

# Информационный поиск. NLP для рекомендательных систем.

Хрыльченко Кирилл

Математические методы анализа текстов 2020

24 ноября, 2020

**Задача:** глядя на запрос и множество документов, отобрать для пользователя наиболее релевантные документы.

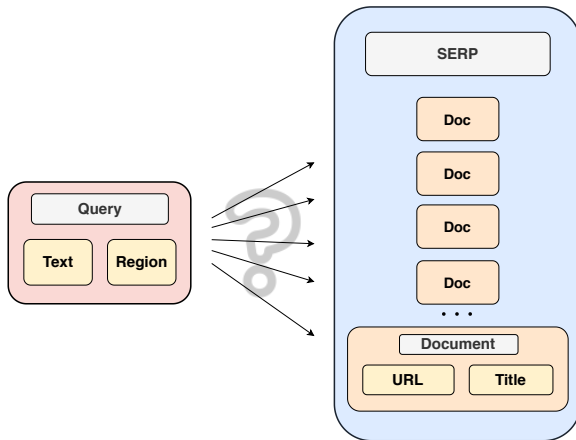
Популярные приложения информационного поиска:

- поиск по базам знаний
- веб-поиск

Методы, основанные на ключевых словах:

- TF-IDF
- BM-25

- Search Engine Results Pages (SERP) — поисковая выдача



×



⋮ More

## Tools

## Query

About 71,600 results (0.77 seconds)

— **Сколько нужно полицейских, чтобы вкрутить лампочку?** — Десять: один стоит на столе и держит **лампочку**, четверо крутят стол по часовой стрелке, а остальные пятеро ходят вокруг стола в противоположную сторону, чтобы у первого не закружилась голова.

[elektrik.info](#) > [6-skolko-ljudej-nuzhno-chtoby-zamenit](#)

Сколько людей нужно, чтобы заменить лампочку » Сайт ...

?

### About Featured Snippets



Feedback

[e-smirnov.livejournal.com](#) > ... [▼ Translate this page](#)

Сколько человек нужно, чтобы вкрутить одну лампочку? Ну ...

Апр 27, 2005 — **Сколько человек нужно, чтобы вкрутить одну лампочку?** Ну, все вы помните кучу анекдотов на эту тему. Недавно я услышал очередную ...

[pikabu.ru](#) > [Общество](#) ▾ [Translate this page](#)

Шутки про лампочку | Пикабу

Document Title

Jun 9, 2012 — Сколько барабанщиков нужно, чтобы вкрутить лампочку? ... Сколько обычных скучных людей требуется, чтобы поменять лампочку?

# Deep Structured Semantic Models<sup>1</sup>

По аналогии с word2vec, хочется видеть схожесть даже при отсутствии общих слов:

- запрос: univ of penn
- заголовок документа: university of pennsylvania wikipedia the free encyclopedia

Как можно это побороть:

- и запрос, и документ переводятся из мешка слов в вектор из «семантического пространства»
- релевантность документа для запроса определяется как косинус между соответствующими векторами

---

<sup>1</sup><https://www.microsoft.com/en-us/research/publication/learning-deep-structured-semantic-models-for-web-search-using-clickthrough-data/>

# Word Hashing

Размерности словарей:

- в словах: 500000. Слишком много
- в буквенных триграммах: 30621

Будем переводить слова (тексты) в векторы побуквенных триграмм:

- good  $\rightarrow$  #good#  $\rightarrow$  (#go, goo, ood, od#)

и представлять их как мешок, т.е. вектор размерности 30621.

Возникают коллизии — разные слова представляются одинаковыми мешками побуквенных триграмм. Но их не очень много:

Word Size	Letter-Bigram		Letter-Trigram	
	Token Size	Collision	Token Size	Collision
40k	1107	18	10306	2
500k	1607	1192	30621	22

**Table 1:** Word hashing token size and collision numbers as a function of the vocabulary size and the type of letter ngrams.

Из плюсов — более менее решили проблему out-of-vocabulary слов.

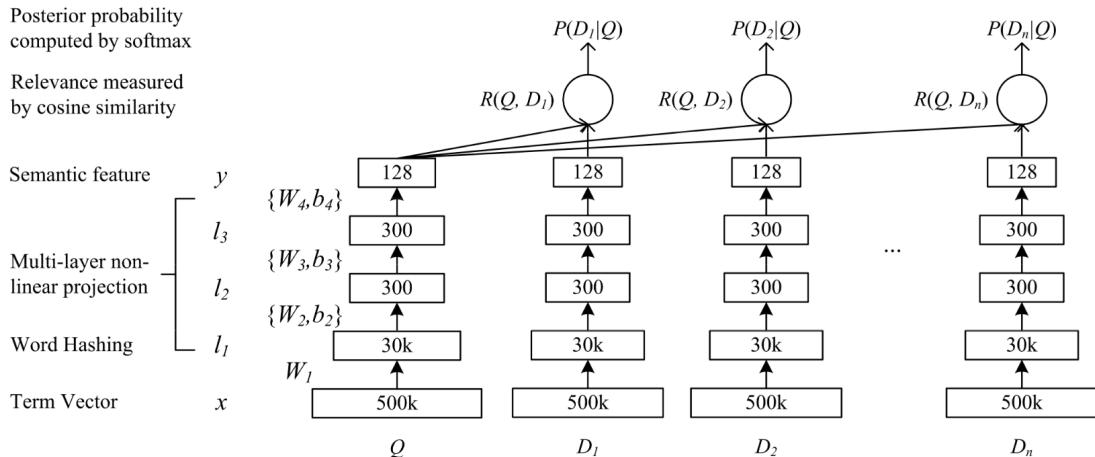
# Word Hashing

Bag of letter trigrams				
Letter trigram	trigram 1	...	trigram 30620	trigram 30621
Count	4	...	3	0



Bag of words							
Word	word 1	word 2	word 3	word 4	...	word 499 999	word 500 000
Count	0	1	3	0	...	1	0

# DSSM. Архитектура



**Figure 1:** Illustration of the DSSM. It uses a DNN to map high-dimensional sparse text features into low-dimensional dense features in a semantic space. The first hidden layer, with 30k units, accomplishes word hashing. The word-hashed features are then projected through multiple layers of non-linear projections. The final layer's neural activities in this DNN form the feature in the semantic space.



Линейные слои:

- перевод из мешка слов в мешок триграмм с помощью умножения на матрицу
- три линейных слоя с  $\tanh$  в качестве функции активации:  $30621 \rightarrow 300 \rightarrow 300 \rightarrow 128$
- проводим через эту сеть текст запроса и документа, считаем косинус
- это двухбашенная архитектура — один раз считаем вектор для запроса

Обучение:

- максимизируем правдоподобие пар (запрос, кликнутый документ)
- негативное сэмплирование по аналогии с word2vec: на каждую пару (запрос, кликнутый документ) берется 4 случайных документа для формирования отрицательных примеров
- SGD, размер батча — 1024

Использование — скорим каждую пару и сортируем по косинусной близости

#	Models	NDCG@1	NDCG@3	NDCG@10
1	TF-IDF	0.319	0.382	0.462
2	BM25	0.308	0.373	0.455
3	WTM	0.332	0.400	0.478
4	LSA	0.298	0.372	0.455
5	PLSA	0.295	0.371	0.456
6	DAE	0.310	0.377	0.459
7	BLTM-PR	0.337	0.403	0.480
8	DPM	0.329	0.401	0.479
9	DNN	0.342	0.410	0.486
10	L-WH linear	0.357	0.422	0.495
11	L-WH non-linear	0.357	0.421	0.494
12	<b>L-WH DNN</b>	<b>0.362</b>	<b>0.425</b>	<b>0.498</b>

**Table 2:** Comparative results with the previous state of the art approaches and various settings of DSSM.

## Convolutional DSSM<sup>2</sup>

Проблема — всё еще рассматриваем запросы и документы как мешки слов в DSSM, не учитывается порядок слов. Модель не способна уловить разницу в значении слова «office» для запросов:

- office excel
- apartment office

Решение — будем смотреть на n-граммы из слов с помощью сверток

---

<sup>2</sup>[https:](https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/cikm2014_cdssm_final.pdf)

[//www.microsoft.com/en-us/research/wp-content/uploads/2016/02/cikm2014\\_cdssm\\_final.pdf](https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/cikm2014_cdssm_final.pdf)

# CDSSM. Архитектура

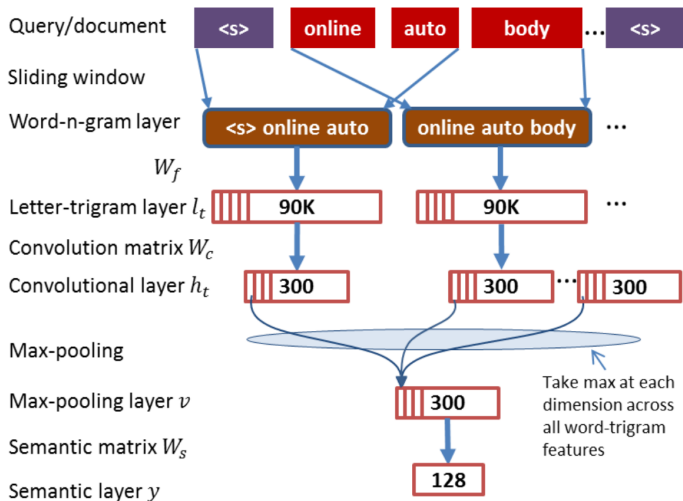


Figure: Архитектура CDSSM с триграммной сверткой.

В начале и конце текста добавляется специальный токен  $\langle s \rangle$ , затем:

- для каждой n-граммы формируется векторное представление
  - ① каждое слово в n-грамме переводим в посимвольный триграммный вектор
  - ② конкатенируем триграммные векторы по всем словам n-граммы
  - ③ применяем линейный слой (свертку) сверху этой конкатенации с последующим гиперболическим тангенсом
- по всем n-граммам берется max-pooling
- линейный слой с tanh

#	Models	NDCG @1	NDCG @3	NDCG @10
1	DSSM ( $J = 50$ )	0.327	0.363	0.438
2	CLSM ( $J = 50$ ) win =1	0.340 <sup><math>\alpha</math></sup>	0.374 <sup><math>\alpha</math></sup>	0.443 <sup><math>\alpha</math></sup>
3	CLSM ( $J = 50$ ) win =3	0.348 <sup><math>\alpha\beta</math></sup>	0.379 <sup><math>\alpha\beta</math></sup>	0.449 <sup><math>\alpha\beta</math></sup>
4	CLSM ( $J = 50$ ) win =5	0.344 <sup><math>\alpha</math></sup>	0.376 <sup><math>\alpha</math></sup>	0.448 <sup><math>\alpha\beta</math></sup>

Figure: Сравнение CDSSM с прошлой SOTA моделью.

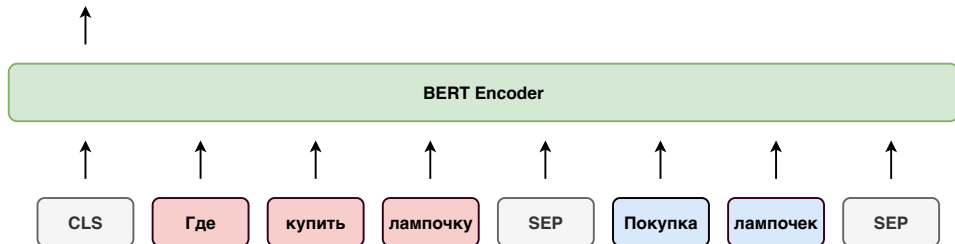
- Обнаружили, что лучше обучать разные сети для запроса и для документа
- Пробовали использовать «тело» документа, вместо заголовка, получили качество значительно хуже

# BERT

Основные проблемы DSSM, CDSSM и других «двухбашенных» моделей:

- только частично учитываем порядок слов
- векторы для документа и запроса формируются независимо<sup>3</sup>

В 2019-м году Google внедрил в поиск BERT<sup>4</sup>:



<sup>3</sup><https://arxiv.org/abs/1711.08611>

<sup>4</sup><https://arxiv.org/pdf/1901.04085.pdf>

# Персонализация веб-поиска. Context-Aware Recommender System<sup>5</sup>

Используем не только последний запрос, но также пользовательскую историю — его прошлые запросы и клики.

Модель **CARS** решает сразу две задачи:

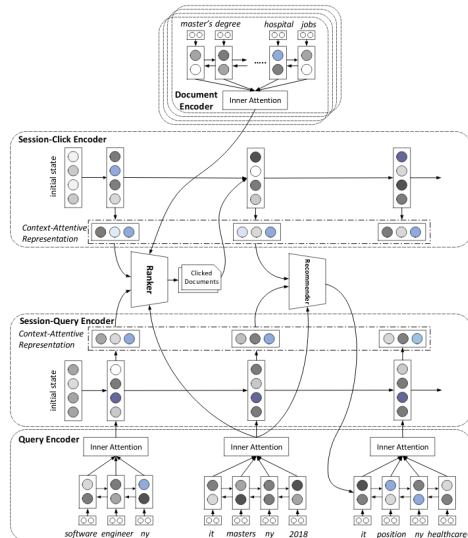
- ранжирует документы
- генерирует саджест для следующего запроса

---

<sup>5</sup><https://arxiv.org/pdf/1906.02329.pdf>



- BiLSTM по словам запроса с inner-attention, формирует векторы запросов
- BiLSTM по векторам запросов
- аналогично две BiLSTM для кликнувших документов
- LSTM — декодер для next query suggestion



Model	MAP	MRR	NDCG		
			@1	@3	@10
Traditional IR-models					
BM25 [40]	0.230	0.206	0.206	0.269	0.319
QL [39]	0.195	0.166	0.166	0.213	0.276
FixInt [43]	0.242	0.224	0.212	0.275	0.332
Single-task Learning					
DRMM [14]	0.201	0.228	0.129	0.223	0.264
DSSM [18]	0.283	0.307	0.188	0.231	0.341
CLSM [44]	0.313	0.341	0.205	0.252	0.373
ARC-I [16]	0.401	0.411	0.259	0.374	0.463
ARC-II [16]	0.455	0.465	0.309	0.434	0.521
DUET [33]	0.479	0.490	0.332	0.462	0.546
Match Tensor [19]	0.481	0.501	0.345	0.472	0.555
Multi-task Learning					
M-NSRF [2]	0.491	0.502	0.348	0.474	0.557
M-Match Tensor [2]	0.505	0.518	0.368	0.491	0.567
CARS	<b>0.531</b>	<b>0.542</b>	<b>0.391</b>	<b>0.517</b>	<b>0.596</b>

Figure: Метрики на датасете AOL.

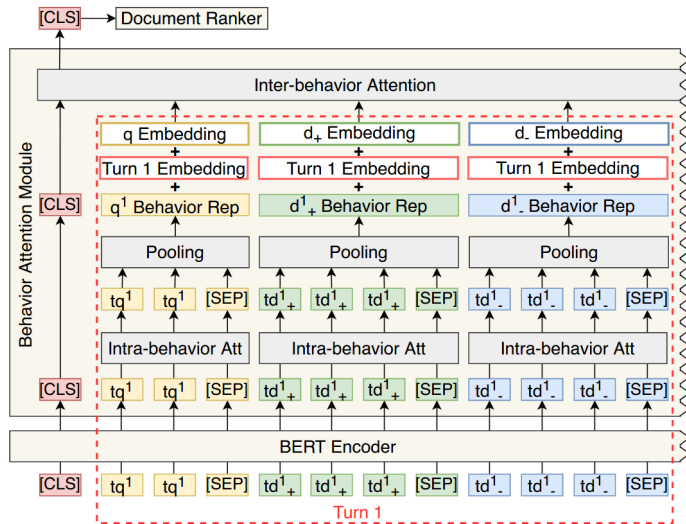
# Behavior Aware Transformer (HBA) <sup>6</sup>

Формируем последовательность:

- 1 запрос
- 2 кликнувший док
- 3 пропущенный док

BERT применяется иерархически:

- 1 ко всему тексту
- 2 к последовательности закодированных событий



<sup>6</sup><http://ciir-publications.cs.umass.edu/getpdf.php?id=1383>

## HBA. Результаты

Models	MRR	nDCG		
		@1	@3	@10
CARS <sup>1</sup> [3]	0.4538	0.2940	0.4249	0.5109
BERT [12]	0.5198	0.3592	0.4984	0.5813
<b>BERT-Concat-Q</b>	0.5196	0.3596	0.4977	0.5806
<b>BERT-Concat-QC</b>	0.5340	0.3759	0.5149	0.5934
<b>BERT-Concat-QCS</b>	0.5366	0.3787	0.5174	0.5954
<b>HBA-Transformers-QC</b>	<b>0.5450<sup>‡</sup></b>	<b>0.3866<sup>‡</sup></b>	<b>0.5291<sup>‡</sup></b>	<b>0.6021<sup>‡</sup></b>
<b>HBA-Transformers-QCS</b>	0.5446 <sup>‡</sup>	0.3850 <sup>‡</sup>	0.5268 <sup>‡</sup>	0.6012 <sup>‡</sup>

Figure: Метрики на датасете AOL. BERT-Concat — обычный BERT над текстом.

## Session-based recommendations

Нас интересуют два типа ранжирования:

- pointwise — рекомендуемые продукты оцениваются независимо друг от друга
- pairwise — при обучении пытаемся увеличивать разницу между скором положительного и отрицательного сэмпла, например:

$$\log \sigma(r_+ - r_-) \rightarrow \max$$

Пусть:

- $U$  — множество продуктов, с которыми взаимодействует пользователь
- $u_1, \dots, u_T$  — пользовательская сессия взаимодействий с продуктами

Тогда можно учить «языковую» модель  $P(u_t | u_1, \dots, u_{t-1})$  для «языка», в котором словами являются продукты, а документы — это истории пользователей.

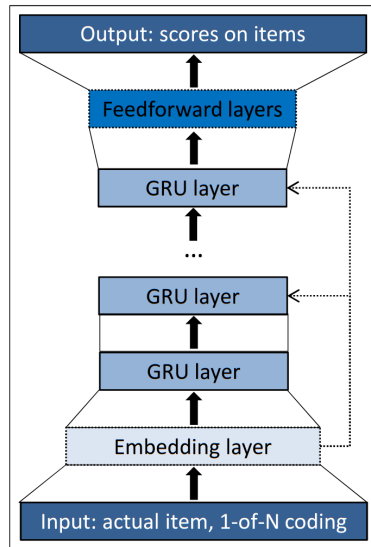
Используем негативное семплирование — учим pointwise ранжирование.

## Обучение:

- используют попарное ранжирование
- утверждают, что при pointwise ранжировании модель хуже сходится

Table 3: Recall@20 and MRR@20 for different types of a single layer of GRU, compared to the best baseline (item-KNN). Best results per dataset are highlighted.

Loss / #Units	RSC15		VIDEO	
	Recall@20	MRR@20	Recall@20	MRR@20
TOP1 100	0.5853 (+15.55%)	0.2305 (+12.58%)	0.6141 (+11.50%)	0.3511 (+3.84%)
BPR 100	0.6069 (+19.82%)	0.2407 (+17.54%)	0.5999 (+8.92%)	0.3260 (-3.56%)
Cross-entropy 100	0.6074 (+19.91%)	0.2430 (+18.65%)	0.6372 (+15.69%)	0.3720 (+10.04%)
TOP1 1000	0.6206 (+22.53%)	<b>0.2693 (+31.49%)</b>	<b>0.6624 (+20.27%)</b>	<b>0.3891 (+15.08%)</b>
BPR 1000	<b>0.6322 (+24.82%)</b>	0.2467 (+20.47%)	0.6311 (+14.58%)	0.3136 (-7.23%)
Cross-entropy 1000	0.5777 (+14.06%)	0.2153 (+5.16%)	—	—



<sup>7</sup><https://arxiv.org/pdf/1511.06939.pdf>

SASRec<sup>8</sup> — однонаправленный декодер трансформера (аналогично GPT), существенно обогнал прошлые SOTA модели.

BERT4Rec:

- модель: энкодер трансформера
- задача: маскируют и учатся восстанавливать взаимодействия в пользовательской истории — Cloze task, masked language modeling
- применяют без дообучения, добавляют <MASK> в конце истории

---

<sup>8</sup><https://arxiv.org/pdf/1808.09781.pdf>

<sup>9</sup><https://arxiv.org/pdf/1904.06690.pdf>

# BERT4Rec. Результаты

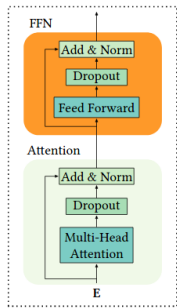
Datasets	Metric	POP	BPR-MF	NCF	FPMC	GRU4Rec	GRU4Rec <sup>+</sup>	Caser	SASRec	BERT4Rec	Improv.
Beauty	HR@1	0.0077	0.0415	0.0407	0.0435	0.0402	0.0551	0.0475	<u>0.0906</u>	<b>0.0953</b>	5.19%
	HR@5	0.0392	0.1209	0.1305	0.1387	0.1315	0.1781	0.1625	<u>0.1934</u>	<b>0.2207</b>	14.12%
	HR@10	0.0762	0.1992	0.2142	0.2401	0.2343	0.2654	0.2590	<u>0.2653</u>	<b>0.3025</b>	14.02%
	NDCG@5	0.0230	0.0814	0.0855	0.0902	0.0812	0.1172	0.1050	<u>0.1436</u>	<b>0.1599</b>	11.35%
	NDCG@10	0.0349	0.1064	0.1124	0.1211	0.1074	0.1453	0.1360	<u>0.1633</u>	<b>0.1862</b>	14.02%
	MRR	0.0437	0.1006	0.1043	0.1056	0.1023	0.1299	0.1205	<u>0.1536</u>	<b>0.1701</b>	10.74%
Steam	HR@1	0.0159	0.0314	0.0246	0.0358	0.0574	0.0812	0.0495	<u>0.0885</u>	<b>0.0957</b>	8.14%
	HR@5	0.0805	0.1177	0.1203	0.1517	0.2171	0.2391	0.1766	<u>0.2559</u>	<b>0.2710</b>	5.90%
	HR@10	0.1389	0.1993	0.2169	0.2551	0.3313	0.3594	0.2870	<u>0.3783</u>	<b>0.4013</b>	6.08%
	NDCG@5	0.0477	0.0744	0.0717	0.0945	0.1370	0.1613	0.1131	<u>0.1727</u>	<b>0.1842</b>	6.66%
	NDCG@10	0.0665	0.1005	0.1026	0.1283	0.1802	0.2053	0.1484	<u>0.2147</u>	<b>0.2261</b>	5.31%
	MRR	0.0669	0.0942	0.0932	0.1139	0.1420	0.1757	0.1305	<u>0.1874</u>	<b>0.1949</b>	4.00%
ML-1m	HR@1	0.0141	0.0914	0.0397	0.1386	0.1583	0.2092	0.2194	<u>0.2351</u>	<b>0.2863</b>	21.78%
	HR@5	0.0715	0.2866	0.1932	0.4297	0.4673	0.5103	0.5353	<u>0.5434</u>	<b>0.5876</b>	8.13%
	HR@10	0.1358	0.4301	0.3477	0.5946	0.6207	0.6351	<u>0.6692</u>	0.6629	<b>0.6970</b>	4.15%
	NDCG@5	0.0416	0.1903	0.1146	0.2885	0.3196	0.3705	0.3832	<u>0.3980</u>	<b>0.4454</b>	11.91%
	NDCG@10	0.0621	0.2365	0.1640	0.3439	0.3627	0.4064	0.4268	<u>0.4368</u>	<b>0.4818</b>	10.32%
	MRR	0.0627	0.2009	0.1358	0.2891	0.3041	0.3462	0.3648	<u>0.3790</u>	<b>0.4254</b>	12.24%
ML-20m	HR@1	0.0221	0.0553	0.0231	0.1079	0.1459	0.2021	0.1232	<u>0.2544</u>	<b>0.3440</b>	35.22%
	HR@5	0.0805	0.2128	0.1358	0.3601	0.4657	0.5118	0.3804	<u>0.5727</u>	<b>0.6323</b>	10.41%
	HR@10	0.1378	0.3538	0.2922	0.5201	0.5844	0.6524	0.5427	<u>0.7136</u>	<b>0.7473</b>	4.72%
	NDCG@5	0.0511	0.1332	0.0771	0.2239	0.3090	0.3630	0.2538	<u>0.4208</u>	<b>0.4967</b>	18.04%
	NDCG@10	0.0695	0.1786	0.1271	0.2895	0.3637	0.4087	0.3062	<u>0.4665</u>	<b>0.5340</b>	14.47%
	MRR	0.0709	0.1503	0.1072	0.2273	0.2967	0.3476	0.2529	<u>0.4026</u>	<b>0.4785</b>	18.85%



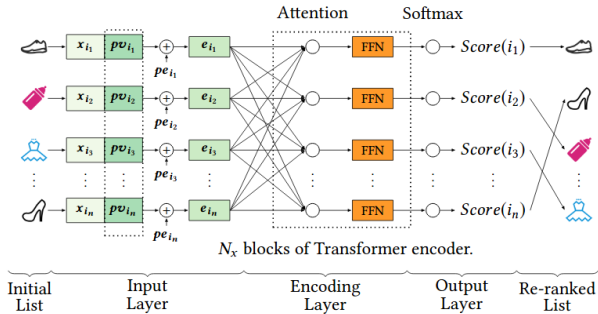
# BERT4Rec. Ablation Study

Architecture	Dataset			
	Beauty	Steam	ML-1m	ML-20m
$L = 2, h = 2$	0.1832	0.2241	0.4759	0.4513
w/o PE	0.1741	0.2060	0.2155↓	0.2867↓
w/o PFFN	0.1803	0.2137	0.4544	0.4296
w/o LN	0.1642↓	0.2058	0.4334	0.4186
w/o RC	0.1619↓	0.2193	0.4643	0.4483
w/o Dropout	0.1658	0.2185	0.4553	0.4471
1 layer ( $L = 1$ )	0.1782	0.2122	0.4412	0.4238
3 layers ( $L = 3$ )	<b>0.1859</b>	<b>0.2262</b>	<b>0.4864</b>	<b>0.4661</b>
4 layers ( $L = 4$ )	<b>0.1834</b>	<b>0.2279</b>	<b>0.4898</b>	<b>0.4732</b>
1 head ( $h = 1$ )	<b>0.1853</b>	0.2187	0.4568	0.4402
4 heads ( $h = 4$ )	0.1830	<b>0.2245</b>	<b>0.4770</b>	<b>0.4520</b>
8 heads ( $h = 8$ )	0.1823	<b>0.2248</b>	0.4743	<b>0.4550</b>

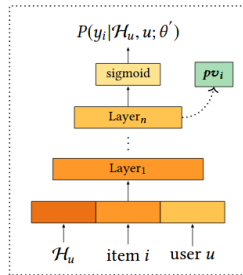
# Personalized Re-ranking Model<sup>10</sup>



(a) One block of Transformer encoder.



(b) Architecture of PRM.



(c) The pre-trained model to generate  $p v_i, i = i_1, \dots, i_n$ .

<sup>10</sup><https://arxiv.org/pdf/1904.06813.pdf>

## PRM. Результаты

Init. List	Reranking	Yahoo Letor dataset.				
		Precision@5(%)	Precision@10(%)	MAP@5(%)	MAP@10(%)	MAP(%)
SVMRank	SVMRank	50.42	42.25	73.71	68.28	62.14
	LambdaMART	51.35	43.08	74.94	69.54	63.38
	DLCM	52.54	43.26	76.52	70.86	64.50
	PRM-BASE	<b>53.29</b>	<b>43.66</b>	<b>77.62</b>	<b>72.02</b>	<b>65.60</b>
LambdaMART	SVMRank	50.41	42.34	73.82	68.27	62.13
	LambdaMART	52.04	43.00	75.77	70.49	64.04
	DLCM	52.54	43.16	77.81	71.88	65.24
	PRM-BASE	<b>53.63</b>	<b>43.41</b>	<b>78.62</b>	<b>72.67</b>	<b>65.72</b>

Figure: Сравнение моделей на датасете Yahoo Letor.

# Time-Aware User Embeddings as a Service<sup>11</sup>

Действия пользователя происходят неравномерно:

- $a_1, \dots, a_T$  — последовательность действий пользователя, где  $a \in A$  — конечное множество возможных действий пользователя
- каждому действию пользователя  $a \in A$  сопоставляется обучаемый эмбединг  $v \in \mathbb{R}^d$
- $t_1, \dots, t_T$  — таймстемпы действий

Сделаем следующие преобразования:

$$\begin{aligned}\tau_j &= \frac{t_j}{t_T} \\ \sigma_j &= \sigma(\theta_j + \mu_j \tau_j) \\ \hat{v}_j &= \sigma_j v_j,\end{aligned}$$

где  $v_j$  — исходный вектор действия  $a_j$ .

---

<sup>11</sup><https://research.yahoo.com/publications/9272/time-aware-user-embeddings-service>

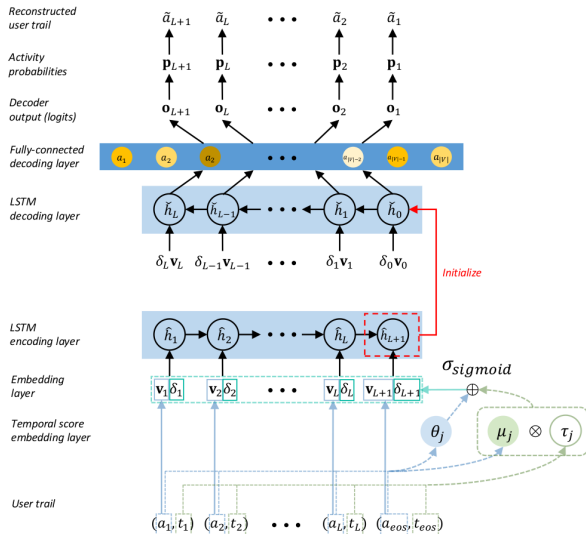
# TASA. Архитектура

## Обучение:

- кодируем исходную последовательность событий в вектор
- реконструируем исходную последовательность

Model	Rec. Acc.	BLEU <sub>n</sub>		ROUGE <sub>n</sub>		
		n = 1	n = 2	n = 1	n = 2	n = w
AE	0.0136	0.0136	0.0085	0.0122	0.0082	0.0136
seq2seq	0.1235	0.2396	0.0709	0.2551	0.0742	0.2254
TA-seq2seq	0.1725	0.2664	0.0936	0.2807	0.0965	0.2527
ISA	0.1979	0.2535	0.0927	0.2851	0.0991	0.2464
TASA	0.5244	0.5500	0.3952	0.5691	0.4012	0.5441

Figure: Качество реконструкции на датасете RecSys 2015.



# TASA. Результаты

Supervised Task	LR(AE)	LR(seq2seq)	LR(TA-seq2seq)	LR(ISA)	LR(TASA)	LR 1-hot	attRNN
RecSys 2015 Challenge	0.5555	0.6022	0.7523	0.7420	0.7563	0.7277	0.7591

