

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных систем и сетей  
Кафедра Информатики

*К защите допустить:*

Заведующий кафедрой ПИКС

\_\_\_\_\_ И. Н. Цырельчук

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к дипломному проекту  
на тему:

**АЛГОРИТМЫ ПОСТРОЕНИЯ ВЕРОЯТНОСТНЫХ  
СЕТЕЙ**

БГУИР ДП 1-31 03 04 07 093 ПЗ

Студент

Ю. А. Ярошевич

Руководитель

А. А. Волосевич

Консультанты:

*от кафедры информатики*

А. А. Волосевич

*по экономической части*

А. В. Рябов

*по охране труда*

Е. А. Колосова

Нормоконтролёр

С. И. Сиротко

Рецензент

Минск 2013

## РЕФЕРАТ

*Ключевые слова:* вероятностные модели; байесовы сети; вывод структуры сети по данным; принцип минимальной длины описания; оценка апостериорной вероятности.

Дипломный проект выполнен на 6 листах формата А1 с пояснительной запиской на 18 страницах, без приложений справочного или информационного характера. Пояснительная записка включает 2 глав, 7 рисунков, 0 таблиц, 25 формул и 9 литературный источник.

Целью дипломного проекта является разработка удобного в использовании инструмента, пригодного для решения практических задач, возникающих в реальных проектах, связанных с вероятностным моделированием.

Для достижения цели дипломного проекта была разработана библиотека кода для Microsoft .NET, предназначенная для представления и обучения структуры вероятностной сети по экспериментальным данным. Библиотека может быть использована в реальных проектах, использующих вероятностный подход к решению проблемы. В библиотеке реализовано несколько алгоритмов, имеющих различные качественные характеристики.

В разделе технико-экономического обоснования был произведён расчёт затрат на создание ПО, а также прибыли от разработки, получаемой разработчиком. Проведённые расчёты показали экономическую целесообразность проекта.

Пояснительная записка включает раздел по охране труда, в котором была произведена оценка пожарной безопасности на предприятии, где частично разрабатывался данный дипломный проект.

## АННОТАЦИЯ

на дипломный проект «Алгоритмы построения вероятностных сетей» студента УО «Белорусский государственный университет информатики и радиоэлектроники» Ярошевича Ю. А.

*Ключевые слова:* вероятностные модели; байесовы сети; вывод структуры сети по данным; принцип минимальной длины описания; оценка апостериорной вероятности.

Дипломный проект выполнен на 6 листах формата А1 с пояснительной запиской на 18 страницах, без приложений справочного или информационного характера. Пояснительная записка включает 2 глав, 7 рисунков, 0 таблиц, 25 формулы, 9 литературный источник.

Целью дипломного проекта является разработка удобного в использовании инструмента, пригодного для решения практических задач, возникающих в реальных проектах, связанных с вероятностным моделированием.

Для достижения цели дипломного проекта была разработана библиотека кода для Microsoft .NET, предназначенная для представления и обучения структуры вероятностной сети по экспериментальным данным. Библиотека может быть использована в реальных проектах, использующих вероятностный подход к решению проблемы. В библиотеке реализовано несколько алгоритмов, имеющих различные качественные характеристики.

Во введении производится ознакомление с проблемой, решаемой в дипломном проекте.

В первой главе производится обзор предметной области проблемы решаемой в данном дипломном проекте. Приводятся необходимые теоретические сведения, а также производится обзор существующих разработок.

Во второй главе производится краткий обзор технологий, использованных для реализации ПО в рамках дипломного проекта.

В третьей главе производится обзор реализованного ПО. Описываются его составные части и особенности. Приводятся результаты практических испытаний и производится сравнение с существующим ПО.

В четвертой главе производится оценка пожарной безопасности предприятия, на котором частично разрабатывался данный дипломный проект.

В пятой главе производится технико-экономическое обоснование разработки.

В заключении подводятся итоги и делаются выводы по дипломному проекту, а также описывается дальнейший план развития проекта.

## СОДЕРЖАНИЕ

1	ОБЗОР ЛИТЕРАТУРЫ . . . . .	6
1.1	Обзор аналогов . . . . .	6
1.2	Процесс выделения особенностей . . . . .	7
1.3	Технология word2vec . . . . .	8
1.4	Классическая рекурсивная нейронная сеть . . . . .	10
1.5	Long Short Term Memory . . . . .	11
1.6	Рекурсивная тензорная нейронная сеть . . . . .	13
1.7	Tree LSTM . . . . .	14
1.8	Классификация . . . . .	15
2	СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ . . . . .	16
	Список использованных источников . . . . .	18

# 1 ОБЗОР ЛИТЕРАТУРЫ

## 1.1 Обзор аналогов

Крупнейшими продуктами на рынке анализа тональности текста для множества естественных языков являются: Natural Language API, поставляемый на Google Cloud Platform; Text Analysis API, поставляемый на платформе Microsoft Azure; IBM Watson; WIT.AI принадлежащий Facebook. Данные решения представляют собой облачные сервисы, предоставляющие API пользователям, которое позволяет загружать тексты на естественных языках и возвращает их анализ. Кроме анализа тональности эти сервисы предоставляют и другие возможности, как определение частей речи (Part of Speech Tagging, PoS Tagging), морфологический разбор слов и синтаксический разбор предложений. Такие системы носят название NLP-конвейера (Natural Language Processing). К сожалению исходные коды этих систем закрыты, а применяемые в них принципы если и распространяются, то описаны в платных научных изданиях. Однако можно уверенно предположить, что Google применяет Globally Normalized Transition-Based Neural Networks[1]. Так как реализацию данного подхода Google анонсировал в виде мощного и высокопроизводительного NLP инструмента SyntaxNet.

Так же стоит обратить внимание на крупный академический проект от Стенфордского университета — CoreNLP. CoreNLP — это открытый NLP конвейер, демонстрирующий практические возможности методов, изобретенных в результате широкого ряда исследований сотрудников Стенфорда.

Итак, вне зависимости от огласки принципов работы NLP-конвейеров, все ещё можно сравнить их бизнес-логику. Сравнение возможностей описанных выше закрытых коммерческих проектов друг с другом возможно только в ходе мощного маркетингово исследования. Но если сравнить их с некоммерческим CoreNLP, то станет очевидно, что CoreNLP, являясь лишь демонстрацией, очень отстаёт по производительности, но предоставляет более детальную визуализацию анализа тональности. CoreNLP в результате анализа предложения выдает синтаксическое дерево, узлами которого являются слова, а ветвями — синтаксические связи между словами. Таким образом каждое поддерево представляет собой фразу. И оценка тональности указывается для каждого узла. То есть, потенциальный пользователь может легко понять каким образом давалась оценка тональности всего предложения, глядя на оценки отдельных фраз. Коммерческие же продукты такой возможности не предоставляют, так как дают оценку всему предложению целиком в силу своей структуры. Таким образом, задача проектирования в создании модели с высокими возможностями визуализации, с использованием высокопроизводительных технологий.

## 1.2 Процесс выделения особенностей

Обычно задачи, связанные с классификацией текстов, можно разделить на три этапа: выделение особенностей, сжатие предложений и классификация. На первом этапе из текста выбирают слова для обработки и выбирают соответствующие им векторы. Затем производят сжатие набора векторов соответствующих словам в предложении в один вектор, который будет представлять все предложение. И далее полученные векторы предложений классифицируют[2]. Для классификации предложений выделение особенностей начинают со статической обработки текста, например убирают символы переноса строки, или заменяют символы, которые могут использоваться в формате представления набора данных, на аналоги или ключевые слова. Часто используется замена символов круглых скобок на сокращения “-LRB-” и “-RRB-”. С этим легко справится механизм регулярных выражений. Затем необходимо разбить предложения на единицы языка, несущие независимое семантическое значение — произвести токенизацию. Для большинства языков регулярные выражения так же справятся с задачей. Для русского языка хватит разбиения по символу пробела, в английском надо будет учитывать ещё и апострофы (“It’s” разобьется на “It” и “’s”). Однако для обработки большинства восточных языков так же придется прибегать к машинному обучению, так как синтаксическое разделение слов на письме часто отсутствует, и токенизацию можно произвести только анализируя семантическое значение предложения. Для ограничения масштаба в данной работе методы токенизации рассматриваться не будут[2]. В работе для токенизации и синтаксического разбора будут использованы средства CoreNLP встроенные в библиотеку NLTK для Python.

Итак, после токенизации предложения представляют из себя набор единиц языка в строковом формате. Но нейронные сети работают только с числами. Поэтому необходимо представить слова в векторном виде. Данный процесс называется *встраивание слов*. Классический метод one-hot-encoding предлагает представлять слова в виде позиционных кодов. Для всех уникальных слов в корпусе строится словарь, где каждому слову соответствует вектор заполненный нулями и одной единицей, соответствующей позиции слова в словаре. Очевидно, это очень производительный метод встраивания слов, так как алгоритм кодирования слов в новом корпусе имеет линейную сложность. Однако прямая зависимость размера вектора от количества уникальных слов в корпусе может вызвать проблемы с хранением этих векторов. Например, корпус книг от Google содержит 1 миллион уникальных слов, и с one-hot-encoding каждое слово будет занимать 30 мегабайт памяти, если пользоваться в вычислениях хотя бы 32-битными числами. Так же семантическое значение слов в данном методе теряется — сравнение синонимов, антонимов и никак не связанных между собой по значению слов даст один и тот же результат. Метод сжатия векторов слов в единый век-

## Document 1

The quick brown fox jumped over the lazy dog's back.

## Document 2

Now is the time for all good men to come to the aid of their party.

### Term

Document 1

Document 2

aid	0	1
all	0	1
back	1	0
brown	1	0
come	0	1
dog	1	0
fox	1	0
good	0	1
jump	1	0
lazy	1	0
men	0	1
now	0	1
over	1	0
party	0	1
quick	1	0
their	0	1
time	0	1

Рисунок 1.1 – Пример мешков для двух предложений[2]

тор, соответствующий предложению, заключается в простом суммировании векторов слов и носит название *Мешок слов*. [2]. Пример мешков слов представлен на рисунке 1.1.

Скалярное произведение двух таких векторов предложений даст степень схожести этих предложений, основывающейся только на количестве одинаковых слов встречающихся в обоих предложениях. Таким образом данный метод для трех предложений “Мальчик ударил мяч.”, “Мальчик ударился в учебу” и “Дети играют в футбол” сделает вывод о том, что схожи первые два предложения, хотя очевидно что по смыслу больше связаны первое и третье. Обучение нейронной сети на выборке из мешков слов не даст никаких результатов для задач классификации по значению не только по причине отсутствия семантики в векторных представлениях слов и фраз, но и из-за представления в виде позиционных кодов, когда большая часть вектора не несет никакой нагрузки, а содержит лишь ноли[2].

### 1.3 Технология word2vec

Существует ряд методов встраивания слов, в которых векторы сохраняют семантическое значение обрабатываемых слов. В основе этих методов лежит идея о том, что семантика слова заключена в контексте его применения. Самым значимым результатом исследований на почве этой идеи является технология word2vec. На сегодняшний день это самый распростра-

ненный и эффективный метод встраивания слов. Качественно обученная модель word2vec представляет собой словарь, в котором словам соответствуют так называемые плотные вектора[3].

Процесс обучения модели word2vec начинается с генерации произвольных значений векторов для изучаемых слов. Каждому слову будет соответствовать два плотных вектора, так как слово в процессе обучения может участвовать в роли центрального слова, и в роли слова из контекста центрального слова. На каждом шаге обучения в тексте последовательно выбирается центральное слово и его контекст — слова которые отстоят от центрального на  $m$  слов слева и справа. Для каждого центрального слова  $t$  делается предсказание слов в контексте[3]. Целевой функцией оптимизации в данном случае будет

$$J'(\Theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} p(w_{t+j}|w_t; \Theta), \quad (1.1)$$

где  $\Theta$  — это параметры модели, изменяемые в ходе обучения.

Тогда отрицательная логарифмическая функция максимального правдоподобия  $J(\Theta)$  будет равна:

$$J(\Theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log(p(w_{t+j}|w_t)). \quad (1.2)$$

Для предсказания вероятности слова в контексте применяется функция softmax, описанная в общем виде в выражении

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^K e^{z_k}}. \quad (1.3)$$

Вероятность нахождения слова  $o$  в контексте слова  $c$  — это softmax функция для  $c$  по  $o$

$$p(o|c) = \frac{\exp(u_o^T \cdot v_c)}{\sum_{w=1}^V \exp(u_w^T \cdot v_c)}. \quad (1.4)$$

Матрица  $U$  хранит вектора для слов из контекста, а  $V$  — для центральных слов. Градиент для обратного прохода для 1.2 и 1.4 будет равен:

$$\frac{\partial}{\partial v_c} \frac{\exp(u_o^T \cdot v_c)}{\sum_{w=1}^V \exp(u_w^T \cdot v_c)} = u_o - \sum_{x=1}^V p(x|c) \cdot u_x. \quad (1.5)$$

После того, как градиент рассчитан значение градиента отнимается от всех обучаемых параметров модели, то есть от матриц  $U$  и  $V$ .

В результате множества итераций обучения будет получена простейшая softmax модель word2vec, которая будет представлять из себя матрицу  $V$ . Полученные вектора хранят семантическое значение слов, таким обра-



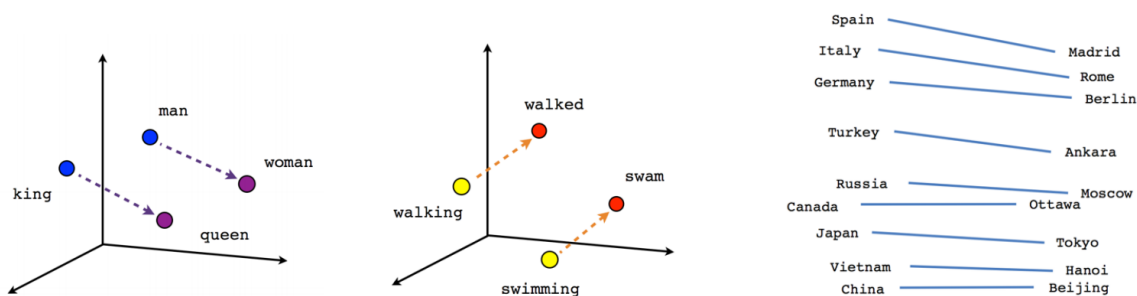


Рисунок 1.2 – Отношение между векторами в примере word2vec[3]

зом близкие по смыслу слова и синонимы будут располагаться близко друг от друга. А математические операции над этими векторами будут давать интересные результаты. Например если отнять от вектора “Король” вектор “Мужчина” и добавить вектор “Женщина” то будет получен вектор слова “Королева”. На рисунке 1.2 видно, что вектора располагаются параллельно вдоль некоторых осей, выученных моделью, по признакам пола, части речи и географического положения. Реализовать статическую модель распознающую подобные признаки невероятно трудно[3].

#### 1.4 Классическая рекурсивная нейронная сеть

Когда каждое предложение представляет из себя набор вектор, соответствующих языковым единицам, необходимо провести сжатие — обработать вектора слов таким образом, чтобы каждому предложению соответствовал один плотный вектор. Простейшая нейронная сеть, которая может быть применена в данной задаче — это рекурсивная нейронная сеть (RNN). Данная сеть имеет два входа:  $x_t$  — входной вектор,  $s_{t-1}$  — вектор состояния с прошлой итерации. Сеть последовательно применяется к векторам в предложении слева направо, и выдает на каждой итерации выходной вектор  $o_t$  и вектор состояния  $s_t$ .

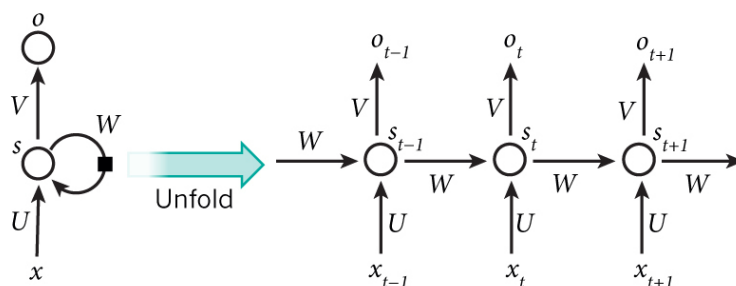


Рисунок 1.3 – Рекурсивная нейронная сеть в развертке во времени[2]

Состоит сеть из трех матриц весов: входная матрица  $U$ , матрица состояний  $W$ , матрица выходов  $V$ . В описывается двумя функциями 1.6 и 1.7.

Softmax функция описана в 1.3. Пример рекурсивной нейронной сети и её развертки во времени представлен на рисунке 1.3.

$$s_t = f(U \cdot x_t + W \cdot s_{t-1}), \quad (1.6)$$

$$o_t = \text{softmax}(V s_t), \quad (1.7)$$

где  $f$  — это функция активации, обычно выбирают тангенс.

После обхода всего предложения, получается выходной вектор, и появляется возможность посчитать функцию потерь, и вычислить значение градиента. На рисунке fig:overview:rnn показана развертка во рекурсивной сети во времени. Таким образом модель можно представить в виде многослойной сети. А значит модель страдает от проблемы затухающего градиента, когда градиентный спуск необходимо произвести на множество слоев вниз, и значение функции ошибки может очень сильно отличаться от реального её значения. Но так как применяется одна и та же сеть на каждом слое, то проблема затухающего градиента для рекурсивной сети усиливается. Поэтому с помощью RNN не было получено высоких результатов в классификации предложений. Так же рекурсивная сеть сталкивается с проблемой отдаленных зависимостей. Суть проблемы в том, что семантически связанные слова, которые имеют главную роль в понимании предложения, могут находиться удаленно друг от друга в предложении. И так как RNN обходит предложение слева на право, то информация о том, что первое слово встречалось в предложении может уже быть потеряно к тому моменту, как на вход придет второе, и модель сделает неверный вывод.

## 1.5 Long Short Term Memory

Проблему отдаленных зависимостей решает нейронная сеть под названием Long Short Term Memory (LSTM). Это особый вид рекурсивной нейронной сети, способная выучить удаленные зависимости. Данное свойство было получено за счет усложнения структуры сети. Принцип применения и обучения остался таким же: предложение обходится слева направо последовательно подавая на вход сети плотные вектора соответствующие словам. Схема сети представлена на рисунке 1.4.

Внутреннее состояние сети описывается вектором состояния ячейки  $C_t$  и вектором скрытого слоя  $h_t$ . На вход принимает входной вектор  $x_t$  и состояние с прошлой итерации:  $C_{t-1}$  и  $h_{t-1}$ . Выходной вектор  $o_t$  равен вектору скрытого слоя  $h_t$ . Введена система врат. Врата блокируют или пропускают векторы между различными состояниями нейронной сети.  $f_t$  — сигнал забывания, сбрасывает состояния ячейки.  $i_t$  — входные врата, блокируют или пропускают входной вектор. И  $o_t$  — выходные врата, блокируют или пропускают выходной вектор[4].

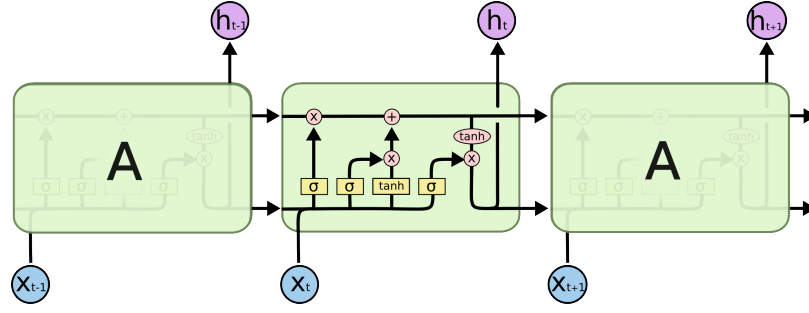


Рисунок 1.4 – Схематическое изображение LSTM-сети[4]

Состояния врат описаны выражениями 1.9, 1.11 и 1.10.

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad (1.8)$$

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i), \quad (1.9)$$

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f), \quad (1.10)$$

$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o), \quad (1.11)$$

где  $\sigma$  это функция сигмоида описанная 1.8;

$W$  и  $U$  — это тензоры весов LSTM.

Состояния ячейки вычисляются согласно выражениям 1.13, 1.14 и 1.15.

$$(A \odot B)_{i,j} = (A)_{i,j} \cdot (B)_{i,j}, \quad (1.12)$$

$$\tilde{C}_t = \tan(W_C \cdot x_t + U_c \cdot h_{t-1} + b_c), \quad (1.13)$$

$$C_t = f_t \odot c_{t-1} + i_t \odot \tilde{C}_t, \quad (1.14)$$

$$h_t = o_t \odot \tan(c_t), \quad (1.15)$$

где  $\tilde{C}_t$  носит название кандидата в состояние ячейки;

$\odot$  — это операция произведения Адамара, описанная в 1.12 для двух матриц  $A$  и  $B$ .

LSTM на сегодняшний день лежит в основе самой эффективной модели анализа тональности предложений Sentiment Neuron от сообщества OpenAi. Помимо лидерства на текущий момент, Sentiment Neuron обучается без учителя — это единственная успешная модель способная сжимать вектора слов в плотный вектор предложения и обучающаяся без учителя. Однако обучение этой модели крайне дорого. OpenAi обучали её на четырех Nvidia Pascal Titan X и обучение заняло приблизительно один месяц[5].

## 1.6 Рекурсивная тензорная нейронная сеть

Проблема отложенных связей может решаться иначе. Очевидно, что порядок слов в предложении редко совпадает с нитью размышлений автора. Обход предложения слева направо не может запомнить все семантически значимые последовательности, особенно в языках со специфической грамматикой. Поэтому исследователи решили изменить порядок обхода предложений. Один из итогов исследований — это модель рекурсивной нейронной тензорной сети (RNTN), которая легла в основу CoreNLP. Обход предложения производится по синтаксическому дереву, построенному согласно генеративной грамматике Хомского[6]. Итак, в процесс выделения особенностей добавляется ещё один шаг — синтаксический анализ предложения. Для RNTN необходимо на входе иметь синтаксическое дерево составляющих — один из видов синтаксических деревьев. Это дерево удобно тем, что его можно нормализовать, то есть привести к виду бинарного дерева. На рисунке 1.5 показан пример синтаксического дерева составляющих[6].

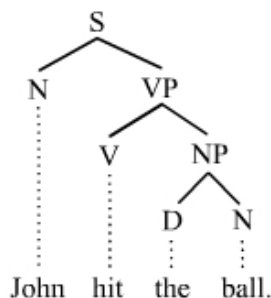


Рисунок 1.5 – Пример синтаксического дерева составляющих[7]

Как видно из рисунка 1.5, слова представлены листьями дерева, а его узлы — это различные виды составляющих в предложении. Типы связей не нужны в модели RNTN, интересен только сам факт их наличия и какие слова и фразы они объединяют. Для того, чтобы эффективно обрабатывать дерево составляющих, RNTN модифицирована и имеет два входа, и один выход. То есть она принимает два вектора с нижних узлов и передает верхнему, и т.д., пока в результате обработки всего дерева не будет получен один вектор, соответствующий всему предложению. Так же модель выдает вектор для каждого узла в дереве, что соответствует фразам в предложении. Это свойство эффективно применяется при обучении. Специально для обучения RNTN был создан Stanford Sentiment Treebank (SST) — Стенфордский набор деревьев тональности. Это набор более чем из десяти тысяч синтаксических деревьев, все узлы которых оценены по тональности носителями языка. SST стал очень популярен и вне исследований RNTN и является классическим набором для измерения эффективности метода оценки тональности[8].

Пусть  $\begin{bmatrix} a \\ b \end{bmatrix}$  — два вектора размерности  $d$  объединенные в один размерностью  $2d$ . Тогда значение вектора результата  $p$  для входных векторов  $a$  и  $b$  вычисляется по формуле 1.16[8].

$$p = f\left(\begin{bmatrix} a \\ b \end{bmatrix}^T \cdot V \cdot \begin{bmatrix} a \\ b \end{bmatrix} + W \cdot \begin{bmatrix} a \\ b \end{bmatrix}\right), \quad (1.16)$$

## 1.7 Tree LSTM

На следующий год после релиза RNTN, изменяется подход в синтаксическом разборе предложений, так как выходит работа с описанием алгоритма синтаксического разбора с линейной сложностью. Однако этот алгоритм возвращает дерево зависимостей, а RNTN работает с деревом составляющих. Пример дерева зависимостей представлен на рисунке 1.6. Дерево зависимостей несет в себе меньше информации, чем дерево составляющих, так как дерево составляющих можно сконвертировать в зависимости без потерь, а обратный процесс без потерь невозможен[6].

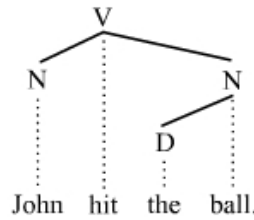


Рисунок 1.6 – Пример синтаксического дерева зависимостей[7]

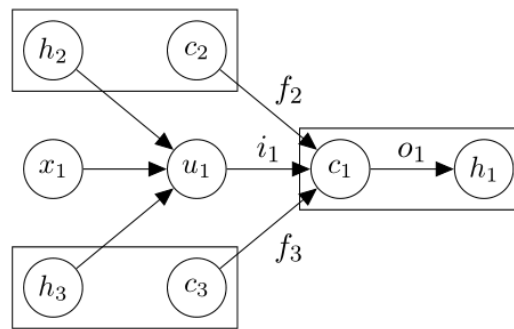


Рисунок 1.7 – Пример подключения узла Tree LSTM к дочерним узлам.  $h_2$ ,  $c_2$ ,  $h_3$ ,  $c_3$  — состояния двух дочерних узлов,  $h_1$ ,  $c_1$  — состояние узла родителя.  $x_1$  — входной вектор узла.[9]

Дерево зависимостей для задачи классификации отличается тем, что каждый узел дерева содержит вектор слова. Так же дерево не является би-

нарным — каждый узел может иметь произвольное количество детей. Для обработки подобных деревьев была разработана модель Tree LSTM, которая существует в двух модификациях:  $N$ -арная Tree LSTM, для деревьев составляющих и Child-Sum Tree LSTM, для работы с деревьями зависимостей. В данной работе была реализована модель Child-Sum Tree LSTM[9].

Итак, ячейка Child-Sum Tree LSTM в некотором узле дерева принимает на вход состояния дочерних узлов. Пример подключения показан на рисунке 1.7. Состояние ячейки Tree LSTM, как и в обычной LSTM, описано двумя векторами:  $h_j$  и  $c_j$  — значение скрытого слоя и внутреннего состояния соответственно. Состояние для узла  $j$ , имеющего множество детей  $C(j)$  можно выразить следующим образом:

$$\tilde{h}_j = \sum_{k \in C(j)} h_k, \quad (1.17)$$

$$i_j = \sigma(W^{(i)} \cdot x_j + U^{(i)} \cdot \tilde{h}_j + b^{(i)}), \quad (1.18)$$

$$f_{jk} = \sigma(W^{(f)} \cdot x_j + U^{(f)} \cdot h_k + b^{(f)}), \quad (1.19)$$

$$o_j = \sigma(W^{(o)} \cdot x_j + U^{(o)} \cdot \tilde{h}_j + b^{(o)}), \quad (1.20)$$

$$u_j = \tan(W^{(u)} \cdot x_j + U^{(u)} \cdot \tilde{h}_j + b^{(u)}), \quad (1.21)$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k, \quad (1.22)$$

$$h_j = o_j \odot \tan(c_j), \quad (1.23)$$

где  $k \in C(j)$  для выражения 1.19.

## 1.8 Классификация

Последним этапом NLP-конвейера будет классификация плотных векторов предложений, полученных в результате сжатия. Стоит отметить, что плотные вектора предложений кодируют различные семантические свойства этих предложений, а значит с их помощью можно решать различные задачи классификации. Например сравнивать предложения по значению, похожи ли они, или противоположны, либо же нейтральны. Однако в данной работе интересная задача оценки тональности текста[2].

Для оценки тональности обычно обучают простейшую регрессионную модель

$$p_{\Theta}(y|x_j) = \text{softmax}(W \cdot h_j + b), \quad (1.24)$$

$$y_j = \arg \max_y (p_{\Theta}(y|x_j)), \quad (1.25)$$

где  $h_j$  — это плотный вектор предложения;

$\arg \max_y$  — функция, которая вернет индекс максимального элемента вектора.

## 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Изучив теоретические аспекты разрабатываемого приложения и выработав список требований необходимых для разработки модуля, разбиваем систему на компоненты. В разрабатываемом приложении можно выделить следующие блоки:

- модуль сбора статистики;
- модуль объектно-реляционного отображения;
- база данных объектов для анализа;
- модуль развертки;
- тренировочный набор Стенфорда;
- модуль выделения особенностей;
- модуль модели анализатора;
- модуль ячейки Tree LSTM;
- визуализации CoreNLP;
- модуль визуализации TensorBoard.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.009 С1.

Чтобы начать работу модели распознавания тональностей, необходимы входные данные. Первый тип входных данных — это тренировочный набор Стенфорда. Он необходим для обучения модели распознаванию тональностей. Стенфордский набор Представляет собой выборку синтаксических деревьев составляющих. Однако модель должна работать с синтаксическими деревьями зависимостей. Поэтому необходимо привести его к виду синтаксических деревьев зависимостей. Так как исследование синтаксического разбора не входит в задачи проектирования, то для этой цели будет использоваться синтаксический анализатор из CoreNLP. Он не является самым производительным, но самым простым в применении. За несколько секунд он обрабатывает более 10000 предложений, составляющих Стенфордский тренировочный набор. Обработать тренировочный набор необходимо всего один, при установке программы, поэтому скорость в несколько секунд вполне удовлетворительна. Объем анализатора вместе с моделью нейросети, обученной синтаксическому разбору, составляет более 350 мегабайт. Далее нужен предобученный набор встроенных векторов GloVe. Он содержит в себе 5 гигабайт слов и соответствующих им векторам размерностью в 300 составляющих каждый. Чтобы сократить размер, занимаемый набором GloVe, он будет отфильтрован: будут выброшены все слова, которые не встречаются в Стенфордском тренировочном наборе. После такой фильтрации набор встраиваемых векторов будет занимать около 30 мегабайт дискового пространства. Фильтрацию так же необходимо произвести только один раз при установке программы. То есть, если перед началом работы программы произвести предобработку входных данных, то это повысит производительность и сократить объем, занимаемый системой

на диске. Итак, чтобы решать проблему управления зависимостями, введен модуль развертки. В его задачи входит:

- проверить наличие отфильтрованного набора GloVe;
- проверить наличие Стенфорского тренировочного набора в виде синтаксических деревьев зависимости;
- проверить наличие синтаксического анализатора и скачать его, если отсутствует;
- при необходимости скачать оригинальные набор GloVe и тренировочный набор;
- при необходимости отфильтровать GloVe и удалить оригинальную версию;
- при необходимости обработать оригинальный Стенфордский тренировочный набор с помощью синтаксического анализатора.



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Globally Normalized Transition-Based Neural Networks / Daniel Andor [et al.].
- 2 Goodfellow, Ian. Deep Learning / Ian Goodfellow, Yoshua Bengio, Aaron Courville. — MIT Press, 2016. — <http://www.deeplearningbook.org>.
- 3 Distributed Representations of Words and Phrases and their Compositionality / Tomas Mikolov [et al.].
- 4 Hochreiter, Sepp. Long Short-Term Memory / Sepp Hochreiter, Jürgen Schmidhuber // Neural Computation. — 1997. — Vol. 9, no. 8. — Pp. 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>.
- 5 Radford, Alec. Learning to Generate Reviews and Discovering Sentiment / Alec Radford, Rafal Józefowicz, Ilya Sutskever.
- 6 Chomsky, Naom. Three models for the description of language / Naom Chomsky. — IRE Transactions on Information Theory, 1956.
- 7 Wikipedia contributors. Dependency grammar — Wikipedia, The Free Encyclopedia. — 2004. — [Online; accessed 11-April-2018]. [https://en.wikipedia.org/wiki/Dependency\\_grammar](https://en.wikipedia.org/wiki/Dependency_grammar).
- 8 Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank / Richard Socher [et al.]. — 2013.
- 9 Tai, Kai Sheng. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks / Kai Sheng Tai, Richard Socher, Christopher D. Manning.