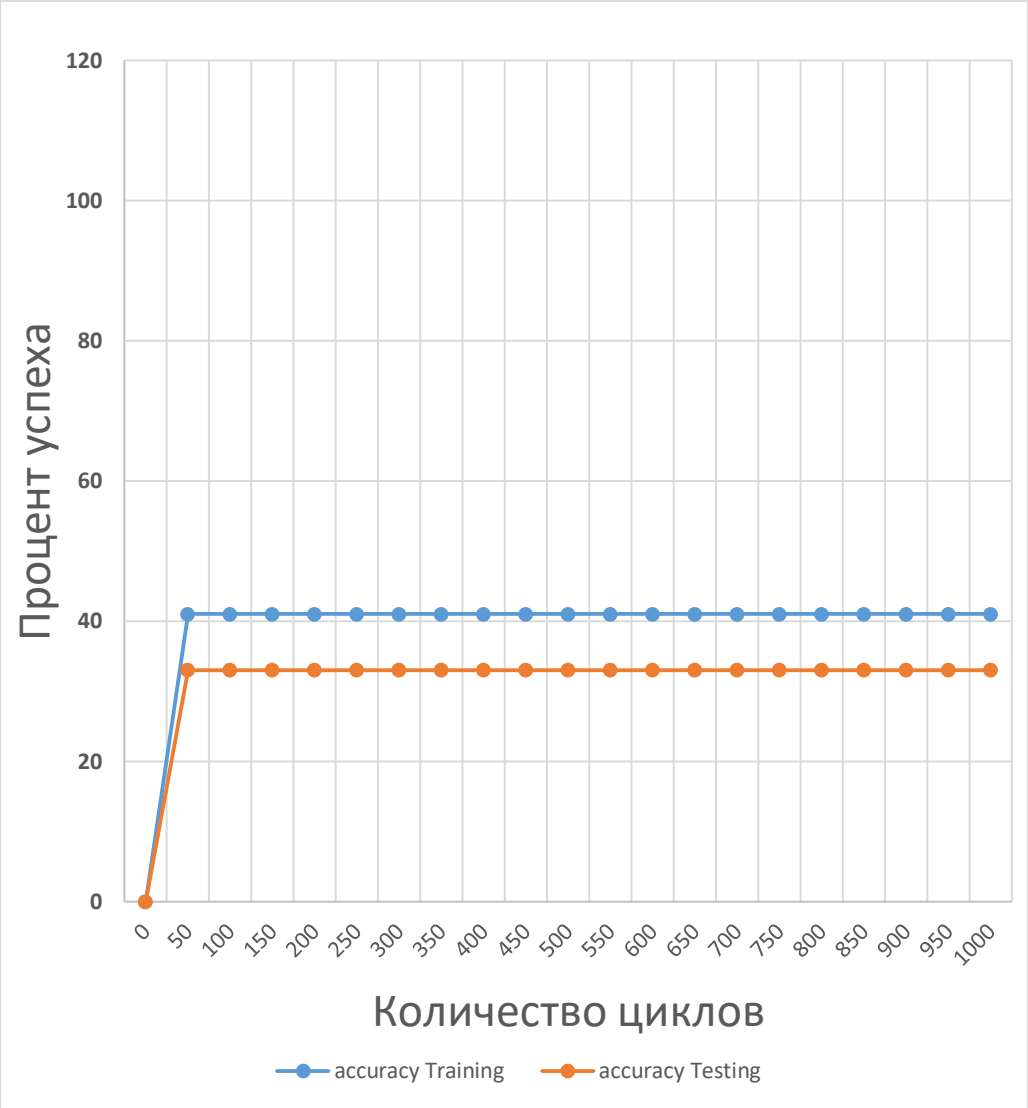

Рисунок 3.2. Adagrad

AdamDef. На рисунке 3.3. представлены результаты работы оптимизатора AdamDef. Аналогично с предыдущими оптимизаторами, наблюдается стремительный рост показателей до 50 цикла, после чего тестовые значения остаются стабильными до конца работы. Для тренировочных же значений виден рост на 200-250 цикл до 100%, за чем следует спад до 33% и снова рост до 100%.

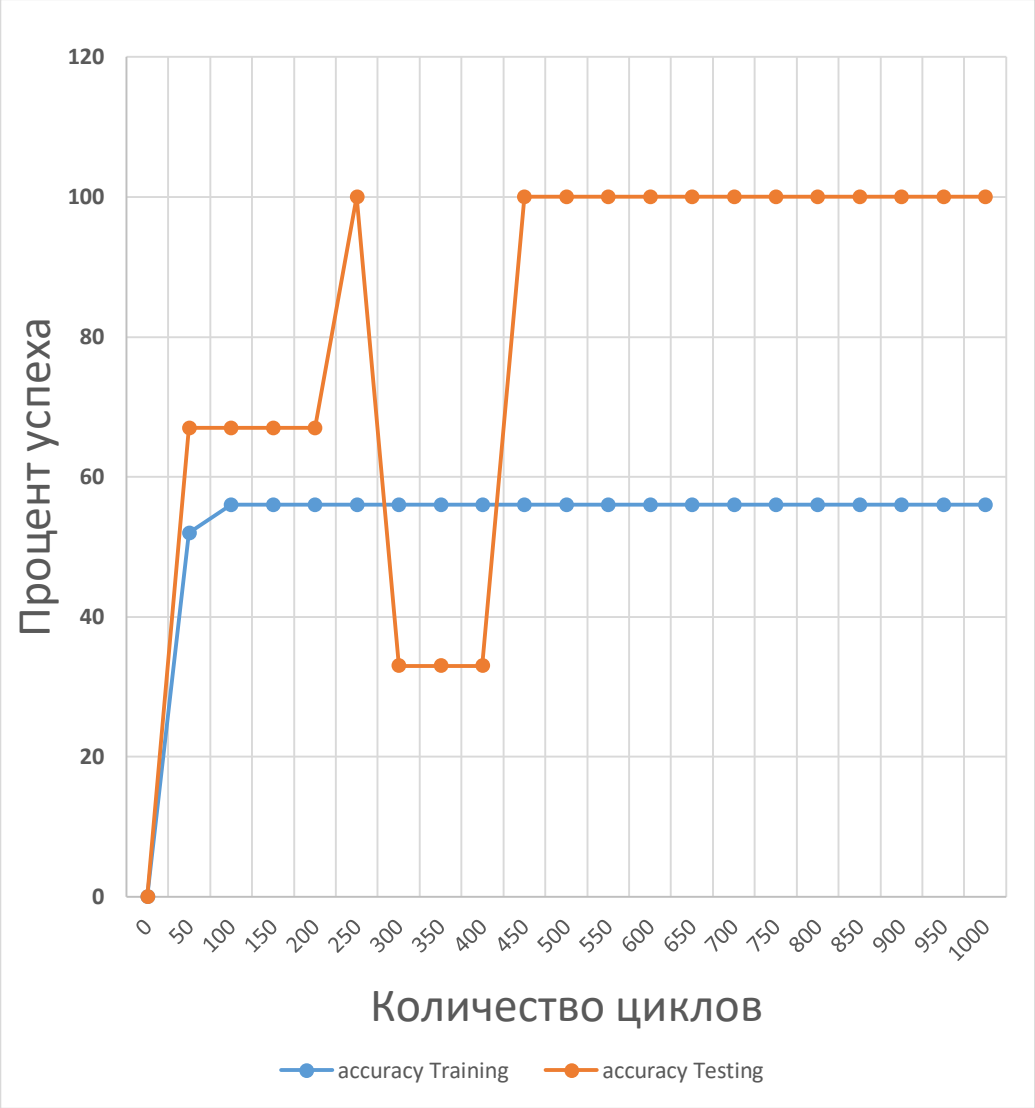

Рисунок 3.3. AdamDef.

AdamW. По результатам работы оптимизатора AdamW (рис. 3.4) виден скачок тестовых значений от 0% до 66% на промежутке 5-150 циклов, спад до 0% и после 200-го цикла значения стабильно равны 33%.На основании этих результатов можно сделать вывод о переобучении в начале работы оптимизатора.

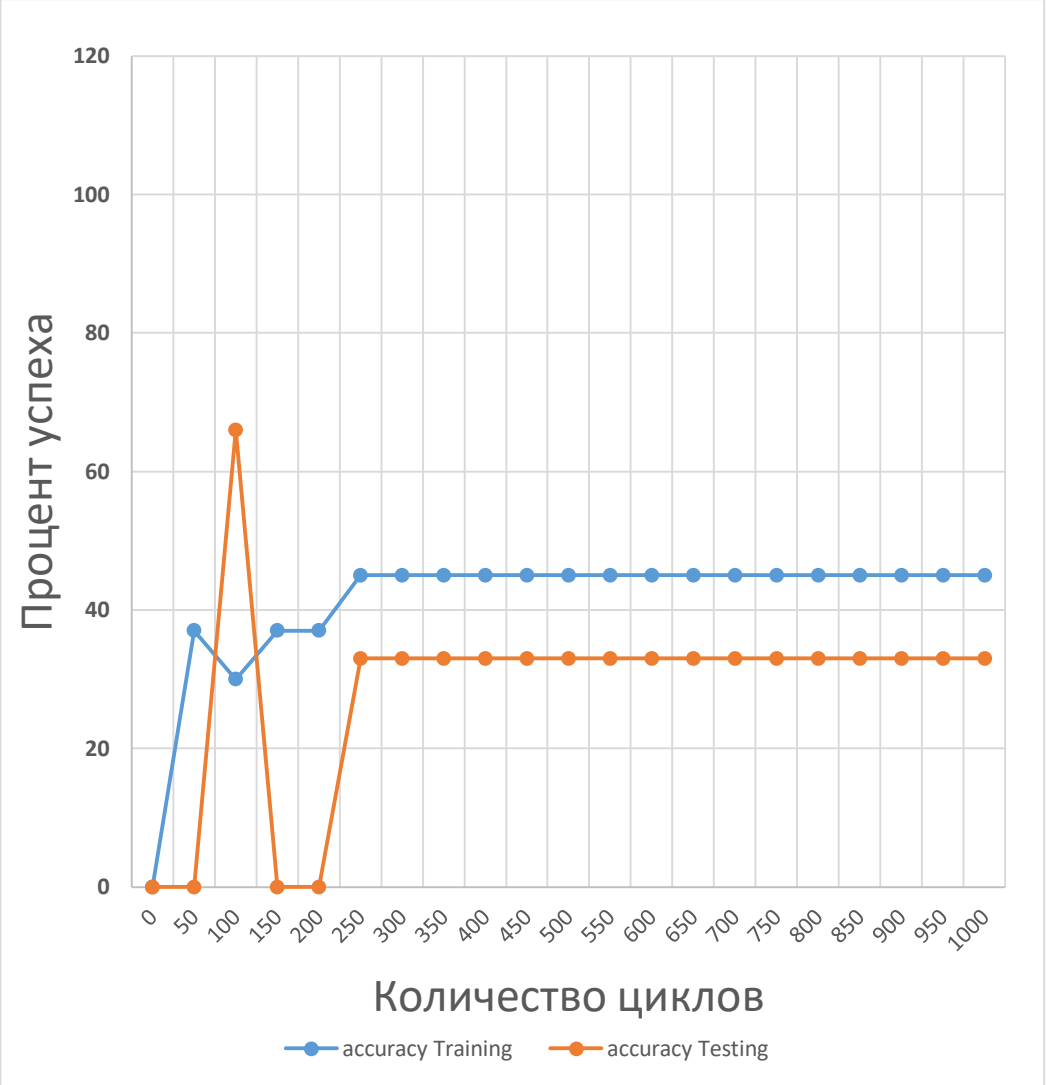

Рисунок 3.4. AdamW

Adamax. На рисунке 3.5. представлена работа оптимизатора Adamax. Для тестовых значений видно резкое увеличение показателей до 150-го цикла, затем рост становится боле плавным. Тренировочные значения также растут до 150-го цикла, увеличиваясь до 100%, однако на 550-м цикле происходит резкий спад, после чего значения возвращаются в норму, что видно и на 950-м цикле. Данный результат свидетельствует о том, что Adamax приближен к качественной оптимизации, однако присутствует переобучение, что повторяется с определённой периодичностью, при увеличении общего количества циклов.

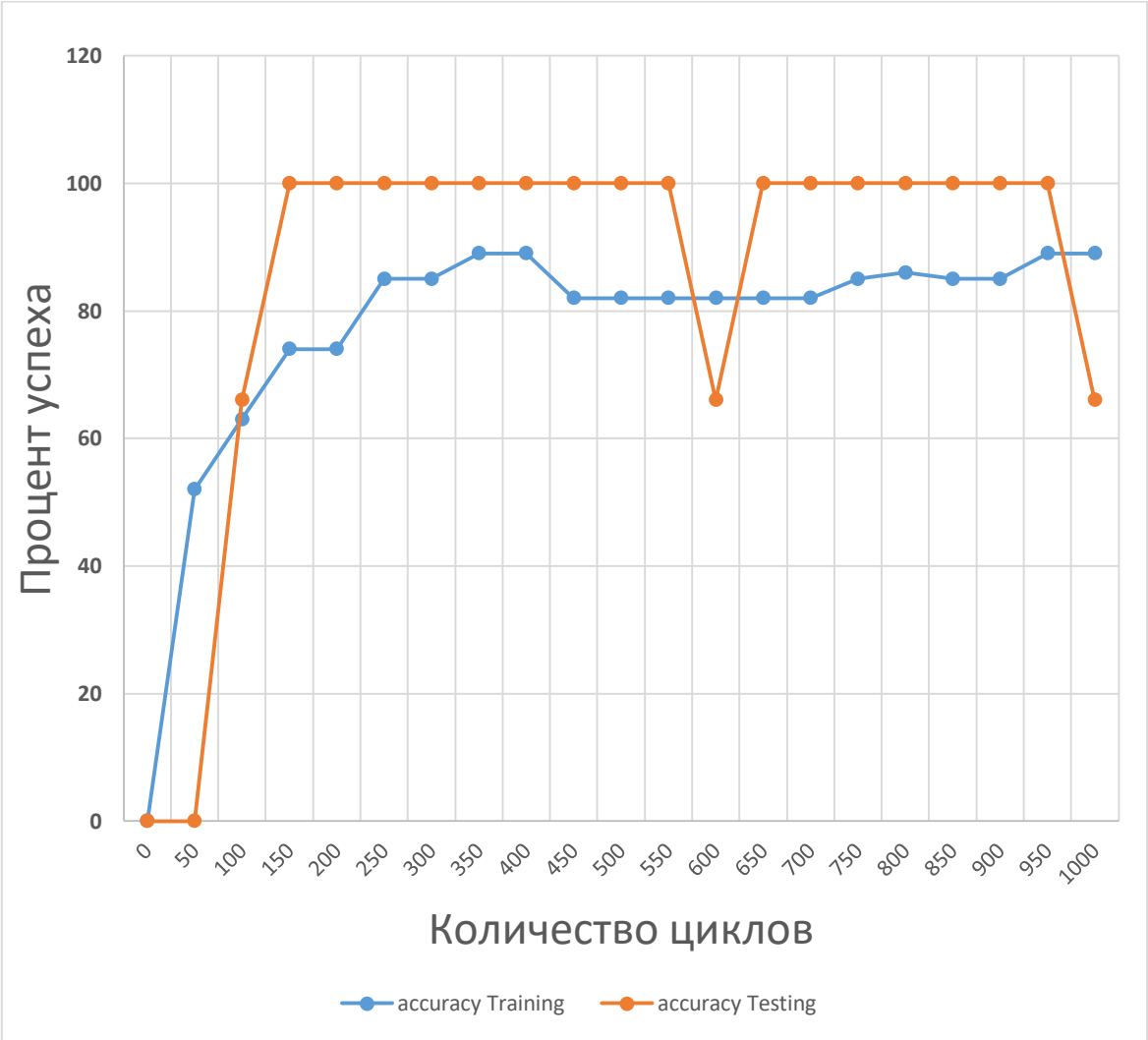

Рисунок 3.5. Adamax

ASGD. Работа оптимизатора на основе адаптивного стохастического градиентного спуска представлена на рисунке 3.6. На промежутке от 0 до 200 циклов тестовые и тренировочные значения совпадают, показатели стремительно растут до 50-го цикла, затем стабильно равны 33%, затем тренировочные значения стабильно равны 37% до конца работы, а тестовые падают до 0%. Это показывает, что данный оптимизатор совершенно не подходит для обработки речевых сообщений.

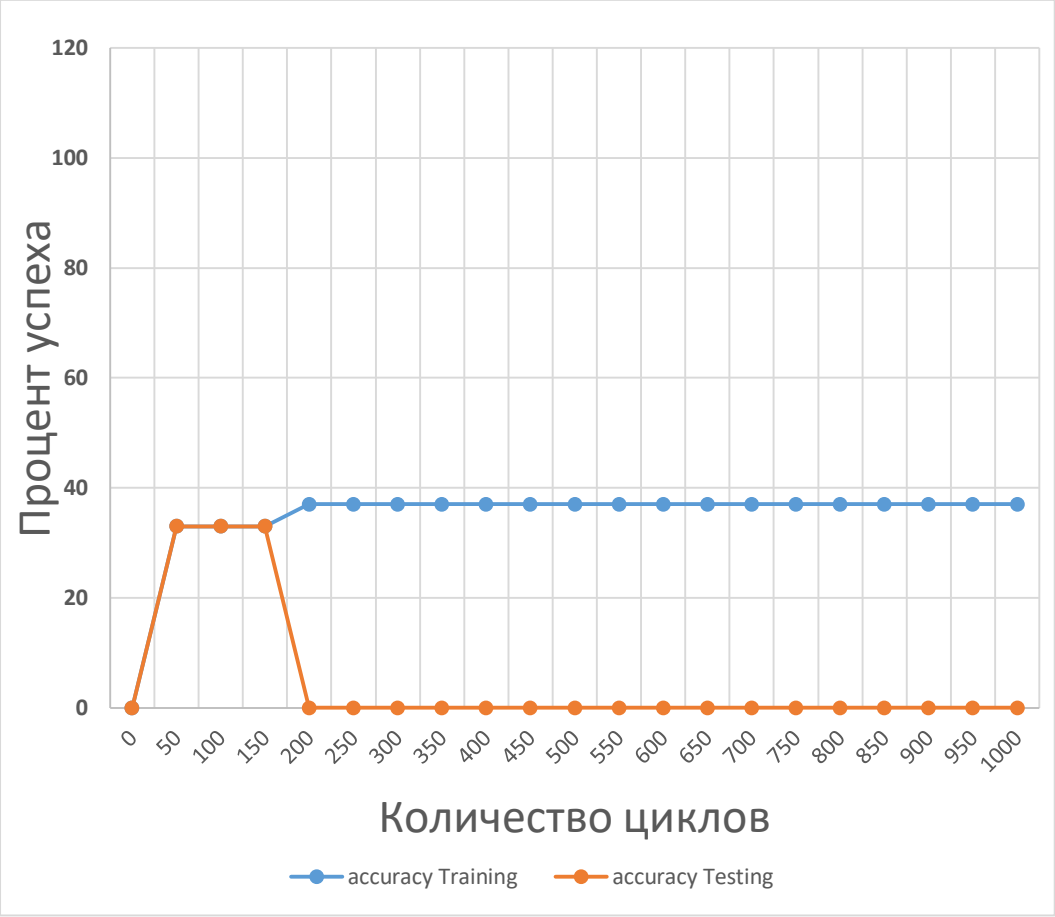

Рисунок 3.6. ASGD

AdamME – это усовершенствованный изменением шага обучаемости оптимизатор Adam (рис. 3.6.). За 250 итераций, процент успеха тренировочных возрастает до 100%. Тестовые значения не увеличиваются до 150-го цикла, однако после наблюдается стремительный рост до 100%. Этот оптимизатор является наиболее подходящим для данной задачи, за счёт быстрого и качественного обучения.

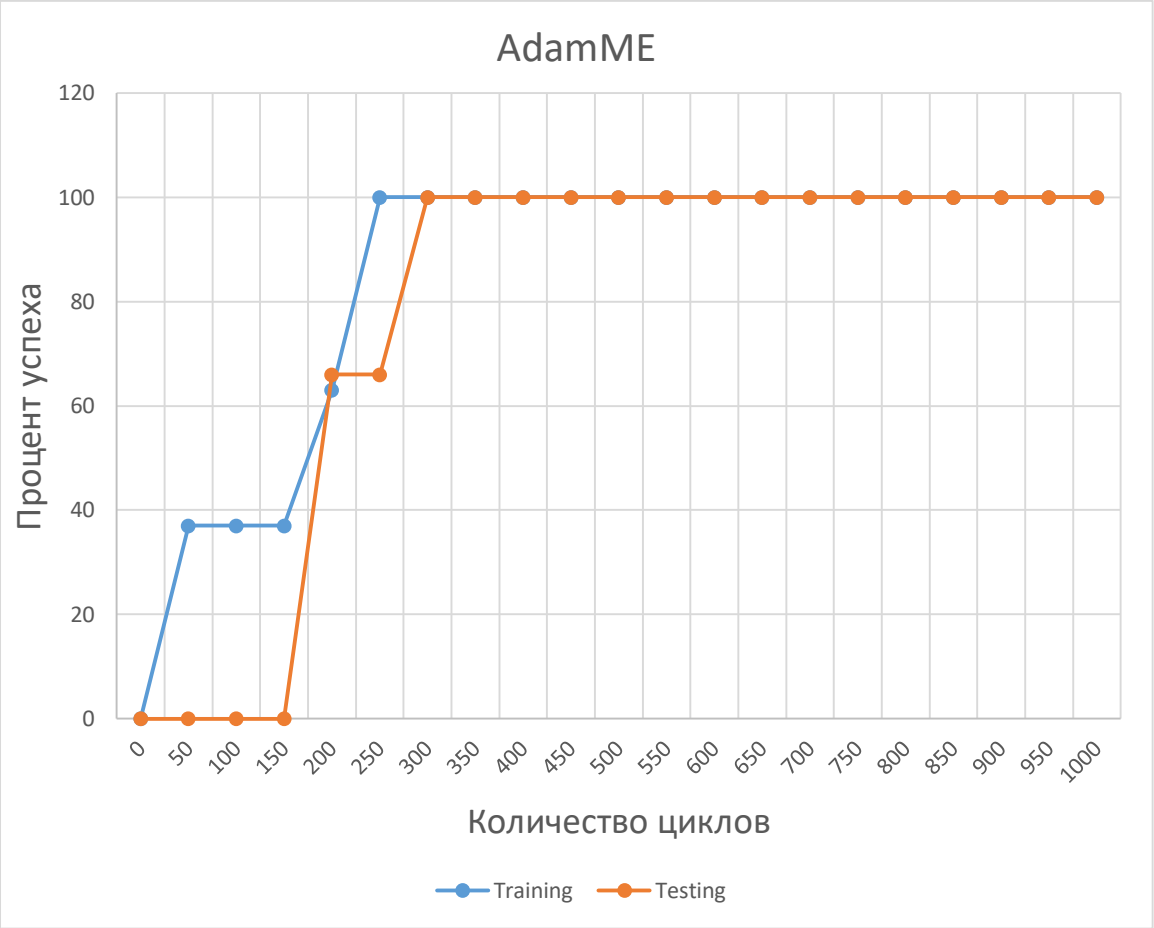

Рисунок 3.7. AdamME

AdamME vs ASGD

На рисунке 3.8. приведена сравнительная характеристика оптимизаторов AdamME и ASGD, которые показали, соответственно, лучший и худший результаты.

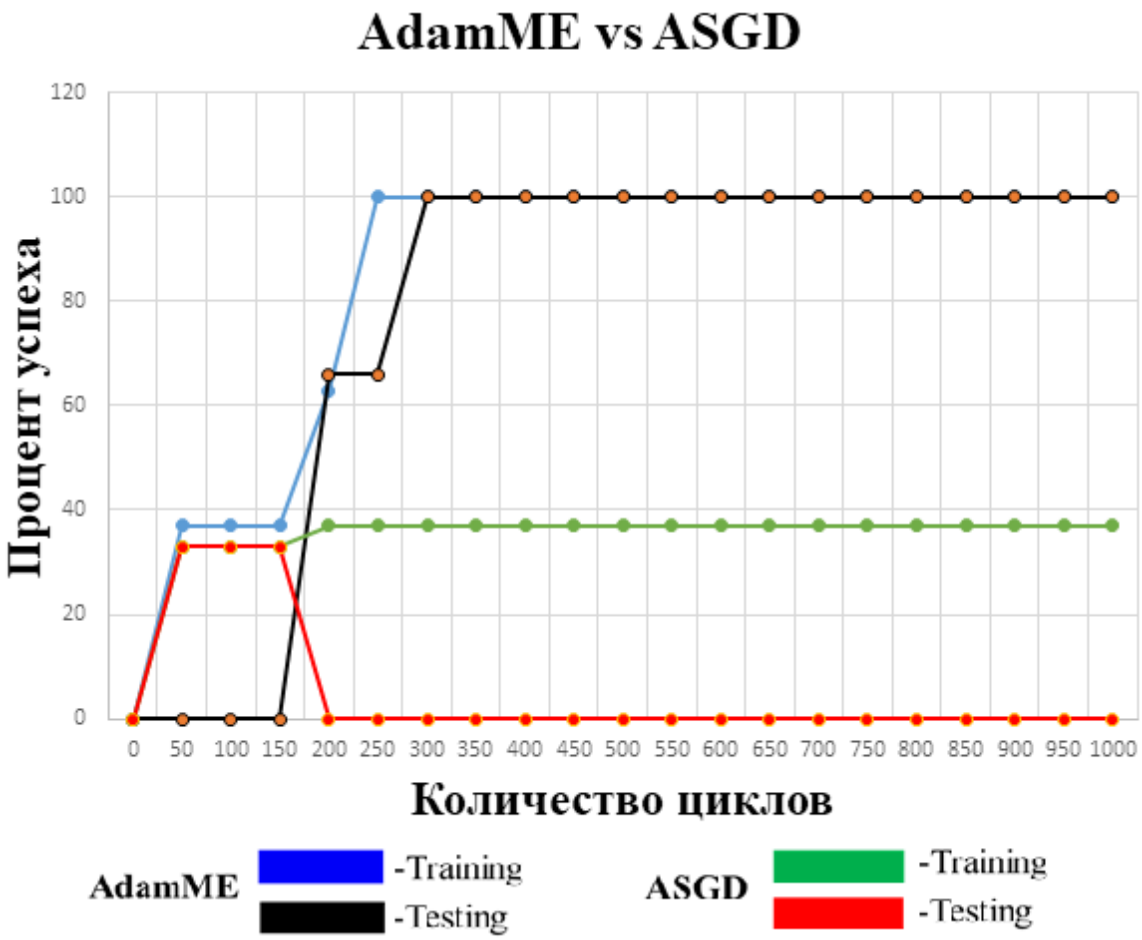

Рисунок 3.8. AdamME vs ASGD

РАЗДЕЛ 4. ОХРАНА ТРУДА

Работа с компьютером. Основные правила организации пространства вокруг рабочего места:

При длительном и интенсивном использовании, на поверхности модулей ПК (системный блок, монитор, мышка и т.д.) возникают небольшие разряды тока. Эти частицы активизируются во время прикосновений к ним и приводят к выходу техники из строя. Нужно регулярно использовать нейтрализаторы, увлажнители воздуха, антистатики; вокруг стола не должно быть свисающих проводов, пользователь не должен контактировать с ними; важна целостность корпуса розетки и штепсельной вилки; отсутствие заземления пред экранного фильтра проверяется с помощью измерительных приборов.

Желательно во время строительных работ в офисе использовать минимальное количество легко воспламеняемых материалов (дерева, пенопласта), а также горючего пластика в изоляции.

Рекомендуется отдавать предпочтение кирпичу, стеклу, металлу и т.д.; помещение должно хорошо вентилироваться и охлаждаться в жаркую пору года. Важен своевременный отвод избыточного тепла от техники.

В случае обнаружения трещины на корпусе или повреждений другого рода, нужно обратиться за помощью в сервисный центр. Это же относится к ПК с неисправным индикатором включения/выключения. предметы на столе не должны мешать обзору, пользованию мышкой и клавиатурой.

Поверхность экрана должна быть абсолютно чистой; на системном блоке не должно находиться никаких предметов, так как в результате вибраций может нарушиться работа устройства. Нужно убедиться в том, что никакие посторонние предметы не мешают работе системе охлаждения.

Запрещается начинать работу в помещениях с повышенной влажностью, а также в случае, если рядом присутствуют открытые источники влажности (лужи, мокрый пол).

Включить технику можно лишь после полного высыхания окружающих предметов. недопустимо часто включать и выключать компьютер в течение рабочего дня без особой нужды. Система просто не справляется с необходимостью быстро сворачивать все процессы [9].

При выполнении работы. Поскольку персональный компьютер обладает всеми свойствами электрического прибора, то на него распространяются основные правила безопасности при взаимодействии с проводниками тока: нельзя размещать какие-либо вещи на поводах, а также самостоятельно менять их расположение без особой нужды; рекомендуется избегать расположения жидкостей рядом с модулями компьютера.

Поэтому кулер с водой или кофейный автомат необходимо размещать в стороне от рабочих мест в офисе. Пользователи должны осознавать опасность потенциального замыкания в случае пролития воды на клавиатуру или системный блок. Нельзя работать на ПК с мокрыми руками; нельзя очищать поверхность компьютера от загрязнений, когда он находится во включенном состоянии; недопустимо снимать корпус любой из составных частей ПК во время его работы.

Кроме того, разбор и ремонт техники имеют совершают только специализированные работники; во время работы на компьютере нельзя одновременно прикасаться к другим металлическим конструкциям, которые стоят на той же поверхности. Это касается отопительных батарей или трубопроводов; в помещении с компьютерами непозволительно курить или употреблять пищу непосредственно на рабочем месте; при ощущении даже незначительного запаха гари, нужно как можно быстрее выключить ПК из сети и обратиться к ответственному за обслуживание компьютерной техники [9].

ЗАКЛЮЧЕНИЕ

Выбрана и реализована модель нейронной сети, позволяющая распознать человека по голосу

Разработана и реализована автоматизированная система распознавания голоса

Тестовые эксперименты с использованием различных оптимизаторов показали, что, наилучшие показатели были у усовершенствованной модели Адам 100%/100%, наихудшее ASGD 37%/0%

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. М.Л. Зельцер и Дж. Дроппо. Многозадачное обучение в глубоких нейронных сетях для улучшения распознавания фонем. In Proceedings of ICASSP. 2013 г. Ванкувер, Канада

2. К. Мехротра, К. К. Мохан, С. Ранка. Элементы искусственных нейронных сетей. MIT Press,

3. Дж. Хинтон, Л. Дэн, Ю. Донг, Дж. Даль и Ко. Глубокие нейронные сети для акустического моделирования в распознавании речи. 2012 г.

4. Ю. Сабато, М. Синискальчи, Л. Дэн, К. Ли. Повышение точности атрибутов и оценки телефона с помощью глубоких нейронных сетей для распознавания речи. В материалах ICASSP. 2012 г

5. П. Ладефогед и К. Джонсон. Курс фонетики. 6-е изд. Бостон: Уодсворт. 2011 г.

6. Detection of Anomalous Behavior in Modern Smartphones Using Software Sensor-Based Datahttps [Электронный ресурс] – Режим доступа: www.mdpi.com/1424-8220/20/10/2768

7. Продвинутое использование библиотеки компьютером [Электронный ресурс] – Режим доступа: https://habr.com/ru/post/553716/

8. Франсуа Шолле. Deep Learning with Python. 2018г ISBN 978-5-4461-0770-4

9. Техника безопасности при работе с компьютером [Электронный ресурс] – Режим доступа: http://proremontpk.ru/ustanovka/tehnika-bezopasnosti-pri-rabote-s-personalnym-kompjuterom.htm