

Анализ неструктурированных данных

Лекция

Базовые векторные представления слов и документов

Мурат Апишев (mel-lain@yandex.ru)

ФКН ВШЭ

11 сентября, 2019

Векторные представления

- ▶ *Векторное представление (embedding)* — сопоставление произвольному объекту некоторого числового вектора в пространстве фиксированной размерности
- ▶ Наиболее известный вид – векторные представления слов (word embedding)
- ▶ Векторы могут обладать разнообразными полезными свойствами, отражать близость объектов в разных смыслах
- ▶ Для слов это может быть семантическая близость

Зачем нужны векторные представления

В современных подходах эмбединги используются в качестве признаков для решения почти любых задач машинного обучения

В текстовой аналитике это:

1. выделение именованных сущностей (NER)
2. выделение частей речи (POS-tagging)
3. машинный перевод
4. кластеризация документов
5. классификация документов, анализа тональности (sentiment)
6. ранжирование документов
7. генерация текста

One-hot encoding

Самый простой способ кодирования категориальных признаков:

"a"	"abbreviations"		"zoology"	"zoom"
1	0		0	0
0	1		0	1
0	0		0	0
.
.	.		.	.
.	.		.	.
0	0		0	0
0	0		1	0
0	0		0	1

Полученные векторы огромные и ортогональные

SVD для получения эмбедингов слов

1. По корпусу текстов D со словарём T строим *матрицу со-встречаемостей* $X_{|T| \times |T|}$

Возможны различные варианты учёта со-встречаемости слов:

- ▶ сумма по всей коллекции числа попаданий пары слов в окно фиксированного размера
- ▶ количество документов, хоть раз содержащих пару слов
- ▶ количество документов, хоть раз содержащих пару слов в окне

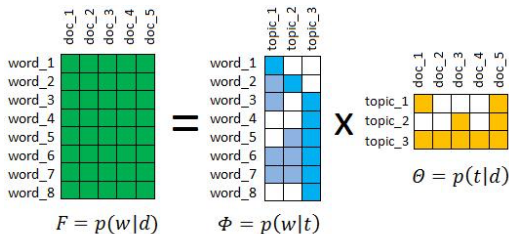
Понижаем размерность:

2. SVD-разложение: $X = USV^T$
3. Из столбцов матрицы U выбираются первые K компонент

Недостатки SVD

1. Относительно низкое качество получаемых представлений
2. Сложность работы с очень большой и разреженной матрицей
3. Сложность добавления новых слов/документов (решается инкрементальными методами построения)

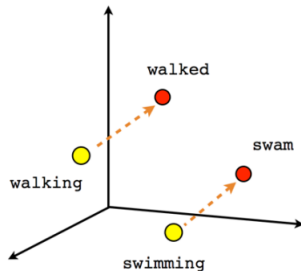
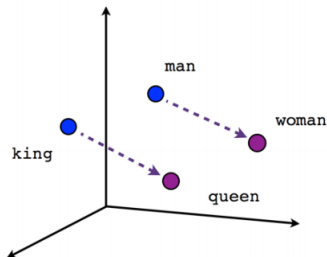
Тематическое моделирование



- ▶ Классические тематические модели получают на вход матрицу «мешка слов» или tf-idf и строят два типа распределений:
 - ▶ слов в кластерах-темах
 - ▶ тем в документах
- ▶ По факту получается стохастическое матричное разложение
- ▶ Строки матрицы «слова-темы» можно использовать в качестве эмбедингов
- ▶ Современные реализации инкрементально обучаются на больших данных
- ▶ Как и SVD, простые ТМ не учитывают локальный контекст

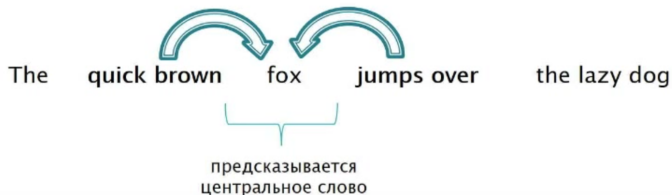
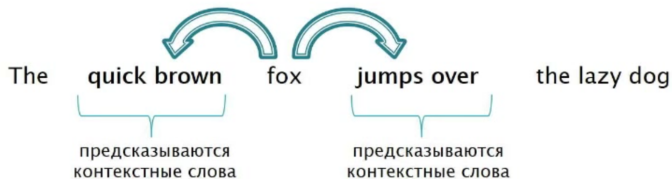
word2vec

- ▶ **word2vec** — группа алгоритмов, предназначенных для получения вещественных векторных представлений слов
- ▶ **Идея:** «Слова со схожими значениями разделяют схожий контекст»
- ▶ Как правило, в векторном представлении семантически близкие слова оказываются рядом



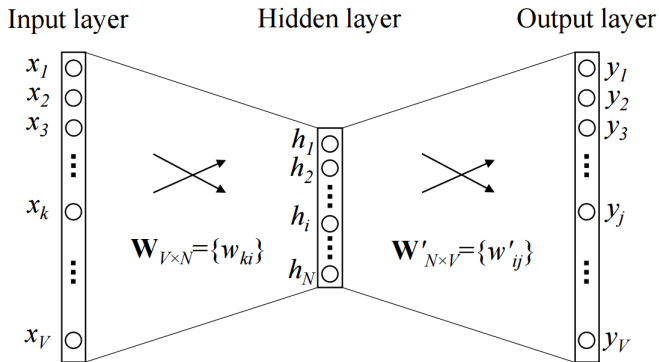
Don't count, predict!

Две модели: **Skip-gram** и **Continuous BOW**

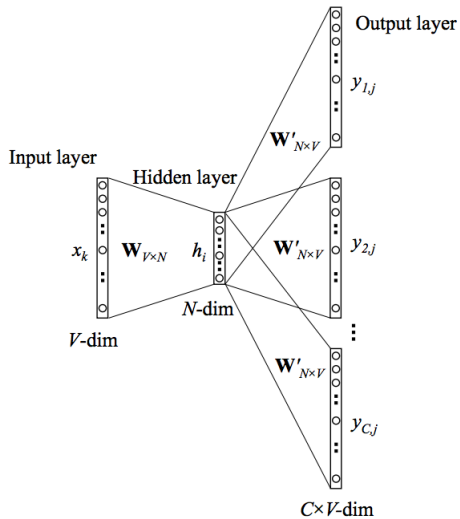


[Ссылка на источник картинки](#)

Модель CBOW (единичный контекст)



Модель Skip-gram



Производительность обучения Skip-gram

Подсчёт softmax — вычислительно дорогая операция

Применяются различные методы аппроксимации:

1. Softmax-based

- ▶ **Иерархический softmax**
- ▶ Дифференциальный softmax
- ▶ CNN-softmax
- ▶ ...

2. Sampling-based

- ▶ **Negative Sampling**
- ▶ Noise Contrastive Estimation
- ▶ ...

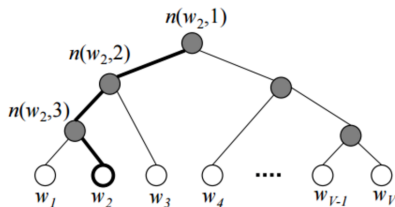
Иерархический softmax

Вычисление softmax — дорогая операция, $O(V)$

Иерархический softmax — $O(\log(V))$

Построим бинарное дерево, листьями которого будут уникальные слова коллекции (например дерево Хаффмана)

Выходы скрытого слоя связываются с внутренними узлами дерева ($V-1$ штук)



Иерархический softmax

Идея: в процессе обучения при фиксированном контексте нас интересует только предсказываемое слово

Вероятность того, что w будет выходным словом:

$$p(w = w_{out}) = \prod_{j=1}^{L(w)-1} \sigma(\{n(w, j+1) = lch(n(w, j))\} v_{n(w, j)}^T u)$$

- ▶ $L(w)$ — длина пути от корня до слова w
- ▶ $n(w, j)$ — j -я вершина на этом пути
- ▶ $\sigma(x)$ — сигмоида
- ▶ $\{\text{true}\} = +1, \{\text{false}\} = -1$
- ▶ $lch(n)$ — левый потомок вершины n
- ▶ $u = v_{w_{inp}}$ в случае skip-gram, $u = 1/h \sum_{k=1}^h v_{inp, k}$ (усреднённый вектор контекста) в случае CBOW

Иерархический softmax

Вероятность того, что w будет выходным словом:

$$p(w = w_{out}) = \prod_{j=1}^{L(w)-1} \sigma(\{n(w, j+1) = lch(n(w, j))\} v_{n(w, j)}^T u)$$

Считаем множество бинарных вероятностей, на каждом шаге можно пойти налево или направо с вероятностями

$$p(n, left) = \sigma(v_n^T u)$$

$$p(n, right) = 1 - p(n, left) = \sigma(-v_n^T u)$$

Затем на каждом шаге вероятности перемножаются и получается искомая формула

Negative Sampling

- ▶ Можно изменить постановку задачи и функционал качества.
- ▶ Решаем задачу бинарной классификации: $z = 1$ — пара $(w, s) \in D$, $z = 0$ — нет. ($s \in c(w)$)

$$p(z = 1 | (w, s)) = \frac{1}{1 + \exp(-v_w^T v_s)} = \sigma(v_w^T v_s)$$

- ▶ Запишем новый функционал правдоподобия:

$$\mathcal{L} = \sum_{(w,s) \in D_1} \log \sigma(v_w^T v_s) + \sum_{(w,s) \in D_2} \log \sigma(-v_w^T v_s),$$

$$D_1 = \{(w, s) : s \in c(w)\}, \quad D_2 = \{(w, c) : s \notin c(w)\}$$

Negative Sampling

$$\mathcal{L} = \sum_{(w,s) \in D_1} \log \sigma(v_w^T v_s) + \sum_{(w,s) \in D_2} \log \sigma(-v_w^T v_s),$$

- ▶ Но множество всех отрицательных примеров отсутствует
- ▶ Выход — для каждого рассматриваемого слова w генерировать в качестве отрицательных примеров случайные слова из T
- ▶ Функционал оптимизируется с помощью SGD

word2vec и PMI

Pointwise Mutual Information:

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

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

$$SPPMI(w, c) = PPMI(w, c) - \log k$$

k — число отрицательных примеров (negative samples)

Можно показать, что модель SGNS является разложением матрицы SPPMI (строки — слова, столбцы — контексты)

Важно: под контекстом в данном случае понимается *центральное слово*, на основании которого предсказываются стоящие рядом слова

Global Vectors

- ▶ На пальцах: GloVe = SVD + word2vec
- ▶ Строим матрицу $X \in \mathbb{R}^{V \times V}$, x_{ij} — количество раз, когда слово i встречается в контексте слова j (в окне ≤ 9 слов)
- ▶ $P_{ij} = \frac{x_{ij}}{X_i}$ — вероятность j в контексте i (X_i — сумма i -й строки).
- ▶ Строим функцию $F(w_i, w_j, \hat{w}_k)$, показывающую, какое из слов i и j вероятнее увидеть в контексте k (> 1 если i чаще)

$$F(w_i, w_j, \hat{w}_k) = \frac{P_{ik}}{P_{jk}}$$

- ▶ Все w_x — векторные представления соответствующих слов

- ▶ Предлагается следующая функция

$$F((w_i - w_j)^T \hat{w}_k) = \frac{F(w_i^T \hat{w}_k)}{F(w_j^T \hat{w}_k)}, \quad F(w_i^T \hat{w}_k) = P_{ik}$$

- ▶ Тогда можно взять $F(x) = \exp(x)$, а w_i таким, что

$$w_i^T \hat{w}_k = \log(P_{ik}) = \log(x_{ik}) - \log(X_i)$$

- ▶ С учётом фиксированности данных о коллекции, перепишем задачу

$$w_i^T \hat{w}_k + b_i + \hat{b}_k = \log(x_{ik}), \quad b_i + \hat{b}_k = \log(X_i), \quad b_x - \text{bias}$$

- ▶ Оптимизация с помощью AdaGrad

- ▶ Оптимизируемый функционал:

$$J = \sum_{i,j=1}^V f(x_{ij})(w_i^T \hat{w}_j + b_i + \hat{b}_j - \log(x_{ij}))^2$$

- ▶ $f(x)$ должна
 - ▶ $f(0) = 0$
 - ▶ $f(x)$ не убывает
 - ▶ $f(x)$ относительно мала для больших x

▶

$$f(x) = \begin{cases} (x/x_{max})^\alpha, & x < x_{max} \\ 1, & \text{else} \end{cases}$$

- ▶ $\alpha = 0.75, x_{max} = 100$

N-символьные эмбединги: FastText

- ▶ N-символьные эмбединги показывают более высокое качество, чем эмбединги символов
 - ▶ При этом они тоже хорошо справляются с редкими и OOV-словами
- Пример: N-граммы для слова apple (min=3, max=6):

<ap, <app, <appl, <apple, app, appl, apple,
apple>, ppl, pple, pple>, ple, ple>, le>

FastText – библиотека (и алгоритм) для получения векторных представлений слов и классификации текстов

- ▶ Архитектурно такая же, как у word2vec (CBOW и skip-gram)
- ▶ Строит эмбединги символьных N-грамм
- ▶ Эмбединг слова получается усреднением эмбедингов N-грамм

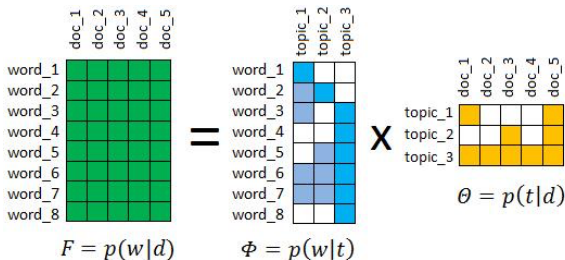
FastText

- ▶ Для борьбы с ростом размерности признакового пространства используется *hashing trick*
- ▶ В FastText есть встроенный классификатор на основе двуслойной полносвязной сети
- ▶ Вход – эмбединг документа как сумма эмбедингов слов
- ▶ На выходе стоит иерархический софтмакс с деревом Хаффмана
- ▶ За счёт использования символьных N-грамм может из коробки хорошо работать с сырыми текстами
- ▶ Ссылки на статьи: <https://arxiv.org/pdf/1607.01759.pdf>,
<https://arxiv.org/pdf/1607.04606.pdf>

Эмбединги предложений и документов

- ▶ Часто в задачах текстовой аналитики нужны эмбединги не слов в документах, а самих документов или их частей
- ▶ Самый простой способ получения – взвешенная сумма эмбедингов слов (например, с tf-idf в качестве весов)
- ▶ Такой подход показывает хорошие результаты и часто используется на практике, но есть и более интересные методы
- ▶ Если есть хороший способ найти эмбединг предложения, вектор документа можно опять-таки получить усреднением по предложениям
- ▶ Качество оценивается по метрике задачи (внешний критерий) или, например, качеству поиска размеченных аналогий (внутренний)

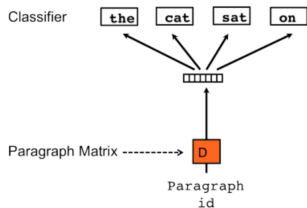
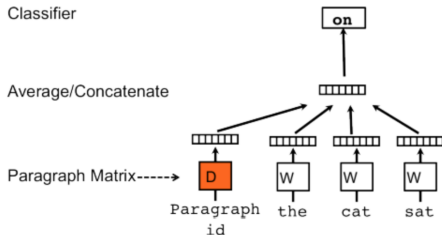
Снова тематическое моделирование



- ▶ Классические тематические модели получают на вход матрицу «мешка слов» или tf-idf и строят два типа распределений:
 - ▶ слов в кластерах-темах
 - ▶ тем в документах
- ▶ Столбцы матрицы «темы-документы» можно использовать в качестве эмбедингов документов
- ▶ Они отражают близость документов со схожей тематикой

doc2vec (paragraph2vec)

- ▶ **doc2vec** – дополненная архитектура word2vec для получения эмбедингов текстовых фрагментов
- ▶ Идеологически схож с тематическими моделями
- ▶ Обученные векторы для слов используются при обучении векторов новых документов



StarSpace

- ▶ Библиотека для решения широкого класса задач машинного обучения
 1. Многоклассовая и multilabel классификация
 2. Построение рекомендательных систем
 3. Ранжирование сущностей
 4. Обучение эмбедингов слов, предложений, документов
 5. Обучение эмбедингов графов
- ▶ Строит эмбединги сущностей разных типов (документы, пользователи, картинки, метки классов) в едином пространстве
- ▶ Эмбединги обучаются под решение конкретной задачи, поэтому лучше всего решать задачу полностью в самом StarSpace, но можно и извлекать полученные векторы
- ▶ Ссылка на статью: <https://arxiv.org/pdf/1709.03856.pdf>, ссылка на репозиторий: <https://github.com/facebookresearch/StarSpace>

Модель StarSpace

- ▶ Дано:

- ▶ коллекция признаков W , например слов, каждый признак $w \in W$ имеет своё векторное представление $v_w \in \mathbb{R}^m$
- ▶ коллекция сущностей D , например документов, каждая сущность $d \in D$ соответствует вектор $v_d = \sum_{w \in d} v_w$

- ▶ На множестве сущностей задано бинарное отношение E^+ (пример: **юзер** кликнул в **документ**)

- ▶ Найти: векторы v_w и, как следствие, v_d

- ▶ Критерий:

$$\sum_{\substack{(d, d^+) \in E^+ \\ d_1^-, \dots, d_k^- \in E^-}} L(\text{sim}(d, d^+), \text{sim}(d, d_1^-), \dots, \text{sim}(d, d_k^-)) \rightarrow \min_{v_w, v_d}$$

Модель StarSpace

- ▶ Критерий:

$$\sum_{\substack{(d, d^+) \in E^+ \\ d_1^-, \dots, d_k^- \in E^-}} L(\text{sim}(d, d^+), \text{sim}(d, d_1^-), \dots, \text{sim}(d, d_k^-)) \rightarrow \min_{v_w, v_d}$$

- ▶ E^+ – пары сущностей, наблюдающиеся в коллекции
 - ▶ E^- – сгенерированные negative samples
 - ▶ sim – функция близости, $\langle \cdot, \cdot \rangle$ или $1 - \cos(\cdot, \cdot)$
 - ▶ L – функционал потерь, зависящий от задачи
-
- ▶ Обучение производится с помощью Adagrad
 - ▶ Для предсказания используются обученные векторы и функция sim (например, близость векторов документа и метки класса)

StarSpace: примеры

- ▶ **Многоклассовая multilabel классификация:**

- ▶ $W = \{ \text{слова, классы} \}$
- ▶ $D = \{ \text{документы (сумма слов), их классы (сумма классов)} \}$
- ▶ $E^+ = \{ \text{документ с одним из своих классов} \}$
- ▶ $E^- = \{ \text{случайные классы} \}$

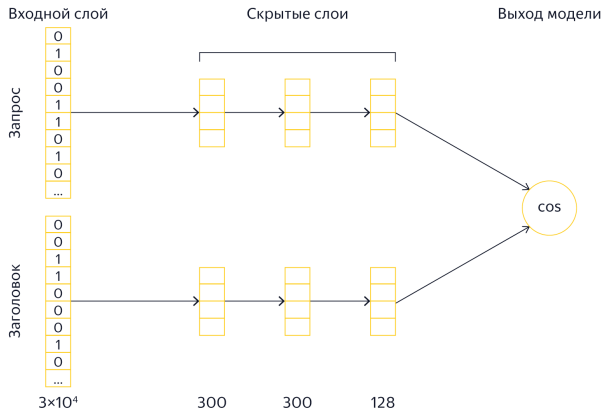
- ▶ **Построение эмбедингов слов:**

- ▶ $W = \{ \text{слова} \}$
- ▶ $D = \{ \text{слова} \}$
- ▶ $E^+ = \{ \text{слово, слово из его контекста} \}$
- ▶ $E^- = \{ \text{случайные слова} \}$

- ▶ **Deep Structured Semantic Model** – модель поиска смысловой близости между предложениями (заголовками и запросами)
- ▶ Представляет собой пример т.н. *сиамской сети*
- ▶ Две одинаковые полносвязные нейронные сети строят представления для пары предложений
- ▶ Между выходами обеих сетей считается функция потерь на основе косинусного расстояния
- ▶ Модель обучается сближать релевантные пары запрос-заголовков, и отдалять не релевантные
- ▶ Ссылка на оригинальную статью:
https://www.researchgate.net/publication/262289160_Learning_deep_structured_semantic_models_for_web_search_using_clickthrough_data
- ▶ Ссылка на статью Яндекса:
<https://habr.com/company/yandex/blog/314222>

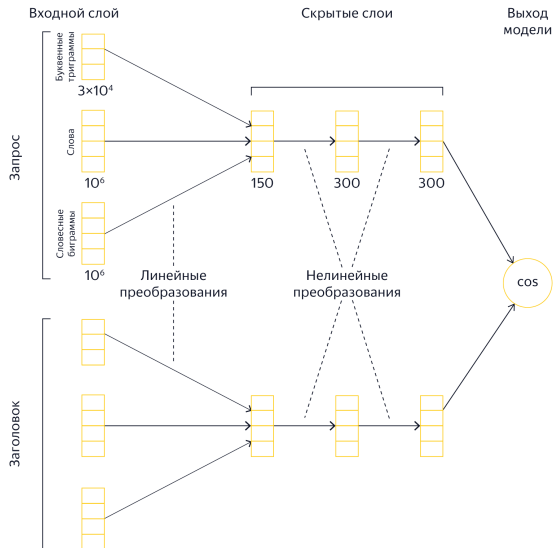
Оригинальный DSSM

В оригинальной модели вход представляет собой векторы встречаемости символьных N-грамм заголовка и запроса



В Яндексе решили разнообразить входные векторы...

DSSM Яндекса



DSSM Яндекса

- ▶ Вход – символьные N-граммы, слова и словосочетания (30K → 2M)
- ▶ Для обучения оригинальной модели положительные примеры набирались по кликам
- ▶ Отрицательные примеры набирались по принципу «был показ – не было клика»
- ▶ Такой подход далёк от идеала: клик не означает релевантность и наоборот
- ▶ Клики производятся только по документам, попавшим в выдачу
- ▶ Попадание в выдачу в обычном поиске происходит при совпадении слов и наличии статистики, а не при близкой семантике

DSSM Яндекса: генерация отрицательных сэмплов

- ▶ Положительные примеры берутся по длинным кликам

Кстати: В общем случае их можно генерировать синтетически с помощью семантически близких замен слов и N-грамм

- ▶ Более разумная генерация отрицательных примеров дала существенный прирост качества
- ▶ **Первая итерация:** генерировались случайные документы – модель стала учиться на наличие/отсутствие слов запроса в заголовках ⇒ провал
- ▶ **Вторая итерация:** генерировались случайные документы, но в их заголовки стали подмешиваться слова из запроса – модель помучилась и научилась отличать реальные вхождения от синтетических ⇒ провал

DSSM Яндекса: генерация отрицательных сэмплов

- ▶ Экспериментальным путём было установлено, что результат сильно улучшается при использовании стратегии наподобие той, что применяется в GAN:
 - ▶ Модель обучается итеративно
 - ▶ Сперва отрицательными считаются случайные документы
 - ▶ Обучившись на них, модель находит новые документы, которые считает релевантными, но они с большой вероятностью случайные
 - ▶ Модель переобучается на лучших из этих отрицательных документов
 - ▶ Процесс останавливается, когда лучшими из предсказанных документов становятся в основном релевантные

Мультиязычные эмбединги

- ▶ Обычно семантические эмбединги строятся внутри в рамках одного языка
- ▶ Хочется иметь пространство, в котором векторы слов на разных языках, но с общей семантикой, были близки
- ▶ Тогда можно обучать модель на одном языке, а результаты использовать для другого
- ▶ Схожая задача явным образом решается в моделях машинного перевода путём оптимизации функционала глубокой нейросети
- ▶ Но строить эмбединги можно неявно (даже без чёткой разметки) и значительно проще с вычислительной точки зрения

Виды подходов

- ▶ **Моноязычный:** обучаем эмбединги для каждого языка отдельно и обучаем линейные преобразования между пространствами
- ▶ **Псевдо-мультязычный:** обучаем эмбединги одновременно на синтетическом корпусе, в котором перемешаны слова на разных языках
- ▶ **Мультязычный:** обучаемся на параллельном корпусе так, чтобы эмбединги похожих слов на разных языках были близки

Рассмотрим ниже несколько примеров.

Подробный структурированный обзор есть доступен по адресу
<http://runder.io/cross-lingual-embeddings/>

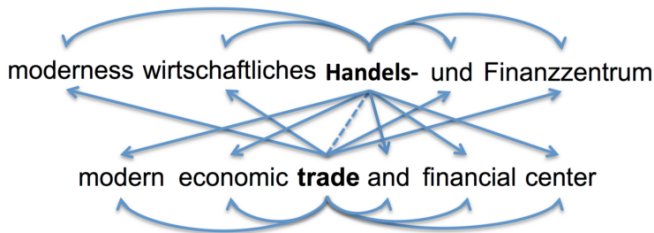
Linear projection

- ▶ Замечено, что схожие сущности (числа, имена животных) образуют в своих языковых пространствах на английском и испанском схожие по форме кластеры
- ▶ Отсюда вывод, что можно обучить независимо два векторных пространства и далее построить линейное преобразование между ними
- ▶ Для обучения преобразования отбирается 5000 наиболее популярных слов в исходном языке, для них подбираются переводы
- ▶ Дальше на этих 5000 парах с помощью SGD обучается матрица преобразования векторов
- ▶ Mikolov, T., Le, Q. V., Sutskever, I. (2013). Exploiting Similarities among Languages for Machine Translation. Retrieved from <http://arxiv.org/abs/1309.4168>

Random translation replacement

- ▶ Берётся исходный язык, все слова из него переводятся на целевой с помощью Google Translate
- ▶ На основании корпуса на исходном языке формируется корпус, в котором каждое слово с вероятностью 50% заменяется на свой перевод
- ▶ Далее запускается обучение модели CBOW (как в w2v)
- ▶ Gouws, S., Søgaard, A. (2015). Simple task-specific bilingual word embeddings. NAACL, 1302-1306.

Bilingual skip-gram



- ▶ Очевидно необходима разметка выравнивания между словами
- ▶ Luong, M.-T., Pham, H., Manning, C. D. (2015). Bilingual Word Representations with Monolingual Quality in Mind. Workshop on Vector Modeling for NLP, 151-159.

Multilingual unsupervised and supervised embeddings

- ▶ Есть предобученные векторы FastText для большого числа языков
- ▶ Как и в других мооязычных подходах, здесь обучается линейное преобразование из исходного пространства в целевое
- ▶ Но здесь используются никакие параллельные данные
- ▶ Вместо этого обучается GAN
 - ▶ параметрами генератора является матрица преобразования
 - ▶ дискриминатор учится отличать преобразованные векторы исходного пространства от векторов целевого
- ▶ <https://github.com/facebookresearch/MUSE>

Выводы: чем пользоваться в жизни

- ▶ В общем случае самым простым, легковесным и эффективным инструментом является FastText
- ▶ В ситуации, когда нужно работать с большими текстами и нужна интерпретируемость векторов – подойдут тематические модели
- ▶ Также при наличии разметки (пусть и синтетической) бывает полезным обучать модели наподобие DSSM
- ▶ Глубокие нейросетевые эмбединги в индустрии используются нечасто:
 - ▶ они требуют много железа, данных и времени (даже на дообучение)
 - ▶ скорость вывода может быть недостаточной, нужно думать, как ускорять (например, делать дистилляцию сети)
 - ▶ в жизни улучшение качества по сравнению с более простыми подходами может оказаться слишком небольшим
- ▶ Но если есть время на решение технических вопросов и дообучение, и точно понятно, что работать будет сильно круче, то они могут стать лучшим решением

Что дальше?

- ▶ context2vec
- ▶ USE
- ▶ ULMFIT
- ▶ ELMo
- ▶ BERT
- ▶ ERNIE
- ▶ GPT-2
- ▶ ...

А на сегодня всё!