

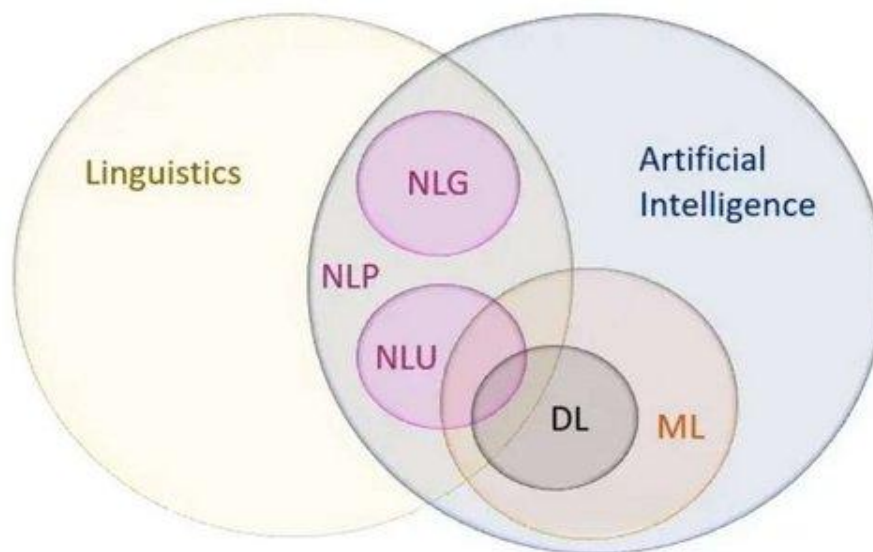
Машинное обучение

Лекция 5.

Обработка естественного языка

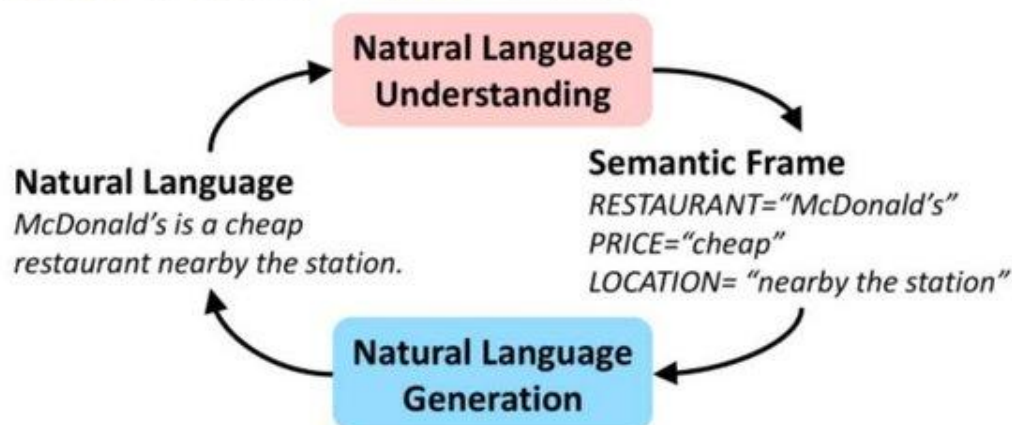
Обработка естественного языка

Обработка естественного языка (Natural Language Processing, NLP) – общее направление **искусственного интеллекта** и **лингвистики**, изучающее методы **анализа** и **синтеза естественного языка**



Обработка естественного языка

- ▶ Внутри NLP условно выделяются два направления: понимание языка (NLU) и генерация языка (NLG)
- ▶ текст → NLU → смысл → NLG → текст



- ▶ Граница между NLU и NLG нечёткая
 - ▶ Пример: задачи категоризации и анализа тональности
- ▶ Смежные области — распознавание (ASR) и генерация (TTS) речи

Задачи NLP

Можно формулировать задачи с технической и продуктовой точек зрения

Технические задачи:

- ★ ▶ Классификация
- ▶ POS-теггинг
- ★ ▶ Извлечение сущностей
- ▶ Разрешение кореференции
- ▶ Лемматизация
- ▶ Сегментация
- ▶ Разметка семантических ролей
- ▶ Векторизация
- ▶ Машинный перевод
- ★ ▶ Суммаризация
- ★ ▶ Тематическое моделирование

Продуктовые задачи:

- ▶ Анализ тональности
- ▶ Машинный перевод
- ▶ Генерация подписей к изображениям
- ★ ▶ Генерация сниппетов для новостей
- ▶ Ведение диалога
- ▶ Исправление опечаток
- ★ ▶ Поисковое и рекомендательное ранжирование
- ★ ▶ Анализ трендов
- ▶ ...

Подготовка текстовых данных

Этапы подготовки (предобработки) текстовых данных:

- приведение к одному регистру
- замена «нетекстовой» информации
- удаление пунктуации
- токенизация
- удаление *стоп-слов*, редких и частых слов
- стемминг
- лемматизация
- векторизация

Токенизация

- ▶ Токенизацию можно производить по словам и/или предложениям
- ▶ Используются как подходы, основанные на правилах, так и модели ML
- ▶ В nltk есть много различных токенизаторов, например RegexpTokenizer и sent_tokenize

```
1 from nltk.tokenize import sent_tokenize
2 text = 'Это предложение. И это тоже предложение и т.д.'
3 print(sent_tokenize(text, language='russian'))
4 #['Это предложение.', 'И это тоже предложение и т.д.']
```

- ▶ Часто слова грубо выделяют разделением по пробелам с помощью метода split:

```
1 text = 'Набор слов, составляющий какое-то предложение.'
2 print(text.split(' '))
3 #['Набор', 'слов,', 'составляющий', 'какое-то', 'предложение.']
```

Удаление пунктуации

- ▶ В ряде задач пунктуация полезна, но часто от неё лучше избавляться
- ▶ Удалять пунктуацию можно напрямую, полезен модуль `string`:

```
1 import string
2 text = 'Набор из 7 слов, составляющий какое-то предложение.'
3 for c in string.punctuation:
4     text = text.replace(c, '')
5 print(text)
6 # Набор из 7 слов составляющий какое-то предложение
```

- ▶ Если нужно точно удалить все символы, кроме алфавитных, лучше использовать регулярные выражения:

```
1 import re
2 text = 'Набор из 7 слов, составляющий какое-то предложение.'
3 print(re.sub('[^a-яА-ЯёЁй ]+', '', text))
4 # Набор из 7 слов составляющий какое-то предложение
```

Регистр и пунктуация

- ▶ Есть задачи, в которых пунктуация и регистр несут важную информацию
- ▶ Это важно для определения границ предложений, для решения задачи выделения именованных сущностей

в комнату вошёл **лев** и, потянувшись, достал из кармана сигару

лев обитает в саванне, в арктике не обитает

- ▶ В задаче анализа тональности существенное значение имеют **смайлы** (текстовые или символы в Unicode)

✗ Одежда у вас в магазине очень своеобразная:/

✓ Одежда у вас в магазине очень своеобразная:)

Лемматизация и стемминг

- ▶ Слова в тексте могут иметь различную формы, часто такая информация скорее мешает, чем помогает анализу
- ▶ Для нормализации применяется один из подходов:
 - ▶ лемматизация (py morphology2, py mystem3)
 - ▶ стемминг (реализации в nltk)
- ▶ Лемматизация приводит слова к нормальной форме
- ▶ Стемминг приводит слова к псевдооснове (убирает окончания и формообразующие суффиксы)

```
1 import morphology2
2 text = 'я хотел бы поговорить с вами'.split(' ')
3 lemmatizer = morphology2.MorphAnalyzer()
4 print([lemmatizer.parse(t)[0].normal_form for t in text])
5 # ['я', 'хотеть', 'бы', 'поговорить', 'с', 'вы']
```

Фильтрация словаря

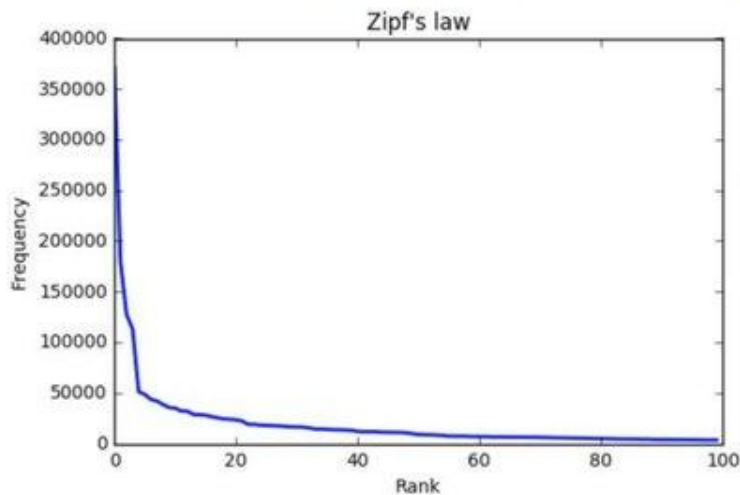
- ▶ Часто из текстов нужно удалять лишние слова
- ▶ Обычно это стоп-слова, очень редкие и очень частые слова
- ▶ К стоп-словам относятся союзы, предлоги, модальные глаголы, местоимения, вводные слова
- ▶ Большой набор стоп-слов есть в nltk:

```
1 from nltk.corpus import stopwords
2 sw = set(stopwords.words('russian'))
3
4 for w in ['я', 'хотеть', 'бы', 'поговорить', 'с', 'вы']:
5     if w not in sw:
6         print(w, end=' ')
7     # хотеть поговорить
```

- ▶ При необходимости можно урезать или расширять этот список

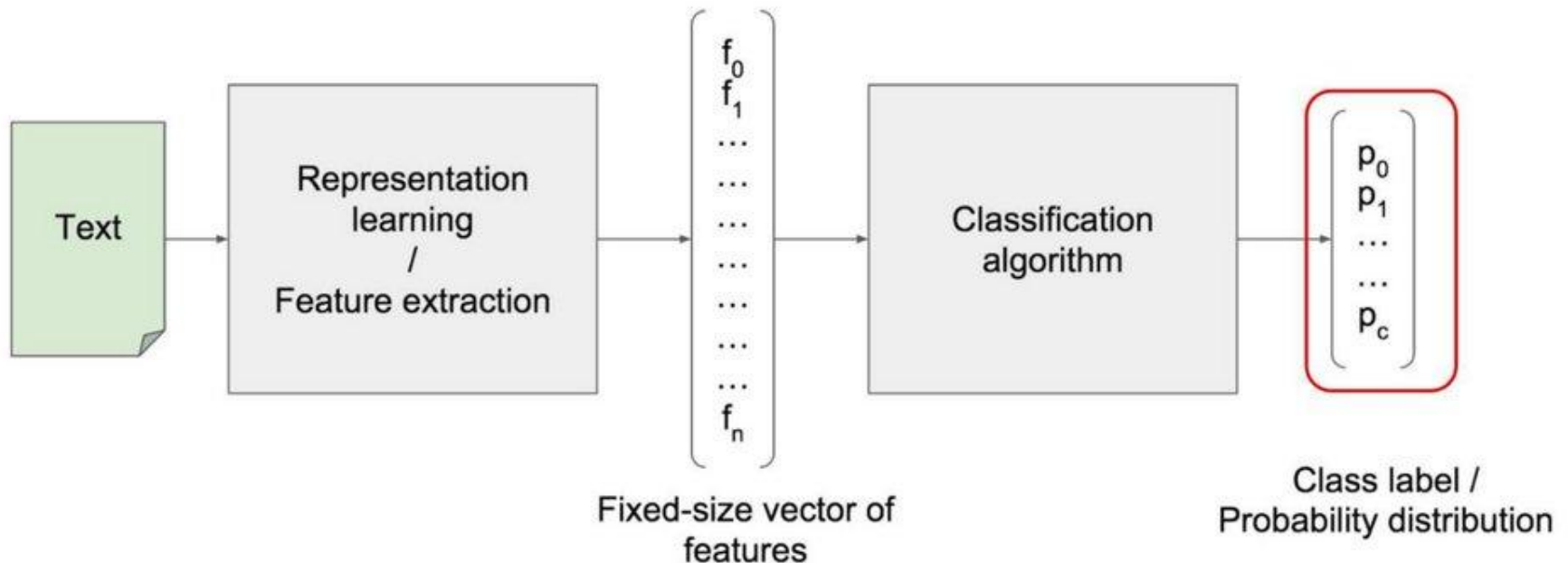
Фильтрация словаря

- ▶ Слишком частые или редкие слова тоже могут оказаться вредными
- ▶ Такие слова могут и мешать обучению модели, и увеличивать затраты ресурсов памяти и времени счёта
- ▶ При обработке коллекции стоит проверить выполнение закона Ципфа:

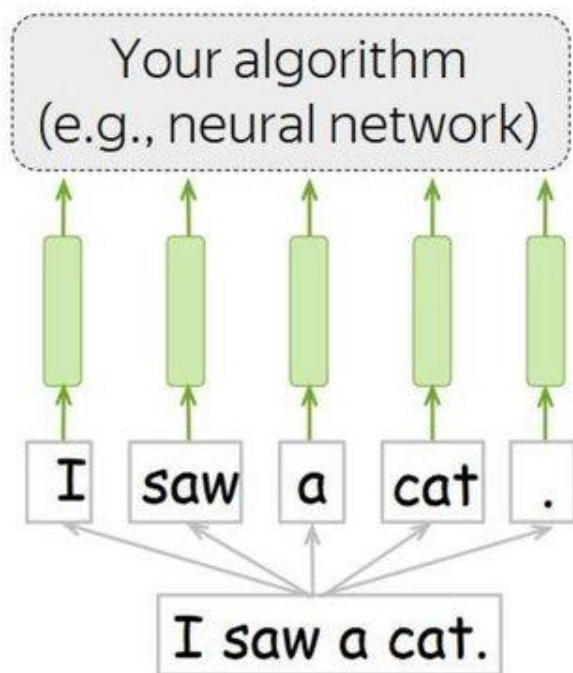


- ▶ Имеет смысл обрезать это распределение по порогам с обеих сторон

Задача классификации текстовых данных



Представление текстовых данных



Any algorithm for solving a task

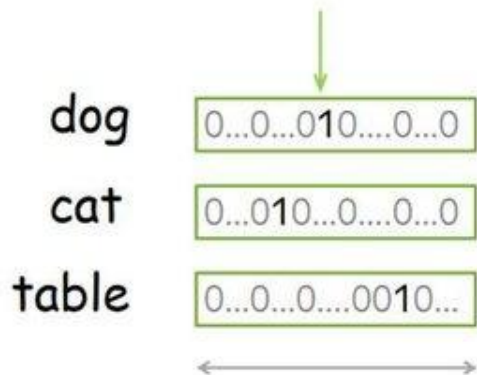
Word representation - vector
(input for your model/algorithm)

Sequence of tokens

Text (your input)

One-Hot Vector и мешок слов (BOW)

One is 1, the rest are 0



Embedding dimension =
vocabulary size

Problems:

- Vector size is too large
- Vectors know nothing about meaning

e.g., **cat** is as close to
dog as it is to **table**!

Гипотеза дистрибутивной семантики

«Смысл слова определяется множеством его контекстов»

- Words that occur in the same contexts tend to have similar meanings [Harris, 1954].
- You shall know a word by the company it keeps [Firth, 1957].

Синтагматическая близость слов:

сочетаемость слов в одном контексте

(здание–строитель, кран–вода, функция–точка)



Парадигматическая близость слов:

взаимозаменяемость слов в одном контексте

(здание–дом, кран–смеситель, функция–отображение)



Count-based подходы

Задача:

- вложение (embedding, вектор) слова должно содержать информацию о контексте

Count-based решение:

- рассчитать вложение на основе глобальных статистик документа

TF-IDF

Context:

- document d (from a collection D)

Matrix element:

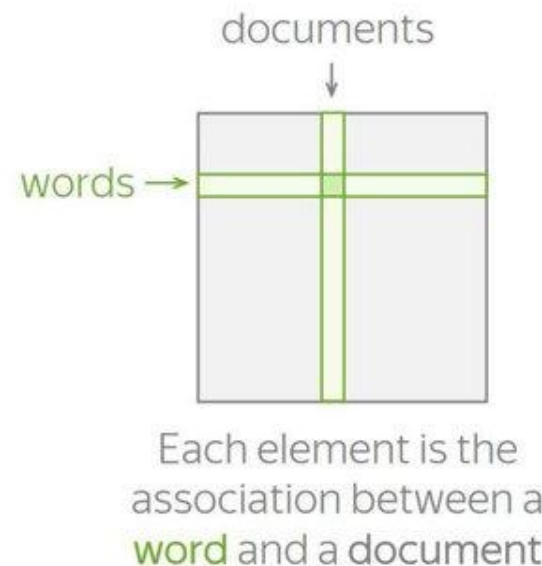
- $\text{tf-idf}(w, d, D) = \text{tf}(w, d) \cdot \text{idf}(w, D)$

$$N(w, d)$$

term frequency

$$\log \frac{|D|}{|\{d \in D: w \in d\}|}$$

inverse document frequency



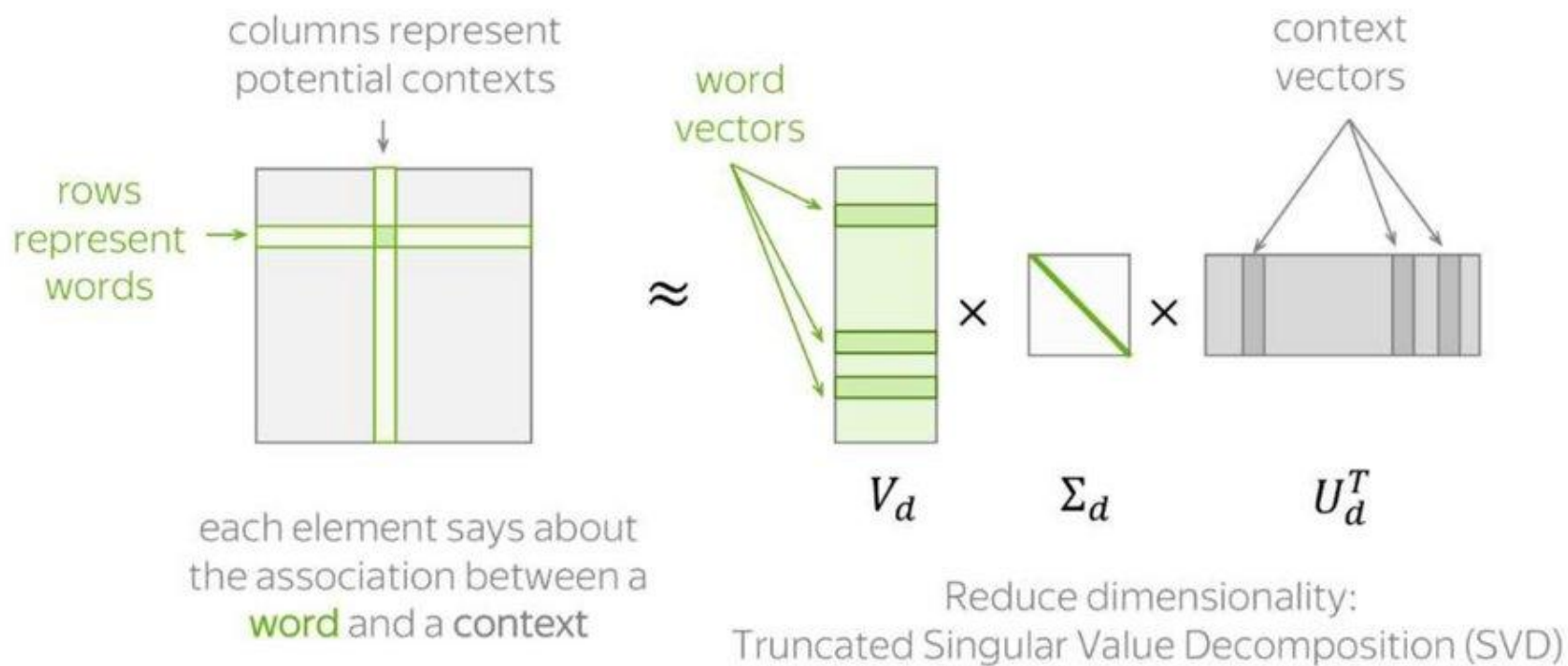
TF-IDF (2)

- TF (Term Frequency), частотность термина, отражает, насколько часто слово появляется в документе. Он может быть рассчитан путем деления количества вхождений слова в документ на общее количество слов в документе. Большое значение TF обычно указывает на то, что слово является важным для данного документа.
- IDF (Inverse Document Frequency) рассчитывается как обратная вероятность встречи слова в коллекции документов. Это снижает значимость слов, которые встречаются в большом количестве документов, поскольку они могут быть общими и не имеющими высокой специфичности для конкретного документа. IDF может быть рассчитан путем деления общего количества документов в коллекции на количество документов, в которых встречается данное слово, а затем применения логарифма для сглаживания значений.

TF-IDF (3)

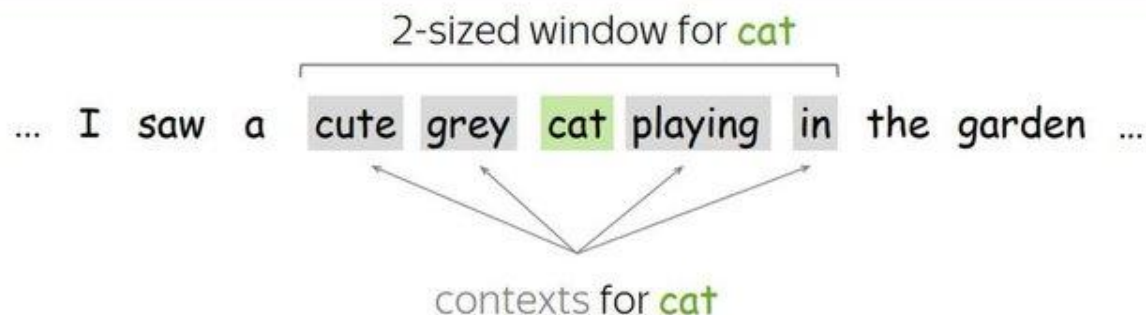
- TF-IDF рассчитывается путем умножения TF на IDF. Большое значение TF-IDF указывает на то, что слово имеет высокую важность для данного документа или коллекции документов.
- TF-IDF используется в различных задачах обработки текстов, таких как поиск информации, кластеризация документов, категоризация текста и разработка рекомендательных систем. Эта метрика помогает выделить ключевые слова и определить их релевантность для определенного контекста. Она также помогает отфильтровать общие слова (стоп-слова), которые могут быть менее значимыми для анализа текста.

SVD-разложение



Поточечная взаимная информация

Поточечная взаимная информация (Pointwise Mutual Information, PMI)



Context:

- surrounding words in a L-sized window

Matrix element:

- $PPMI(w, c) = \max(0, PMI(w, c))$,
where

$$PMI(w, c) = \log \frac{P(w, c)}{P(w)P(c)} = \log \frac{N(w, c)|(w, c)|}{N(w)N(c)}$$

How much one variable tells about another

Prediction-based подходы

Задача:

- вложение (embedding, вектор) слова должно содержать информацию о контексте

Prediction-based решение:

- обновлять параметры вложений во время обучения модели

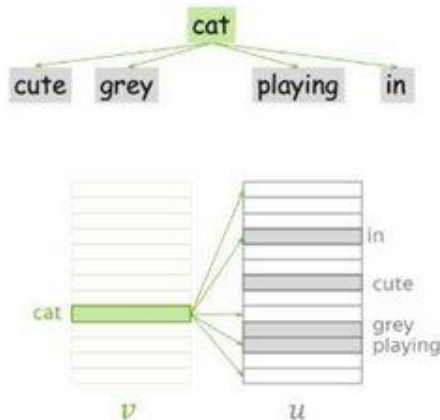
Модели векторных представлений на основе prediction-based подходов:

- word2vec
- GloVe
- fasttext
- ...

Word2vec: Skip-Gram and CBOW

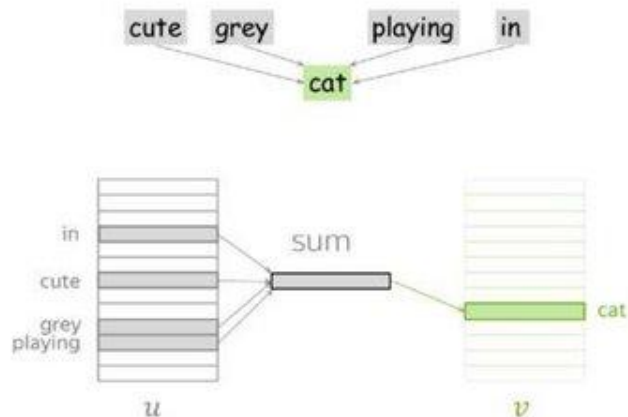
... I saw a cute grey cat playing in the garden ...

Skip-Gram: from central predict context
(one at a time)



(this is what we did so far)

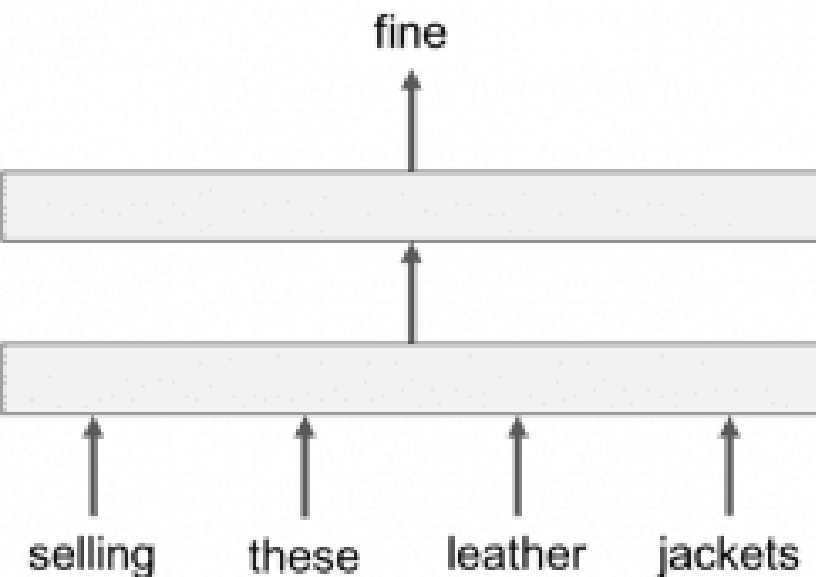
CBOW: from sum of context predict central



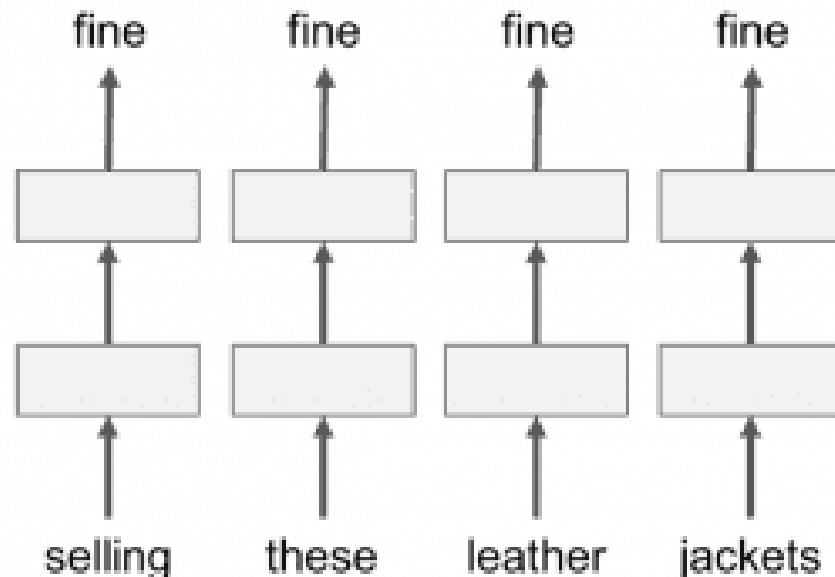
(Continuous Bag of Words)

Разновидности Word2vec

CBOW



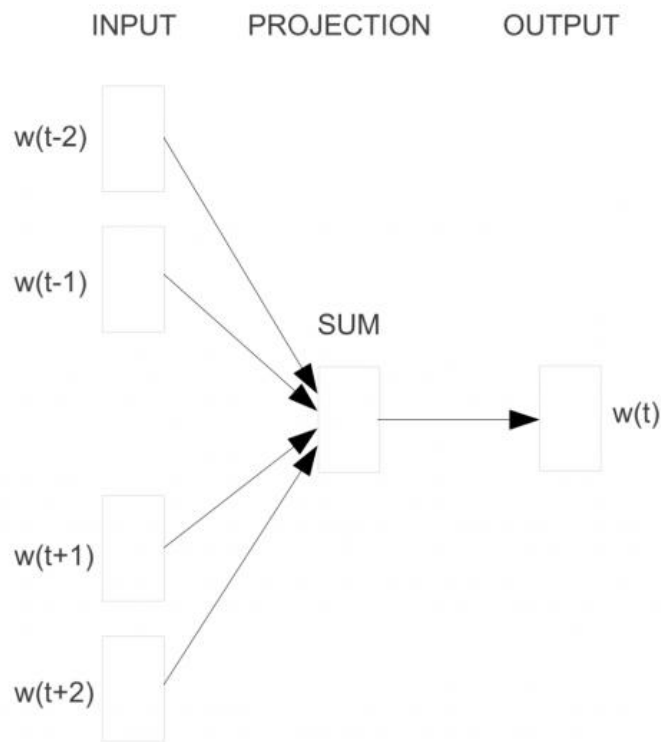
SKIPGRAM



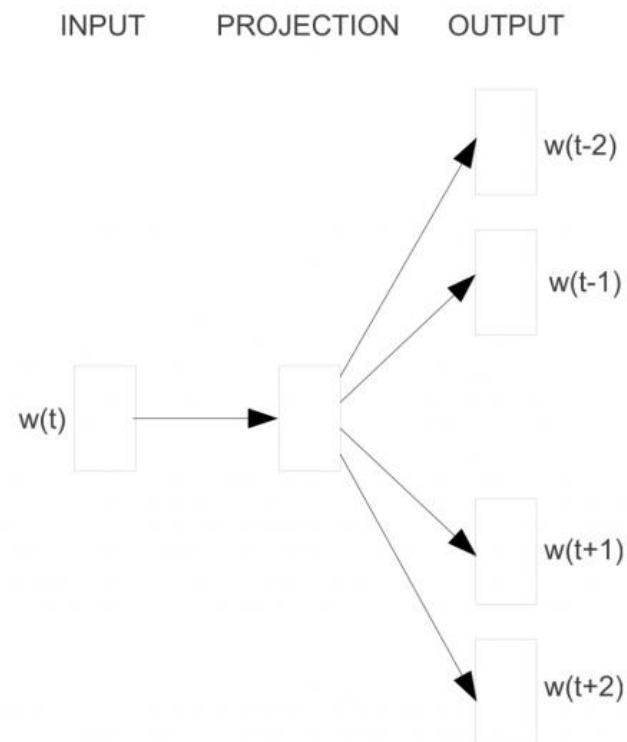
I am selling these fine leather jackets

- **Continuous Bag of Words (CBOW):** берется много предложений в корпусе. Каждый раз, когда алгоритм видит слово, берется соседнее слово. Далее на вход нейросети подаются контекстные слова и предсказывается слово в центре этого контекста. В случае тысяч таких контекстных слов и центрального слова, получаем один экземпляр датасета для нашей нейросети. Нейросеть тренируется и, наконец, выход закодированного скрытого слоя представляет вложение (embedding) для определенного слова. То же происходит, если нейросеть тренируется на большом числе предложений, и словам в схожем контексте приписываются схожие вектора.
- **Skip-Gram:** рассматривается контекстное окно, содержащее k последовательных слов. Далее пропускается одно слово и обучается нейронная сеть, содержащая все слова, кроме пропущенного, которое алгоритм пытается предсказать. Следовательно, если 2 слова периодически делят схожий контекст в корпусе, эти слова будут иметь близкие векторы.
- Их недостаток – использование окна ограниченного размера
- **GloVe** стремится решить эту проблему захватом значения одного word embedding со структурой всего обозримого корпуса. Чтобы сделать это, модель ищет глобальные совпадения числа слов и использует достаточно статистики, минимизирует среднеквадратичное отклонение, выдает пространство вектора слова с разумной субструктурой. Такая схема в достаточной степени позволяет отождествлять схожесть слова с векторным расстоянием.

- «Непрерывный мешок слов (continuous bag of words)» и «словосочетание с пропуском (skip-gram)».
- Они различаются по своей цели: первая предсказывает центральное слово на основе окружающих его слов, а вторая выполняет обратное действие.



CBOW



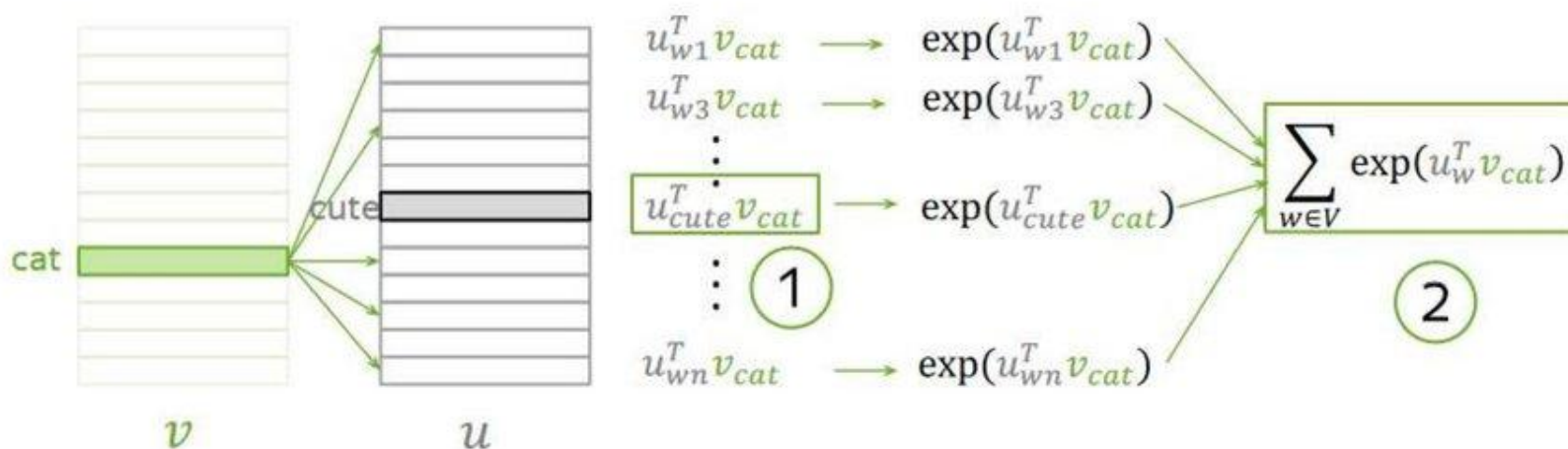
Skip-gram

Word2vec: обучение

1. Take dot product of v_{cat} with all u

2. exp

3. sum all



Word2vec: обучение

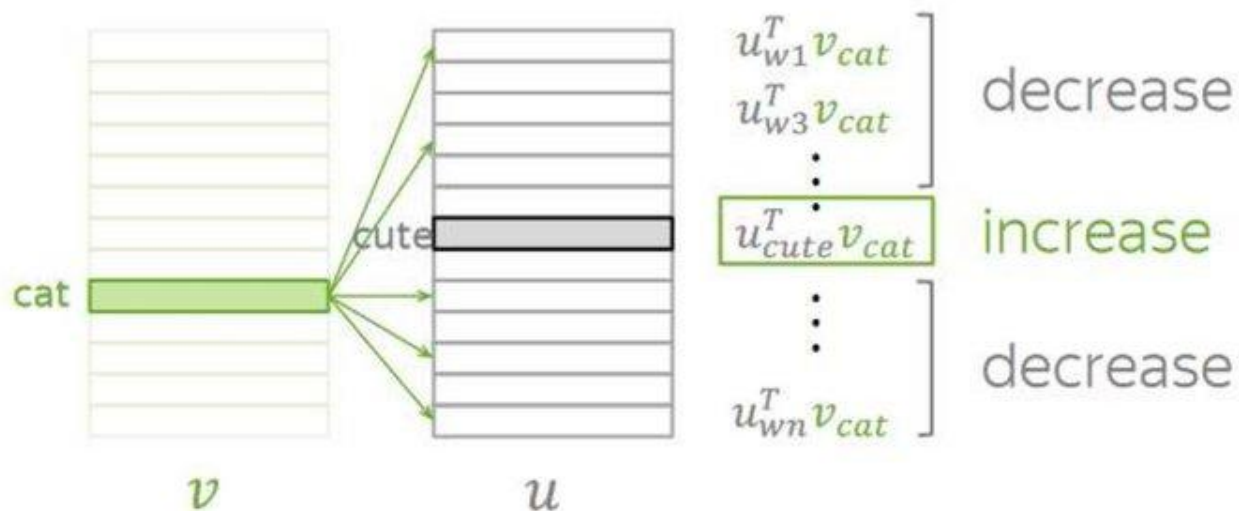
4. get loss (for this one step)

$$J_{t,j}(\theta) = \underbrace{-u_{cute}^T v_{cat}}_{\textcircled{1}} + \log \underbrace{\sum_{w \in V} \exp(u_w^T v_{cat})}_{\textcircled{2}}$$

5. evaluate the gradient, make an update

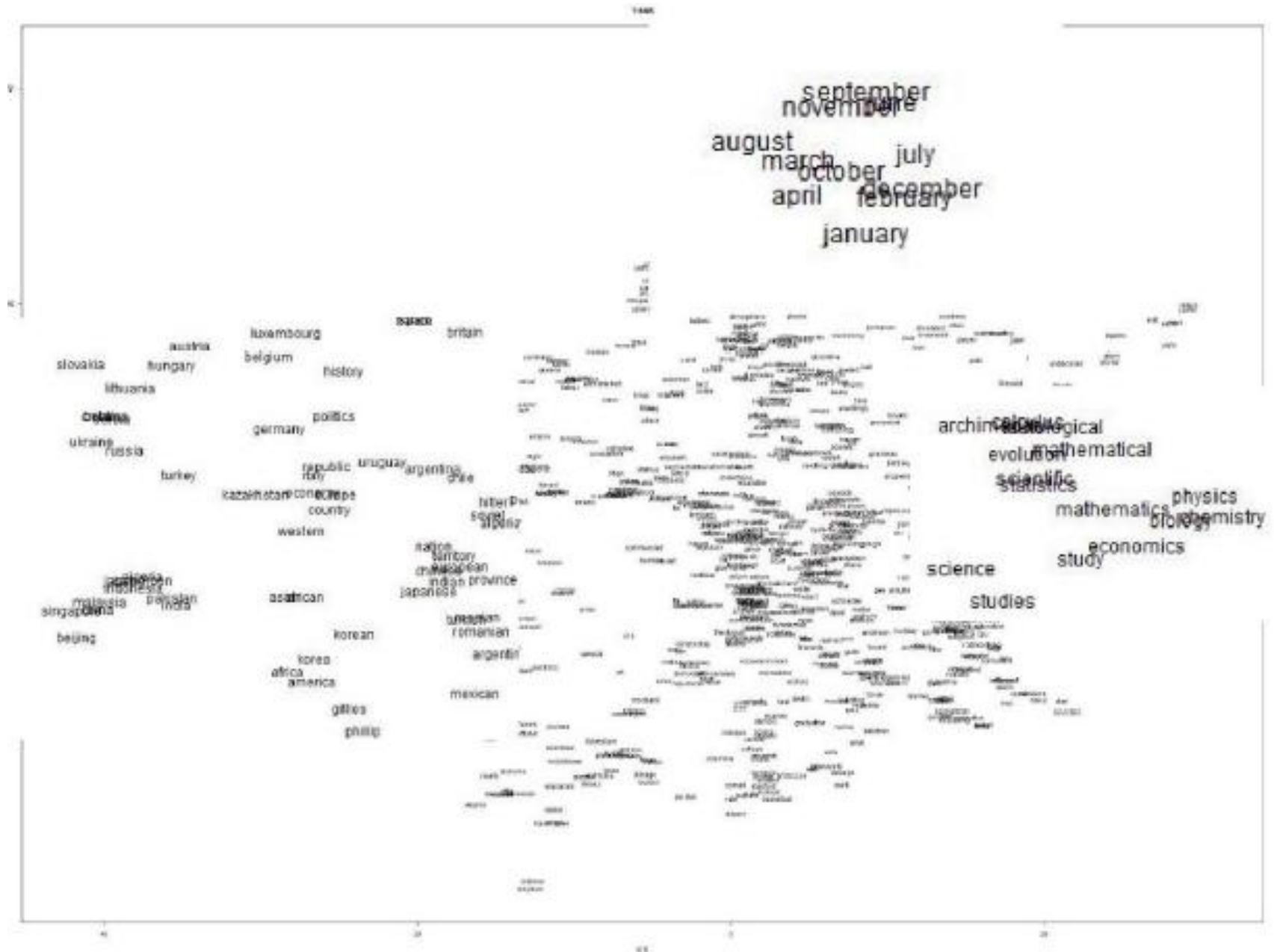
$$v_{cat} := v_{cat} - \alpha \frac{\partial J_{t,j}(\theta)}{\partial v_{cat}}$$
$$u_w := u_w - \alpha \frac{\partial J_{t,j}(\theta)}{\partial u_w} \quad \forall w \in V$$

Word2vec: обучение

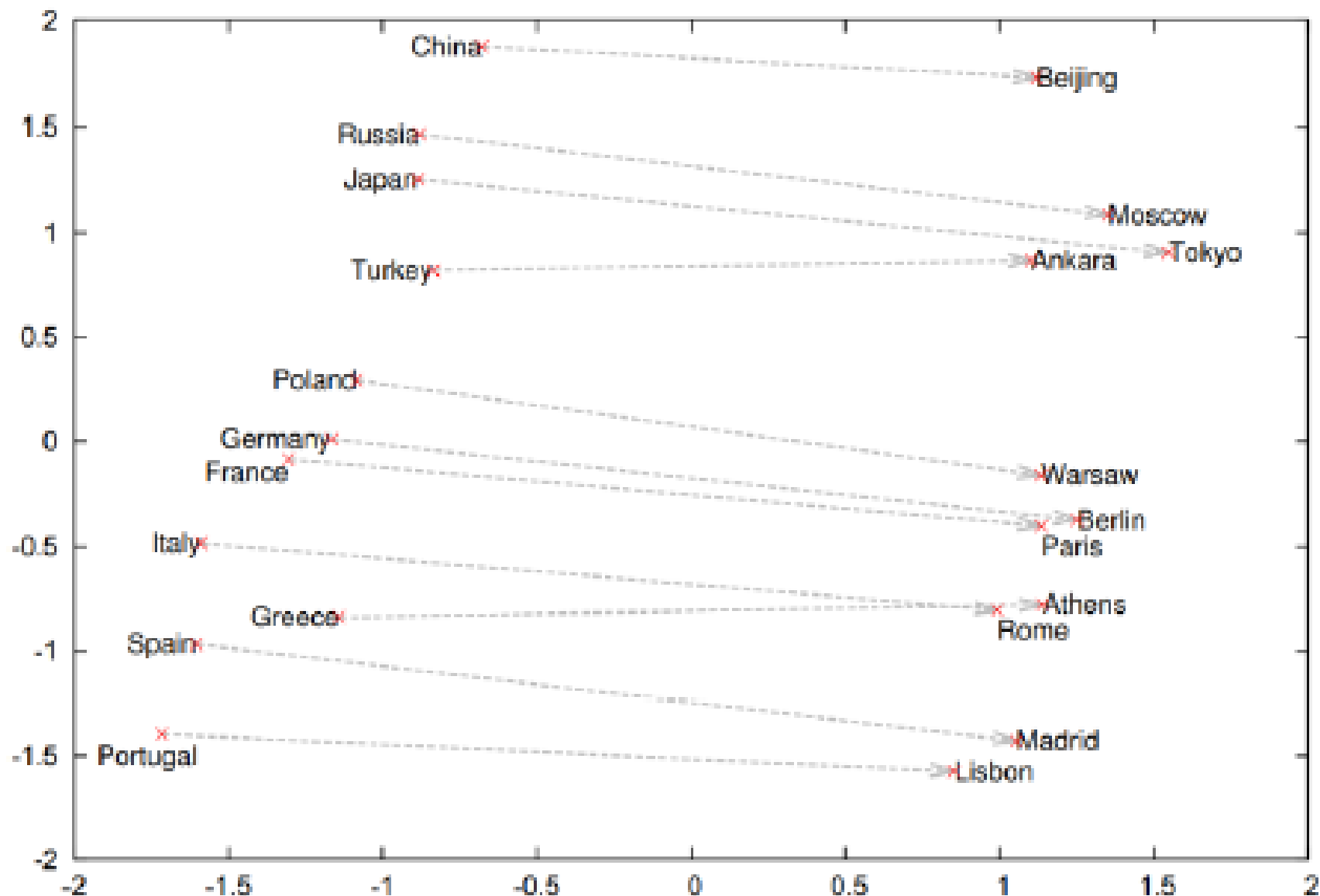


В 2013 году, Google

Близость семантически близких слов

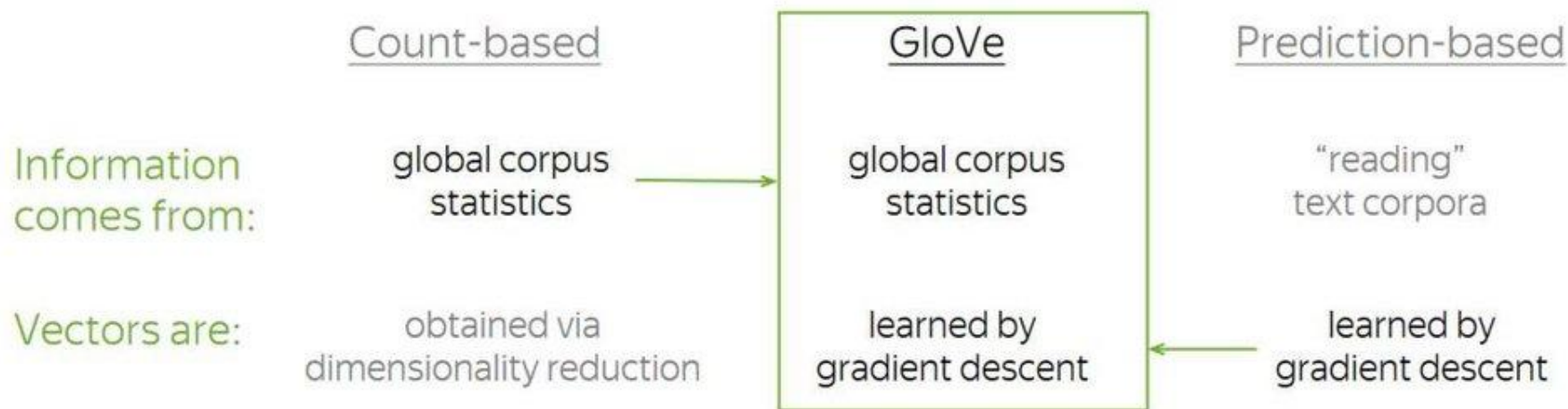


Схожее смещение у пар со схожим семантическим отношением



- Достоинства:
 - Слова, близкие по значению, будут находиться рядом в N-мерном пространстве результирующего вектора.
 - Одни и те же слова в разных языках при несложных трансформациях дают схожие вектора.
- Недостатки:
 - На вход такой модели подаётся тот самый *one-hot representation*, неэффективный при изменяющемся размере словаря.
 - Относительно трудоемкий процесс кодирования слов

GloVe: Global Vectors for Word Representation



FastText

Идея: векторное представление слова w определяется как сумма векторов всех его буквенных n -грамм $G(w)$:

$$u_w = \sum_{g \in G(w)} u_g$$

В Skip-gram вместо векторов слов u_w обучаются векторы u_g

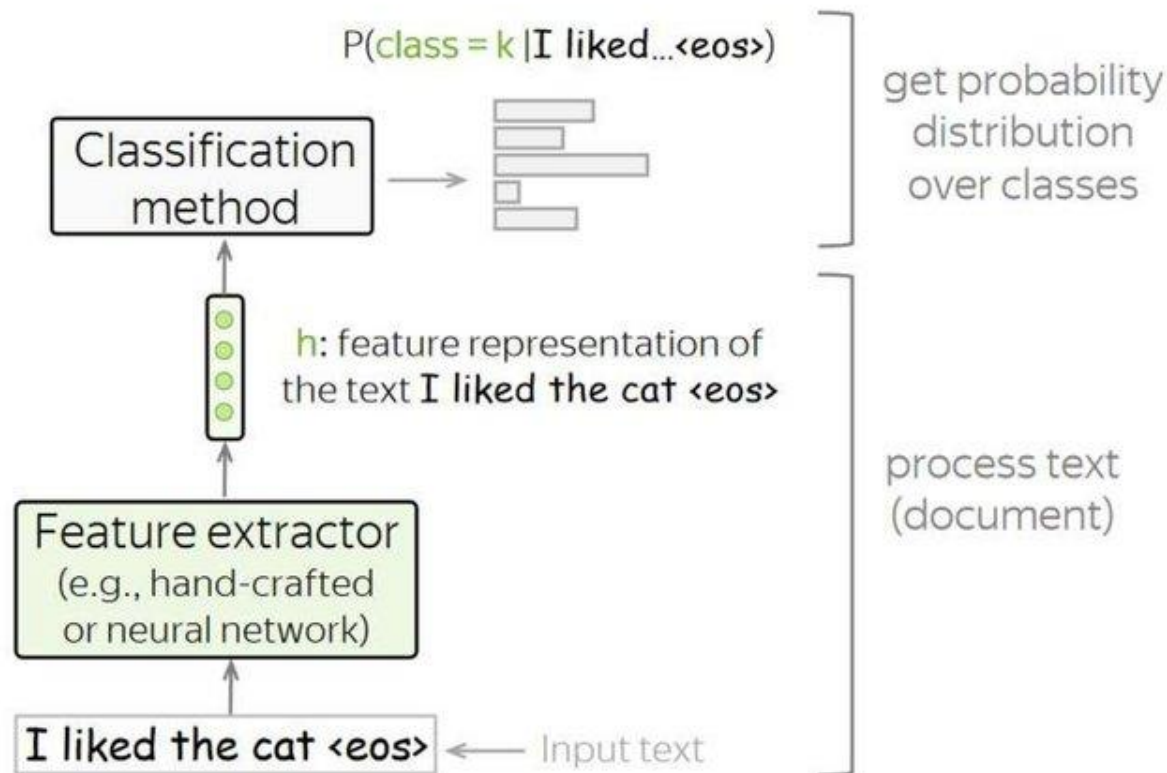
Пример: $G(\text{дармолуб}) = \{ \langle \text{да, арм, рмо, мол, олю, люб, юб} \rangle \}$

Преимущества:

- Это решает проблемы новых слов и слов с опечатками
- Подходит для обработки текстов социальных медиа
- Словарь 2- и 3-грамм обычно меньше словаря W
- Существует много предобученных моделей

Bojanowski et al. Enriching word vectors with subword information. 2016.

Задача классификации текстовых данных



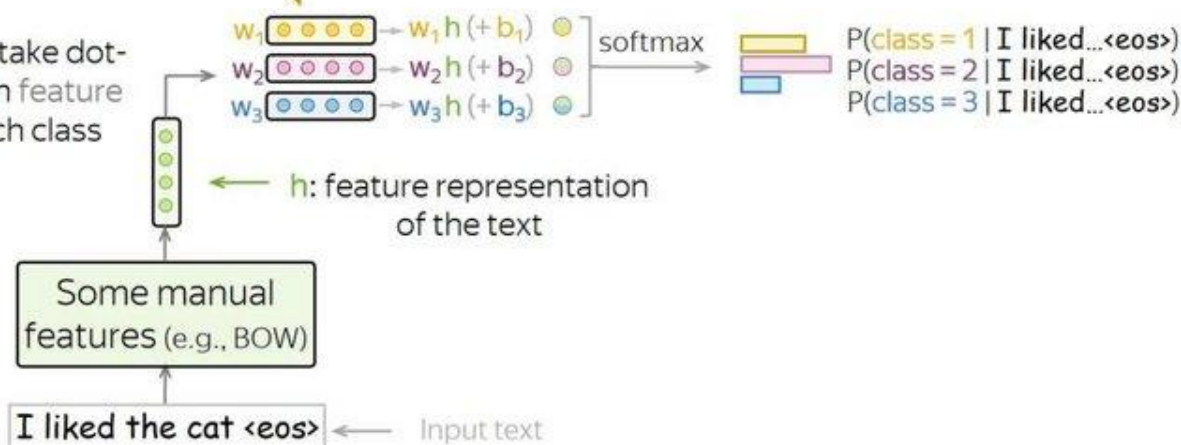
LogReg для классификации текстовых данных

- $h = (f_1, f_2, \dots, f_n)$ – features of an input text
- $w^{(k)} = (w_1^{(k)}, w_2^{(k)}, \dots, w_n^{(k)})$ – feature weights for class k

$$w^{(k)} h = w_1^{(k)} \cdot f_1 + \dots + w_n^{(k)} \cdot f_n, \quad k = 1, \dots, K$$

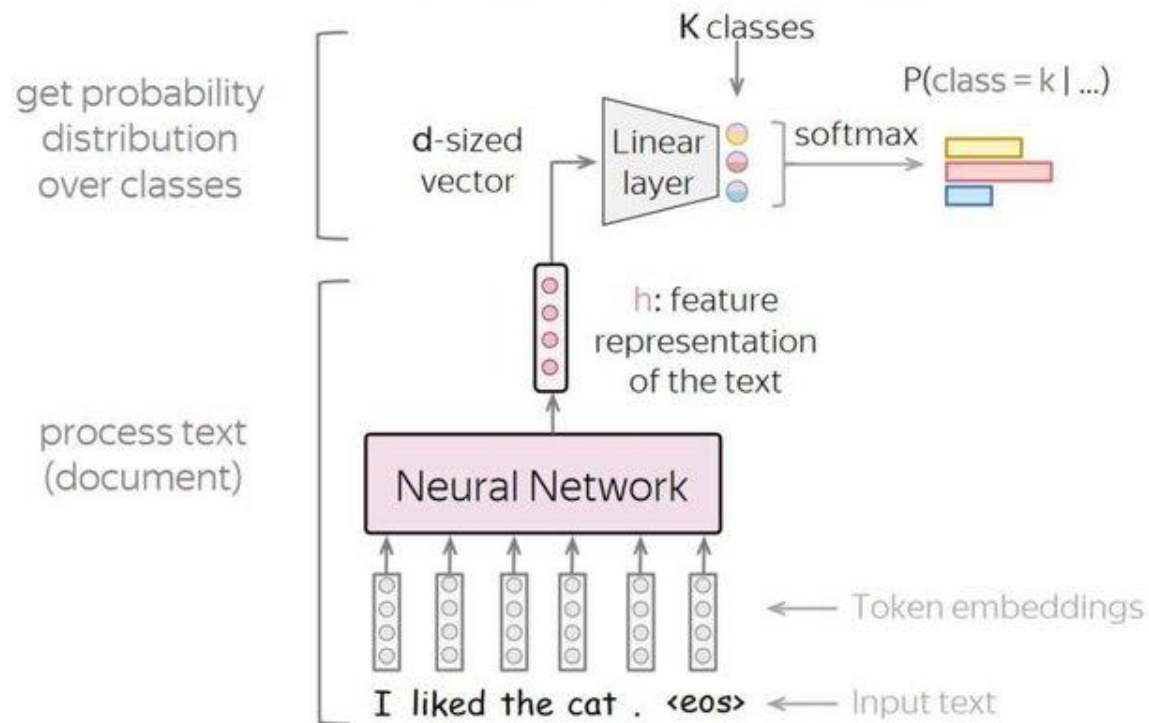
$$P(\text{class} = k | h) = \frac{\exp(w^{(k)} h)}{\sum_{i=1}^K \exp(w^{(i)} h)}$$

Weigh features: take dot-product of h with feature weights for each class



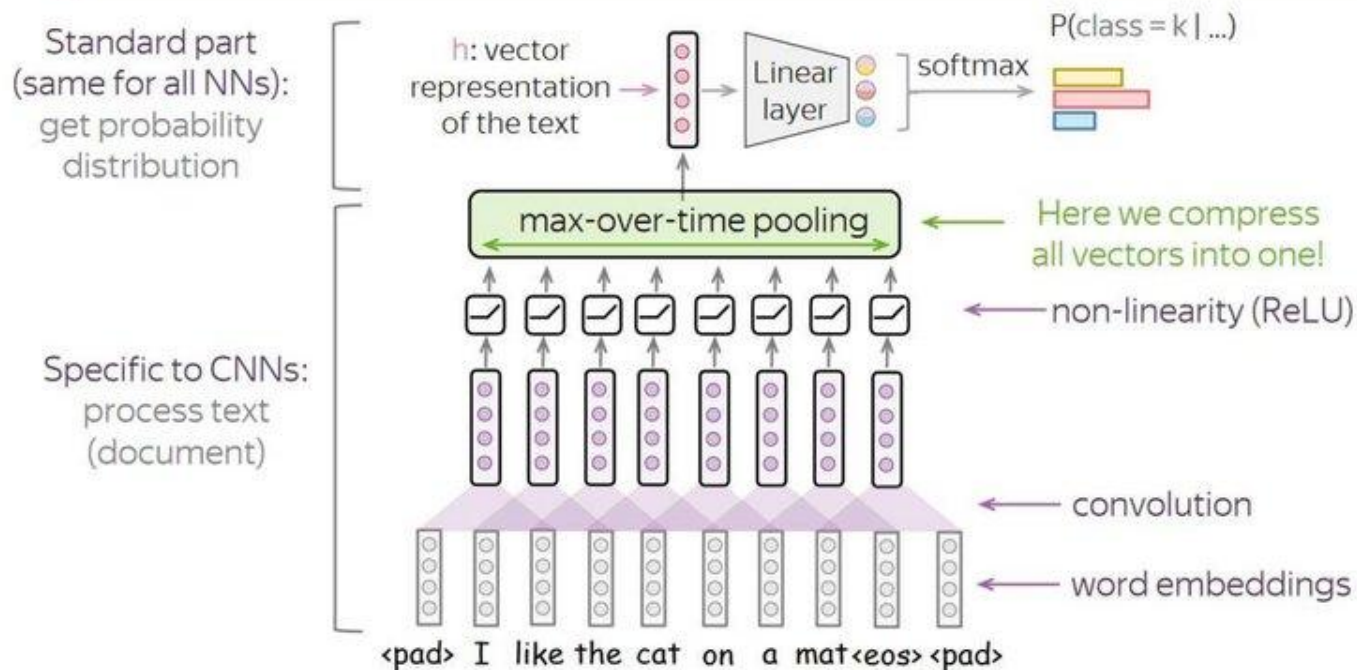
NN для классификации текстовых данных

- Classification with Neural Networks



CNN для классификации текстовых данных

We need a model that can produce a **fixed-sized** vector for inputs of **different** lengths.



Инструменты для NLP

Специфичные:

- [NLTK](#) (предобработка, ...)
- [Gensim](#) (векторизация, тематическое моделирование, ...)
- [SpaCy](#) (основные задачи NLP)
- [Natasha](#) (для русского языка: предобработка, векторизация, NER)
- [DeepPavlov](#) (для русского языка: векторизация, NER и остальные основные задачи)

Знакомые:

- [Sklearn](#) (предобработка, векторизация, тематическое моделирование, ...)
- [Hugging Face](#) (предобученные модели, наборы данных)
- [PyTorch](#)
- [re](#)

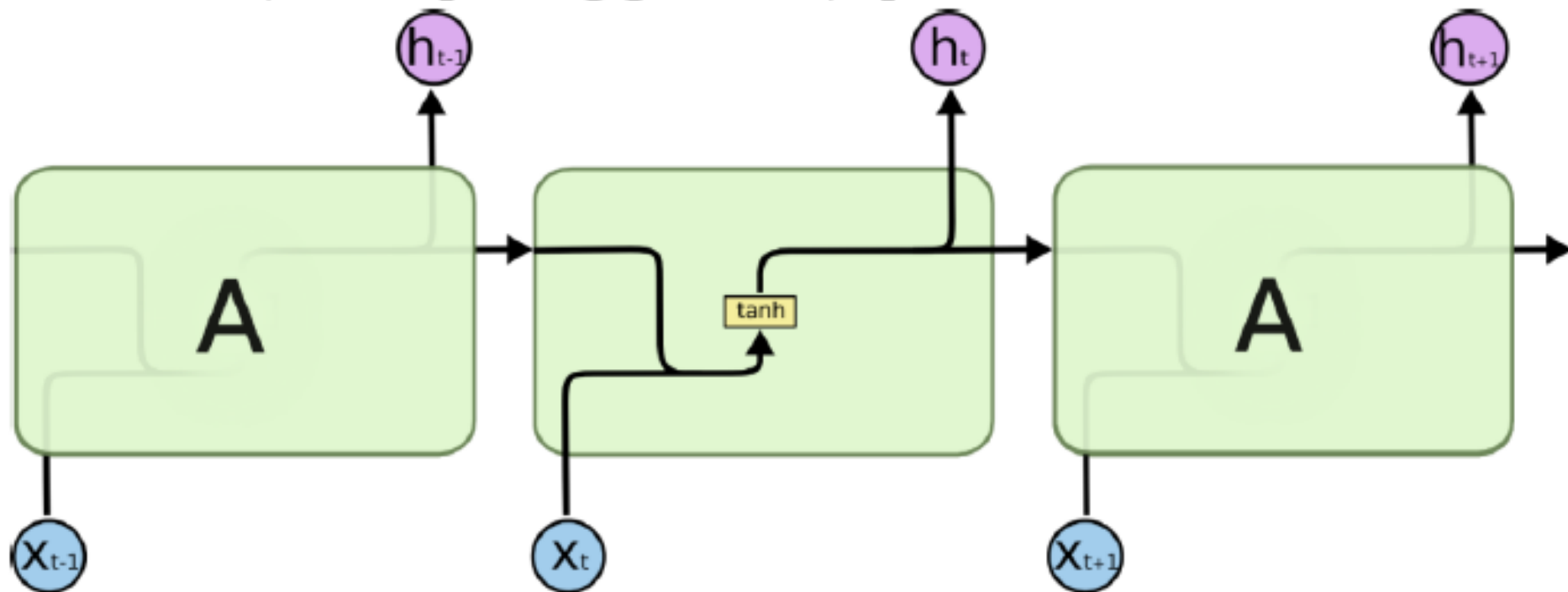
Модели, используемые для обработки ЕЯ нейронными сетями.

Здесь текст рассматривается как
последовательность токенов

- Рекуррентные
 - Простые RNN
 - LSTM или GRU
- Трансформеры или сети с вниманием

Большинство современных нейросетевых моделей преобразования последовательностей имеют структуру энкодер-декодер [5, 2, 35]. Так, в энкодере символьное представление входной последовательности $(x_1, \dots, x_n)$ переводится в последовательность непрерывных представлений $(z = (z_1, \dots, z_n))$. При заданном z , декодер затем генерирует выходную последовательность символов $(y_1, \dots, y_m)$, по одному элементу за один раз. На каждом шаге модель является авторегрессионной [10], потребляя ранее сгенерированные символы в качестве дополнительных входных данных при генерации следующего символа.

Обработка последовательности слов (токенов) простой RNN



- Главная проблема RNN — проблема исчезновения градиента, когда информация теряется с течением времени. Интуитивно кажется, что это не является серьезной проблемой, так как это только веса, а не состояния нейронов. Но с течением времени веса становятся местами, где хранится информация из прошлого. Если вес примет значение 0 или 100000, предыдущее состояние не будет слишком информативно. Как следствие, RNN будут испытывать сложности в запоминании слов, стоящих дальше в последовательности, а предсказания будут делаться на основе крайних слов, что создает проблемы.
- Этому недостатка отчасти лишены рекуррентные модели с памятью LSTM (Long Short Term Memory) и GRU (Gated Recurrent Unit)

Ячейка LSTM

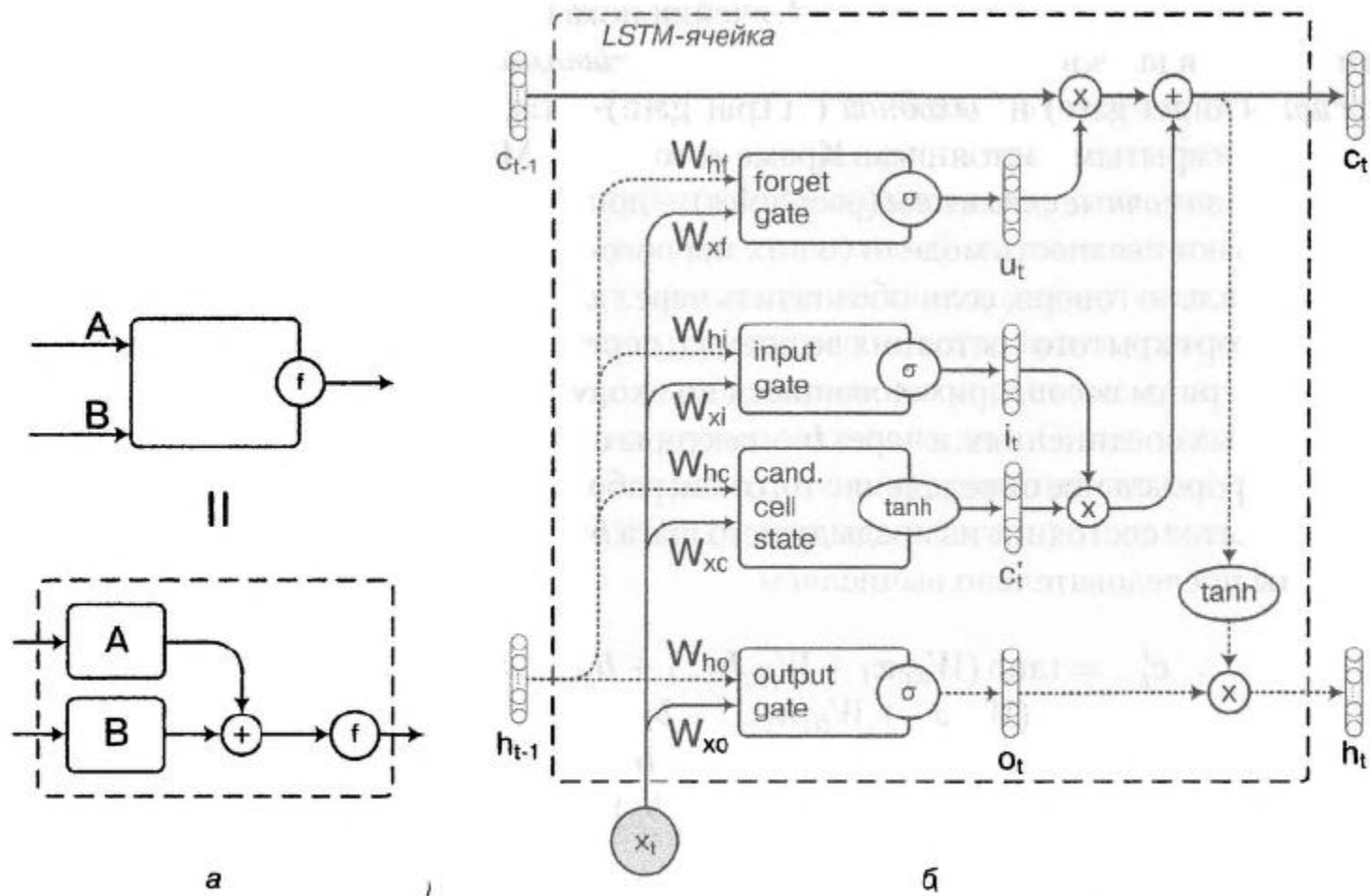


Рис. 6.6. LSTM: а – обозначение гейта с двумя входами; б – структура LSTM-ячейки

• **Сети краткосрочной-долгосрочной памяти** (Long/short term memory, далее LSTM) пытаются бороться с проблемой исчезновения градиента вводя гейты (gates) и вводя ячейку памяти. Каждый нейрон представляет из себя ячейку памяти с тремя гейтами: на вход, на выход и забывания (forget). Эти затворы выполняют функцию телохранителей для информации, разрешая или запрещая её поток.

–Входной гейт определяет, какое количество информации из предыдущего слоя будет храниться в этой ячейке;

–Выходной гейт выполняет работу на другом конце и определяет, какая часть следующего слоя узнает о состоянии текущей ячейки.

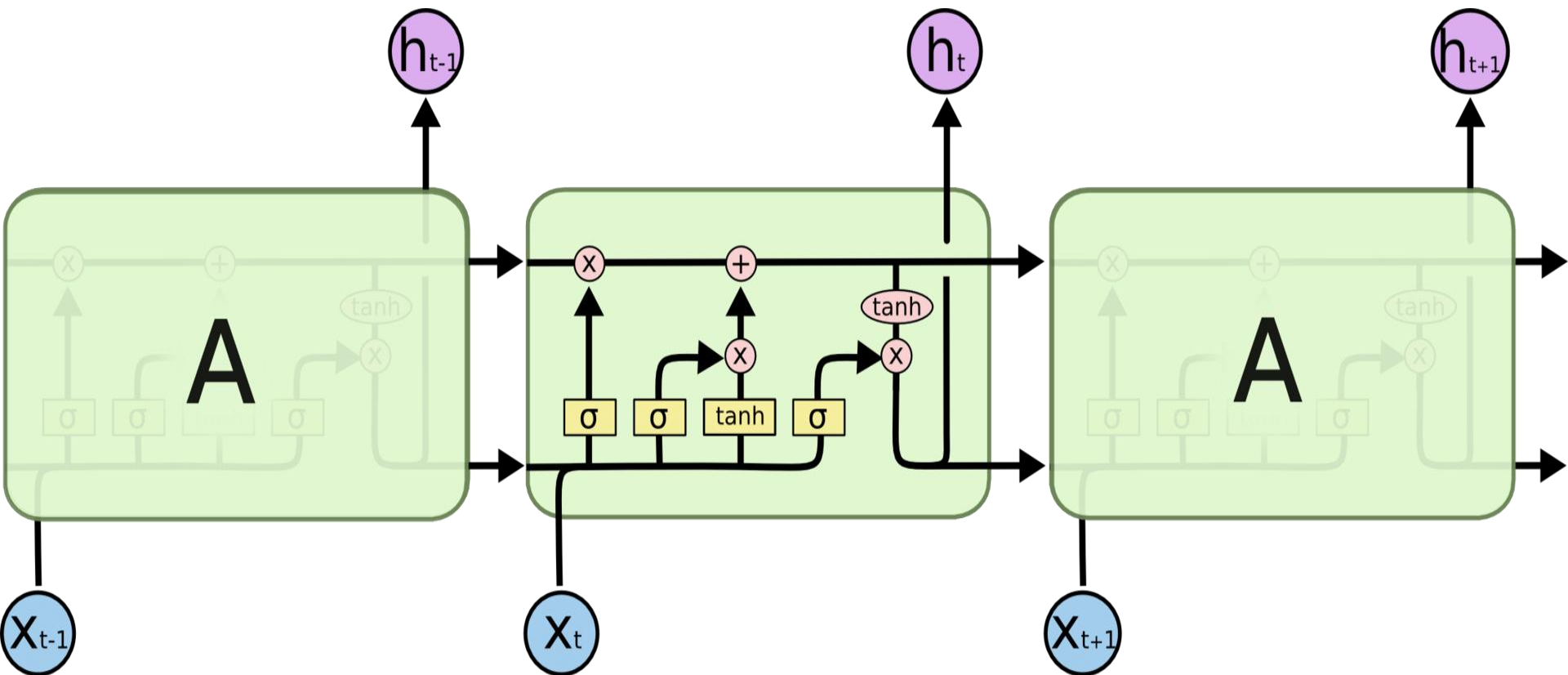
–Гейт забывания контролирует меру сохранения значения в памяти: если при изучении книги начинается новая глава, иногда для нейросети становится необходимым забыть некоторые слова из предыдущей главы.

• Было показано, что LSTM способны обучаться на сложных последовательностях и, например, писать в стиле Шекспира или сочинять примитивную музыку. Заметим, что каждый из гейтов соединен с ячейкой на предыдущем нейроне с определенным весом, что требует больше ресурсов для работы. LSTM распространены и используются в машинном переводе. Помимо этого, это стандартная модель для большинства задач маркировки (labeling) последовательности, которые состоят из большого количества данных.

Формально говоря, если обозначить через $\mathbf{x}_t$ входной вектор во время t , через $\mathbf{h}_t$ — вектор скрытого состояния во время t , через W_i (с разными вторыми индексами) — матрицы весов, применяющиеся ко входу, через W_h — матрицы весов в рекуррентных соединениях, а через $\mathbf{b}$ — векторы свободных членов, мы получим следующее формальное определение того, как работает LSTM: на очередном входе $\mathbf{x}_t$, имея скрытое состояние из предыдущего шага $\mathbf{h}_{t-1}$ и собственно состояние ячейки $\mathbf{c}_{t-1}$, мы последовательно вычисляем

$$\begin{aligned}
 \mathbf{c}'_t &= \tanh(W_{xc}\mathbf{x}_t + W_{hc}\mathbf{h}_{t-1} + \mathbf{b}_{c'}) && \text{candidate cell state} \\
 \mathbf{i}_t &= \sigma(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i) && \text{input gate} \\
 \mathbf{f}_t &= \sigma(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f) && \text{forget gate} \\
 \mathbf{o}_t &= \sigma(W_{xo}\mathbf{x}_t + W_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o) && \text{output gate} \\
 \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{c}'_t, && \text{cell state} \\
 \mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t) && \text{block output}
 \end{aligned}$$

Работа ячейки LSTM



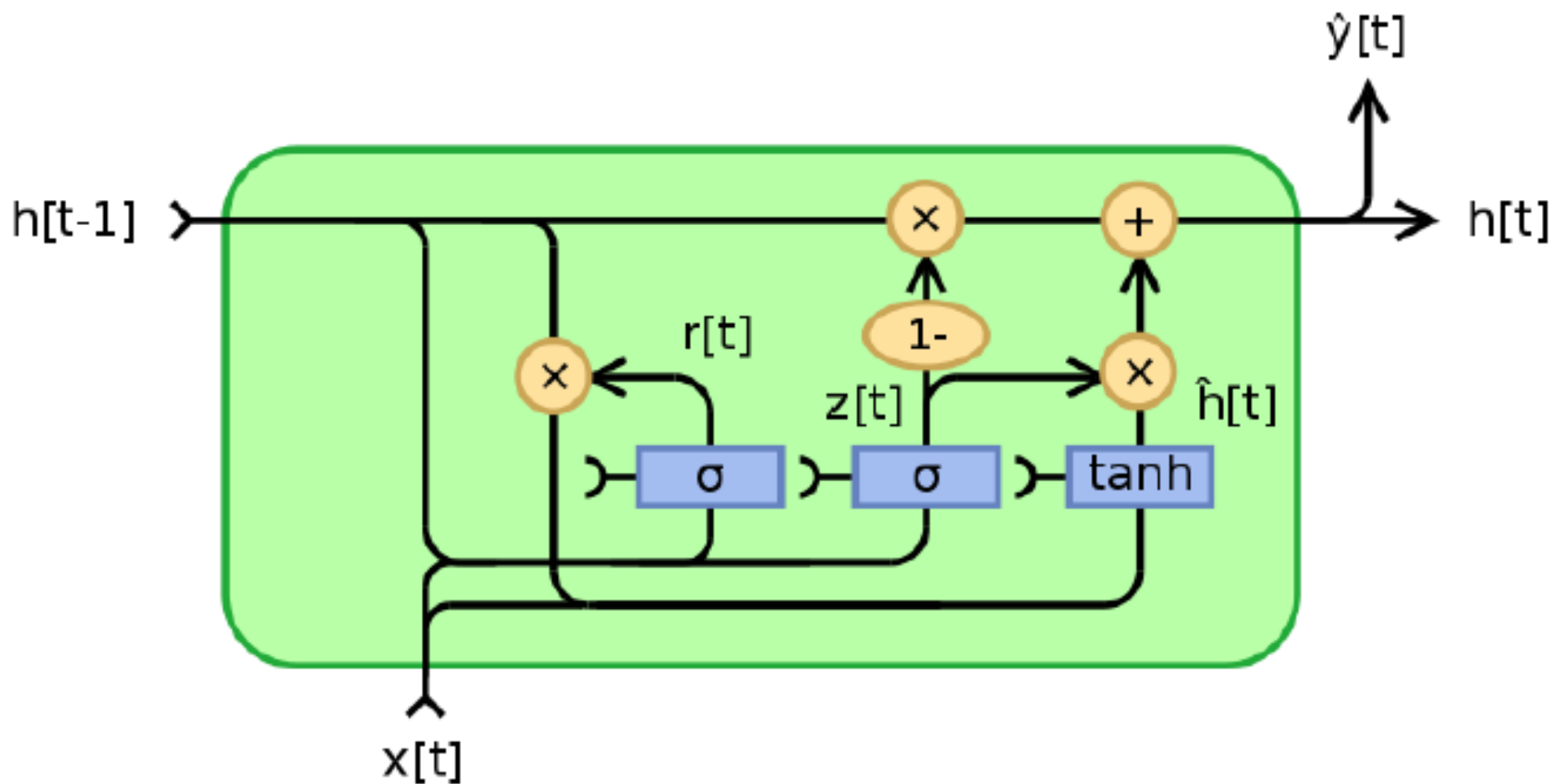
Ключевой компонент LSTM – это состояние ячейки (cell state) – горизонтальная линия, проходящая по верхней части схемы.

LSTM на Python with Keras

```
from keras.models import Sequential
from keras.layers import LSTM, Dense
# Создание модели
model = Sequential()
model.add(LSTM(128, input_shape=(timesteps, input_dim)))
# Добавление слоя вывода
model.add(Dense(num_classes, activation='softmax'))
# Компиляция модели
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Обучение модели
model.fit(X_train, y_train, batch_size=32, epochs=10, validation_data=(X_val, y_val))
# Оценка модели на тестовых данных
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Test Loss:', loss)
print('Test Accuracy:', accuracy)
```

Параметры `batch_size` и `epochs` указывают количество образцов, обрабатываемых за одну итерацию, и количество эпох обучения соответственно.

Модель GRU



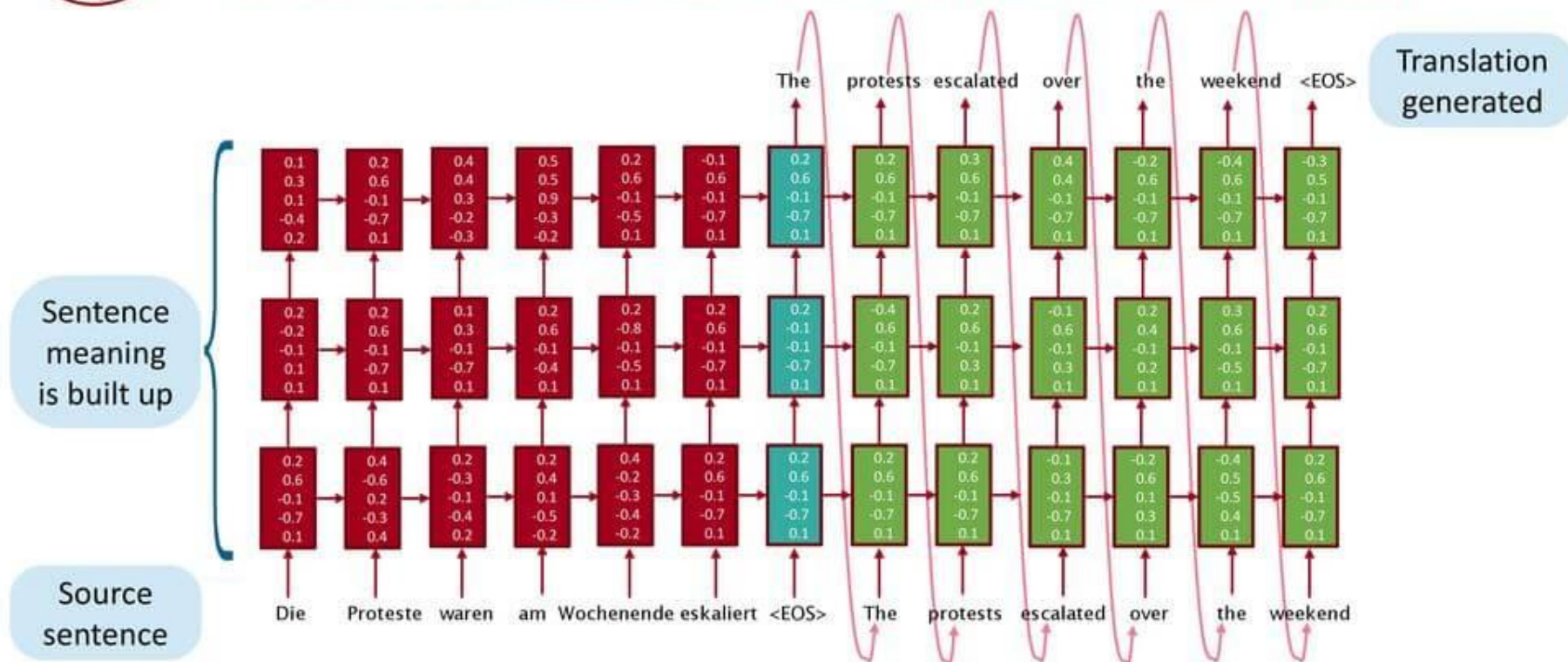
GRU

- **Закрытые рекуррентные блоки** (Gated recurrent units, далее GRU) отличаются от LSTM, хотя тоже являются расширением для нейросетевого машинного обучения. В GRU на один гейт меньше, и работа строится по-другому: вместо входного, выходного и забывания, есть **гейт обновления** (update gate). Он определяет, сколько информации необходимо сохранить с последнего состояния и сколько информации пропускать с предыдущих слоев.
- Функции сброса гейта (reset gate) похожи на затвор забывания у LSTM, но расположение отличается. GRU всегда передают свое полное состояние, не имеют выходной затвор. Часто эти затвор функционирует как и LSTM, однако, большим отличием заключается в следующем: в GRU затворы работают быстрее и легче в управлении (но также менее интерпретируемые). На практике они стремятся нейтрализовать друг друга, так как нужна большая нейросеть для восстановления выразительности (expressiveness), которая сводит на нет приросты в результате. Но в случаях, где не требуется экстра выразительности, GRU показывают лучше результат, чем LSTM.



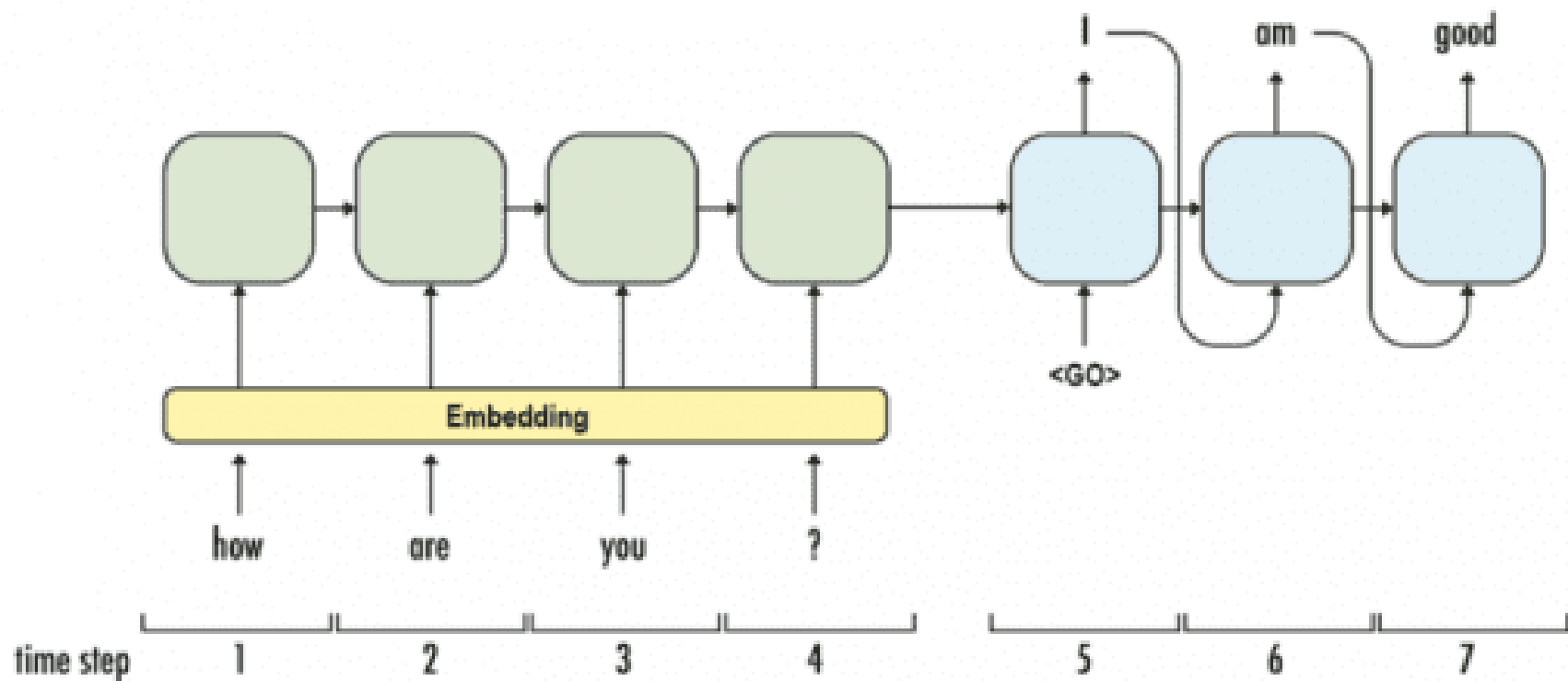
Neural Machine Translation

[Sutskever et al. 2014, Bahdanau et al. 2014, Luong and Manning 2015]



ENCODER

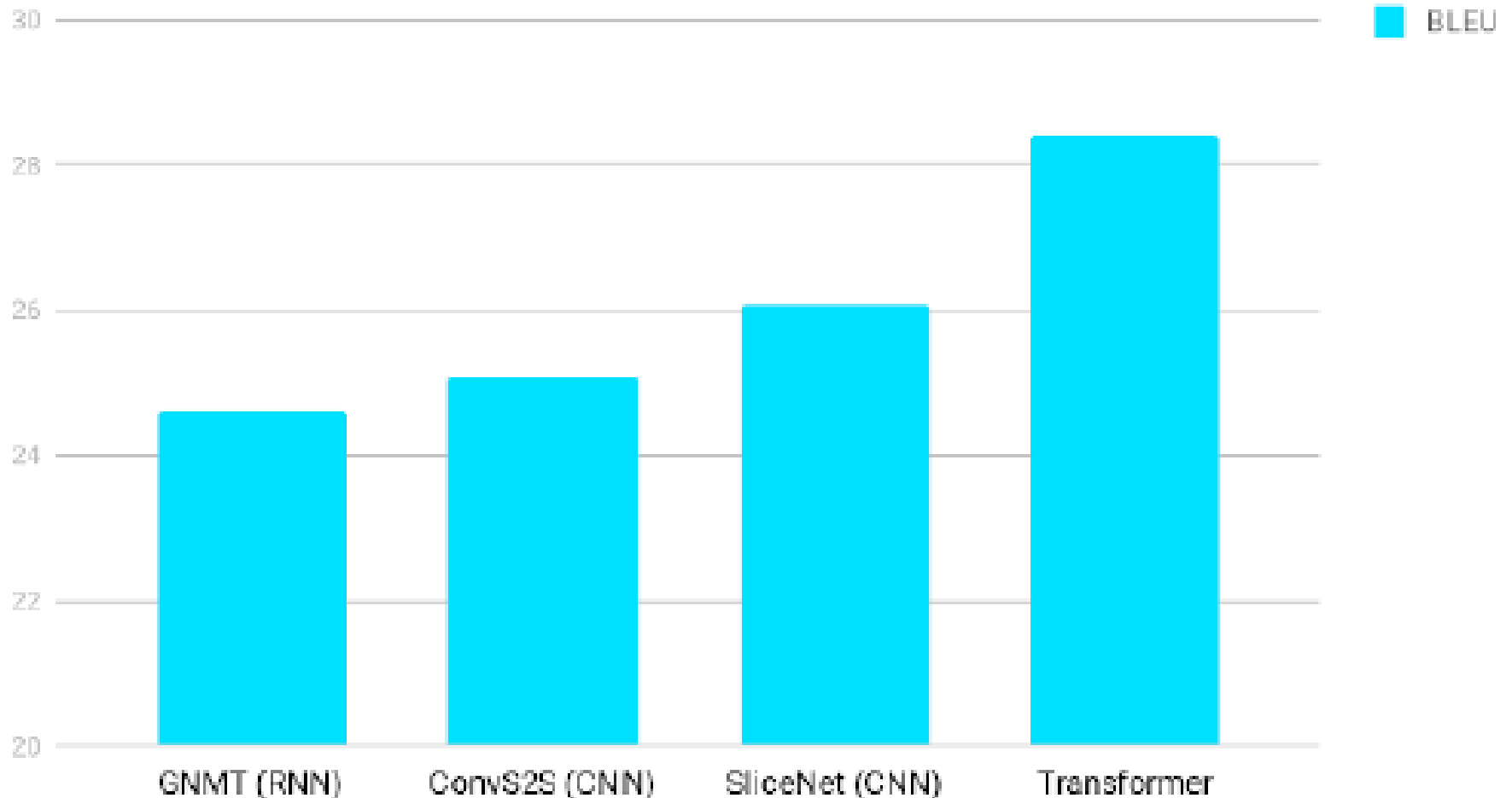
DECODER



- Стандартная модель нейро-машинного перевода является сквозной нейросетью, где исходное предложение кодируется RNN, называемой **кодировщик** (encoder), а целевое слово предсказывается с помощью другой RNN, называемой **декодер** (decoder). Кодировщик «читает» исходное предложение со скоростью один символ в единицу времени, далее объединяет исходное предложение в последнем скрытом слое. Декодер использует обратное распространение ошибки для изучения этого объединения и возвращает переведённую вариант. Удивительно, что находившийся на периферии исследовательской активности в 2014 году нейро-машинный перевод стал стандартом машинного перевода в 2016 году.

Точность перевода с английского на немецкий

English German Translation quality



Рекуррентные модели в своих расчётах обычно учитывают позиции символов на входе и выходе последовательности. В ходе выравнивания позиций по шагам во время вычисления, они генерируют последовательность скрытых состояний h_t в зависимости от предыдущего скрытого состояния h_{t-1} и входных данных для позиции t . Этот последовательный характер вычислений исключает распараллеливание в обучающих примерах, что становится критическим при более длинных последовательностях, поскольку ограничения памяти препятствуют пакетной обработке примеров. Последние исследования в области позволили добиться значительного повышения эффективности вычислений с помощью таких приёмов, как факторизация [21] и условные вычисления [32], а также, как в случае с последним, повышения производительности. Однако на фундаментальном уровне ограничение последовательных вычислений остаётся.

Трансформеры на базе RNN

- Кодер преобразует исходную входную последовательность в ее скрытое представление в виде векторов скрытого состояния.
- Декодер пытается предсказать выходную последовательность, используя это скрытое представление.
- Но у подхода, основанного на RNN, есть свойственный недостаток
- Из-за фундаментального ограничения последовательных вычислений невозможно распараллелить сеть, что затрудняет обучение на длинных последовательностях. Это, в свою очередь, накладывает ограничение на размер пакета, который можно использовать во время обучения.

Внимание и трансферные сети (Transformers)

- <https://habr.com/ru/companies/ruvds/articles/723538/>

Часть 1

- <https://habr.com/ru/companies/ruvds/articles/725618/>

Часть 2

- Ashish Vaswani et al. Attention Is All You Need. -
arXiv:1706.03762v7 [cs.CL] 2 Aug 2023.

Submitted on 12 Jun 2017 (v1), last revised 2 Aug 2023 (v7)
Proc. of NIPS-2017

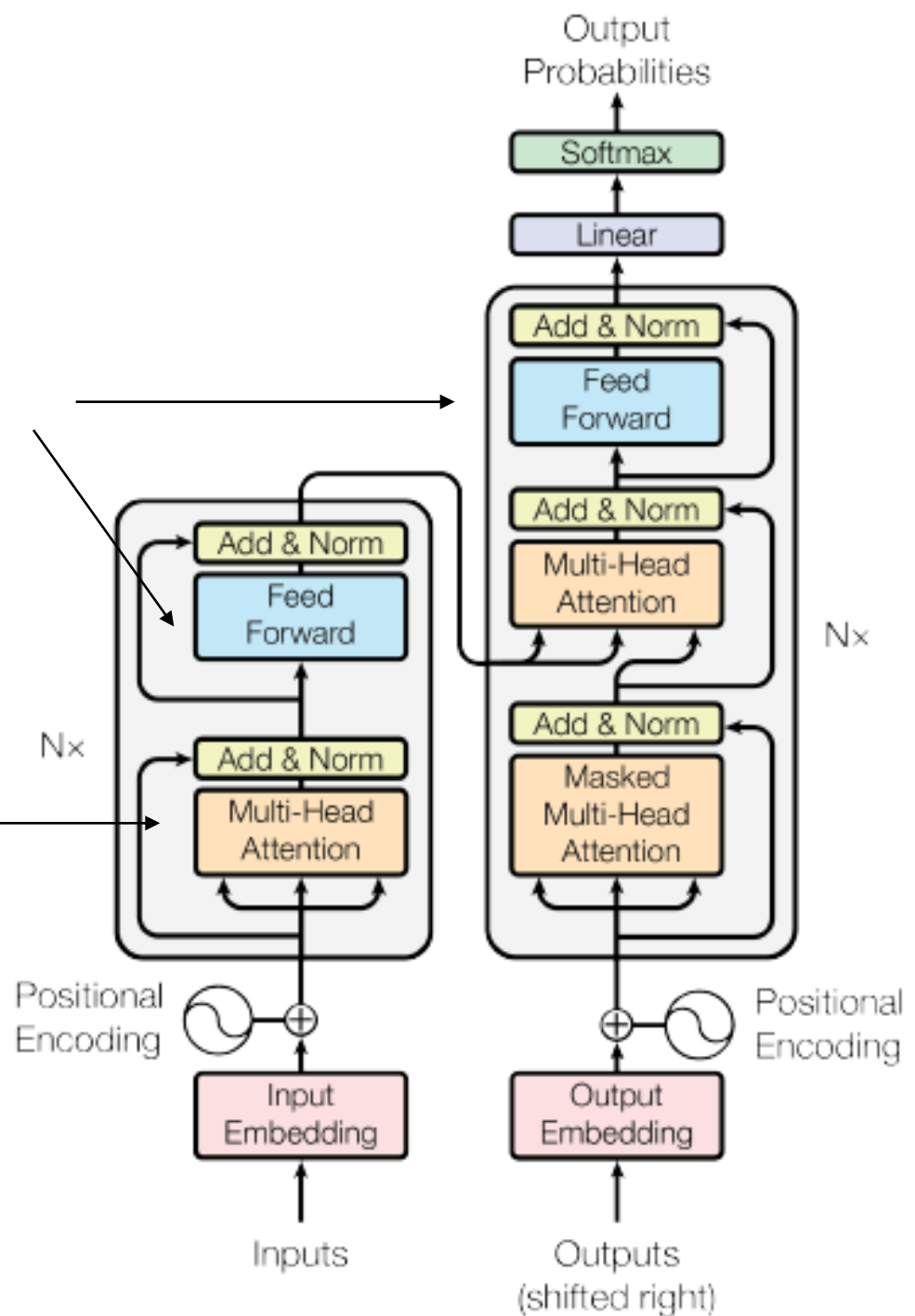
С этой публикации начинаются большие языковые модели

Transformer с ВНИМАНИЕМ

Полносвязные сети
Прямого распространения

Механизм многоголового
внимания

Позиционное кодирование
слов



Энкодер: Энкодер состоит из стеков в $N = 6$ идентичных слоёв. Каждый слой имеет два подслоя. Первый представляет собой механизм самовнимания с несколькими головами, а второй — простой полносвязный слой прямого распространения. Авторы используют остаточное соединение [11] вокруг каждого из двух подслоев с последующей нормализацией слоя [1]. То есть выходом каждого подслоя является $LayerNorm(x + Sublayer(x))$, где $Sublayer(x)$ — это функция, реализуемая самим подслоем. Чтобы облегчить эти остаточные соединения, все подслои в модели, а также слои эмбедингов, производят выходные данные размерности $d_{model} = 512$.

Декодер: Декодер также состоит из стека в $N = 6$ идентичных слоёв. В дополнение к двум подслоям в каждом слое энкодера, в декодере добавляется третий подслой, который запускает механизм внимания на нескольких головах на выходе со стека энкодера. Аналогично энкодеру, авторы используют остаточные соединения вокруг каждого из подслоев с последующей нормализацией слоя. Помимо этого в подслоях декодера модифицируется механизм самовнимания для того, чтобы избежать обращения к последующим позициям. Это маскирование в сочетании с тем фактом, что выходные эмбединги смещены на одну позицию, гарантирует, что предсказания для позиции i могут зависеть только от известных выходов в позициях, меньших i .

Функцию внимания можно описать как сопоставление запроса (query) и набора пар ключ-значение (key-value) с выходными данными, где запрос, ключи, значения и выходные данные являются векторами. Результат вычисляется как взвешенная сумма значений, где вес, присвоенный каждому значению, вычисляется функцией совместимости запроса и соответствующего ключа.

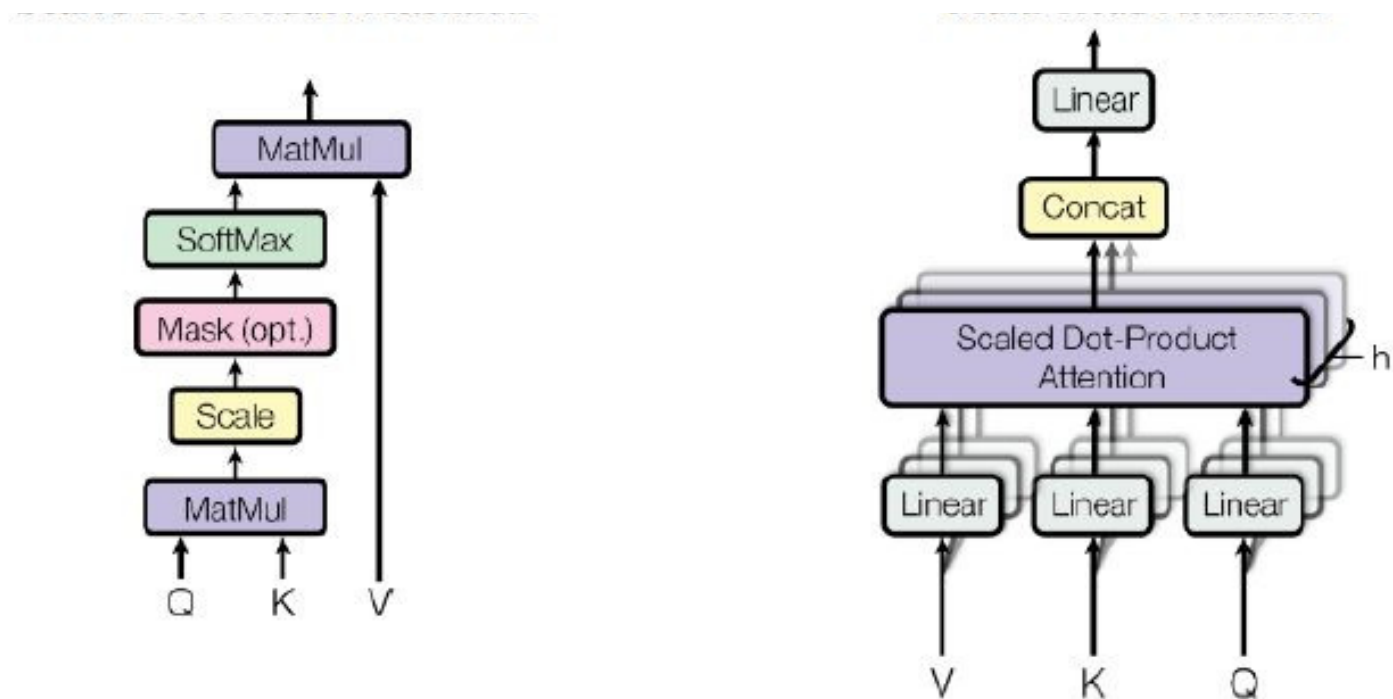


Рис. 2: Слева — внимание на основе масштабированного скалярного произведения. Справа — «многоголовое» внимание состоит из нескольких слоёв внимания, работающих параллельно

Авторы называют свою реализацию механизма внимания «Вниманием на основе масштабированного скалярного произведения» (см. рис. 2). Вход представляет собой запросы и ключи размерностью d_k и значения размерностью d_v . Сначала подсчитываются скалярные произведения запроса со всеми ключами, затем каждое делится на $\sqrt{d_k}$, после чего применяется функция softmax для получения весов для значений.

На практике, функция внимания одновременно подсчитывается на наборе запросов, упакованных в матрицу Q . Ключи и значения также сложены вместе в матрицы K и V соответственно. Итоговая выходная матрица подсчитывается как:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Двумя наиболее часто используемыми функциями внимания являются аддитивное внимание [2] и внимание скалярного произведения (мультипликативное внимание). Последнее идентично предложенному выше алгоритму, за исключением коэффициента масштабирования $\frac{1}{\sqrt{d_k}}$. Аддитивное внимание вычисляет функцию совместимости, используя сеть прямого распространения с одним скрытым слоем. Несмотря на то что теоретическая сложность этих двух методов одинакова, внимание скалярного произведения на практике намного быстрее и экономичнее с точки зрения памяти, поскольку его можно реализовать в виде высокооптимизированного кода умножения матриц.

В то время как при малых значениях d_k оба механизма работают одинаково, при больших значениях d_k аддитивное внимание превосходит внимание на основе скалярного произведения без масштабирования [3]. Авторы предполагают, что при больших значениях d_k , скалярные произведения становятся слишком большими, что толкает функцию softmax в области, где она имеет очень маленькие градиенты.

Чтобы противодействовать этому эффекту, авторы масштабируют скалярные произведения на $\frac{1}{\sqrt{d_k}}$.

Многоголовое внимание

Вместо того, чтобы выполнять единую функцию внимания с ключами, значениями и запросами размерности d_{model} , было решено полезным линейно проецировать запросы, ключи и значения h раз с разными, выученными линейными проекциями на размерности d_k , d_k и d_v соответственно. На каждой из этих спроецированных версий запросов, ключей и значений параллельно выполняется функция внимания, в результате чего получаются выходные значения размерности d_v . Они конкатенируются и снова проецируются, благодаря чему получают окончательные значения, как изображено на рисунке 2.

Многоголовое внимание позволяет модели совместно воспринимать информацию из разных подпространств представлений на разных позициях. При одной голове внимания усреднение препятствует этому.

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^O, \text{ где } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Где проекции — это параметрические матрицы $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ и $W^O \in \mathbb{R}^{hd_v \times d_{model}}$.

В этой работе значение h брали равным 8 параллельным слоям внимания, или «головам». Для каждой из них бралось $d_k = d_v = d_{model}/h = 64$. Из-за уменьшенного размера каждой головы общие вычислительные затраты схожи с затратами на внимание одной головы с полной размерностью.

Трансформер использует многоголовое внимание тремя различными способами:

- В слоях внимания энкодера-декодера запросы приходят от предыдущего слоя декодера, и ключи и значения памяти приходят из выхода энкодера. Это позволяет каждой позиции в декодере обращаться ко всем позициям в выходной последовательности. Это имитирует типичные механизмы внимания энкодера-декодера в моделях типа sequence-to-sequence.
- Энкодер содержит слои самовнимания. В таком слое все ключи, значения и запросы приходят из одного и того же места, в данном случае — выхода предыдущего слоя энкодера. Каждая позиция в энкодере может обращаться ко всем позициям в предыдущем слое энкодера.
- Схожим образом слои самовнимания в декодере позволяют каждой позиции в декодере обращаться ко всем позициям в декодере до текущей позиции включительно. Здесь важно предотвратить движение информации влево в декодере для сохранения свойства авторегрессии. Авторы реализовали внутри механизма внимания на основе масштабированного скалярного произведения, маскировав (установив значение $-\infty$) все значения во входных данных для функции softmax, которые соответствуют нежелательным соединениям

Большие языковые модели

- Большие языковые модели (LLM) — это передовые системы искусственного интеллекта (ИИ), предназначенные для обработки, понимания и создания текста, подобного человеческому. Они основаны на методах глубокого обучения и обучены на массивных наборах данных, обычно содержащих миллиарды слов (или изображений) из различных источников, таких как веб-сайты, книги и статьи. Это обширное обучение позволяет LLM понимать нюансы языка, грамматики, контекста и даже некоторые аспекты общих знаний.
- <https://ru.shaip.com/blog/a-guide-large-language-model-llm/> - 2024 г.

- Некоторые популярные LLM, такие как OpenAI GPT-3 (-3.5, -4, -o1), используют тип нейронной сети, называемый преобразователем (transformer), который позволяет им решать сложные языковые задачи с замечательным мастерством. Эти модели могут выполнять широкий спектр задач, таких как:
 - Отвечать на вопросы
 - Обобщать текст (text summarization)
 - Переводить с одного языка на другой
 - Генерировать контент (ЕЯ текст, программу или ее фрагмент на языке Python, музыку, изображение, видеоряд)
 - Участвовать в интерактивных беседах с пользователями

Примеры бесплатных НС для общения

1. *Replika* — это умный чат-бот, работающий как на платной, так и бесплатной основе и позволяющий создавать виртуального собеседника. *Replika* использует нейросеть для анализа ваших сообщений и создания натуральных ответов, она также учится на основе ваших предпочтений и становится все более персонализированной по мере общения с вами. Служит неплохим источником психотерапевтической помощи.

2. *AI Dungeon* — это уникальная текстовая приключенческая игра, которая использует нейросеть для создания интерактивных историй. Пользователь может предлагать свои действия и решения, а нейросеть будет развивать историю в соответствии с указаниями. При этом открываются безграничные возможности для интерактивного повествования.

3. *Cleverbot* является одним из самых известных чат-ботов, использующих искусственный интеллект. Этот бот способен вести разговор на различные темы, отвечая на вопросы пользователя и учась на основе предыдущих бесед, чтобы обеспечить более уместные ответы. Клевербот способен распознавать даже эмоциональное состояние «человеческого» собеседника и подстраиваться под него в ходе разговора.

4. *ChatGPT* — это нейросеть, обученная на огромных объемах текстовых данных, которая может участвовать в разговоре на любые темы, от обсуждения новостей до философских бесед. Эта нейросеть предоставляет впечатляющую способность генерировать качественные, культурно насыщенные и информативные ответы.

5. *Xiaolce* — это AI-чатбот китайского происхождения, разработанный компанией *Microsoft*, который был обучен на миллионах разговоров (что, согласитесь, несложно сделать в Китае с его численностью населения). Нейросеть способна поддерживать обсуждения на различные темы, а также предоставлять советы по запросу. Это скорее голосовой ассистент, который, помимо функции коммуникации, осуществляет и другие действия - вплоть до написания стихов, пения песен, и так далее. Хороший тамада, другими словами.

GPT-3

- На высоком уровне обучение нейронной сети GPT-3 состоит из двух этапов:
 - На первом этапе необходимо создать словарь, различные категории и правила генерации. Это делается путем кормления GPT-3 книгами. Для каждого слова модель должна предсказать категорию, к которой принадлежит слово, а затем создать правило генерации.
 - Второй шаг состоит из создания словаря и правил генерации для каждой категории. Это делается путем подачи в модель предложений. Для каждого предложения модель должна предсказать категорию, к которой принадлежит каждое слово, а затем создать правило генерации. Результатом обучения является словарный запас и правила генерации для каждой категории.
- У модели также есть несколько хитростей, которые позволяют ей улучшить свои способности к генерации текстов. Например:
 - Она способна угадать начало слова, наблюдая за контекстом слова.
 - Она также может предсказать следующее слово, глядя на последнее слово предложения.
 - Она также способна предсказывать длину предложения. Хотя эти два шага и связанные с ними приемы могут показаться простыми в теории, на практике они требуют огромного количества вычислений. Обучение 175 миллиардов параметров в середине 2020 года стоило примерно 4,6 миллиона долларов, хотя по некоторым другим оценкам это может занять до 12 миллионов долларов в зависимости от того, как было подготовлено оборудование.

Категории слов

- Noun – существительное
- Verb – глагол
- Subject – субъект
- Adjective – прилагательное
- Adverb – наречие
-

Некоторые правила генерации

- noun + verb = subject + verb
- noun + verb + adjective = subject + verb + adjective
- verb + noun = subject + verb
- noun + verb + noun = subject + verb + noun
- noun + noun = subject + noun
- noun + verb + noun + noun = subject + verb + noun + noun

Проблемы и ограничения LLM

- LLM не мыслят, они предсказывают последовательности слов. У них нет понимания мира, как у человека. Можно сказать, что их цель – обмануть человека и заставить его поверить, что с ним разговаривает личность.
- Они могут галлюцинировать (придумывать факты) и быть чрезмерно уверенными в своих ошибках.
- Они чувствительны к формулировкам: небольшое изменение в запросе может кардинально изменить ответ.
- Они отражают «смещения», присутствующие в их обучающих данных, т.е. фокусируются на общепринятых (широко представленных в Интернете) представлениях.
- Ограниченная длина контекста, т.е. существующие модели могут обрабатывать тексты ограниченной длины, хотя в последних моделях (например, Deepseek) величина контекста достигает внушительного размера, сравнимого с объемом книги.
- Ошибки, которые могут появиться в начале сгенерированного текста могут вызывать появление дальнейших ошибок в нем (эффект накопления ошибок).
- LLM обучаются на очень больших наборах данных методом обучения с учителем, в то время как СИИ должен иметь возможность обучаться на одинарных примерах и самообучаться (без учителя).

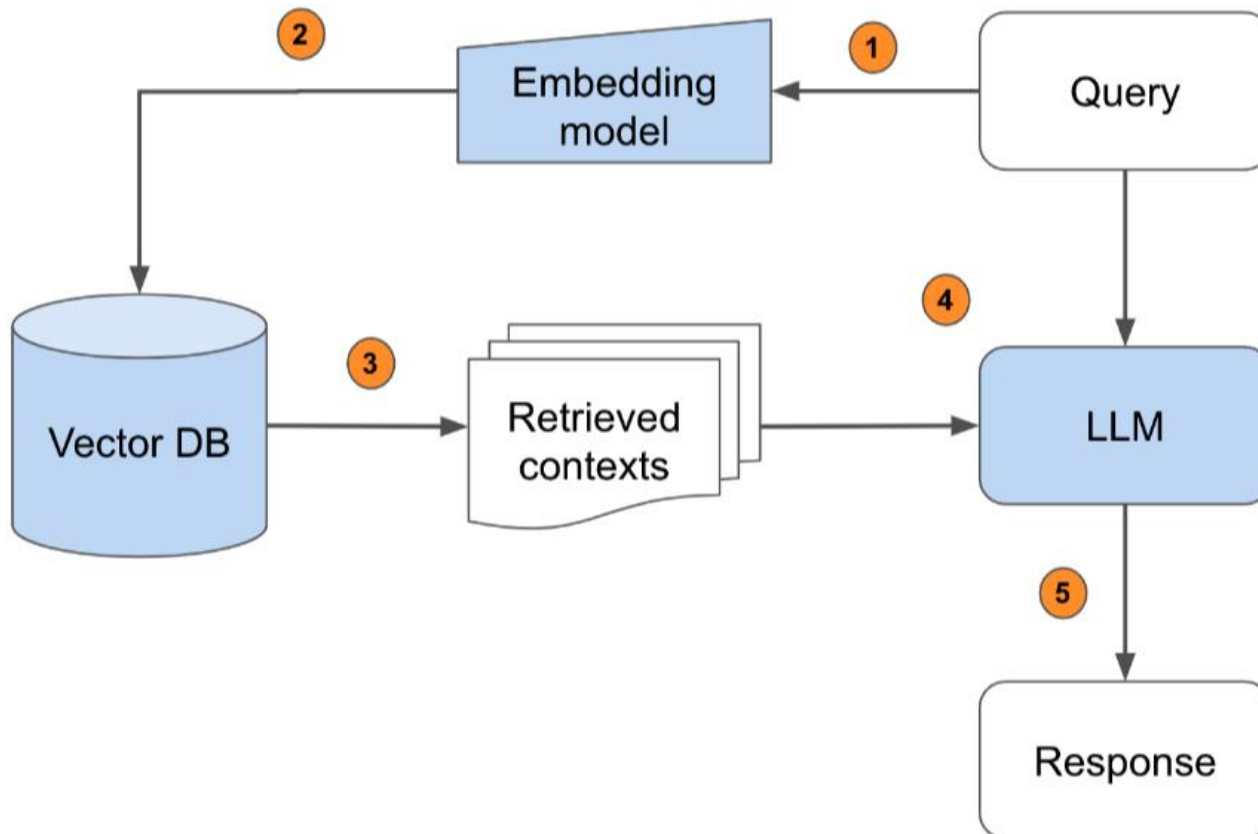
Пути совершенствования LLM

- Объединение с другими методами обработки данных (гибридный подход)
- Метод MoE (Mixture of Experts)

RAG

What is Retrieval Augmented Generation (RAG)?

The purpose of RAG is to enhance the ability of a generative model, like a large language model (LLM), by providing it access to a vast external knowledge base it can draw from in real-time to generate more accurate, detailed, and contextually relevant responses.

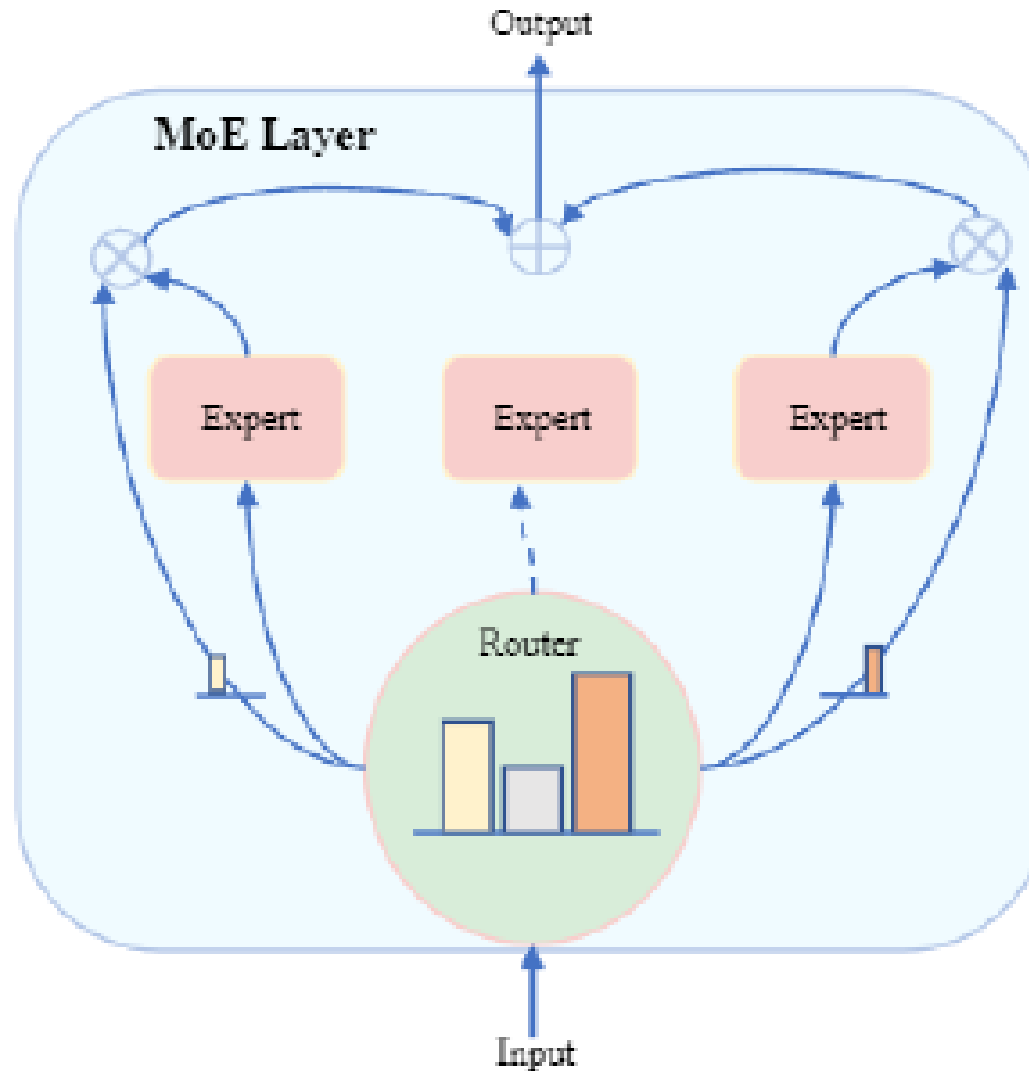


MoE

- Появилась в 1991 году
 - R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, “Adaptive mixtures of local experts,” *Neural computation*, vol. 3, no. 1, pp. 79–87, 1991.
- Популярность получила только в рамках LLM в последние несколько лет.
- Стратегия «Разделяй и властвуй». В то время как обычные модели активируют все параметры для каждого входа, модели MoE динамически выбирают и активируют только наиболее релевантное подмножество параметров в зависимости от характеристик входных данных.

- Этот подход не только повышает специализацию отдельных «экспертов», но и смягчает трудности, связанные с обучением на разнообразных и конфликтующих задачах.
- Механизм выборочной активации позволяет моделям МоЕ значительно расширить свою емкость и обрабатывать различные области знаний без пропорционального увеличения вычислительных затрат, достигая тем самым оптимального баланса между производительностью и эффективностью

Архитектура MoE



Функция маршрутизации

- Функция маршрутизации служит математической реализацией роутера, определяя, как входные данные распределяются между назначенными экспертами. Очевидно, что эффективность моделей МоЕ неразрывно связана с этой стратегией распределения, что делает разработку функции маршрутизации критическим аспектом.

- Линейная маршрутизация

$$G(x)_i = \text{softmax}(\text{TopK}(g(x) + R_{\text{noise}}, k))_i \quad (1)$$

$$\text{TopK}(v) = \begin{cases} v_i & \text{если } v_i \text{ является одним из лучших элементов } k, \\ -\infty & \text{иначе.} \end{cases}$$

- Обычно МоЕ встраивается в Трансформер вместо слоя прямого распространения (FFN)

Нелинейная маршрутизация

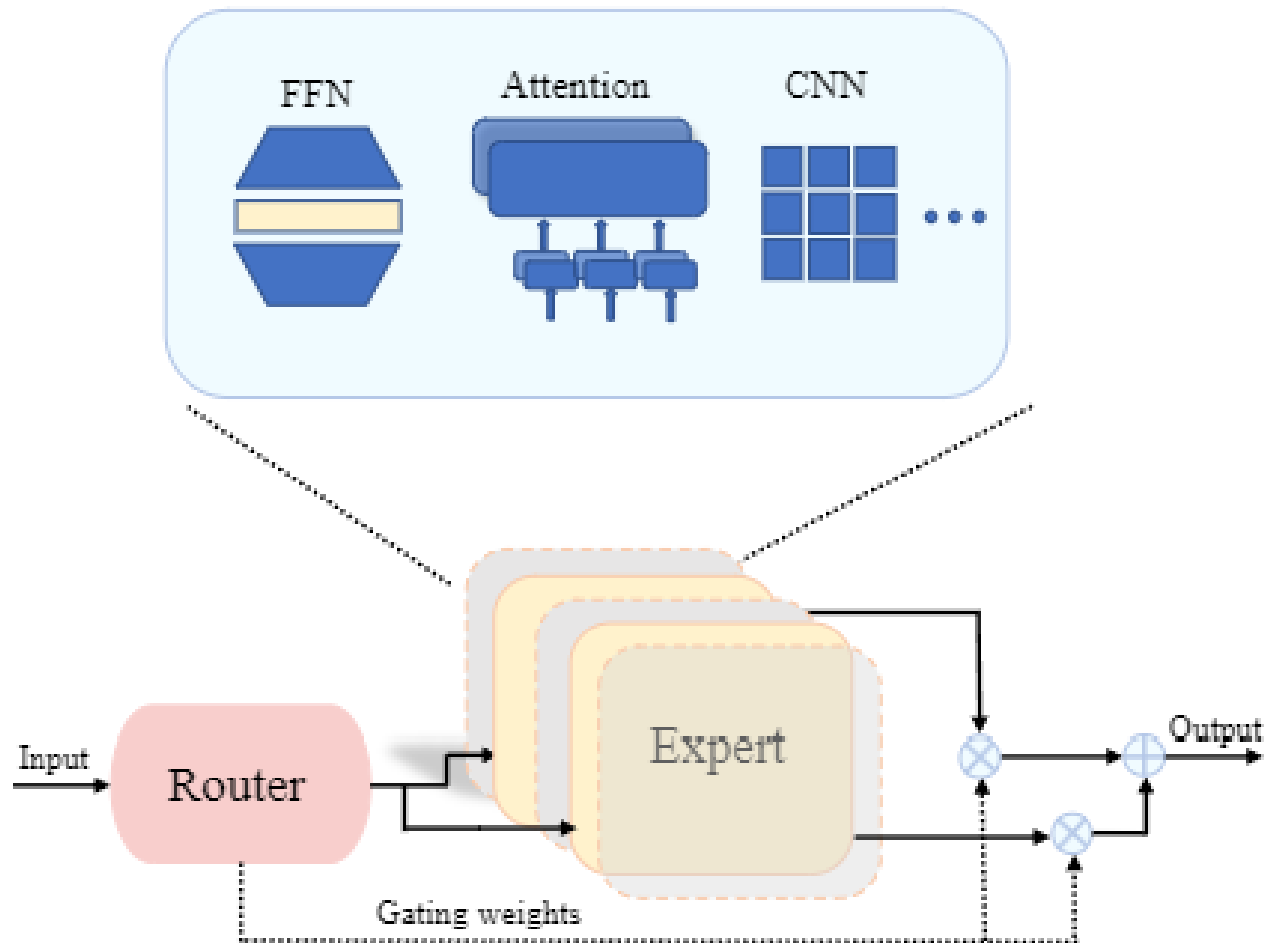
- на основе косинусного расстояния, которая выглядит следующим образом:

$$G(x) = \text{TopK}(\text{softmax} \left(\frac{E^T W_{linear} x}{\tau \|W_{linear} x\| \|E\|} \right))$$

где $x \in R^{d_e}$ – вход модуля MoE, $W_{linear} \in R^{d_e}$ – обучаемое линейное преобразование для x , отвечающее за проецирование x в пространство гиперсферы. Количество экспертных сетей равно N , а $E \in R^{d_e * N}$ представляет признаки N экспертных сетей. В функции маршрутизации вход x проецируется в новое пространство и сравнивается с эмбедингами экспертов E с использованием косинусного сходства для определения того, каким экспертам должен быть назначен каждый вход x . Таким образом, эмбединги экспертов E могут захватывать представления признаков различных экспертов и постепенно оптимизируются во время обучения для лучшей адаптации к требованиям задачи. Кроме того, температурный параметр τ контролирует резкость распределения маршрутизации. Меньшее τ делает распределение более резким, отдавая предпочтение выбору нескольких экспертов, в то время как большее τ делает распределение более гладким, позволяя участвовать большему числу экспертов.

- Существуют другие нелинейные функции маршрутизации, например:
- функция маршрутизации в Soft MoE, которая не использует механизм дискретного распределения токенов.
- В частности, Soft MoE вычисляет веса между каждым токеном и каждым экспертом, затем использует эти веса для взвешенного усреднения для генерации входных слотов. Следовательно, слоты, обрабатываемые каждым экспертом, представляют собой взвешенную комбинацию всех токенов, а не отдельные токены. Этот подход избегает потенциальных проблем с потерей токенов из-за дисбаланса нагрузки и трудностей оптимизации, вызванных дискретными операциями, такими как TopK.

Различные экспертные сети для схемы уровня MoE



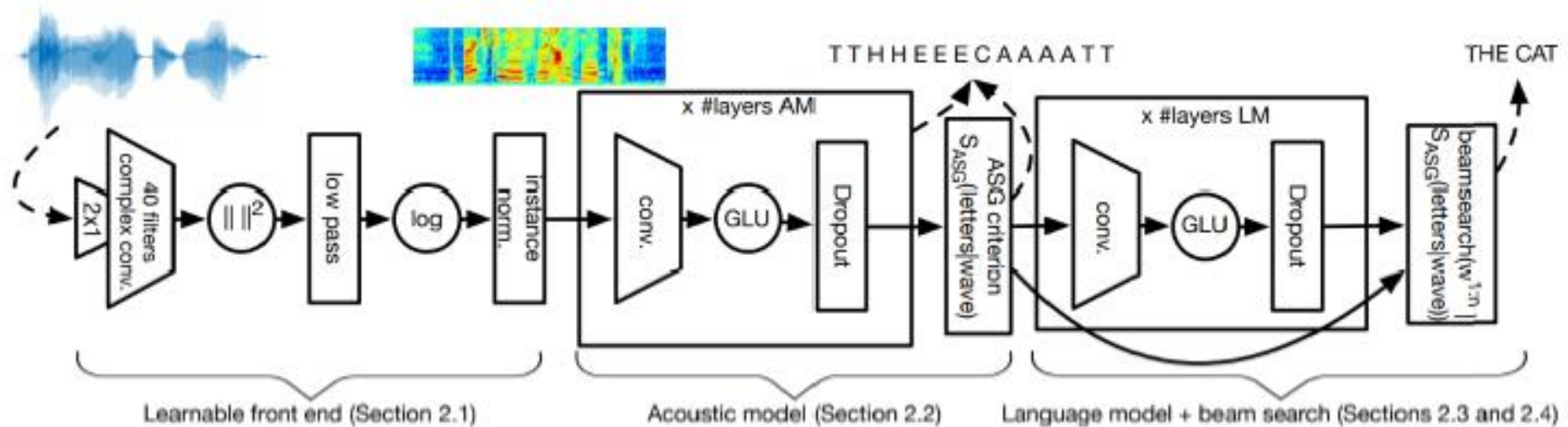
ИСТОЧНИК о МоЕ

- Siyuan Mu and Sen Lin. A Comprehensive Survey of Mixture-of-Experts: Algorithms, Theory, and Applications, 2025. – [Электронный ресурс] - <https://arxiv.org/abs/2503.07137>

Голосовые помощники

- Большинство разработок фокусируется на вертикальных чат-ботах, мессенджер-платформах, возможностях для стартапов (Amazon Alexa, Apple Siri, Facebook M, Google Assistant, Microsoft Cortana, Яндекс Алиса).
- Способности ИИ понимать естественный язык пока остаются ограничены, поэтому создание полноценного разговорного ассистента остается открытой задачей.

Сверточная сеть для распознавания речи



Вопросно-ответные (QA) системы

- Идея **вопросно-ответных (Question-answering, далее — QA) систем** заключается в извлечении информации непосредственно из документа, разговора, онлайн поиска или любого другого места, удовлетворяющего потребности пользователя. Вместо того, чтобы заставлять пользователя читать полный текст, QA системы предпочитают давать короткие и лаконичные ответы. Сегодня QA системы легко комбинируются с чат-ботами, выходят за пределы поиска текстовых документов и извлекают информацию из набора картинок.
- Большинство NLP задач могут быть рассмотрены как вопросно-ответные задачи. Парадигма проста: отправляется запрос, на который машина предоставляет ответ. Через чтение текста или набора инструкций разумная система должна находить ответ на большой круг вопросов. Естественно, требуется создать модель для ответов на общие вопросы.

Примеры работы QA-системы с вопросами о содержании изображений



What vegetable is on the plate?

Neural Net: **broccoli**
Ground Truth: broccoli



What color are the shoes on the person's feet ?

Neural Net: **brown**
Ground Truth: brown



How many school busses are there?

Neural Net: **2**
Ground Truth: 2



What sport is this?

Neural Net: **baseball**
Ground Truth: baseball



What is on top of the refrigerator?

Neural Net: **magnets**
Ground Truth: cereal



What uniform is she wearing?

Neural Net: **shorts**
Ground Truth: girl scout



What is the table number?

Neural Net: **4**
Ground Truth: 40



What are people sitting under in the back?

Neural Net: **bench**
Ground Truth: tent

- Специально для QA задач создана и оптимизирована мощная архитектура глубокого обучения — Сеть Динамической Памяти (Dynamic Memory Network, далее — DNM). Обученная на тренировочном наборе из входных данных и вопросов, DNM формирует эпизодические воспоминания и использует их для генерации подходящих ответов. Эта архитектура состоит из следующих компонент:
 - **Модуль семантической памяти**, аналогичный базе знаний, состоит из предварительно подготовленных GloVe векторов, которые используются для создания последовательностей векторных представлений слов из входящих предложений. Эти вектора будут использоваться, как входные данные модели.
 - **Входной модуль** перерабатывает связанные с вопросом входящие вектора в наборов векторов, называемый фактами. Этот модуль реализован с помощью Управляемого рекуррентного блока (Gated Recurrent Unit, далее — GRU), который позволяет сети узнать релевантность рассматриваемого предложения.
 - **Вопросный модуль** обрабатывает вопрос слово за словом и выдает вектор, используя тот же GRU, что и в входном модуле, с такими же весами.
 - **Модуль эпизодической памяти** сохраняет извлеченные на входе векторы фактов и вопросов, закодированные как вложения. Это похоже на происходящий в гиппокампе мозга процесс по извлечению временных состояний в ответ на звук или вид.
 - **Ответный модуль** генерирует подходящий ответ. На последнем шаге эпизодическая память содержит необходимую для ответа информацию. Этот модуль использует другой GRU, обученный с классификацией кросс-энтропийной ошибки верной последовательности, которая конвертируется обратно на естественный язык.

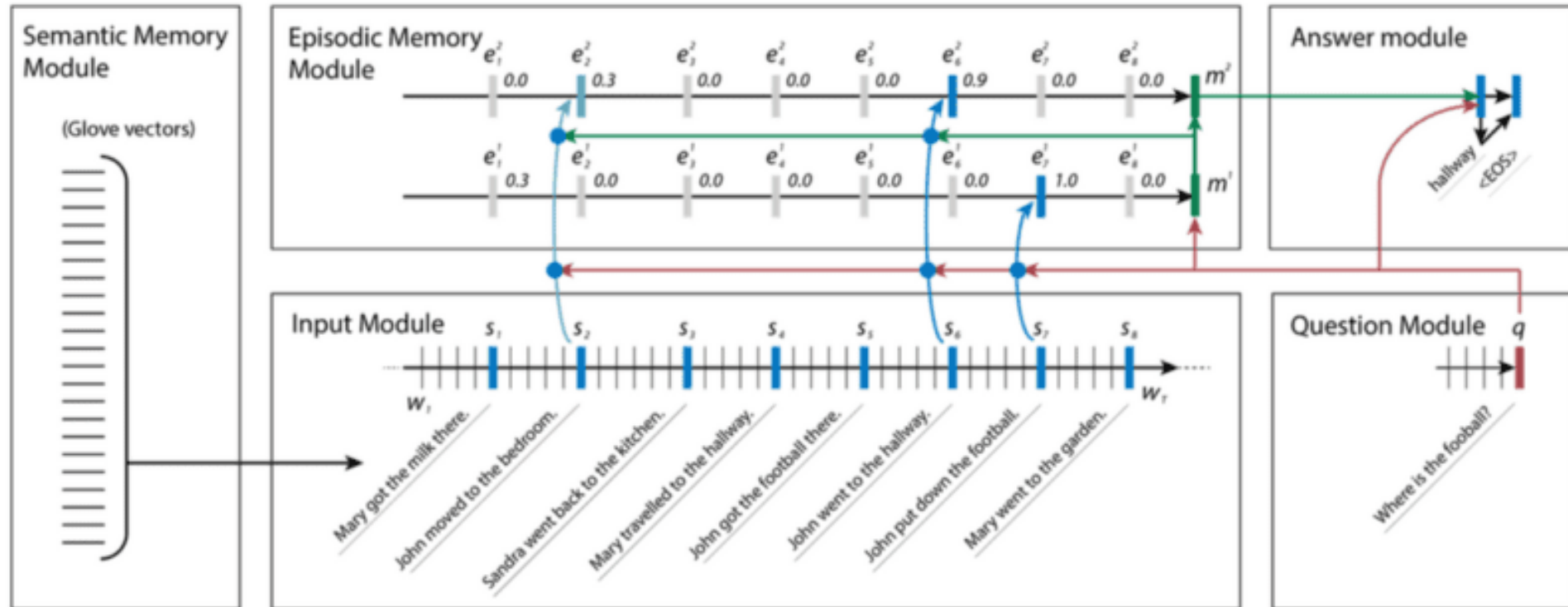


Illustration of DMN performing transitive inference.

- DNM хорошо справляется с QA задачами и превосходит в результатах другие архитектуры для семантического анализа и частеречной разметки (part-of-speech tagging). С момента выпуска начальной версии DNM претерпела ряд улучшений для дальнейшего совершенствования точности в QA задачах:
- DNM для ответов на текстовые и визуальные вопросы — DNM применимая к изображениям. Здесь входные модули и модули памяти модернизированы для ответов на визуальные вопросы. Такая модель улучшает результат существующей архитектуры на большинстве тестов на визуальных вопросно-ответных датасетах без учителя.
- Динамические coattention-сети для ответов на вопросы выступают с решением проблемы выхода из локального максимума, соответствующего неправильному ответу. Модель сливает со-зависимые представления вопроса и текст, чтобы сфокусироваться на их подходящих частях. Далее динамический указательный декодер проходит через полный набор потенциальных ответов.