ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(НИУ «БелГУ»)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА МАТЕМАТИЧЕСКОГО И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ВИРТУАЛЬНОГО СОБЕСЕДНИКА НА ОСНОВЕ РЕКУРРЕНТНОЙ НЕЙРОННОЙ СЕТИ

Выпускная квалификационная работа
обучающегося по направлению подготовки 02.04.01 Математика и компьютерные науки
очной формы обучения, группы 07001631
Соловьева Алексея Николаевича

Научный руководитель
Доцент, к.т.н. Муромцев В.В.

Резензент
Профессор, к.т.н. Ломакин В. В.

БЕЛГОРОД 2018

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

С момента появления самой концепции искусственного интеллекта, исследователи стремились создавать системы, которые могли бы взаимодействовать с людьми в реальном времени. Основополагающей работой в этой области считается статья английского ученого Алана Тьюринга «Вычислительные машины и разум» [1], написанная и опубликованная в 1950 году, в которой впервые появляется концепция, впоследствии ставшей известной под названием тест Тьюринга. В своей работе Тьюринг задает вопрос «Думают ли машины?» Поскольку слова «думать» и «машина» не могут быть определены четким образом, Тьюринг заменяет вопрос на другой «который тесно связан с исходным и формулируется относительно недвусмысленно». Тьюринг предлагает заменить вопрос «Думают ли машины?» вопросом «Могут ли машины делать то, что можем делать мы (как мыслящие создания)?». Преимуществом нового вопроса, утверждает Тьюринг, является то, что он проводит «чёткую границу между физическими и интеллектуальными возможностями человека». Для демонстрации данного подхода, Тьюринг предлагает тест — «игра в имитацию» (англ. Imitation game). Суть теста заключается в следующем: экзаменатор находится в отдельной комнате, из которой он может общаться и с машиной, и с человеком. При этом ответы должны быть представлены в текстовой форме и передаваться через телетайп или с помощью посредника. И машина, и человек пытаются убедить экзаменатора, что являются людьми. Если экзаменатор не может уверенно сказать, кто есть кто, считается, что машина выиграла игру. С 1990 года существует премия Лёбнера (англ. Loebner prize), которая вручается победителю ежегодного конкурса «AI

Loebner», в котором соревнуются программы на прохождение теста Тьюринга.

Другим знаменитым экспериментом в области философии сознания и философии искусственного интеллекта является мысленный эксперимент под названием Китайская комната (англ. Chinese room) [4], впервые опубликованный Джоном Сёрлом в 1980 году [5]. Цель эксперимента состоит в опровержении утверждения о том, что цифровая машина, наделенная «искусственным интеллектом» путём её программирования определенным образом, способна обладать сознанием в том же смысле, в котором им обладает человек. Иными словами, целью является опровержение гипотезы так называемого «сильного» искусственного интеллекта и критика теста Тьюринга. Этот философский аргумент до сих пор является одним из самых обсуждаемых в области когнитивистики. Некоторые исследователи даже определяют когнитивистику как «исследовательский проект по опровержению аргумента Сёрла». Наука «о создании искусственного разума» не могла не привлечь внимание философов. С появлением первых интеллектуальных систем были затронуты фундаментальные вопросы о человеке и знании, а отчасти о мироустройстве, такие как «Что такое ИИ, возможно ли его создание, и, если возможно, то как это сделать?» и «Каковы последствия создания ИИ?». Создание ИИ считается одной из важнейших задач человечества сегодня.

Глава 1. АНАЛИЗ ПРОБЛЕМЫ РАЗРАБОТКИ ВИРТУАЛЬНОГО СОБЕСЕДНИКА И ПОСТАНОВКА ЗАДАЧИ

1.1 Разработка виртуального собеседника с точки зрения задач разработки искусственного интеллекта

Идея о том, чтобы позволить пользователям «выражать свои интересы, пожелания или запросы напрямую и в естественной манере, говоря, печатая и указывая на что-то напрямую» была впервые опубликованная Задрозны и соавторами в 2000 году и стала основой для развития современных диалоговых систем. Виртуальный собеседник (англ. Chatbot) это программа, которая взаимодействует с пользователями с использованием естественного языка. Для виртуального собеседника использовались различные термины, такие как разговорная система, виртуальный агент, диалоговая система или chatterbot. Цель виртуального собеседника — имитировать человеческий разговор. Его архитектура объединяет языковые модели и вычислительные алгоритмы для эмуляции неформального общения между человеком и компьютером с использованием естественного языка. Первоначально разработчики создавали и использовали виртуальных собеседников для удовольствия и использовали простые методы сопоставления ключевых слов, чтобы найти совпадение с информацией, вводимой пользователем. Первым таким собеседником была модель ELIZA, разработанная Вайзенбаумом в 1966-1967 годах. В семидесятые и восьмидесятые годы, до появления графических пользовательских интерфейсов, наблюдался быстрый рост исследований в области естественного языка. Можно выделить работы Клиффа и Атвелла 1987 года и Виленски и соавторов 1988 года. С этого

времени был разработан целый ряд новых архитектур моделей для виртуальных собеседников, таких как MegaHAL (Хатченс, 1996 год), CONVERSE (Батачария и соавторы, 1999 год), ELIZABETH (Абу Шавар и Атвелл, 2002 год), HEXBOT (2004 год) и ALICE (2007 год). Благодаря совершенствованию методов интеллектуального анализа данных и машинного обучения, улучшению возможностей принятия решений, доступности большого количества текстовых документов, надежных лингвистических стандартов аннотации и обработки данных, таких как XML, виртуальные собеседники стали более эффективны, более практичны и получили развитие во многих коммерческих приложениях.

Первоначальная цель создания диалоговых систем заключалась в том, чтобы имитировать человеческий разговор и, тем самым, развлекать пользователей. Первой попыткой создания виртуального собеседника была модель ELIZA, созданная в 60-х годах Джозефом Вайзенбаумом для симуляции диалога между пациентом и психотерапевтом. Идея была простой и заключалась в сопоставлении ключевых слов. Первой итерацией входные данные проверяются на наличие ключевого слова. Если такое слово найдено, предложение отображается в соответствии с правилом, связанным с ключевым словом; если нет, отображается не связанный свободный ответ или, при определенных условиях, отображается более раннее предложение. Например, если входные данные содержат ключевое слово «мать», ELIZA может ответить «Расскажите подробнее о своей семье». Это правило вдохновлено теорией о том, что мать и семья занимают центральное место в психологических проблемах, поэтому терапевт должен поощрять пациента к открытию информации о своей семье. Программа ELIZA не «понимает» эту психологическую стратегию, но всего лишь сопоставляет ключевому слову некий ответ. Для того чтобы разговор продолжился, ELIZA должна давать

ответы, которые побуждают пациента размышлять явно и интроспективно, и в случае когда совпадений по ключевым фразам нет, используются общие фразы вида, например «Очень интересно». Пожалуйста, продолжайте », или «Можете ли вы привести пример?». Когда проект ELIZA был запущен, по крайней мере некоторые пользователи полагали, что они разговаривают с настоящим терапевтом, и часами говорили о своих проблемах. Несмотря на то, что ELIZA не могла понять на самом деле что говорит пользователь, а пользователь через некоторое время мог понять тот факт, что многие ответы ELIZA извлекаются из его собственной информации, эксперимент послужил началом для многих современных диалоговых систем, главной задачей которых является попытка убедить пользователей в том, что они разговаривают с другим человеком, а не с машиной, что и является главным критерием «игры в имитацию».

После проекта ELIZA было разработано много человеко-компьютерных диалоговых систем, чтобы имитировать разных вымышленных или реальных личностей, используя разные алгоритмы сопоставления с образцом, такие как сопоставление ключевых слов в ELIZA или более лингвистически сложные, использующие марковские модели, такие как MegaHAL — модель разработанная Хатченсом и Алдером в 1998 году. Еще одним примером хорошей реализации подобных систем является ALICE, виртуальный собеседник, который был реализован в основном для развлечения пользователей и общения с ними как с реальным человеком. В 2000, 2001 и 2004 годах модель ALICE выиграла конкурс премию Лёбнера.

Другая сфера применения диалоговых систем это практика изучение языков. Здесь проблема отсутствия лингвистических знаний и невозможность понять то о чем говорит пользователь еще больше приводит к тому что многие ответы не будут иметь явного смысла. Для того чтобы выяснить,

может ли виртуальный собеседник заменить диалогового партнера для пользователей, изучающих иностранные языки, Джиа и соавторами был проведен эксперимент: студентов из университетов и колледжей попросили пообщаться с ALICE и сказали им что они разговаривают с партнером, который будет помогать им изучать английский язык. После некоторого времени большинство студентов выяснило, что ALICE была диалоговым агентом, а не настоящим человеком. Анализ всех полученных диалогов и отзывов от студентов показал что общее количество пользователей говоривших с ALICE составило 1256 человек. 88% из них беседовали только один раз и больше не возвращались на сайт. Продолжительность одного диалога была короткой; 17% респондентов высказали положительные отзывы, 24% оценили эксперимент отрицательно. Темы диалогов охватывали многие аспекты повседневной жизни, такие как учеба, эмоции, компьютер, свободное время, путешествия и работа. 11,39% студентов говорили об изучении английского языка и экзаменах, 13% о любви. Студенты младше 30 лет в основном относились к ALICE как к другу, а не как к учителю, и рассказали ей о некоторых частных эмоциональных проблемах и переживаниях. По итогам эксперимента был сделан вывод что алгоритм сопоставления слов без какой-либо попытки понять сказанное, использовавшийся в модели ALICE, не подходит для целей построения диалогового ассистента. После этого Джиа в 2004 разработала интеллектуальную систему для обучения иностранному языку, состоящую из разметки естественного языка, маркирующая элементы грамматики, объектная модель Java, которая представляет собой набор грамматических элементов, база данных естественного языка, механизм ответа на коммуникацию, рассматривающий контекст диалога, личность пользователя и самой системы. В контексте предыдущего эксперимента Чен Вонг в 2005

году в своей работе указал на то, что «ответы большинства виртуальных собеседников часто предсказуемы, избыточны, лишены какой-либо личности и не хранят информацию о предыдущих ответах, что может привести к диалогу с повторяющимися элементами». С другой стороны, Фрайер и Карпентер в работе 2006 года утверждают, что «виртуальные собеседники могут помочь в практике языковых навыков для студентов в любое время и практически в любом месте». Несмотря на то, что большинство виртуальных собеседников не могут обнаружить орфографические и грамматические ошибки, они все равно могут быть полезны для студентов у которых уже есть некоторый опыт изучения языка. Фрайер и Карпентер провели эксперимент, в котором 211 ученика попросили пообщаться с виртуальными агентами ALICE и Jabberwocky. Положительная обратная связь заключалась в том, что студенты чувствовали себя более комфортно и расслабленно разговаривая с виртуальным собеседникам чем со знакомыми по университету или учителем. Авторы перечисляли другие преимущества виртуальных агентов в рассматриваемом контексте: агент мог повторять один и тот же материал со студентами несколько раз, не уставая, многие агенты могли использовать в ответах текстовый или речевой режим, в зависимости от необходимости практиковать чтение или навыки слушания, а также сам характер виртуального собеседника как агента, представляющего новые тенденции в технологиях улучшают мотивацию студентов к обучению. В дополнение к этому, учителя могут поощрять студентов, которые рано закончили свою классную работу, поговорить с виртуальным агентом и дать им определенную тему для практики. Анализ результатов может быть получен через хранимую расшифровку беседы, которая может помочь учащимся самостоятельно оценить себя.

В 2004 году Книлл и соавторы разработали виртуального собеседника Sofia для того, чтобы помочь в преподавании математики. У Sofia есть возможность как общаться с пользователями так и с другими математическими виртуальными агентами: Pari и Mathmatica. Информация Sofia содержит текстовые файлы, в основном ориентированные на математику и другие общие знания, для того чтобы сделать виртуального собеседника более дружественной по отношению к пользователю, для чего Sofia обучали на шутках и фильмах, в которых математика играет центральную роль. Виртуальный собеседник использовался на кафедре математики Гарвардского университета. Результаты показали, что учителя могут использовать агента для поиска новых задач, тогда как учащиеся могут использовать его для решения этих задач. Исследователи, занимающиеся информационным поиском, признают, что методы, применяемые для генерации ответов на вопросы из наборов документов, имеют широкое применение не только в образовании. Одной из наиболее ранних систем, созданных для ответа на вопросы, была система разработанная Виленски в 1988 году, чей информационный домен касался операционной системы Unix. Гиббс использовал виртуального собеседника ALICE для изучения социальной теории через создание базы знаний и способная отвечать на вопросы вида «расскажи мне больше». Модель была опробована студентами социологии, изучающими курс естественной теории. В 2006 году Шумахер и соавторы переучили ALICE на 298-ми, связанными с телекоммуникациями, определениях. Экспериментальная система была отнесена к другому разделу вводного курса «Управление информационной системой». Оценки и результаты показали, что «диалоговая система ALICE является многообещающей разработкой, работа с которой помогает в поиске и приобретении знаний». Тем не менее, использование виртуального

собеседника в качестве информационно-поисковой системы не ограничивается только образовательной сферой. Диалоговая система YPA, разработанная Крушвиц и соавторами в 1999 и использующая естественный язык, позволяет пользователям получать информацию из «Желтых страниц». «Желтые страницы» содержат рекламные объявления с именем рекламодателя и контактной информацией. Система YPA возвращает адреса, и если адрес не найден, система начинает диалог в котором она запрашивает у пользователей более подробную информацию, для того чтобы предоставить пользователю требуемый адрес. YPA состоит из Dialog Manager, компонента Query Construction Component и базы данных Backend. Реляционная база данных содержит таблицы с информацией, извлеченные из «Желтых страниц». Диалог начинается с ввода пользователем запроса через графический интерфейс, после чего менеджер диалога отправляет пользовательского запроса в Natural Language Fronted для синтаксического анализа. После этого результат парсинга отправляется в Query Constructed Component, который преобразует данные в запрос базы данных Backend и возвращает извлеченный адрес. Если адрес не найден, менеджер диалогов начинает задавать вопросы пользователю, чтобы получить больше информации. Чтобы оценить систему YPA, 75 запросов были извлечены из базы данных, после чего возвращаемые адреса были проверены на соответствие с тестовым списком ответов. Были оценены количество соответствующих исходному запросу возвращаемых ответов, количество необходимых для диалога шагов и общее количество отобранных адресов. Результаты показывают что 62 из 75 запросов удалось вернуть адреса, и 74% этих адресов были релевантны исходному запросу. YPA способна отвечать на такие вопросы как «Мне нужен водопроводчик с экстренной службой?», «Какие рестораны есть в средней школе Колчестера?» В другом примере

Уэббер разработал аналогичного виртуального собеседника на основе системы ALICE под названием Virtual Patient bot. VPbot имитирует пациента, с которым могут беседовать студенты-медики. VPbot имел успех в виртуальной программе Гарвардской школы. Студенты, которые использовали виртуального пациента, набрали более высокие оценки на экзаменах.

Еще один тип виртуальных собеседников, широко использующийся сегодня, представляют собой систему навигации которая помогает пользователям получать доступ к сайтам электронной коммерции для поиска релевантной информации о продуктах и услугах. Одной из первых подобных систем стал Happy Assistant, разработанный Чаи и соавторами в 2000 году. Система состоит из трех основных модулей: менеджера презентаций, диспетчера диалогов и диспетчера действий. Диспетчер презентаций применяет метод неглубокого парсинга информации подающейся на вход пользователем для идентификации семантической и синтаксической информации, могущей представлять интерес. Затем он преобразует эту информацию в хорошо сформированное XML-сообщение, называемое логической формой. Менеджер диалога отвечает за сопоставление понятий с запросом пользователя с бизнес-правилами, находящимися в домене знаний. Бизнес-правила состоят из списка понятий вместе с метаданными о целевом продукте или услуге. Если совпадение найдено, веб-страница, связанная с этим правилом, предоставляется пользователю. В противном случае наиболее важная недостающая концепция определяется путем задавания дополнительных вопросов пользователю. После этого идет обращение к диспетчеру действий, который находит соответствующий запросу продукт, и, в случае особых условий пользователя, применяется алгоритм сортировки для получения ранжированного списка продуктов. Чтобы пользователи

доверяли системе, система должна предоставить отчет перед получением результата, в котором система суммирует запрос пользователя, перефразируя его с использованием истории контекста. Результаты показали, что пользователи оценили систему Happy Assistant как простую в использовании и удовлетворяющую их потребностям. Пользователям также положительно отметили возможность общаться с системой на своем язык, тот факт что компьютер сделал для них всю работу, и сокращение времени взаимодействия с системой. Другими подобными виртуальными собеседниками являются Sanelma, разработанный в 2003 году вымышленный ассистент музея, предоставляющий справочную информация об определенном произведении искусства, Rita, технический помощник в режиме реального времени в Интернете, работающая в банке ABN AMRO, графический аватар eGain, который помогает клиенту выполнять некоторые финансовые задачи, такие как денежные переводы.

В этой главе были рассмотрены несколько систем виртуальных собеседников, эффективно применяющихся сегодня в различных практических областях, таких как образование, поиск информации, бизнес, электронная коммерция и развлечения. Можно с большой уверенностью предположить что в будущем виртуальные собеседники могу также найти себя в качестве говорящих книг для детей, помощников в обучении иностранным языкам и обучения в целом. Однако в области образования, многими разработчиками были сделаны выводы на данный момент именно учитель является основой учебного процесса, а диалоговые агенты могут играть роль дополнительных средств обучения, но не заменять хорошего преподавателя. Сегодня целью разработчиков виртуального собеседника должно быть создание инструментов, которые помогают людям, облегчают их работу и их взаимодействие с компьютерами с помощью использования

естественного языка. В своей работе 1999 года Колби пишет: «Нам не нужно рассматривать диалог человека с человеком в качестве золотого стандарта обмена информацией. Если бы кто-нибудь создал идеальную симуляцию человеческого разговора, тогда это был бы разговор человека с человеком, а не человека с машиной — разговор с его иногда странными, но уместными свойствами».

1.2 Постановка задачи

В предыдущей главе была рассмотрена история разработки и применения виртуальных агентов. Для решения текущей задачи разработки виртуального собеседника необходимо использовать специальные библиотеки для реализации архитектуры сети долгой краткосрочной памяти.
Цель: создать средство, которое предоставляет удобный набор функций, позволяющих работать с виртуальным собеседником и решать конкретные задачи, такие как преобразование информации от пользователя в формат, использующийся нейронной сетью, получение ответа от модели и трансформация ответа в текстовую форму. Также важно реализовать возможность сохранения информации для ее дальнейшего анализа и обучения модели.

Исходя из этого необходимо реализовать следующие задачи:

1. Сделать обзор существующих методов и средств разработки и реализации виртуального собеседника.
2. Изучить средства для работы с нейронными сетями долгой краткосрочной памяти.
3. Изучить средства для работы с векторными представлениями слов.
4. Изучить задачи, связанные с генерацией обучающей выборки, в объеме, достаточном для обучения модели.

5. Разработать требования к библиотеке для работы с нейронными сетями долгой краткосрочной памяти.
6. Выполнить обоснование выбора инструментальных средств для реализации библиотеки.
7. Выполнить программную реализацию и тестирование библиотеки.

Глава 2. МЕТОДЫ ПОСТРОЕНИЯ ВИРТУАЛЬНОГО СОБЕСЕДНИКА

2.1 Проблемы существующих средств моделирования последовательностей данных

Нейронные сети это мощные обучающиеся модели, с помощью которые можно достигнуть высоких результатов в широком диапазоне контролируемых и неконтролируемых задач машинного обучения. Нейронные сети особенно подходят для решения задач машинного восприятия, где скрытые базовые характеристики, как правило, не интерпретируются индивидуально. Такой результат объясняется способностью нейронных сетей обучаться иерархическим представлениям характеристик, в отличии от традиционных методов машинного обучения, которые используют характеристики, сконструированные вручную. За последние несколько лет хранилища данных стали более доступными, сами данные стали намного больше, а область параллельных вычислений значительно продвинулась вперед. В случае использования больших наборов данных простые линейные модели, как правило, не обучаются в достаточной степени и часто не в полной мере используют вычислительные ресурсы. С другой стороны, методы глубокого обучения, в частности те, которые основаны на сетях глубокого доверия (deep belief network, DNN) [8] и которые работают на основе жадного алгоритма, использующий стекинг ограниченных машин Больцмана и сверточных нейронных сетей, продемонстрировали хорошие результаты в различных приложениях.

Однако, несмотря на все свои возможности, стандартные нейронные сети также имеют определенные ограничения. В первую очередь, они

полагаются на предположение о независимости между обучающей выборкой и целевой величиной. После обработки каждого примера (экземпляра данных) текущее состояние сети полностью теряется. Если экземпляры данных независимы друг от друга, это не создает проблем, но если примеры обучающей выборки связаны во времени или пространстве, применение стандартной нейронной сети становится невозможным. Кадры из видео, фрагменты аудио сигналов и последовательности слов, взятых из предложений, являются примерами, к которым допущение об их независимости не может быть применено. Кроме того, стандартные сети обычно работают с допущением того, что экземпляры данных представляют собой векторы фиксированной длины, что также неприменимо в случае с последовательностями данных. Таким образом, появилась необходимость расширения функционала этих мощных обучающих инструментов для работы с данными, представляющими временную или последовательную структуру, или являющимися последовательностями неопределенной длины. Таким развитием стандартных нейронных сетей стали рекуррентные нейронные сети (англ. Recurrent Neural Networks, RNN), модели, способные выборочно передавать информацию по всей последовательности данных, при этом обрабатывая эти данные по одному элементу за раз. В результате, ввод и вывод стало возможно представлять как последовательности элементов зависимыми друг от друга.

Для чего это необходимо? Метод опорных векторов, логистическая регрессия и сети прямого распространения показали хорошие результаты без явного моделирования последовательностей во времени. Скорее всего, именно положение о независимости входных и выходных данных привело к значительному прогрессу в машинном обучении. Кроме того, многие модели неявно моделируют временные зависимости, объединяя каждую итерацию

входной информации с некоторым количеством ее непосредственных предшественников на прошлых итерациях, формируя модель машинного обучения с скользящим окном контекста для каждой интересующей точки. Этот подход использовался в сетях глубокого доверия для моделирования речи Маасом и соавторами в 2012 году. К сожалению, несмотря на все положительные стороны которые дает предположение о независимости входных и выходных данных, это полностью исключает моделирование долгосрочных зависимостей. Например, модель, прошедшая обучение с использованием контекстного окна конечной длины длиной 5, не способна дать ответ на вопрос о значении точки 6 шагов назад. В случае практического применения модели, такого как автоматизация центра обработки вызовов, такая ограниченная система может научиться маршрутизировать вызовы, но будет не в состоянии участвовать в продолжительном диалоге.

Рекуррентные нейронные сети это не единственные модели, способные представлять временные зависимости. Цепи Маркова, моделирующие переходы между состояниями в наблюдаемой последовательности, впервые были описаны математиком Андреем Марковым в 1906 году. Скрытые марковские модели, которые моделируют наблюдаемую последовательность как вероятностно зависимую от последовательности ненаблюдаемых состояний, были описаны в 1950-х годах и широко изучались с 1960-х годов. Однако традиционные марковские модели ограничены их состояниями, взятыми из дискретного пространства с небольшим размером S. Алгоритм динамического программирования, который используется для получения логического вывода в скрытых марковских моделях, масштабируется во времени как $O(|S|^2)$. Кроме того, матрица переходов, фиксирующая вероятность перемещения между любыми двумя соседними состояниями

модели, имеет размер $|S|^2$. Таким образом, любые стандартные операции становятся неосуществимыми при росте набора возможных скрытых состояний. Более того, каждое скрытое состояние может непосредственно зависеть только от предыдущего состояния. При том что модель Маркова теоретически можно расширить для учета информации в большем контекстном окне с помощью создания нового пространства состояний, равное векторному произведению возможных состояний в каждый момент времени в окне, эта процедура экспоненциально расширяет пространство состояний согласно размеру окна, что приводит к непрактичности использования моделей Маркова с точки зрения вычислений при моделирования долгих зависимостей.

2.2 Рекуррентные нейронные сети

Рекуррентные нейронные сети являются развитием концепции нейронных сетей прямого распространения, дополненными направленными ребрами, соединяющими последовательность смежных временных периодов. Подобно сетям прямого распространения, рекуррентные нейронные сети могут не иметь циклов среди обычных ребер, однако ребра, соединяющие смежные временные интервалы, и носящие названия рекуррентных ребер, могут образовывать циклы, включая циклы единичной длины, которые представляют собой замыкание ребра на себя во времени. В момент времени t узлы с рекуррентными ребрами принимают на вход данные от текущего значения $x^{(t)}$ и значения $h^{(t-1)}$ скрытого узла в предыдущем состоянии сети. Выход $y'^{(t)}$ в момент времени t вычисляется с учетом значений скрытых узлов $h^{(t)}$ в момент времени t. Значение $x^{(t-1)}$ на входе в момент времени $t-1$ может влиять на выход $y'^{(t)}$ в момент времени t путем повторного рекуррентного соединения. Два уравнения определяют вычисления,

необходимые для всех необходимых значений в каждый момент времени на прямом проходе в рекуррентной нейронной сети:

$$h^{(t)} = (W^{hx}x^{(t)} + W^{hh}h^{(t-1)} + b_h)$$

$$y^{(t)} = softmax(W^{yh}h^{(t)} + b_y)$$

Здесь W^{hx} это матрица весов между входом и скрытым слоем, а W^{hh} — матрица рекуррентных весов между скрытым слоем замкнутым на себя на смежных временных переходах. Векторы b_h и b_y являются параметрами смещения которые позволяют каждому узлу учитывать смещение при обучении (см. рис. 2.1).

Архитектура рекуррентной сети может быть интерпретирована не как циклическая, а как глубокая сеть с одним слоем на каждый шаг времени и общими весами для смежных временных шагов. Из этого следует, что развернутая во времени рекуррентная сеть может быть обучена на последовательности временных шагов, используя метод обратного распространения. Концепция метода обратного распространения во времени (англ. Backpropagation through time, BPTT), была впервые представлена Вербо в 1990 году. Данные алгоритм используется всеми современными рекуррентными нейронными сетями. Основополагающие исследования рекуррентных сетей начались в 1980-х годах. В 1982 году Хопфилд представил семейство рекуррентных нейронных сетей, обладающих возможностями распознавания образов. Характеристиками сети являются значения весов между слоями, с функцией порогового значения в нуле. Сети Хопфилда могут применяться для восстановления поврежденной

информации и являются предшественниками машин Больцмана и авто энкодеров.

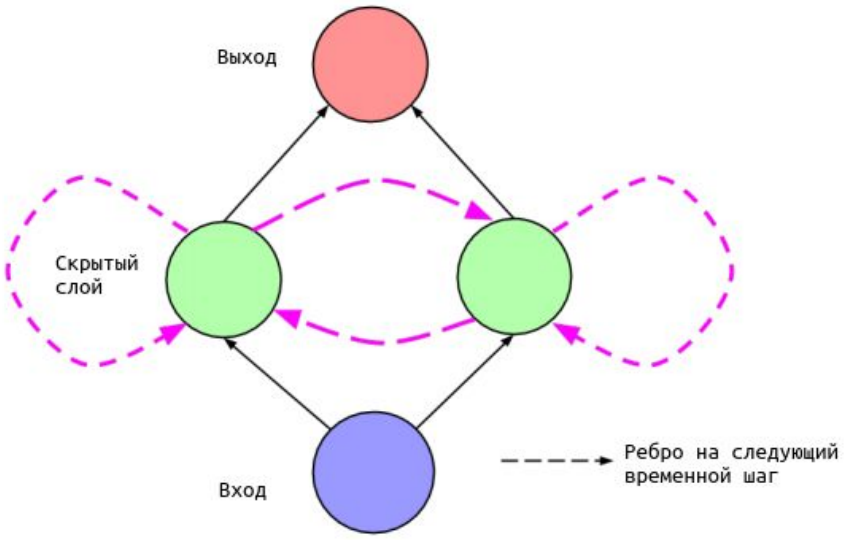

Рис. 2.1 Схема рекуррентной нейронной сети. На каждом временном переходе t активация передается вдоль сплошных ребер, также как и в сети прямого распространения. Пунктиром показаны ребра, соединяющие текущий узел в каждый момент времени t и целевой узел в каждый последующий момент времени $t+1$.

Ранняя архитектура нейронной сети для обучения с учителем, использующая последовательности данных, была представлена Джорданом в 1986 году. Такая сеть представляет собой сеть прямого распространения с одним скрытым слоем, расширяемым специальными элементами. Значения выходных узлов передаются в специальные элементы или блоки, которые затем передают эти значения скрытым узлам на следующем временном шаге (см. рис. 2.2).

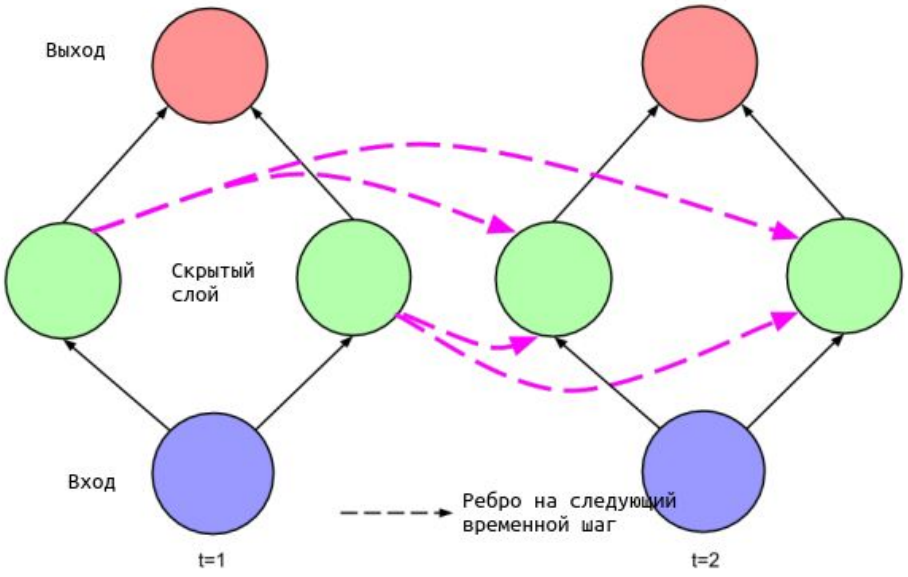

Рис. 2.2 Развернутая во времени рекуррентная сеть.

Если выходные значения являются действиями, специальные блоки позволяют сети запоминать эти действия, предпринятые на предыдущих временных шагах. Саскевер и соавторы в своей работе 2014 года, посвященной проблеме перевода между естественными языками, демонстрируют генерацию текстовой последовательности слов в подобной сети, в которой выбранное на каждом временном шаге слово, подается в сеть в качестве входного сигнала на следующем временном шаге. Специальные блоки этой сети соединены друг с другом, что, в свою очередь, позволяет ребрам сети передавать информацию на несколько временных шагов вперед без пересечения с ребрами вывода на каждом промежуточном шаге.

Архитектура сети, предложенная Эльманом в 1990 году, имеет более простую структуру, чем архитектура, разработанная Джорданом. Здесь, каждый элемент скрытого слоя связывается с контекстным блоком. Каждый такой блок j' принимает входное состояние соответствующего скрытого узла j на предыдущем временном шаге с помощью ребра с фиксированным весом $w_{j'j} = 1$.

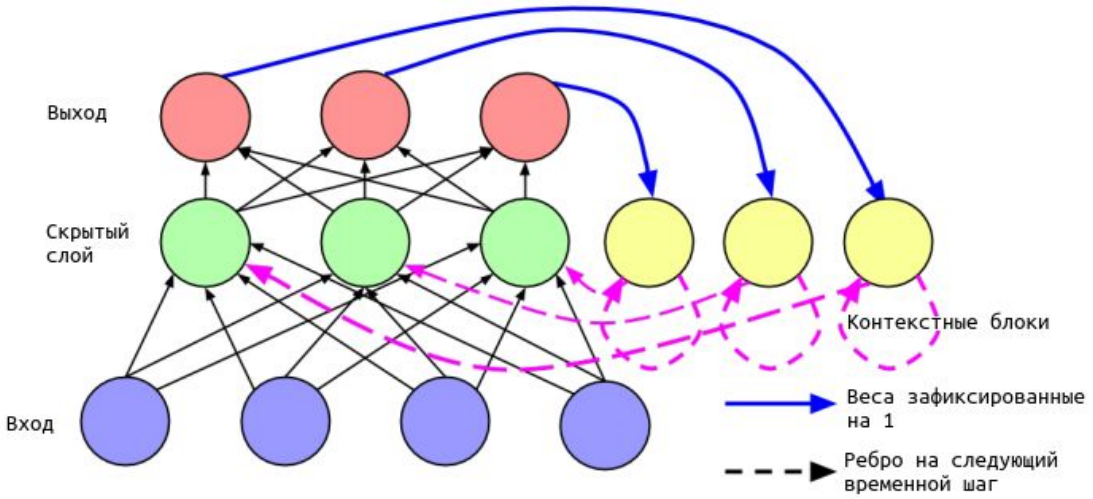

Рис. 2.3 Рекуррентная нейронная сеть, описанная в работе Джордана в 1986 году. Выходные блоки подключаются к специальным блокам, которые на следующем шаге подают информацию как в себя, так и в скрытые блоки.

Это значение затем возвращается обратно в тот же скрытый узел j с помощью стандартного ребра сети. Такая архитектура эквивалентна первоначальной архитектуре рекуррентной нейронной сети, в которой каждый скрытый узел имеет одно рекуррентное ребро, замкнутое на себя. Идея рекуррентных ребер с фиксированными весами, использующими концепцию скрытых узлов, замкнутых на себя, имеет основополагающее значение в последующих работах, посвященных сетям долгой краткосрочной памяти (см. рис. 2.3).

В своей работе 1990 года Эльман тренирует нейронную сеть, используя метод обратного распространения, и демонстрирует, что такая сеть может обучаться распознаванию временных зависимостей. В статье представлены два вида экспериментов. Первый применяет концепцию логических операций XOR на задачи временной области путем объединения последовательностей из трех текстовых элементов или токенов.

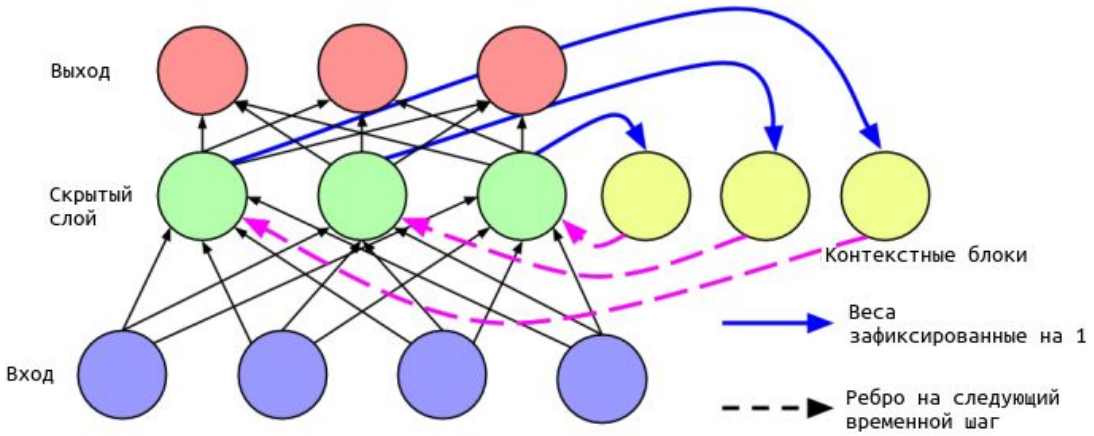

Рис. 2.4 Рекуррентная нейронная сеть, описанная Эльманом в 1990 году. Скрытые блоки подключаются к контекстным блокам, которые возвращаются в скрытые блоки на следующем временном шаге.

Для каждой последовательности, состоящей из трех токенов, например. «011» первые два токена («01») выбираются случайным образом, а третий токен («1») определяется с помощью выполнения логической операции XOR на первых двух токенах. Случайное угадывание в таком случае должно достигать точности 50%. Идеальная система должна выполнять такого рода операции с точностью не меньшей точности случайного угадывания первых двух токенов и определять третий токен уже с точностью 66,7%. Простая сеть Эльмана действительно способна достигать этого максимально возможного результата (см. рис. 2.4).

Обучение рекуррентных нейронных сетей долгое время считалось трудной задачей [11]. Даже для обычных сетей прямого распространения задача оптимизации является NP-полной задачей. Для рекуррентных сетей обучение осложняется фактом необходимости использования долгосрочных зависимостей на что было указано Бенджио и соавторами в 1994 году, а в

дальнейшем более полно описано в 2001 году в работах Хокрейтера. Расчет обратного распространения ошибки для долгосрочных зависимостей по множеству временных шагов приводит к проблеме исчезающего или экспоненциально разрастающегося градиента. В качестве примера можно рассмотреть нейронную сеть с одним входным узлом, одним выходным узлом и одним рекуррентным узлом скрытого типа. На вход передаются входные данные в момент времени τ, после чего в момент времени t рассчитывается ошибка, с учетом того что вводные данные на промежуточных временных шагах равны нулю. Соотношение весов во временных шагах означает, что рекуррентное ребро скрытого узла j всегда имеет одинаковое значение веса.

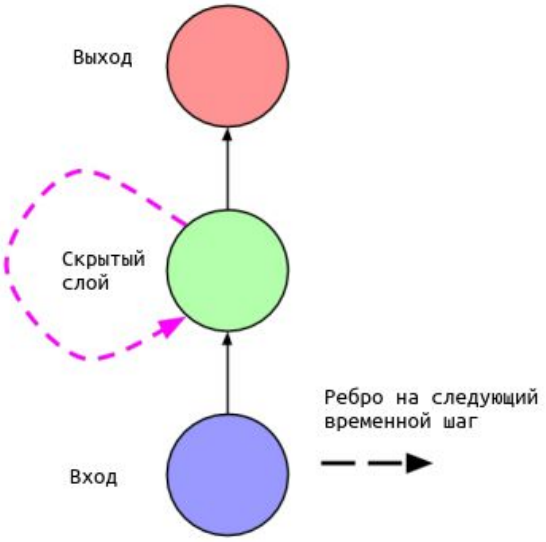

Рис. 2.5 Простая рекуррентная сеть с с одним входным узлом, одним выходным узлом и одним рекуррентным узлом скрытого типа.

Следовательно, добавленная величина от входных данных в момент времени τ к выходному сигналу в момент времени t будет либо разрастаться, либо стремиться к нулю, по мере того как $t-\tau$ будет расти. Следовательно, производная от ошибки по отношению к переменной, поданной на вход

будет также или экспоненциально расти, или экспоненциально стремиться к нулю.

Какое из этих двух событий произойдет, зависит от того, будет ли вес рекуррентного ребра $|w_{jj}| > 1$ или $|w_{jj}| < 1$ и от функции активации в скрытом узле. Учитывая сигмоидный характер функции активации, проблема исчезающего градиента еще более осложняется. Проблему исчезающего градиента можно решить с помощью функции активации с выпрямленным линейным модулем $max(0, x)$. В 2012 году Паскану и соавторы в своей работе дали полное математическое обоснование проблем исчезающих и разрастающихся градиентов, и описали точные условия, при которых могут возникать эти проблемы. Учитывая эти условия, они предлагают подход к обучению через блок регуляризации, который будет принуждать веса нейронной сети принимать значения только в тех случаях, где градиент не исчезает и не разрастается.

Усеченный метод обратного распространения ошибки во времени (анг. Truncated backpropagation through time, TBPTT) является одним из решений проблемы расширяющегося градиента для рекуррентных сетей использующихся на постоянной основе. Этот метод впервые был представлен Уильямсом, Зипцером и соавторами в 1989 году. Метод устанавливает некоторое максимальное количество шагов по времени, на которых будет использоваться метод обратного распространения ошибки. Но несмотря на то что такой подход может использоваться для решения проблемы расширяющегося градиента, это становится причиной другой проблемы — невозможности изучения долгосрочных зависимостей. Архитектура рекуррентной нейронной сети долгой краткосрочной памяти, которая будет рассмотрена ниже, использует узлы с рекуррентными ребрами

и фиксированными весами в качестве решения проблемы исчезающего градиента.

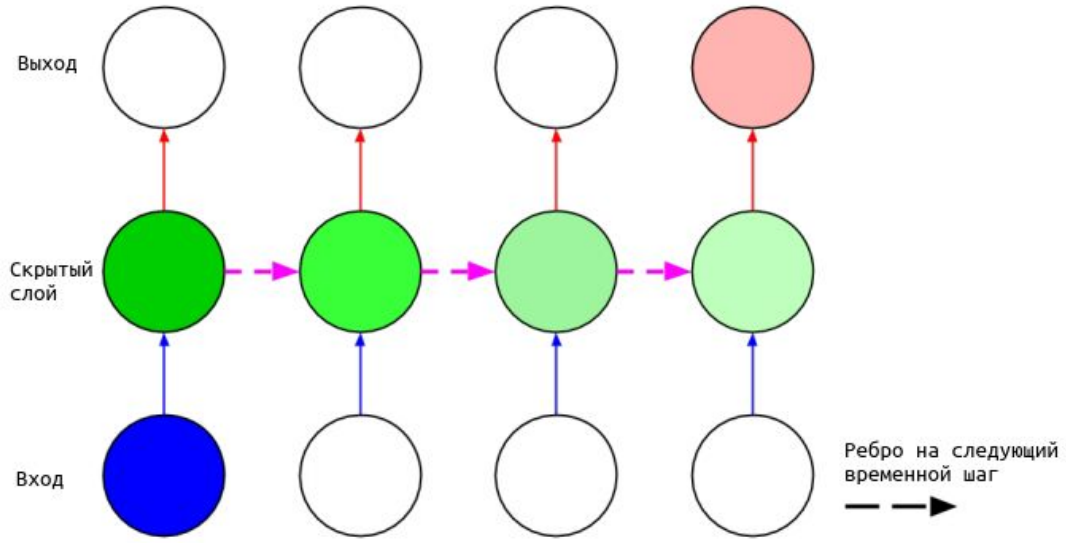

Рис. 2.6 Визуализация проблемы исчезающего градиента в случае с использованием рекуррентной сети, изображенной на рисунке 2.6. Если вес вдоль рекуррентного ребра меньше единицы, добавленная величина входной величины на первом шаге по времени к выходной величине на конечном шаге по времени будет экспоненциально убывать, пропорционально длине временного интервала между ними.

В целом, наряду с улучшенными архитектурами нейронных сетей, которые будут описаны ниже, более эффективные реализации алгоритмов и более эффективные эвристики расчета градиентов сделали обучение рекуррентных нейронных сетей возможным. Использованием графических процессоров для расчета обратного распространения ошибки, применяющихся в таких библиотеках как Theano, разработанной Бергстра и соавторами в 2010 году, а также библиотекой Torch, разработанной Коллбертом в 2011 году, сделало возможной реализацию быстрых алгоритмов обучения рекуррентных нейронных сетей. В 1996 году, до

появления нейронных сетей долгой краткосрочной памяти, попытки обучить рекуррентные сети показывали результаты не лучше, чем случайное угадывание. Тем не менее, сегодня рекуррентные нейронные сети в некоторых задачах могут быть обучены с использованием малого числа графических процессоров и за довольно короткое время.

2.3 Архитектура долгой краткосрочной памяти

Наиболее успешные архитектуры рекуррентных нейронных сетей для обучения на последовательностях данных основаны на двух работах, опубликованных в 1997 году. В первой статье «Долгая краткосрочная память» Хохрейтера и Шмидхубера [30] представлена ячейка памяти, блок вычислений, который заменяет традиционные узлы в скрытых слоях сети. С помощью этих блоков памяти сети могут преодолевать те проблемы во время обучения, которые характерны для ранних архитектур рекуррентных нейронных сетей (см. рис. 2.3.1). В второй статье «Двунаправленные рекуррентные нейронные сети» Шустера и Паливала, представлена архитектура, в которой информация из будущего и прошлого используется для определения результата вычисления в любой точке последовательности. Такая архитектура отличается от предыдущих архитектур сетей, где только входная информация из прошлого могла влиять на результат. Новый метод был успешно использован для задач маркировки последовательностей при обработке естественного языка. Эти два нововведения не являются взаимоисключающими, что позволило их объединить для классификации фонем в работе Грейвса и Шмидхубера 2005 года и распознавания рукописного текста в работе Грейвса в 2009 году.

Хохрейтер и Шмидхубер реализовали модель долгой краткосрочной памяти в первую очередь для решения проблемы исчезающих градиентов.

Эта модель похожа на стандартную рекуррентную нейронную сеть со скрытым слоем, но каждый обычный узел в скрытом слое заменяется ячейкой памяти.

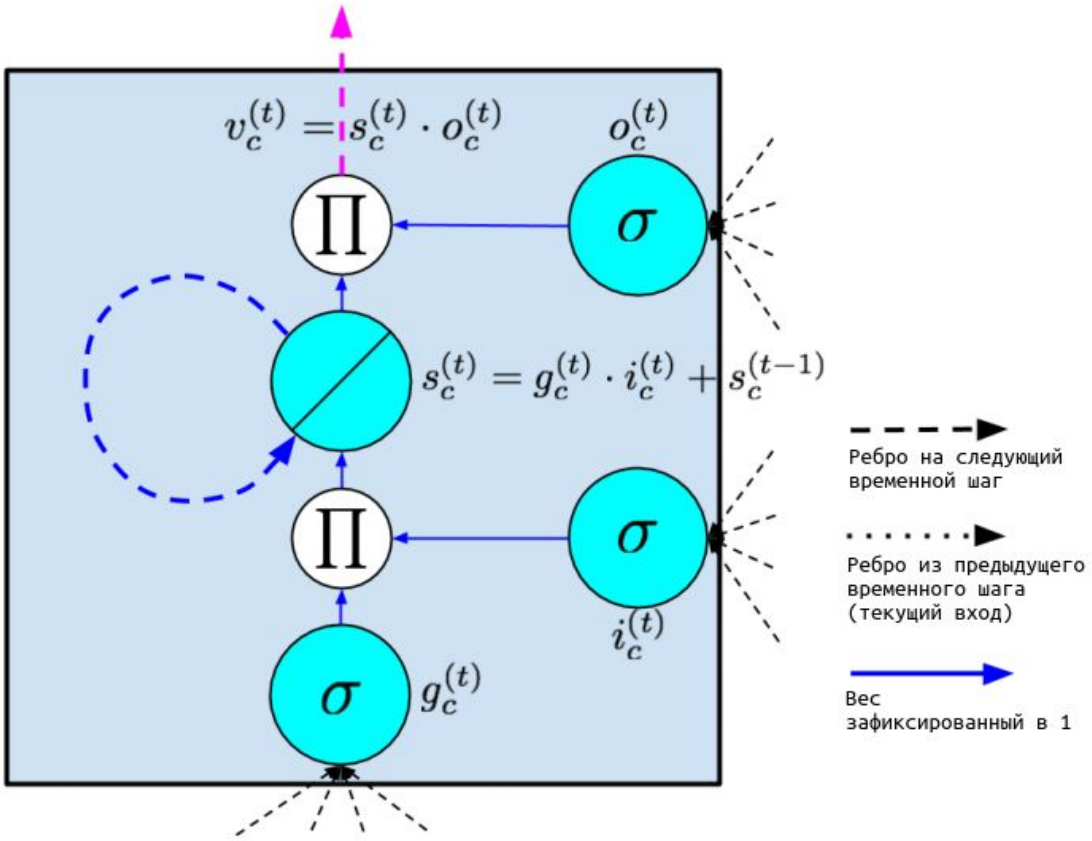

Рис. 2.7 Один блок памяти сети долгой краткосрочной памяти, предложенная Хохрейтером и Шмидхубером в 1997 году. Замкнутым на себя узлом является внутреннее состояние s. Диагональная линия указывает, что функция линейна, т.е. применяется метод точечного связывания. Синяя пунктирная линия - это рекуррентное ребро, имеющее фиксированный вес. Результатом вычислений узлов, обозначенных Π, является скалярное произведение их входных данных. Все ребра в и из этих узлов также имеют фиксированный вес.

Каждая ячейка памяти содержит узел с замкнутым на себя рекуррентным ребром с фиксированным весом, гарантирующий, что градиент может

проходить через множество временных шагов без экспоненциального исчезновения или расширения. Далее, чтобы отличить обозначение ячеек памяти, от обычных узлов нейронной сети, будет использоваться индекс c.

Термин «долгая краткосрочная память» происходит из следующей концепции: обычные рекуррентные нейронные сети реализуют модель долговременной памяти в виде весов. Во время обучения веса постепенно меняются, кодируя общее представление о данных. Одновременно с этим рекуррентные сети реализуют модель кратковременной памяти в виде активационных функций, передающих информацию от каждого узла к последующим узлам. Модель долгой краткосрочной памяти использует промежуточный блок памяти с помощью ячейки памяти. Ячейка памяти представляет собой составной блок, построенный из более простых узлов, расположенных в определенном порядке и по определенному шаблону, с включением узлов нового типа, мультипликативных узлов, представленных на диаграммах буквой П (см. рис. 2.8).

Входной узел (англ. Input node), блок с меткой g_c, представляет собой узел, который запускает активацию из входного слоя $x^{(t)}$ на текущем временном шаге (и вдоль рекуррентных ребер) со скрытого слоя на предыдущем шаге времени $h^{(t-1)}$. Как правило, суммированный взвешенный результат вычислений входных данных проходит через функцию активации $tanh$, хотя в оригинальной статье об архитектуре долгой краткосрочной памяти функция активации представляла собой сигмоидальную функцию.

Входной коммутатор (англ. Input gate), является отличительной особенностью архитектуры долгой краткосрочной памяти. Коммутатор это сигмоидальная функция, которая, подобно входному узлу, принимает результат активации текущего элемента данных $x^{(t)}$ и результат активации скрытого слоя на предыдущем временном шаге. Коммутатор назван именно

так, потому что его значение используется для умножения значения другого узла. Коммутатор является затвором в том смысле, что если его значение равно нулю, то поток данных из другого узла не проходит дальше. Если значение коммутатора равно единице, весь поток пропускается. Значение входного коммутатора i_c умножает значение входного узла.

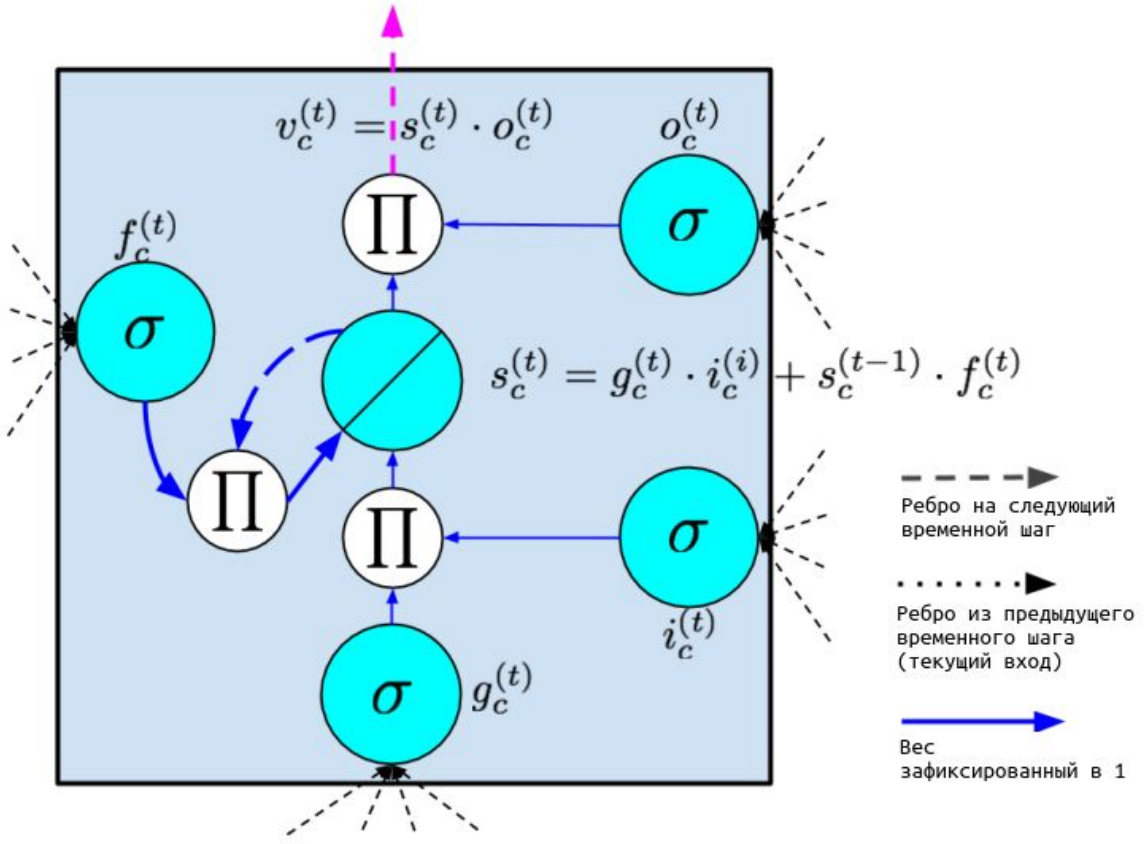

Рис. 2.8 Ячейка памяти рекуррентной нейронной сети долгой краткосрочной памяти с коммутатором забывания, впервые описанная в работе Герса в 2000 году.

Внутреннее состояние (англ. Internal state) представляет собой узел s_c с линейной активацией, находящийся в основе каждой ячейки памяти, и упоминающийся в оригинальной статье как «внутреннее состояние» ячейки. Узел внутреннего состояния s_c состоит из замкнутого на себя рекуррентного ребра с фиксированным удельным весом. Поскольку это ребро включает в

себя результаты смежных временных интервалов, но с постоянным весом, метод обратного распространения ошибки может быть использован по временным шагам без того чтобы градиент стал исчезать или распространяться. Это ребро часто называют каруселью постоянной погрешности. В векторном обозначении корректировка для узла внутреннего состояния выглядит следующим образом:

$$s^{(t)} = g^{(t)} \odot i^{(t)} + s^{(t-1)}$$

где $\odot$ это покомпонентное произведение.

Коммутатор забывания (англ. Forget gate) f_c впервые был представлен Герсом и соавторами в 2000 году. Суть данного метода заключается в способности сети очищать содержимое узла внутреннего состояния. Такая способность особенно полезна в сетях, работающих на длительных временных промежутках. Уравнение для вычисления внутреннего состояния в коммутаторе забывания в прямом распространении можно записать следующим уравнением:

$$s^{(t)} = g^{(t)} \odot i^{(t)} + f^{(t)} \odot s^{(t-1)}$$

Выходной коммутатор (англ. Output gate) это значение v_c, которое в конечном итоге производится ячейкой памяти, представляет собой значение внутреннего состояния s_c, умноженное на значение выходного коммутатора o_c. Как правило внутреннее состояние проходит через функцию активации $tanh$, так как это дает результату на выходе из каждой ячейки тот же динамический диапазон, что и в обычном скрытом элементе. Тем не менее, в других исследованиях нейронных сетей, узлы использующие функцию

активации, представляющую собой выпрямленный линейный модуль и имеющую больший динамический диапазон, обучаются гораздо быстрее. Таким образом, логично допустить что использование любой нелинейной функции в узле внутреннего состояния может быть проигнорировано.

После изначальной разработки модели долгой краткосрочной памяти были представлены несколько модификаций. Коммутатор забывания, описанный выше, был предложен в 2000 году и не был частью оригинального дизайна модели долгой краткосрочной памяти [25]. Доказав свою эффективность, коммутатор забывания, стал стандартным модулем для большинства современных реализаций модели.

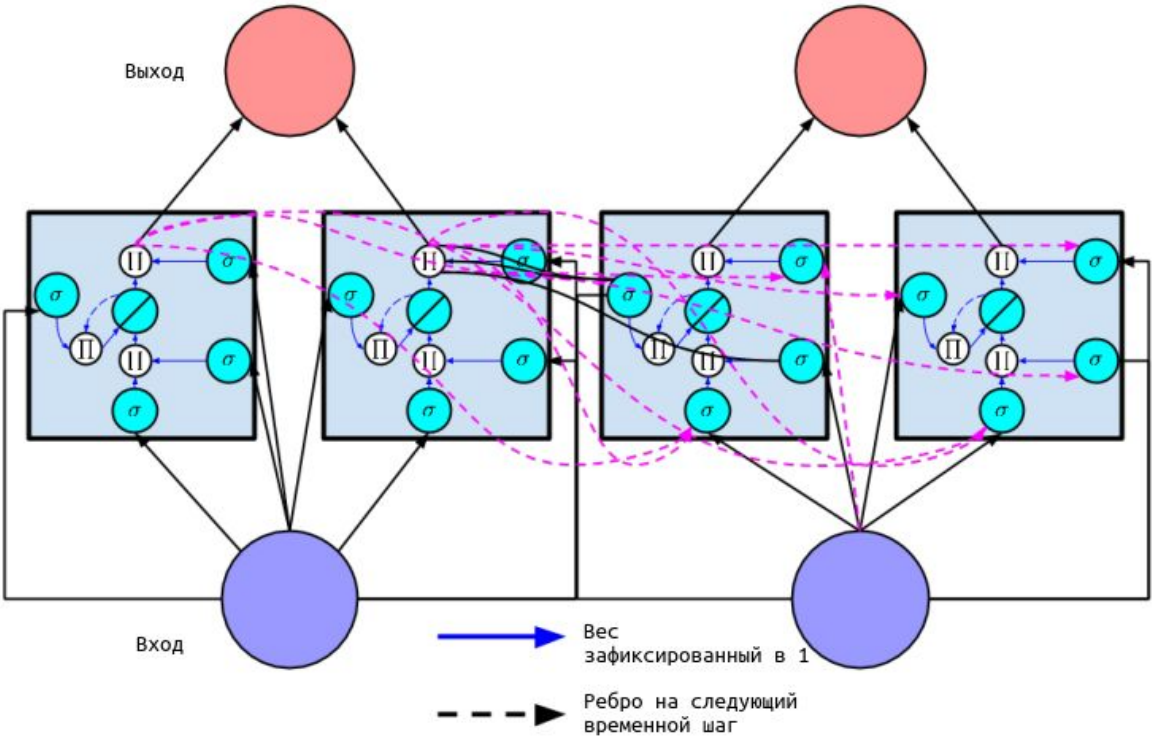

Рис. 2.9 Рекуррентная нейронная сеть со скрытым слоем, состоящим из двух ячеек памяти. Сеть показана развернутой во времени через два шага.

В том же году Герс и Шмидхубер предложили соединения с глазком, которые переходят из внутреннего состояния непосредственно во входные и

выходные коммутаторы этого же узла, без необходимости предварительного модифицирования выходным коммутатором. Авторы делают вывод, что подобные соединения улучшают производительность по задачам синхронизации, в которых сети необходимо обучаться измерять точные интервалы между событиями.

Формально, вычисление в модели долгой краткосрочной памяти происходит в соответствии с расчетами, которые выполняются на каждом временном шаге. Следующие уравнения описывают алгоритм современной модели долгой краткосрочной памяти с коммутатором забывания:

$$g^{(t)} = \phi(W^{gx}x^{(t)} + W^{gh}h^{(t-1)} + b_g)$$

$$i^{(t)} = \sigma(W^{ix}x^{(t)} + W^{ih}h^{(t-1)} + b_i)$$

$$f^{(t)} = \sigma(W^{fx}x^{(t)} + W^{fh}h^{(t-1)} + b_f)$$

$$o^{(t)} = \sigma(W^{ox}x^{(t)} + W^{oh}h^{(t-1)} + b_o)$$

$$s^{(t)} = g(t) \odot i(i) + s(t-1) \odot f^{(t)}$$

$$h^{(t)} = \phi(s(t)) \odot o^{(t)}$$

Здесь вектор $h^{(t)}$ является результатами вычислений скрытого слоя модели долгой краткосрочной памяти в момент времени t, а вектор $h^{(t-1)}$ — результатами вычислений ячейки памяти в скрытом слое в предыдущий временной шаг. Необходимо обратить внимание, на то что уравнения

включают в себя коммутатор забывания , но не включают соединения с глазком. Вычисления для более простой модели долгой краткосрочной памяти без учета коммутаторов забывания получаются установкой $f^{(t)} = 1$ для всех t. Следуя последним предложениям Заремба и Суцкевера, опубликованных в 2014 году, для входного узла g используется функция активации $tanh\ \phi$, в то время как в первоначальной работе использовалась функция активации сигмоидного типа σ.

Во время прямого прохода, модель долгой краткосрочной памяти принимает решение о том, добавлять ли нет результаты активации в узел внутреннего состояния. До тех пор пока входной коммутатор имеет нулевое значение, результат активация не может пройти дальше. Выходной коммутатор, со своей стороны, решает когда результат может быть выведен. Когда оба коммутатора закрыты, результат активации оказывается запертым в ячейке памяти. Он не растет, не уменьшается и не влияет на результаты на промежуточных временных шагах. Во время обратного прохода, карусель постоянной погрешности позволяет вычислять градиент по методу обратного распространения ошибки на необходимое количество временных шагов назад, не давая ему не экспоненциально расти и не распространяться, а коммутаторы обучаются, когда пропускать ошибку и когда давать ей проходить дальше.

2.4 Векторные представления слов

Метод векторных представлений слов, работающий по принципу обучения без учителя, в последние годы получил широкое распространение за счет его успешного применения во многих задачах обработки естественного языка. Полученные результаты были настолько хорошими, что многие модели, традиционно применяемые в данной сфере, такие как

Латентно-семантический анализ (англ. Latent semantic analysis, LSA) и кластеры Брауна, возможно скоро будут полностью заменены моделями, использующими методы векторных представлений слов.

Модели, работающие с векторными пространствами, использовались в дистрибутивной семантике с 1990-х годов. С тех пор ряд моделей, используемых для оценки непрерывных представлений слов, получили дальнейшее развитие, в частности это модели Латентного размещения Дирихле (англ. Latent Dirichlet allocation, LDA) и Латентно-семантического анализа. Термин «векторное представление» первоначально был предложен Бенджио и соавторами в 2003 году, которые получили эти векторные представления в результате обучения нейронной сети. Тем не менее, именно Коллоберт и Уэстон были первыми, кто продемонстрировал возможности векторных представлений слов в своей статье 2008 года «Единая архитектура для обработки естественного языка», в которой они определяют векторные представления слов в качестве высокоэффективного инструмента для решения последующих задач, а также публикуя архитектуру нейронной сети, на которой, в последующем, основывались многие современные подходы. И в 2013 году Миколов с соавторами [27] вывел данную концепцию на передний план путем создания word2vec, инструментария, позволяющего обучать и использовать предварительно обученные векторные представления слов (см. рис. 2.10). Через год Пеннингтон с соавторами представил GloVe, аналогичный набор инструментов для работы с векторными представлениями, что было признаком того, что векторное представление слов внезапно стало одним из основных направлений в машинном обучении [35].

Векторные представления слов в настоящее время считаются одним из немногих успешных применений алгоритмов работающих по принципу

обучения без учителя. Тот факт, что они не требуют дорогой, с экономической точки зрения, ручной маркировки, вероятно, является их главным преимуществом. Скорее наоборот, векторные представления слов могут быть получены из уже доступных не размеченных текстовых данных. Очевидно, что каждая нейронная сеть прямого распространения, которая принимает слова из текстового документа в качестве входных данных и выдает результат в виде векторов в векторное пространство меньшей размерности, которые затем корректируется методом обратного распространения ошибки, неизбежно приводит к векторному представлению слов как результат полученных весов первого слоя, который носит имя вложенного слоя. Ключевое различие между такой сетью и методом word2vec — его вычислительная сложность, которая объясняет, почему до 2013 года векторные представления слов не были сколь либо заметными в сфере обработки естественного языка. Недавнее и быстрое расширение и доступность вычислительных мощностей, безусловно, также дали толчок к их развитию. Цель обучения для GloVe и word2vec также отличается: оба этих инструмента предназначены для создания векторных представлений, которые кодируют общие семантические отношения между словами и могут быть эффективно использованы во многих последующих задачах. Обычные нейронные сети, с другой стороны, обычно создают векторные представления специфические только для текущей задачи, тем самым ограничивая их использования в других задачах.

Для последующего сравнения моделей, представим что наша обучающая выборка состоит из последовательности T слов: $w_1, w_2, w_3, \cdots, w_T$ принадлежащие словарю V размерности $|V|$. Модели, как правильно, используют контекст из n слов. Сопоставим каждое слово с векторным представлением v_w на входе размерности d и векторным

представлением v'_w на выходе. Оптимизация целевой функции J_w будет происходить с помощью параметров модели θ, в результате чего будет получена оценка точности модели $f_\theta(x)$ для каждого x.

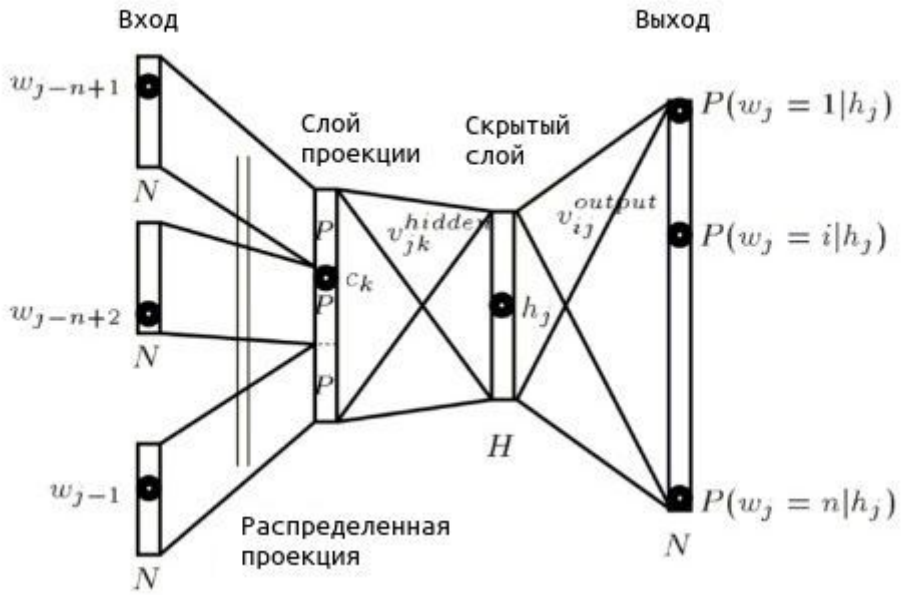

Рис. 2.10 Нейронная лингвистическая модель (Бенджио и соавторы, 2006)

Классическая нейронная лингвистическая модель, предложенная Бенджио и соавторами в 2003 году состоит из нейронной сети прямого распространения с одним скрытым слоем, задача которой заключается в прогнозировании каждого последующего слова в последовательности, где целевая функция описывается уравнением:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log f(w_t, w_{t-1}, \cdots, w_{t-n+1})$$

Здесь $f(w_t, w_{t-1}, \cdots, w_{t-n+1})$ это данные на выходе модели, например вероятности $p(w_t \mid w_{t-1}, \cdots, w_{t-n+1})$ получившиеся в результате использования много переменной логистической активационной функции

софтмакс (англ. Softmax function). Бенджио был один из первых, представившим концепцию того, что впоследствии получило название векторного представления слов: вещественный вектор значений, принадлежащий пространству $\mathbb{R}$. Основа их модели по-прежнему может быть встречена в современных нейронных моделях векторных представлений слов. Главные характеристики модели включают в себя: Embedding Layer, слой который генерирует векторные представления слов, умножая индексный вектор на матрицу векторных представлений слов, Intermediate Layer, один или несколько слоев, которые создают промежуточное представление входных данных, то есть полностью связанный слой, который применяет нелинейную функцию к конкатенации векторных представлений слов n предыдущих слов и Softmax Layer, последний слой, который генерирует распределение вероятности по всем словам в V.

После первоначальных исследований Бенджио, в сфере нейронных лингвистических моделей наступила пауза, поскольку, на тот момент, вычислительная мощность и алгоритмы еще не были на уровне, позволяющем модели обучаться на большом наборе данных. В 2008 году Колберт и Уэстон продемонстрировали, что векторные представления слов, подготовленные на достаточно большом наборе данных, содержат синтаксическое и семантическое значение и улучшают производительность в последующих задачах. Чтобы уйти от вычислений дорогостоящей, с точки зрения используемых ресурсов, функции софтмакс, было предложено использование альтернативной целевой функции. Вместо критерия кросс-энтропии Бенджио, целью которой является максимизация вероятности следующего слова, с учетом предыдущих слов, Колберт и Уэстон учат нейронную сеть на получение более высокого результата f_θ для правильной последовательности слов (вероятная последовательность слов в модели

Бенджио), чем для неправильной. Для этой цели они используют критерий попарного ранжирования, который выглядит следующим образом:

$$J_\theta = \sum_{x \in X} \sum_{w \in V} \max\{0, 1 f_\theta(x) + f_\theta(x^{(w)})\}$$

Далее берется окно x, содержащие N слов из множества всех возможных окон X в корпусе. Для каждого окна x затем создается некорректный образец $x^{(w)}$ путем замены центрального слова в x другим словом w из V. Таким образом целевая функция максимизирует расстояние между результатами, полученными моделью для правильного и неправильного окна с запасом 1 (см. рис 2.11). Результирующая языковая модель создает векторные представления слов, которые обладают многими из тех семантических свойств, которые стали характерны векторным представлениям в дальнейшем, например, страны сгруппированные в кластеры, и близкое местоположение синтаксически похожих слов в векторном пространстве. Хотя и новая ранжирующая целевая функция устраняет вычислительную сложность функции софтмакс, данная модель сохраняет промежуточный, полностью связанный, скрытый слой, присутствующий в оригинальной модели Бенджио, что является еще одним источником сложных вычислений.

Word2Vec, возможно, является самой популярной из моделей векторного представления слов . Поскольку векторные представления слов являются ключевым элементом моделей глубокого обучения для задач обработки естественного языка, считается что они принадлежат к одной и той же группе. Однако word2vec технически не является моделью нейронной сети глубокого обучения, поскольку аргументация состоит в том, что

архитектура word2vec не является ни глубокой, ни использует нелинейности (в отличие от модели Бенджио и модели Колберта и Уэстона) [27].

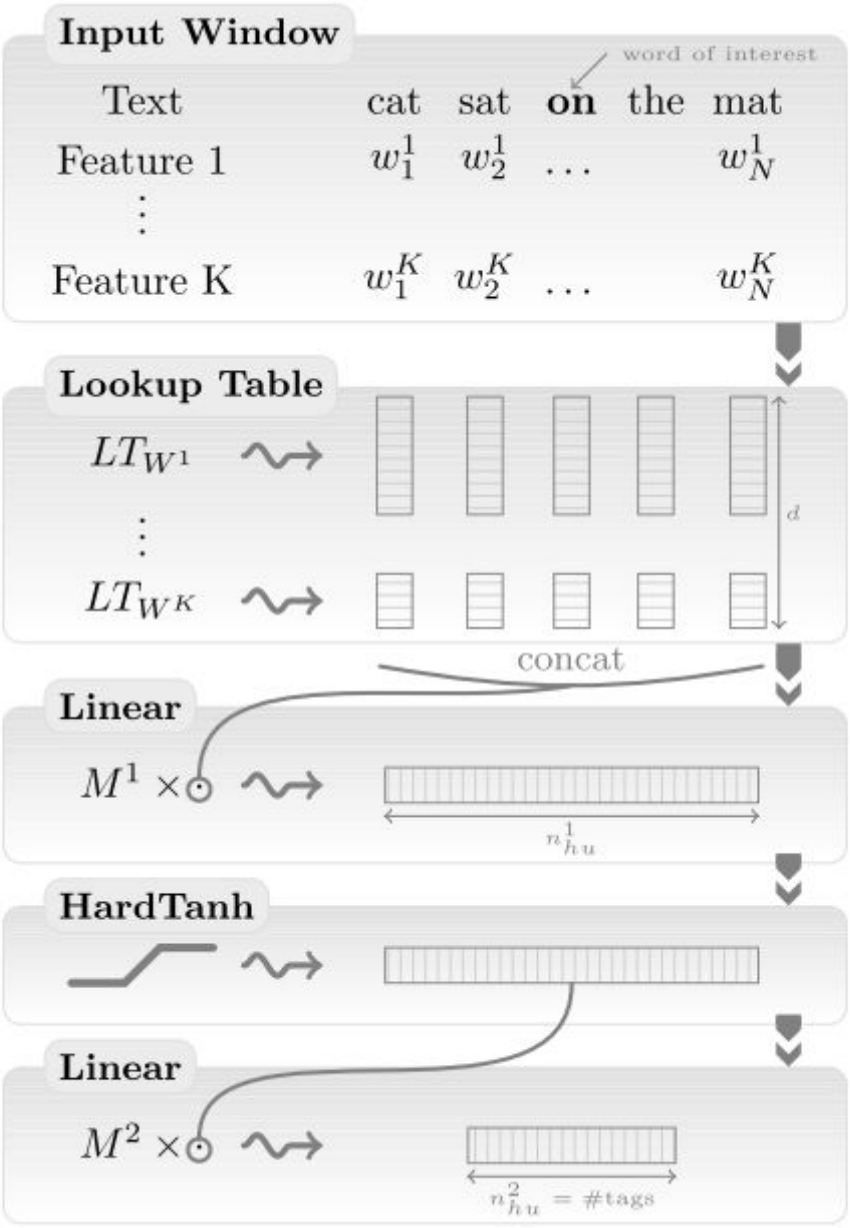

Рис. 2.11 Модель C&W без целевой функции ранжирования (Колберт и соавторы, 2011)

Миколов и соавторы разработали две архитектуры для обучения векторного представления слов, которые по сравнению с предыдущими моделями,

являются вычислительно менее дорогостоящими. Два ключевых преимущества, которые эти архитектуры имеют над моделями Бенджио и Колберта и Уэстона это отказ от вычислительно дорогостоящего скрытого слоя и возможность использования моделью дополнительного контекста. В отличие от лингвистической модели, которая основывает свои предсказания на прошлых словах, и которая оценивается на основе ее способности предсказать каждое следующее слово в корпусе, модель, которая предназначена только для создания точных векторных представлений, не имеет такого ограничения.

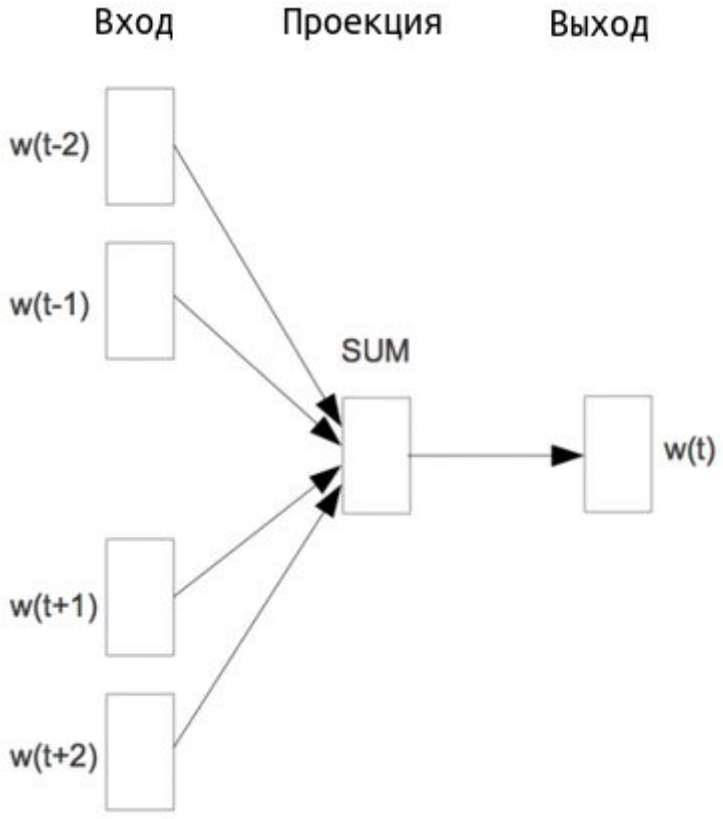

Рис. 2.12 Непрерывный мешок слов CBOW (Миколов и соавторы, 2013)

Как результат, Миколов использует n слов как до так и после центрального слова w_t для его предсказания. Данная модель называется непрерывный

мешок слов (англ. Continuous bag-of-words, CBOW) из-за того, что она использует непрерывные представления слов, порядок которых не имеет значения. Целевая функция CBOW лишь незначительно отличается от целевой функции лингвистической модели:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(w_t \mid w_{t-n}, \cdots, w_{t-1}, w_{t+1}, \cdots, w_{t+n})$$

Вместо добавления n предыдущих слов функция получает окно n слов вокруг целевого слова w_t на каждом временном шаге t.

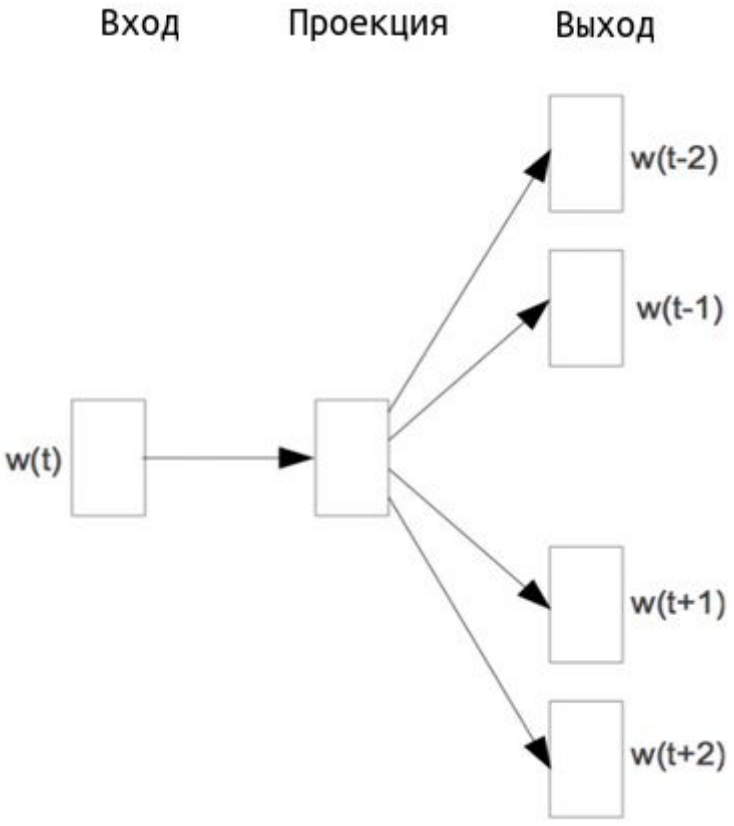

Рис. 2.13 Скип-грамма (Миколов и соавторы, 2013)

В отличии от CBOW, скип-грамма (англ. skip-gram) [33], не использует окружающие слова для прогнозирования центрального слова, а использует

центральное слово для прогнозирования окружающих слов (см. рис. 2.13). Целевая функция скип-граммы, таким образом, представляет собой сумму логарифмических вероятностей окружающих n слов влево и вправо от центрального слова w_t:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \sum_{-n \leq j \leq n, \neq 0} \log p(w_{t+j} \mid w_t)$$

Не смотря на то, что дистрибутивные семантические модели и модели векторных представлений слов используют одну и ту же статистику — количество совпадений между словами, они применяют различные алгоритмы для получения векторных форм слов. Исследования показали что word2vec и GloVe гораздо более эффективно решают ряд задач в области обработки естественного языка, чем традиционные дистрибутивные модели, что в результате принесло им более широкую популярность.

Глава 3. ПРОЕКТНАЯ ЧАСТЬ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

3.1 Разработка требований к библиотеке с учетом поставленных задач

В предыдущих главах сформированы основные цели, которые необходимы для реализации виртуального собеседника. В данной главе необходимо сформировать требования к программному обеспечению и структуру библиотеки, которые позволят реализовать интерфейс для работы с виртуальным собеседником.

Исходя из поставленных задач, были сформулированы следующие требования к библиотеке:

1. Разработка средств парсинга текстовых документов.
2. Подготовка обучающей выборки в формате, использующимся архитектурой сетей долгой-краткосрочной памяти.
3. Разработка архитектуры нейронной сети долгой краткосрочной памяти.
4. Разработка средств, позволяющих сохранять промежуточные и конечные результаты обучения нейронной сети.
5. Разработка средств, позволяющих использовать полученные ранее результаты в работе виртуального собеседника.

Для решения данных задач необходимо:

1. Использовать средства Интернет для получения набора текстовых документов в объеме, достаточным для обучения нейронной сети.
2. Провести парсинг полученных документов.
3. Создать обучающую выборку.

4. Преобразовать обучающую выборку в числовую матрицу последовательностей слов в формат необходимый нейронной сети долгой краткосрочной памяти.
5. Разработать архитектуру нейронной сети долгой краткосрочной памяти.
6. Обучить модель с помощью обучающей выборки.
7. Сохранить модель и веса для ее последующего использования в диалогах с пользователями.

3.2 Обоснование выбора программных и инструментальных средств

Для разработки данной библиотеки выбран язык программирования Python, так как он имеет значительные преимущества по сравнению с другими языками, а его особенности делают его пригодным практически для любой задачи программирования. Python это высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен, а стандартная библиотека включает большой объем различных функций. Python поддерживает несколько парадигм программирования, в том числе структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений и удобные высокоуровневые структуры данных. Код в Python организовывается в функции и классы, которые могут объединяться в модули и пакеты. Эталонной реализацией Python является интерпретатор CPython, поддерживающий большинство активно используемых платформ. Он

распространяется под свободной лицензией Python Software Foundation License, позволяющей использовать его без ограничений в любых приложениях, включая проприетарные. Существуют реализации интерпретаторов для JVM (с возможностью компиляции), MSIL (с возможностью компиляции), LLVM и других. Проект PyPy предлагает реализацию Python с использованием JIT-компиляции, которая значительно увеличивает скорость выполнения Python-программ. Python портирован и работает почти на всех известных платформах — от КПК до мейнфреймов. Существуют порты под Microsoft Windows, практически все варианты UNIX (включая FreeBSD и Linux), Mac OS и Mac OS X, iPhone OS 2.0 и выше, Windows Mobile, Symbian и Android.

На языке Python разработан ряд библиотек для обработки и анализа данных, а также библиотеки, предоставляющие средства работы в области машинного обучения, такие как NumPy, pandas, scikit-learn и Keras.

Библиотека NumPy поддерживает работу с большими многомерными массивами и матрицами, вместе с большой библиотекой высокоуровневых математических функций для операций с этими массивами. Математические алгоритмы, реализованные на интерпретируемых языках, часто работают гораздо медленнее тех же алгоритмов, реализованных на компилируемых языках (например, Фортран, Си, Java). Библиотека NumPy предоставляет реализации вычислительных алгоритмов (в виде функций и операторов), оптимизированные для работы с многомерными массивами. В результате любой алгоритм, который может быть выражен в виде последовательности операций над массивами (матрицами) и реализованный с использованием NumPy, работает так же быстро, как эквивалентный код, выполняемый в MATLAB.

Библиотека pandas это программная библиотека на языке Python для работы с данными. Эта работа строится поверх библиотеки NumPy, являющейся инструментом более низкого уровня. pandas предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временными рядами. Название библиотеки происходит от эконометрического термина «панельные данные», используемого для описания многомерных структурированных наборов информации. pandas распространяется под новой лицензией BSD. Основные возможности библиотеки pandas включают в себя объекты DataFrame для манипулирования индексированными массивами двумерных данных, инструменты для обмена данными между структурами в памяти и файлами различных форматов, встроенные средства совмещения данных и способы обработки отсутствующей информации, переформатирование наборов данных, в том числе создание сводных таблиц, срез данных по значениям индекса, расширенные возможности индексирования, выборка из больших наборов данных, вставка и удаление столбцов данных, возможности группировки которые позволяют выполнять трехэтапные операции типа «разделение, изменение, объединение» (англ. split-apply-combine), слияние и объединение наборов данных, иерархическое индексирование, которое позволяет работать с данными высокой размерности в структурах меньшей размерности, работа с временными рядами — формирование временных периодов и изменение интервалов и т. д. Библиотека оптимизирована для высокой производительности, наиболее важные части кода написаны на Cython и C.

Scikit-learn это библиотека для языка программирования Python, которая включает в себя различные алгоритмы классификации, регрессии и кластеризации данных, такие как метод опорных векторов, случайный лес,

градиентный бустинг, k-средних и DBSCAN. Библиотека предназначена для взаимодействия с численными и научными библиотеками NumPy и SciPy.

Keras — открытая нейросетевая библиотека, написанная на языке Python. Она представляет собой надстройку над фреймворками TensorFlow и Theano. Нацелена на оперативную работу с сетями глубокого обучения, при этом спроектирована так, чтобы быть компактной, модульной и расширяемой. Библиотека была создана как часть исследовательских усилий проекта ONEIROS (англ. Open-ended Neuro-Electronic Intelligent Robot Operating System), а ее основным автором и поддерживающим является Франсуа Шолле (фр. François Chollet), инженер Google. Планировалось что Google будет поддерживать Keras для их основной библиотеке TensorFlow, однако Шолле выделил Keras в отдельную надстройку, так как согласно концепции Keras является скорее интерфейсом, чем сквозной системой машинного обучения. Keras предоставляет высокоуровневый, более интуитивный набор абстракций, который делает простым формирования нейронных сетей независимо от используемой на нижнем уровне библиотеки научных вычислений. Keras содержит многочисленные реализации широко применяемых строительных блоков нейронных сетей, таких как слои, целевые и передаточные функции, оптимизаторы, и множество инструментов для упрощения работы с изображениями и текстом. Ее код размещен на GitHub, а форумы поддержки включают страницу вопросов GitHub, канал Gitter и канал Slack. На сентябрь 2016 года Keras является второй по скорости роста системой глубокого обучения после TensorFlow Google, и третьей по размеру после TensorFlow и Caffe.

TensorFlow — открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и

классификации образов, достигая качества человеческого восприятия. Применяется как для исследований, так и для разработки собственных продуктов Google. Основное API для работы с библиотекой реализовано для Python, также существуют реализации для C++, Haskell, Java и Go. Является продолжением закрытого проекта DistBelief. Изначально TensorFlow была разработана командой Google Brain для внутреннего использования в Google, в 2015 году система была переведена в свободный доступ с открытой лицензией Apache 2.0. TensorFlow является системой машинного обучения Google Brain второго поколения. В то время как эталонная реализация работает на единичных устройствах, TensorFlow может работать на многих параллельных процессорах, как CPU, так и GPU, опираясь на архитектуру CUDA для поддержки вычислений общего назначения на графических процессорах). TensorFlow доступна для 64-разрядных Linux, macOS, Windows, и для мобильных вычислительных платформ, включая Android и iOS. Вычисления TensorFlow выражаются в виде потоков данных через граф состояний. Название TensorFlow происходит от операций с многомерными массивами данных, которые также называются «тензорами». В мае 2016 года Google сообщила о применении для задач глубокого обучения аппаратного ускорителя собственной разработки — тензорного процессора (TPU) — специализированной интегральной схемы, адаптированной под задачи для TensorFlow, и обеспечивающей высокую производительность в арифметике пониженной точности (например, для 8-битной архитектуры) и направленной скорее на применение моделей, чем на их обучение. Сообщалось, что после использования TPU в собственных задачах Google по обработке данных удалось добиться на порядок лучших показателей продуктивности на ватт затраченной энергии.

Глава 4. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ

4.1 Программная реализация библиотеки

В предыдущей главе были рассмотрены основные средства, использующиеся для реализации данной библиотеки. Теперь необходимо подробно описать каждую функцию и каждый класс, использующийся для реализации виртуального собеседника. Для реализации модуля парсинга текста создается ряд функций, в результате применения которых генерируется ряд текстовых файлов, которые в последующем объединяются. Пример описания функций на языке Python в среде разработки приведен в листинге 4.1:

```
def is_time_stamp(l):
    if l[:2].isnumeric() and l[2] == ':':
        return True
    return False

def has_letters(line):
    if re.search('[а-яА-Я]', line):
        return True
    return False

def has_no_text(line):
    l = line.strip()
    if not len(l):
        return True
    if l.isnumeric():
        return True
    if is_time_stamp(l):
```

```
        return True
    if l[0] == '(' and l[-1] == ')':
        return True
    if not has_letters(line):
        return True
    return False

def is_lowercase_letter_or_comma(letter):
    if letter.isalpha() and letter.lower() == letter:
        return True
    if letter == ',':
        return True
    return False

def clean_up(lines):
    new_lines = []
    for line in lines[1:]:
        if has_no_text(line):
            continue
        elif len(new_lines) and is_lowercase_letter_or_comma(line[0]):
            new_lines[-1] = new_lines[-1].strip() + ' ' + line
        else:
            new_lines.append(line)
    return new_lines

def main(args):
    file_name = args[1]
    file_encoding = 'utf-8' if len(args) < 3 else args[2]
    with open(file_name, encoding=file_encoding, errors='replace') as f:
        lines = f.readlines()
        new_lines = clean_up(lines)
    new_file_name = file_name[:-3] + 'txt'
```

```
with open(new_file_name, 'w') as f:
    for line in new_lines:
        f.write(line)
```

Листинг 4.1 Описания функций парсинга текста на языке Python

Полученный текст необходимо преобразовать в формат, использующимся архитектурой сетей долгой-краткосрочной памяти. Для этого нужно подготовить два отдельных текстовых файла, один в котором будет содержатся контекст, и второй, в котором будут содержаться ответы в соответствии с контекстным файлом. Также необходимо модифицировать некоторые сокращения, приведя их к полной форме, для получения однородных данных.

```
text = open('dialog_simple', 'r')
q = open('context', 'w')
a = open('answers', 'w')
pre_pre_previous_raw = ''
pre_previous_raw = ''
previous_raw = ''
person = ' '
previous_person = ' '

l1 = ['won’t', 'won\'t', 'wouldn’t', 'wouldn\'t', '’m', '’re', '’ve', '’ll', '’s','’d', 'n’t', '\'m', '\'re', '\'ve',
'\'ll', '\'s', '\'d', 'can\'t', 'n\'t', 'B: ', 'A: ', ',', ';', '.', '?', '!', ':', '. ?', ',  .', '. ,', 'EOS', 'BOS', 'eos', 'bos']
l2 = ['will not', 'will not', 'would not', 'would not', ' am', ' are', ' have', ' will', ' is', ' had', ' not', ' am',
' are', ' have', ' will', ' is', ' had', 'can not', ' not', '', '', ' ,', ' ;', ' .', ' ?', ' !', ' :', '? ', '.', ',', '', '', '', '']
l3 = ['-', '_', ' *', ' /', '* ', '/ ', '\'"', ' \\"', '\\ ', '--', '...', '. . .']

for i, raw_word in enumerate(text):
```

```
    pos = raw_word.find('+++$+++')

    if pos > -1:
        person = raw_word[pos+7:pos+10]
        raw_word = raw_word[pos+8:]
    while pos > -1:
        pos = raw_word.find('+++$+++')
        raw_word = raw_word[pos+2:]

    raw_word = raw_word.replace('$+++', '')
    previous_person = person

    for j, term in enumerate(l1):
        raw_word = raw_word.replace(term, l2[j])

    for term in l3:
        raw_word = raw_word.replace(term, ' ')

    raw_word = raw_word.lower()

    if i>0:
        q.write(pre_previous_raw[:-1] + ' ' + previous_raw[:-1] + '\n')
        a.write(raw_word[:-1] + '\n')

    pre_pre_previous_raw = pre_previous_raw
    pre_previous_raw = previous_raw
    previous_raw = raw_word

q.close()
a.close()
```

Листинг 4.2 Описания функций создания контекстных данных на языке Python

Следующим шагом необходимо преобразовать полученные данные в форму, необходимую для обучения рекуррентной сетью долгой-краткосрочной памяти. Необходимо определить такие переменные как размер используемого словаря, максимальная длина входной и выходной последовательности данных. Также необходимо ввести специальные токены для обозначения начала и конца предложения, BOS (begin of sentence) и EOS (end of sentence). Пример преобразования данных приведен на листинге 4.3:

```
questions_file = 'context'
answers_file = 'answers'
vocabulary_file = 'vocabulary_movie'
padded_questions_file = 'Padded_context'
padded_answers_file = 'Padded_answers'
unknown_token = 'something'

vocabulary_size = 7000
max_features = vocabulary_size
maxlen_input = 50
# Максимальная длина слов в предложении
maxlen_output = 50

q = open(questions_file, 'r')
questions = q.read()

a = open(answers_file, 'r')
answers = a.read()
all = answers + questions
```

```
# Токенизация ответов
paragraphs_a = [p for p in answers.split('\n')]
paragraphs_b = [p for p in all.split('\n')]
paragraphs_a = ['BOS '+p+' EOS' for p in paragraphs_a]
paragraphs_b = ['BOS '+p+' EOS' for p in paragraphs_b]
paragraphs_b = ' '.join(paragraphs_b)
tokenized_text = paragraphs_b.split()
paragraphs_q = [p for p in questions.split('\n') ]
tokenized_answers = [p.split() for p in paragraphs_a]
tokenized_questions = [p.split() for p in paragraphs_q]

# Подсчет частоты слов
word_freq = nltk.FreqDist(itertools.chain(tokenized_text))

# Получение наиболее частых слов и построение index_to_word и word_to_index векторов
vocab = word_freq.most_common(vocabulary_size-1)

# Сохранение словаря
with open(vocabulary_file, 'wb') as v:
    pickle.dump(vocab, v)

vocabulary_file = 'vocabulary_movie'
vocab = pickle.load(open(vocabulary_file, 'rb'))

index_to_word = [x[0] for x in vocab]
index_to_word.append(unknown_token)
word_to_index = dict([(w,i) for i,w in enumerate(index_to_word)])

# Замена всех слов, отсутствующих в словаре на незнакомый токен
for i, sent in enumerate(tokenized_answers):
    tokenized_answers[i] = [w if w in word_to_index else unknown_token for w in sent]
```

```
for i, sent in enumerate(tokenized_questions):
    tokenized_questions[i] = [w if w in word_to_index else unknown_token for w in sent]

# Создание обучающей выборки
X = np.asarray([[word_to_index[w] for w in sent] for sent in tokenized_questions])
Y = np.asarray([[word_to_index[w] for w in sent] for sent in tokenized_answers])

Q = sequence.pad_sequences(X, maxlen=maxlen_input)
A = sequence.pad_sequences(Y, maxlen=maxlen_output, padding='post')

with open(padded_questions_file, 'wb') as q:
    pickle.dump(Q, q)

with open(padded_answers_file, 'wb') as a:
    pickle.dump(A, a)
```

Листинг 4.3 Подготовка данных

Реализуемая архитектура рекуррентной нейронной сети долгой-краткосрочной памяти использует язык Python и библиотеку Keras в качестве front-end, а также библиотеку TensorFlow в качестве back-end. Функцией потерь является категорийная перекрестная энтропия, оптимизация происходит за счет алгоритма Адам, который представляет собой алгоритм на основе градиента стохастических целевых функций первого порядка, основанный на адаптивных оценках моментов младшего порядка. Данный алгоритм прост в реализации, является эффективным с точки зрения вычислительной мощности, имеет небольшие требования к памяти, инвариантен к диагональному масштабированию градиентов и

хорошо подходит для задач, которые являются большими с точки зрения данных и / или параметров. Пример реализации приведен на листинге 4.4:

```
input_context = Input(shape=(maxlen_input,), dtype='int32', name='the context text')
input_answer = Input(shape=(maxlen_input,), dtype='int32', name='the answer text up to the current token')
LSTM_encoder = LSTM(sentence_embedding_size, kernel_initializer= 'lecun_uniform', name='Encode context')
LSTM_decoder = LSTM(sentence_embedding_size, kernel_initializer= 'lecun_uniform', name='Encode answer up to the current token')

if os.path.isfile(weights_file):
    Shared_Embedding = Embedding(output_dim=word_embedding_size, input_dim=dictionary_size, input_length=maxlen_input, name='Shared')
else:
    Shared_Embedding = Embedding(output_dim=word_embedding_size, input_dim=dictionary_size, weights=[embedding_matrix], input_length=maxlen_input, name='Shared')
word_embedding_context = Shared_Embedding(input_context)
context_embedding = LSTM_encoder(word_embedding_context)

word_embedding_answer = Shared_Embedding(input_answer)
answer_embedding = LSTM_decoder(word_embedding_answer)

merge_layer = concatenate([context_embedding, answer_embedding], axis=1, name='concatenate the embeddings of the context and the answer up to current token')
out = Dense(dictionary_size//2, activation="relu", name='relu activation')(merge_layer)
out = Dense(dictionary_size, activation="softmax", name='likelihood of the current token using softmax activation')(out)

model = Model(inputs=[input_context, input_answer], outputs = [out])
```

```
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.00005))
model.summary()
```

Листинг 4.4 Архитектура рекуррентной нейронной сети.

Общее количество обучаемых весов модели составляет 28 272 900. Веса сохраняются в отдельный файл в формате иерархического формата данных hdf5 для последующего использования и до обучения модели (см. рис. 4.1).

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
the context text (InputLayer)  (None, 50)              0
_________________________________________________________________________________________________
the answer text up to the curren (None, 50)            0
_________________________________________________________________________________________________
Shared (Embedding)           (None, 50, 100)           700000
_________________________________________________________________________________________________
Encode context (LSTM)        (None, 300)               481200
_________________________________________________________________________________________________
Encode answer up to the current  (None, 300)           481200
_________________________________________________________________________________________________
concatenate the embeddings of th (None, 600)           0
_________________________________________________________________________________________________
relu activation (Dense)      (None, 3500)              2103500
_________________________________________________________________________________________________
likelihood of the current token  (None, 7000)          24507000
=================================================================================================
Total params: 28,272,900.0
Trainable params: 28,272,900.0
Non-trainable params: 0.0
```

Рис. 4.1 Схема рекуррентной нейронной сети с указанием количества параметров для каждого слоя.

Каноническая seq2seq модель рекуррентной сети стала широко востребованной в задачах автоматического перевода, где распределения слов на входе и на выходе различаются, так как они принадлежат к разным языкам. Реализуемая архитектура предполагает одинаковое распределение для входа и выхода. В результате модель содержит, используемый совместно процессами кодирования и декодирования, слой с векторными представлениями слов, полученными с помощью библиотеки GloVe (Global Vectors for Word Representation) . Пример обработки вопроса и формирование

ответа с помощью модели рекуррентной нейронной сети приведен на листинге 4.6:

```
print("CHAT: \n")

# Обработка вопроса юзера
prob = 0
que = ''
last_query  = ' '
last_last_query = ''
text = ' '
last_text = ''
print("computer: hi ! please type your name.\n")
name = input('user: ')
print("computer: hi , " + name +" ! My name is " + name_of_computer + ".\n")

while que != 'exit .':
    que = input('user: ')
    que = preprocess(que, name_of_computer)
    # Сохранение данных
    q = last_query + ' ' + text
    a = que
    qf.write(q + '\n')
    af.write(a + '\n')
    # Формирование ответа
    if prob > 0.2:
        query = text + ' ' + que
    else:
        query = que

    last_text = text
```

```
    Q = tokenize(query)

    # Использование обученной модели для предсказания ответа
    predout, prob = greedy_decoder(Q[0:1])
    start_index = predout.find('EOS')
    text = preprocess(predout[0:start_index], name)
    print ('computer: ' + text + '     (with probability of %f)' % prob)

    last_last_query = last_query
    last_query = que

qf.close()
af.close()
```

Листинг 4.6 Обработка вопроса и диалог.

4.2 Тестовые примеры

Диалог с разговорным агентом ведется на английском языке. Для большего понимания контекста все диалоги дополнительно переведены на русский язык. Ниже представлены примеры диалогов.

```
user: how do you do?
computer:  i want to apply a loan.
user: why do not you go to the bank?
computer:  i have to make a house.
user: what kind of house?
computer:  no, i do not. i have never been on the month.
user: do you have a car?
computer:  yes, i do. i have a good friend.
user: tell me about your friend?
computer:  of course. i would have to tell you know he had a long.
user: what else could you tell me about your friend?
computer:  of course.
user: where do you live?
computer:  i live in san francisco.
user: tell me about san francisco. do you like it there?
computer:  i do not know, either to the.
user: ok. see you soon.
computer:  i hate the car first time.

user: как дела?
компьютер: я хочу применить кредит.
пользователь: почему бы вам не пойти в банк?
компьютер: я должен сделать дом.
пользователь: какой дом?
компьютер: нет, я этого не делаю. я никогда не был в этом месяце.
пользователь: у вас есть машина?
компьютер: да, я знаю. У меня есть хороший друг.
user: расскажите мне о своем друге?
компьютер: конечно. я должен был сказать вам, что он долгое время.
user: что еще вы могли бы рассказать мне о своем другом?
компьютер: конечно.
пользователь: где вы живете?
компьютер: я живу в Сан-Франциско.
пользователь: расскажи мне о Сан-Франциско. тебе там нравится ?
компьютер: я не знаю, либо.
пользователь: ок. пока.
компьютер: мне не понравилась машина в первый раз.
```

Рисунок 4.2 Диалог с разговорным агентом.

```
user: hello.
computer:  you know why i did not love it.
user: why you did not you love it?
computer:  just yesterday, i was on a picnic in the.
user: tell me about yesterday.
computer:  how did you get it?
user: tell me about the picnic.
computer:  what do you like about it?
user: what do you like about it?
computer:  well, i am kind of short and, so it is hard to find a piece of clothing that fits me perfectly, but it
is hard to find.
user: what is your favorite color?
computer:  my favorite is a. i have to work with computers.
user: what do you do except working with computers?
computer:  i usually shop for clothes.
user: ok. it was nice talking to you. bye.
computer:  no problem.

user: привет.
компьютер: вы знаете, почему я не любил его.
user: почему вы этого не любили?
компьютер: только вчера я был на пикнике в.
пользователь: скажите мне вчера.
компьютер: как вы его получили?
user: расскажите мне о пикнике.
компьютер: что вам нравится?
user: что вам нравится?
компьютер: ну, я вроде как короткий, и поэтому трудно найти одежду, которая мне подходит, но ее трудно найти.
user: какой твой любимый цвет?
компьютер: мой любимый. я должен работать с компьютерами.
user: что вы делаете, кроме работы с компьютерами?
компьютер: я обычно покупаю одежду.
пользователь: ок. приятно было с тобой поговорить. пока.
компьютер: никаких проблем.
```

Рисунок 4.3 Диалог с разговорным агентом.

```
user: good morning.
computer:  good morning. please sit down, i am adam, the car just hit another car!
user: i live in moscow.
computer:  what are you going to do with them?
user: they are my friends. we travel together.
computer:  i am looking for my job.
user: what job?
computer:  yes, so do.
user: do you have a family?
computer:  not. i have noticed a lot of my.
user: i have been to england.
computer:  good. please, fill out this form.
user: i want a loan.
computer:  what are you going to do?
user: i want to rent a car.
computer:  what kind of car do you have?
user: a red one.
computer:  is it a good job?
user: i work with computers.
computer:  i am glad sunday is old.
user: weekends are good.
computer:  thank you.
user: bye.
computer:  bye. bye.
```

```
пользователь: доброе утро.
компьютер: доброе утро. пожалуйста, присаживайся, я адам, машина просто врезалась в другую машину!
пользователь: я живу в москве.
компьютер: что вы собираетесь с ними делать?
пользователь: они мои друзья. мы путешествуем вместе.
компьютер: я ищу свою работу.
пользователь: какая работа?
компьютер: да, так и есть.
пользователь: у вас есть семья?
компьютер: нет. я заметил много своего.
пользователь: я был в англии.
компьютер: хорошо. пожалуйста, заполните эту форму.
пользователь: я хочу получить кредит.
компьютер: что вы собираетесь делать?
пользователь: я хочу взять напрокат автомобиль.
компьютер: какой автомобиль у вас есть?
пользователь: красный.
компьютер: это хорошая работа?
user: я работаю с компьютерами.
компьютер: я рад, что воскресенье давно.
пользователь: выходные хорошие.
компьютер: спасибо.
пользователь: пока.
компьютер: пока. пока.
```

Рисунок 4.4 Диалог с разговорным агентом.

```
user: how is it going?
computer:  that means it is too much.
user: do you think so?
computer:  yes, my friend.
user: do you have many friends?
computer:  yes, i do. i have two brothers and one sister.
user: tell me about your family.
computer:  i am sorry for you.
user: what about your parents?
computer:  i am a.
user: tell me about your friends.
computer:  you are wearing out of your right hand.
user: what kind of clothes do you prefer?
computer:  i am a puppy.
user: do you like cats?
computer:  i am a big fashion fan.
user: do you like animals?
computer:  yes, i am a puppy.
user: i live in a house.
computer:  that is because they are funny.
user: are they?
computer:  they are in the dining room.
user: do you like bananas?
computer:  yes, i am looking for a sweater.
user: where do you live?
computer:  i live in san francisco.
user: ok. see you later.
computer:  i usually eat in a while.
```

```
user: как дела?
компьютер: это означает, что это слишком много.
пользователь: вы так думаете?
компьютер: да, мой друг.
пользователь: у вас много друзей?
компьютер: да. у меня есть два брата и одна сестра.
user: расскажите мне о своей семье.
компьютер: я сожалею.
пользователь: как насчет ваших родителей?
компьютер: я.
user: расскажите о своих друзьях.
компьютер: вы носите свою правую руку.
пользователь: какую одежду вы предпочитаете?
компьютер: я щенок.
user: вам нравятся кошки?
компьютер: я большой поклонник моды.
user: вам нравятся животные?
компьютер: да, я щенок.
пользователь: я живу в доме.
компьютер: это потому, что они забавные.
пользователь: они?
компьютер: они сейчас в столовой.
user: вам нравятся бананы?
компьютер: да, я ищу свитер.
пользователь: где вы живете?
компьютер: я живу в сан-франциско.
пользователь: ok. увидимся.
компьютер: я обычно ем в это время.
```

Рисунок 4.5 Диалог с разговорным агентом.

```
user: nice to meet you.
computer:  i am glad to see you.
user: me too. what are you going to do this evening?
computer:  i do not know any time to get.
user: are you busy? for how long are you staying in town?
computer:  i am going to get to the city.
user: what business do you have there? is it about your house?
computer:  yes, there are .
user: are you going to visit your sister?
computer:  i am sleepy.
user: did you drive here?
computer:  i went to work.
user: have you been on vacation recently?
computer:  i have been in a few minutes.
user: not much, i should say.
computer:  why not?
user: i would have stayed longer if i were you.
computer:  why not?
user: let us talk about something else.
computer:  do you know what i do not like?
user: what?
computer:  well, if i have a big test at school.
user: makes sense.
computer:  i hate.
user: i see.
computer:  you should not lie in the sun for too long.
user: i agree.
computer:  what are you talking about?
user: i have to go. see you soon. bye.
computer:  bye. bye.
```

```
пользователь: приятно познакомиться.
компьютер: я рад вас видеть.
пользователь: я тоже. что вы собираетесь делать этим вечером?
компьютер: я не знаю, сколько времени можно получить.
пользователь: вы заняты? как долго вы будете находиться в городе?
компьютер: я собираюсь добраться до города.
пользователь: какие у вас дела? что-то о вашем доме?
компьютер: да.
пользователь: вы собираетесь посетить свою сестру?
компьютер: я хочу спать.
пользователь: вы сюда приехали на машине?
компьютер: я ходил на работу.
пользователь: вы недавно были в отпуске?
компьютер: я был несколько минут.
пользователь: немного, я должен сказать.
компьютер: почему бы и нет?
пользователь: я бы остался дольше, если бы был вами.
компьютер: почему бы и нет?
пользователь: давайте поговорим о чем-то другом.
компьютер: знаете ли вы, что мне не нравится?
пользователь: что?
компьютер: например, если у меня большой тест в школе.
пользователь: имеет смысл.
компьютер: я ненавижу.
пользователь: я понимаю.
компьютер: вы не должны долго лежать на солнце.
пользователь: я согласен.
компьютер: о чем вы говорите?
пользователь: я должен идти. скоро увидимся. пока.
компьютер: пока. пока.
```

Рисунок 4.6 Диалог с разговорным агентом.

ЗАКЛЮЧЕНИЕ

Таким образом, в данной работе были решены все поставленные задачи: был сделан обзор существующих методов и средств разработки и реализации виртуального собеседника, а также методов, средств разработки и реализации виртуального собеседника. Были изучены средства для работы с рекуррентными нейронными сетями долгой краткосрочной памяти и векторными представлениями слов, были решены задачи по генерации обучающей выборки и разработаны требования к библиотеке. Результатом данной работы стала программная реализация виртуального собеседника.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Вычислительные машины и разум [Текст] / Тьюринг А. - Mind, 1950. - 433–460 с.
2. Чего не могут вычислительные машины: критика искусств [Текст] / Дрейфус Х. - УРСС, 2009. - 333 с.
3. Искусственный интеллект. Современный подход [Текст] / Рассел С., Норвиг П. - Вильямс, 2006. - 1408 с.
4. Китайская комната [Текст] / Коул Д. - Стэнфордская философская энциклопедия, 2015.
5. Размышления над статьёй Сёрла Minds, Brains, and Programs [Текст] / Эскина М., 2003.
6. Joint language and translation modeling with recurrent neural networks [Текст] / Auli M., Galley M., Quirk C. и др. - EMNLP, 2013. - 1044–1054 с.
7. Working memory and executive control. Philosophical Transactions of the Royal Society [Текст] / Baddeley A., Della Sala S., Robbins T. - Biological Sciences, 1996. - 1397–1404 с.
8. The principled design of large-scale recursive neural network architectures DAG-RNNs and the protein structure prediction problem [Текст] / Baldi P., Pollastri G. - The Journal of Machine Learning Research, 2003. - 602 с.
9. Evolving memory cell structures for sequence learning [Текст] / Bayer J., Wierstra D., Togelius J. и др. – ICANN 2009, 755–764 с.
10. Evolving networks: Using the genetic algorithm with connectionist learning [Текст] / Belew R., McInerney J., Schraudolph N. - Citeseer, 1990.
11. Learning long-term dependencies with gradient descent is difficult [Текст] / Bengio Y., Simard P., Frasconi P. - Neural Networks, 1994. - 157–166 с.

12. A Neural Probabilistic Language Model [Текст] / Bengio Y., Ducharme R., Vincent P. и др. - The Journal of Machine Learning Research, 2003. - 1137–1155 с.
13. Efficient Estimation of Word Representations in Vector Space [Текст] / Mikolov T., Corrado G., Chen K. и др. - Proceedings of the International Conference on Learning Representations (ICLR 2013), 2013. - 1–12 с.
14. Distributed Representations of Words and Phrases and their Compositionality [Текст] / Mikolov T., Chen K., Corrado G., Dean, J. - NIPS, 2013. - 1–9 с.
15. A unified architecture for natural language processing [Текст] / Collobert R., Weston J. - Proceedings of the 25th International Conference on Machine Learning 2008. - 160–167 с.
16. Glove: Global Vectors for Word Representation [Текст] / Pennington J., Socher R., Manning C. - Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, 2014. - 1532–1543 с.
17. Character-Aware Neural Language Models [Электронный ресурс] / Kim Y., Jernite Y., Sontag D. и др. - Режим доступа: http://arxiv.org/abs/1508.06615, 2015.
18. Exploring the Limits of Language Modeling [Электронный ресурс] / Jozefowicz R., Vinyals O., Schuster M. и др. - Режим доступа: http://arxiv.org/abs/1602.02410, 2016.
19. Natural Language Processing (almost) from Scratch [Текст] / Collobert R., Weston J., Bottou L., Karlen M., Kavukcuoglu K. и др. - Journal of Machine Learning Research, 2011. - 2493–2537 с.
20. Strategies for Training Large Vocabulary Neural Language Models [Электронный ресурс] / Chen W., Grangier D., Auli M. - Режим доступа: http://arxiv.org/abs/1512.04906, 2015.

21. Improving Distributional Similarity with Lessons Learned from Word Embeddings [Текст] / Levy O., Goldberg Y., Dagan I. - Transactions of the Association for Computational Linguistics, 2015. - 211–225 с.
22. Don’t count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors [Текст] / Baroni M., Dinu G., Kruszewski G. - ACL, 2014. - 238–247 с.
23. Neural Word Embedding as Implicit Matrix Factorization. Advances in Neural Information Processing Systems [Электронный ресурс] / Levy O., Goldberg Y. - Режим доступа: http://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization, 2014.
24. Inducing Domain-Specific Sentiment Lexicons from Unlabeled Corpora [Электронный ресурс] / Hamilton W. L., Clark K., Leskovec J. - Режим доступа: http://arxiv.org/abs/1606.02820, 2016.
25. Recurrent nets that time and count [Текст] / Felix G., Schmidhuber J. - In Neural Networks, Proceedings of the IEEE-INNS-ENNS International Joint Conference on, volume 3, 2000. - 189–194 с.
26. Learning to forget: Continual prediction with LSTM [Текст] / Felix G., Schmidhuber J., Cummins F. - Neural computation, 12(10), 2000. - 2451–2471 с.
27. word2vec explained: deriving Mikolov et al.’s negative-sampling word-embedding method [Текст] / Goldberg Y., Levy O. - arXiv:1402.3722, 2014.
28. Supervised sequence labelling with recurrent neural networks [Текст] / Graves A. - Springer, том 385, 2012.

29. Framewise phoneme classification with bidirectional LSTM and other neural network architectures [Текст] / Graves A., Schmidhuber J. - Neural Networks, 18(5), 2005. - 602–610 с.
30. Long short-term memory [Текст] / Hochreiter S., Schmidhuber J. - Neural Computation, 9(8), 1997. - 1735–1780 с.
31. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies [Текст] / Hochreiter S., Bengio Y., Frasconi P. и др. - A field guide to dynamical recurrent neural networks, 2001.
32. The unreasonable effectiveness of recurrent neural networks [Электронный ресурс] / Andrej Karpathy - Режим доступа: http://karpathy.github.io/2015/05/21/rnn-effectiveness/, 2015.
33. Efficient estimation of word representations in vector space [Текст] / Mikolov T., Chen K., Corrado G. и др. - arXiv:1301.3781, 2013.
34. BLEU: a method for automatic evaluation of machine translation [Текст] / Papineni K., Roukos S., Ward T., Zhu W. - In Proceedings of the 40th annual meeting on association for computational linguistics, 2002. - 311–318 с.
35. Glove: Global vectors for word representation [Текст] / Pennington J., Socher R., Manning C. - Proceedings of the Empirical Methods in Natural Language Processing, 2014. - 12 с.
36. Learning continuous phrase representations and syntactic parsing with recursive neural networks [Текст] / Socher R., Manning C., Ng A. - In Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop, 2010. - 1–9 с.
37. Generating text with recurrent neural networks [Текст] / Sutskever I, Martens J., Hinton G. - In Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011. - 1017–1024 с.

38. Sequence to sequence learning with neural networks [Текст] / Sutskever I, Vinyals O., Le Q. - In Advances in Neural Information Processing Systems, 2014. - 3104–3112 с.
39. Backpropagation through time: what it does and how to do it [Текст] / Werbos P. - Proceedings of the IEEE, 78(10), 1990. - 1550–1560 с.
40. Wikipedia. Backpropagation [Электронный ресурс] / Wikipedia, the free encyclopedia - Режим доступа: http://en.wikipedia.org/wiki/Backpropagation, 2015.
41. A learning algorithm for continually running fully recurrent neural networks [Текст] / Williams R., Zipser D. - Neural Computation, 1(2), 1989. - 270–280 с.
42. On rectified linear units for speech processing [Текст] / Zeiler M., Ranzato M., Monga R. и др. - In Acoustics, Speech and Signal Processing (ICASSP), 2013. - 3517–3521 с.

ПРИЛОЖЕНИЯ

```
import re, sys

def is_time_stamp(l):
    if l[:2].isnumeric() and l[2] == ':':
        return True
    return False

def has_letters(line):
    if re.search('[а-яА-Я]', line):
        return True
    return False

def has_no_text(line):
    l = line.strip()
    if not len(l):
        return True
    if l.isnumeric():
        return True
    if is_time_stamp(l):
        return True
    if l[0] == '(' and l[-1] == ')':
        return True
    if not has_letters(line):
        return True
    return False

def is_lowercase_letter_or_comma(letter):
    if letter.isalpha() and letter.lower() == letter:
        return True
```

```
    if letter == ',':
        return True
    return False

def clean_up(lines):
    new_lines = []
    for line in lines[1:]:
        if has_no_text(line):
            continue
        elif len(new_lines) and is_lowercase_letter_or_comma(line[0]):
            new_lines[-1] = new_lines[-1].strip() + ' ' + line
        else:
            new_lines.append(line)
    return new_lines

def main(args):
    file_name = args[1]
    file_encoding = 'utf-8' if len(args) < 3 else args[2]
    with open(file_name, encoding=file_encoding, errors='replace') as f:
        lines = f.readlines()
        new_lines = clean_up(lines)
    new_file_name = file_name[:-3] + 'txt'
    with open(new_file_name, 'w') as f:
        for line in new_lines:
            f.write(line)

file_name = 'file.srt'
file_encoding = 'utf-8'
with open(file_name, encoding=file_encoding, errors='replace') as f:
    lines = f.readlines()
    new_lines = clean_up(lines)
new_file_name = file_name[:-4] + '.txt'
```

```
with open(new_file_name, 'w') as f:
    for line in new_lines:
        f.write(line)


import numpy as np

text = open('dialog_simple', 'r')
q = open('context', 'w')
a = open('answers', 'w')
pre_pre_previous_raw = ''
pre_previous_raw = ''
previous_raw = ''
person = ' '
previous_person = ' '

l1 = ['won’t', 'won\'t', 'wouldn’t', 'wouldn\'t', '’m', '’re', '’ve', '’ll', '’s','’d', 'n’t', '\'m', '\'re', '\'ve',
'\'ll', '\'s', '\'d', 'can\'t', 'n\'t', 'B: ', 'A: ', ',', ';', '.', '?', '!', ':', '. ?', ',    .', '. ,', 'EOS', 'BOS', 'eos', 'bos']
l2 = ['will not', 'will not', 'would not', 'would not', ' am', ' are', ' have', ' will', ' is', ' had', ' not', ' am',
' are', ' have', ' will', ' is', ' had', 'can not', ' not', '', '', ' ,', ' ;', ' .', ' ?', ' !', ' :', '? ', '.', ',', '', '', '', '']
l3 = ['-', '_', ' *', ' /', '* ', '/ ', '\'', ' \\"', '\\ ', '--', '...', '. . .']

for i, raw_word in enumerate(text):
    pos = raw_word.find('+++$+++')

    if pos > -1:
        person = raw_word[pos+7:pos+10]
        raw_word = raw_word[pos+8:]
    while pos > -1:
        pos = raw_word.find('+++$+++')
        raw_word = raw_word[pos+2:]
```

```
    raw_word = raw_word.replace('$+++', '')
    previous_person = person

    for j, term in enumerate(l1):
        raw_word = raw_word.replace(term, l2[j])

    for term in l3:
        raw_word = raw_word.replace(term, ' ')

    raw_word = raw_word.lower()

    if i>0:
        q.write(pre_previous_raw[:-1] + ' ' + previous_raw[:-1] + '\n')
        a.write(raw_word[:-1] + '\n')

    pre_pre_previous_raw = pre_previous_raw
    pre_previous_raw = previous_raw
    previous_raw = raw_word

q.close()
a.close()


import numpy as np
import pandas as pd
import os
import csv
import nltk
import itertools
import operator
import pickle
import numpy as np
```

```
from keras.preprocessing import sequence
from scipy import sparse, io
from numpy.random import permutation
import re
np.random.seed(0)

questions_file = 'context'
answers_file = 'answers'
vocabulary_file = 'vocabulary_movie'
padded_questions_file = 'Padded_context'
padded_answers_file = 'Padded_answers'
unknown_token = 'something'

vocabulary_size = 7000
max_features = vocabulary_size
maxlen_input = 50
# Максимальная длина слов в предложении
maxlen_output = 50

q = open(questions_file, 'r')
questions = q.read()

a = open(answers_file, 'r')
answers = a.read()
all = answers + questions

# Токенизация ответов
paragraphs_a = [p for p in answers.split('\n')]
paragraphs_b = [p for p in all.split('\n')]
paragraphs_a = ['BOS '+p+' EOS' for p in paragraphs_a]
paragraphs_b = ['BOS '+p+' EOS' for p in paragraphs_b]
paragraphs_b = ' '.join(paragraphs_b)
```

```
tokenized_text = paragraphs_b.split()
paragraphs_q = [p for p in questions.split('\n') ]
tokenized_answers = [p.split() for p in paragraphs_a]
tokenized_questions = [p.split() for p in paragraphs_q]

# Подсчет частоты слов
word_freq = nltk.FreqDist(itertools.chain(tokenized_text))

# Получение наиболее частых слов и построение index_to_word и word_to_index векторов
vocab = word_freq.most_common(vocabulary_size-1)

# Сохранение словаря
with open(vocabulary_file, 'wb') as v:
    pickle.dump(vocab, v)

vocabulary_file = 'vocabulary_movie'
vocab = pickle.load(open(vocabulary_file, 'rb'))

index_to_word = [x[0] for x in vocab]
index_to_word.append(unknown_token)
word_to_index = dict([(w,i) for i,w in enumerate(index_to_word)])

# Замена всех слов, отсутствующих в словаре на незнакомый токен
for i, sent in enumerate(tokenized_answers):
    tokenized_answers[i] = [w if w in word_to_index else unknown_token for w in sent]

for i, sent in enumerate(tokenized_questions):
    tokenized_questions[i] = [w if w in word_to_index else unknown_token for w in sent]

# Создание обучающей выборки
X = np.asarray([[word_to_index[w] for w in sent] for sent in tokenized_questions])
Y = np.asarray([[word_to_index[w] for w in sent] for sent in tokenized_answers])
```

```
Q = sequence.pad_sequences(X, maxlen=maxlen_input)
A = sequence.pad_sequences(Y, maxlen=maxlen_output, padding='post')

with open(padded_questions_file, 'wb') as q:
    pickle.dump(Q, q)

with open(padded_answers_file, 'wb') as a:
    pickle.dump(A, a)



from keras.layers import Input, Embedding, LSTM, Dense, RepeatVector, Dropout, merge
from keras.optimizers import Adam
from keras.models import Model
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras.preprocessing import sequence
from keras.layers import concatenate

import keras.backend as K
import numpy as np
import pickle
import os.path
import sys
import nltk
import re
import time

np.random.seed(0)

word_embedding_size = 100
sentence_embedding_size = 300
```

```
dictionary_size = 7000
maxlen_input = 50

vocabulary_file = 'vocabulary_movie'
weights_file = 'weights.h5'
unknown_token = 'something'
file_saved_context = 'saved_context'
file_saved_answer = 'saved_answer'
name_of_computer = 'adam'

def greedy_decoder(input):
    flag = 0
    prob = 1
    ans_partial = np.zeros((1, maxlen_input))
    ans_partial[0, -1] = 2 # Индекс символа BOS (begin of sentence, начало предложения)
    for k in range(maxlen_input - 1):
        ye = model.predict([input, ans_partial])
        yel = ye[0,:]
        p = np.max(yel)
        mp = np.argmax(ye)
        ans_partial[0, 0:-1] = ans_partial[0, 1:]
        ans_partial[0, -1] = mp
        if mp == 3: # Индекс символа EOS (end of sentence, конец предложения)
            flag = 1
        if flag == 0:
            prob = prob * p

    text = ''
    for k in ans_partial[0]:
        k = k.astype(int)
        if k < (dictionary_size-2):
            w = vocabulary[k]
```

```
            text = text + w[0] + ' '
    return(text, prob)


def preprocess(raw_word, name):
    l1 = ['won’t', 'won\'t', 'wouldn’t', 'wouldn\'t', '’m', '’re', '’ve', '’ll', '’s','’d', 'n’t', '\'m', '\'re', '\'ve',
'\'ll', '\'s', '\'d', 'can\'t', 'n\'t', 'B: ', 'A: ', ',', ';', '.', '?', '!', ':', '. ?', ',    .', '. ,', 'EOS', 'BOS', 'eos', 'bos']
    l2 = ['will not', 'will not', 'would not', 'would not', ' am', ' are', ' have', ' will', ' is', ' had', ' not', ' am', ' are', ' have', ' will', ' is', ' had', 'can not', ' not', '', '', ' ,', ' ;', ' .', ' ?', ' !', ' :', '? ', '.', ',', '', '', '', '']
    l3 = ['-', '_', ' *', ' /', '* ', '/ ', '\'"', ' \\"', '\\ ', '--', '...', '. . .']
    l4 = ['jeffrey', 'fred', 'benjamin', 'paula', 'walter', 'rachel', 'andy', 'helen', 'harrington', 'kathy',
'ronnie', 'carl', 'annie', 'cole', 'ike', 'milo', 'cole', 'rick', 'johnny', 'loretta', 'cornelius', 'claire', 'romeo',
'casey', 'johnson', 'rudy', 'stanzi', 'cosgrove', 'wolfi', 'kevin', 'paulie', 'cindy', 'paulie', 'enzo',
'mikey', 'i\97', 'davis', 'jeffrey', 'norman', 'johnson', 'dolores', 'tom', 'brian', 'bruce', 'john', 'laurie',
'stella', 'dignan', 'elaine', 'jack', 'christ', 'george', 'frank', 'mary', 'amon', 'david', 'tom', 'joe', 'paul',
'sam', 'charlie', 'bob', 'marry', 'walter', 'james', 'jimmy', 'michael', 'rose', 'jim', 'peter', 'nick', 'eddie',
'johnny', 'jake', 'ted', 'mike', 'billy', 'louis', 'ed', 'jerry', 'alex', 'charles', 'tommy', 'bobby', 'betty',
'sid', 'dave', 'jeffrey', 'jeff', 'marty', 'richard', 'otis', 'gale', 'fred', 'bill', 'jones', 'smith', 'mickey']

    raw_word = raw_word.lower()
    raw_word = raw_word.replace(', ' + name_of_computer, '')
    raw_word = raw_word.replace(name_of_computer + ' ,', '')

    for j, term in enumerate(l1):
        raw_word = raw_word.replace(term,l2[j])

    for term in l3:
        raw_word = raw_word.replace(term,' ')

    for term in l4:
        raw_word = raw_word.replace(', ' + term, ', ' + name)
        raw_word = raw_word.replace(' ' + term + ' ,' ,' ' + name + ' ,')
```

```
            raw_word = raw_word.replace('i am ' + term, 'i am ' + name_of_computer)
            raw_word = raw_word.replace('my name is' + term, 'my name is ' + name_of_computer)

        for j in range(30):
            raw_word = raw_word.replace('. .', '')
            raw_word = raw_word.replace('.  .', '')
            raw_word = raw_word.replace('..', '')

        for j in range(5):
            raw_word = raw_word.replace('  ', ' ')

        if raw_word[-1] !=  '!' and raw_word[-1] != '?' and raw_word[-1] != '.' and raw_word[-2:] !=
'! ' and raw_word[-2:] != '? ' and raw_word[-2:] != '. ':
            raw_word = raw_word + ' .'

        if raw_word == ' !' or raw_word == ' ?' or raw_word == ' .' or raw_word == ' ! ' or raw_word
== ' ? ' or raw_word == ' . ':
            raw_word = 'what ?'

        if raw_word == '  .' or raw_word == ' .' or raw_word == '  . ':
            raw_word = 'i do not want to talk about it .'

        return raw_word

    def tokenize(sentences):

        # Токенизация предложений в слова:
        tokenized_sentences = nltk.word_tokenize(sentences)
        index_to_word = [x[0] for x in vocabulary]
        word_to_index = dict([(w,i) for i,w in enumerate(index_to_word)])
        tokenized_sentences = [w if w in word_to_index else unknown_token for w in
tokenized_sentences]
```

```
    X = np.asarray([word_to_index[w] for w in tokenized_sentences])
    s = X.size
    Q = np.zeros((1,maxlen_input))
    if s < (maxlen_input + 1):
        Q[0,- s:] = X
    else:
        Q[0,:] = X[- maxlen_input:]

    return Q

# Сохранение диалога для последующего обучения
qf = open(file_saved_context, 'w')
af = open(file_saved_answer, 'w')

input_context = Input(shape=(maxlen_input,), dtype='int32', name='the context text')
input_answer = Input(shape=(maxlen_input,), dtype='int32', name='the answer text up to the current token')
LSTM_encoder = LSTM(sentence_embedding_size, kernel_initializer= 'lecun_uniform', name='Encode context')
LSTM_decoder = LSTM(sentence_embedding_size, kernel_initializer= 'lecun_uniform', name='Encode answer up to the current token')

if os.path.isfile(weights_file):
    Shared_Embedding = Embedding(output_dim=word_embedding_size, input_dim=dictionary_size, input_length=maxlen_input, name='Shared')
else:
    Shared_Embedding = Embedding(output_dim=word_embedding_size, input_dim=dictionary_size, weights=[embedding_matrix], input_length=maxlen_input, name='Shared')
word_embedding_context = Shared_Embedding(input_context)
context_embedding = LSTM_encoder(word_embedding_context)
```

```
word_embedding_answer = Shared_Embedding(input_answer)
answer_embedding = LSTM_decoder(word_embedding_answer)

merge_layer = concatenate([context_embedding, answer_embedding], axis=1,
name='concatenate the embeddings of the context and the answer up to current token')
out = Dense(dictionary_size//2, activation="relu", name='relu activation')(merge_layer)
out = Dense(dictionary_size, activation="softmax", name='likelihood of the current token using
softmax activation')(out)

model = Model(inputs=[input_context, input_answer], outputs = [out])

model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.00005))
model.summary()

if os.path.isfile(weights_file):
    model.load_weights(weights_file)

# Загрузка данных
vocabulary = pickle.load(open(vocabulary_file, 'rb'))

print("CHAT: \n")

# Обработка вопроса юзера
prob = 0
que = ''
last_query  = ' '
last_last_query = ''
text = ' '
last_text = ''
print("computer: hi ! please type your name.\n")
name = input('user: ')
print("computer: hi , " + name +" ! My name is " + name_of_computer + ".\n")
```

```
while que != 'exit .':
    que = input('user: ')
    que = preprocess(que, name_of_computer)
    # Сохранение данных
    q = last_query + ' ' + text
    a = que
    qf.write(q + '\n')
    af.write(a + '\n')
    # Формирование ответа
    if prob > 0.2:
        query = text + ' ' + que
    else:
        query = que

    last_text = text

    Q = tokenize(query)

    # Использование обученной модели для предсказания ответа
    predout, prob = greedy_decoder(Q[0:1])
    start_index = predout.find('EOS')
    text = preprocess(predout[0:start_index], name)
    print ('computer: ' + text + '    (with probability of %f)' % prob)

    last_last_query = last_query
    last_query = que

qf.close()
af.close()
```