Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки Прикладная математика

РАБОТА ПРОВЕРЕНА
Рецензент, доцент кафедры____
ЗИ, ЮУрГУ, к. т. н._____
_____\В.П. Мартынов
«____»_____2018г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
доцент
_____/А.А.Замышляева
«____»_____2018 г.

Применение нейросетевого программирования в реализации криптографических систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–01.03.04.2018.123.ПЗ ВКР

Руководитель работы, профессор
_____/Н.Д. Зюляркина
«____»_____2018 г.
Автор работы
студент группы ЕТ-413
_____/К.Г. Кузьменко
«____»_____2018 г.
Нормоконтролер, к. н., доцент
_____/Д.А. Дрозин
«____»_____2018 г.

Челябинск 2018

АННОТАЦИЯ

Кузьменко К.Г. Применение нейросетевого программирования в реализации криптографических систем. – Челябинск: ЮУрГУ, ЕТ-413, 61 с., 21 ил., библиогр. список – 24 наим., 5 прил,

Выпускная квалификационная работа выполнена с целью помочь специалистам по криптографическим системам в разработке шифров более устойчивых к криптоанализу.

В работе исследован криптоанализ шифров с целью выявления их отличительных особенностей для дальнейшего их распознавания программой.

Разработана сверточная нейронная сеть для анализа зашифрованных текстов. Реализована программа для распознавания шифров, которыми были зашифрованы тексты.

В ходе поиска аналогичных решений в открытом доступе таковых найдено не было.

Программа реализована на языке Python 3.6, в интегрированной среде разработки программ Spyder.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы. Криптография – наука о методах обеспечения конфиденциальности и аутентичности информации. Криптография включает в себя методы шифрования информации, ассиметричные криптосистемы, системы электронной цифровой подписи, хеш-функции, управление ключами, получение скрытой информации, а также квантовую криптографию. Ключевой задачей криптографии является создание стойких алгоритмов шифрования. Любой конструируемый алгоритм подвергается тщательному анализу с целью выявления его слабых мест и возможности взлома. Алгоритм является относительно стойким до тех пор, пока не будут обнаружены методы и пути его анализа, позволяющие получить секретный ключ шифрования значительно быстрее, чем это можно сделать с использованием метода «грубого перебора».

Криптоанализом называют науку восстановления открытого текста без доступа к ключу. Основная задача криптоанализа состоит в том, чтобы определить вероятность взлома шифра и, таким образом, оценить его применимость в той или иной области. В ходе работы был рассмотрен криптоанализ некоторых шифров с целью определить по зашифрованным текстам отличительные особенности данных шифров.

В современном мире с ростом вычислительных мощностей компьютеров стойкость алгоритмов шифрования снижается. Необходимостью разработки программ для проверки стойкости к криптоанализу алгоритмов шифрования, обуславливается актуальность данной работы.

Цель работы – разработка нейронной сети для реализации компьютерной программы, позволяющей определить тип шифра, которым был зашифрован имеющийся текст.

Задачи работы:

1) провести анализ предметной области;

2) разработка математической модели и архитектуры нейронной сети, решающей задачу распознавания типа шифров;

3) создание компьютерной программы, реализующей данную модель;

4) проверка работы программы на экспериментальных данных.

1 ПРЕДВАРИТЕЛЬНЫЕ СВЕДЕНИЯ

1.1 Шифр перестановки

Шифры перестановки – это метод шифрования, в котором ключом шифра перестановки является перестановка номеров символов открытого текста. Это, в частности, означает, что длина ключа шифрования должна быть равна длине преобразуемого текста. Для того чтобы из секретного ключа получить ключ шифрования, удобный для использования в шифрах перестановки, предложен ряд методов.

Широкое распространение получили маршрутные перестановки. Открытый текст записывается в некоторую фигуру по некоторой траектории. Шифрованный текст получается путем выписывания текста по другой траектории. Например, текст можно записать в таблицу (рисунок 1) горизонтально слева направо, а зашифрованный текст считывается вертикально, с левого верхнего угла и двигаясь сверху вниз.

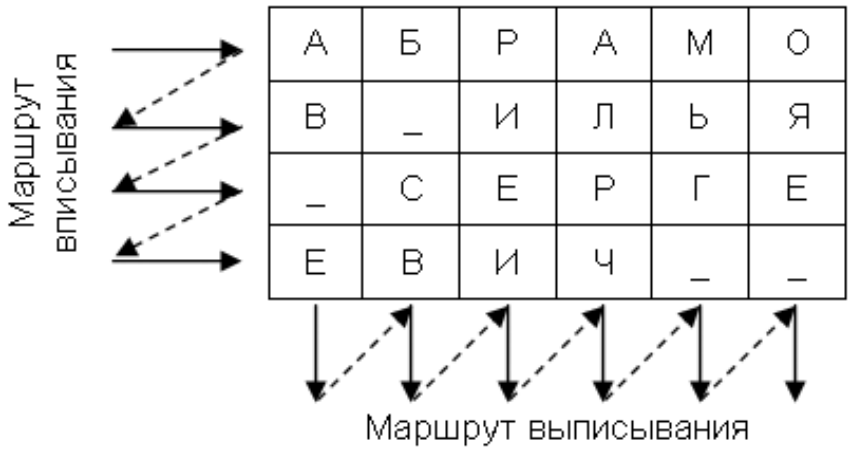

Рисунок 1 – Пример маршрутной перестановки

Поэтому при дешифровании следует попытаться соединить две группы последовательных букв криптограммы так, чтобы они образовывали хорошие (читаемые) с точки зрения обычного текста комбинации. Для этого естественно использовать наиболее частые биграммы открытого текста, которые можно составить из букв рассматриваемого шифрованного текста. Если для первой пробы выбрано,

скажем, сочетание СТ (самая частая биграмма русского языка), то мы можем по очереди приписывать к каждой букве С криптограммы букву Т из нее (рисунок 2). При этом несколько букв, стоящих до и после данной буквы С, и несколько букв, стоящих до и после данной буквы Т, соединяются в пары, то есть получаются два столбца букв, записанных рядом: С Т.

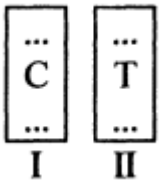

Рисунок 2 – Составление биграммы

Конечно, мы не знаем длины столбцов, но некоторые ограничения на них можно получить, используя положение конкретных букв. Так, столбцы должны иметь одинаковые длины или первый столбец может быть длиннее второго на одну букву, и тогда эта буква – последняя буква сообщения: С…АМ; Т…Б (рисунок 3)

Рисунок 3 – Признак последнего символа сообщения

Если приписываемые друг к другу буквы разделены, скажем, только двумя буквами, то, как легко видеть, мы можем ставить в соседних столбцах не более трех пар, и длина каждого столбца не превышает четырех. Кроме того, ограничением может послужить появление запретной биграммы (рисунок 4) (например, гласная – мягкий знак):

Рисунок 4 – Пример запретной биграммы

Для выбранного сочетания СТ получается по одной паре столбцов для каждого конкретного выбора букв С и Т из криптограммы, и из них целесообразно отобрать ту пару, которая содержит наиболее частые биграммы.

Заметим, что при автоматизации этого процесса можно приписать каждой биграмме вес, равный частоте ее появления в открытом тексте. Тогда целесообразно отобрать ту пару столбцов, которая имеет наибольший вес. Кстати, появление одной биграммы с низкой частотой может указать на то, что длину столбца надо ограничить.

Выбрав пару столбцов, мы аналогичным образом можем подобрать к ним третий (справа или слева) и т.д. Описанная процедура значительно упрощается при использовании вероятных слов, то есть слов, которые могут встретиться в тексте с большой вероятностью.

Рассмотрим также метод, применимый к любым шифрам перестановки. Допустим, что к двум или более сообщениям (или отрезкам сообщений) одинаковой длины применяется один и тот же шифр перестановки. Тогда очевидно, что буквы, которые находились на одинаковых местах в открытых текстах, окажутся на одинаковых местах и в шифрованных текстах.

Выпишем зашифрованные сообщения одно под другим так, что первые буквы всех сообщений оказываются в первом столбце, вторые – во втором и т.д. Если предположить, что две конкретные буквы в одном из сообщений идут одна за другой в открытом тексте, то буквы, стоящие на тех же местах в каждом из остальных сообщений, соединяются подобным же образом. Значит, они могут служить проверкой правильности первого предположения, подобно тому как комбинации, которые дают два столбца в системе вертикальной перестановки, позволя-

ют проверить, являются ли соседними две конкретные буквы из этих столбцов. К каждому из указанных двухбуквенных сочетаний можно добавить третью букву для образования триграммы и т.д. Если располагать не менее чем четырьмя сообщениями одинаковой длины, то можно с уверенностью гарантировать их вскрытие подобным образом.

1.2 Шифры простой замены

Шифры простой замены – класс методов шифрования, которые сводятся к созданию по определенному алгоритму таблицы шифрования, в которой для каждой буквы открытого текста существует единственная сопоставленная ей буква шифр-текста. Само шифрование заключается в замене букв согласно таблице.

Примером шифра простой замены может служить шифр Цезаря. Согласно описаниям историка Светония в книге «Жизнь двенадцати цезарей» данный шифр использовался Гаем Юлием Цезарем для секретной переписке со своими генералами (I век до н.э.). Применительно к русскому языку суть его состоит в следующем. Выписывается исходный алфавит (А, Б, ..., Я), затем под ним выписывается тот же алфавит, но с циклическим сдвигом на 3 буквы влево (рисунок 5).

А	Б	В	Г	Д	Е	Ё	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ы	Ь	Ъ	Э	Ю	Я
Г	Д	Е	Ё	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ы	Ь	Ъ	Э	Ю	Я	А	Б	В

Рисунок 5 – Таблица шифрозамен для шифра Цезаря

При зашифровке буква А заменяется буквой Г, Б - на Д и т. д. Так, например, исходное сообщение «АБРАМОВ» после шифрования будет выглядеть «ГДУГПСЕ». Получатель сообщения «ГДУГПСЕ» ищет эти буквы в нижней строке и по буквам над ними восстанавливает исходное сообщение «АБРАМОВ».

Ключом в шифре Цезаря является величина сдвига нижней строки алфавита. Количество ключей для всех модификаций данного шифра применительно к алфавиту русского языка равно 33.

Любой метод вскрытия шифра простой однобуквенной замены основан на том обстоятельстве, что частотные характеристики m-грамм шифр-текста и открытого

текста одинаковы. При этом существенно используются априорные частотные характеристики предполагаемого открытого текста, получаемые с учетом «характера переписки». Чем менее рельефно распределение знаков текста, тем сложнее задача вскрытия шифра простой замены. Для открытых текстов с «почти равномерным» распределением знаков эта задача становится практически не решаемой.

При проверке гипотез о значениях шифробозначений полезен поиск в шифртексте слов с характерной структурой, которые часто встречаются в открытом тексте. Для русского языка – это, например, слова сколько, которое, что и т.п. Для английского языка – слова that, every, the , look и т.п. Такие слова выделяются в шифртексте посредством интервалов между повторяющимися частыми буквами, характерными сочетаниями гласных и согласных.

Задача дешифрования еще более упрощается, если известно, что использовался сдвиговый или аффинный шифр. Так, для аффинного шифра бывает достаточно идентифицировать лишь пару шифробозначений с тем, чтобы полностью восстановить открытый текст.

1.3 Шифры гаммирования

Шифры гаммирования – суть метода заключается в следующем. С помощью секретного ключа k генерируется последовательность символов $g=g_1g_2\ldots g_i\ldots$, эта последовательность называется гаммой. При шифровании гамма накладывается на открытый текст $t=t_1t_2\ldots t_i\ldots$, т.е. символы шифртекста получаются из соответствующих символов открытого текста и гаммы с помощью некоторой обратимой операции $c_i=t_i*g_i$, $i=1,2,\ldots$ В качестве обратимой операции обычно используется либо сложение по модулю количества букв в алфавите N:

(1)

либо, при представлении символов открытого текста в виде двоичного кода, операция поразрядного суммирования по модулю 2 (операция ‘побитовый XOR’)

, (2)

Расшифрование осуществляется применением к символам шифртекста и гаммы обратной операции: или .

По традиции можно выделить два основных этапа.

1. получение длины гаммы.
2. получение значения гаммы.

Первая часть задачи решается с помощью параметра, называемого индексом совпадений. Он определяется как вероятность совпадения двух произвольных символов в строке. Считается по этой формуле.

(3)

где n – длина текста;

f_i – количество появлений в тексте i-го символа алфавита.

Мы будем перебирать длину гаммы и разбивать шифр на столбцы до тех пор, пока индекс совпадений первого столбца не будет максимально близок к 0.066 - это табличное значение индекса совпадений для литературного текста на английском языке.

На практике замечено, что если длина подходящая, то индекс первого столбца будет гарантированно превышать значение 0.053. А когда длина плохая, он будет колебаться от 0.03 до 0.044.

На втором этапе будет использоваться уже взаимный индекс совпадений двух столбцов. Он считается по такой формуле.

(4)

где f_i – количество появлений i-го символа в первом столбце;

g_i – во втором;

n – длина первого столбца;

n' - длина второго.

В нашем случае n = n`, столбцы одинаковой длины.

Мы разбиваем код на столбцы по длине гаммы, которую нашли на прошлом этапе, и перебираем все их пары, начиная с первого. У второго столбца в паре перебираем все сдвиги (от 1 до 26) и считаем для каждого взаимный индекс совпадений. В тот момент, когда индекс будет близок к 0.066, запоминаем смещение и переходим к следующей паре столбцов.

После этой процедуры мы получим относительные сдвиги второго столбца относительно первого, третьего относительно второго, четвертого относительно третьего и так далее для всех столбиков.

Следующим шагом будет вычисление правильного сдвига для первого столбца, чтобы затем применить все относительные сдвиги и получить наконец открытый текст. Сдвиг будем искать с помощью частотного анализа, в точности как в шифре Цезаря. После того, как мы нашли сдвиг первого столбика, сдвигаем все столбцы относительно своих соседей и получаем значение гаммы. На этом описание алгоритма закончено.

1.4 Искусственные нейронные сети и их составляющие

Искусственные нейронные сети были построены по принципу биологических нейронных сетей, которые представляют собой сети нервных клеток, выполняющие определенные физиологические функции. Составным элементом нейронных сетей являются нейроны (рисунок 6).

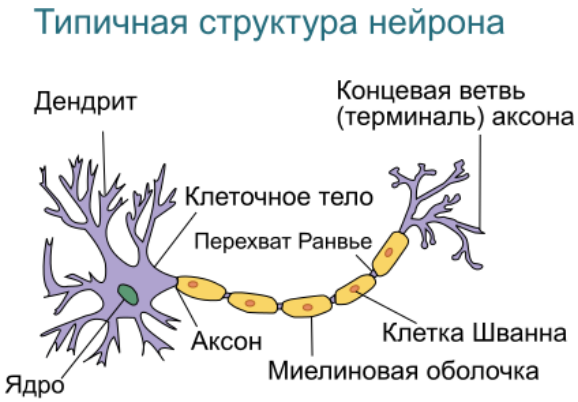

Рисунок 6 – Типичная структура нейрона

У нейрона есть несколько функций:

1) Приемная функция: синапсы получают информацию;

2) Интегративная функция: на выходе нейрона сигнал, который несет информацию обо всех суммированных в нейроне сигналах;

3) Проводниковая функция: по аксону проходит информация к синапсу;

4) Передающая функция: импульс, достигший окончания аксона, заставляет медиатор передавать возбуждение следующему нейрону.

Синапсами называют связи, по которым выходные сигналы одних нейронов поступают на входы других. Каждая связь характеризуется своим весом. Связи с положительным весом называются возбуждающими, а с отрицательным тормозящими. Выход нейрона называется аксоном. В искусственной нейронной сети искусственный нейрон это некоторая нелинейная функция, аргументом которой является линейная комбинация всех входных сигналов. Такая функция называется активационной. Затем результат активационной функции посылается на выход нейрона. Объединяя такие нейроны с другими, получают искусственную нейронную сеть.

Активационная функция. Функция активации нейрона характеризует зависимость сигнала на выходе нейрона от суммы сигналов на его входах. Обычно функция является монотонно возрастающей и находится в области значений [-1,1] (гиперболический тангенс) и [0,1] (сигмоида). Для некоторых алгоритмов обучения необходимо, чтобы активационная функция была непрерывно дифференцируемой на всей числовой оси. Искусственный нейрон характеризуется своей активационной функцией (например, название «сигмоидальный нейрон»).

Основными активационными функциями являются:

1) Пороговая активационная функция (функция Хевисайда). Нельзя использовать для алгоритма обратного распространения ошибки;

(5)

,

2) Сигмоидальная активационная функция;

(6)

3) Гиперболический тангенс.

Перцептрон. Перцептрон – тип искусственного нейрона, разрабатываемый Фрэнком Розенблаттом в 1950-ых и 1960-ых годах. В современных работах чаще всего используют другую модель искусственного нейрона – сигмоидальный нейрон. Чтобы понять, как работает сигмоидальный нейрон, необходимо рассмотреть структуру и принцип работы перцептрона. Перцептрон принимает на вход значения x_1,x_2… и выдает бинарный результат (рисунок 7).

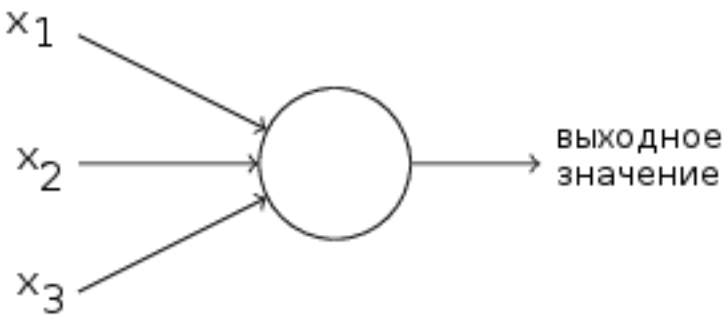

Рисунок 7 – Схема перцептрона

Розенблатт предложил использовать веса – числа, выражающие важность вклада каждого входа в конечный результат. Взвешенная сумма (или веса) сравнивается с пороговым значением (threshold), и по результатам определяется, будет ли выдан 0 или 1. Пороговое значение также является параметром нейрона.

Перцептроны могут быть классифицированы как искусственный нейронные сети

1) с одним слоем;

2) с пороговой активационной функцией;

3) с прямым распространением сигнала.

Обучение перцептрона состоит в изменении матрицы весов. Существуют 4 исторически сложившихся видов перцептронов:

1) перцептрон с одним скрытым слоем;

2) однослойный перцептрон: входные элементы напрямую соединены с выходными с помощью системы весов. является простейшей сетью прямого распространения (feedforward network);

3) многослойный перцептрон (по розенблатту): присутствуют дополнительные скрытые слои;

4) многослойный перцептрон (по румельхарту): присутствуют дополнительные скрытые слои, а обучение проводится по методу обратного распространения ошибки (backpropagation algorithm).

Если бы небольшое изменение весов (или смещения) вызывало небольшое же изменение на выходе сети, то желаемое поведение нейронной сети можно было бы получить с помощью простых модификаций смещений и весов в процессе обучения. Однако обучение не так просто осуществить, если нейронная сеть состоит из прецептронов. Небольшое изменение весов или смещения одного из перцептронов сети может кардинально изменить выходное значение перцептрона, например, с 0 на 1. Поэтому самое незначительное изменение значений одного из элементов сети может создать значительные трудности в понимании изменения поведения сети. Поскольку задача обучения нейронной сети является задачей поиска минимума функции ошибки в пространстве состояний обучения, то для ее решения могут применяться стандартные методы теории оптимизации. Для однослойного перцептрона с n входами и m выходами речь идет о поиске минимума в nm-мерном пространстве.

Сигмоидальный нейрон. Сигмоидальные нейроны похожи на перцептроны, однако небольшие изменения в их весах и смещениях незначительно изменят выход нейрона. Этот факт позволяет сети из сигмоидальных нейронов обучаться. На

вход сигмоидального нейрона подаются любые значения между 0 и 1. На выходе также выдается значение между 0 и 1, так как в качестве активационной функции используется сигмоида, являющаяся нелинейной (рисунок 8).

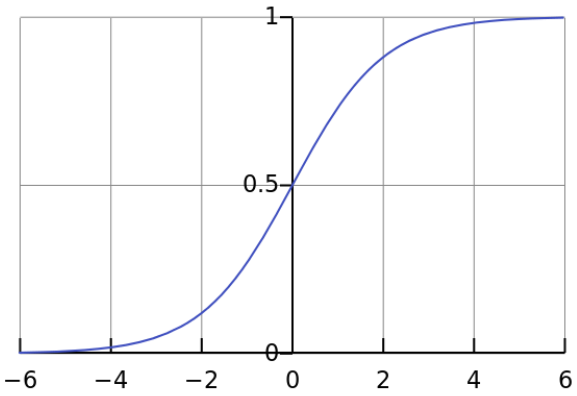

Рисунок 8 – График логистической кривой сигмоиды

Чем больше β (параметр наклона сигмоидальной функции активации), тем сильнее крутизна графика. При $\beta \to \infty$ сигмоида стремится к функции Хевисайда.

Важным свойством сигмоидальной функции является ее дифференцируемость. Применение непрерывной функции активации позволяет использовать при обучении градиентные методы.

Нейроны можно разделить на группы в зависимости от их положения в сети:

1) входные нейроны принимают исходный вектор данных;

2) в промежуточных нейронах происходят основные вычислительные операции – обучение;

3) выходные нейроны – результат работы сети.

Архитектура нейронных сетей. Рассмотрим задачу обучения с учителем. Дано множество тренировочных примеров X с метками (labels) Y. Нейронные сети определяют нелинейную гипотезу $h_{W,b}(x)$ с параметрами W и b. Нейронная сеть составлена из множества простых нейронов так, что выход одного из нейронов будет входом другого (см. рис.). Крайний левый слой называется входным, (тут

должен быть рисунок) а крайний правый слой – выходным. Слой посередине является скрытым и называется так из-за того, что его значения не наблюдаются в тренировочных примерах. Таким образом в данной сети три элемента входа, три скрытых элемента и один выходной элемент. Пусть n_l – количество слоев в сети (в данном случае 3). Параметры сети $(W,b)=(W^{(1)},b^{(1)},W^{(2)},b^{(2)})$ (рисунок 9).

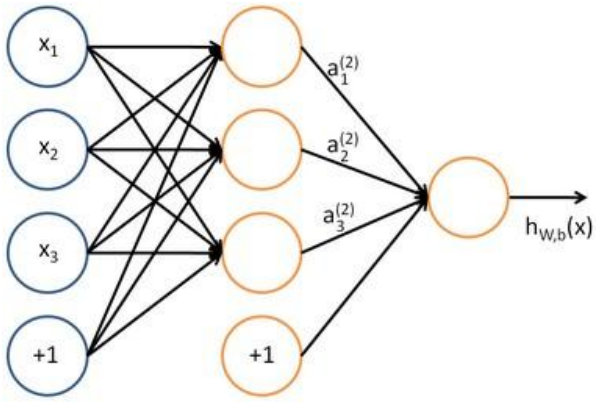

Рисунок 9 – Схема простейшей сети прямого распространения

Результат применения функции активации (выхода) обозначается a_i для i-ого элемента. Получаем такую систему:

Обозначив функцию суммирования через z, получим в векторной форме:

Общая формула будет выглядеть таким образом:

Сетью прямого распространения называются нейронные сети, которые используют выход одного слоя в качестве входных данных для следующего слоя.

1.5 Обучение нейронных сетей

1.5.1 Общие понятия в обучении нейронных сетей

Эпоха – прямой и обратный проход по всем тренировочным примерам.

Размер серии (batch) – количество тренировочных примеров для одной итерации прямого и обратного проходов.

Количество итераций – количество проходов: каждый проход использует примеры (batch). Один проход = прямой + обратный проход.

То есть имея 1000 примеров, batch = 500, нам потребуется две итерации, чтобы завершить одну эпоху.

С математической точки зрения, обучение нейронных сетей – многопараметрическая задача нелинейной оптимизации.

1.5.2 Алгоритм обратного распространения ошибок

Алгоритм обратного распространения ошибки определяет стратегию подбора весов многослойной сети с применением градиентных методов оптимизации. Поскольку целевая функция, обычно определяемая как квадратичная разность суммы между фактическими и ожидаемыми выходными значениями, является непрерывной, градиентные методы оптимизации являются эффективными при обучении сети. При обучении многослойной нейронной сети необходимо вычислить вектор градиента относительно параметров всех слоев сети. Пусть имеется конечный набор тренировочных данных (m примеров). Для обучения нейронной сети применяют пакетный градиентный спуск (batch gradient descent). Квадратичная ошибка целевой функции (squared-error cost function) для одного примера будет вычислена по формуле:

Тогда целевая функция для m примеров будет выглядеть так:

$$= \qquad (14)$$

Первый член выражения J(W,b) это сумма квадратов ошибок, второй член регуляризации (L2 – уменьшение весов – weight decay), позволяющий уменьшить значения весов и предотвратить переобучение. Параметр регуляризации весов λ используют для проверки относительной значимости частей данного выражения. В задачах бинарной классификации y представлен 0 или 1 (так как сигмоидная функция выдает значение в пределах [0;1]; однако при использовании гиперболического тангенса лейблами классов были бы -1 и 1). Задача – минимизировать J(W,b). Для обучения нейронной сети необходимо инициализировать каждый параметр и малыми случайными величинами, близкими к нулю (например случайное распределение $(0;e^2)$, где e = 0.01) а затем применить алгоритм оптимизации (упоминавшийся выше градиентный спуск).

Так как J(W,b) не является выпуклой функцией, то градиентный спуск восприимчив к локальным оптимумам. Каждая итерация градиентного спуска обновляет параметры таким образом:

где α – скорость обучения (learning rate).

Шаги алгоритма обратного распространения ошибки:

1) осуществляется прямой проход по сети, вычисляются активации слоев L2, L3 и так далее до выходного слоя L_{ni};
2) для каждого выходного элемента i в выходном слое n_l (the output layer) рассчитывается ошибка

$$\overline{\qquad}$$

3) для $l=n_l - 1, n_l - 2, n_l - 3, \dots, 2$:
   для каждого элемента в слое l, рассчитывается

4) вычисляются частные производные

$$\overline{\qquad}$$

В матричной форме алгоритм будет записан таким образом (• обозначено произведение Адамара – покомпонентное произведение двух матриц):

1) осуществляется прямой проход по сети, вычисляются активации слоев l2, l3 и так далее до выходного слоя l_{ni}.
2) матрица ошибок для выходного слоя n_l

3) для слоя $l = n_l - 1, n_l - 2, n_l - 3, \dots, 2$

4) вычисление частных производных

1.5.3 Псевдокод

1) (матрица или вектор нулей) для всех l;

2) For i = 1 to m

   – Использовать алгоритм обратного распространения ошибки, чтобы вычислить and .

   – .

   – .

3) Обновление параметров

—

—

Для обучения нейронной сети, нужно повторно выполнить шаги градиентного спуска, чтобы уменьшить значение целевой функции J(W,b).

1.5.4 Недостатки градиентного спуска

Основная трудность обучения нейронных сетей состоит в методах выхода из локальных минимумов. Недостатками градиентного спуска при обучении сети являются:

1) Паралич сети. Значения весов в результате коррекции могут стать очень большими величинами. Поскольку ошибка, посылаемая обратно в процессе обучения, пропорциональна производной сжимающей функции, то процесс обучения может почти остановиться. Это можно предотвратить, уменьшая шаг η, однако процесс обучения будет происходить дольше.

2) Размер шага. Если значение шага не изменяется и оно довольно мало, то метод сходится слишком медленно. Если же шаг слишком велик, то может возникнуть паралич сети. Необходимо изменять значение шага: увеличивать до тех пор, пока не прекратится улучшение оценки в направление антиградиента и уменьшать, если оценка не улучшается.

1.5.5 Сравнение стохастического и пакетного градиентных спусков

Если для пакетного градиентного спуска функция потер вычисляется для всех образцов вместе взятых после окончания эпохи, а потом изменяются весовые коэффициенты нейронов, то для стохастического метода весовые коэффициенты изменяются после вычисления входа сети на одном из обучающих примеров. Недостатком пакетного градиентного спуска является его “застревание” в локальных минимумах. Несмотря на то, что стохастический метод работает медленнее пакетного, он способен выходить из локальных минимумов, что приводит к лучшим результатам обучения сети (стохастический метод использует недовычисленный градиент).

1.6 Мониторинг состояния сети

1.6.1 Функция перекрестной энтропии в качестве целевой функции

Функция перекрестной энтропии используется в качестве функции потерь:

- предсказанные значения, y_i – верные значения.

–

1.6.2 Техники регуляризации

1. L1-регуляризация: происходит изменение нерегуляризованной целевой функции путем добавления суммы абсолютных значений весов:

–

2. При использовании L1-регуляризации происходит стремление одного или более весовых значений к 0.0, поэтому соответствующая функция (feature) больше не требуется. Этот эффект называется селекцией функций (feature selection);

3. L2-регуляризация (также известная как weight decay). В отличии от L1-регуляризации, в L2 веса уменьшаются на величину, пропорциональную весам:

—

4. Dropout не влияет на значение целевой функции: изменяется структура сети. Каждый нейрон удаляется из сети с некоторой вероятностью p. По полученной прореженной сети делается обратное распространение ошибки, для оставшихся весов делается градиентный шаг. После этого удаленные нейроны восстанавливают в сети. При обучении нейросети выход каждого нейрона домножается на (1-p). Так будет получено матожидание ответа сети по ее 2N (где N – количество нейронов в сети) архитектурам. Обученная таким образом сеть является результатом усреднения 2N сетей. Отдельная нейронная сеть, обученная при помощи раннего останова, имеет слишком большую ошибку, однако усреднение нескольких нейронных сетей приводит к существенному снижению ошибки;

5. Искусственное расширение данных для обучения.

Гиперпараметры сети

1. Темп обучения: сначала необходимо оценить пороговое значение для η, в котором значение целевой функции мгновенно начинает снижаться без колебаний. Сначала значение оценки устанавливается $\eta = 0.01$. Если значение целевой функции снижается во время первых эпох, то нужно увеличивать темп обучения, пока будет не найдено значение колебания целевой функции. Если же при начальном темпе обучения значения целевой функции. Если же при начальном темпе обучения значения целевой функции колеблются, то необходимо его уменьшать. Темп обучения регулирует размер шага в градиентном спуске и наблюдает за значениями целевой функции, определяя, был ли размер шага градиентного спуска слишком большим;

2. Использовать раннюю остановку (early stopping) для определения размера эпох обучения: ранняя остановка значит, что в конце каждой эпохи нужно вычислить точность классификации на данных проверки (validation set). Когда улучшение точности прекратится, остановить процесс обучения. Такая остановка также предотвращает переобучение;

3. График обучения: идея сохранить темп обучения неизменным до момента, когда точность данных проверки не начнет ухудшаться. Тогда необходимо уменьшить темп обучения (уменьшив, например, в 10 раз);

4. Параметр регуляризации λ: после определения η, можно начать с $\lambda=1.0$ и затем увеличивать или уменьшать значение (в 10 раз);

5. Размер пакетов: если размер пакетов слишком мал, невозможно полностью использовать преимущества хороших матричных библиотек, оптимизированных для быстрого оборудования. Если же размер пакетов слишком велик, то веса сети будут обновлять очень нечасто. Необходимо выбрать компромиссное значение, которое максимизирует скорость обучения.

1.7 Глубокие нейронные сети

1.7.1 Обзор

Глубокими нейронными сетями называются такие сети, в которых есть несколько скрытых слоев. Поскольку каждый скрытый слой высиляет нелинейное преобразование предыдущего слоя, глубокая сеть может иметь значительно большую репрезентативную мощность (то есть может представлять значительно более сложные функции), чем малослойная. При обучении глубокой сети важно использовать нелинейную функцию активации в каждом скрытом слое. Это связано с тем, что множество слоев линейных функций сами вычисляли бы только линейную функцию ввода и, следовательно, не были бы более выразительными, чем использование только одного скрытого слоя.

Главным достоинством глубинных сетей является сжатое преставление достаточно большого множества функций. Можно показать, что существуют функции, которые k-слойная сеть может предсатвлять сжато, а (k-1)-слойная сеть не может этого сделать, если только она не имеет экспоненциально большое количество элементов в скрытых слоях.

1.7.2 Доступность данных

С помощью метода, описанного выше, можно полагаться только на маркированные данные для обучения. Однако помеченных данных часто бывает недоста-

точно, и, следовательно, для многих задач трудно получить достаточное количество примеров для соответствия параметрам сложной модели. Например, учитывая высокую степень выразительности глубинных сетей, обучение при небольшом количестве данных приведет к переобучению.

1.7.3 Локальный оптимум

Обучение малослойной сети (с 1 скрытым слоем) с использованием контролируемого обучения обычно приводит к сближению параметров с подходящими значениями. Но при обучении глубокой сети, это работает намного реже. В частности, обучение нейронной сети с использованием обучения с учителем включает в себя решение проблемы с невыпуклой оптимизацией (например, минимизация ошибки обучения (формула) в зависимости от сетевых параметров W). В глубокой сети появляется большое количество локальных оптимумов, поэтому обучение с градиентным спуском перестает работать.

1.7.4 Градиентная диффузия

При использовании метода обратного распространения ошибки для вычисления производных, градиенты, которые распространяются от выходного слоя до более ранних слоев сети, быстро уменьшаются по мере увеличения глубины сети. В результате производная от общей стоимости по отношению к весам в более ранних слоях очень мала. Таким образом, при использовании градиентного спуска веса ранних слоев медленно меняются и более ранние слои не могут многому научиться. Эту проблему часто называют “диффузией градиентов” (diffusion of gradients).

1.8 Проблемы обучения глубоких сетей и их решения

1.8.1 Исчезающий градиент

Проблема исчезающего градиента – это трудность, возникающая при обучении искусственных нейронных сетей с использованием методов обучения на основе градиента и обратного распространения ошибки. В таких методах каждый вес нейронной сети обновляется пропорционально градиенту функции ошибки относительно текущего веса на каждой итерации обучения. Стандартные функции ак-

тивации, такие как гиперболический тангенс, имеют градиенты в диапазоне (-1,1), а метод обратного распространения ошибки вычисляет их по цепному правилу. После умножения этих чисел для вычисления градиентов “фронтальных” слоев в n-слойной сети, что означает, что градиент (сигнал ошибки) экспоненциально уменьшается вместе с n, а передние слои обучаются очень медленно.

Когда используются функции активации, производные которых могут принимать большие значения, есть риск столкнуться с exploding gradient problem. Возможными решениями являются:

1. Многоуровневая иерархия: слой сети предварительно обучается, используя методы обучения без учителя, а затем его значение регулируется с помощью метода обратного распространения ошибки. Таким образом каждый слой сети изучает сжатое представление наблюдений, которое подается на следующий слой;
2. Остаточные сети (Residual networks): один из наиболее эффективных методов решения проблемы исчезающего градиента является использование остаточных нейронных сетей (ResNets). Более глубокая сеть будет иметь более высокую ошибку обучения, чем малослойная сеть. Команда Microsoft Research обнаружила, что разделение глубокой сети на части (скажем, каждая часть представляет собой три слоя сети) и передача входных данных в каждый фрагмент до следующего фрагмента (наряду с остаточным выходом из куска минус входные данные вновь введенного фрагмента) помогли устранить большую часть этой проблемы с исчезновением градиента. Никаких дополнительных параметров или изменений в алгоритме обучения не требуется. ResNets показали более низкую ошибку обучения (и тестовую ошибку), чем их более малослойные аналоги, путем повторного ввода выходов из более мелких слоев в сети для компенсации исчезающий данных.

1.8.2 Сигмоидальные активационные функции

Использование сигмоидальных активационных функций может вызвать проблемы в обучении глубинных сетей, а именно значения активаций в конечном слое будут близки к нулю на ранних этапах обучения, замедляя этот процесс. Бы-

ли предложены альтернативные активационные функции, которые не так страдают от ограничения.

1.8.3 Выбор подходящих весов

Выбор подходящих весов и momentum schedule в импульсном стохастическом градиентном спуске (momentum-based stochastic gradient descent) существенно влияют на способность обучать глубокие сети.

1.9 Сверточные нейронные сети

1.9.1 Обзор

Свертка является операцией, которая применяется к двум последовательностям f и g b порождает третью последовательность

Формула для двумерной свёртки:

Рассмотрим одномерный свёрточный слой с входами xn и выходами yn (см. рис.). Тогда функция для выходов будет представлена следующим образом:

В свёрточном слое находится множества копий одного и того же нейрона (рисунок 10).

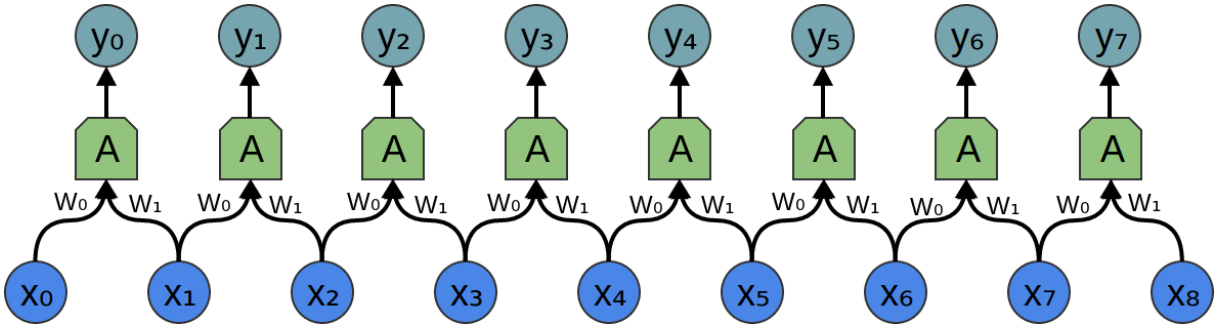

Рисунок 10 – Пример одномерного сверточного слоя

Поэтому многие веса появляются в нескольких позициях.

Стандартная матрица весов соединяет каждый вход с каждым нейроном с разными весами. Матрица для свёрточного слоя отличается тем, что различные веса могут появляться на нескольких позициях, а поскольку нейроны не соединены со всеми возможными входами, матрица содержит множество нулевых элементов:

То есть умножение на матрицу выше — то же самое, что и свёртка с $[\ldots 0, w_1, w_0, 0 \ldots]$. Ядро свёртки, скользящее по разным частям изображения, соответствует наличию нейронов в этих частях.

Свёртку можно пояснить на примере обработки изображений. Если представить, что изображения – двумерные функции, то различные преобразования изображений не что иное, как свёртка функции изображения с локальной функцией, которая называется ядром свёртки.

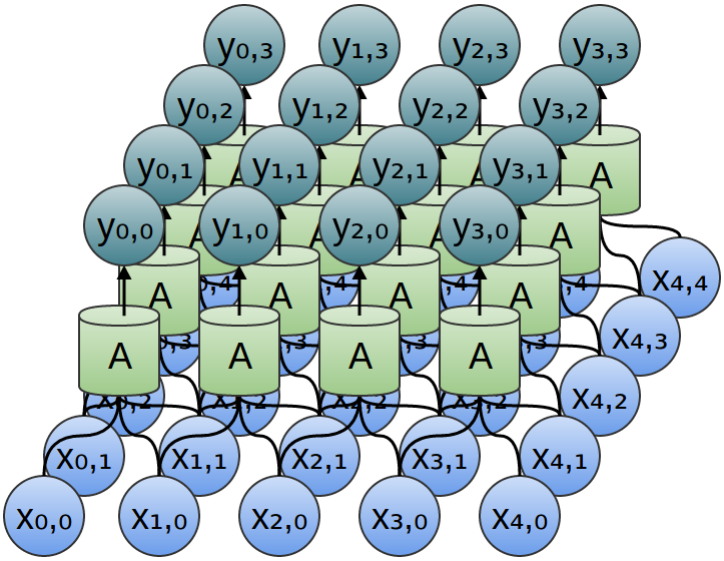

Рисунок 11 – Двумерный сверточный слой

Каждый новый пиксель изображения представляет собой взвешенную сумму пикселей, которые ядро прошло к этому моменту времени. Двумерный свёрточный слой представлен на рис 11.

Свёрточная нейронная сеть — архитектура нейронных сетей, изначально созданная и использованная для эффективного распознавания изображений: чередуются свёрточные слои (convolutions) с нелинейными активационными функциями (ReLU или гиперболический тангенс tanh) и слои объединения (pooling layers).

В отличии от сети прямого распространения, где каждый входной нейрон соединяется с выходным нейроном в следующем слое, в свёрточных сетях для получения выходных значений используются свёртки над каждым входным слоем. В операции свёртки используется матрица весов небольшого размера, которая сдвигается по всему обрабатываемому слою, формируя после каждого сдвига сигнал активации для нейрона следующего слоя с аналогичной позицией. Эта матрица называется ядром свёртки; она используется для различных нейронов выходного слоя.

При выполнении операции свёртки каждый фрагмент (например, изображения) поэлементно умножается на матрицу свёртки, а результат суммируется и записывается в аналогичную позицию выходного изображения. Матрицу свёртки представляет собой графическое кодирование какого- либо признака. Получившийся в результате операции свёртки следующий слой показывает наличие данного признака. В свёрточной нейронной сети существуют много наборов весов, которые кодируют элементы изображений. Ядра свёртки формируются в процессе обучения сети. При проходе каждым набором весов формируется карта признаков. Поскольку появляется много независимых карт признаков на одном слое, то сеть становится многоканальной.

В каждом слое свёртки для каждого канала свой фильтр, ядро свёртки которого обрабатывает предыдущий слой по фрагментам. Результат применения различных фильтров объединяется. Так получаются слои объединения. Операция субдискретизации выполняет уменьшение размерности сформированных картпризна-

ков. В данной архитектуре сети считается, что информация о факте наличия искомого признака важнее точного знания его координат, поэтому из нескольких соседних нейронов карты признаков выбирается максимальный и принимается за один нейрон уплотнённой карты признаков меньшей размерности. За счёт данной операции, помимо ускорения дальнейших вычислений, сеть становится инвариантной к масштабу входного изображения.

После начального слоя сигнал проходит серию свёрточных слоёв, в которых чередуется операции свёртки и объединения(pooling). Чередование слоёв позволяет составлять карты признаков: на каждом следующем слое карта уменьшается в размере, а количество каналов увеличивается. Практически это означает способность распознавания сложных иерархий признаков.

После прохождения нескольких слоев карта признаков вырождается в вектор или скаляр, но таких карт признаков становится сотни. На выходе свёрточных слоёв сети дополнительно устанавливают несколько слоев полносвязной нейронной сети (например, перцептрон), на вход которому подаются конечные карты признаков.

1.9.2 Гиперпараметры сети

Гиперапараметрами свёрточной нейронной сети являются:

1. Узкая и широкая свёртки (wide and narrow convolutions): дополнение нулями (zero padding) позволяет сделать свёртку широкой в случае, когда, например, первый элемент матрицы не имеет соседних элементов слева и сверху. Без использования дополнения нулями получаем узкую свёртку;

2. Размер шага (stride): Размер шага определяет величину сдвига фильтра на каждом шаге. Чем больше шаг, тем меньше фильтр применяется и тем меньше размер выходной матрицы. Обычно использует шаг равный единице, однако больший шаг может позволить построить модель, поведение которой будет напоминать рекурсивную нейронную сеть (т.е. свёрточная сеть с большим шагом будет выглядеть как дерево);

3. Слои объединения: Слои объединения помогают сократить размерность

выходной информации, при этом сохраняя самую заметную информацию. Например, если фильтр определяет, содержит ли предложение отрицание ("not good"). Если где-то в предложении есть эта фраза, то результат применения фильтра к этому региону даст большое значение, но малое для других регионов. После применения операции максимума для региона, остается только информация, появлялось ли заявленное отрицание в предложение, однако информация о том, где оно появлялось, исчезает. То есть информация о местоположении пропадает, а локальная информация остаётся (очевидно, что “not good” сильно отличается от “good not”);

4. Каналы (channels): каналы — это разные “взгляды” на входные данные. Например, в распознавании изображений, у нас обычно три канала — RGB. В обработке естественного языка такими каналами могут являться различные векторные представления слов (word2vec или GloVe), предложение на разных языках или перефразированные предложения.

1.9.3 Типовая структура

Структура сети является однонаправленной, а для обучения обычно используется метод обратного распространения ошибки. Сеть состоит из большого количества слоёв (рисунок 12).

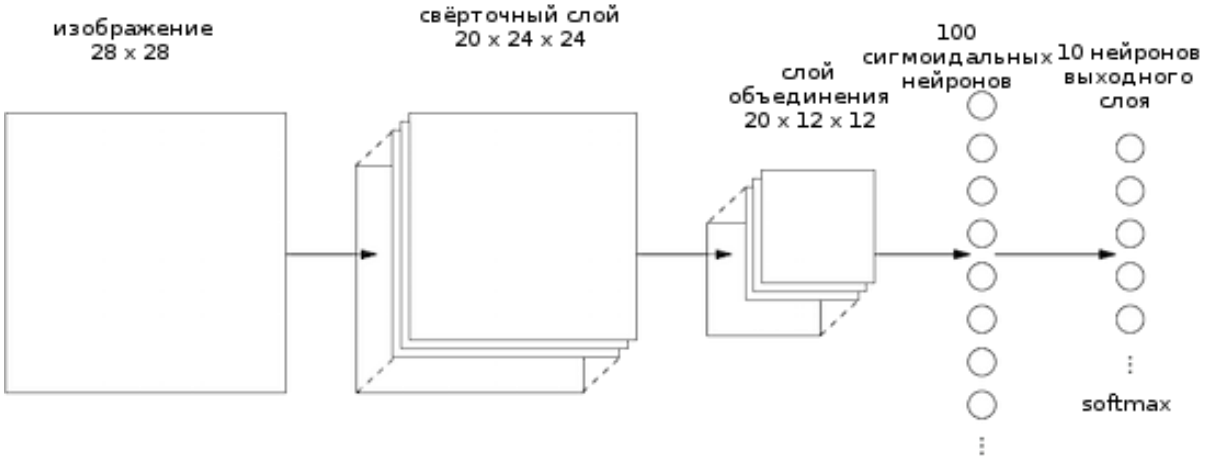

Рисунок 12 – Пример архитектуры сверточной нейронной сети

1.9.4 Слой свёртки (convolutional layer)

Основной компонент свёрточной нейронной сети. Слой свёртки включает в себя свой фильтр для каждого канала. Ядро свёртки фильтра обрабатывает предыдущий слой по фрагментам. Весовые коэффициенты ядра свёртки определяются в процессе обучения сети.

1.9.5 Усеченное линейное преобразование (rectifted linear unit)

Также переводимый как блок линейной ректификации. Слой ReLU представляет является функцией активации после свёрточного слоя. Функция f (x) = max(0, x) выбирается вместо сигмоиды или гиперболического тангенса, поскольку показывает хорошие результаты при обучении нейронных сетей и “отсекает” ненужные детали в канале. Ректификатор, приближение ректификтора, также называемый softplus, и производная softplus логистическая функция.

1.9.6 Пулинг или слой объединения

Слой пулинга – это нелинейное уплотнение карты признаков. Использование пулинга позволяет существенно уменьшить пространственный объём изображения. Операция объединения интерпретируется следующим образом: если на предыдущей операции свёртки уже были выявлены некоторые признаки, то для дальнейшей обработки изображение уплотняется до менее подробного. Фильтрация ненужных деталей помогает сети не переобучаться.

Обычно слой пулинга вставляется после слоя свёртки перед слоем следующей свёртки: для реализации обычно используется функция максимума.

Можно также применить L2-pooling: вместо выбора максимальной активации из карты признаков, нужно взять квадратный корень от суммы квадратов активаций этой карты признаков.

1.9.7 Полносвязная нейронная сеть

После нескольких свёрток и уплотнения изображения с помощью пулинга из конкретной сетки пикселей с высоким разрешением получаются более абстрактные карты признаков: на каждом следующем слое увеличивается число каналов, при этом размерность изображения в каждом канале уменьшается. Таким образом остаётся большой набор каналов, в которых хранится небольшое число данных. Эти данные интерпретируются как абстрактные признаки, выявленные из исходного изображения. Затем их объединяют и передают на обычную полносвязную нейронную сеть, которая также может быть многослойной. Поскольку полносвязные слои уже не соответствуют пространственной структуре пикселей, они обладают относительно небольшой размерностью.

1.10 Использование сверточных нейронных сете в анализе текстов

Посимвольный подход для классификации текстов с помощью сверточных нейронных сетей был предложен в статье [13]. Опишем данный метод подробнее.

Назовем алфавитом упорядоченный набор символов. Пусть выбранный алфавит состоит из m символов. Каждый символ алфавита в тексте закодирован с помощью 1 – m – кодировки. (т.е. каждому символу будет сопоставлен вектор длины m элемент которого равен единице, в позиции равной порядковому номеру символа в алфавите, а нулю во всех остальных позициях). Если в тексте встретится символ, который не вошел в алфавит, то необходимо закодировать его вектором длины m состоящим из одних нулей. Из текста выбираются первые l символов. Параметр l должен быть большим, чтобы в первых l символах содержалось достаточно информации для определения класса всего текста.

(рисунок)

Далее полученные векторы составляются в матрицу размера m × l, в которой каждый столбец будет иметь не более одной единицы. Каждая строка полученной

матрицы используется как отдельная карта признаков. На вход сверточной нейронной сети подается m карт признаков размера 1×l аналогично изображению. Архитектуру сети необходимо выбирать исходя из задачи. На рис. приведен пример посимвольного подхода для l = 6, m = 3. В примере показан одни сверточный и субдискретизирующий слой. Опишем формально данный подход.

Пусть xi – вектор i-го символа в тексте.

Здесь ⊕ операция объединения векторов.

Сверточный слой:

где f – функция активации нейронной сети;
b – константа.

MAX-pooling слой:

Dropout слой:

где – посимвольное умножение;
r – вектор состоящий из нулей и единиц.

В статье [13] были приведены эксперименты, которые показали, что описанный подход с высокой точностью классифицирует тексты, по сравнению с большинством других известных на данный момент методов классификации текстов, если размер выборки достаточно велики. Так на выборке размером 1400000 объектов сверточная нейронная сеть с посимвольным подходом дала качество классификации по метрике accuracy – 0.712, а методом Bag of words удалось достичь лишь – 0.689.

Выводы

В данном разделе были рассмотрены шифры и нейронные сети.

Из шифров были рассмотрены: шифр простой замены, шифры перестановки и шифры гаммирования. Было дано краткое их описание, принцип по которому они зашифровывают текст. Также был рассмотрен криптоанализ каждого шифра и выделены признаки по которым в дальнейшем программа будет их распознавать.

Для написания программы распознавания были рассмотрены нейросети, в частности сверточная нейронная сеть с помощью которой зашифрованные тексты будут анализироваться.

2 ОПИСАНИЕ РАЗРАБОТКИ

2.1 Создание базы данных зашифрованных текстов

Для создания базы данных зашифрованных текстов были написаны 3 функции на языке Python. Результаты работы функций не записываются в отдельный файл, а сразу подаются на вход нейронной сети.

Перед тем как зашифровать текст, он проходит предобработку. Из текста удаляются знаки препинания, прописные буквы приводятся к строчным. Открытый текст состоит из строчных букв русского алфавита и символа пробела:

Открытым текстом для шифрования были выбраны 3 книги большого содержания. На каждый текст обучающей выборки брались по 1000 символов и подавались на вход нейронной сети.

2.1.1 Функция для шифра перестановки

Если текст не делится на длину перестановки, его нужно дополнить. Иногда дополняют нулями, но так как это визуально просматривается, текст дополнялся случайными символами из алфавита.

На вход функция получает текст в виде массива символов и саму перестановку. Для перестановки воспользуемся обычным массивом. Пусть значение каждого индекса будет позицией, в которую перейдет символ. Например, перестановка (012) в виде массива это [1,2,0] (реализация в приложении В).

2.1.2 Функция для шифра простой замены

Каждый символ открытого текста должен быть заменен на соответствующий ему символ из специальной таблицы. Рассматриваться будет случай, когда символы алфавита заменяются на символы из этого же алфавита, но в произвольном порядке. Ключом шифрования будет служить перемешанный алфавит.

Реализованная функция принимает на вход открытый текст и ключ. Оба в виде массива (list) символов. Еще есть алфавит (правильный). Функция бежит по символам и формирует шифротекст, согласно ключу (реализация в приложении Д).

2.1.3 Функция шифра гаммирования

Для того, чтобы получить зашифрованный текст достаточно сложить каждый символ открытого текста с символом гаммы. В качестве гаммы будет выступать символьная последовательность произвольной длины. В случае, если ее длина меньше длины текста, мы просто повторим последовательность нужное количество раз, чтобы хватило на зашифровку всего текста.

Так как шифроваться будет не двоичная последовательность, а текст на русском языке, то и сложение будет выполняться не по модулю 2, а по модулю 33.

На вход поступает открытый текст без пробелов, в виде массива символов, и ключ – гамма. Анализируется длина текста, «растягивается» гамма до нужного размера и выполняем посимвольное сложение (реализация в приложении Б).

2.2 Библиотека TensorFlow

2.2.1 Обзор

TensorFlow - это библиотека программного обеспечения с открытым исходным кодом для задач машинного обучения, разработанная Google. Она позволяет создавать и обучать нейронные сети различной архитектуры для обнаружения и распознавания образцов и поиска взаимосвязей. TensorFlow также включает в себя TensorBoard, который представляет собой средство визуализации в браузере для оценки эффективности обучения и сетевых параметров модели.

TensorFlow достигает своей производительности благодаря распараллеливанию задач между центральным и графическими процессорами. Ядро каждой операции реализовано на C++ с использованием библиотек Eigen и cuDNN для лучшей производительности.

Каждое вычисление в TensorFlow представляется как граф потока данных, он же граф вычислений. Граф вычислений является моделью, описывающей как будут выполняться вычисления. Важно заметить, что составление графа вычислений и выполнение операций в заданной структуре — два разных процесса. Граф состоит из плэйсхолдеров (tf.Placeholder), переменных (tf.Variable) и операций. В нём про-

изводится вычисление тензоров — многомерных массивов, которые могут быть числом или вектором.

Графы выполняются в сессиях (tf.Session). Существуют два типа сессий — обычные и интерактивные (tf.InteractiveSession); интерактивная сессия подходит для выполнения в консоли. Сессия хранит состояние переменных (Variables) и очередей (queues). Явное создание сессий и графов гарантирует надлежащее освобождение ресурсов памяти [15]

В графе каждая вершина имеет 0 или больше входов и 0 или больше выходов, и представляет собой реализацию операции. Тензоры представляют собой рёбра графа, а именно массивы произвольного размера (тип массива указывается во время построения графа). Особые вершины, управляющие зависимости (control dependencies), также могут быть в графе: они указывают, что исходный узел для контрольной зависимости должен закончить выполнение до того, как узел получателя контрольной зависимости начнет выполняться.

Каждая операция имеет название и представляет собой абстрактное вычисление (например, суммирование). У операции могут быть атрибуты: например, возможность сделать операцию полиморфной для разных типов тензоров. Ядро — специфическая реализация операции, которая может выполнена на определенном типе устройства (центральный или графический процессор).

Переменная — особый вид операции, возвращающий указатель на постоянно меняющийся тензор: такая переменная не исчезает после единичного использования графа. Указатели на подобные тензоры передаются многочисленным операциям, которые затем изменяют указанный тензор. В задачах машинного обучения, параметры модели обычно хранят тензоры в переменных, которые обновляются на каждом шаге обучения.

Данная работа выполнена на единственном устройстве с использованием CPU.

2.2.2 Примеры

Пусть стоит задача классифицировать цветы 3-х видов, используя ширину и длину их чашелистника. Для решения задачи будет использована однослойная

нейронная сеть со смещением b и двухэлементным вектором весов W для получения двухэлементного входного вектора X. В качестве активационной функции использована сигмоидальная функция. В обучении использован градиентный спуск, а в качестве функции ошибок – кроссэнтропия (cross entropy loss).

где z – фактический класс.

Сначала необходимо явно определить переменные, которые будут использованы в графе вычислений [16] (рисунок 13):

```
import tensorflow as tf
...
W = tf.Variable(tf.random_normal([2, 1], stddev=.01), name="W")
b = tf.Variable(tf.random_normal([1], stddev=.01), name="b")
X = tf.placeholder(tf.float32, [None, 2], name="X")
Z = tf.placeholder(tf.float32, name="Z")
```

Рисунок 13 – Фрагмент кода для определения переменных в графе

Затем определить активационную функцию (рисунок 14):

```
Y = tf.nn.sigmoid(tf.matmul(X, W) + b)
```

Рисунок 14 – Пример кода для определения активационной функции

Скалярное произведение и функция потерь (рисунок 15):

```
tf.scalar_product = lambda a, b: tf.matrix_determinant(\
        tf.matmul(tf.expand_dims(a, 1), b, transpose_a=True))

E = - (tf.scalar_product(Z, tf.log(Y)) + \
        tf.scalar_product(1 - Z, tf.log(1 - Y)))
```

Рисунок 15 – Фрагмент кода для определения скалярного произведения и функции потерь

Функция оптимизации (градиентный спуск) (рисунок 16):

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate).
   minimize(E)
```

Рисунок 16 – Фрагмент кода для определения функции оптимизации

Обучение сети (рисунок 17):

```
sess = tf.Session()
sess.run(tf.initialize_all_variables())
for _ in range(epochs):
        sess.run(optimizer, feed_dict={X: data.X, Z: data.Z})
sess.close()
```

Рисунок 17 – Фрагмент кода для обучения сети

2.2.3 Особенности

В Tensorflow существуют несколько форм параллелизма:

1. Параллелизм в отдельных операциях (например, tf.nn.conv2d () и tf.matmul ()). Эти операции имеют эффективные параллельные реализации для многоядерных процессоров и графических процессоров, и TensorFlow использует эти реализации во всех возможных случаях;

2. Параллелизм между операциями. TensorFlow использует представление графа вычислений и там, где есть два узла, которые не связаны прямым путем, они могут выполняться параллельно;

3. Параллелизм между копиями моделей. Стандартная схема для параллельного обучения – разделить данные между workers, провести одинаковые вычисления для разных данных и обменяться параметрами модели между workers.

Уникальность Tensorflow заключается в возможности проводить частичные подграфовые вычисления. Эта особенность позволяет сделать разбиение нейронной сети, а значит можно использовать распределенное обучение. Также

1) Tensorboard: визуализация модели и возможность исследовать порядок вычислений в графе;

2) Tensorflow может использована как на мобильных, так и на более мощных устройствах.

В Tensorflow производные задаются автоматически: этот процесс называется автоматическим дифференцированием.

Традиционные методы оценки производной сложно реализовать на практике, так как они имеют ряд недостатков. Например, использование метода конечных разностей требует обоснованного выбора значения приращения аргумента. Однако существует способ автоматического вычисления вместе с функцией f(x0) её производной fj(x0) при некотором значении аргумента x = x0. Данный метод называется автоматическим дифференцированием, так как вычисления значение f(x0) и fj(x0) осуществляется одновременно на основе исходного кода только функции f(x). Он позволяет получить точное (до ошибок округления) значения производной, а программу вычислений достаточно выполнить только один раз [17]. Tensorflow использует обратный режим автоматического дифференцирования для операций градиентов и метода конечных разностей для тестов, которые проверяют правильность работы градиента.

Обычно в системах автоматического дифференцифирования оператор (сумма, разность) определён вместе с его производными. То есть после написания функции, в которой определено несколько операторов, программа может сама выяснить, как вычислить соответствующие производные (используя граф вычислений и цепное правило). Выгода очевидна, так как не нужно самостоятельно разрабатывать математические операции и численно проверять каждую производную (рисунок 18).

```
tf.reset_default_graph()
x = tf.Variable(0.)
y = tf.square(x)
z = tf.gradients([y], [x])
```

Рисунок 18 – Фрагмент кода для демонстрации автоматического дифференцирования

2.3 Сверточная нейронная сеть

Архитектура сверточной сети представлена на рисунке 19.

2.3.1 Параметры сети

Параметрами сверточной нейронной сети являются:

1) Размерность векторного представления символа = 33;
2) Размер батча = 50;
3) Темп обучения = 0.001;
4) Количество шагов обучения = 2950;
5) Dropout = 0.8;
6) Размерность фильтров = (2, 3, 4);
7) L2 regularization = 4.

В качестве функции оптимизации используется Adam (Adaptive Moment Estimation). Его отличительными особенностями являются:

1) оценка первого момента вычисляется как скользящее среднее;
2) так как оценки первого и второго моментов инициализируются нулями, используется небольшая коррекция, чтобы результирующие оценки не были смещены к нулю.

Метод также инвариантен к масштабированию градиентов [18].

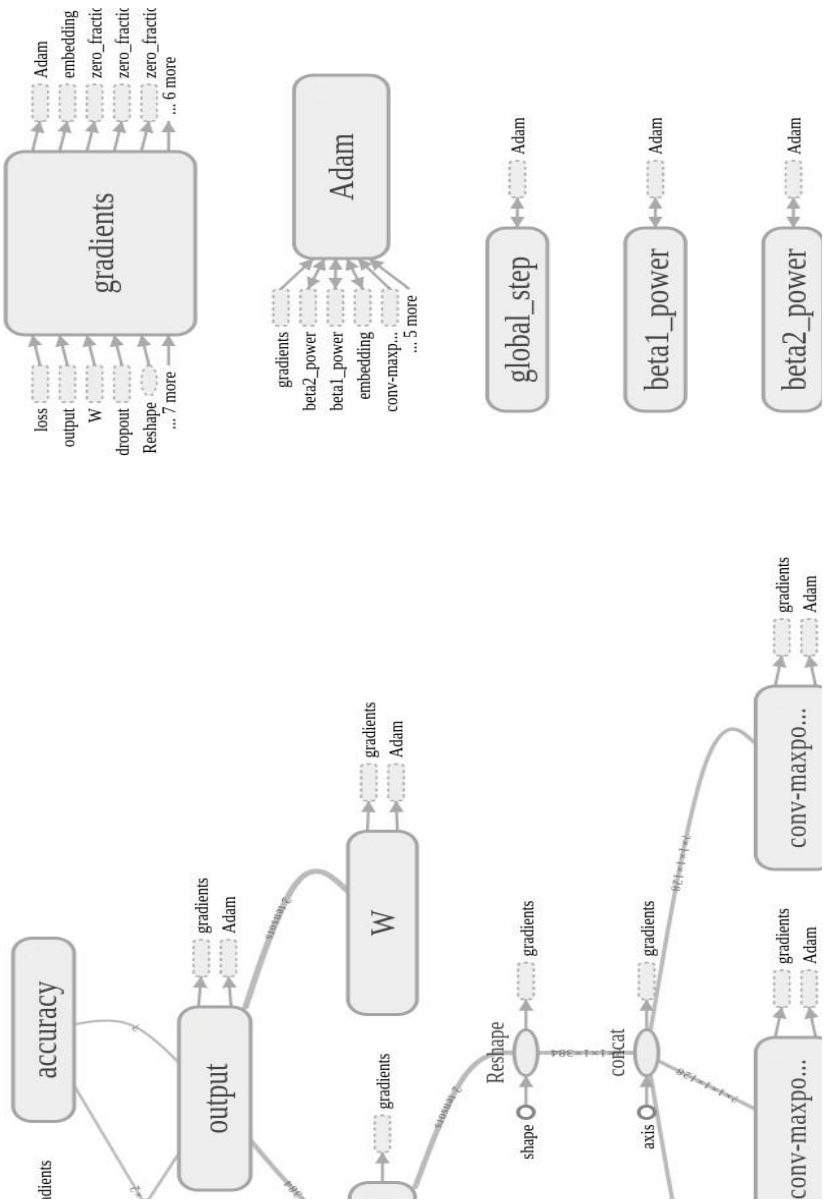

Рисунок 19 – Архитектура сверточной сети, представленная в tensorboard

2.4 Выводы

В начале главы были приведены описания функций, которые будут создавать примеры для тренировки и проверки нейронной сети.

В работе для написания нейронной сети была использована библиотека глубокого обучения Tensorflow разработанную компанией Google.

Так же была описана архитектура и модель разработанной нейронной сети. Были приведены ее параметры и модель обучения сети.

3 РЕЗУЛЬТАТЫ РАЗРАБОТКИ

Результаты эксперимента

Результаты классификации с помощью сверточной нейронной сети оценивались с помощью метрики accuracy, т. е. считалась доля верно классифицированных объектов к общему количеству объектов. Результаты представлены на рисунке 20, 21. Точность 0.799.

Графики точности модели и функции потерь представлены на рис. соответственно.

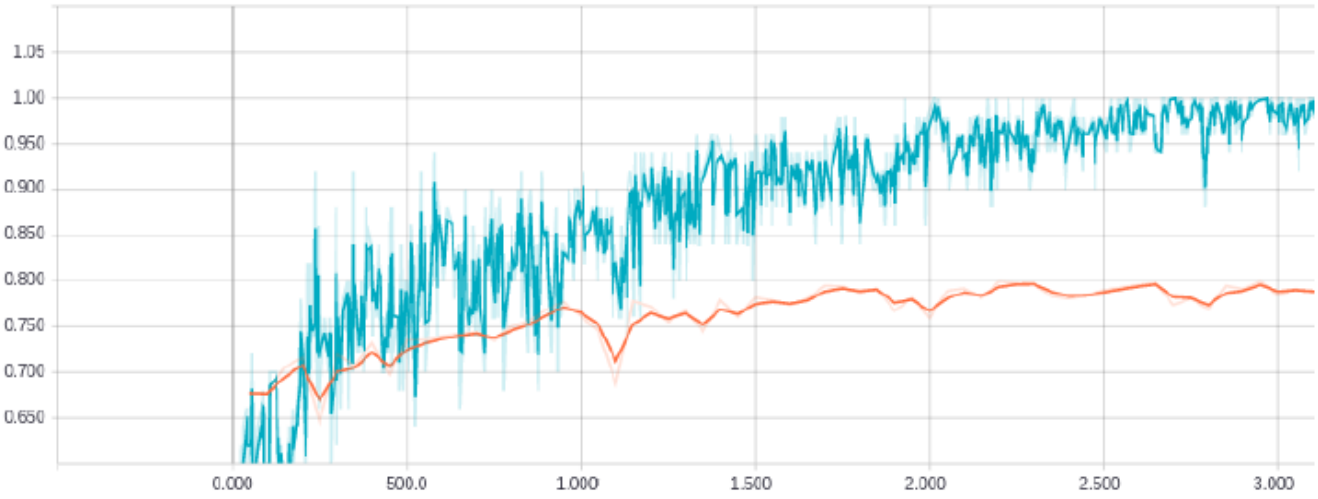

Рисунок 20 – Доля корректных прогнозов (accuracy): синий график – обучение, красный – проверка

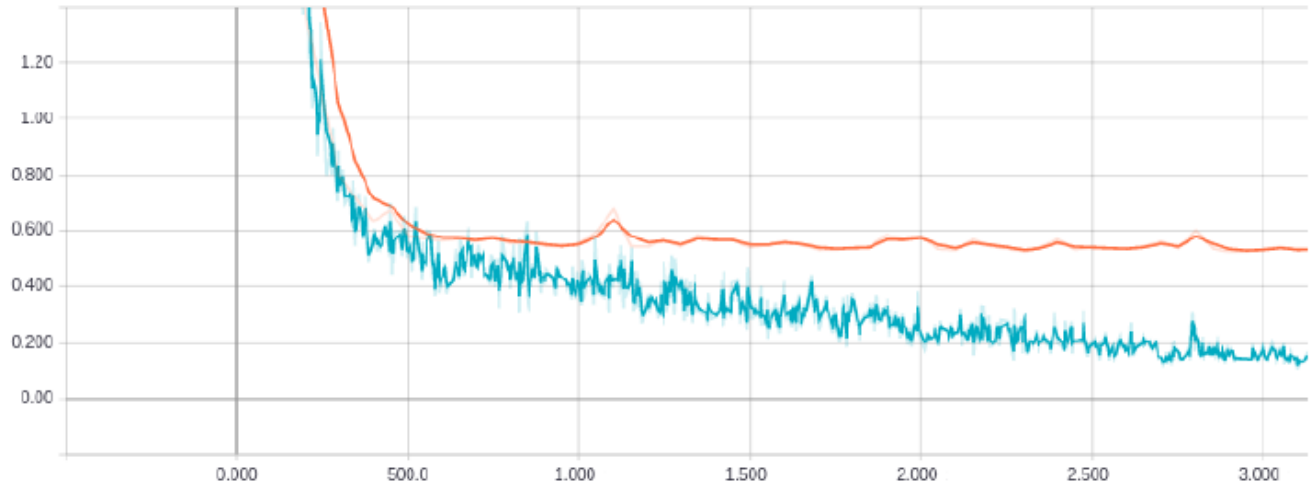

Рисунок 21 – Функция потерь: синий график – обучение, красный – проверка

3.1 Выводы

На основании результатов проведенных экспериментов реализованной сверточной нейронной сети с посимвольным подходом можно сделать следующие выводы:

1. Сверточные нейронные сети с посимвольным подходом – эффективный метод для классификации текстов. Наиболее важный вывод из проведенных экспериментов, что данный метод может классифицировать тексты с высокой точностью без использования слов. Это значи, что естественный язык можно рассматривать как сигнал.
2. Посимвольный подход позволяет решать задачу классификации текстов с нуля, т. е. не нужны никакие знания о синтаксической или семантической структуре языка, чтобы с хорошей точностью классифицировать текст.
3. На небольших наборах (до 1000) данных нейронная сеть показала очень плохие результаты. Когда данных становится больше сверточная нейронная сеть с посимвольным подходом работает лучше.

ЗАКЛЮЧЕНИЕ

В данной работе проводилось исследование по криптоанализу шифров с применением сверточной нейронной сети.

Нейронные сети, зарекомендовавшие себя, как мощный алгоритм для классификации изображений, в последнее время стали активно использоваться и для других задач машинного обучения.

В данной работы были рассмотрены 3 алгоритма шифрования текстов и их криптоанализ. Реализована сверточная нейронная сеть с посимвольным подходом, работа которой проверена на обучающей выборке текстов. Произведены подсчеты результатов работы нейронной сети. Показано что данная нейронная сеть с посимвольным подходом справляется со своей задачей на достаточно высоком уровне и вполне может быть использована в реальных задачах распознавания шифров.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. B. Pang and L. Lee. Opinion mining and sentiment analysis. Foundations and Trends in Information Retrieval archive, 2008.

2. Y. Kim. Convolutional neural networks for sentence classification. *arXiv:1408.5882 [cs.CL]*, 2014.

3. K. S. Tai et al. Improved semantic representations from tree-structured long short-term memory network. *arXiv:1503.00075 [cs.CL]*, 2015.

4. Q. Le and T. Mikolov. Distributed representations of sentences and documents. *arXiv:1405.4053 [cs.CL]*, 2014.

5. J. Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation 4(2). c. 234-242*, 1992.

6. X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Aistats 9, , c. 249-256*, 2010.

7. I. Sutskever et al. On the importance of initialization and momentum in deep learning. *ICML'13 Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, c. 1139- 1147*, 2013.

8. A. Karpathy. The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/. (дата обращения 25.05.2018).

9. R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. *ICML '08 Proceedings of the 25th international conference on Machine learning, c. 160-167*, 2008.

10. T. Mikolov et al. Distributed representations of words and phrases and their compositionality. *arXiv:1310.4546 [cs.CL]*, 2013.

11. T. Mikolov et al. Efficient estimation of word representations in vector space. *arXiv:1301.3781 [cs.CL]*, 2013.

12. Google Research Team. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv:1603.04467 [cs.DC]*, 2016.

13. M. Schrimpf. Should i use tensorflow? *arXiv:1611.08903 [cs.LG]*, 2016.

14. К. К. Семёнов. Автоматическое дифференцирование функций, выраженное программным кодом. *Вестник Саратовского государственного технического университета*, 2011.

15. В. Д. Чабаненко. Модификации метода стохастического градиентного спуска для задач машинного обучения с большими объемами данных. Master's thesis, Московский государственный университет имени М.В. Ломоносова, 2016.

16. J. Turian et al. Word representations: A simple and general method for semi-supervised learning. *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, c. 384-394*, 2010.

17. R. Kadlec et al. Improved deep learning baselines for ubuntu corpus dialogs. *arXiv:1510.03753*, 2015.

18. D. Britz. Understanding convolutional neural networks for nlp. http://www.wildml.com/2015/11/ understanding-convolutional-neural-networks-for-nlp/.

19. C. Manning and R. Socher. Лекции Стэнфордского университета по курсу "natural language processing with deep learning". http://web. stanford.edu/class/cs224n/.

20. Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

21. Stanford University. Unsupervised feature learning and deep learning tutorial. http://deeplearning.stanford.edu/tutorial/.

22. Zhang, X. Character-level convolutional network for text classification / Xiang Zhang, Junbo Zhao, Yann LeCun // In Advances in Neural Information Processig Systems. – 2015. – Feb. – 649 – 657 p.

23. Воронцов, К.В. Курс лекций по машинному обучению / К. В. Воронцов – 2015.

24. Алферов А.П. Основы криптографии / А.П. Алферов – 2002.

Граф вычислений для сверточной нейронной сети

```
import tensorflow as tf


class ConvolutionalNN(object):
    def __init__(self, sequence_length, vocabulary_size,
                 embedding_size, filter_sizes, num_filters,
                 l2_reg_lambda=0.0):
        self.x = tf.placegolder(tf.int32, [None, sequence_length],
                                name="x")
        self.y = tf.placeholder(tf.float32, [None, 2], name="y")
        self.dropout_keep_prob = tf.placeholder(tf.float32,
                                                name="dropout_keep_prob")

        l2_reg_loss = tf.constant(0.0)

        #Слой векторного представления букв
        with tf.name_scope("embedding"):
            self.W = tf.Variable(tf.random_uniform([vocabulary_size,
                                                    embedding_size],
                                -1.0, 1.0), name="W")
            self.embedded_words =
tf.expand_dims(tf.nn.embedding_lookup(self.W, self.x), -1)

        pooled_outputs = []

        #Сверточный слой и слой max-pooling'а для каждого фильтра
        for i , filter_size in enumerate(filter_sizes):
            with tf.name_scope("conv-maxpool-%s" % filter_size):

                filter_shape = [filter_size, embedding_size, 1,
                                num_filters]
                W = tf.Variable(tf.truncated_normal(filter_shape,
                                                    stddev=0.1), name="W")
                b = tf.Variable(tf.constant(0.1, shape=[num_filters]),
name="b")
                conv = tf.nn.conv2d(self.embedded_words, W,
                                    strides=[1, 1, 1, 1],
                                    padding="VALID", name="conv")

                #max(tf.nn.bias_add(conv,b),0)
                h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
```

```
                pooled = tf.nn.max_pool(h, ksize=[1,sequence_length - fil-
ter_size + 1, 1, 1],
                                        strides=[1,1,1,1], paddig="VALID",
                                        name="max-pool")
                pooled_outputs.append(pooled)

        #Объединение признаков, полученных в результате пулинга
        num_filters_total = num_filters * len(filter_sizes)
        self.pool = tf.reshape(tf.concat(pooled_outputs, 3),
                               [-1,num_filters_total])

        #Добавление дропаута
        with tf.name_scope("dropout"):
            self.drop = tf.nn.dropout(self.pool, self.dropout_keep_prob)

        #Final(unnormalized) scores and predictions
        with tf.nn.name_scope("output"):
            W = tf.get_variable("W", shape=[num_filters_total, 2],
                                initializer =
tf.contrib.layers.xavier_initializer())
            b = tf.Variable(tf.constant(0.1, shape=[2]), name="b")
            l2_reg_loss += tf.nn.l2_reg_loss(W)
            l2_reg_loss += tf.nn.l2_reg_loss(b)
            self.scores = tf.nn.xw_plus_b(self.drop, W, b, name="scores")
            self.predictions = tf.argmax(self.scores, 1,
name="predictions")

        #Минимизация потер перекрестной энтропии и ее среднее значение
        with tf.name_scope("loss"):
            losses = tf.nn.softmax_cross_entropy_with_logits(
                    logits=self.scores, labels=self.y)
            self.loss = tf.reduce_mean(losses) + l2_reg_lambda *
l2_reg_loss

        #Полученная точность
        with tf.name_scope("accuracy"):
            correct_predictions = tf.equal(self.predictions,
tf.argmax(self.y,1))
            self.accuracy = tf.reduce_mean(tf.cast(
                    correct_predictions, "float"), name="accuracy")
```

```
def encrypt(text, gamma):
    textLen = len(text)
    gammaLen = len(gamma)

    #Формируем ключевое слово(растягиваем гамму на длину текста)
    keyText = []
    for i in range(textLen // gammaLen):
        for symb in gamma:
            keyText.append(symb)
    for i in range(textLen % gammaLen):
        keyText.append(gamma[i])

    #Шифрование
    code = []
    for i in range(textLen):
        code.append(alphabeth[(alphabeth.index(text[i]) +
alphabeth.index(keyText[i])) % 26])

    return code
```

```
def encrypt(text, permutation):
    blockSize = len(permutation)
    textSize = len(text)

    #Выравнивание текста
    difference = (textSize) % blockSize
    if difference != 0:
        #добить нужным количеством символов
        for i in range(blockSize - difference):
            text.append(chr(random.randrange(ord('a'), ord('я'), 1)))

        #взять новый размер строки
        textSize = len(text)

    #Шифрование
    for i in range(0, textSize, blockSize):
        string = [ text[i+j] for j in range(blockSize)]
        for j in range(blockSize):
            text[i + j] = string[permutation[j]]

    return text
```

```
import tensorflow as tf
import numpy as np
import ComvolutionalNN
#модуль для получения обработанных данных
import get_data

tf.flags.DEFINE_string("word2vec", None,)
tf.flags.DEFINE_string("filter_sizes", "2,3,4")
tf.flags.DEFINE_integer("num_filters", 128)
tf.flags.DEFINE_integer("embedding_dim", 150)
tf.flags.DEFINE_float("dropout_keep_prob", 0.8)
tf.flags.DEFINE_float("l2_reg_lambda", 4)
tf.flags.DEFINE_integer("batch_size", 50)
tf.flags.DEFINE_integer(
        "num_epochs", 20, "Number of training epochs (default: 100)")
tf.flags.DEFINE_integer("evaluate_every", 100)

FLAGS = tf.flags.FLAGS

...

cnn = ConvolutionalNN(
        sequence_length=x_train.shape[1], num_classes=y_train.shape[1],
        vocab_size=len(vocab_processor.vocabulary_),
        embedding_size=FLAGS.embedding_dim, filter_sizes=list(
                map(int, FLAGS.filter_sizes.split(","))),
        num_filters=FLAGS.num_filters, l2_reg_lambda=FLAGS.l2_reg_lambda)
global_step = tf.Variable(0, name="global_step", trainble=False)
optimizer = tf.train.AdamOptimizer(1e-3)
grads_and_vars = optimizer.compute_gradients(cnn.loss)
train_op       =       optimizer.apply_gradients(grads_and_vars,       glob-
al_step=global_step)
batches = get_data.batches(list(zip(x_train, y_train)),
                           FLAGS.batch_size, FLAGS.num_epochs)
sess = tf.Session()
sess.run(tf.initialize_all_variables())
saver = tf.train.Saver()

if FLAGS.word2vec:
    initW = np.random.uniform(-0.25, 0.25,
                              (len(vocab_processor.vocabulary_),
                              FLAGS.embedding_dim))
    #load any vectors from the word2vec
    print("Load word2vec file {}\n".format(FLAGS.word2vec))
    with open(FLAGS.word2vec, "rb") as f:
```

```
            header = f.readline()
            vocab_size, layer1_size = map(int, header.split())
            for line in f:
                line = line.decode('ascii')
                word = line.split(' ', 1)[0]
                idx = vocab_processor.vocabulary_.get(word)
                if idx != None:
                    numbers = line.split(' ', 1)[1]
                    initW[idx] = np.fromstring(numbers, dtype=float, sep=' ')
        sess.run(cnn.W.assign(initW))
        for batch in batches:
            x_batch, y_batch = zip(*batch)
            train_step(x_batch, y_batch)
            current_step = tf.train.global_step(sess, global_step)
            if current_step % FLAGS.evaluate_every == 0:
                print("\nEvaluation:")
                validation_step(x_validation, y_validation,
                                writer=validation_summary_writer)
                print("")
            if current_step % FLAGS.checkpoint_every == 0:
                path = saver.save(sess, checkpoint_prefix,
                                  global_step=current_step)
                print("Saved model checkpoint to {}\n".format(path))


    def train_step(x_batch, y_batch):
        feed_dict = {
                cnn.x: x_batch,
                cnn.y: y_batch,
                cnn.dropout_keep_prob: FLAGS.dropout_keep_prob
            }
        step, summaries, loss, accuracy = sess.run(
                [train_op, global_step, train_summary_op, cnn.loss,
                 cnn.accuracy], feed_dict)
        time_str = datetime.datetime.now().isoformat()
        print("{}: step {}, loss {:g}, acc {:g}".format(
                time_str, step, loss, accuracy))
        train_summary_writer.add_summary(summaries, step)


    def validation_step(x_batch, y_batch, writer=None):
        feed_dict = {
                cnn.x: x_batch,
                cnn.y: y_batch,
                cnn.dropout_keep_prob: 1.0
            }
```

```
step, sumamries, loss, accuracy = sess.run(
        [global_step, validation_summary_op, cnn.loss,
         cnn.accuracy], feed_dict)
time_str = datetime.datetime.now().isoformat()
print("{}: step {}, loss {:g}, acc {:g}".format(
        time_str, step, loss, accuracy))
if writer:
    writer.add_summary(summaries, step)
```

```
alphabet = ['а','б','в','г','д','е','ё','ж','з','и',
            'й','к','л','м','н','о','п','р','с','т',
            'у','ф','х','ц','ч','ш','щ','ъ','ы','ь',
            'э','ю','я',' ']

#Функция шифрования по ключу
#text - list символов текста
#key - list с перестановкой на алфавите
def encrypt(text, key):
    result = []
    for i in range(len(text)):
        result.append(key[alphabet.index(text[i])])

    return result
```