Нейросетевые модели поиска. Часть I

Семантический матчинг

О преподавателях

Иван Быстров

Программист в команде ML технологий движка

Станислав Петров

Программист в Core ML

Содержание раздела

1 Семантический матчинг

2 Переранжирование

3 Векторный поиск

4 Мультимедийный поиск

Содержание лекции

1. Применения нейросетей в поиске
2. Задача текстового ранжирования
3. Эмбеддинги слов и способы их построения
4. Эмбеддинги текстов
5. Использование эмбеддингов для поиска

Применение нейросетей в поиске

Предобработка запросов и исправление опечаток

На предыдущих лекциях уже обсуждали методы исправления ошибок:

- NeuSpell;
- SAGE;

Выделение именованных сущностей.

Spelling correction systems in NeuSpell (Word-Level Accuracy / Correction Rate)

	Synthetic		Natural		Ambiguous	
	WORD-TEST	PROB-TEST	BEA-60K	JFLEG	BEA-4660	BEA-322
ASPELL (Atkinson, 2019)	43.6 / 16.9	47.4 / 27.5	68.0 / 48.7	73.1 / 55.6	68.5 / 10.1	61.1 / 18.9
JAMSPELL (Ozinov, 2019)	90.6 / 55.6	93.5 / 68.5	97.2 / 68.9	98.3 / 74.5	**98.5** / 72.9	**96.7** / 52.3
CHAR-CNN-LSTM (Kim et al., 2015)	97.0 / 88.0	96.5 / 84.1	96.2 / 75.8	97.6 / 80.1	97.5 / 82.7	94.5 / 57.3
SC-LSTM (Sakaguchi et al., 2016)	97.6 / 90.5	96.6 / 84.8	96.0 / 76.7	97.6 / 81.1	97.3 / 86.6	94.9 / 65.9
CHAR-LSTM-LSTM (Li et al., 2018)	98.0 / 91.1	97.1 / 86.6	96.5 / 77.3	97.6 / 81.6	97.8 / 84.0	95.4 / 63.2
BERT (Devlin et al., 2018)	**98.9 / 95.3**	**98.2 / 91.5**	93.4 / 79.1	**97.9 / 85.0**	98.4 / **92.5**	96.0 / **72.1**
SC-LSTM						
+ ELMO (input)	98.5 / 94.0	97.6 / 89.1	96.5 / **79.8**	97.8 / **85.0**	98.2 / 91.9	96.1 / 69.7
+ ELMO (output)	97.9 / 91.4	97.0 / 86.1	**98.0** / 78.5	96.4 / 76.7	97.9 / 88.1	95.2 / 63.2
+ BERT (input)	98.7 / 94.3	97.9 / 89.5	96.2 / 77.0	97.8 / 83.9	98.4 / 90.2	96.0 / 67.8
+ BERT (output)	98.1 / 92.3	97.2 / 86.9	95.9 / 76.0	97.6 / 81.0	97.8 / 88.1	95.1 / 67.2

Model	RUSpellRU			MultidomainGold			MedSpellChecker			GitHubTypoCorpusRu		
	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1	Prec.	Rec.	F1
Yandex.Speller	83.0	59.8	69.5	52.9	51.4	52.2	**80.6**	47.8	60.0	**67.7**	37.5	48.3
JamSpell	42.1	32.8	36.9	25.7	30.6	28.0	24.6	29.7	26.9	49.5	29.9	37.3
Hunspell	31.3	34.9	33.0	16.2	40.1	23.0	10.3	40.2	16.4	28.5	30.7	29.6
gpt-3.5-turbo-0301												
With Punctuation	55.8	75.3	64.1	33.8	72.1	46.0	53.7	66.1	59.3	43.8	57.0	49.6
W/O Punctuation	55.3	75.8	63.9	30.8	70.9	43.0	53.2	67.6	59.6	43.3	56.2	48.9
gpt-4-0314												
With Punctuation	57.0	75.9	65.1	34.0	**73.2**	46.4	54.2	67.7	60.2	44.2	57.4	50.0
W/O Punctuation	56.4	**76.2**	64.8	31.0	72.0	43.3	54.2	69.4	60.9	45.2	**58.2**	51.0
text-davinci-003												
With Punctuation	55.9	75.3	64.2	33.6	72.0	45.8	48.0	66.4	55.7	45.7	57.3	50.9
W/O Punctuation	55.4	75.8	64.0	31.2	71.1	43.4	47.8	68.4	56.3	46.5	58.1	**51.7**
M2M100-1.2B	**88.8**	71.5	**79.2**	**63.8**	61.1	**62.4**	78.8	**71.4**	**74.9**	47.1	42.9	44.9

arXiv:2010.11085, arXiv:2308.09435

Текстовое ранжирование

- Обогащение запросов и документов;
- Построение текстовых признаков.

Method	Dev	Eval
BM25 (Microsoft Baseline)	16.7	16.5
IRNet	27.8	28.1
monoBERT (Jan 2019)	36.5	35.9
Anserini (BM25)	18.7	19.0
+ monoBERT	37.2	36.5
+ monoBERT + duoBERT$_{MAX}$	32.6	-
+ monoBERT + duoBERT$_{MIN}$	37.9	-
+ monoBERT + duoBERT$_{SUM}$	38.2	37.0
+ monoBERT + duoBERT$_{BINARY}$	38.3	-
+ monoBERT + duoBERT$_{SUM}$ + TCP	39.0	37.9
Leaderboard best	39.7	38.3

Table 1: MS MARCO Results.

	MRR@10 Dev	MRR@10 Test	R@1000 Dev	Latency (ms/query)
BM25 (Anserini)	0.184	0.186	0.853	55
doc2query, top-k, 10 samples	0.218	0.215	0.891	61
docTTTTTquery, top-k, 5 samples	0.259	-	0.929	58
docTTTTTquery, top-k, 10 samples	0.265	-	0.939	61
docTTTTTquery, top-k, 20 samples	0.272	-	0.944	62
docTTTTTquery, top-k, 40 samples	0.277	0.272	0.947	64
docTTTTTquery, top-k, 80 samples	0.278	-	0.945	66
DeepCT [2]	0.243	0.239	0.913	55
Best non-ensemble, non-BERT [5]	0.290	0.277	-	-
BM25 + BERT Large [7]	0.375	0.368	0.853	3,500

Table 1: Main results on MS MARCO the passage retrieval task.

arXiv:2301.03266, arXiv:1910.14424

Нейронные сети vs CatBoost

Даже на табличных данных нейросети показывают себя лучше градиентного бустинга, правда при наличии огромных датасетов.

Table 1: For all the evaluation metrics, MMoE outperforms Catboost across all signals.

Signal	Model Type	learning rate	logloss	*ROC-AUC*	*PR-AUC*
Like	MMOE	0.0003	**0.0628**	**0.9420**	**0.4142**
	Catboost	0.01	0.0650	0.9388	0.3922
Share	MMOE	0.0003	**0.0254**	**0.9317**	**0.1332**
	Catboost	0.01	0.0260	0.9236	0.1116
Favorite	MMOE	0.0003	**0.0551**	**0.9145**	**0.1874**
	Catboost	0.01	0.0560	0.9077	0.1700
Video Play	MMOE	0.0003	**0.4029**	**0.8096**	**0.5238**
	Catboost	0.01	0.4290	0.7907	0.4961

Model Type	Dataset Size	Like	Video Play	Favourites	Shares
MMoE	20M	0.933	0.7903	0.8961	0.909
	300M	0.939	0.8048	0.9089	0.9275
	9B	0.9417	0.8065	0.9133	0.9306
Catboost	18M	0.9313	0.7876	0.895	0.9114
	75M	0.9388	0.7907	0.9077	0.9236
	110M (in batches)	0.939	0.7912	0.906	0.9238

arXiv:2312.01760

Задача текстового ранжирования

Текстовое ранжирование

Возвращаемся к работе с текстами.

Задача текстового ранжирования:

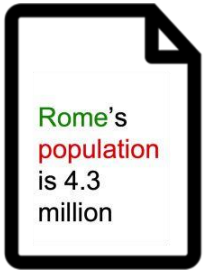
- Есть большой корпус текстовых документов;
- Есть источник текстовых запросов;
- Для каждой пары запрос-документ хотим вычислить некоторый ранк Score(q, d) таким образом, что при сортировке по нему документов максимизируется метрика nDCG@k.

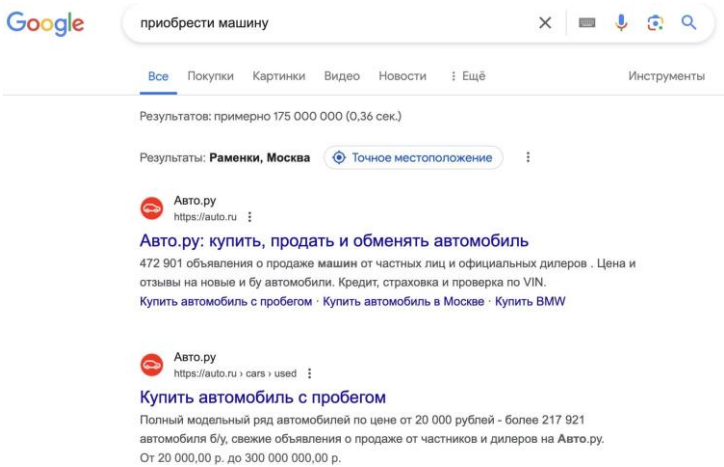
Нашим бейзлайном будет BM25.

Vocabulary mismatch

- Одинаковый по смыслу текстовый запрос можно задать большим количеством различных способов;
- Использование методов текстового ранжирования, основанных на точном совпадении термов не способно преодолеть эту проблему.
- Основные способы решения:
  - Обогащение запросов;
  - Обогащение документов;
  - Построение моделей, способных выделять “смысл” текста.

How many people live in Rome?





Семантический поиск

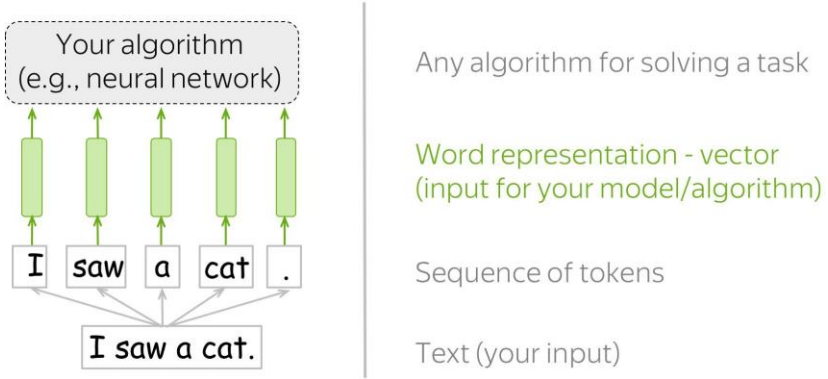
- Это поиск, который сопоставляет запросы и документы с точки зрения их “смысла”.
- Формализовать понятие “смысл текста” очень сложно.
- Один и тот же текстовый запрос может иметь несколько “смыслов”.
- На практике получается обучать такие модели машинного обучения, которые по двум текстам способны оценить их “смысловую” близость.

На запрос “Кто придумал Леголаса” более релевантными будут документы не про персонажа, а про автора, хотя никакая форма слова “Толкиен” в запрос не входит

Эмбеддинги слов и способы их построения

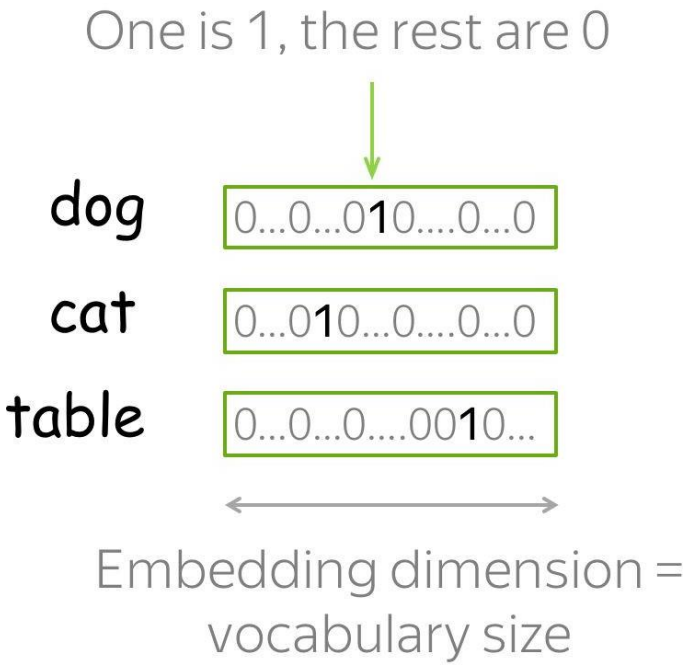
Понятие эмбеддинга слова

- Модели машинного обучения не способны понимать текст так же, как понимает его человек - им нужны числа!
- Представление слова в виде вектора вещественных чисел называется его эмбеддингом.
- Для построения моделей семантического поиска мы хотим, чтобы эмбеддинги слов содержали их смысл.



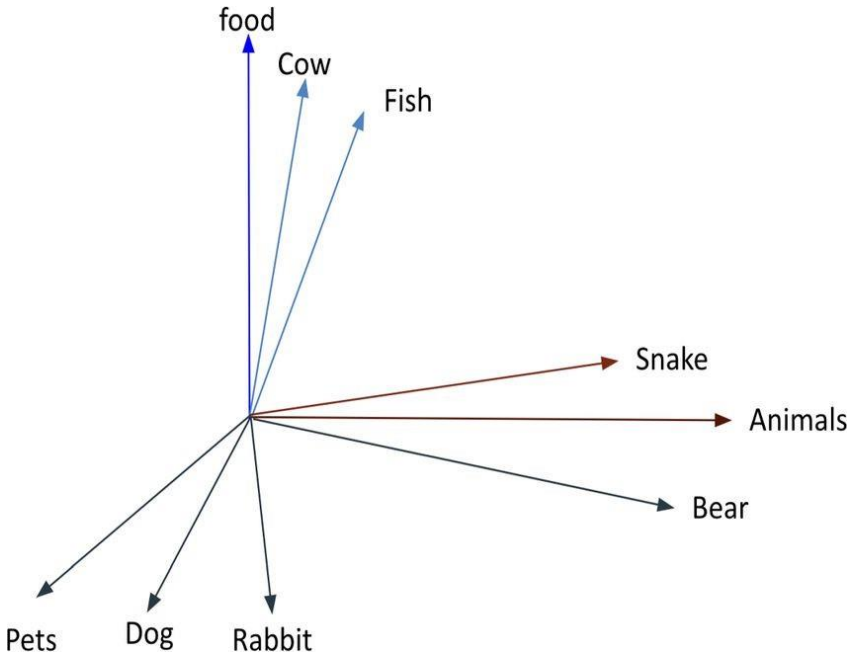
Наивный подход: one-hot вектор

- Пронумеруем все слова в словаре V числами от 0 до |V| - 1.
- Тогда слову с номером i можно поставить в соответствие вектор размерности |V|, содержащий 1 на позиции i и 0 - на всех остальных.
- Есть ли у такого подхода какие-нибудь проблемы?



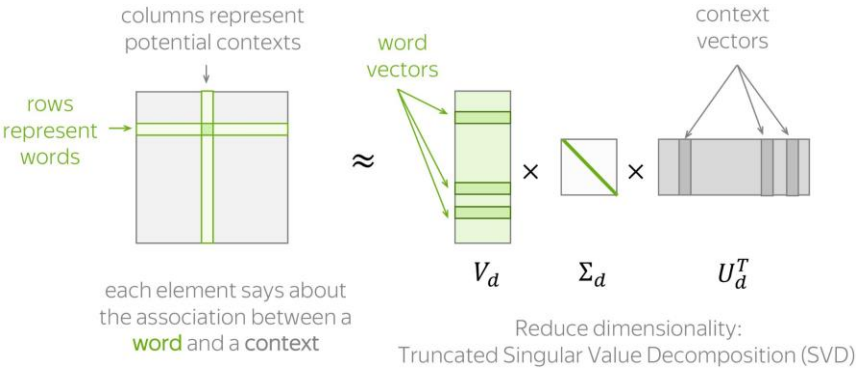
Дистрибутивная гипотеза

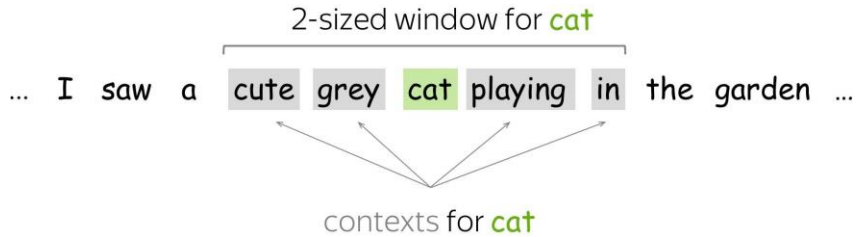
- Дистрибутивная гипотеза заключается в том, что слова часто встречающиеся в одинаковых контекстах имеют похожее значение.
- Таким образом, задачу построения эмбеддингов, содержащих информацию о смысле слова можно свести к задаче построения эмбеддингов, содержащих информацию о контекстах, в которых встречается данное слово.



Count-based методы построения эмбеддингов: совстречаемость

- Составим матрицу совстречаемостей слов из словаря и возможных контекстов;
- Каждый элемент таблицы с индексами i и j содержит количество раз, которое слово w_i встречалось в контексте c_j.
- Применим к данной матрице SVD, одна из матриц будет содержать вектора слов.



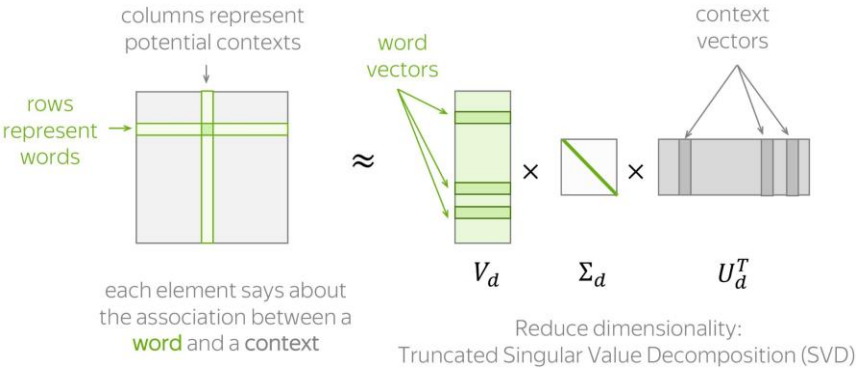


Count-based методы построения эмбеддингов: PPMI

- Кроме количества встречаемостей матрица может содержать и другие статистики взаимоотношения слов и контекстов, например, PPMI:

$$\text{PPMI}(w, c) = \max(0, \text{PMI}(w, c))$$

$$\text{PMI}(w, c) = \log \frac{P(w, c)}{P(w)P(c)} = \log \frac{N(w, c)|(w, c)|}{N(w)N(c)}$$



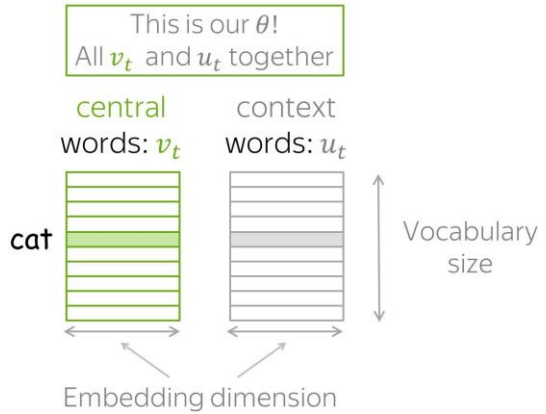
- Можно показать, что при обучении эмбеддингов с помощью Word2Vec происходит неявное приближение разложения похожей матрицы.

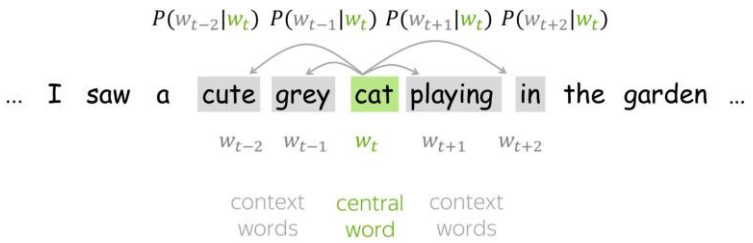
Neural word embedding as implicit matrix factorization

Prediction-based методы: Word2Vec Идея

Основная идея обучения эмбеддингов:

- Для каждого слова из словаря будем хранить 2 вектора - “центральный”, “контекстный”
- Пройдем скользящим окном по большому корпусу текстов;
- Для каждого слова из контекста вычислим вероятность его появления при условии центрального слова;
- Изменим эмбеддинги слов таким образом, чтобы данные вероятности увеличились.





arXiv:1301.3781, arXiv:1310.4546, arXiv:1411.2738

Prediction-based методы: Word2Vec
Обучение

- Мы хотим выучить эмбеддинги слов, которые максимизируют правдоподобие:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m, j \neq 0} P(w_{t+j} | w_t, \theta)$$

$$\text{Loss} = J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m, \\ j \neq 0}} \log P(w_{t+j} | w_t, \theta)$$

agrees with our plan above ⟼ go over text; with a sliding window; compute probability of the context word given the central

- Вероятность слова из контекста при условии центрального:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Dot product: measures similarity of o and c
Larger dot product = larger probability

Normalize over entire vocabulary to get probability distribution

Prediction-based методы: Word2Vec
Обучение

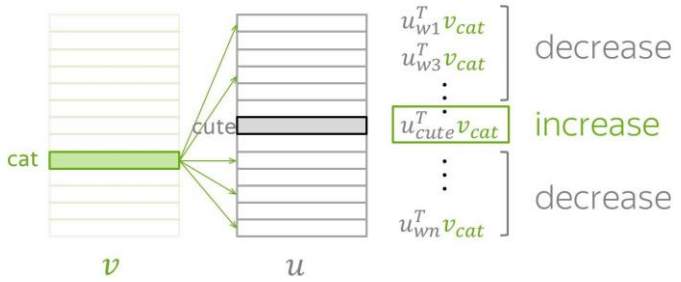
- Будем обучать вектора с помощью градиентного спуска, делая шаг для каждого окна и каждого слова из контекста:

… I saw a cute grey cat playing in the garden …

$$J_{t,j}(\theta) = -\log P(cute|cat) = -\log \frac{\exp u_{cute}^T v_{cat}}{\sum_{w \in Voc} \exp u_w^T v_{cat}} = -u_{cute}^T v_{cat} + \log \sum_{w \in Voc} \exp u_w^T v_{cat}$$

$$v_{cat} := v_{cat} - \alpha \frac{\partial J_{t,j}(\theta)}{\partial v_{cat}}$$

$$u_w := u_w - \alpha \frac{\partial J_{t,j}(\theta)}{\partial u_w} \; \forall w \in V$$



Prediction-based методы: Word2Vec
Семплирование негативов

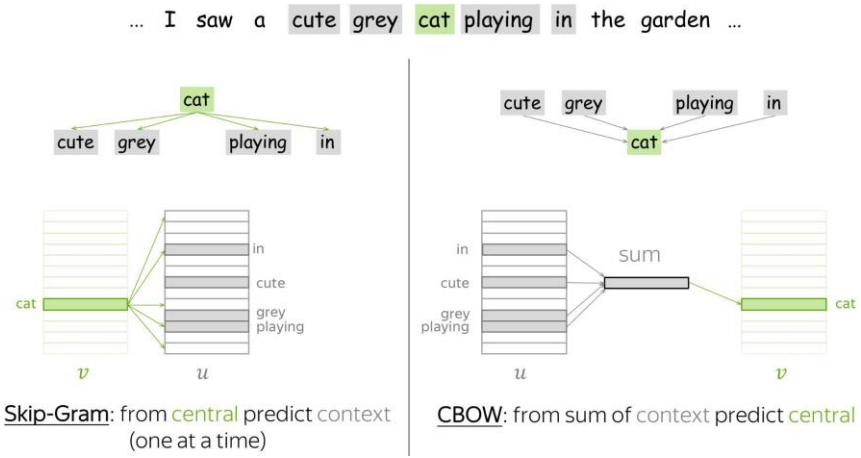
- Обновлять на каждом шаге все контекстные эмбеддинги очень неэффективно. Вместо этого на каждом шаге можем выбирать K векторов для обновления. Тогда функция потерь будет выглядеть так:

$$J_{t,j}(\theta) = -\log \sigma(u_{cute}^T v_{cat}) - \sum_{w \in \{w_{i_1}, \ldots, w_{i_K}\}} \log(1 - \sigma(u_w^T v_{cat}))$$

- Как выбирать негативные примеры?

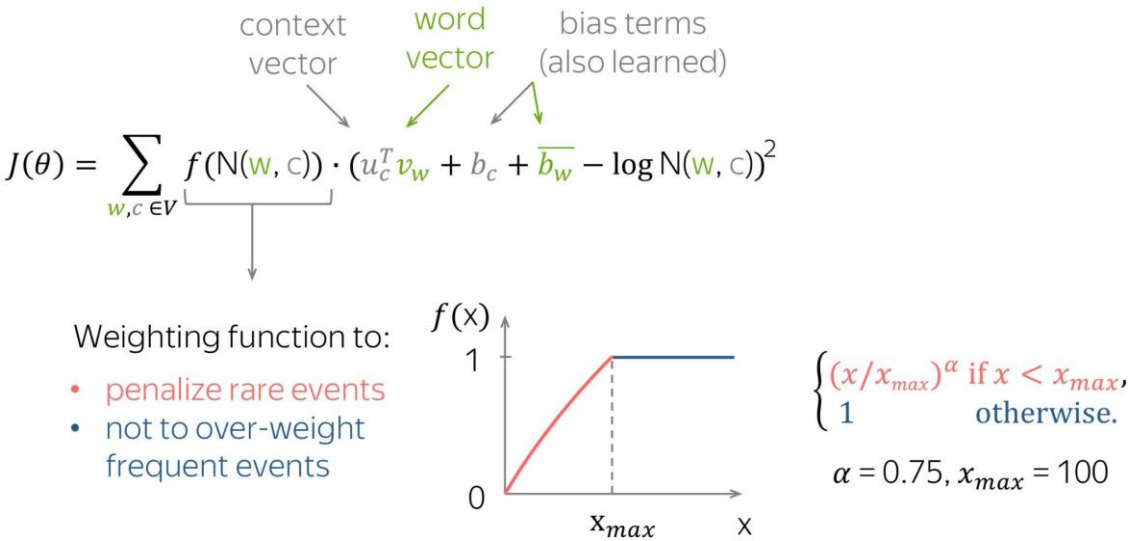
Prediction-based методы: Word2Vec Skip-Gram vs CBOW

- До сих пор мы рассматривали Skip-Gram вариант Word2Vec - он пытается предсказать слова контекста при условии центрального слова.
- Другой вариант CBOW - предсказывает центральное слова при условии слов из контекста.



Prediction-based методы: GloVe

- GloVe представляет из себя комбинацию count-based и prediction-based подходов:



GloVe: Global Vectors for Word Representation

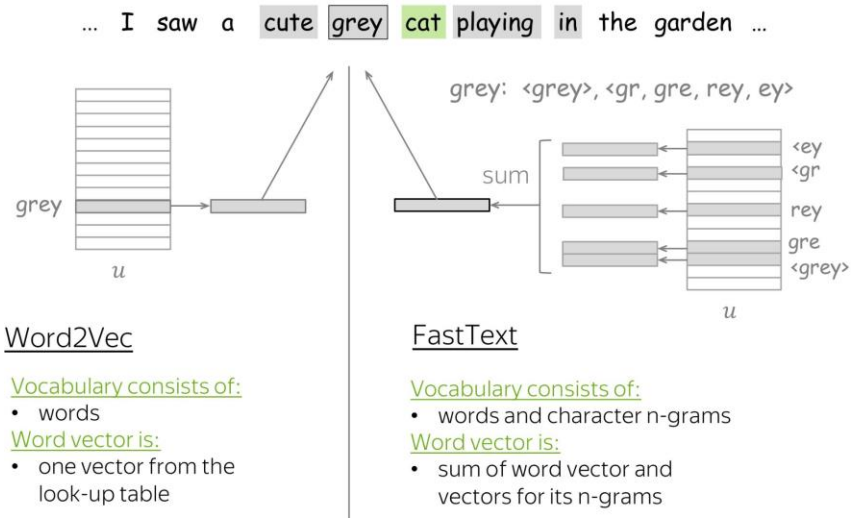
Prediction-based методы: FastText

У Word2Vec есть несколько проблем:

- Обработка слов не из словаря;
- Работа с разными формами слов.

Решение:

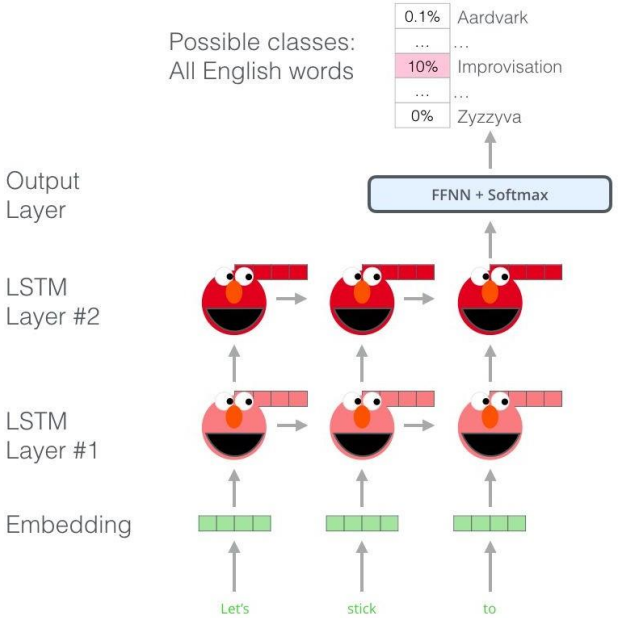
- Добавим в словарь еще и все n-грамы;
- Эмбеддинг слова будем получать как сумму вектора, соответствующего этому слову, и всех входящих в него n-грам.



arXiv:1607.04606

Контекстуализация эмбеддингов. ELMo Embeddings from Language Model

- Мы учим только словарь эмбеддингов - на этапе инференса “stick” (палка, придерживаться) теряет контекст
- Будем учить модель контекстуализировать представления
- biLSTM - двунаправленная LSTM, рекуррентная сеть с памятью
- Претрейн: language modeling (моделирование $p(w_i|w_1, ..., w_{i-1})$)
- Файнтюн: “голова”, решающая конечную задачу

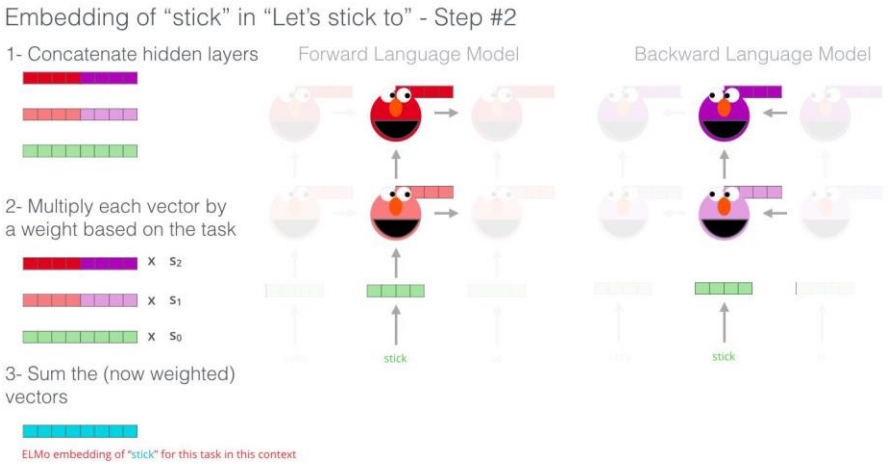


Претрейн: предсказываем следующее слово
“Голова” - MLP+Softmax
Файнтюн строится из тех же принципов

arXiv:1802.05365v2

Контекстуализация эмбеддингов. ELMo Embeddings from Language Model

- Мы учим только словарь эмбеддингов - на этапе инференса “stick” (палка, придерживаться) теряет контекст
- Будем учить модель контекстуализировать представления
- biLSTM - двунаправленная LSTM, рекуррентная сеть с памятью
- Претрейн: language modeling (моделирование $p(w_i|w_1, ..., w_{i-1})$)
- Файнтюн: “голова”, решающая конечную задачу



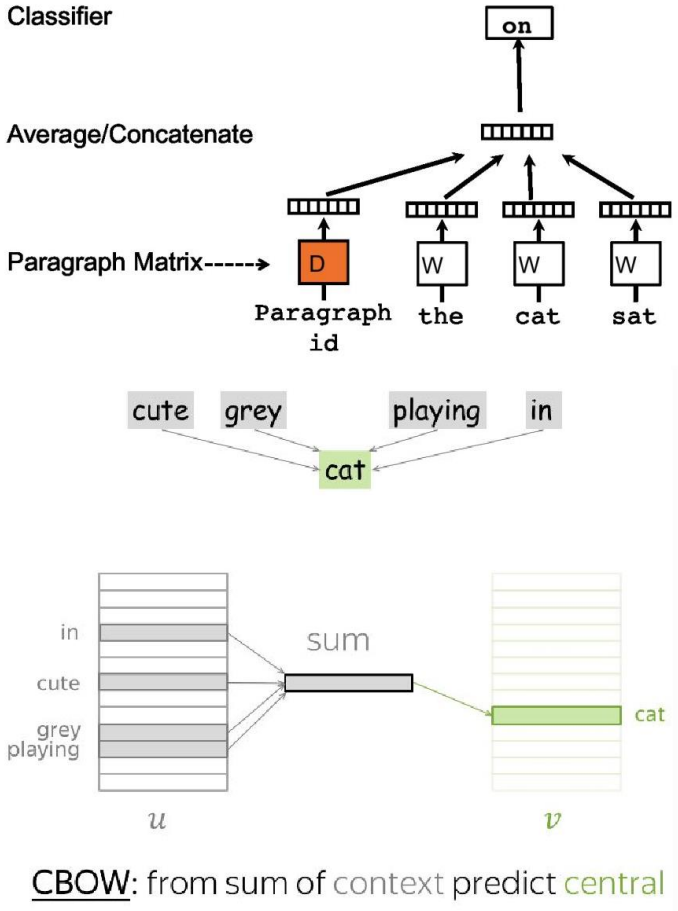
Собираем представление слова
- латентные представления зеркальных слоев
- взвешиваем их детерминировано
- суммируем

arXiv:1802.05365v2

Эмбеддинги текстов

Эмбеддинги текстов: Doc2Vec Distributed memory model

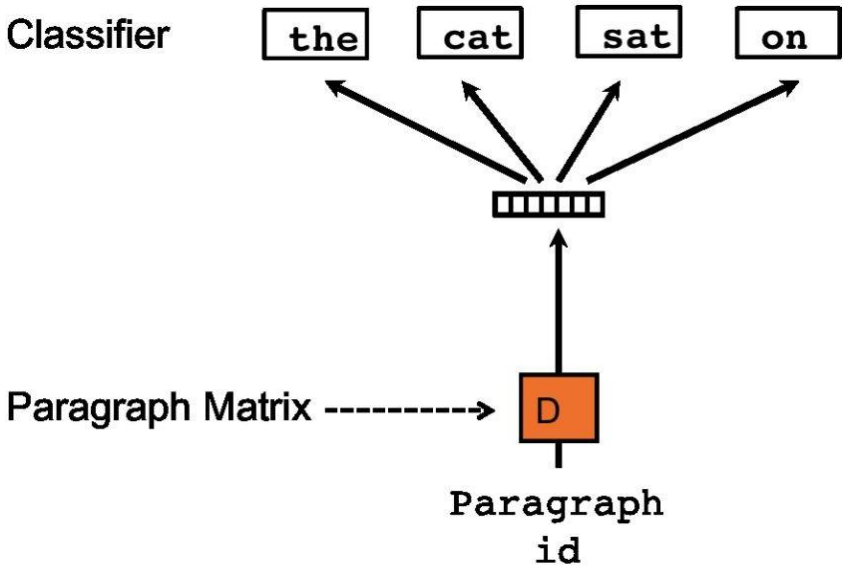
- Кроме эмбеддингов слов будем еще обучать эмбеддинги всех документов трейн датасета.
- Обучение похоже на Word2Vec CBOW с той лишь разницей, что к векторам слов контекста добавляется вектор документа.
- Чтобы получить эмбеддинг для нового документа, нужно проделать ту же самую процедуру, но обучая лишь его эмбеддинг. Вектора слов при этом остаются неизменными.



arXiv:1405.4053v2

Эмбеддинги текстов: Doc2Vec Distributed bag of words

- Будем решать задачу классификации, в которой количество классов равно размеру словаря.
- На каждой итерации обучения будем выбирать случайный набор слов из текущего документа и предсказывать их из его эмбеддинга.



arXiv:1405.4053v2

Эмбеддинги текстов: doc2vec

- Можно использовать один из описанных методов или конкатенировать эмбеддинги, полученные двумя этими методами.
- Результаты показывают, что такой подход получает значительно более качественные эмбеддинги, чем простое усреднение эмбеддингов слов.

Table 1. The performance of our method compared to other approaches on the Stanford Sentiment Treebank dataset. The error rates of other methods are reported in (Socher et al., 2013b).

Model	Error rate (Positive/ Negative)	Error rate (Fine-grained)
Naïve Bayes (Socher et al., 2013b)	18.2 %	59.0%
SVMs (Socher et al., 2013b)	20.6%	59.3%
Bigram Naïve Bayes (Socher et al., 2013b)	16.9%	58.1%
Word Vector Averaging (Socher et al., 2013b)	19.9%	67.3%
Recursive Neural Network (Socher et al., 2013b)	17.6%	56.8%
Matrix Vector-RNN (Socher et al., 2013b)	17.1%	55.6%
Recursive Neural Tensor Network (Socher et al., 2013b)	14.6%	54.3%
Paragraph Vector	**12.2%**	**51.3%**

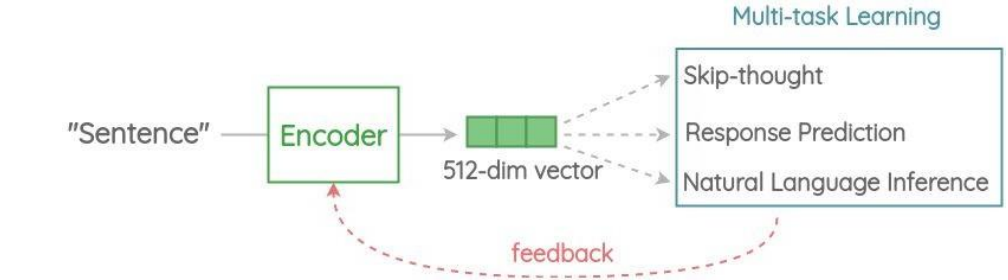
Table 3. The performance of Paragraph Vector and bag-of-words models on the information retrieval task. "Weighted Bag-of-bigrams" is the method where we learn a linear matrix W on TF-IDF bigram features that maximizes the distance between the first and the third paragraph and minimizes the distance between the first and the second paragraph.

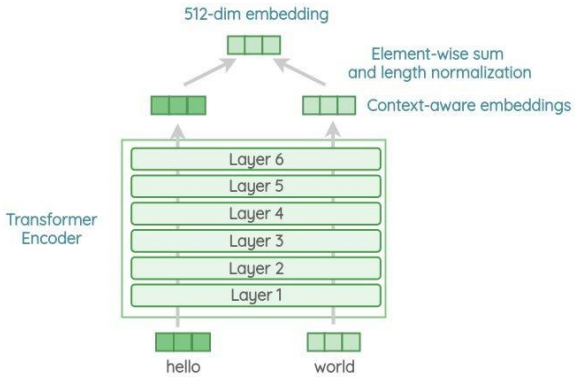
Model	Error rate
Vector Averaging	10.25%
Bag-of-words	8.10 %
Bag-of-bigrams	7.28 %
Weighted Bag-of-bigrams	5.67%
Paragraph Vector	**3.82%**

arXiv:1405.4053v2

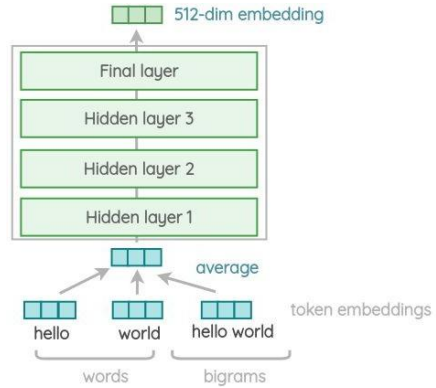
Эмбеддинги текстов: USE Universal Sentence Encoder

- Давайте выучим представления слов, которые одинаково хорошо подходят для **большого количества задач NLP**
- Два варианта архитектуры encoder-модели: BERT-like и MLP (DAN)
- Представление предложения как сумма векторов
- Предвещая T5 (статья)…





BERT-like architecture

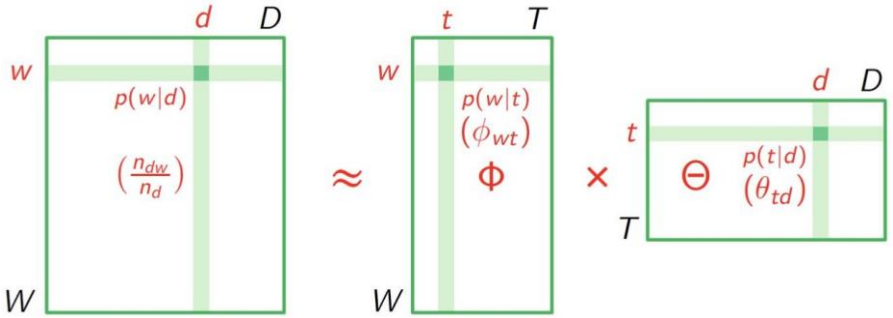


Deep Average Network

arXiv:1803.11175

Эмбеддинги текстов: тематическое моделирование

- Статистический подход к построению представлений документов: распределение слов порождает документы, но так ли это?
- Введем понятие темы как некоторую абстракцию, порождаемую словами и порождающую документы
- Представим матрицу частот в виде произведения двух: Φ, Θ
- Φ - распределение слов внутри тем
- Θ - распределение тем внутри документов
- Число тем - гиперпараметр
- Байесовский вывод: оптимизация Φ, Θ через EM-алгоритм
- Матрица Θ - представления документов



Использование эмбеддингов для поиска

Близость векторов

- Обученные вектора имеют следующее свойство: эмбеддинги близких по смыслу слов располагаются ближе друг к другу.
- Таким образом, мы можем использовать косинусную близость между эмбеддингами запроса и документа в качестве текстового ранка.

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Dual Embedding Space Model (DESM)

- Вычислим эмбеддинг документа как среднее значение нормализованных эмбеддингов входящих в него слов.
- Получим скор релевантности как среднее значение косинусных близостей слов запроса и вектора документа.
- Авторы показывают, что лучше всего работают эмбеддинги, обученные на текстах запросов. Также приносит профит использование контекстных эмбеддингов для документов, вместо центральных.

$$\overline{\mathbf{D}} = \frac{1}{|D|} \sum_{\mathbf{d}_j \in D} \frac{\mathbf{d}_j}{\|\mathbf{d}_j\|}$$

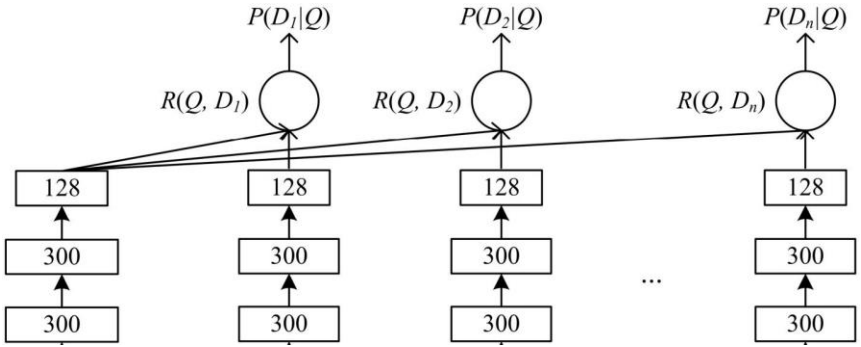
$$DESM(Q, D) = \frac{1}{|Q|} \sum_{q_i \in Q} \frac{\mathbf{q}_i^T \overline{\mathbf{D}}}{\|\mathbf{q}_i\| \|\overline{\mathbf{D}}\|}$$

	Explicitly Judged Test Set		
	NDCG@1	NDCG@3	NDCG@10
BM25	23.69	29.14	44.77
LSA	22.41*	28.25*	44.24*
DESM (IN-IN, trained on body text)	23.59	29.59	45.51*
DESM (IN-IN, trained on queries)	23.75	29.72	46.36*
DESM (IN-OUT, trained on body text)	24.06	30.32*	46.57*
DESM (IN-OUT, trained on queries)	**25.02***	**31.14***	**47.89***

arXiv:1602.01137

Deep Structured Semantic Model (DSSM)

- Возьмем предобученные эмбеддинги слов, например, FastText.
- Применим несколько линейных слоев к эмбеддингам слов запроса.
- Усредним результаты, чтобы получить эмбеддинг запроса.
- Повторим процедуру для документа.
- Чтобы оценить релеватность запроса документу, считаем косинусную близость их эмбеддингов.



$$R(Q,D) = \text{cosine}(y_Q, y_D) = \frac{{y_Q}^T y_D}{\|y_Q\| \|y_D\|}$$

Learning Deep Structured Semantic Models for Web Search using Clickthrough Data

Deep Structured Semantic Model (DSSM)
Обучение

- Обучаем на кликовых данных: для каждого запроса есть список документов и информация о том, кликнул ли пользователь.
- **D** = {D^+, D^-_1, D^-_2, D^-_3, D^-_4} - случайные негативы.
- Hard negatives:
  - Выберем случайное подмножество из всех документов;
  - В качестве негативов возьмем ближайшие к запросу документы.

$$P(D|Q) = \frac{\exp\left(\gamma R(Q,D)\right)}{\sum_{D' \in \boldsymbol{D}} \exp\left(\gamma R(Q,D')\right)}$$

$$L(\Lambda) = -\log \prod_{(Q,D^+)} P(D^+|Q)$$

Полезные ссылки

- В нижней части слайдов приведены ссылки на статьи;
- Для детального погружения в применение нейросетей можно обратиться к Pretrained Transformers for Text Ranking: BERT and Beyond;
- Подробное описание Word2Vec и других методов построения эмбеддингов слов можно найти в NLP Course for You: Word Embeddings;
- Подробнее про RNN: раздел “Models: Reccurent (RNN/LSTM/etc)”.

Спасибо
за внимание!