



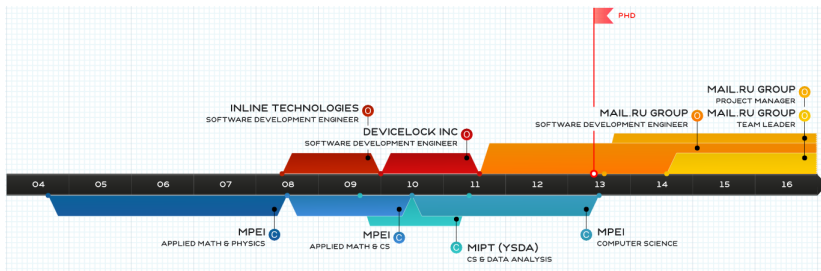
# ТЕХНОСФЕРА

## Лекция Текстовая релевантность

Владимир Гулин

сентябрь 2019 г.

# Владимир Гулин



e-mail: [gulin.vladimir.sfera@mail.ru](mailto:gulin.vladimir.sfera@mail.ru)

тел.: +7 (915) 416-95-75

# Структура курса

1. Текстовая релевантность 2.0. Unsupervised learning to match. *H*
2. Оценка качества поиска
3. Ссылочное ранжирование *H*
4. Learning to rank *H*
5. Поведенческое ранжирование
6. Сглаживание поведенческих факторов *H*
7. Learning to match *H*
8. Индексные структуры данных в эпоху нейросетей
9. Мультимедийный поиск
10. Conversational systems

# Контроль знаний

## Правила игры

















8 домашних задания на 1-8 часов самостоятельной работы  
каждое (в сумме 100 баллов) + курсовой проект (30 баллов)

На выполнение каждого дз дается 2 недели, + еще две недели,  
чтобы досдать и получить оценку.

Через 4 недели после получения домашки оценку получить за  
данную домашку будет невозможно!!!

**ОБРАТИТЕ ВНИМАНИЕ!!! НЕ ОБЯЗАТЕЛЬНО СДАВАТЬ  
ВСЕ ДОМАШКИ!!!**

# В чем сложность?

#	$\Delta$ pub	Team Name	Kernel	Team Members	Score ?	Entries	Last
1	—	Роман Васильев			0.31691	34	10mo
2	$\uparrow 1$	Vladimir Bugaevsky			0.30655	9	10mo
3	$\downarrow 1$	Timur Mubarakshin			0.30516	13	10mo
4	—	Михаил Постников			0.30497	34	10mo
5	$\uparrow 1$	ralina			0.30466	21	10mo
		Baseline			0.30175		
6	$\uparrow 2$	PashaAdamenko			0.29895	34	10mo
7	$\downarrow 2$	Alexey Katsman [ITMO]			0.29232	22	9mo
8	$\uparrow 1$	Dana Zlochevskaya			0.29154	15	10mo
9	$\downarrow 2$	Adel Lesuchevskiy [ITMO]			0.28924	7	9mo
10	—	sergei.grabalin			0.28290	42	10mo
		Simple baseline			0.24552		
11	—	(ITMO)Амир Муратов			0.14487	1	9mo
12	—	Admiralskiy Yury			0.13538	1	10mo

# План лекции

Мотивация

Векторная модель ранжирования

Вероятностная модель ранжирования

Латентные модели

# Ранжированный поиск

- ▶ Поиск, который мы видели до этого был булевым
  - ▶ Документ либо подходит, либо нет
- ▶ Хорошо подходит для продвинутых экспертов, которые хорошо понимают что им нужно и что они могут найти в коллекции документов
  - ▶ Также хорошо подходит для приложений (анализ тысяч результатов)
- ▶ Плохо для большинства пользователей
  - ▶ Обычный пользователь никогда не будет писать булев запрос
  - ▶ Большинство пользователей не хотят просматривать тысячи результатов поиска

# Проблемы булева поиска

- ▶ Булевы запросы часто возвращают либо слишком мало ( $=0$ ) либо слишком много результатов (тысячи)
- ▶ Запрос 1: “скачать бесплатно” (4 млн. результатов)
- ▶ Запрос 2: “скачать бесплатно без регистрации без смс без ключа без хурмы без кидалова без торрента” (0 результатов)
- ▶ От пользователя это требует соответствующего навыка, чтобы составить запрос, который вернул бы приемлимое количество результатов
  - ▶ AND приводит к малому числу результатов
  - ▶ OR к слишком большому



# Модели ранжированного поиска

- ▶ Ранжированный поиск возвращает упорядоченный список документов из коллекции по запросу
- ▶ Вместо языка запросов и операторов, используются просто слова из человеческого языка

## Вопрос:

- ▶ Применяется ли в современных поисковых системах булев поиск?

# Схема ранжирования в поиске

## Этапы ранжирования



# Релевантность

Прежде чем начать...

Вопрос:

- ▶ Что такое релеватный документ?

# Релевантность

Сложное комплексное понятие, учитывающее множество факторов и зачастую крайне субъективное

В информационном поиске релевантность рассматривается с нескольких сторон

- ▶ тематическая релевантность
- ▶ пользовательская релевантность
- ▶ текстовая релевантность
- ▶ ...

# Ранжированный поиск

Когда поисковая система возвращает ранжированный список результатов, большой объем не является проблемой

- ▶ Количество найденных результатов не является проблемой для пользователя
- ▶ Мы показываем только top  $k (\approx 10)$  результатов
- ▶ Таким образом, мы не огорчаем пользователя

Предположение:

- ▶ Алгоритм ранжирования работает хорошо :)

# Ранжированный поиск

## Оценка релеватности документа

- ▶ Хотим вернуть документы, в порядке наиболее полезных для пользователя
- ▶ Каким образом мы можем составить такой порядок в соответствии с запросом?
- ▶ Назначим каждому документу оценку (score) для каждого документа по запросу (например из  $[0, 1]$ )
- ▶ Эта оценка должна отражать на сколько хорошо документ подходит запросу

# Вычисление веса

- ▶ Нужен способ назначения веса паре запрос-документ
- ▶ Начнем с запроса из одного термина
- ▶ Если термина нет в документе, то вес равен 0
- ▶ Чем чаще встречается термин в документе, тем выше вес

# Модель мешка слов

- ▶ Не учитывается порядок слов в документе
- ▶ John is quicker than Mary и Mary is quicker than John
- ▶ Вася быстрее Маши и Маша быстрее Васи
- ▶ НО Мать любит дочь и Дочь любит мать (не ясно кто кого любит)
- ▶ Такая модель называется моделью мешка слов
- ▶ Это шаг назад, так как координатный индекс может различить такие докуенты
- ▶ Вернемся к использованию координатной информации позже



# Частота термина

- ▶ Частота  $tf_{t,d}$  термина  $t$  в документе  $d$  определяется как количество раз, сколько  $t$  встречается в  $d$
- ▶ Хотим использовать  $tf$  при расчете весов. Но как?
- ▶ Просто частота не торт!
  - ▶ Документ с 10 вхождениями релевантнее документа с 1 вхождением (в 10 раз!!!).
- ▶ Релеватность не увеличивается пропорционально частоте

# Логарифмическое взвешивание

- ▶ Логарифмическая частота термина  $t$  в  $d$ :

$$w_{t,d} = \begin{cases} 1 + \log tf_{t,d}, & \text{if } tf_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- ▶ Вес для пары запрос-документ: сумма по всем терминам  $t$ , входящим в  $q$  и  $d$ :
- ▶ Вес

$$= \sum_{t \in q \cap d} (1 + \log tf_{t,d})$$

- ▶ Вес равен 0, если в документе нет ни одного термина из запроса.

# Документная частота

- ▶ Редкие термины информативнее частотных (стоп-слова)
- ▶ Рассмотрим термин запроса, который редко встречается в корпусе (например *“серобуромалиновый”*)
- ▶ Любой документ, содержащий в себе этот термин, скорее всего будет релевантен запросу *“серобуромалиновый”*
- ▶ То есть, имеет смысл давать больший вес редким терминам

# Обратная документная частота

## IDF

- ▶  $df_t$  - документная частота термина  $t$  (количество документов, содержащих  $t$ )
- ▶  $df_t$  - обратная мера информативности  $t$
- ▶  $df_t \leq N$
- ▶ Определим  $idf$  (inverse document frequency) термина как

$$idf_t = \log \frac{N}{df_t}$$

## Пример idf

term	$df_t$	$idf_t$
calpurnia	1	6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

$$idf_t = \log \frac{N}{df_t}$$

- ▶  $N = 10^6$
- ▶ Для каждого термина в корпусе только одно значение idf

# IDF в ранжировании

Значим ли IDF для запросов для одного термина

- ▶ iPhone

IDF не влияет на однословные запросы

- ▶ Idf влияет на ранжирование запросов из двух и более слов
- ▶ Для запросов типа “серобуромалиновые штаны”, взвешивание по IDF приводит к большому вкладу термина “серобуромалиновый”, чем термин “штаны”

# Взвешивание TF-IDF

- ▶ Вес термина  $tf - idf$  это произведение его весов  $tf$  и  $idf$ :

$$w_{t,d} = (1 + \log tf_{t,d}) \cdot \log \frac{N}{df_t}$$

- ▶ Самая известная модель взвешивания в информационном поиске
- ▶ Растет с ростом числа вхождений слова в документ
- ▶ Растет со степенью редкости термина

## Ранжирование по TF-IDF

$$Score(q, d) = \sum_{t \in q \cap d} tf \cdot idf_{t,d}$$

# N-gramm TF-IDF

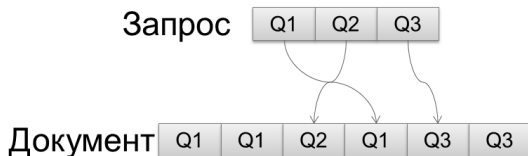
- ▶ Ничего не мешает нам аналогично с вхождениями слов рассматривать вхождения n-грамм
- ▶ А еще мы их можем взять с разным весом

$$Score(q, d) = \sum_n \alpha_n \sum_{t_n \in q \cap d} tf.idf_{t_n, d}$$



# Пассажный алгоритм

**Пассаж** - фрагмент документа, размера, не превышающего заданный, в котором встречаются все термы запроса, либо значительная часть термов запроса, суммарный IDF которых превышает заданное ограничение.



# Пассажный алгоритм

## Оценка пассажа

- ▶ TF-IDF
- ▶ Полнота
- ▶ Порядок слов
- ▶ Правильность словоформ
- ▶ Кучность
- ▶ Близость к началу
- ▶ Особенность зоны документы

# Параметрические зоны и индексы

- ▶ До сих пор документ представлялся последовательностью терминов
- ▶ Обычно документы состоят из нескольких частей, с определённой семантикой
  - ▶ Автор
  - ▶ Заголовок
  - ▶ Дата публикации
  - ▶ Язык
  - ▶ Формат
  - ▶ И т.п.
- ▶ Это все метаданные

# Компактность вхождений

- ▶ Текстовые запросы: набор терминов, введённых в поисковую строку
- ▶ Пользователи предпочитают документы, где термины запроса находятся на небольшом расстоянии друг относительно друга
- ▶ Пусть  $w$  будет наименьшим окном в документе, содержащим все термины запроса
- ▶ Например для запроса [strained mercy] такое окно в документе The quality of mercy is not strained равно 4 (в словах)
- ▶ Как учесть это в итоговом score?

## Как объединить все вместе?

$$Score(q, d) = \sum_n \alpha_n \sum_{t_n \in q \cap d} tf.idf_{t_n, d} + \sum_p score_p(q, d)$$

- ▶ Линейная модель
- ▶ Первое слагаемое n-граммный TF-IDF
- ▶ Второе слагаемое взвешенная сумма пассажных ранков

$$score_p(q, d) = score(pos, proximity, tf.idf, zone \dots)$$

# Векторная модель ранжирования

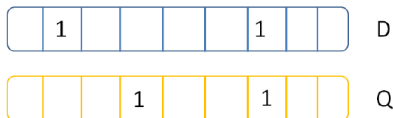
Документы и запросы - это вектора в  $T$  мерном пространстве, где  $T$  - общее количество термов (словоформ, основ, фраз и т.д.)

$$D_i = (d_{i1}, d_{i2}, \dots, d_{iT}) \quad Q = (q_1, q_2, \dots, q_T)$$

Коллекция документов представляется матрицей.

	$Term_1$	$Term_2$	$\dots$	$Term_t$
$Doc_1$	$d_{11}$	$d_{12}$	$\dots$	$d_{1t}$
$Doc_2$	$d_{21}$	$d_{22}$	$\dots$	$d_{2t}$
$\vdots$	$\vdots$			
$Doc_n$	$d_{n1}$	$d_{n2}$	$\dots$	$d_{nt}$

# Векторная модель ранжирования



Документы ранжируются в соответствии с схожестью вектора соответствующего запросу и вектора соответствующего документу

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

Вопрос:

- ▶ А почему именно косинус?

# Векторная модель

- ✓ простая модель ранжирования
- ✓ можно использовать любую меру схожести векторов
- ✓ можно использовать любую схему взвешивания термов
- ✗ Модель работает в предположении о независимости термов
- ✗ Невозможно определить способ оптимального ранжирования



# Вероятностная модель ранжирования

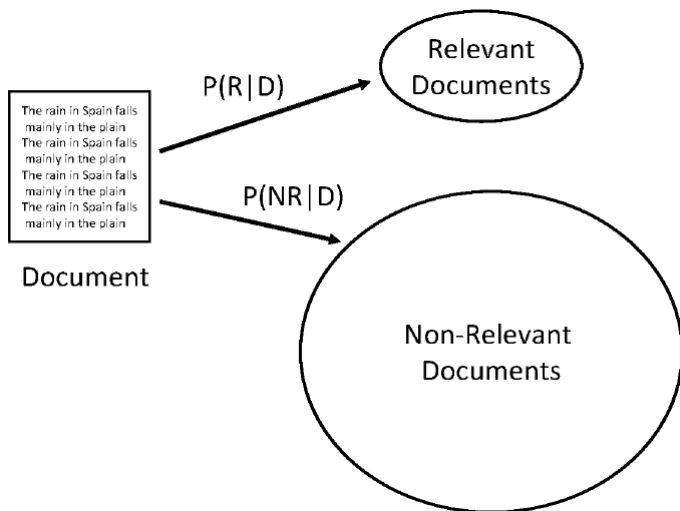
## Принцип вероятностного ранжирования

- ▶ Если поисковая система в ответ на запрос пользователя отдает документы в порядке уменьшения вероятности их релевантности запросу, то качество такой поисковой системы будет максимальным (Robertson, 1977)

## Ключевой вопрос:

- ▶ Какова вероятность того, что пользователь оценит данный документ как релеватный для этого запроса?

# Решаем задачу бинарной классификации



# Байесовский классификатор

## Оптимальное решающее правило

- ▶ Документ  $D$  релевантен запросу  $Q$ , если  $P(R = 1|D, Q) > P(R = 0|D, Q)$

## Оценка вероятностей

- ▶ Воспользуемся формулой Байеса

$$p(R|D) = \frac{P(D|R)P(R)}{P(D)}$$

# Модель бинарной независимости

- ▶ Документ представляет собой бинарный вектор термов
- ▶ Запрос представляет собой бинарный вектор термов
- ▶ Предположение о независимости появления термов в документе

$$P(D|R) = \prod_{i=1}^t P(d_i|R)$$

# Модель бинарной независимости

$p_i$  - вероятность встретить  $i$ -ый термин в релевантных документах

$s_i$  - вероятность встретить  $i$ -ый термин в нерелевантных документах

$$\begin{aligned}\frac{P(D|R)}{P(D|NR)} &= \prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i} = \\ &= \prod_{i:d_i=1} \frac{p_i}{s_i} \cdot \left( \prod_{i:d_i=1} \frac{1-p_i}{1-s_i} \cdot \prod_{i:d_i=1} \frac{1-s_i}{1-p_i} \right) \cdot \prod_{i:d_i=0} \frac{1-p_i}{1-s_i} = \\ &= \prod_{i:d_i=1} \frac{p_i(1-s_i)}{s_i(1-p_i)} \cdot \prod_i \frac{1-p_i}{1-s_i}\end{aligned}$$

# Модель бинарной независимости

- ▶ Оценка отношения правдоподобия

$$\sum_{i:d_i=1} \log \frac{p_i(1 - s_i)}{s_i(1 - p_i)}$$

- ▶ Запрос несет информацию о релеватных документах
- ▶ Если нет никакой дополнительной информации, то полагаем, что  $p_i$  постоянна, а  $s_i$  можно оценить по коллекции

$$\log \frac{0.5(1 - \frac{n_i}{N})}{\frac{n_i}{N}(1 - 0.5)} = \log \frac{N - n_i}{n_i}$$

# Модель бинарной независимости

## Честная оценка

	Relevant	Non-relevant	Total
$d_i = 1$	$r_i$	$n_i - r_i$	$n_i$
$d_i = 0$	$R - r_i$	$N - n_i - R + r_i$	$N - r_i$
Total	$R$	$N - R$	$N$

$$p_i = (r_i + 0.5)/(R + 1)$$

$$s_i = (n_i - r_i + 0.5)/(N - R + 1)$$

- Функция оценки:

$$\sum_{i:d_i=q_i=1} \log \frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)}$$

Классический алгоритм ранжирования, основанный на модели бинарной независимости

$$\sum_{i \in Q} \log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \cdot \frac{(k_1 + 1)f_i}{K + f_i} \cdot \frac{(k_2 + 1)qf_i}{k_2 + qf_i}$$

- ▶ значения  $k_1, k_2, K$  подбираются эмпирически

$$K = k_1((1 - b) + b \cdot \frac{dl}{avgdl})$$

- ▶ На TREC  $k_1 = 1.2$ ,  $k_2 \in [0, 1000]$ ,  $b = 0.75$



## Пример:

- ▶ Запрос: “president lincoln” ( $qf = 1$ )
- ▶  $r = R = 0$  (нет информации о релевантных документах)
- ▶ Размер коллекции  $N = 500000$  документов
- ▶ Термин “president” содержится в 40000 документов ( $n_1 = 40000$ )
- ▶ Термин “lincoln” содержится в 300 документов ( $n_2 = 300$ )
- ▶ Термин “president” встречается 15 раз в документе ( $f_1 = 15$ )
- ▶ Термин “lincoln” встречается 25 раз в документе ( $f_2 = 25$ )
- ▶  $\frac{dl}{avdl} = 0.9$
- ▶  $k_1 = 1.2$ ,  $b = 0.75$ ,  $k_2 = 100$

## Пример:

$$\begin{aligned}
 BM25(Q, D) &= \\
 &= \log \frac{(0 + 0.5)/(0 - 0 + 0.5)}{(40000 - 0 + 0.5)/(500000 - 40000 - 0 + 0 + 0.5)} \times \frac{15(1.2 + 1)}{1.11 + 15} \times \frac{1(100 + 1)}{100 + 1} + \\
 &+ \log \frac{(0 + 0.5)/(0 - 0 + 0.5)}{(300 - 0 + 0.5)/(500000 - 300 - 0 + 0 + 0.5)} \times \frac{25(1.2 + 1)}{1.11 + 25} \times \frac{1(100 + 1)}{100 + 1} = \\
 &= \log \left[ \frac{460000.5}{40000.5} \cdot \frac{33}{16.11} \cdot \frac{101}{101} \right] + \log \left[ \frac{499700.5}{300.5} \cdot \frac{55}{26.11} \cdot \frac{101}{101} \right] = 20.66
 \end{aligned}$$

# BM25F

## BM25

$$\text{score}(Q, D) = \sum_{i=1}^n \text{l}df(q_i) \frac{f(q_i, D)(k_1 + 1)}{f(q_i, D) + k_1(1 - b + b \frac{dl}{\text{avg}dl})}$$

## BM25F

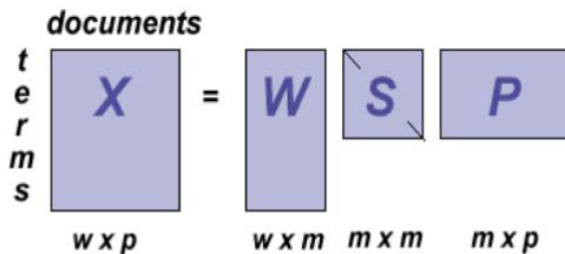
$$\text{score}(Q, D) = \sum_{i=1}^n \text{l}df(q_i) \frac{\sum_E \text{rank}(E)(k_1 + 1)}{\sum_E \text{rank}(E) + k_1(1 - b + b \frac{dl}{\text{avg}dl})}$$

$\text{rank}(E)$  - функция взвешивания вхождения слова в документ  
(зависит от зоны, позиции и т.д.)

# Недостатки рассмотренных моделей

Какие недостатки есть в тех моделях, которые рассмотрели?

# Latent semantic analysis (1988)



$$\hat{D} = A^T D, \quad \hat{Q} = A^T Q,$$

$$A = W_k S^{-1}$$

$$\text{sim}(Q, D) = \frac{\hat{Q} \hat{D}}{\|Q\| \|D\|}$$

# Latent semantic analysis (1988)

Фактически латентно-семантический анализ - это применение SVD разложения к матрице “термин-документ”

- ✓ Оценка близости документов
- ✓ Оценка близости терминов
- ✓ Кластеризация документов
- ✓ Взвешивание пары запрос-документ
- ✗ Низкая скорость для больших коллекций

# Тематическое моделирование

## Что такое тема?

- ▶ тема - семантический кластер текстов
- ▶ тема - набор терминов предметной области
- ▶ тема - условное распределение на множестве слов

$p(w|t)$  — вероятность слова  $w$  в теме  $t$

- ▶ тема - тематический профиль документа

$p(t|d)$  — вероятность темы  $t$  в документе  $d$

## Цель тематической модели:

Найти латентные темы документов коллекции по наблюдаемым распределениям слов  $p(w|d)$  в документах.

# Тематическое моделирование

## Основные положения:

- ▶ Модель мешка слов для документов (порядок не важен)
- ▶ Модель мешка документов для коллекции (порядок не важен)
- ▶ Коллекция - это i.i.d. выборка  $(d_i, w_i, t_i)_{i=1}^n \sim p(d, w, t)$
- ▶  $d_i, w_i$  - наблюдаемые переменные,  $t_i$  - скрытые
- ▶ Гипотеза условной независимости:  $p(w|d, t) = p(w|t)$
- ▶ Считаем, что тексты предобработаны (стемминг, лемматизация, удаление стоп-слов и т.д.)



# Тематическое моделирование

$$p(w|d) = \sum_{t \in T} p(w|t) \cdot p(t|d)$$

Темы

gene 0.04  
dna 0.02  
genetic 0.01  
...

life 0.02  
evolve 0.01  
organism 0.01  
...

brain 0.04  
neuron 0.02  
nerve 0.01  
...

data 0.02  
number 0.02  
computer 0.01  
...

Документы

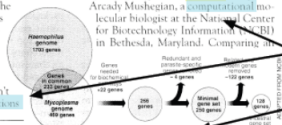
Пропорции и состав  
тем в документе

## Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK—How many **genes** does an **organism** need to **survive**? Last week at the genome meeting here,\* two genome researchers with radically different approaches presented complementary views of the basic genes needed for **life**. One research team, using **computer** analyses to compare known **genomes**, concluded that today's **organisms** can be sustained with just 250 genes, and that the earliest life forms required a mere 128 **genes**. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those **predictions**

"are not all that far apart," especially in comparison to the 75,000 **genes** in the human genome, notes Siv Andersson, a medical geneticist at Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a **genetic** numbers game, particularly if more and more **genomes** are rapidly sequenced and sequenced. "It may be a way of organizing any newly **sequenced genome**," explains Arcady Mushegian, a **computational** molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an



\* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

**Stripping down.** Computer analysis yields an estimate of the minimum modern and ancient genomes.

SCIENCE • VOL. 272 • 24 MAY 1996

# Тематическое моделирование

## Дано:

- ▶  $W$  - словарь слов
- ▶  $D$  - коллекция документов
- ▶  $d = \{w_1 \dots w_{n_d}\}$  - документ
- ▶  $n_{dw}$  - число раз, когда слово  $w$  встретилось в документе  $d$
- ▶  $n_d$  - длина документа  $d$

## Найти:

Параметры модели  $\frac{n_{dw}}{n_d} \approx p(w|d) = \sum_{t \in T} \psi_{wt} \theta_{td}$

$\psi_{wt} = p(w|t)$  - вероятности слов  $w$  в каждой теме  $t$

$\theta_{td} = p(t|d)$  - вероятности тем  $t$  в каждой документе  $d$

# Тематическое моделирование

- ▶  $X = (d_i, w_i)_{i=1}^n$  - исходные данные
- ▶  $T = (t_i)_{i=1}^n$  - скрытые переменные, темы
- ▶  $\Omega = (\Psi, \Theta)$  - параметры

Нужно по  $X$  найти  $\Omega$

Максимизируем неполное правдоподобие

$$\ln p(X|\Omega) = \ln \sum_T p(X, T|\Omega) \rightarrow \max_{\Omega}$$

ЕМ алгоритм:

$$\text{E-step: } q(T) = p(T|X, \Omega)$$

$$\text{M-step: } \sum_T q(T) \ln p(X, T|\Omega) \rightarrow \max_{\Omega}$$

# Тематическое моделирование

$p(\Omega)$  - априорное распределение параметров модели

Принцип максимума правдоподобия

$$p(X, \Omega) = p(X|\Omega)p(\Omega) \rightarrow \max_{\Omega}$$

$$\ln p(X, \Omega) = \ln p(X|\Omega) + \ln p(\Omega) \rightarrow \max_{\Omega}$$

Обозначим  $R(\Omega) = \ln p(\Omega)$

**PLSA [Hofmann, 1999]:**  $R(\Omega) = 0$

**LDA [Blei, 2003]:**  $R(\Omega) = \ln \prod_{t \in T} \text{Dir}(\psi_t | \beta) \prod_{d \in D} \text{Dir}(\theta_d | \alpha)$

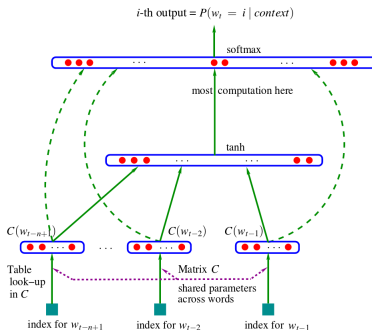
ЕМ алгоритм:

$$\text{E-step: } q(T) = p(T|X, \Omega)$$

$$\text{M-step: } \sum_T q(T) \ln p(X, T|\Omega) + R(\Omega) \rightarrow \max_{\Omega}$$

# A Neural Probabilistic Language Model (Y. Bengio 2003)

Пытаемся с помощью нейросетей оценить вероятность следующего слова по набору из предыдущих слов (сеть может быть как последовательной так и рекуррентной)



✗ Долго и сложно обучать

# Дистрибутивная гипотеза

## Гипотеза:

Лингвистические единицы, встречающиеся в схожих контекстах, имеют близкие значения.

## Вывод:

Значит, векторы слов, можно построить с помощью контекстов этих слов.

# Представление слов контекстами

Дано:

- ▶  $V$  - словарь слов
- ▶  $C$  - множество контекстов

Можем построить матрицу  $S$  размера  $|V| \times |C|$ , элементы которой будут описывать связь слова  $w_i$  с контекстом  $c_j$ .

Например, можно взять положительную поточечную взаимную информацию (PPMI):

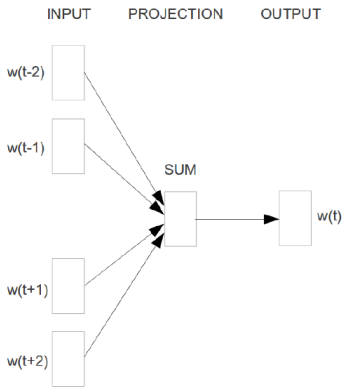
$$S_{i,j} = \max(PMI(w_i, c_j), 0),$$

$$PMI(w, c) = \log \frac{p(w, c)}{p(w)p(c)} = \log \frac{\text{freq}(w, c)|V|}{\text{freq}(w)\text{freq}(c)}$$

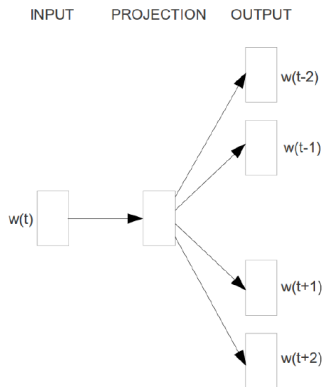
✗ Очень большая размерность матрицы

# Word2Vec (2013)

## Архитектуры CBOW и Skip-gram



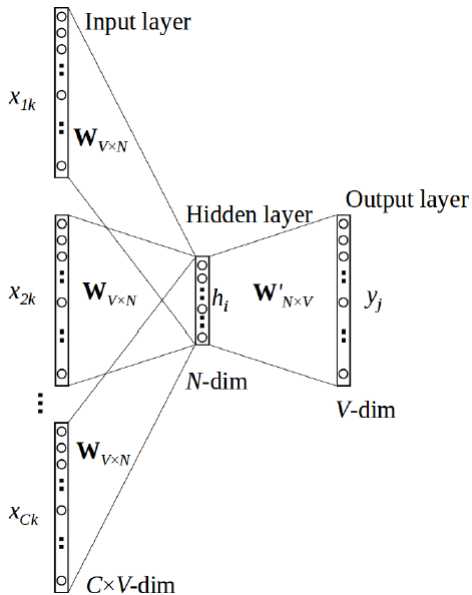
**CBOW**



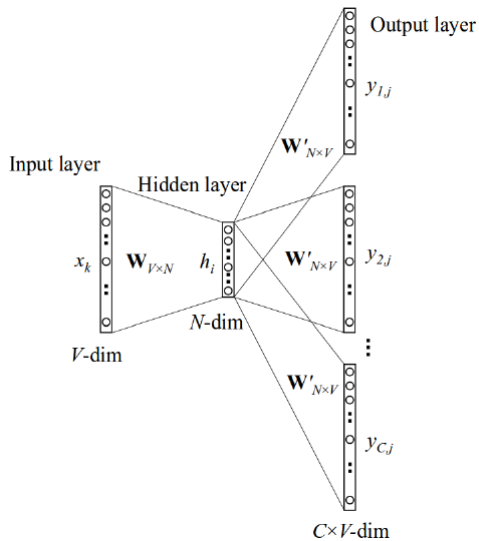
**Skip-gram**



# CBOW (Continuous Bag of Words)



# Skip-gram



# Skip-gram

Оптимизируем

$$\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t)$$

# Вычисление вероятностей выходных слов

Используется softmax

$$p(w_O|w_I) = \frac{\exp \langle \mathbf{v}'_{w_O}, \mathbf{v}_{w_I} \rangle}{\sum_{w=1}^{|V|} \exp(\langle \mathbf{v}'_w, \mathbf{v}_{w_I} \rangle)}$$

На практике эту формулу применять сложно, так как вычисление градиента пропорционально  $|V|$ .

На практике применяют разные аппроксимации: иерархический softmax или negative sampling.

# Negative sampling

## Идея:

Не будем рассматривать все слова из словаря, а учтем только рассматриваемое слово + подмешаем еще  $k$  отрицательных примеров.

Заменяем  $\log p(w_O|w_I)$  на

$$\log \sigma(\langle \mathbf{v}'_{w_O}, \mathbf{v}_{w_I} \rangle) + \sum_{i=1}^k \mathbb{E}_{w_i \sim P_n(w)} [\log \sigma(-\langle \mathbf{v}'_{w_i}, \mathbf{v}_{w_I} \rangle)]$$

- ▶  $k \approx 5 - 20$  для небольших выборок
- ▶  $k \approx 2 - 5$  для больших данных

# Смысл Negative Sampling

- ▶ Фактически, мы имеем два распределения слов: “положительное” (D) и “отрицательное” (N).
- ▶ Мы их смешиваем в пропорции  $1 : k$
- ▶ Задача модели: угадать, из какого распределения пришло слово

# Смысл Negative Sampling

По предположению,

$$p(D|w, c) = \sigma(\langle w, c \rangle)$$

Но по формуле Байесса

$$p(D|w, c) = \frac{p(w, c|D)p(D)}{p(w, c|D)p(D) + p(w, c|N)p(N)}$$

Считаем, что контексты в негативных примерах не зависят от слова:

$$p(w, c|N) = p(w|D)p(c|D)$$

Таким образом

$$\begin{aligned} p(D|w, c) &= \frac{p(w, c|D)^{\frac{1}{k+1}}}{p(w, c|D)^{\frac{1}{k+1}} + p(w, c|N)^{\frac{k}{k+1}}} = \\ &= \frac{1}{1 + k \frac{p(w|D)p(c|D)}{p(w, c|D)}} \end{aligned}$$

# Смысл Negative Sampling

Заметим, что выражение стоящее в знаменателе очень похоже на взаимную информацию

$$PMI(w, c) = \log \frac{p(w, c|D)}{p(w|D)p(c|D)}$$

Таким образом,

$$p(D|w, c) = \frac{1}{1 + ke^{-PMI(w, c)}}$$

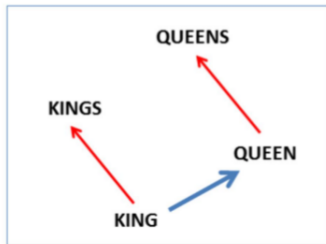
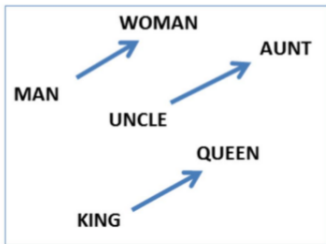
$$\langle w, c \rangle = PMI(w, c) - \ln k - \text{“сдвинутый” } PMI$$

Skip-gram Negative Sampling эквивалентен факторизации матрицы “сдвинутого”  $PMI$ .



# Свойства выученных представлений

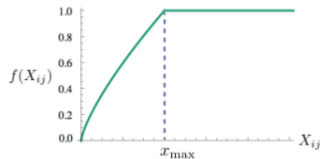
$$v(\text{king}) - v(\text{man}) + v(\text{woman}) \approx v(\text{queen})$$



# GLoVe (2014)

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^W f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

$$X_{final} = U + V$$



# Word2vec

## Вопрос

- ▶ Как это использовать в поиске?

# Word2vec

## Вопрос

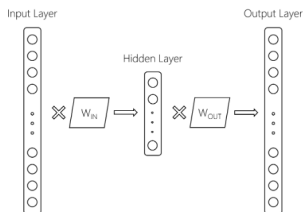
- ▶ Какие Вы видите недостатки?

# Word2vec

## Недостатки

- ✗ Качество сильно зависит от обучающих данных, их количества, размера векторов (на вектора большой размерности нужно много вычислительных затрат)
- ✗ Усреднение векторов слов работает плохо уже на текстах среднего размера (не говоря уже о больших)
- ✗ Из word2vec нельзя получить представление фиксированного размера для текстов переменной длины
- ✗ Не ясно что делать со словами, которых нет в словаре

# Dual Embedding Space Model (DESM) (2016)



- ▶ Word2vec optimizes IN-OUT dot product which captures co-occurrence statistics of words from the training corpus
- ▶ We can gain by using these two embeddings differently

# Dual Embedding Space Model (DESM)

yale			seahawks			eminem		
IN-IN	OUT-OUT	IN-OUT	IN-IN	OUT-OUT	IN-OUT	IN-IN	OUT-OUT	IN-OUT
yale	yale	yale	seahawks	seahawks	seahawks	eminem	eminem	eminem
harvard	uconn	faculty	49ers	broncos	highlights	rihanna	rihanna	rap
nyu	harvard	alumni	broncos	49ers	jerseys	ludacris	dre	featuring
cornell	tulane	orientation	packers	nfl	tshirts	kanye	kanye	tracklist
tulane	nyu	haven	nfl	packers	seattle	beyonce	beyonce	diss
tufts	tufts	graduate	steelers	steelers	hats	2pac	tupac	performs

- ▶ IN-IN and OUT-OUT cosine similarities are high for words that are similar by function or type
- ▶ IN-OUT cosine similarities are high between words that often co-occur in the same query or document

# Dual Embedding Space Model (DESM)

*Albuquerque* is the most populous *city* in the U.S. state of *New Mexico*. The high-altitude *city* serves as the county seat of *Bernalillo* County, and it is situated in the *central* part of the state, straddling the *Rio Grande*. The *city population* is 557,169 as of the July 1, 2014, *population* estimate from the United States Census Bureau, and ranks as the 32nd-largest *city* in the U.S. The *Metropolitan Statistical Area* (or MSA) has a *population* of 902,797 according to the United States Census Bureau's most recently available estimate for July 1, 2013.

(a)

Allen suggested that they could program a BASIC interpreter for the device; after a call from Gates claiming to have a working interpreter, MITS requested a demonstration. Since they didn't actually have one, Allen worked on a simulator for the Altair while Gates developed the interpreter. Although they developed the interpreter on a simulator and not the actual device, the interpreter worked flawlessly when they demonstrated the interpreter to MITS in *Albuquerque, New Mexico* in March 1975; MITS agreed to distribute it, marketing it as Altair BASIC.

(b)

**Figure 3: Two different passages from Wikipedia that mentions "Albuquerque" (highlighted in orange) exactly once. Highlighted in green are all the words that have an IN-OUT similarity score with the word "Albuquerque" above a fixed threshold (we choose -0.03 for this visualization) and can be considered as providing supporting evidence that (a) is *about* Albuquerque, whereas (b) happens to only *mention* the city.**



# Dual Embedding Space Model (DESM)

- ▶ A document is represented by the centroid of its word OUT\_vectors:

$$\vec{v}_{d,\text{OUT}} = \frac{1}{|d|} \sum_{t_d \in d} \frac{\vec{v}_{t_d,\text{OUT}}}{\|\vec{v}_{t_d,\text{OUT}}\|}$$

- ▶ Query-document similarity is average of cosine similarity over query words:

$$\text{DESM}_{\text{IN-OUT}}(q, d) = \frac{1}{q} \sum_{t_q \in q} \frac{\vec{v}_{t_q,\text{IN}}^\top \vec{v}_{t_d,\text{OUT}}}{\|\vec{v}_{t_q,\text{IN}}\| \|\vec{v}_{t_d,\text{OUT}}\|}$$

- ▶ IN-OUT captures more Topical notion of similarity than IN-IN and OUT-OUT.
- ▶ DESM is effective at, but only at, ranking at least somewhat relevant documents.

# Dual Embedding Space Model (DESM)

	Explicitly Judged Test Set			Implicit Feedback based Test Set		
	NDCG@1	NDCG@3	NDCG@10	NDCG@1	NDCG@3	NDCG@10
BM25	23.69	29.14	44.77	13.65	27.41	49.26
LSA	22.41*	28.25*	44.24*	16.35*	31.75*	52.05*
DESM (IN-IN, trained on body text)	23.59	29.59	45.51*	18.62*	33.80*	53.32*
DESM (IN-IN, trained on queries)	23.75	29.72	46.36*	18.37*	35.18*	54.20*
DESM (IN-OUT, trained on body text)	24.06	30.32*	46.57*	19.67*	35.53*	54.13*
DESM (IN-OUT, trained on queries)	<b>25.02*</b>	<b>31.14*</b>	<b>47.89*</b>	<b>20.66*</b>	<b>37.34*</b>	<b>55.84*</b>

# BM25 + DESM

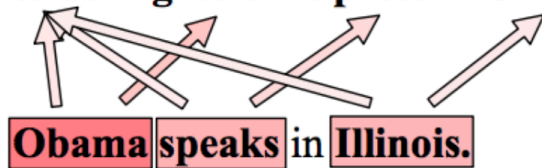
$$M(q, d) = \alpha DESM(q, d) + (1 - \alpha) BM25(q, d), \quad 0 \leq \alpha \leq 1$$

	Explicitly Judged Test Set			Implicit Feedback based Test Set		
	NDCG@1	NDCG@3	NDCG@10	NDCG@1	NDCG@3	NDCG@10
BM25	21.44	26.09	37.53	11.68	22.14	33.19
LSA	04.61*	04.63*	04.83*	01.97*	03.24*	04.54*
DESM (IN-IN, trained on body text)	06.69*	06.80*	07.39*	03.39*	05.09*	07.13*
DESM (IN-IN, trained on queries)	05.56*	05.59*	06.03*	02.62*	04.06*	05.92*
DESM (IN-OUT, trained on body text)	01.01*	01.16*	01.58*	00.78*	01.12*	02.07*
DESM (IN-OUT, trained on queries)	00.62*	00.58*	00.81*	00.29*	00.39*	01.36*
BM25 + DESM (IN-IN, trained on body text)	21.53	26.16	37.48	11.96	22.58*	33.70*
BM25 + DESM (IN-IN, trained on queries)	<b>21.58</b>	26.20	37.62	11.91	22.47*	33.72*
BM25 + DESM (IN-OUT, trained on body text)	21.47	26.18	37.55	11.83	22.42*	33.60*
BM25 + DESM (IN-OUT, trained on queries)	21.54	<b>26.42*</b>	<b>37.86*</b>	<b>12.22*</b>	<b>22.96*</b>	<b>34.11*</b>

## Word Mover Distance

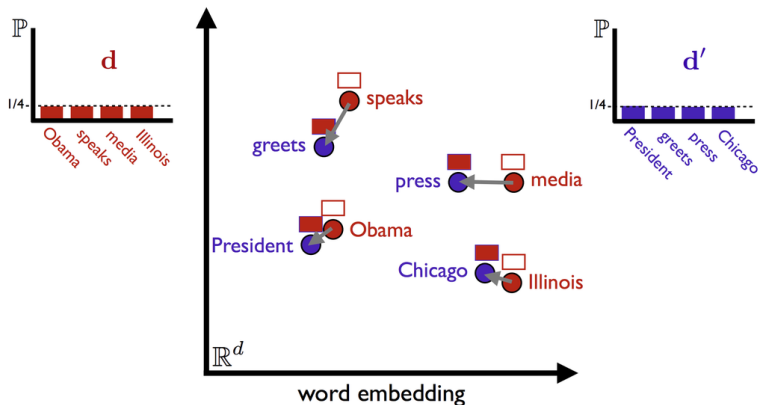
$$WMD(q, d) = \sum_{ij} dist(q_i, d_j)$$

**President greets the press in Chicago.**



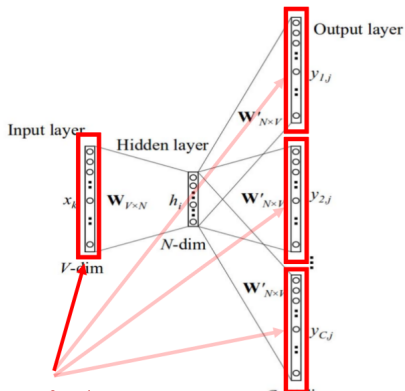
# Word Mover Distance

$$WMD(q, d) = \sum_{ij} dist(q_i, d_j)$$

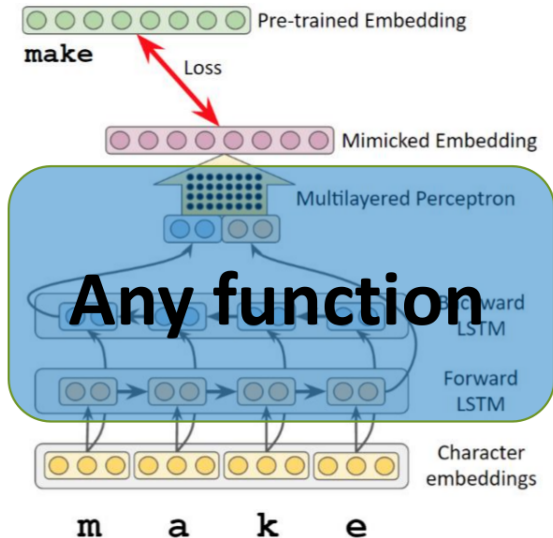


# FastText

- ▶ Каждое слово представляется мешком символьных n-грамм
- ▶ {wh, whe, her, ere, re}  
{**where**}
- ▶ Выучиваем свое представление для каждой n-граммы
- ▶ Каждое слово является суммой векторных представлений своих символьных n-грамм



# Генерализация на уровень символов



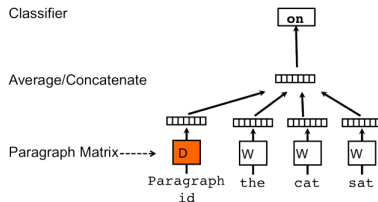
# Doc2vec (Mikolov, 2014)

- ▶ Обобщение word2vec модели на целые документы (фразы, предложения и т.д.)
- ▶ Преобразует текст произвольной длины в вектор фиксированного размера
- ▶ Distributed Memory (DM)
- ▶ Distributed Bag of Words (DBOW)



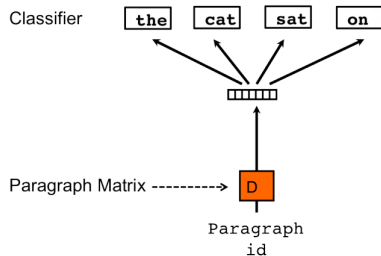
## Distributed Memory (DM)

- ▶ Назначим и рандомно проинициализируем paragraph vector
- ▶ Будем предсказывать слово из текста используя контекст и paragraph vector
- ▶ Идем скользящим окном по всему документу, сохраняя при этом paragraph vector фиксированным (поэтому Distributed Memory)
- ▶ Обновление происходит при помощи SGD и backprop

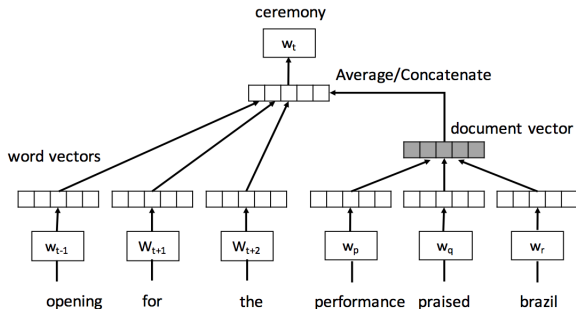


## Distributed Bag of Words (DBOW)

- ▶ Используем только paragraph vector (вектора слов не используем)
- ▶ Берем окно из слов в параграфе и случайно семплируем какое из слов предсказать используя paragraph vector (игнорируем порядок слов)
- ▶ Очень просто и требует меньше ресурсов
- ▶ Но при этом хуже по качеству, чем DM (однако DM + DBOW работают лучше!)



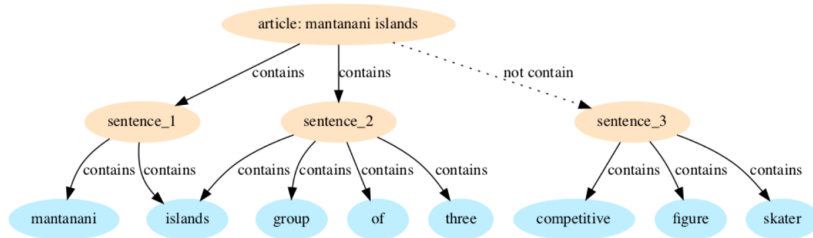
# Doc2VecC (Document Vector Through Corruption) 2017



# SentenceSpace: Learning Sentence Embeddings

**Setting:** Learning the mapping between sentences. Given the embedding of one sentence, one can find semantically similar/relevant sentences.

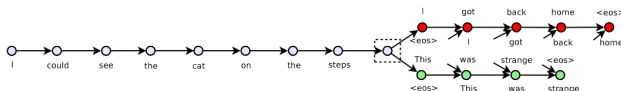
**Model:** Each example is a collection of sentences which are semantically related. Two are picked at random using trainMode 3: one as the input and one as the label, other sentences are picked as random negatives. One easy way to obtain semantically related sentences without labeling is to consider all sentences in the same document are related, and then train on those documents.



# Skip-Thought Vectors

## ► Skip-Thought Vectors

- Conceptually similar to distributional semantics: a unit's representation is a function of its neighbouring units, except units are sentences instead of words.



- Similar to auto-encoding objective: encode sentence, but decode neighboring sentences.
- Pair of LSTM-based seq2seq models with shared encoder.

# Skip-Thought Vectors

$x(0)$ : Hi, My name is Sanyam

$x(1)$ : Today, I went to the zoo.

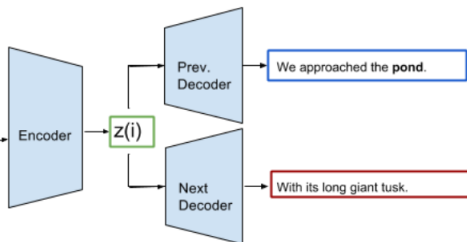
...

$x(i-1)$ : We approached the tree.

$x(i)$ : The elephant was still.

$x(i+1)$ : It was taking a nap probably.

...

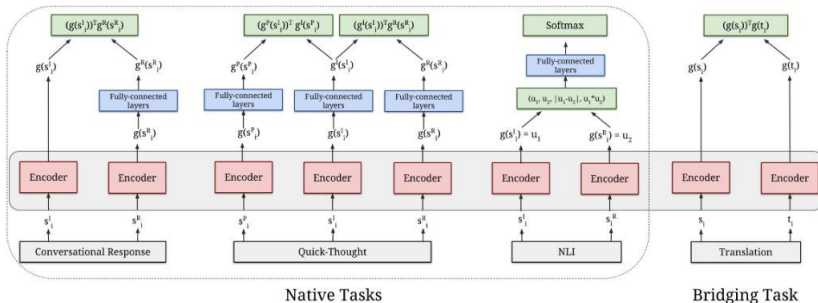


Skip Thoughts model overview

# ELMO (Embeddings from Language Model)

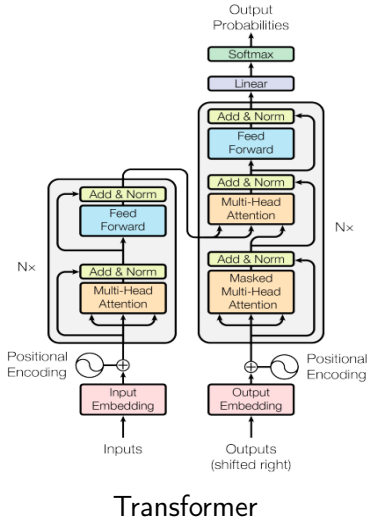
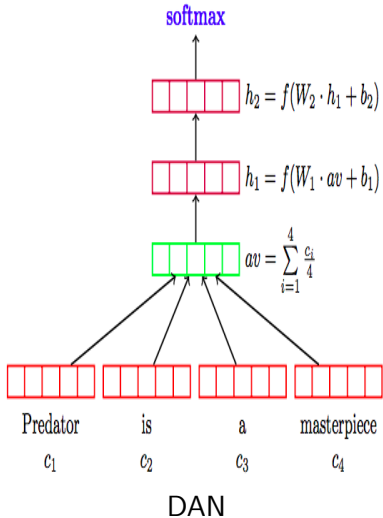
- ▶ Word embeddings: concat (embedding of a word, char-cnn)
- ▶ Train Bi-LSTM LM on a large corpus
- ▶ Use weighted sum of hidden representations from different layers, weighteds are trained with the task

# Universal Sentence Encoder Architecture

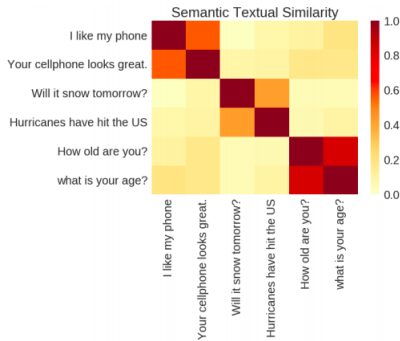




# Universal Sentence Encoder Architecture

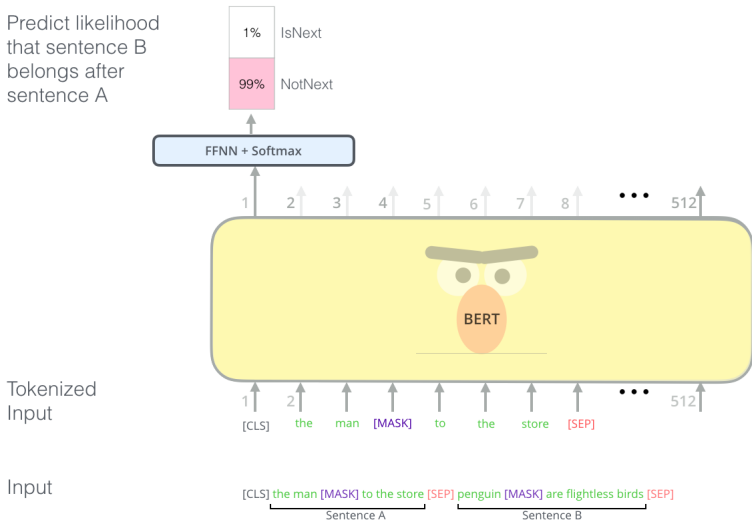


# Universal Sentence Encoder

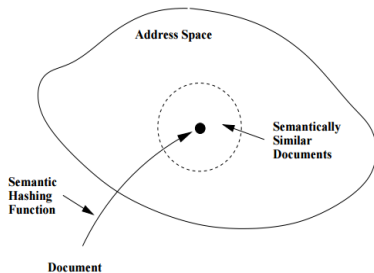
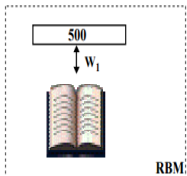
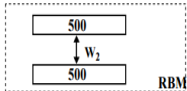
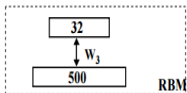
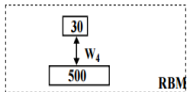


# BERT (Bidirectional Transformers) 2018

Predict likelihood  
that sentence B  
belongs after  
sentence A



# Semantic hashing (Hinton, Salakhutdinov, 2009)



# Document & entity representation learning toolkits

- ▶ **Gensim** <https://github.com/RaRe-Technologies/gensim>
- ▶ **SERT** <http://www.github.com/cvangysel/SERT>
- ▶ **cuNVSM** <http://www.github.com/cvangysel/cuNVSM>
- ▶ **HEM** <https://ciir.cs.umass.edu/downloads/HEM>

# Задача

## Дано:

- ▶ Набор вопросных запросов из поиска@mail.ru
- ▶ Набор документов, соответствующих этим запросам

## Задание:

- ▶ Используя модели текстового ранжирования (tf-idf, BM25, BM25F, passage, LSA, LDA, word2vec, doc2vec) необходимо отранжировать данные документы по запросам
- ▶ В качестве метрики качества будет выступать NDCG

## Формат:

- ▶ Kaggle конкурс сроком на 1 месяц

# Вопросы

