# ПРАКТИЧЕСКАЯ ЧАСТЬ

# Реализация алгоритма

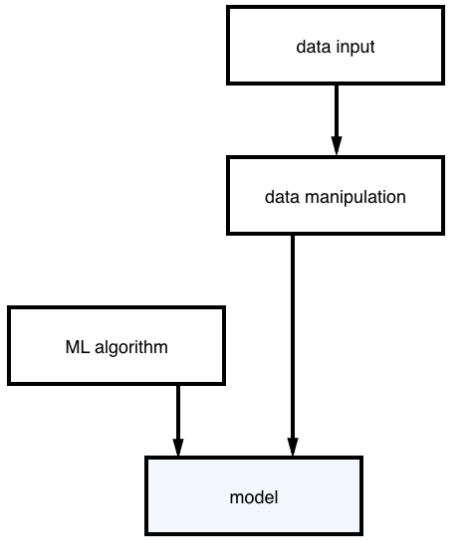
## 6.1. Формализация задачи

* на вход поступают тексты произвольной длины и содержания;
* данный текст необходимо как-то проанализировать и классифицировать его по нескольким категориям;
* для каждого текста может присвоена только одна категория;
* имеется несколько сотен уже классифицированных текстов.

## 6.2. Прогнозирующая модель

**Алгоритм машинного обучения + Данные = Прогнозирующая модель**

Процесс построения таков:



Можно заметить, что модель состоит из алгоритма машинного обучения, «натренированного» на данных. Из которых формируется модель прогнозирования, потом выдается соответствующий результат:



Целью модели, которая будет создаваться, будет заключаться в том, чтобы классифицировать текст по категориям, их можно задать:

* ввод: текст;
* результат: категория.

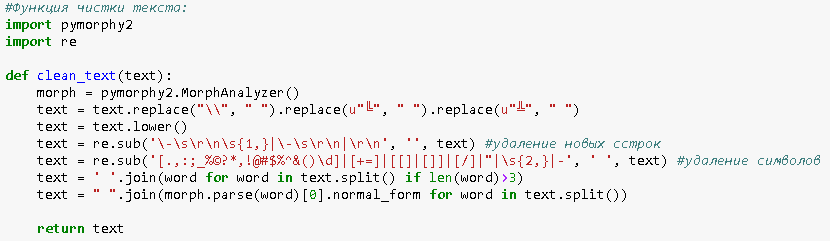
Так как некий есть тренировочный набор данных, в котором все тексты распределены по категориям, то можно использовать обучение с учителем.

## 6.3. Подготовка данных

Первая задача, которая возникает, это то каким образом подать данные в нейронную сеть. Забегая немного вперед, что суть классификации текста заключается в том, чтобы составить словарь и заменить каждое слово во входном тексте на некий номер или индекс, после чего выровнять длину каждого текста до необходимого нам размера. Дальше уже можно будет применять различные методы и виды нейронных сетей для решения задачи и определить какой будет давать наилучшую точность и производительность.

Словарь можно составить несколькими способами. Например, можно взять все слова русского языка или английского языка, в зависимости от языка на котором писался текст, и это будет словарем. Но такой подход не подходит, так как размер словаря будет огромным, и в нем будут находится слова, которые никогда не будут встречаться в наших текстах, что будет нецелесообразным использованием ресурсов. Второй метод, это составить словарь из слов, встречающихся в обучающей и тестовой выборках. Но просто взять и составить словарь еще нельзя, потому что в тексте встречаются различные спецсимволы, предлоги, местоимения, которые не несут смысловой нагрузки, ну и конечно же все эти слова желательно привести к нормальной форме, т.к. слова с разными окончаниями будут записываться в разные слова, что опять же будет занимать лишнюю память.

Есть вспомогательные инструменты, с помощью которых можно провести «чистку» текста, которые принимают строку на вход, разбивают все предложения на отдельные слова, удаляют спецсимволы, выбрасывают все слова (преимущественно предлоги), длинна которых меньше 3-х символов и возвращают «очищенный» текст, состоящий из слов нормальной формы.



Пример:

* исходный текст: «уже давно известные маркетологи используют различные приёмы воздействия на клиента.»
* отфильтрованный текст: «давно известно маркетолог использовать

различный приём воздействие клиент»

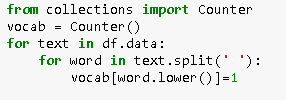
Теперь, когда текст отфильтрован мы можем собрать словарь слов из текстов, которые у нас имеются, также использовав вспомогательные

библиотеки. Составлять тоже можно несколькими способами, учитывая или не учитывая то, сколько раз слово встретилось в выборке. Способ при

котором учитывается количество вхождений удобен, когда получается очень большой словарь и нам нужно отбросить часть слов, которые встречаются

очень редко.

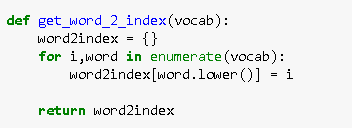
Пример такого кода:



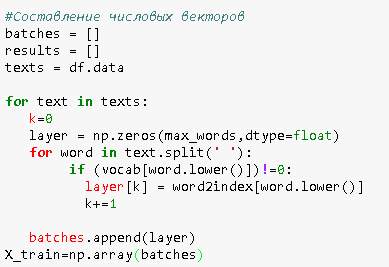
df – это выборка, по которой будет обучаться и тестироваться текст.

Словарь составлен, нужно определиться то, в каком виде данные будут подаваться в сеть, в какой размерности. Если взять размерность словаря, создать вектор с таким же размером, заполнить нулями, а потом расставить единички там, где слово из словаря встретилось во входном тексте один раз, двойку там, где два раза и т.д. Этот метод будет эффективным только при малом объеме данных, для словаря из нескольких тысяч слов это будет очень долго работать, плюс ко всему такой вектор будет содержать преимущественно одни нули, т.к. словарь очень большой. Поэтому будет эффективнее взять за основу самый длинный текст из выборки, посчитать количество слов в данном тексте и взять его размерность за основу. И расставлять уже в векторе индексы слов в словаре. Например, слово «математика» стоит словаре под индексом «10», значит во входном векторе на месте, где стоит слово «математика» будет стоят число «10».

Определим функцию, которая будет возвращать индексы:

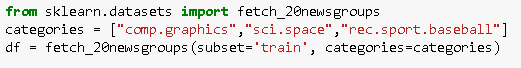


Пример кода создания числового вектора:

****

Входной вектор определили, а на выходите будет вектор, размерностью равный количеству категорий в данных текстах. Например, для трех категорий вектор [0,1,0] означает, что текст относится ко второй категории.

Для решения задачи и сравнения результатов разных архитектур и видов нейронных сетей будем использовать  [20 Newsgroups](http://qwone.com/~jason/20Newsgroups/)— набор данных, содержащий 18 тысяч постов на 20 тем. Для экономии времени вычислений, загрузим только 3 категории:



После загрузки и фильтрации текста размерность словаря составила: 20702 слова.

Всего в выборке получилось 1774 текста со следующим разбиением

по темам: [597. 593. 584.]. Разделим на обучающую и тестовую выборки. Для тестовый сделаем 300 текстов, для обучающей – 1474 (выборку с категориями разбиваем таким же способом). Самые длинный текст равен 6300 слов, что очень много. Но таких текстов всего пару десятков во всей выборке, поэтому его можно сократить без особого влияния на результат, сократим такие

тексты до 500 слов.

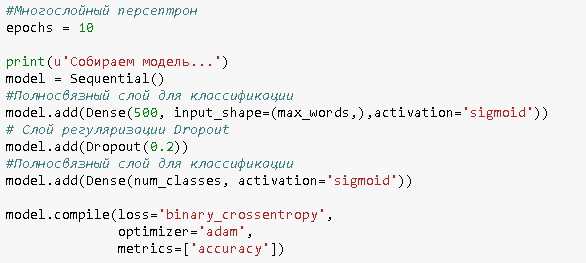
Размерности получились следующие:

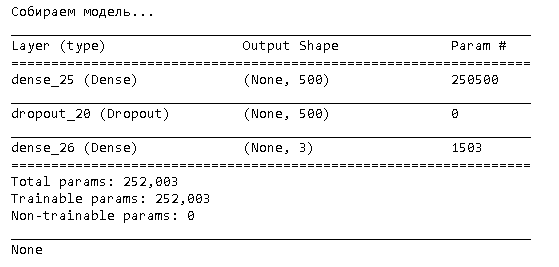
Размерность обучающей выборки: (1474, 500)

Размерность тестовой выборки: (300, 500)

## **6.4. Конструирование моделей и сравнение результат**ов

Все данные подготовлены, теперь необходимо собрать нейросеть, выбрать тип и модель, затем обучить ее. Для начала возьмем самый простой тип – это многослойный полносвязный персептрон:





Мы создали слои, в которых каждый нейрон связан со всеми входами или предыдущими слоями. Такие слои в Keras называются "Dense". Задаем функцию активации выходного слоя - sigmoid, которая применяется для задач классификации. Сеть создана двуслойная, потому что однослойная сеть дала результаты еще хуже, чем эта сеть.

Model.compile - компилирует модель. Это значит, что мы построим функцию Python, которая позволит вычислять результат работы сети на входном векторе, вызывая функцию model.predict(...). Но, главное, посчитается функция вычисления градиента функции ошибки по весам сети, что необходимо для осуществления обучения (настройки) параметров сети.

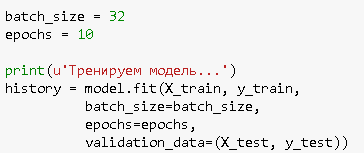
Здесь 'adam' - тип градиентного спуска, 'binary\_crossentropy' - функция штрафа, кроссэнтропия, которую следует использовать для задач классификации, как у нас, metrics=['accuracy'] значит, что мы будем вычислять в модели не только функцию штрафа, но и точность работы, то есть, число правильно классифицированных примеров.

Теперь сеть нужно обучить - это самая ресурсоемкая операция. Мы будем выполнять обучение "пачками" (mini-batch) по batch\_size=32 примеров. При этом, осуществляет epochs=10 проходов по всем входным примерам.

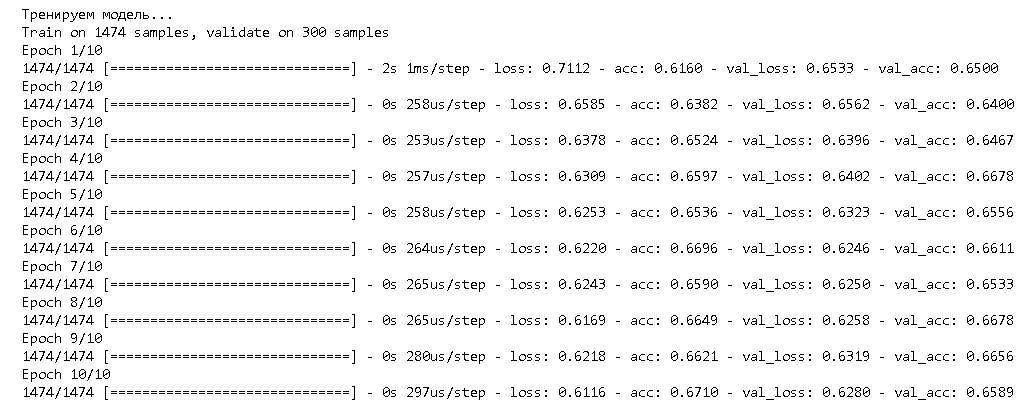
Также есть другие варианты режима обучения:

* по одному примеру - медленная сходимость, и алгоритм "не видит" хороших минимумов,
* по всем примерам сразу - трудоемко, и сходимость к локальному минимуму и склонность к переобучению.

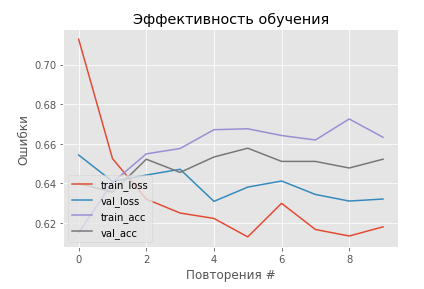
А обработка mini-batch - промежуточный вариант, который называется градиентным стохастическим спуском.



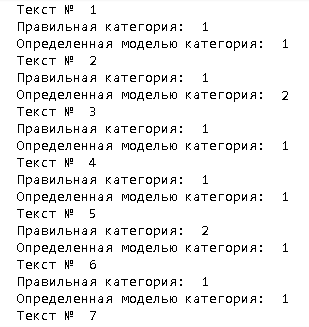
Информация о проходах обучения:

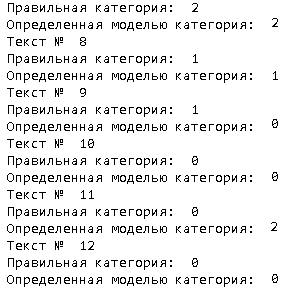


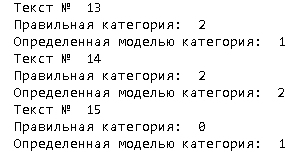
Здесь loss - функция ошибки, acc - точность на обучающей выборке, val\_loss и val\_acc - на тестовой выборке. Время занимает не много, так как сеть простая. Все 10 эпох прошли за пару секунд.



Примеры предсказаний:

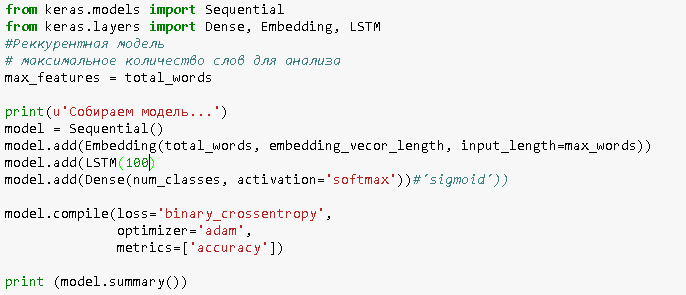


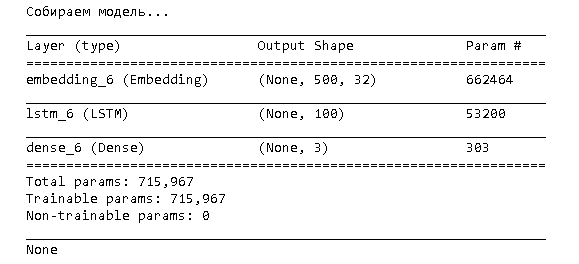




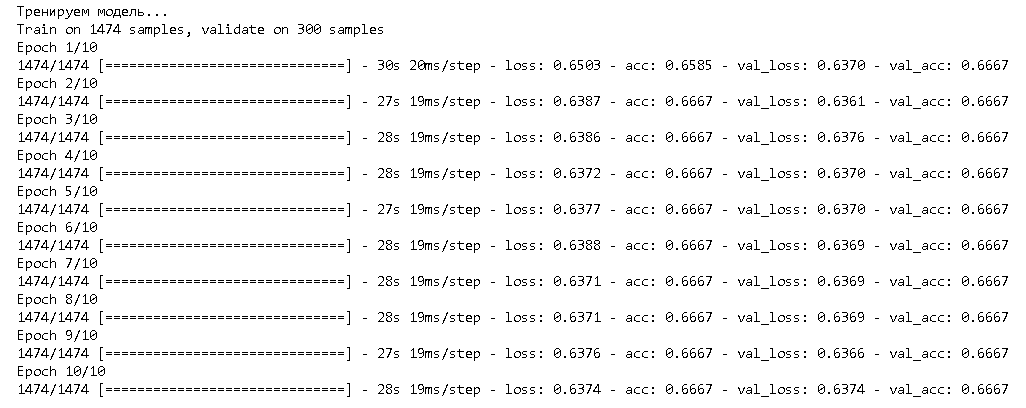
Получилось 6 ошибок из 15.

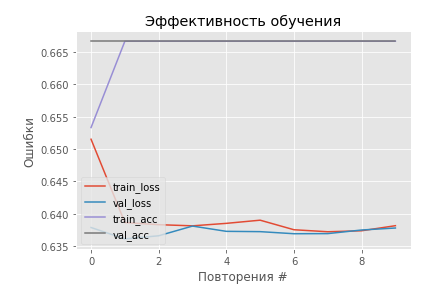
Видим, что точность на тестовой выборке получилась val\_acc: 0.6589, то есть, 65.89%. Точность неудовлетворительная. Попробуем теперь проделать все тоже самое для рекуррентной сети. Забегая вперед скажу, что чисто рекуррентная сеть показала результаты около 50% процентов, поэтому были применены меры создать многослойную рекуррентная нейронную сеть с блоками LSTM:



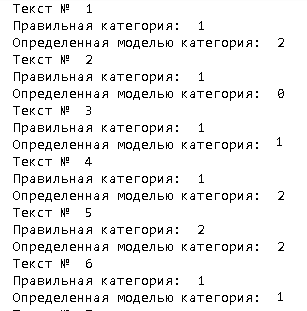


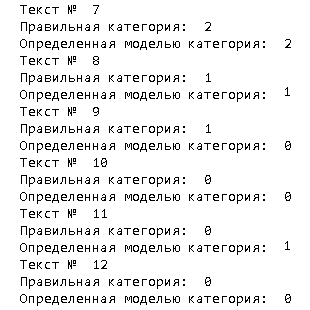
Тренировка всех 10 эпох выглядит так:

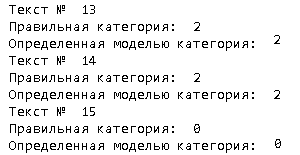




Предсказания:







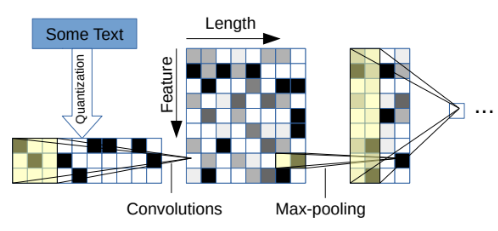
Получилось 5 ошибок из 15, чуть меньше чем в предыдущем примере.

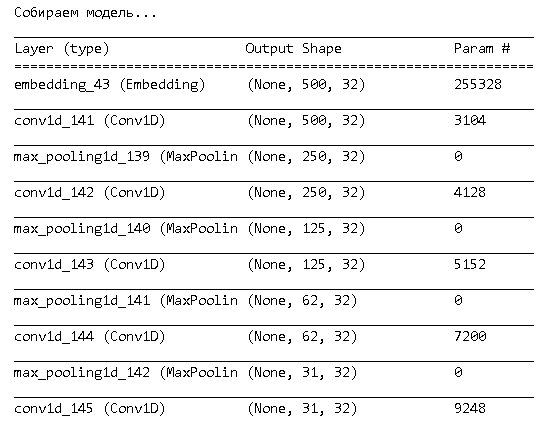
Модель за 1 эпоху достигает своего максимума, в остальных эпохах нет смысла. По времени одной эпохи, модель обучилась за 30 секунд. Точность равно 66,67%, что практически не отличается от предыдущей модели.

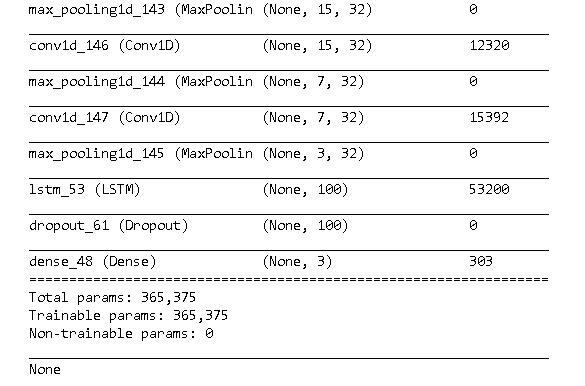
Теперь перейдем к сетям, которые все очень хвалят – это сверточные сети. В нашей задачи используются векторы, поэтому использовать мы будет одномерные сверточные сети. Свернем данные прежде чем подать их в рекуррентную сеть. Путем экспериментов мы пришли к следующей архитектуре сети:



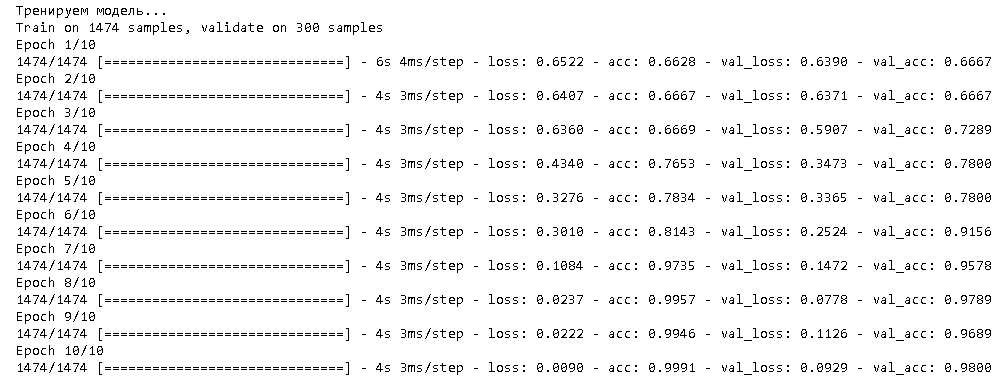
Первым слоем идет Embedding, который переводит целые числа (на самом деле one-hot вектора, в которых место единицы соответствует номеру слова в словаре) в плотные вектора. После embedding идет одномерный сверточный слой Conv1D. Количество фильтров в нем — 32, а ширина окна для фильтров равна 3. Слой объединения может использовать стандартную длину 2, чтобы уменьшить размер карты объектов в два раза. Далее постепенно увеличиваем размер ядра, и сворачиваем вектор до необходимых размеров. Далее идет Max-pooling. Проще всего его объяснить на примере. Представим, что в уже сверточном слое матрица фильтра зафиксирована и является единичной (то есть умножение на нее никак не влияет на входные данные). А еще вместо суммирования всех результатов умножения (входных данных по нашему условию), мы выбираем просто максимальный элемент. То есть мы выберем из всего окна пиксель с наибольшей интенсивностью. Это и есть max-pooling. Конечно, вместо функций максимум может быть другая арифметическая (или даже более сложная) функция.



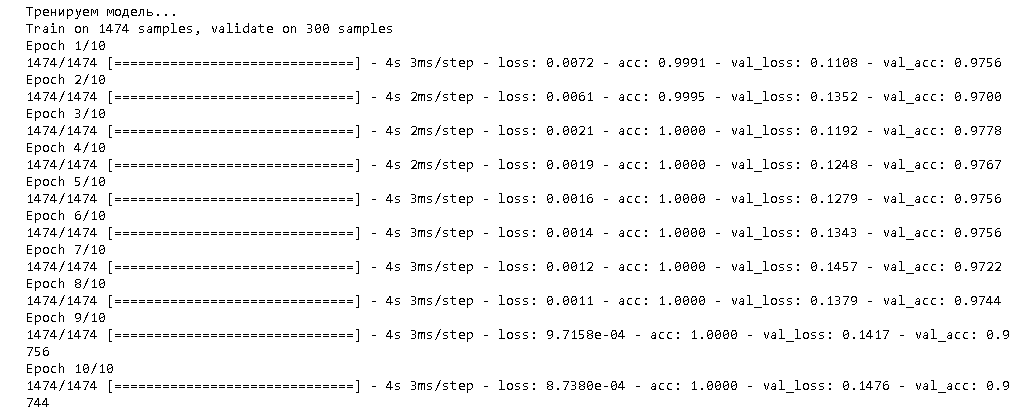


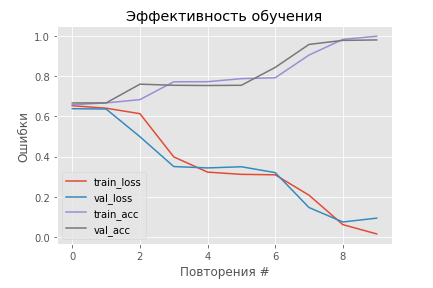


Обучение:



Так как все 10 эпох точность только увеличивалась, есть смысл провести еще 10 эпох и посмотреть, что получится:



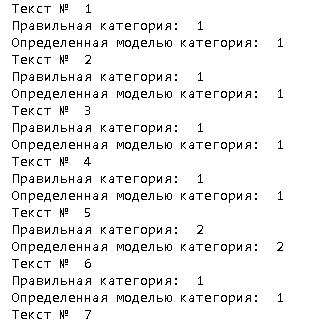


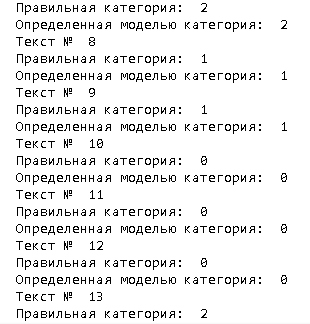
Точность наоборот опустилась с 98% до 97,44%, что означает что сеть переучилась (еще 10 эпох были лишними), 98% является отличным результатом. Обучение заняло 40 секунд, что тоже очень быстро.

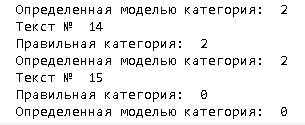
Теперь, когда сеть обучена, можно сделать прогнозы над текстами, которые не участвовали в обучении сети. Для этого будем использовать следующий код:



Проверим прогнозы на первых 15 текстах:







Совпали 15 из 15 прогнозов. В результате, сверточная сеть показала

наилучший результат по точности.

Пример одного из тестовых текстов:

«From: rcasteto@watsol.uwaterloo.ca (Ron Castelletto)

Subject: Orioles Phillies Red Sox

Keywords: orioles phillies red sox baltimore philadelphia boston bosox

Organization: University of Waterloo

Distribution: na

Lines: 20

Can someone send me ticket ordering information for the

following teams: Baltimore, Philadelphia and Boston.

Also, if you have a home schedule available - can you tell me the dates

for all home games between July26-Aug6 and between Aug30-Sept10 and if

any of these games are promotion nights or special discount nights?

Thanks !!! Ron

PS: and also who the opponents are for these games :-)

Do NOT reply to this account,

please reply to: ronc@vnet.ibm.com

\_\_ \_ IBM Canada Lab Database Technology

| \ / \ Associate Development Analyst

|\_\_/ on | astelletto (416) 448-2546 Tie Line: 778-2546

| \\_ \\_/ Internal Mail: 51/843/895/TO»

Обработанный текст: «from rcasteto watsol uwaterloo castelletto subject orioles phillies keywords orioles phillies baltimore philadelphia boston bosox organization university waterloo distribution lines someone send ticket ordering information following teams baltimore philadelphia boston also have home schedule available tell dates home games between july between sept these games promotion nights special discount nights thanks also opponents these games reply this account please reply ronc vnet canada database technology associate development analyst astelletto line internal mail»

Категория, которую определила нейросеть совпала с правильной. Это

категория: «rec.sport.baseball»

В результате мы построили модель, которая почти с точностью 100%

угадывает принадлежность текста, который подается в сеть, к той или иной

категории. Самой эффекивной сетью оказалась сверточная сеть, так как она

сворачивается входные до необходимых нам размеров, что позволяет точнее

настроить веса нейронной сети и потом уже определить выходной вектор.

Казалось бы, все работает все прекрасно, но возникает вполне логичный вопрос, а что будет если на вход подать текста, который не участвовал в составлении словаря, более того если он не относится ни к одной категории,

которые у нас есть? Для этого необходимо добавить еще один класс категорий,

чтобы в случае если входные данные не относятся ни к одной из трех категорий,

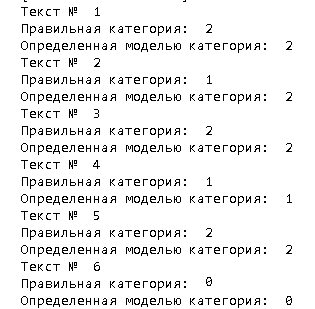
этот текст определился в отдельную категорию, назовем ее «unknow» (не

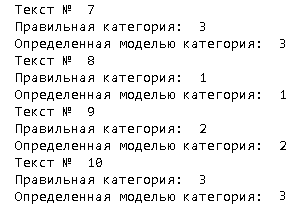
относится ни к одной категории». Соответственно, сеть нужно предварительно

доучить, чтобы, когда на вход подастся нулевая матрица (ни одно слово из

текста не содержится в словаре), она знала, что это категория под названием

«unknown» . Вот пример, уже дообученной сети:

****

****

1 ошибка из 10 примеров. Все примеры содержали тексты, которые не

Участвовали в образовании словаря. Текст под номером 6 был совершенно

другой тематики, сеть определила верно. «0» - не относится ни к одной

категории («unknow» ).

## 6.5. Таблица результатов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Модель сети** | **Время обучения**  **(в сек.)** | **Точность**  **(в %)** | **Количество**  **параметров**  **(в тыс.)** | **Количество ошибок на 15 примерах** |
| Полносвязный персептрон | ~3 | 65,89 | 252,003 | 6 |
| Рекуррентная сеть | 30 | 66,67 | 715,967 | 5 |
| Сверточная и рекуррентная сеть | 40 | 98,00 | 365,375 | 0 |

## 6.6. Модель для данных, собранных вручную

Также была попытка собрать свою выборку. За основу был взят

научный журнал «Universum: экономика и юриспруденция», кототрый

содержит следующие рублики:

**Экономические науки (08.00.00)**

01. Экономическая теория.  
02. Экономика и управление народным хозяйством.  
03. Финансы, денежное обращение и кредит.  
04. Бухгалтерский учет, статистика.  
05. Математические и инструментальные методы экономики.  
06. Мировая экономика.

**Юридические науки (12.00.00)**

07. Теория и история права и государства, история учений о праве и государстве.  
08. Конституционное право, конституционный судебный процесс, муниципальное право.  
09. Гражданское право, предпринимательское право, семейное право, международное частное право.  
10. Финансовое право, налоговое право, бюджетное право.   
11. Трудовое право, право социального обеспечения.  
12. Земельное право, природоресурсное право, экологическое право, аграрное право.  
13. Корпоративное право, энергетическое право.   
14. Уголовное право и криминология, уголовно-исполнительное право.  
15. Уголовный процесс.  
16. Международное право, Европейское право.  
17. Судебная деятельность, прокурорская деятельность, правозащитная и правоохранительная деятельность.  
18. Криминалистика, судебно-экспертная деятельность, оперативно-розыскная деятельность.  
19. Информационное право.  
20. Административное право, административный процесс.  
21. Гражданский процесс, арбитражный процесс.

Журнал выпускается с 2013года, в моем распоряжении было 52

журнала, что составило 231 различный статей. К сожалению, распределение

по категориям было очень не равноемерное:

[13. 39. 25. 23. 6. 6. 19. 10. 22. 7. 4. 6. 2. 18. 6. 5. 6. 1. 3. 8. 2.]

Что не позволило в полной мере протестировать модель, к тому же данных

очень мало для 21 категории. За основу брался не весь текст, а только

ключевые слова, и обзац в котором, по моему мнению содержалась

основня мысль всех статью (в идеальном случае нужен хороший лингвист,

чтобы определить какие слова несут основную смысловую нагрузку).

Выборка также была разбита на тестовую и обучающую:

Размерность обучающей выборки: (216, 119);

Размерность тестовой выборки: (15, 119).

Все модели строились аналогичным способом. Реккурентная модель показала лучшие результаты для малой выборки, около 45% тестовых данных она угадала. Для сверточной сети данных катастрафически мало, она показала всего лишь 13,3% точности.

Вывод можно сделать такой что, для корректной работы классификатора необходимо около 500 текстов на каждую категорию.

# ЗАКЛЮЧЕНИЕ

В соответствии с поставленной целью — разработка алгоритмов

глубокого обучения для классификации текста и сравнение между собой

различных архитектур — были реализованы архитектуры свёрточной

нейронной сети, рекуррентной нейронной сети с LSTM-блоками,

многослойного персептрона. Сверточная нейронная в дополнение к

рекуррентной сети показала лучшие результаты. Точность классификатора на

основе свёрточной нейронной сети оказалась 98%. Результаты исследования

показывают, что использование глубоких нейронных сетей очень хорошо

решает задачу классификации, если сравнивать с результатами обычных

классификаторов, приведенные в различных статьях, то нейросети значительно превосходят по точности определения принадлежности текста к какой-либо

категории.

Также можно сделать вывод, что для решения подобного типа задач,

Необходим относительно большой набор обучающих данных, хотя бы по 500

текстов на одну категорию. Если такие данные есть, то можно решить очень

много интересных и общественно полезных задач. Например, есть известная

проблема по определению негативных постов в социальных сетях,

Призывающие к насилию т.д., или можно применить в различных журналах

для автоматического определения категории текста вновь присланной статьи

для публикации.

Возможным направлением для дальнейшей работы может быть

распознавания ключевых слов, вносящих наибольший вклад в

принадлежность к той или иной категории.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

* 1. Плотникова, Н.П. Модели, алгоритмы и реализация нейронных сетей в

многопроцессорных и распределенных вычислительных средах: дис. канд. Технических наук: 05.13.01 / Н.П. Плотникова; Мордовский государственный университет им. Н.П. Огарева. - Саранск, 2014. 137 с.

2. Каллан, Робертс. Основные концепции нейронных сетей / Каллан Робертс – М.: Издательский дом «Вильямс», 2001. 287 с.

3. Нейрокомпьютеры и их применение: в 18 кн. / под общей ред. А.И. Галушкина, Я.З. Цыпкина. - М.: ИПРЖР, 2001. – Кн.5: Нейронные сети: история развития теории: Учеб. пособие для вузов. 840 с.

4. Цой, Ю.Р. Эволюционный подход к настройке и обучению искусственных нейронных сетей / Ю.Р. Цой, В.Г. Спицын // Нейроинформатика. - 2006. 341 с.

5. Su-Li Yan, Ying Wang, Ji-Cheng Liu.Research on the Comprehensive Evaluation of BusinessIntelligence System Based on BP Neural Network. 2nd International Conference on Complexity Science & Information Engineering.North China Electric Power University, Beijing 102206, China, 2011, 228 с.

6. <https://keras.io/>

7. <https://www.tensorflow.org/>

8. <https://ru.wikipedia.org>

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

10. <http://7universum.com>

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

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

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

14. K. Tran et al. Evaluation of deep learning toolkits. https://github. com/zer0n/deepframeworks/blob/master/README.md.

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

16. S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural Computation 9(8), c. 1735-1780, 1997.

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

18. 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.

19. 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.

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

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

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

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

25. К. К. Семёнов. Автоматическое дифференцирование функций, выра- женное программным кодом. Вестник Саратовского государственного технического университета, 2011.

26. 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.

27. В.И. Фомин. Методические указания по выполнению дополнительного раздела “Информационный маркетинг”. СПбГЭТУ «ЛЭТИ», 2015.

28. 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.

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

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

31. C. Manning and R. Socher. Лекции Стэнфордского университета по курсу “natural language processing with deep learning”. http://web. stanford.edu/class/cs224n/. (дата обращения 20.05.2017).

32. C. Manning and R. Socher. Лекции Стэнфордского университета по курсу “natural language processing with deep learning”. http://web. stanford.edu/class/cs224n/. (дата обращения 20.05.2017).

33. C. Olah. Neural networks, recurrent neural networks, convolutional neural networks. <http://colah.github.io/>.

34. Stanford University. Unsupervised feature learning and deep learning tutorial. http://deeplearning.stanford.edu/tutorial/. (дата обра- щения 25.05.2017).

35. Neural Networks: A Comprehensive Foundation, [Саймон Хайкин](https://www.ozon.ru/person/2480518/), 2016г.

36. Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift."arXiv preprint arXiv:1502.03167 (2015).

37. Shimodaira, Hidetoshi. "Improving predictive inference under covariate shift by weighting the log-likelihood function."Journal of statistical planning and inference 90.2 (2000): 227-244

38. Sutskever, Ilya, et al. "On the importance of initialization and momentum in deep learning."Proceedings of the 30th international conference on machine learning (ICML-13). 2013.

39. Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization."The Journal of Machine Learning Research 12 (2011): 2121-2159.

40. Tieleman, Tijmen, and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude."COURSERA: Neural Networks for Machine Learning 4 (2012): 2.

41. Zeiler, Matthew D. "ADADELTA: an adaptive learning rate method."arXiv preprint arXiv:1212.5701 (2012).

42. Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization."arXiv preprint arXiv:1412.6980 (2014).

43. LeCun, Yann A., et al. "Efficient backprop."Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012. 9-48.

44. Waibel, Alexander, et al. "Phoneme recognition using time-delay neural networks."Acoustics, Speech and Signal Processing, IEEE Transactions on 37.3 (1989): 328-339

45. Kolarik, Thomas, and Gottfried Rudorfer. "Time series forecasting using neural networks."ACM Sigapl Apl Quote Quad. Vol. 25. No. 1. ACM, 1994.

46. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks."Advances in neural information processing systems. 2012.

47. Hecht-Nielsen, Robert. "Theory of the backpropagation neural network."Neural Networks, 1989. IJCNN., International Joint Conference on. IEEE, 1989.

48. Amari, Shunichi. "A theory of adaptive pattern classifiers."Electronic Computers, IEEE Transactions on 3 (1967): 299-307.

49. Mnih, Volodymyr, Nicolas Heess, and Alex Graves. "Recurrent models of visual attention."Advances in Neural Information Processing Systems. 2014.

50. https://habr.com/

# ПРИЛОЖЕНИЕ 1

# coding: utf-8

# In[3]:

#Загрузка даных:

from sklearn.datasets import fetch\_20newsgroups

categories = ["comp.graphics","sci.space","rec.sport.baseball"]

df = fetch\_20newsgroups(subset='train', categories=categories)

# In[4]:

#Функция чистки текста:

import pymorphy2

import re

def clean\_text(text):

morph = pymorphy2.MorphAnalyzer()

text = text.replace("\\", " ").replace(u"╚", " ").replace(u"╩", " ")

text = text.lower()

text = re.sub('\-\s\r\n\s{1,}|\-\s\r\n|\r\n', '', text) #удаление новых строк

text = re.sub('[.,:;\_%©?\*,!@#$%^&()\d]|[+=]|[[]|[]]|[/]|"|\s{2,}|-', ' ', text) #удаление символов

text = ' '.join(word for word in text.split() if len(word)>3)

text = " ".join(morph.parse(word)[0].normal\_form for word in text.split())

return text

# In[ ]:

#Чистка текста и нахождение макс длины текста(или обрезание):

k=0

m=0

# Максимальное количество слов в самом длинном описании заявки

max\_words = 0

for text in df.data:

text=clean\_text(text)

df.data[k]=text

words = len(text.split())

if words > 500:

df.data[k]=text[:500]

words=500

m=k

if words > max\_words:

max\_words = words

k+=1

print('Максимальное количество слов в самом длинном описании заявки: {} слов'.format(max\_words))

# In[ ]:

#Составление словаря

import pandas as pd

import numpy as np

import tensorflow as tf

from collections import Counter

vocab = Counter()

for text in df.data:

for word in text.split(' '):

vocab[word.lower()]=1

total\_words = len(vocab)

print('Всего слов в словаре: ', total\_words)

#Функция, которая возвращает индекс слова в словаре

def get\_word\_2\_index(vocab):

word2index = {}

for i,word in enumerate(vocab):

word2index[word.lower()] = i

return word2index

word2index = get\_word\_2\_index(vocab)

#Составление числовых векторов

batches = []

results = []

texts = df.data

categories = df.target

for text in texts:

k=0

layer = np.zeros(max\_words,dtype=float)

for word in text.split(' '):

if (vocab[word.lower()])!=0:

layer[k] = word2index[word.lower()]

k+=1

batches.append(layer)

X\_train=np.array(batches)

#print(X\_train[0],X\_train.shape)

y=0

num\_classes=3

target=np.zeros((num\_classes))

y\_train=np.zeros((len(df.target),num\_classes))

for у in range(len(df.target)):

#print(y,'= ',df.target[y])

y\_train[y,df.target[y]]=1

target[df.target[y]]+=1

#print(y,'= ',y\_train[y])

y+=1

print('Распределение по рубликам: ',target)

#print(y\_train[0],y\_train.shape)

# In[ ]:

#Разбиение на обучающую и тестовую выборку

n=300

l=len(df.target)

#print(X\_train[i+1-n])

X\_test=X\_train[l-n:l+1]

y\_test=y\_train[l-n:l+1]

y\_train=y\_train[0:l-n]

X\_train=X\_train[0:l-n]

print('Размерность обучающей выборки: ',X\_train.shape)

print('Размерность тестовой выборки: ',X\_test.shape)

# In[ ]:

from keras.models import Sequential

from keras.layers import Dense, Embedding, LSTM

from keras.layers import Activation

from keras.layers import Dropout

#Реккурентная модель

# максимальное количество слов для анализа

max\_features = total\_words

embedding\_vector\_length = 32

print(u'Собираем модель...')

model = Sequential()

#Слой для векторного представления слов

model.add(Embedding(total\_words, embedding\_vector\_length, input\_length=max\_words))

#Слой долгосрочной памяти

model.add(LSTM(100)

#Полносвязный слой для классификации

model.add(Dense(num\_classes, activation='sigmoid'))

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

print (model.summary())

# In[ ]:

#Сверточная модель

# максимальное количество слов для анализа

max\_features = total\_words

print(u'Собираем модель...')

model = Sequential()

embedding\_vector\_length = 32

#Слой для векторного представления слов

model.add(Embedding(total\_words, embedding\_vector\_length, input\_length=max\_words))

#одномерная свертка 1 слоя

model.add(Conv1D(filters=32, kernel\_size=3, padding='same', activation='relu'))

#oбъединения для временных данных 1 слоя

model.add(MaxPooling1D(pool\_size=2))

#одномерная свертка 2слоя

model.add(Conv1D(filters=32, kernel\_size=4, padding='same', activation='relu'))

#oбъединения для временных данных 2слоя

model.add(MaxPooling1D(pool\_size=2))

model.add(Conv1D(filters=32, kernel\_size=5, padding='same', activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(Conv1D(filters=32, kernel\_size=7, padding='same', activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(Conv1D(filters=32, kernel\_size=9, padding='same', activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(Conv1D(filters=32, kernel\_size=12, padding='same', activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(Conv1D(filters=32, kernel\_size=15, padding='same', activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(LSTM(100))

model.add(keras.layers.Dropout(0.3))

model.add(Dense(num\_classes, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())

# In[ ]:

#Многослойный персептрон

epochs = 10

print(u'Собираем модель...')

model = Sequential()

#Полносвязный слой для классификации

model.add(Dense(512, input\_shape=(max\_words,),activation='sigmoid'))

# Слой регуляризации Dropout

model.add(Dropout(0.2))

#Полносвязный слой для классификации

model.add(Dense(num\_classes, activation='sigmoid'))

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

print(model.summary())

# In[ ]:

#Обучение модели

batch\_size = 32

epochs = 10

print(u'Тренируем модель...')

history = model.fit(X\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

validation\_data=(X\_test, y\_test))

# In[ ]:

#Оценка модели

score = model.evaluate(X\_test, y\_test,

batch\_size=batch\_size, verbose=1)

print()

print(u'Оценка теста: {}'.format(score[0]))

print(u'Оценка точности модели: {}'.format(score[1]))

# In[ ]:

#Получение прогноза от модели

import numpy as np

prediction = model.predict(X\_test)

j=0

for j in range(15):

print("Текст № ",j+1)

print('Правильная категория: ',np.argmax(y\_test[j]))

print("Определенная моделью категория: ",np.argmax(prediction[j]))

# ПРИЛОЖЕНИЕ 2

Программа для своих данных.

# coding: utf-8

# In[292]:

# coding: utf-8

import \_pickle as cPickle

#Загрузка даных:

file = open("file.txt", 'rb')

df = cPickle.load(file)

file.close()

i=230

print(df.data[i])

# In[293]:

#Функция чистки текста:

import pymorphy2

import re

def clean\_text(text):

morph = pymorphy2.MorphAnalyzer()

text = text.replace("\\", " ").replace(u"╚", " ").replace(u"╩", " ")

text = text.lower()

text = re.sub('\-\s\r\n\s{1,}|\-\s\r\n|\r\n', '', text)

text = re.sub('[.,:;\_%©?\*,!@#$%^&()\d]|[+=]|[[]|[]]|[/]|"|\s{2,}|-', ' ', text)

text = ' '.join(word for word in text.split() if len(word)>3)

text = " ".join(morph.parse(word)[0].normal\_form for word in text.split())

return text

# In[294]:

# Максимальное количество слов в самом длинном описании заявки

max\_words = 0

for desc in df.data[0:i+1]:

words = len(desc.split())

if words > max\_words:

max\_words = words

print('Максимальное количество слов в самом длинном описании заявки: {} слов'.format(max\_words))

# In[295]:

import pandas as pd

import numpy as np

import tensorflow as tf

from collections import Counter

#Составление словаря

vocab = Counter()

newsgroups\_train=df

for text in newsgroups\_train.data[0:i+1]:

for word in text.split(' '):

vocab[word.lower()]=1

total\_words = len(vocab)

print('Всего слов в словаре: ', total\_words)

#Функция, которая возвращает индекс слова в словаре

def get\_word\_2\_index(vocab):

word2index = {}

for i,word in enumerate(vocab):

word2index[word.lower()] = i

return word2index

word2index = get\_word\_2\_index(vocab)

#Составление числовых векторов

batches = []

results = []

texts = newsgroups\_train.data[0:i+1]

categories = newsgroups\_train.target[0:i+1]

for text in texts:

k=0

layer = np.zeros(max\_words,dtype=float)

for word in text.split(' '):

if (vocab[word.lower()])!=0:

layer[k] = word2index[word.lower()]

k+=1

batches.append(layer)

X\_train=np.array(batches)

y=0

num\_classes=21

target=np.zeros((num\_classes))

y\_train=np.zeros((i+1,num\_classes))

for у in range(i+1):

y\_train[y,df.target[y]-1]=1

target[df.target[y]-1]+=1

y+=1

print('Распределение по рубликам: ',target)

import keras

# In[296]:

#Разбиение на обучающую и тестовую выборку

n=15

X\_test=X\_train[i+1-n:i+1]

y\_test=y\_train[i+1-n:i+1]

y\_train=y\_train[0:i+1-n]

X\_train=X\_train[0:i+1-n]

print('Размерность обучающей выборки: ',X\_train.shape)

print('Размерность тестовой выборки: ',X\_test.shape)

# In[297]:

from keras.models import Sequential

from keras.layers import Dense, Embedding, LSTM

#Реккурентная модель

# максимальное количество слов для анализа

max\_features = total\_words

print(u'Собираем модель...')

model = Sequential()

model.add(Embedding(total\_words, embedding\_vecor\_length, input\_length=max\_words))

model.add(LSTM(64)

model.add(Dense(num\_classes, activation='softmax'))#'sigmoid'))

model.compile(loss='binary\_crossentropy',

optimizer='adam',

metrics=['accuracy'])

print (model.summary())

# In[301]:

from keras.layers import Dropout,Conv1D,MaxPooling1D

#Сверточная модель

# максимальное количество слов для анализа

max\_features = total\_words

print(u'Собираем модель...')

model = Sequential()

embedding\_vecor\_length = 32

model.add(Embedding(total\_words, embedding\_vecor\_length, input\_length=max\_words))

model.add(Conv1D(filters=32, kernel\_size=3, padding='same', activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(keras.layers.Dropout(0.3))

model.add(Conv1D(filters=32, kernel\_size=4, padding='same', activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(keras.layers.Dropout(0.3))

model.add(LSTM(200))

model.add(Dense(num\_classes, activation='softmax'))

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())

# In[302]:

#Обучение модели

batch\_size = 64

epochs = 30

print(u'Тренируем модель...')

history = model.fit(X\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

validation\_data=(X\_test, y\_test))

# In[303]:

#Оценка модели

score = model.evaluate(X\_test, y\_test,

batch\_size=batch\_size, verbose=1)

print()

print(u'Оценка теста: {}'.format(score[0]))

print(u'Оценка точности модели: {}'.format(score[1]))

# In[304]:

#Получение прогноза от модели

prediction = model.predict(X\_test)

j=0

for j in range(15):

print('Правильная категория: ',np.argmax(y\_test[j])+1)

print("Определенная моделью категория: ",np.argmax(prediction[j])+1)

# ПРИЛОЖЕНИЕ 3

Программа с 4 категориями, с категорией «unknow».

# coding: utf-8

# In[13]:

#Загрузка даных:

from sklearn.datasets import fetch\_20newsgroups

categories = ["comp.graphics","sci.space","rec.sport.baseball"]

df = fetch\_20newsgroups(subset='train', categories=categories)

df1 = fetch\_20newsgroups(subset='test', categories=categories)

# In[14]:

#Функция чистки текста:

import pymorphy2

import re

def clean\_text(text):

morph = pymorphy2.MorphAnalyzer()

text = text.replace("\\", " ").replace(u"╚", " ").replace(u"╩", " ")

text = text.lower()

text = re.sub('\-\s\r\n\s{1,}|\-\s\r\n|\r\n', '', text)

text = re.sub('[.,:;\_%©?\*,!@#$%^&()\d]|[+=]|[[]|[]]|[/]|"|\s{2,}|-', ' ', text)

text = ' '.join(word for word in text.split() if len(word)>3)

text = " ".join(morph.parse(word)[0].normal\_form for word in text.split())

return text

# In[15]:

#Чистка текста и нахождение макс длины текста(или обрезание):

k=0

m=0

# Максимальное количество слов в самом длинном описании заявки

max\_words = 0

for text in df.data:

text=clean\_text(text)

df.data[k]=text

words = len(text.split())

if words > 500:

df.data[k]=text[:500]

words=500

if words > max\_words:

max\_words = words

k+=1

print('Максимальное количество слов в самом длинном описании заявки: {} слов'.format(max\_words))

# In[16]:

#Составление словаря

#import pandas as pd

import numpy as np

import tensorflow as tf

from collections import Counter

vocab = Counter()

for text in df.data:

for word in text.split(' '):

vocab[word.lower()]=1

total\_words = len(vocab)

print('Всего слов в словаре: ', total\_words)

#Функция, которая возвращает индекс слова в словаре

def get\_word\_2\_index(vocab):

word2index = {}

for i,word in enumerate(vocab):

word2index[word.lower()] = i

return word2index

word2index = get\_word\_2\_index(vocab)

#Составление числовых векторов

batches = []

results = []

texts = df.data

for text in texts:

k=0

layer = np.zeros(max\_words,dtype=float)

for word in text.split(' '):

if (vocab[word.lower()])!=0:

layer[k] = word2index[word.lower()]

k+=1

batches.append(layer)

X\_train=np.array(batches)

#print(X\_train[0],X\_train.shape)

y=0

#Добавление 4 категории unknow

num\_classes=4

target=np.zeros(num\_classes)

y\_train=np.zeros((len(df.target),num\_classes))

for у in range(len(df.target)):

#print(y,'= ',df.target[y])

y\_train[y,df.target[y]+1]=1

target[df.target[y]+1]+=1

#print(y,'= ',y\_train[y])

y+=1

print('Распределение по рубликам: ',target)

#print(y\_train[0],y\_train.shape)

# In[39]:

#Подготова текстов, которые не участвовали в составлении словаря

texts = df1.data[0:10]

batches = []

results = []

for text in texts:

k=0

text=clean\_text(text)

words = len(text.split())

if words > 500:

text=text[:500]

words=500

layer = np.zeros(max\_words,dtype=float)

for word in text.split(' '):

if (vocab[word.lower()])!=0:

layer[k] = word2index[word.lower()]

k+=1

batches.append(layer)

X\_t=np.array(batches)

print(len(X\_t[1]))

y=0

num\_classes=4

targ=np.zeros(num\_classes)

y\_t=np.zeros((len(df1.target[0:10]),num\_classes))

for у in range(len(df1.target[0:10])):

#print(y,'= ',df.target[y])

y\_t[y,df1.target[y]+1]=1

targ[df1.target[y]+1]+=1

#print(y,'= ',y\_train[y])

y+=1

print('Распределение по рубликам: ',targ)

# In[20]:

#Разбиение на обучающую и тестовую выборку

n=300

l=len(df.target)

X\_test=X\_train[l-n:l+1]

y\_test=y\_train[l-n:l+1]

y\_train=y\_train[0:l-n]

X\_train=X\_train[0:l-n]

print('Размерность обучающей выборки: ',X\_train.shape)

print('Размерность тестовой выборки: ',X\_test.shape)

# In[55]:

import keras

from keras.models import Sequential

from keras.layers import Dense, Embedding, LSTM, CuDNNLSTM

from keras.layers import Activation

from keras.layers import Dropout,Conv1D,MaxPooling1D

# максимальное количество слов для анализа

max\_features = total\_words

print(u'Собираем модель...')

model = Sequential()

embedding\_vector\_length = 32

#Слой для векторного представления слов

model.add(Embedding(total\_words, embedding\_vector\_length, input\_length=max\_words))

#одномерная свертка 1 слоя

model.add(Conv1D(filters=32, kernel\_size=3, padding='same', activation='relu'))

#oбъединения для временных данных 1 слоя

model.add(MaxPooling1D(pool\_size=2))

#одномерная свертка 2слоя

model.add(Conv1D(filters=32, kernel\_size=4, padding='same', activation='relu'))

#oбъединения для временных данных 2слоя

model.add(MaxPooling1D(pool\_size=2))

model.add(Conv1D(filters=32, kernel\_size=5, padding='same', activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(Conv1D(filters=32, kernel\_size=7, padding='same', activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(Conv1D(filters=32, kernel\_size=9, padding='same', activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(Conv1D(filters=32, kernel\_size=12, padding='same', activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(Conv1D(filters=32, kernel\_size=15, padding='same', activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(LSTM(100))

model.add(keras.layers.Dropout(0.3))

model.add(Dense(num\_classes, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())

# In[ ]:

#Симуляция текстов, которые не содержать ни одного слова в словаре (не относится ни к одной теме)

import random

for j in range(100):

r=random.randint(0, 1474)

X\_train[r]=np.zeros(500)

y\_train[r]=np.zeros(4)

y\_train[r][0]=1

# In[57]:

#Обучение модели

batch\_size = 32

epochs = 10

j=0

print(u'Тренируем модель...')

history = model.fit(X\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

validation\_data=(X\_test, y\_test))

# In[ ]:

#Оценка модели

score = model.evaluate(X\_test, y\_test,

batch\_size=batch\_size, verbose=1)

print()

print(u'Оценка теста: {}'.format(score[0]))

print(u'Оценка точности модели: {}'.format(score[1]))

# In[42]:

#Получение прогноза от модели

import numpy as np

#Симуляция текста, не относится ни к одной категории (случайно 20 слов)

X\_t[5]=np.zeros(500)

j=0

for j in range(20):

r=random.randint(0, maw\_words)

X\_t[5][r]=random.randint(0, total\_words)

prediction = model.predict(X\_t)

j=0

for j in range(10):

print("Текст № ",j+1)

print('Правильная категория: ',np.argmax(y\_t[j]))

print("Определенная моделью категория: ",np.argmax(prediction[j]))

# In[28]:

# Посмотрим на эффективность обучения

import matplotlib.pyplot as plt

get\_ipython().run\_line\_magic('matplotlib', 'inline')

plt.style.use("ggplot")

plt.figure()

N = epochs

plt.plot(np.arange(0, N), history.history["loss"], label="train\_loss")

plt.plot(np.arange(0, N), history.history["val\_loss"], label="val\_loss")

plt.plot(np.arange(0, N), history.history["acc"], label="train\_acc")

plt.plot(np.arange(0, N), history.history["val\_acc"], label="val\_acc")

plt.title("Эффективность обучения")

plt.xlabel("Повторения #")

plt.ylabel("Ошибки")

plt.legend(loc="lower left")