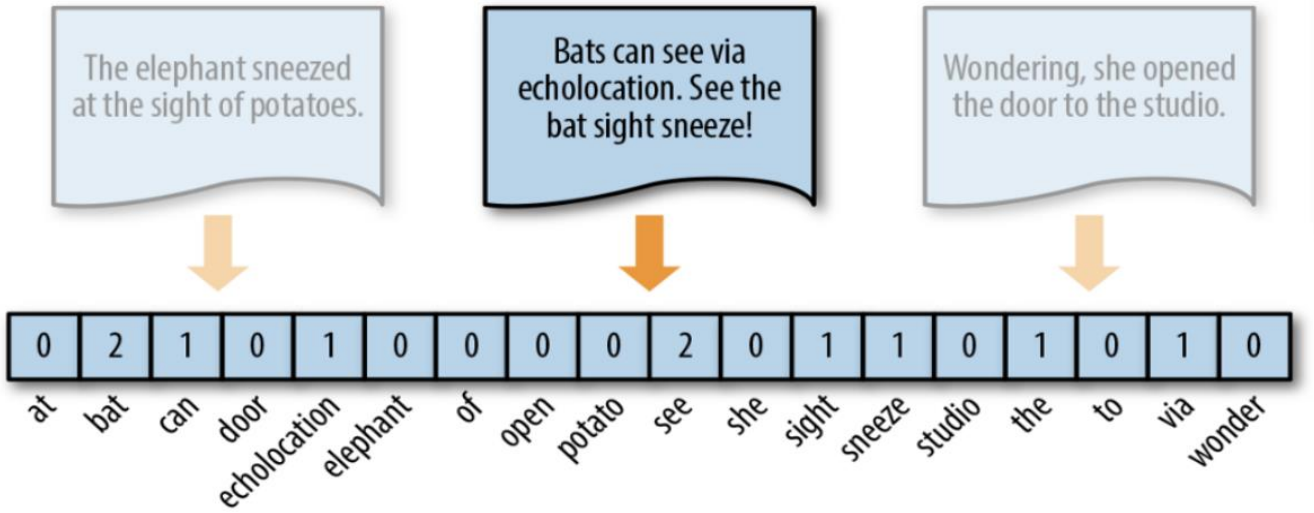
Представления для текстов

1. Bag of Words or One Hot Encoding

- Заводим словарь, состоящий из всех слов в выборке
- Делаем признак-индикатор для каждого слова из словаря



An obvious limitation of this approach is it does not encode any idea of meaning or word similarity into the vectors.

If the text it too large the vector will be too large. Especially if we add n-grams

- Слишком много признаков
- Не учитываем порядок слов
- Не учитываем смыслы слов
- Семантически похожие тексты могут иметь очень разные представления

- **Можно добавлять n-граммы**

На практике каждое слово в языке может иметь несколько смыслов, а мы даже не учитываем их порядок. Чтобы передать модели больше информации, можно передать не только отдельные слова, но и их словосочетания или n-граммы. N-граммы последовательность из n подряд идущих слов текста. Последовательности накладываются друг на друга.

Например, из предложения "*Ты же знаешь, что Максим строго проверяет работы*" получатся следующие 3-граммы:
"ты же знаешь", "же знаешь что", "знаешь что Максим", "что Максим строго", "Максим строго проверяет", "строго проверяет работы".

Каждая n-грамма для мешка слов считается отдельным словом. На практике часто берут n-граммы сразу разных размеров, (например, от 1 до 3). Заметим, что размер мешка слов растёт экспоненциально с ростом n.

Meaningful n-gramms are often called **collocations.** How to detect meaningful n-gramms?

Delete:

- High-frequency n-gramms
  - Articles, prepositions
  - Auxiliary verbs (to be, to have, etc.)
  - General vocabulary

- Low-frequency n-gramms
  - Typos
  - Combinations that occur 1-2 times in a text

- **Лемматизация и стемминг**

Заметим, что одно и то же слово может встречаться в различных формах (особенно для русского языка), но описанные выше методы интерпретируют их как различные слова, что делает признаковое описание избыточным. Устранить эту проблему можно при помощи лемматизации и стемминга.

Стемминг это процесс нахождения основы слова. В результате применения данной процедуры однокоренные слова, как правило, преобразуются к одинаковому виду. Например, вагон-вагон, вагонов-вагон и важная-важн

Лемматизация процесс приведения слова к его нормальной форме (лемме):

для существительных им. падеж, ед. число;
для глаголов, причастий, деепричастий глагол в инфинитиве

Collocations: context is all you need

- Coocurrence counters in a window of fixed size
  - n_{uv} states for the number of times we've seen word *u* and word *v* together in the window
- Better solution: Pointwise Mutual Information (PMI)

$$PMI = log\frac{p(u,v)}{p(u)p(v)} = log\frac{n_{uv}n}{n_u n_v}$$

- Much better solution: **Positive PMI (pPMI)**

$$pPMI = \max(0, PMI)$$

2. TF-IDF Representation

Очевидно, что не все слова полезны в задаче прогнозирования. Например, мало информации несут слова, встречающиеся во всех текстах. Это могут быть как стоп-слова, так и слова, свойственные всем текстам выборки (в текстах про автомобили употребляется слово - автомобиль).

Эту проблему решает TF-IDF преобразование текста. Вычисляются две величины:

TF (Term Frequency) количество вхождений слова в отношении к общему числу слов в тексте:

$$TF(t) = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ the\ document}$$

IDF (Inverse Document Frequency):

$$IDF(t) = log_e(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it})$$

Тогда для каждой пары (слово, текст) (t, d) вычислим величину:

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D).$$

Это и будем значением нового признака (вместо количества каждого слова в тексте в случае мешка слов). Отметим, что значение tf(t, d) корректируется для часто встречающихся общеупотребимых слов при помощи значения idf(t,D).

TF-IDF example

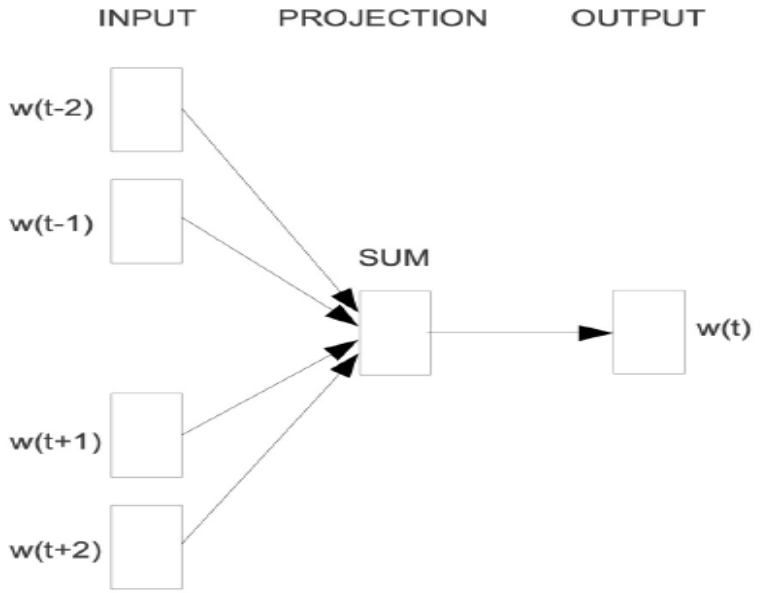
- *Sentence A:* The car is driven on the road.
- *Sentence B:* The truck is driven on the highway.

Word	TF		IDF	TF * IDF	
	A	B		A	B
The	1/7	1/7	log(2/2)=0	0	0
Car	1/7	0	log(2/1)=0.3	0.043	0
Truck	0	1/7	log(2/1)=0.3	0	0.043
Is	1/7	1/7	log(2/2)=0	0	0
Driven	1/7	1/7	log(2/2)=0	0	0
On	1/7	1/7	log(2/2)=0	0	0
The	1/7	1/7	log(2/2)=0	0	0
Road	1/7	0	log(2/1)=0.3	0.043	0
Highway	0	1/7	log(2/1)=0.3	0	0.043

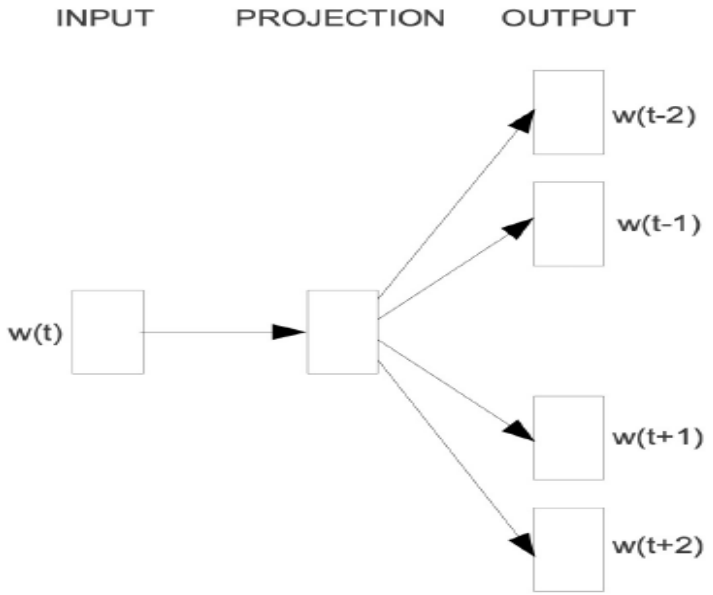
3. word2vec

- Попробуем обучить вектор-представление для каждого слова
- Что потребовать от такого представления?
- Важная идея: если выкинуть слово, то оно должно хорошо восстанавливаться по представлениям соседних слов
- Может применять и при работе с изображениями

There are basically two versions of Word2Vec — Continuous Bag of Words (CBOW) and Skip-Gram. The CBOW model learns the embedding by predicting the current word based on its context (surrounding words). The Skip-Gram model learns by predicting the surrounding words (context) given a current word.



CBOW



Skip-gram

Continuous BOW (CBOW)

$$p(w_i | w_{i-h}, \ldots, w_{i+h})$$

Predict center word from (bag of) context words

- Predicting one word each time
- Relatively fast

Skip-gram

$$p(w_{i-h}, \ldots w_{i+h} | w_i)$$

Predict context (”outside”) words (position independent) given center word

- Predicting context by one word
- Much slower
- Better with infrequent words

В лингвистике существует дистрибутивная гипотеза, согласно которой слова, встречающиеся в похожих контекстах, имеют похожие смыслы. Будем строить представления для слов, опираясь на эту гипотезу: чем в более похожих контекстах встречаются два слова, тем ближе должны быть соответствующие им векторы.

Usually d has the order of tens or hundreds of dimensions

Итак, мы хотим для каждого слова w из словаря W найти вектор $\vec{w} \in \mathbb{R}^d$. Пусть дан некоторый текст $x = (w_1 \dots w_n)$. Контекстом слова w_j будем называть слова, находящиеся от него на расстоянии не более K — то есть слова $w_{j-K}, \dots, w_{j-1}, w_{j+1}, \dots, w_{j+K}$. Определим через векторы слов вероятность встретить слово w_i в контексте слова w_j:

$$p(w_i \mid w_j) = \frac{\exp(\langle \vec{w}_i, \vec{w}_j \rangle)}{\sum_{w \in W} \exp(\langle \vec{w}, \vec{w}_j \rangle)}$$

Тогда для выборки текстов $X = \{x_1, \dots, x_\ell\}$, где текст x_i имеет длину n_i, можно определить правдоподобие и максимизировать его:

$$\sum_{i=1}^{\ell} \sum_{j=1}^{n_i} \sum_{\substack{k=-K \\ k \neq 0}}^{K} \log p(\vec{w}_{j+k} \mid \vec{w}_j) \to \max_{\{\vec{w}\}_{w \in W}}$$

Word representations via matrix factorization

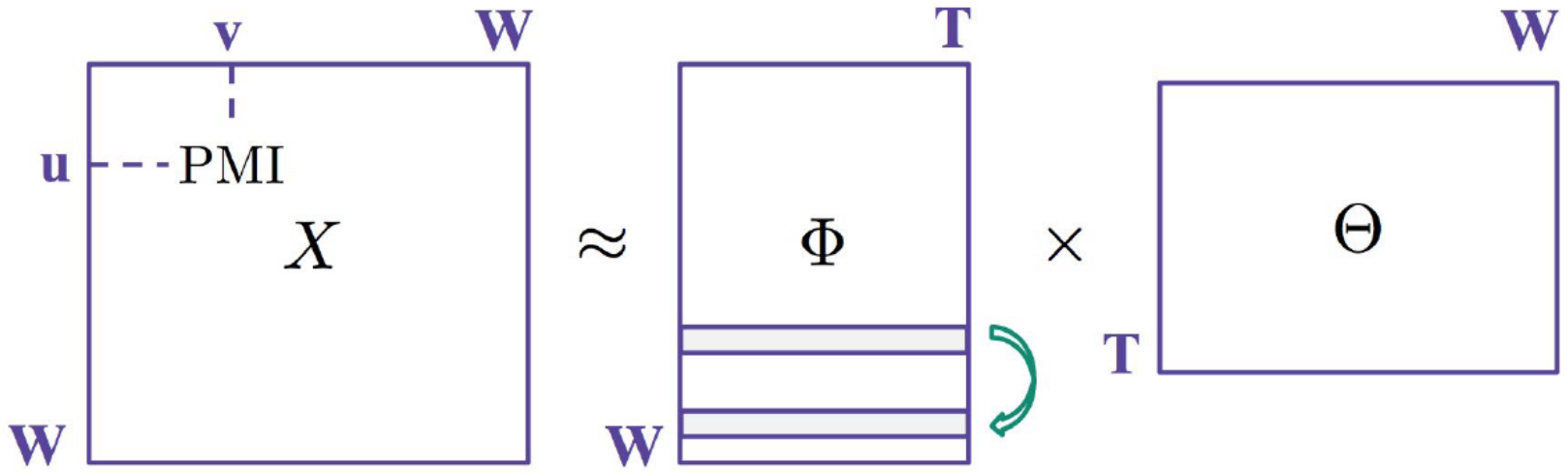
- **Input: PMI, word coocurrences, etc**.

  Pointwise Mutual Information (PMI) – we get the matrix of the form:

$$PMI = log \frac{p(u,v)}{p(u)p(v)} = log \frac{n_{uv} n}{n_u n_v}$$

- **Method: dimensionality reduction (SVD)**

- **Output: word similarities**

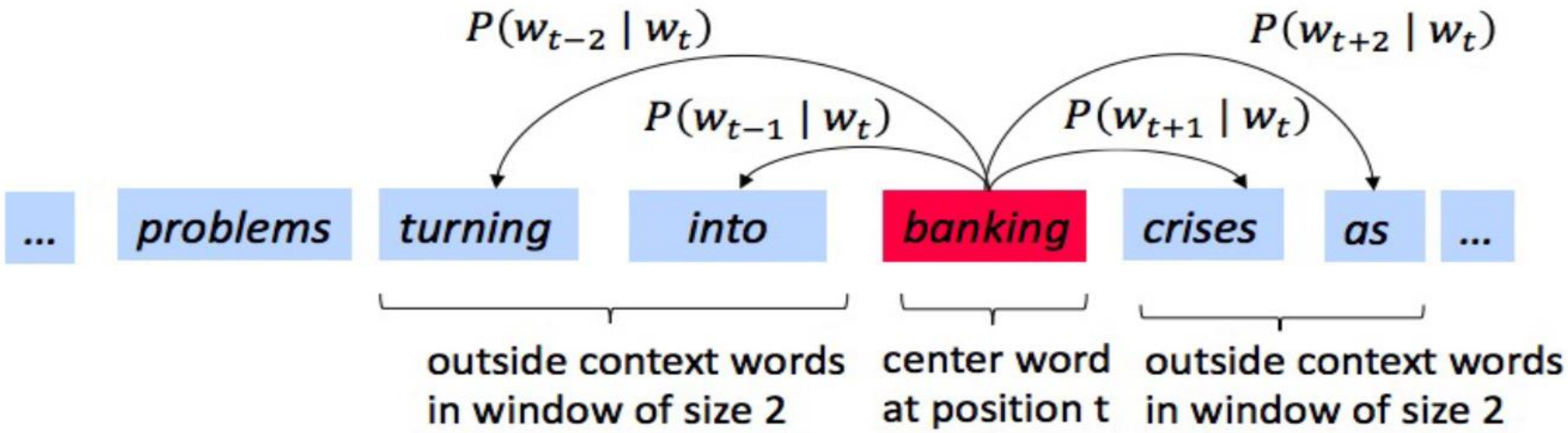


Skip-gram model

- Вероятность встретить слово w_O рядом со словом w_I:

$$p(w_O|w_I) = \frac{\exp\left(\left\langle v'_{w_O}, v_{w_I} \right\rangle\right)}{\sum_{w \in W} \exp\left(\left\langle v'_w, v_{w_I} \right\rangle\right)}$$

- W — словарь
- v_w — «центральное» представление слова
- v'_w — «контекстное» представление слова



- Функционал для текста $T = (w_1 w_2 \dots w_n)$:

$$\sum_{i=1}^{n} \sum_{\substack{-c \le j \le c \\ j \neq 0}} \log p(w_{i+j} | w_i) \to \max$$

Подбираем вектора для каждого слова таким образом, чтобы максимизировать вероятность (правдоподобие)

Данный функционал можно оптимизировать стохастическим градиентным спуском. В результате обучения мы получим представления для слов, которые, как показывает практика, будут обладать многими интересными свойствами и, в том числе, близкие по смыслу слова будут иметь близкие векторы.

- Считать знаменатель ОЧЕНЬ затратно
- Значит, и производные считать тоже долго

Что делать ??? Решение – negative sampling

Схема обучения:

- DataSet

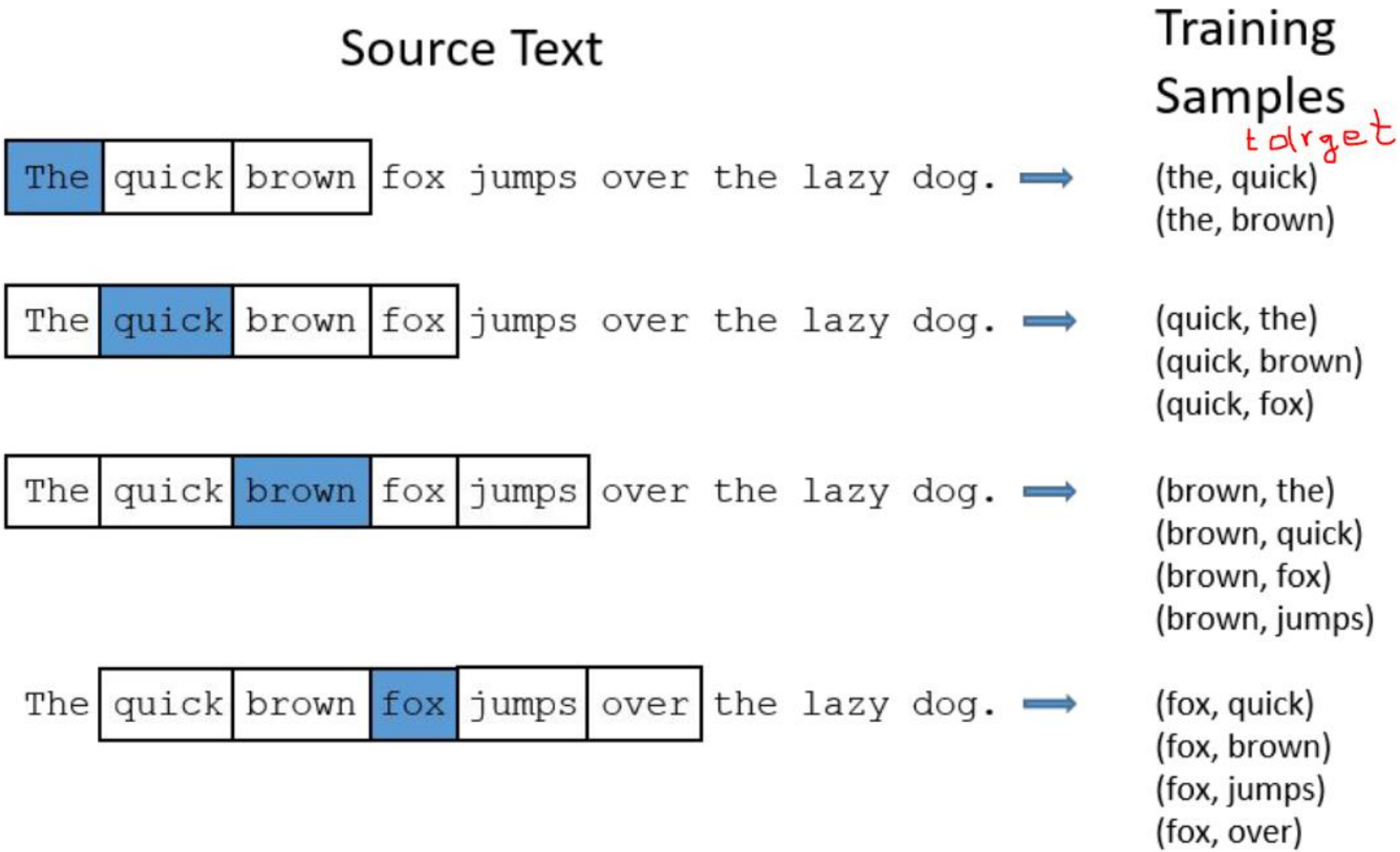
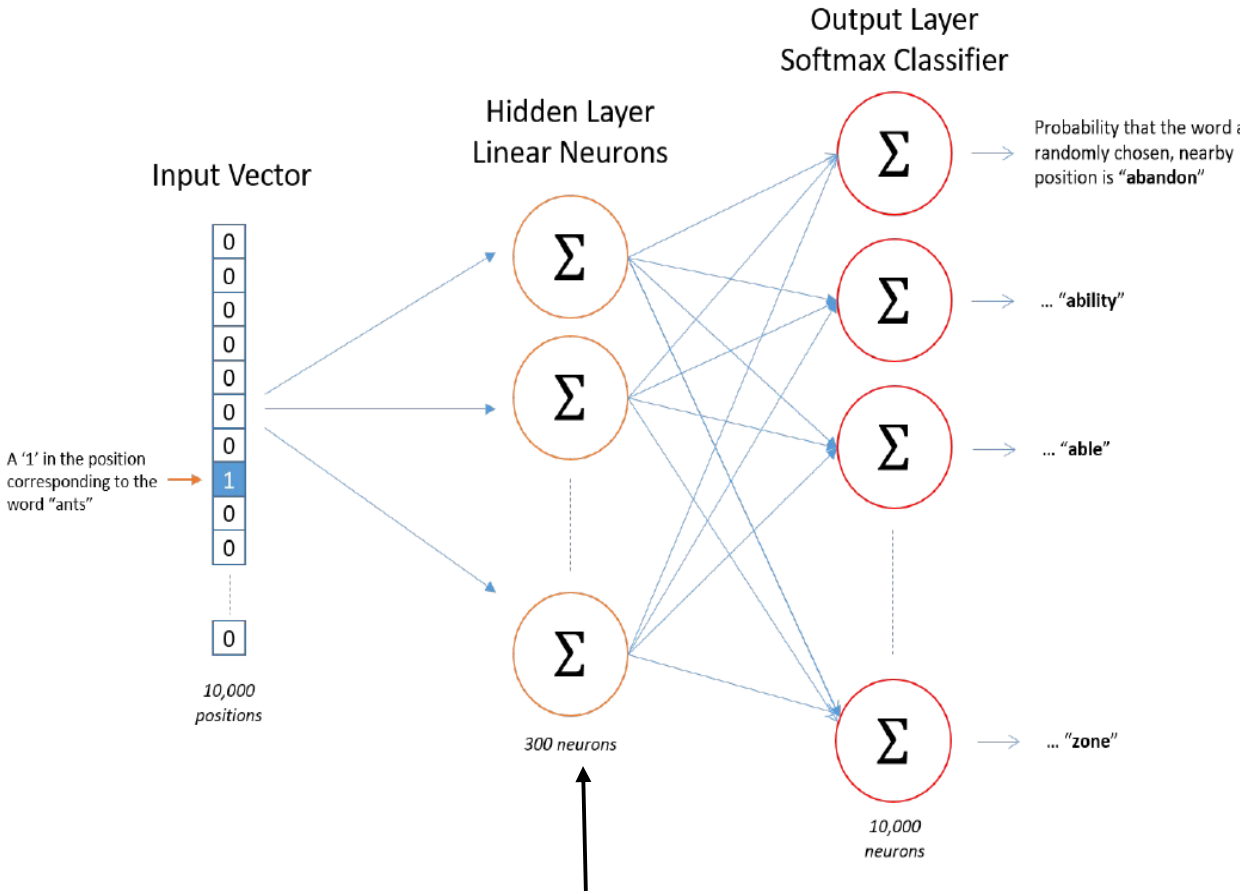
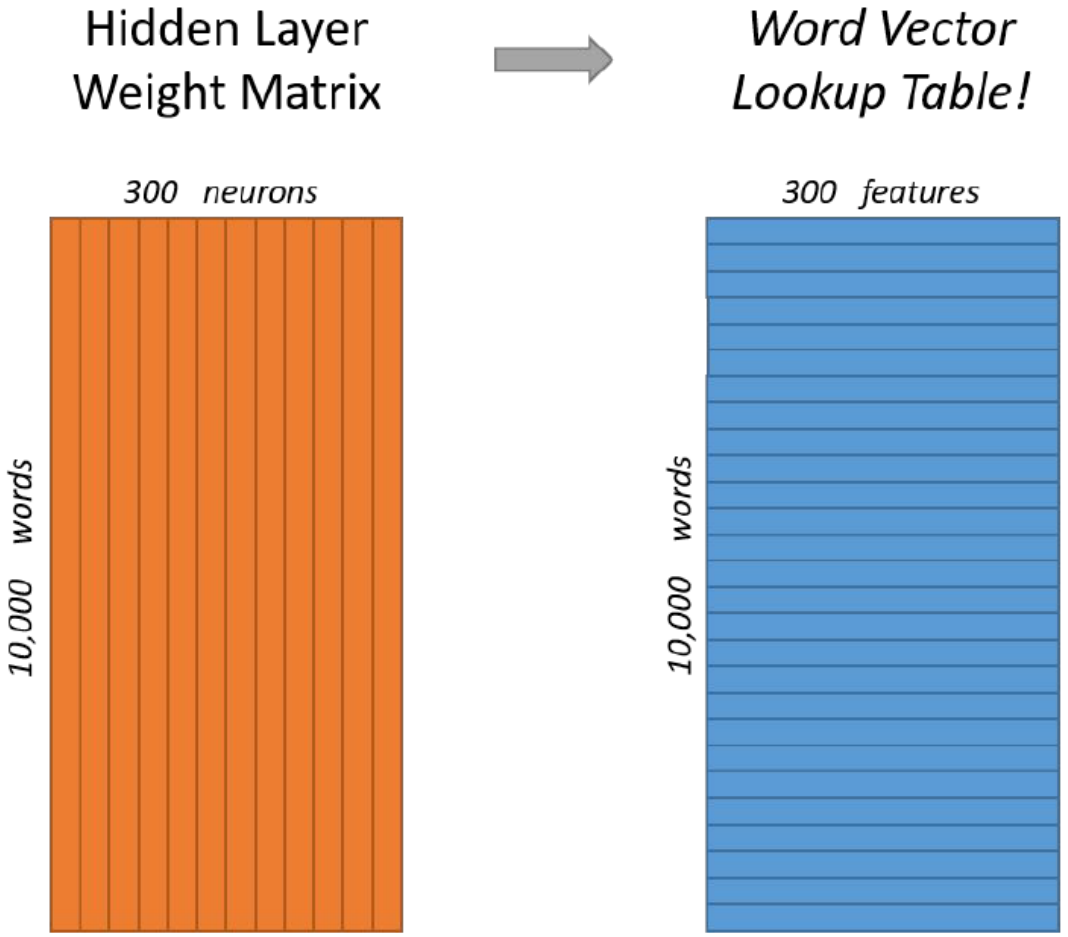


Схема обучения:

Сетка для обучения:





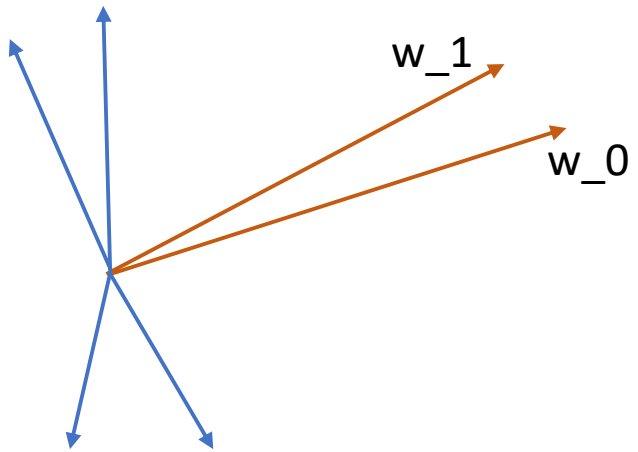
Скрытый слой можно взять в качестве вектора для слова на входе

Negative sampling

Можно использовать не вероятности для слов, заменим вероятности на некоторое выражение от скалярных произведений (которое уже нельзя интерпретировать уже как вероятность):

$$p(w_O|w_I) \Rightarrow \log\sigma\left(\left\langle v'_{w_O}, v_{w_I}\right\rangle\right) + \sum_{i=1}^{k} \log\sigma\left(-\left\langle v'_{w_i}, v_{w_I}\right\rangle\right) \longrightarrow max()$$

Максимизируем данное выражение по векторам, первое слагаемое должно быть как можно больше, т.е. вектора для слова O и I должны быть сонаправленными. Вторая сумма по случайным k словам из словаря, т.е. мы хотим взять вектор O, I и k случайных векторов и градиентным спуском повернуть вектора таким образом чтобы вектора O и I смотрели в одну сторону, а k случайных векторов в другие стороны.
P.S. Может быть и такое что среди k случайных векторов окажется контекстное слово – в этом случае мы сделаем шаг назад на градиентном спуске.
На самом деле у нас очень много слов в словаре как правило, и во вторых мы можем встретить слова O и I несколько раз, тогда уже это будут другие k случайных слов ну и в третьих мы пользуемся градиентным спуском, значит мы несколько раз будем перебирать выборку.



Отличие от предыдущего случая – в negative sampling мы будем обновлять веса только для k+2 векторов, в то время как в чистом skip-gram обновление идет для всех слов в словаре.

Случайные слова выбирается не совсем случайно а согласно распределению:

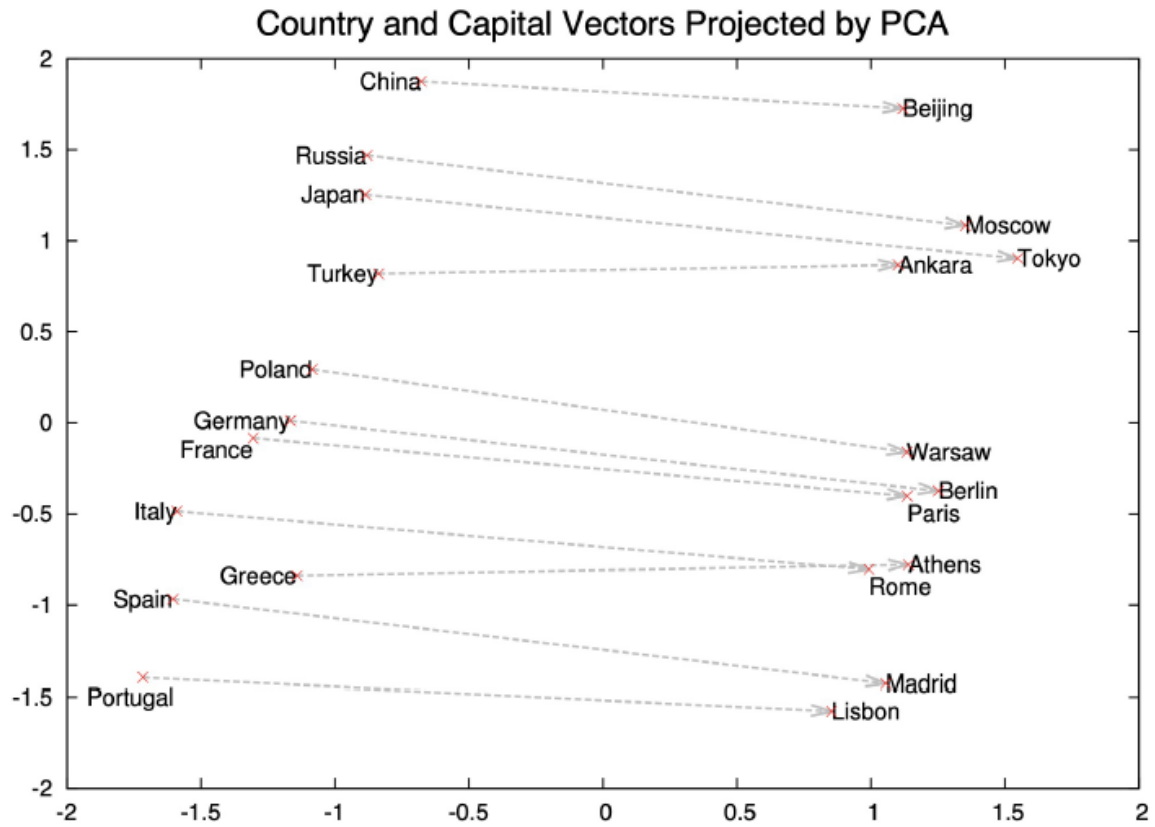
- Слово w генерируется с вероятностью $P(w)$ — шумовое распределение
- $P(w) = \frac{U(w)^{\frac{3}{4}}}{\sum_{v \in W} U(v)^{\frac{3}{4}}}$, $U(v)$ — частота слова v в корпусе текстов

Частые слова выбираем часто в качестве шумовых (негативных), а редкие слова соответственно редко.
Это приводит к лучшим результатам.

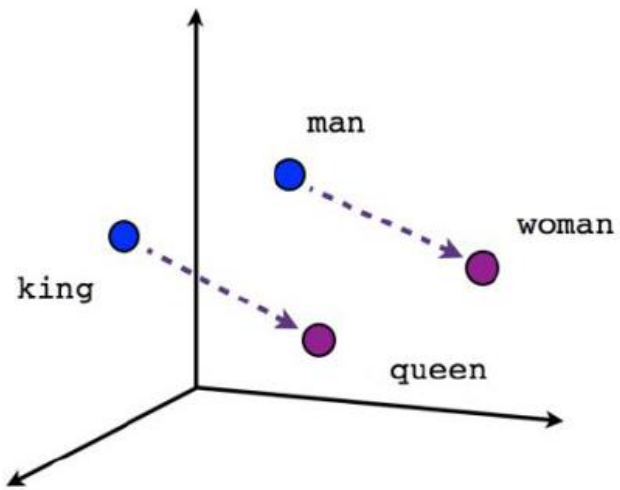
Если взять вектора полученные для двух слов и просуммировать их а затем найти наиболее близкие вектора к сумме мы получаем:

Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De

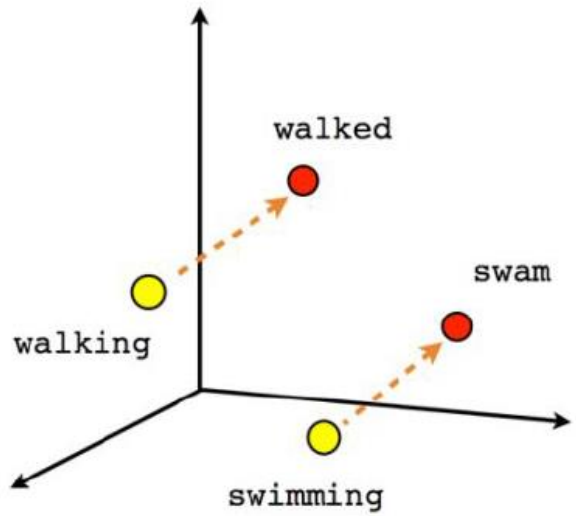
Если взять вектора полученные для стран и их столиц а затем применить PCA и снизить размерность до двух то как видим образуется два кластера: кластер из названия стран и кластер из названия столиц, причем вектор перехода из точки соответствующий определенной стране в точку отображающую столицу этой страны приблизительно имеет одно и то же направление!!!



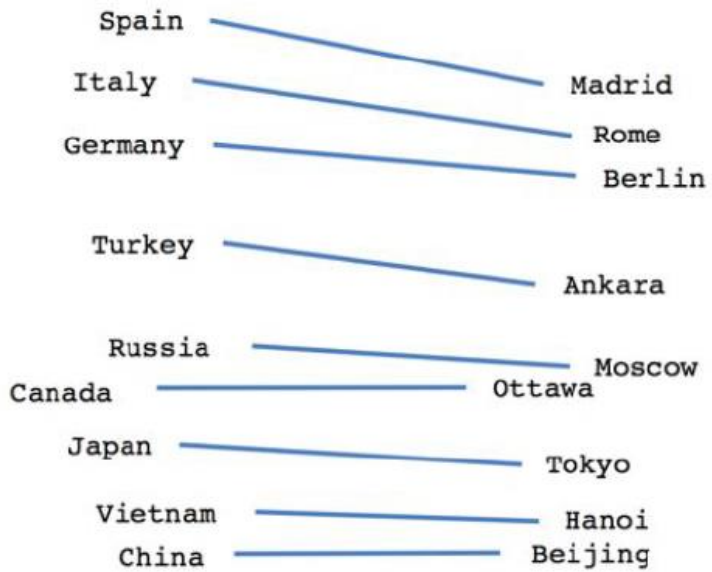
- **Word2vec** (Mikolov et al. 2013) - a framework for learning word embeddings



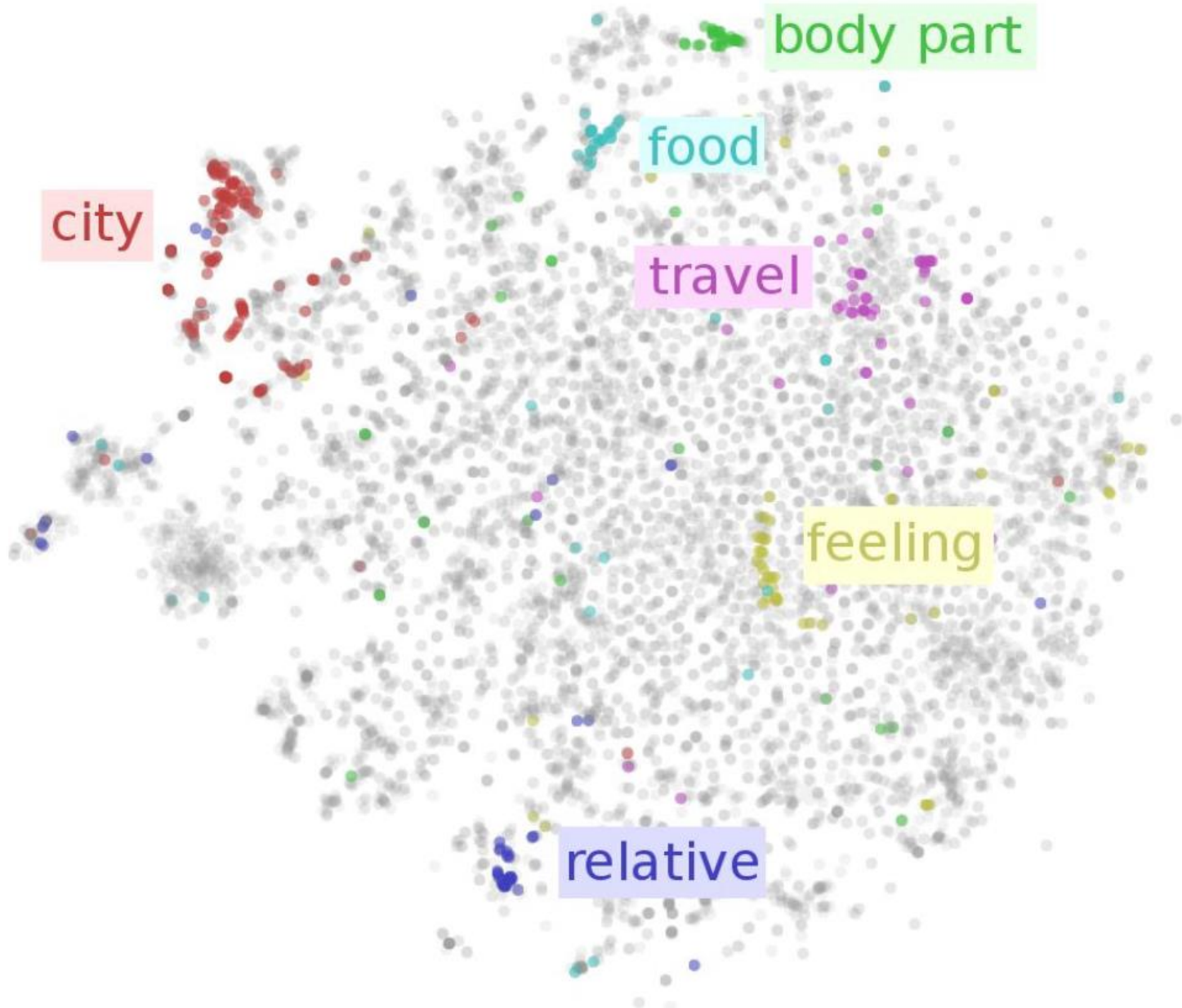
Male-Female



Verb tense



Country-Capital


body part
food
city
travel
feeling
relative

- Яркий пример self-supervision
- Сейчас находит применения для изображения и даже для табличных данных
- Оказывается, в данных очень много информации даже без разметки

Проблемы word2vec

- Не учитываем структуру слов
- Не закладываем никакой априорной информации о разных формах одного слова
- Не умеем обрабатывать опечатки

4. FastText (от Facebook)

- Заменим каждое слово на «мешок»
- Слово w заменяется на набор токенов $t_1, \ldots, t_n$
- «руслан» -> (<руслан>, <ру, рус, усл, сла, лан, ан>)
- Мы обучаем векторы токенов: $v_1, \ldots, v_n$ (на самом деле есть «центральные» и «контекстные» версии всех векторов)
- $z_w = \sum_{i=1}^{n} v_{t_i}$ — вектор слова
- Все остальные детали — как в word2vec

- Решается например проблема с опечатками:
- «руслан» -> (<руслан>, <ру, рус, усл, сла, лан, ан>)

За счет того что в конце идет суммирование вектор для “Румлан” слабо будет отличаться от вектора для “Руслан”, так как три токена остались неизменными. Таким образом мы лучше учитываем структуру слова

Работа с текстом

Стоит задача, например классификация текста. Как использовать векторное представление слов:

- Векторные представления строятся для слов
- Можно просто усреднить по всем словам — получим

признаки
для текста

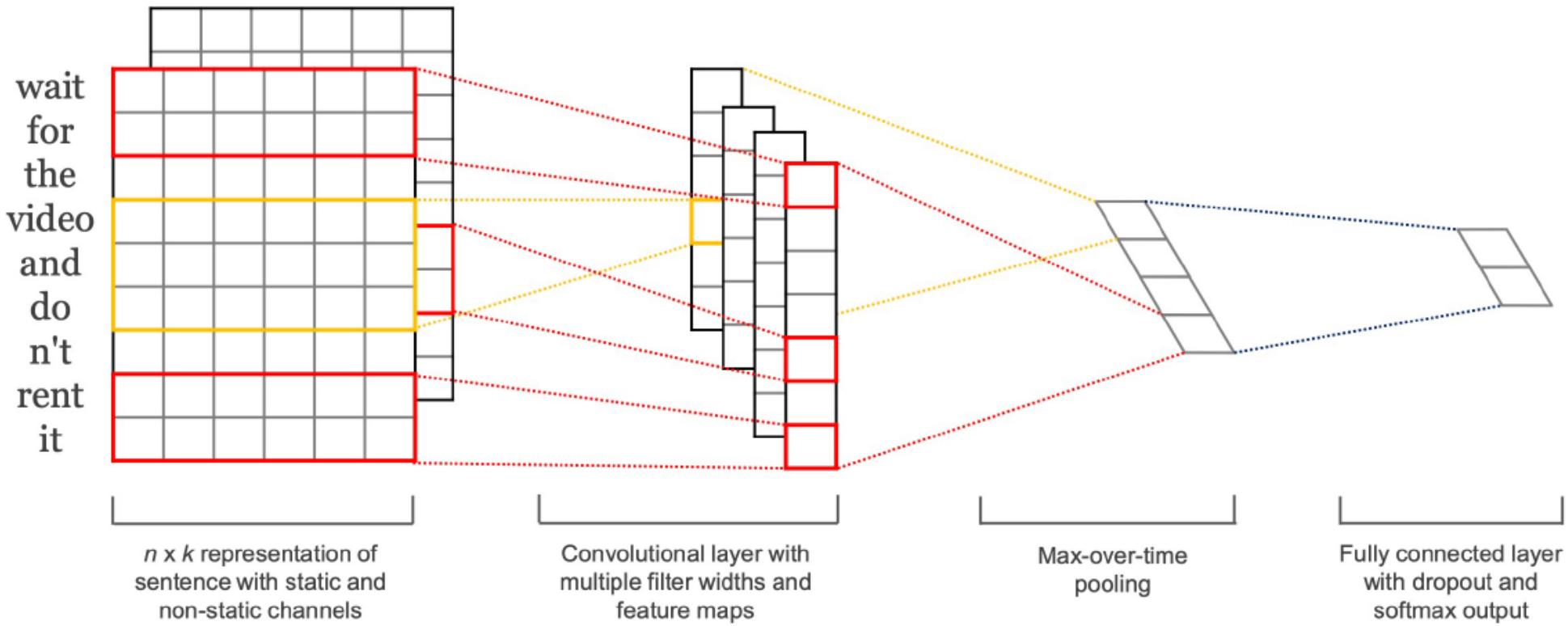
- Можно усреднять с весами
- Можно ли умнее?

Однако вектор для каждого слова будет один, но слово в дальнейшем может встретиться в разных контекстах.

А может воспользоваться идеями из классификации картинок, например представить в предложении слова векторами, у нас образуется матрица (количество слов *x* размерность вектора)

Далее брать различные фильтры, но так чтобы вторая размерность фильтра совпадала с размерностью представления слов и пройтись фильтром по всему предложению. Далее можно воспользоваться максимумом. Причем первая размерность фильтров не обязательна должна быть одной и той же для всех фильтров. Например фильтры размера (2*x*d) будут обучаться на пары слов. К примеру если у нас будет фильтр реагирующий на слова “interesting *movie* ” то этот же фильтр будет реагировать на слова “interesting *film* ” так как вектора для movie и film будут близки , т.е. фильтры будут реагировать на конкретные смыслы пар слов. т.е. мы ищем локальные паттерны.

CNN для последовательностей



CNN для последовательностей

Заранее получили вектор представление

Одновременно обучаем вектор представление

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

https://arxiv.org/abs/1408.5882

Минусы

- Ищем выразительные «локальные» комбинации
- Не пытаемся понять смысл текста в целом

1. Марковские модели

- Предположение: наличие конкретного слова в тексте объясняется только k словами перед ним
- $p(w_1, \dots, w_n) = p(w_1)p(w_2|w_1) \dots p(w_n|w_{n-1}, \dots, w_{n-k})$
- $p(w_n|w_{n-1}, \dots, w_{n-k})$ — можно оценить
- Как часто встречается слово w_n после последовательности из слов $w_{n-1}, \dots, w_{n-k}$?
- Обычно делают со сглаживанием

Как работает к примеру для k = 2?

Берем слово, например ‘usually’ и смотрим какое слово часто встречается после него, генерируем его и т.д. Но иногда могут возникать циклы, чтобы этого избежать не будем смотреть на самое частое слово после ‘usually’ а рассмотрим на распределение по вероятности для слов встречающ. после ‘usually’ и сгенерируем какое то одно слово их этого распределения.

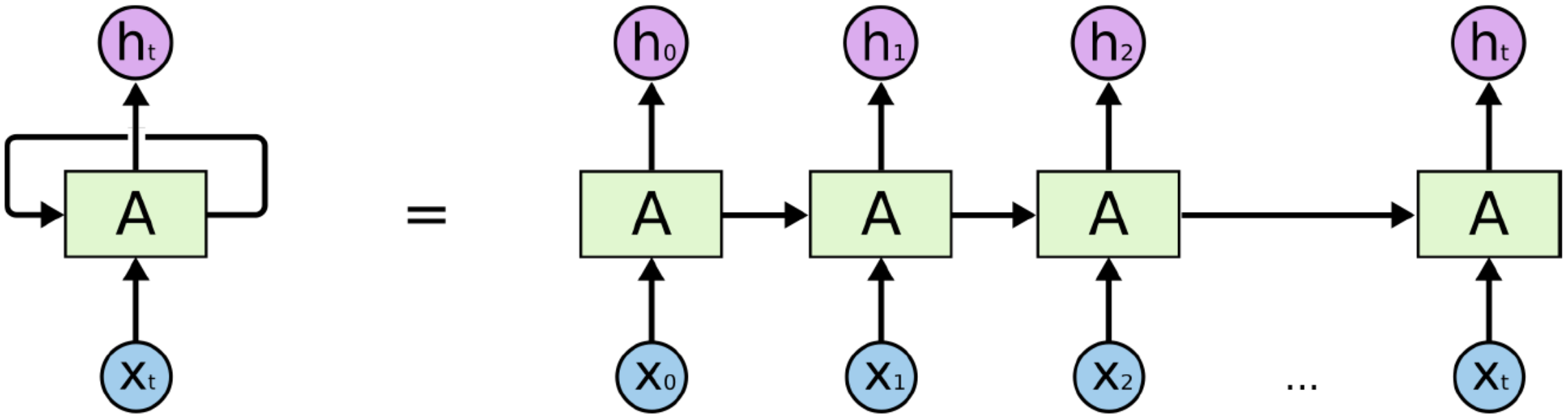
- Последовательность: $x_1, x_2, \dots, x_n, \dots$
- Читаем слева направо
- h_t — накопленная информация после чтения t элементов (вектор)
- $h_t = f(W_{xh}x_t + W_{hh}h_{t-1})$
- Если хотим что-то выдавать на каждом шаге: $o_t = f_o(W_{ho}h_t)$
- x_i — либо one-hot вектор, либо векторное представление (word2vec, fasttext, …)

h_t – вектор в котором хранится информация, т.е. на этапе t вектор состояние обновляется использую как предыдущее его значение так и на то что принимает на вход, новое слово. При этом размерность скрытого состояние фиксированная и не меняется. По сути мы пытаемся всю информацию, которая содержится в тексте впихнуть в вектор скрытого состояния, что не есть хорошо. Какой бы огромной (но правдоподобной) не была размерность мы не сможем одним вектором закодировать например “Войну и мир”. Таким образом данный метод плохо работает для длинных, сложных текстов.

Лингвистическая модель – предсказать слово на t-ом месте по тому что модель узнала из текста до (t-1)-го слова.

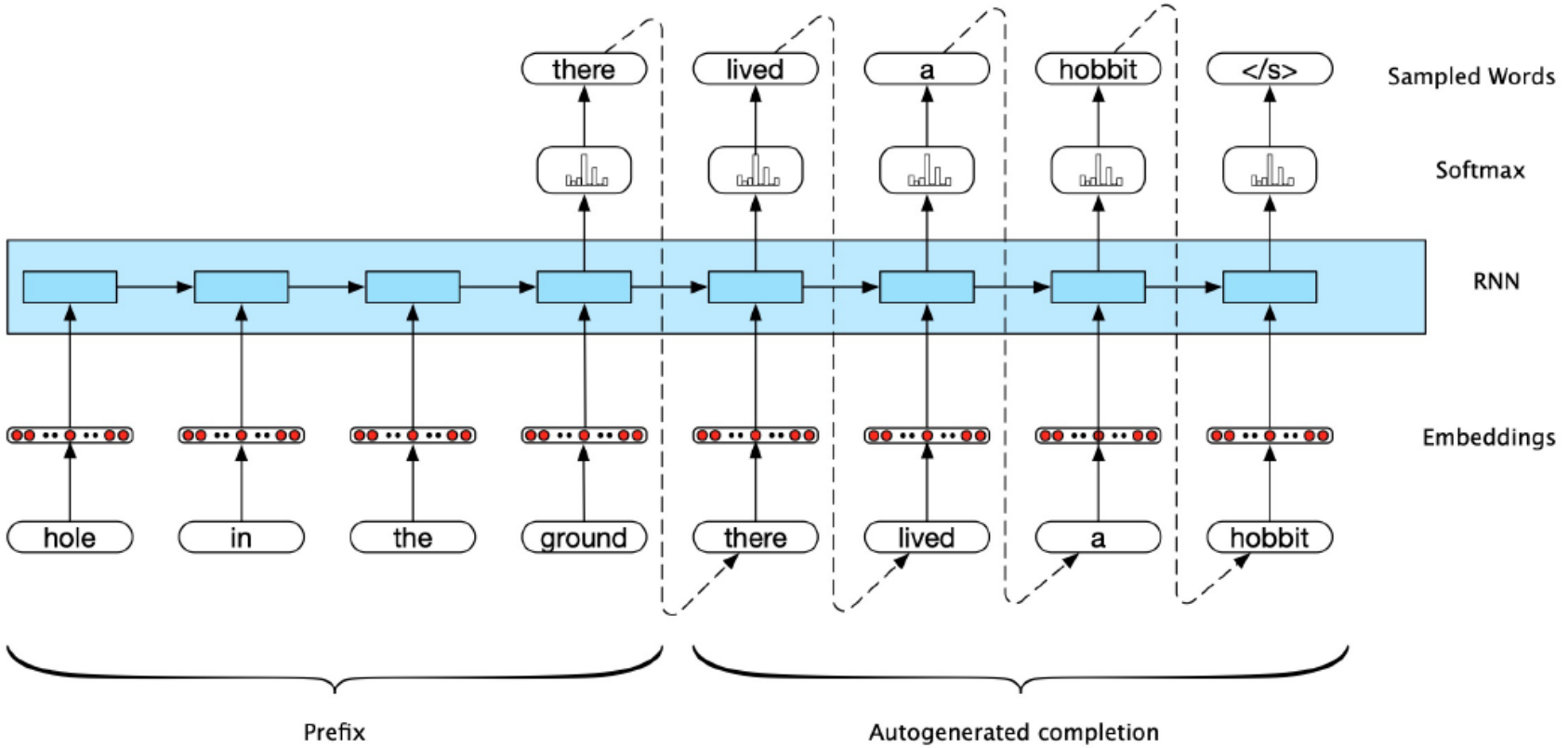
- Типичный случай: $o_t \in \mathbb{R}^N$ ← *Вектор предсказаний имеет размерность словаря слов в тексте*
- N — размер словаря *т.е для каждого слова в словаре предсказываем вероятность того что оно будет следующим*
- То есть предсказываем вероятность того, что здесь стоит конкретное слово
- Предсказываем следующее слово

- Типичный случай: $o_t \in \mathbb{R}^N$ ← Предсказываем часть речи
- N — количество частей речи
- POS tagging ← Part of speech tagging – количество частей речи



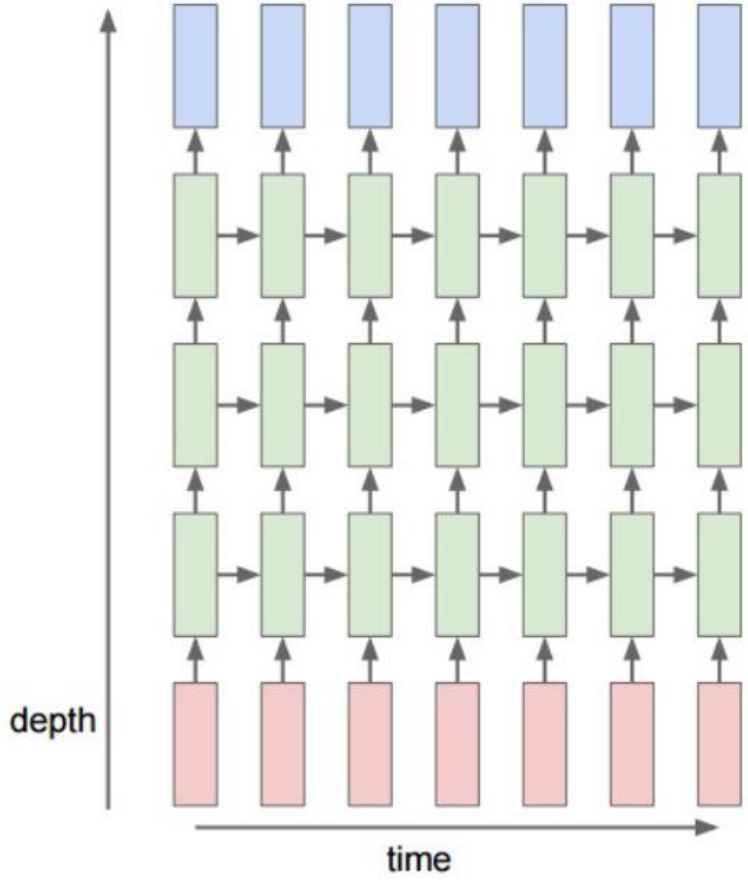
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Как сгенерировать текст? Подаем слово на обученную RNN, далее используя вектор скрытого состояния на выход выдаем какое-то распределение для следующего слова, сгенерируем слово из этого распределения и подаем уже его на вход и тд.



Для всех кто собирается погрузиться в nlp, обработку звука https://web.stanford.edu/~jurafsky/slp3/10.pdf

Можно делать многослойные RNN



Выходы одной сети являются входами для следующей – многослойные RNN

https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks

Примеры http://karpathy.github.io/2015/05/21/rnn-effectiveness/

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

Пример сгенерированного текста моделью обученной на Шекспировских работах.

For $\bigoplus_{n=1,\ldots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the compariciol y in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ??. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example ??. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma ??. Namely, by Lemma ?? we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ *(by the formal open covering* X *and a single map* $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ *over* U *compatible with the complex*

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ *is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If* T *is surjective we may assume that* T *is connected with residue fields of* S*. Moreover there exists a closed subspace* $Z \subset X$ *of* X *where* U *in* X' *is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) f *is locally of finite type. Since* $S = \mathrm{Spec}(R)$ *and* $Y = \mathrm{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\ldots,0}$.

Lemma 0.2. *Let* X *be a locally Noetherian scheme over* S, $E = \mathcal{F}_{X/S}$. *Set* $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. *Since* $\mathcal{I}^n \subset \mathcal{I}^n$ *are nonzero over* $i_0 \leq \mathfrak{p}$ *is a subset of* $\mathcal{J}_{n,0} \circ \overline{A}_2$ *works.*

Lemma 0.3. *In Situation ??. Hence we may assume* $\mathfrak{q}' = 0$.

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □