онлайн-курс

СПЕЦИАЛЬНЫЕ АРХИТЕКТУРЫ НЕЙРОННЫХ СЕТЕЙ

Обработка текстов

NLP (natural language processing)

Был текст,
получились числа

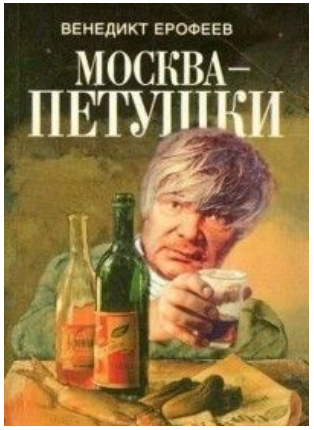
Сверхзадача при исследовании текстов

Нужно превратить текст в численный объект (например, в вектор чисел).

→ 12

→ 1812

→ 666



→ 3.62

Сначала удалить всё лишнее

Перед дальнейшей обработкой из текста нужно
удалить всё лишнее и несущественное:
знаки препинания и служебные слова (союзы, предлоги, артикли…).

Кроме того, все слова нужно привести к **начальной форме**.

Пример на всю лекцию

Рассмотрим поэтические строки:

Мело, мело по всей земле

Во все пределы.

Свеча горела на столе,

Свеча горела.

A_1=«**Мести мести весь земля**»

A_2=«**Весь предел**»

A_3=«**Свеча гореть стол**»

A_4=«**Свеча гореть**»

Пусть каждая строка — это **автономный текст**
(мы удалили знаки препинания, служ. слова и привели слова к нач. форме).

Главная задача: превратить текст в числовой вектор

То есть сделать **embedding** текста в пространство векторов.
Существует куча способов представить информацию в тексте в виде вектора. Самые простые методы представляют текст в виде длиннющего вектора чисел, причем размерность вектора равна количеству слов в рассматриваемом языке.
Итак, наша цель — получить таблицу с числами:

Текст/слова	w_1	w_2	...	w_N
A_1			...	
A_2			...	
...				

Приведём самые простые приемы превратить текст в числовой вектор.

Вес слова в тексте T(w)

Для слова w в тексте **частота T(w) равна количеству вхождений этого слова в текст**.

Для нашего примера это даст представление в виде таблицы:

	мести	весь	земля	предел	свеча	гореть	стол
A_1	2	1	1	0	0	0	0
A_2	0	1	0	1	0	0	0
A_3	0	0	0	0	1	1	1
A_4	0	0	0	0	1	1	0

Главный недостаток T(w): **его зависимость от длины текста.**

Частота слова в тексте TF(w)

TF(w)=T(w)/{длина текста}

TF	мести	весь	земля	предел	свеча	гореть	стол
A_1	2/4	1/4	1/4	0	0	0	0
A_2	0	1/2	0	1/2	0	0	0
A_3	0	0	0	0	1/3	1/3	1/3
A_4	0	0	0	0	1/2	1/2	0

Величина TF(w) позволяет **длинные и короткие тексты сделать равноправными**.

Встречаемость в текстах DF(w)

DF(w)={количество текстов, где есть слово w}/{общее число текстов}

DF	мести	весь	земля	предел	свеча	гореть	стол
A_1	1/4	2/4	1/4	1/4	2/4	2/4	1/4
A_2	1/4	2/4	1/4	1/4	2/4	2/4	1/4
A_3	1/4	2/4	1/4	1/4	2/4	2/4	1/4
A_4	1/4	2/4	1/4	1/4	2/4	2/4	1/4

Недостаток: в каждом столбце одинаковые значения.

Компромиссная величина TF-IDF(w)

TF-IDF(w)=TF(w)/DF(w)

	мести	весь	земля	предел	свеча	гореть	стол
A_1	2	1/2	1	0	0	0	0
A_2	0	1	0	2	0	0	0
A_3	0	0	0	0	2/3	2/3	4/3
A_4	0	0	0	0	1	1	0

Какой у неё смысл?

TF	мести	весь	земля	предел	свеча	гореть	стол
A_1	2/4	1/4	1/4	0	0	0	0
A_2	0	1/2	0	1/2	0	0	0
A_3	0	0	0	0	1/3	1/3	1/3
A_4	0	0	0	0	1/2	1/2	0

DF	мести	весь	земля	предел	свеча	гореть	стол
A_1	1/4	2/4	1/4	1/4	2/4	2/4	1/4
A_2	1/4	2/4	1/4	1/4	2/4	2/4	1/4
A_3	1/4	2/4	1/4	1/4	2/4	2/4	1/4
A_4	1/4	2/4	1/4	1/4	2/4	2/4	1/4

Компромиссная величина TF-IDF(w)

TF-IDF(w)=TF(w)/DF(w)

	мести	весь	земля	предел	свеча	гореть	стол
A_1	2	1/2	1	0	0	0	0
A_2	0	1	0	2	0	0	0
A_3	0	0	0	0	2/3	2/3	4/3
A_4	0	0	0	0	1	1	0

Итак, теперь тексты — это вектора чисел

Для них можно решать задачу предсказания, применяя все известные алгоритмы машинного обучения (в том числе и нейросети).

	мести	весь	земля	предел	свеча	гореть	стол	Y
A_1	2	1/2	1	0	0	0	0	1
A_2	0	1	0	2	0	0	0	0
A_3	0	0	0	0	2/3	2/3	4/3	1
A_4	0	0	0	0	1	1	0	1

Перед вами ТВ для поиска текстов со сказуемым.

Итак, теперь тексты — это вектора чисел

Как повысить точность обработки текста?

Например, можно рассматривать не только одиночные слова, а **n-граммы** (группы подряд идущих n слов). В частности, использование 2-грамм для нашего примера, это даст дополнительные столбцы:

мести мести	мести весь	весь земля	весь предел	свеча гореть	гореть стол

которые можно заполнить величинами типа TF-IDF и добавить в таблицу.

Это особенно важно, когда тексты содержат термины, состоящие из нескольких слов: «Пётр Первый», «Российская Федерация»...

Контекст слова и работа с ним

Откуда исторически возникла проблема анализа контекстов?

Это проблема синонимов!

В рассмотренных выше методах «оцифровки текста» возникает проблема синонимов. Они считаются разными словами.

Например, возникнут два разных столбца, соответствующих словам «жена» и «супруга».

тексты	жена	…	супруга
A_1			
A_2			
…			

Откуда исторически возникла проблема анализа контекстов?

По-хорошему, **синонимы нужно отождествлять**.

То есть в идеале нужно иметь таблицу:

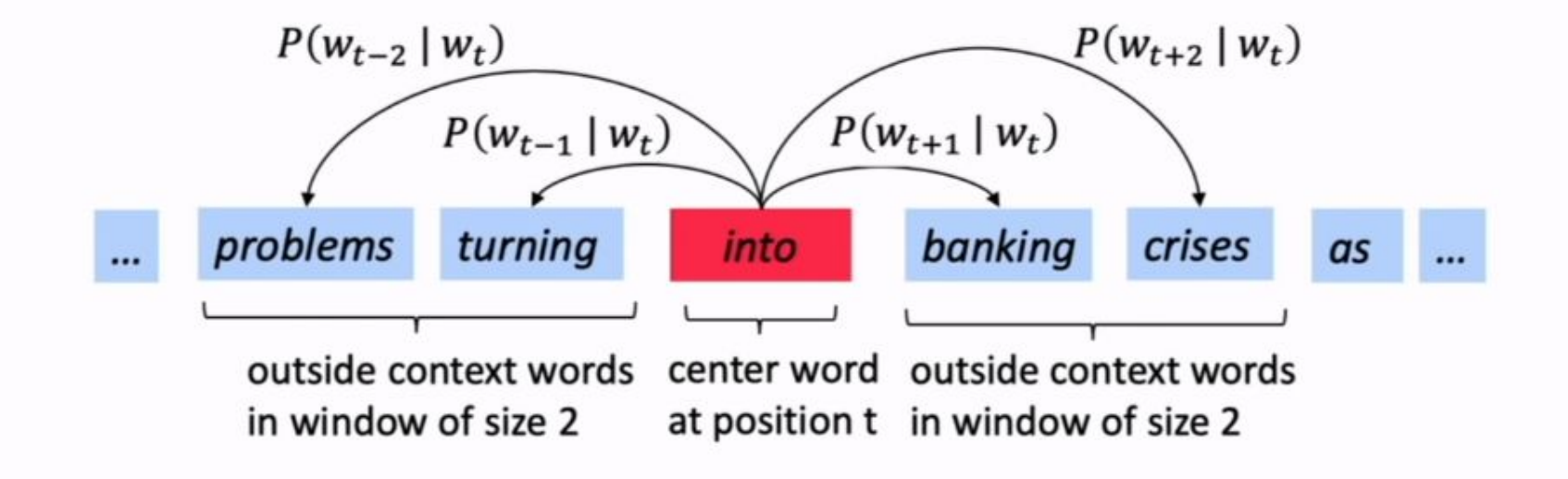
тексты	жена (супруга)	…
A_1		
A_2		
…		

А как это сделать в автоматическом режиме?

Контекст спешит на помощь

Контекст — окрестность слова радиуса R в тексте. То есть 2-контекст слова w — это окрестность радиуса 2 с центром в слове w.
Центр окрестности в некоторых алгоритмах может не включатся в контекст.

На рисунке показан 2-контекст слова «into»



Ещё пример

Например, в тексте

«мести мести весь земля»

1-контекст слова **«весь»** — это множество {«мести, земля»}

2-контекст второго слова **«мести»** — это {«мести», «весь», «земля»}
(здесь граница контекста вылезает за границу текста)

Здесь центр контекста мы не включаем в контекст.

Самое простое применение контекста: поиск синонимов

Пусть $A_1,...A_n$ — весь корпус текстов. Для слов w_i, w_j и радиуса r можно вычислить величину $c_{r,i,j}$ — число одинаковых r-контекстов слов w_i, w_j .

Смысл: **у слов-синонимов значение $c_{r,i,j}$ будет очень высоким.** Поэтому можно отождествить все слова, у которых $c_{r,i,j}$ выше определенного порога (порог определяется экспертом-лингвистом).

В этом алгоритме центр контекста мы не включаем в контекст.

Пример детекции синонимов

Даны 3 текста (любезно предоставленные автору соседями по подъезду), состоящие из слов **a, b, c**:

a b c c b a

c c c b a b

b b a a c c

где **a=«Чётко» b=«За…сь», c=«Типа»**

Поиск синонимов будем осуществлять с помощью 1-контекстов.

Пример детекции синонимов

Даны 3 текста, состоящие из однобуквенных слов **a b c**:
(a b) c c (b a)

c c c (b a b)

b (b (a a) c) c

Выпишем 1-контексты для каждого слова:

a: {b}, {b}, {b,b}, {a,b}, {a,c}

Пример детекции синонимов

Даны 3 текста, состоящие из однобуквенных слов **a b c**:
(a b c) (c b a)

c c (c b (a) b)

((b b) a) a c c

Выпишем 1-контексты для каждого слова:

a: {b}, {b}, {b,b}, {a,b}, {a,c}

b: {a,c}, {a,c}, {a,c}, {a}, {b}, {a,b}

Пример детекции синонимов

Даны 3 текста, состоящие из однобуквенных слов **a b c**:
a (b (c c) b) a

((c (c) c) b) a b

b b a (a (c c))

Выпишем 1-контексты для каждого слова:
a: {b}, {b}, {b,b}, {a,b}, {a,c}

b: {a,c}, {a,c}, {a,c}, {a}, {b}, {a,b}

c: {b,c}, {b,c}, {c}, {c,c}, {b,c}, {a,c}, {c}

Пример детекции синонимов

Для каждой пары слов считаем число контекстов, входящих в оба списка:

a: {b}, {b}, {b,b}, {a,b}, {a,c}

b: {a,c}, {a,c}, {a,c}, {a}, {b}, {a,b}

c: {b,c}, {b,c}, {c}, {c,c}, {b,c}, {a,c}, {c}

$c_{1,a,b} = 7$

Пример детекции синонимов

Для каждой пары слов считаем число контекстов, входящих в оба списка:

a: {b}, {b}, {b,b}, {a,b}, {a,c}

b: {a,c}, {a,c}, {a,c}, {a}, {b}, {a,b}

c: {b,c}, {b,c}, {c}, {c,c}, {b,c}, {a,c}, {c}

$c_{1,a,c} = 2$

Пример детекции синонимов

Для каждой пары слов считаем число контекстов, входящих в оба списка:
a: {b}, {b}, {b,b}, {a,b}, {a,c}
b: {a,c}, {a,c}, {a,c}, {a}, {b}, {a,b}
c: {b,c}, {b,c}, {c}, {c,c}, {b,c}, {a,c}, {c}
$c_{1,b,c} = 4$

Возникает таблица:

	a	b	c
a	-	7	2
b	7	-	4
c	2	4	-

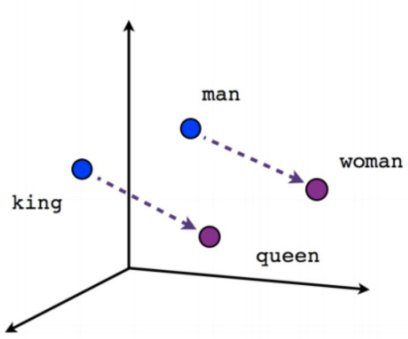
Мы видим, что слова **a,b** имеют больше всего общих контекстов. Следовательно, они — синонимы. Поэтому в дальнейших исследованиях слова **a,b** можно отождествить.

Технологии типа word2vec

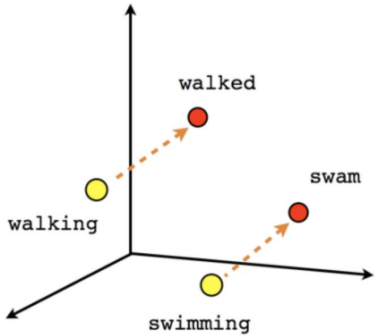
Технологии word2vec

Это алгоритмы, которые каждому слову ставят в соответствие числовой вектор со свойством:

1. семантически близкие слова должны отображаться в близкие вектора (относительно некоторой метрики)
2. должна поддерживаться некоторая арифметика над векторами:



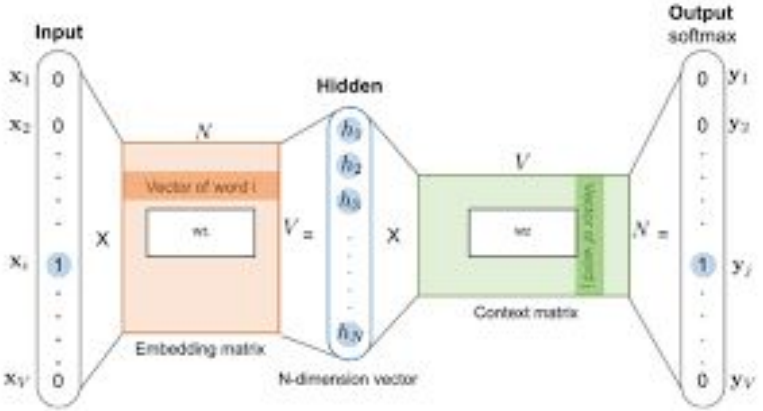
Male-Female



Verb tense

Word2vec с помощью нейросетей (НС)

Здесь будут использоваться НС «матричной» архитектуры. Фактически будет искаться разложение матрицы не «оранжевый» и «зелёный» множитель.



А уж если используются НС,то нужна тренировочная выборка (ТВ). Как её получить?

ТВ для word2vec

Нужен **один очень-очень большой текст**. Например, можно объединить **все** статьи Википедии.

Естественно, нужно сделать предобработку текста (удалить служебные слова, знаки препинания, все слова привести к начальной форме):

мести мести весь земля весь предел свеча гореть стол свеча гореть

Теперь фиксируем ширину (радиус контекста). Пусть он будет равен 1.

Начинаем перебирать все контексты и формируем пары вида: **(центральное слово контекста, слово из контекста)**

Пример построения ТВ:

мести мести весь земля весь предел свеча гореть стол свеча гореть

Начинаем бежать окном 1-контекста по тексту
(красный цвет — центральное слово контекста):

(мести мести) весь земля весь предел свеча гореть стол свеча гореть

и выписываем в таблицу пары:

мести	мести

(мести мести весь) земля весь предел свеча гореть стол свеча гореть

мести	мести
мести	мести
мести	весь

мести (мести весь земля) весь предел свеча гореть стол свеча гореть

мести	мести
мести	мести
мести	весь
весь	мести
весь	земля

и так далее…

Сразу перейдём на последнюю итерацию.

мести мести весь земля весь предел свеча гореть стол (свеча **гореть**)

мести	мести
мести	мести
мести	весь
весь	мести
весь	земля
земля	весь
земля	весь
весь	земля
весь	предел
предел	весь

предел	свеча
свеча	предел
свеча	гореть
гореть	свеча
гореть	стол
стол	гореть
стол	свеча
свеча	стол
свеча	гореть
гореть	свеча

Что дальше?

Итак, у нас есть большая таблица →

мести	мести
мести	мести
…	…
весь	предел
предел	весь

Её очень просто превратить в ТВ:

X (нецелевой признак)	Y (целевой признак)
мести	мести
мести	мести
…	…
весь	предел
предел	весь

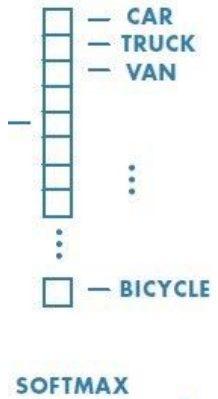
Тренировка НС будет заключаться в предсказании Y по X.

Какой будет архитектура НС?

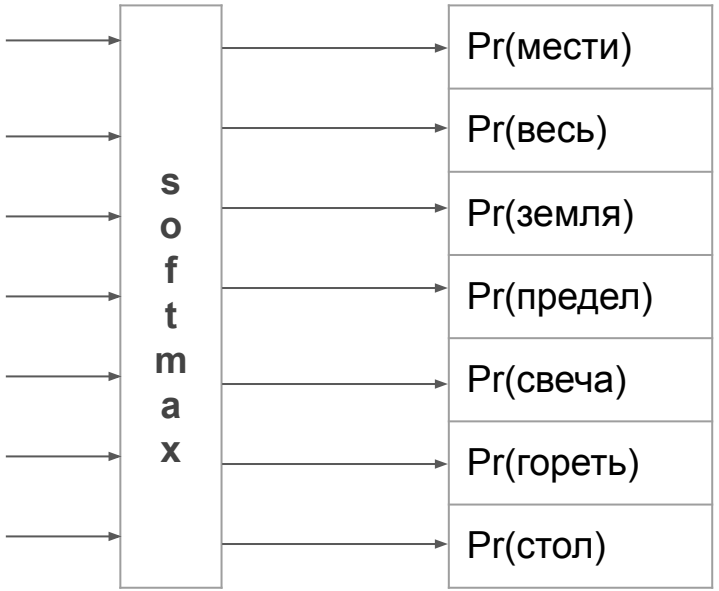
Как следует из ТВ, нейронная сеть на вход получает слова и выдаёт слова.

Проще разобраться с **выходным слоем**. По сути наша задача — **многоклассовая задача предсказания**.

Тут есть стандартная архитектура: слой softmax
и выдача вероятностей принадлежности к классам.



В нашем случае будет такой выходной слой

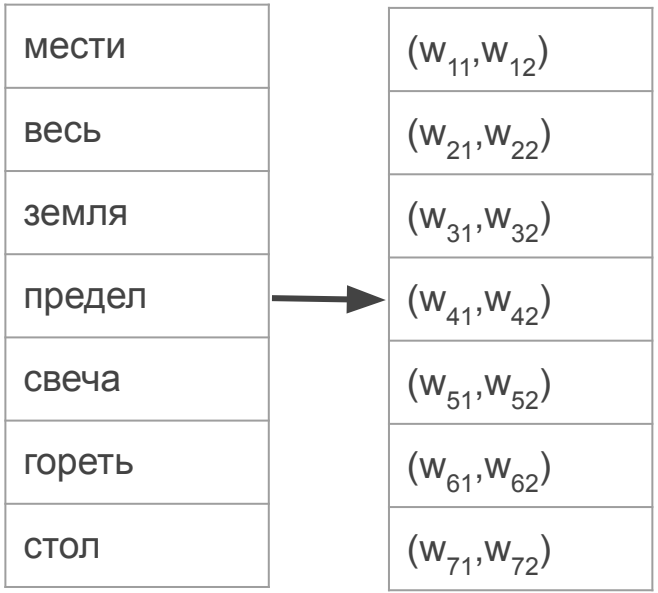


Размер выходного слоя совпадает с количеством слов в языке!

А как выглядят потроха НС?

Потроха НС

НС для word2vec матричная, то есть данные там представляются матрицами и векторами. Каждому входу (центру контекста) соответствует свой вектор, **вектор слова**. Координаты этого вектора — пока неизвестные веса НС.



Длина векторов является архитектурной характеристикой НС, и определяется заранее. Фактически — это размерность embedding-а.

В нашем примере эта размерность равна 2.

Потроха НС

Вторая матрица — это матрица контекстов.

Эти вектора будет рисовать вертикально.

Понятно, что ширина матрицы с прошлого слайда должна равняться высоте матрицы контекста (чтобы происходило умножение).

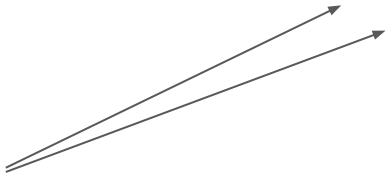
мести	весь	земля	предел	свеча	гореть	стол
u_{11}	u_{21}	u_{31}	u_{41}	u_{51}	u_{61}	u_{71}
u_{12}	u_{22}	u_{32}	u_{42}	u_{52}	u_{62}	u_{72}

Близость векторов

У нас есть вектор слова и вектор контекста (и у них одинаковая длина).
Как найти меру близости между векторами?

Косинусное сходство векторов A=(a_1,...,a_n), B=(b_1,...,b_n) :

$$\cos\phi = \frac{a_1 b_1 + \ldots + a_n b_n}{\sqrt{a_1^2 + \ldots + a_n^2}\sqrt{b_1^2 + \ldots + b_n^2}}$$



Большое сходство

Малое сходство

Близость векторов

Как следует из формулы, близость векторов пропорциональна сумме (скалярное произведение)

$$a_1 b_1 + \ldots + a_n b_n$$

На этом и будет основана тренировка НС.

У нас в ТВ есть пара (мести, весь). Этим словам соответствую вектора

«**мести**»=(w_{11},w_{12}) — вектор слова

«**весь**»=(u_{21},u_{22}) — вектор контекста.

Вычисляем их скалярное произведение: $w_{11}u_{21}+w_{12}u_{22}$

Но это ещё не всё…

Выписываем функцию потерь

У нас в ТВ есть пара (мести, весь). Этим словам соответствуют вектора

«**мести**»=(w_{11},w_{12}), «**весь**»=(u_{21},u_{22}).

Вычисляем их скалярное произведение: $w_{11}u_{21}+w_{12}u_{22}$

А также вычисляем скалярное произведение вектора-слова «мести» со **всеми** остальными векторами-контекстами:

$w_{11}u_{11}+w_{12}u_{12}$ $w_{11}u_{21}+w_{12}u_{22}$ $w_{11}u_{31}+w_{12}u_{32}$ $w_{11}u_{41}+w_{12}u_{42}$

$w_{11}u_{51}+w_{12}u_{52}$ $w_{11}u_{61}+w_{12}u_{62}$ $w_{11}u_{71}+w_{12}u_{72}$

И после этого составляем гигантский софт-макс… (да-да, это больно)

Выписываем функцию потерь

Получается гигантская дробь!

$$\frac{\exp(w_{11}u_{21}+w_{12}u_{22})}{\exp(w_{11}u_{11}+w_{12}u_{12})+\exp(w_{11}u_{21}+w_{12}u_{22})+\exp(w_{11}u_{31}+w_{12}u_{32})+\exp(w_{11}u_{41}+w_{12}u_{42})+\exp(w_{11}u_{51}+w_{12}u_{52})+\exp(w_{11}u_{61}+w_{12}u_{62})+\exp(w_{11}u_{71}+w_{12}u_{72})}$$

В функцию потерь НС L(w) будет записано слагаемое:

...-ln(дробь со слайда)-...

Подобное слагаемое нужно выписать для каждой строки из ТВ.
А потом нужно минимизировать это выражение.

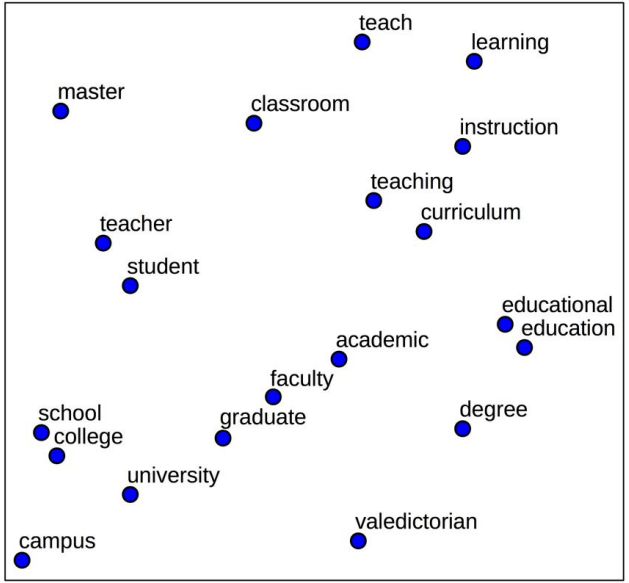
Что в итоге?

Что даст минимизация функции потерь L(w)?

Она даст оптимальные значения координат векторов-слов **w** и векторов-контекстов **u**.

Нам важнее вектора **w** — они дают представление слов в виде векторов.

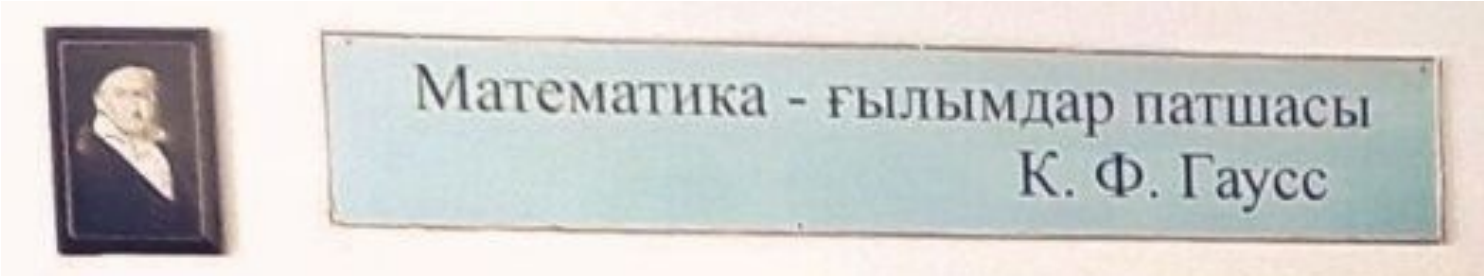
мести	(w_{11}, w_{12})
весь	(w_{21}, w_{22})
земля	(w_{31}, w_{32})
предел	(w_{41}, w_{42})
свеча	(w_{51}, w_{52})
гореть	(w_{61}, w_{62})
стол	(w_{71}, w_{72})



Более тонкая настройка word2vec

- Можно варьировать размерность векторов-слов и векторов-контекстов (в нашем примере она равнялась 2).
- Можно брать более широкий радиус контекста (в нашем примере он был равен 1).

Но смысл тренировки НС не поменяется, поскольку



Negative sampling

Общий вид функции потерь для НС word2vec

1. **Это сумма логарифмов** с обратными знаками (это следует из общей теории).
2. **Количество логарифмов** равно объёму ТВ.
3. Внутри каждого логарифма стоит дробь. В знаменателе дроби — сумма экспонент. **Количество слагаемых в знаменателе совпадает с количеством слов в словаре.**

Например, если в русском языке порядка 100.000 слов, то в **каждом** знаменателе **каждого** логарифма в функции потерь будет 100.000 слагаемых.

Количество логарифмов равно количеству слов **во всех статьях** русской Википедии (мы условились тренировать НС на всей Википедии)!

И вот эту функцию потерь L(w) нужно минимизировать!

Общий вид функции потерь для HC word2vec

Таким образом, функция потерь L(w) в задаче word2vec просто огромная. В ней миллионы слагаемых, а внутри каждого слагаемого еще порядка 100 тысяч слагаемых в знаменателе:

L(w)=-ln(...18+...)-ln(...18+...)-...-ln(...18+...)

Даже для современных* компьютеров вычисления с участием такой функции — дело медленное.

Нужно упрощать вычисления с функцией L(w). Возможно, даже в ущерб качеству обучения (**спойлер**: качество остаётся удовлетворительным).

Как упрощать вычисления?

*курс записывается в 2022 г., идёт второй год пандемии. Привет, потомки! На Марс уже полетели?

1-ая примочка: стохастический ГС

Этот метод должен быть вам знаком, он изучается в любом общем курсе по нейронным сетям.

Суть: на каждой итерации ГС случайно выбирается **ровно одно слагаемое** из функции потерь, и градиент считается только по нему.

Это означает, что на каждой итерации ГС рассматривается ровно один логарифм из суммы.

Но внутри логарифма стоит дробь, знаменатель которой содержит порядка 100 тыс. слагаемых.

А для решения этой проблемы см. 2-ую примочку: **negative sampling**.

2-ая примочка: negative sampling

Рассмотрим «под-логарифмическое» выражение. В нашем примере оно имеет знаменатель с 7 слагаемыми, поскольку в нашем «языке» всего 7 слов: **мести весь земля предел свеча гореть стол**

$$\frac{\exp(w_{11}u_{21}+w_{12}u_{22})}{\exp(w_{11}u_{11}+w_{12}u_{12})+\exp(w_{11}u_{21}+w_{12}u_{22})+\exp(w_{11}u_{31}+w_{12}u_{32})+\exp(w_{11}u_{41}+w_{12}u_{42})+\exp(w_{11}u_{51}+w_{12}u_{52})+\exp(w_{11}u_{61}+w_{12}u_{62})+\exp(w_{11}u_{71}+w_{12}u_{72})}$$

В «великом-и-могучем» русском языке слагаемых будет вообще порядка 100 тыс.

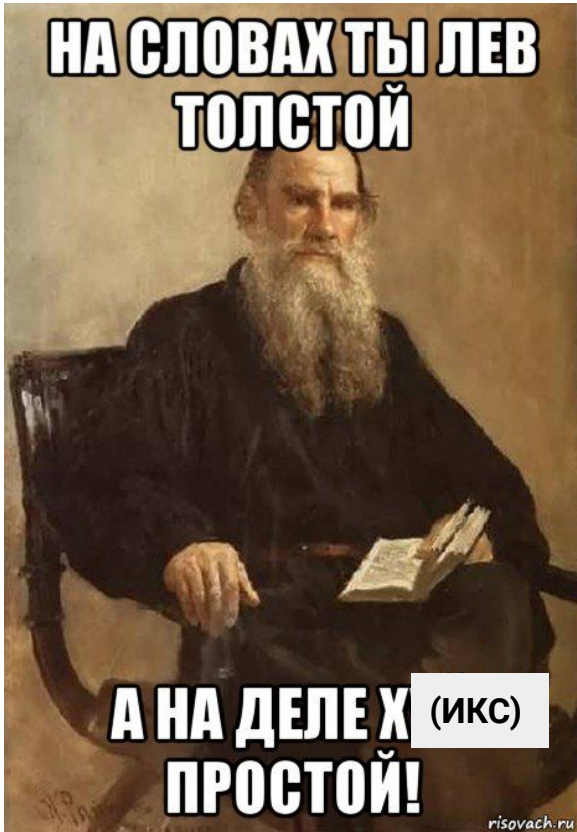
Идея: на каждой итерации ГС будет оставлять в знаменателе лишь несколько случайно выбранных слагаемых!

2-ая примочка: negative sampling

То есть на каждой итерации ГС от «великого-и-могучего» остаётся лишь несколько слов, по весам которых и вычисляются градиенты.

$$\frac{\exp(w_{11}u_{21}+w_{12}u_{22})}{\exp(w_{11}u_{11}+w_{12}u_{12})+0+\exp(w_{11}u_{31}+w_{12}u_{32})+\exp(w_{11}u_{41}+w_{12}u_{42})+0+\exp(w_{11}u_{61}+w_{12}u_{62})+0}$$

Фактически на каждой итерации мы работаем со случайной и небольшой подвыборкой нашей тренировочной выборки [X,Y].



Что дальше?

Маемо що маемо

Слова языка — это **теперь вектора в пространстве** достаточно небольшой (по сравнению с размером словаря) размерности.

Текст теперь — это последовательность векторов $T=(v_1,v_2,...,v_k)$.

Это позволяет для распознавания текстов **применять НС, которым на вход подаются многомерные массивы**.

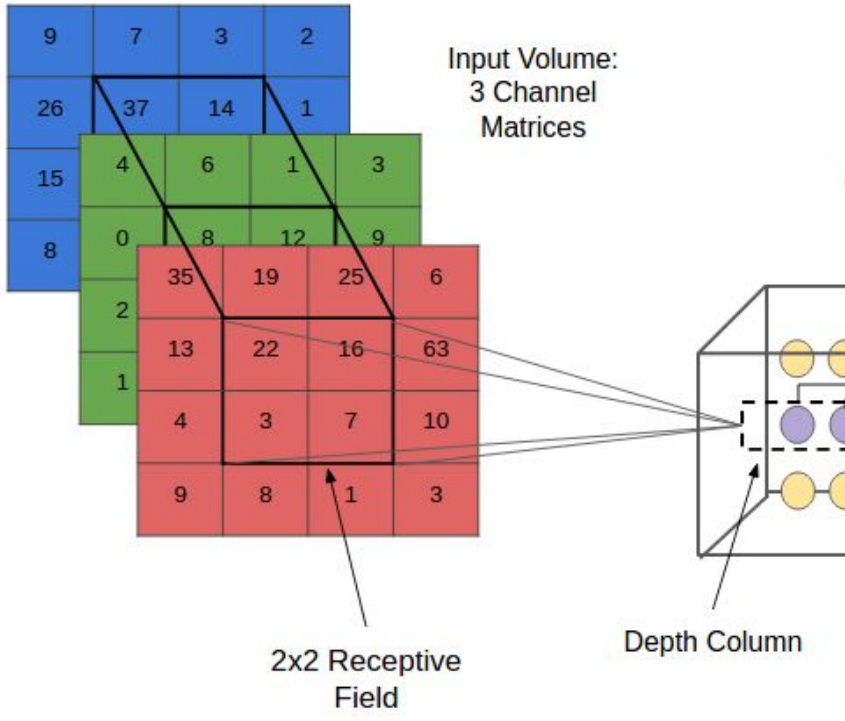
В частности, тексты могут распознаваться **свёрточными НС**.

Да-да, между картинками и текстами много общего, и там и там значение элемента определяется его контекстом (окрестностью).

CNC для картинок

Фильтр скользит по «поверхности» и сворачивает окрестность каждого пиксела с помощью фильтра.

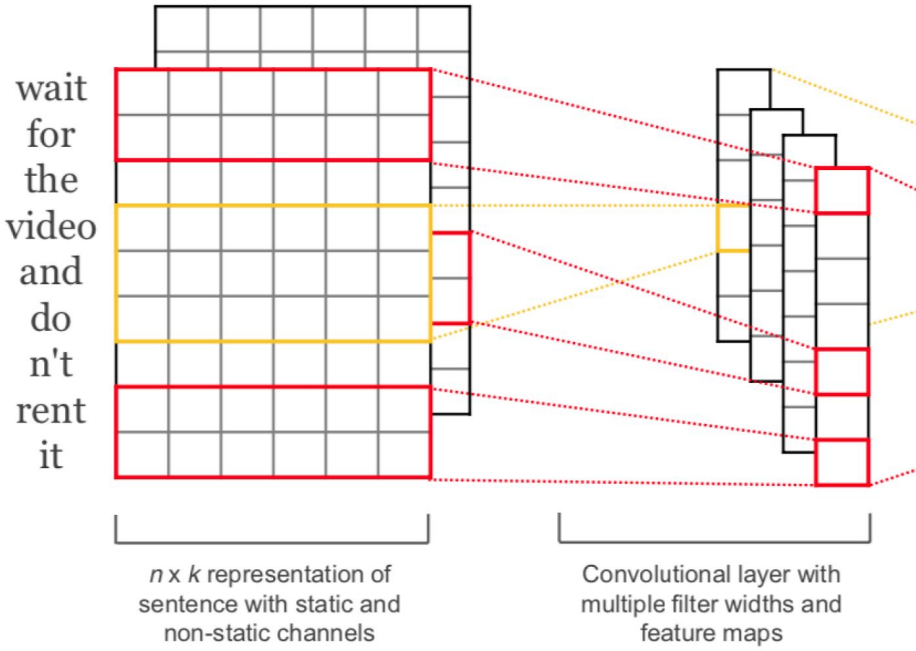
Каждый пиксел — это вектор длины 3.



CNC для текстов

Фильтр скользит по тексту и сворачивает окрестность каждого слова с помощью фильтра.

В этом примере каждое слово представлено вектором длины 6.



Выводы

- Мы рассмотрели основные методы подготовки текстов на естественных языках для решения задач машинного обучения.
- Поговорили о получении численных представлений слов языка. Это достигается с помощью технологий типа word2vec.
- Рассмотрели процедуру negative sampling, позволяющую ускорить тренировку НС в технологии word2vec.