



Тверской  
государственный  
технический  
университет

# Интеллектуальные информационные системы

Архитектуры НС для работы с  
последовательностями

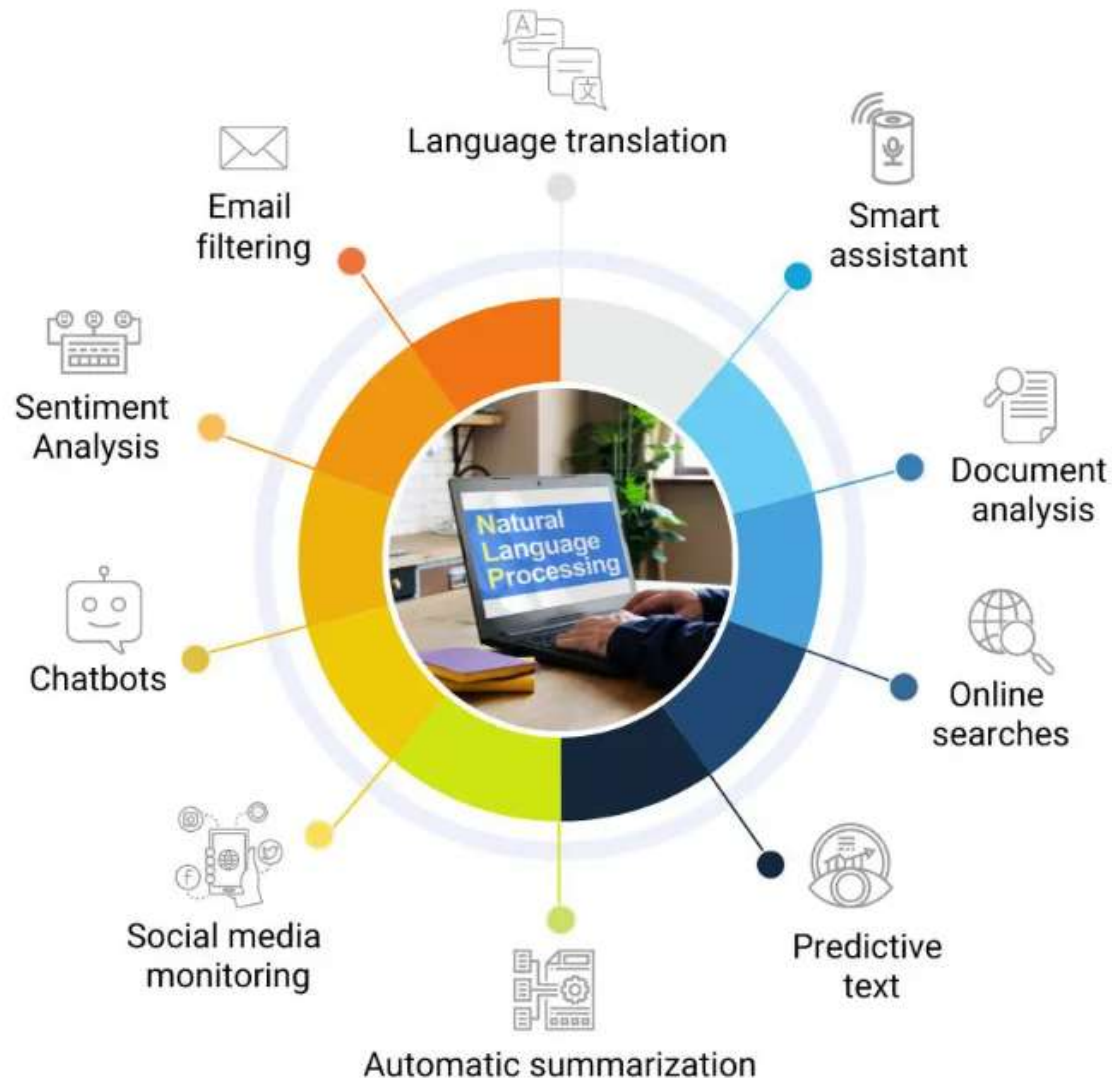
2025 г.

# Задачи обработки последовательностей

*Natural Language Processing (NLP)* – анализ текстов, написанных на естественных языках:

- Классификация текста, анализ тональности документов, поиск релевантных документов по запросу и их ранжирование
- Синтез и распознавание речи
- Машинный перевод
- Генерация текста
- Диалоговые системы и чат-боты
- Суммаризация обращений

# Applications of Natural Language Processing

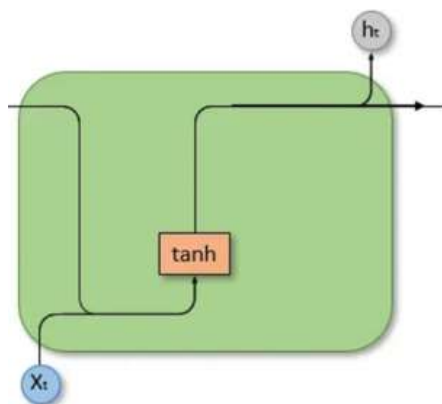


# Задачи обработки последовательностей

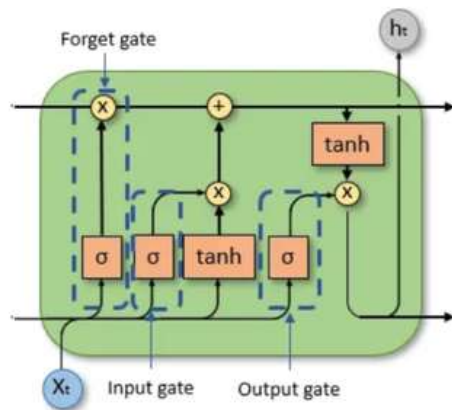
Временные ряды – значения меняющихся во времени признаков, полученные в некоторые моменты времени

- Прогнозирование временных рядов как самостоятельная задача – прогнозирование продаж, спроса, трафика, стоимости и множество других
- Обработка сигналов
- Поиск аномалий
- Интерпретация генома и задачи биоинформатики

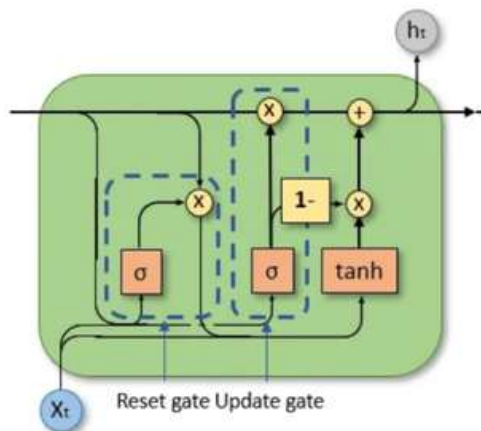
## RNN



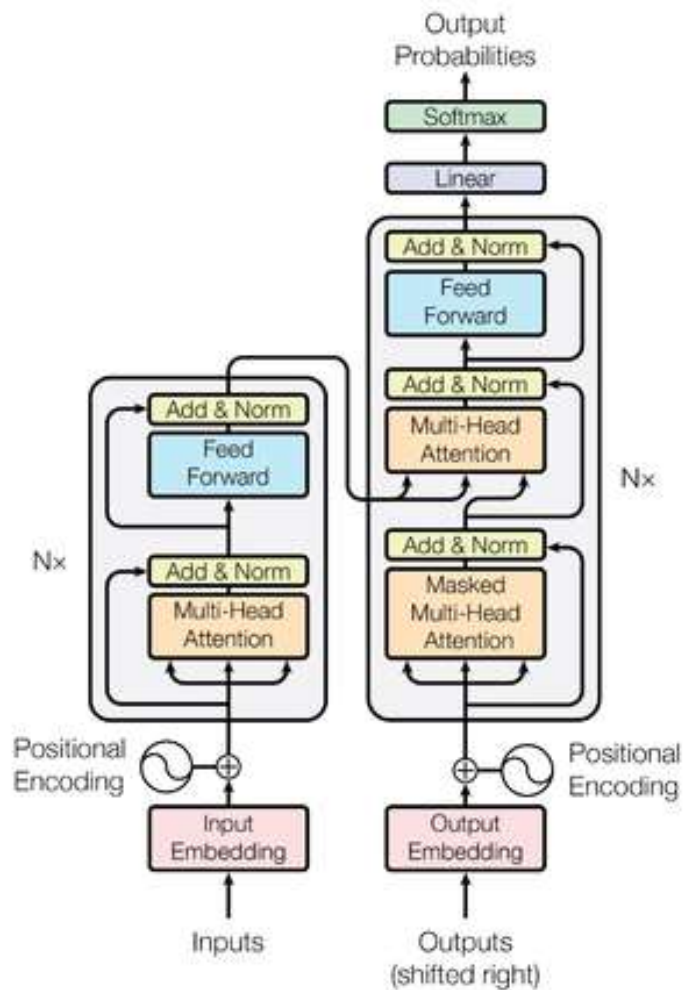
## LSTM



## GRU



# Transformers



# Способы работы с последовательностями:

1. One-to-one
2. One to many
3. Many to one
4. Many to many (синхронизированный или нет)

one to one

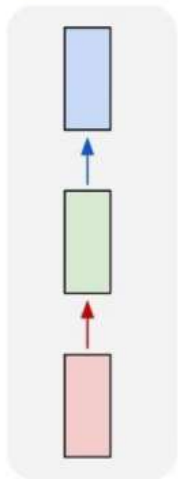


Image in  
Label out

one to many

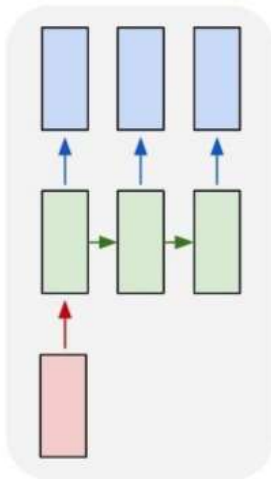
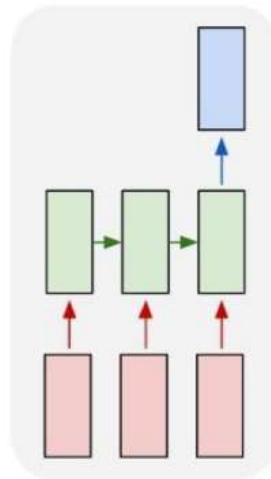


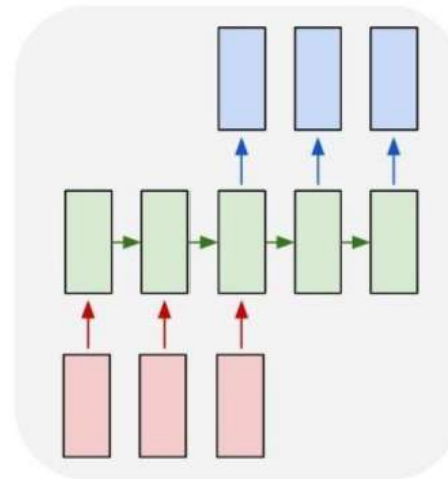
Image in  
Words out

many to one



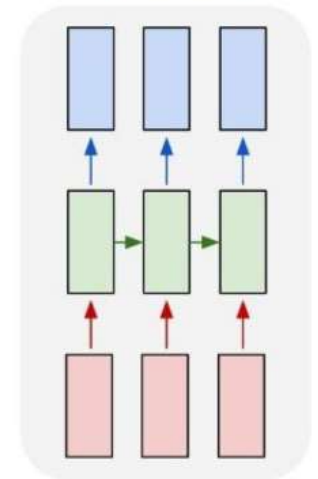
Words in  
Sentiment out

many to many



English in  
Portuguese out

many to many



Video In  
Labels out

Many-to-one

Было сложно, мне понравилось



Позитивный

One-to-many



Пингвин в шляпе

Many-to-many, не синхронизированный

I seem to have overfitted



Кажется, я переобучился

Many-to-many, синхронизированный

Будет



глагол

сложно,



наречие

вам



местоимение

понравится



глагол

# Задача обработки последовательностей

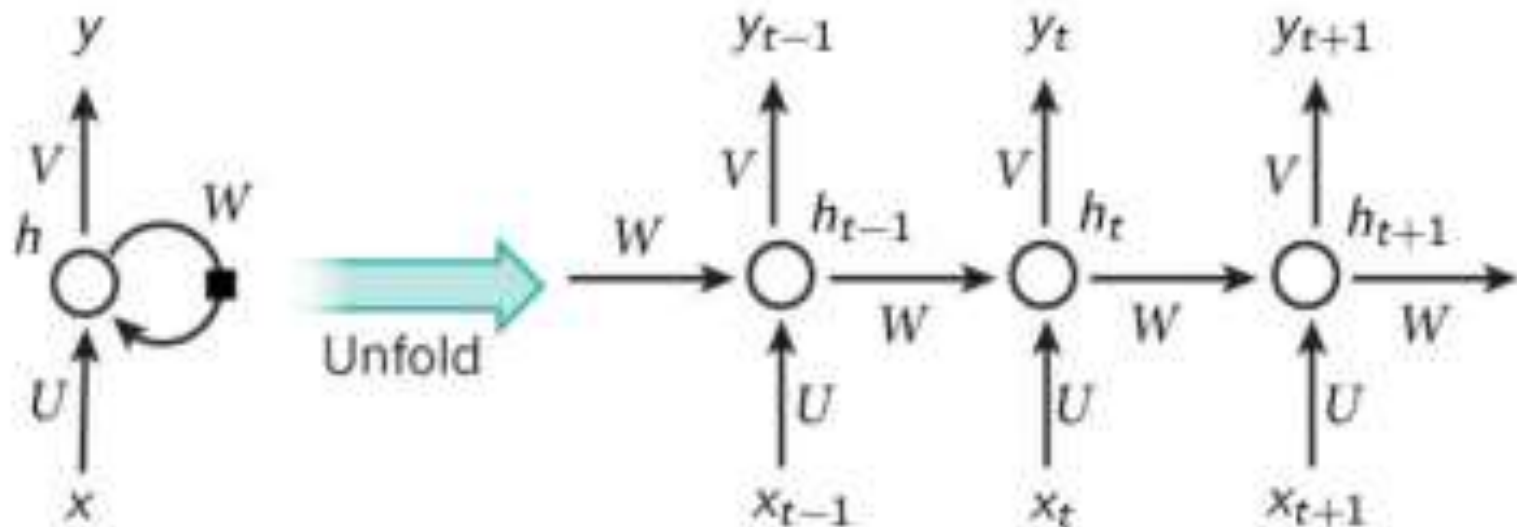
$x_t$  - входной вектор в момент  $t$

$h_t$  - вектор скрытого состояния в момент  $t$

$y_t$  - выходной вектор

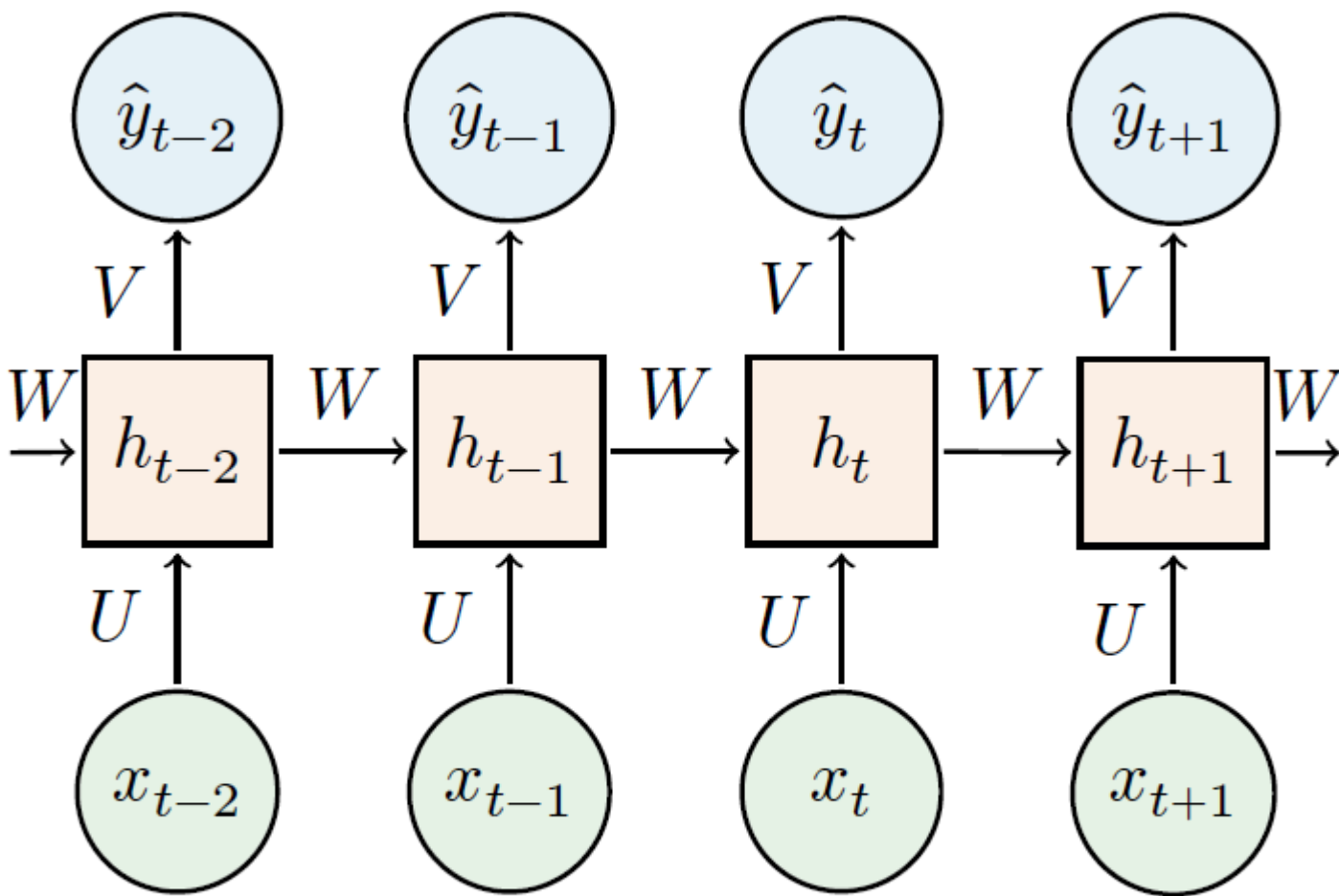
$$h_t = \sigma_h(Ux_t + Wh_{t-1})$$

$$y_t = \sigma_y(Vh_t)$$

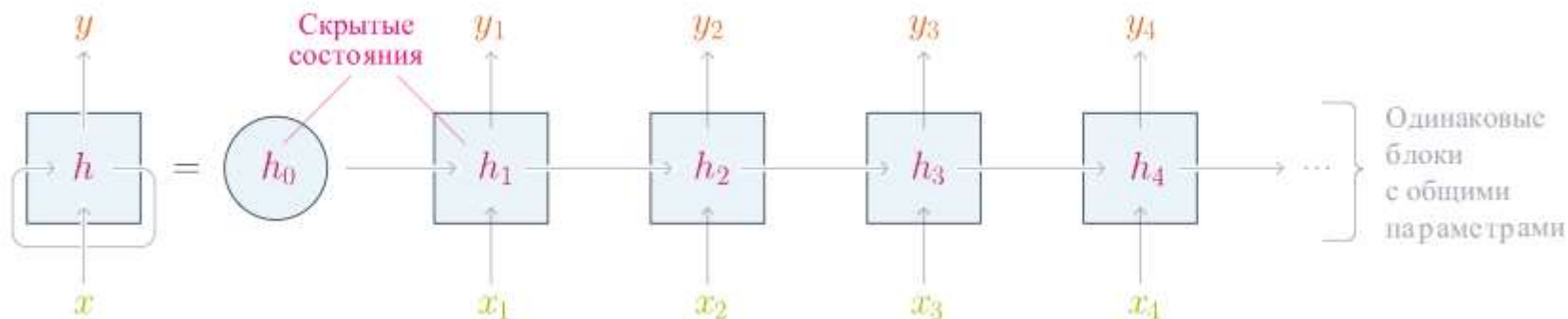




Рекуррентную сеть можно рассматривать, как несколько копий одной и той же сети, каждая из которых передает информацию последующей копии



$$h_t = f_h(Ux_t + Wh_{t-1} + b_h) \quad \hat{y}_t = f_y(Vh_t + b_y)$$



$h_t$  - вектор скрытого состояния в момент  $t$  – «внутренняя память» - для хранения информации о предыдущих элементах последовательности. На каждом дискретном шаге в сеть подаются данные, при этом происходит обновление скрытого состояния:

$$h_t = \tanh(h_{t-1}W_1 + x_tW_2)$$

По скрытому состоянию предсказывается выходной сигнал:

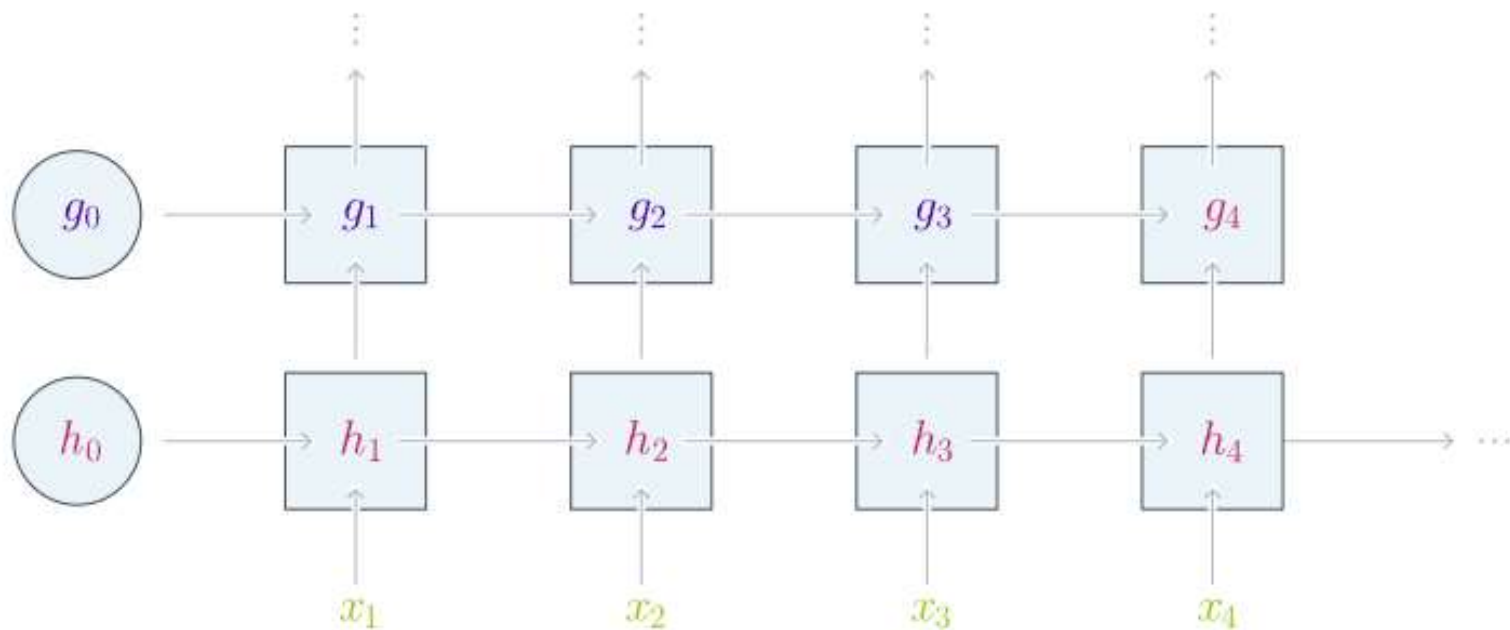
$$y_t = h_tW_3$$

Веса  $W_i$  одинаковых на всех итерациях, то есть очередные  $x_t$  и  $h_{t-1}$  подаются на вход одного и того же слоя, зацикленного на себе

Функция потерь – суммарное отклонение по всем выходным сигналам  $y_t$ :

$$\sum_{t=0}^T \mathcal{L}_t(U, V, W) \rightarrow \min_{U, V, W}$$

$$\mathcal{L}_t(U, V, W) = \mathcal{L}(y_t(U, V, W)) \text{ — потеря от предсказания } y_t$$

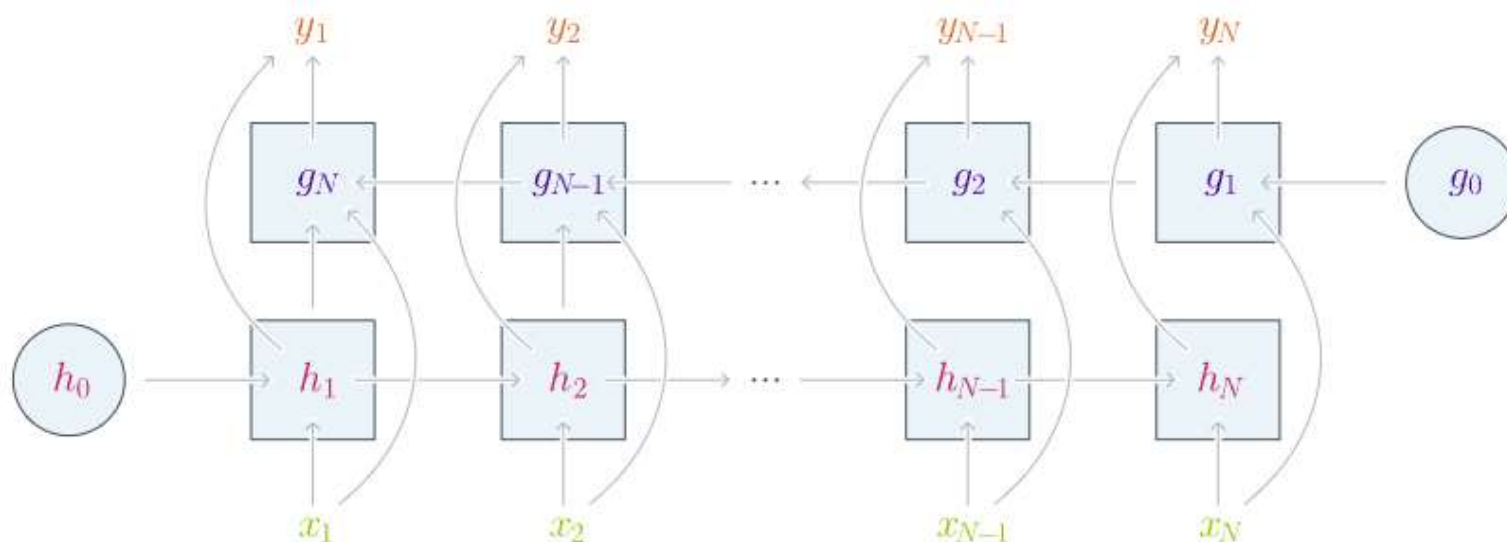


Глубокая *RNN* — несколько рекуррентных слоев:

- Первый слой *RNN* (первая сеть) принимает на вход исходную последовательность
- Второй слой *RNN* — принимает выходы первой сети
- Третий слой *RNN* — принимает выходы второй сети
- И так далее

# Bidirectional RNN

Стандартная RNN учитывает только предыдущий контекст. Но, например, слово в предложении связано не только с предыдущими, но и с последующими словами. Для таких случаев используется двунаправленная рекуррентная сеть (BRNN) – состоящая из прямой (элементы подаются от первого к последнему) и обратной (элементы подаются в обратном порядке) рекуррентных сетей.

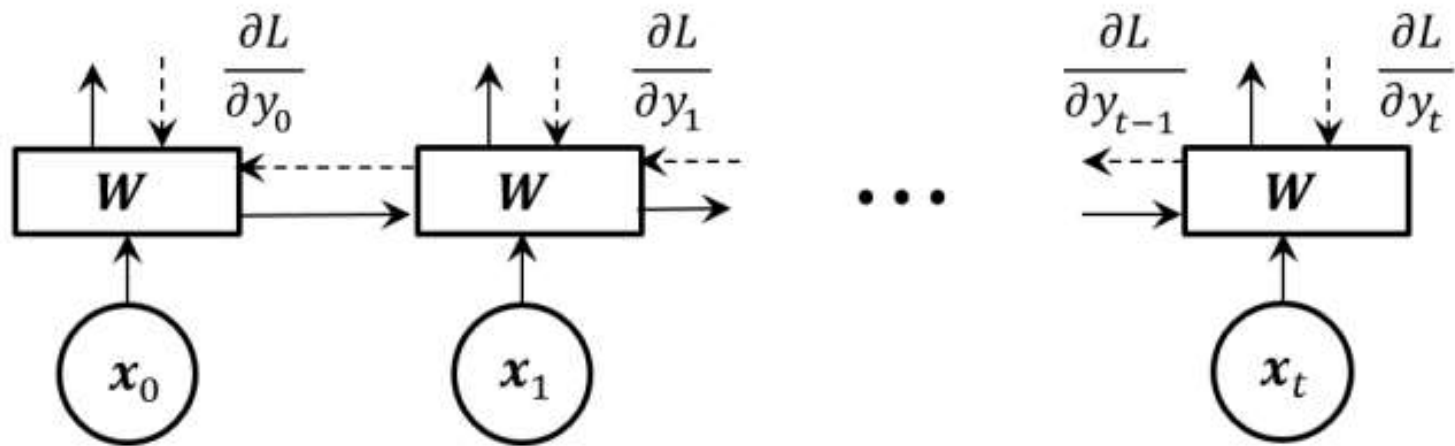


$$y_n = \underbrace{[h_n, g_n]}_{\text{Конкатенация}} w + b$$

Конкатенация

# Backpropagation Through Time (BPTT)

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \sum_{k=0}^t \left( \prod_{i=k+1}^t \frac{\partial h_i}{\partial h_{i-1}} \right) \frac{\partial h_k}{\partial W}$$



*Для предотвращения затухания/взрыва градиентов:*

*частные производные должны стремиться к 1:  $\frac{\partial h_i}{\partial h_{i-1}} \rightarrow 1$*

*Достигается за счет выбора функций активации*

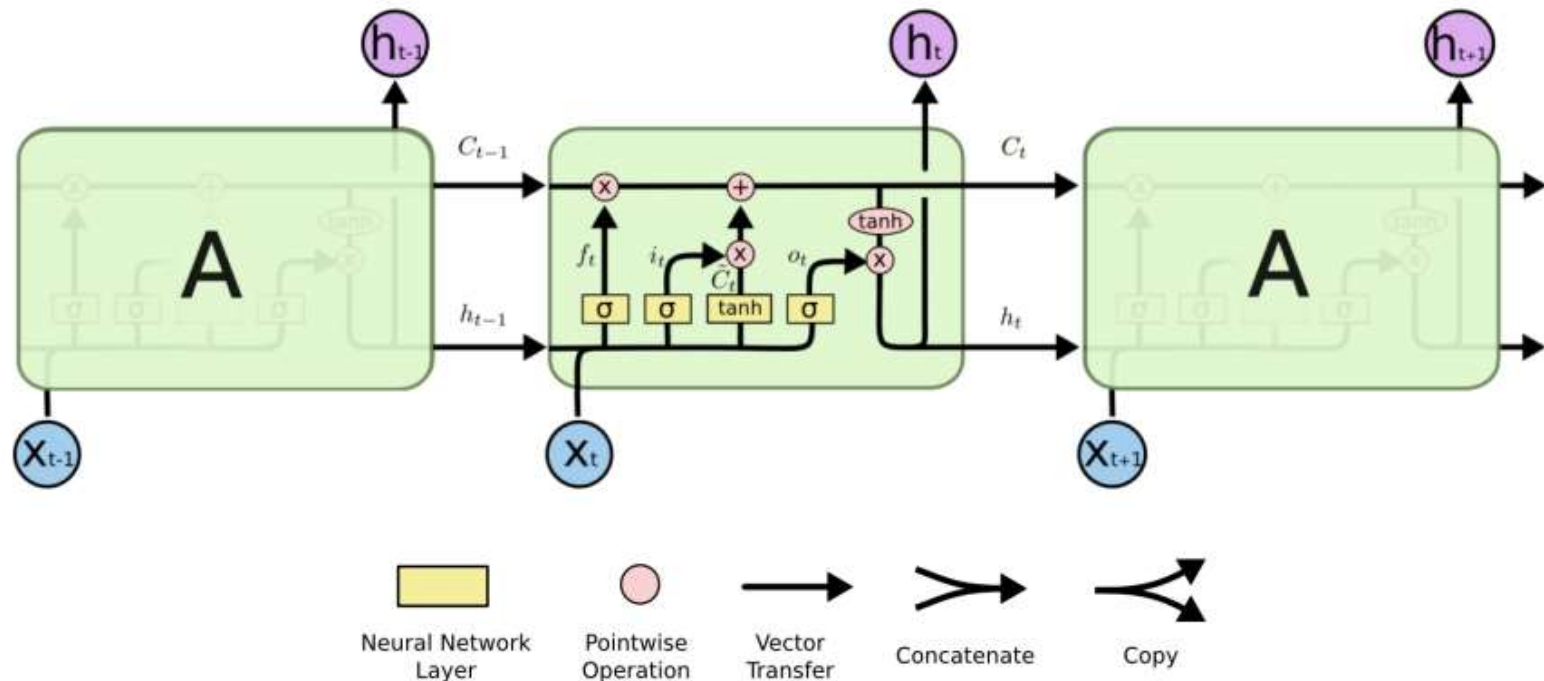
# Сети долгой кратковременной памяти

## Long Short-term memory (LSTM)

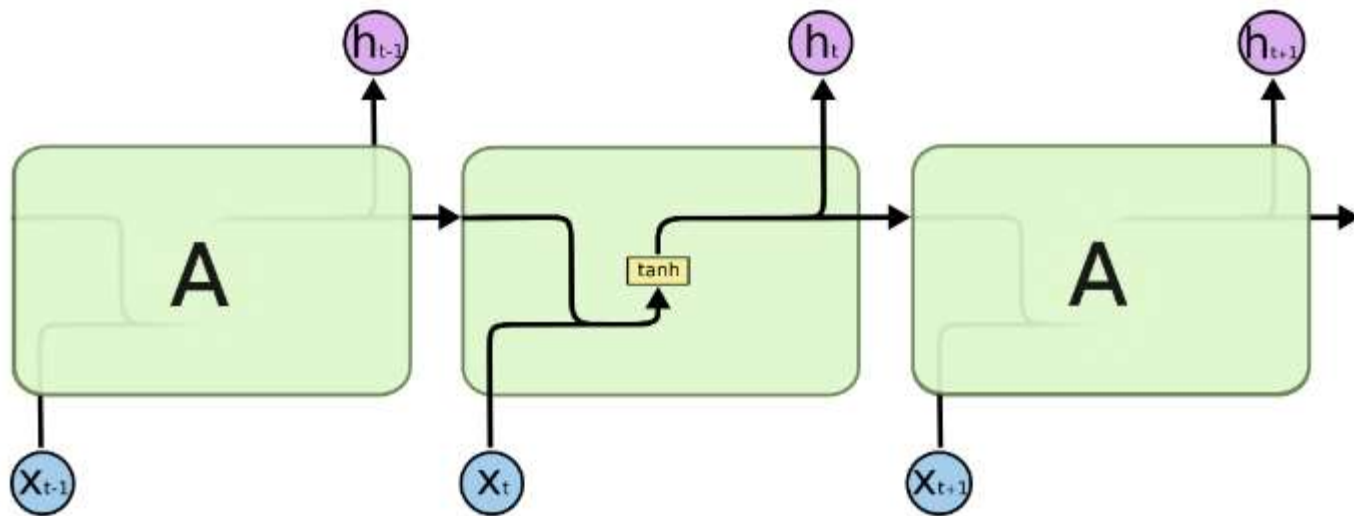
Мотивация LSTM: сеть должна сама долго помнить контекст, какой именно – сеть должна определить сама.

Вводится вектор  $C_t$  состояния сети в момент времени  $t$ .

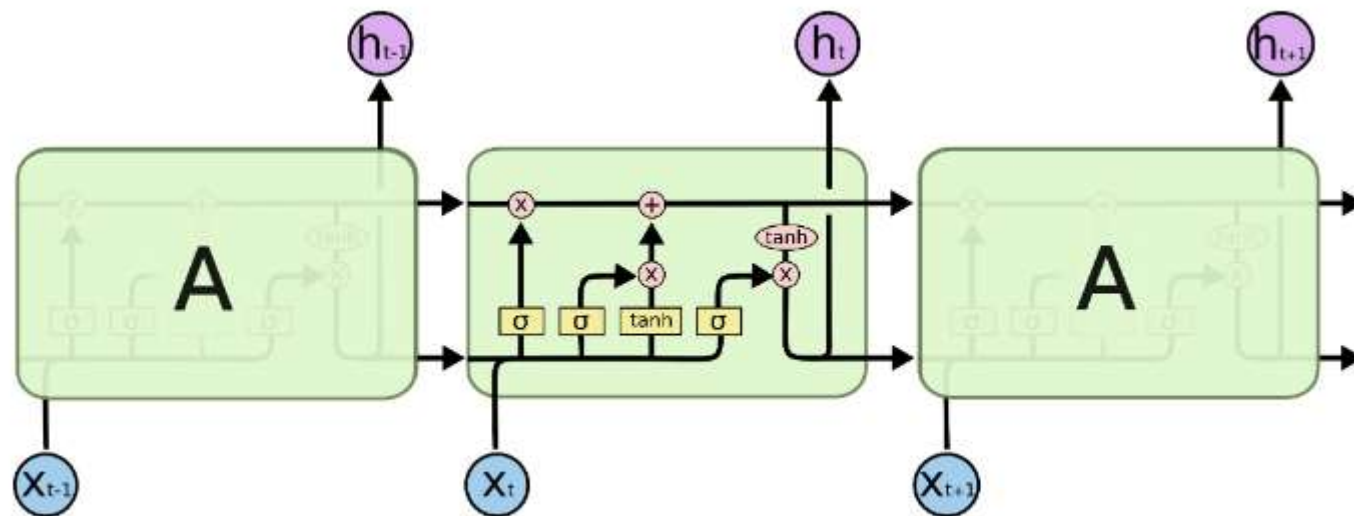
Разделяются векторы кратковременной и долговременной памяти.



- LSTM частично решает проблему исчезновения и взрыва градиентов в процессе обучения
- Все RNN можно представить в виде цепочки повторяющихся блоков (линейный слой + функция активации). В LSTM повторяющийся блок имеет более сложную структуру, состоящую не из одного, а из четырех слоев. Появляется понятие состояния блока (cell state,  $c_n$ )
- Cell state используется в роли внутренней (закрытой) информации LSTM-блока. Скрытое состояние  $h_t$  передается наружу, как в следующий блок так и в следующий слой или выход сети.
- LSTM может добавлять или удалять определенную информацию из cell state с помощью специальных механизмов – gates



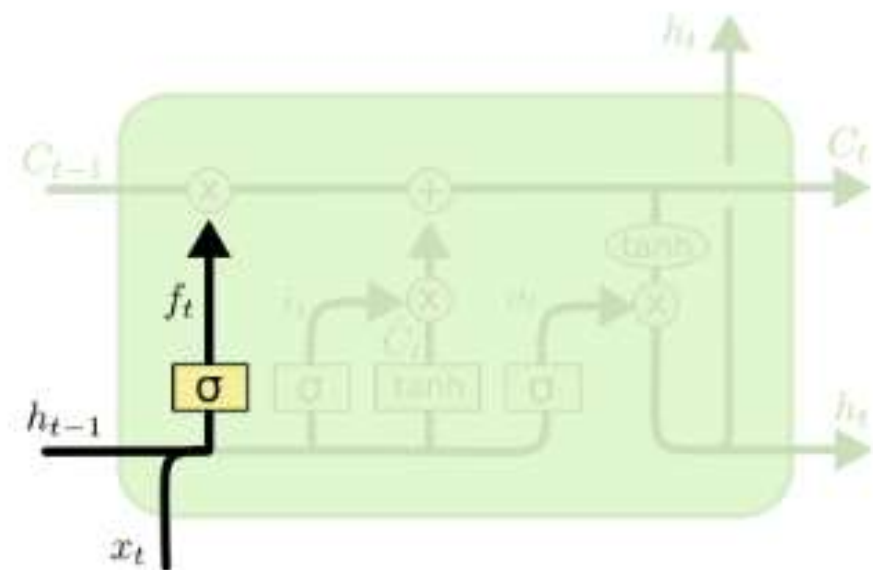
The repeating module in a standard RNN contains a single layer.



The repeating module in an LSTM contains four interacting layers.



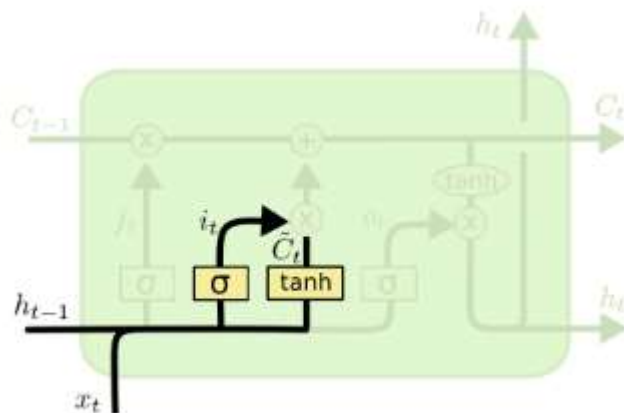
- Forget gate (вентиль забывания) – на основе предыдущего скрытого состояния  $h_{t-1}$  и нового входа  $x_t$  определить, какую долю информации из  $c_{t-1}$  (состояния предыдущего блока) стоит пропустить дальше, а какую забыть



$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

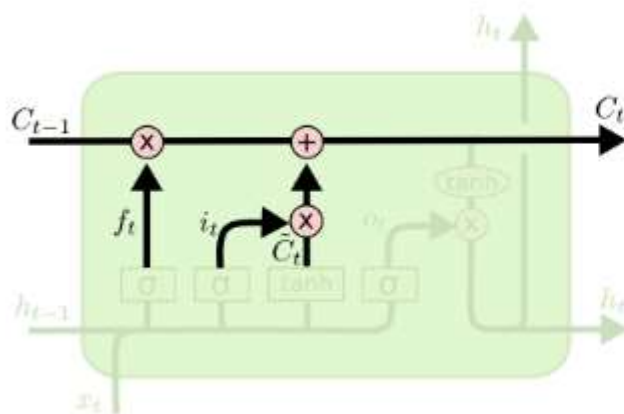
- Следующий шаг – определить что вносится в cell state

- Input gate (вентиль входного состояния) – решает, какие слагаемые надо «забыть», а какие внести



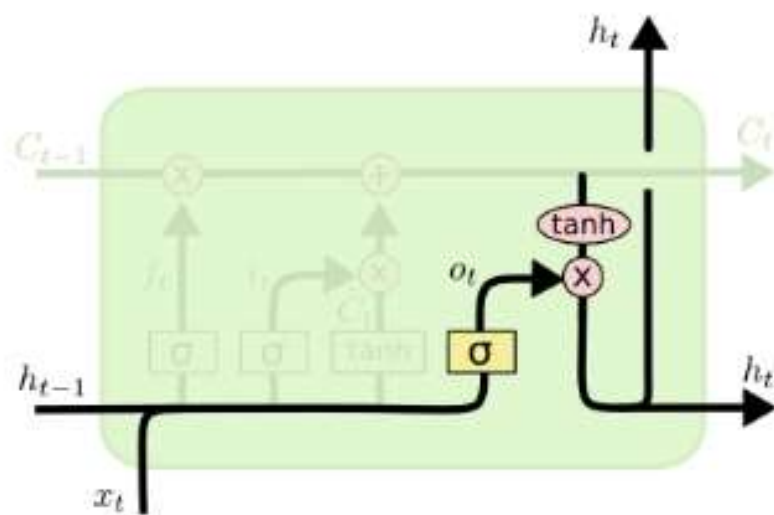
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Output gate (вентиль выходного состояния) – отвечает на вопрос о том, сколько информации из cell state следует отдавать на выход из LSTM-блока. Доля передаваемой информации  $o_t$



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

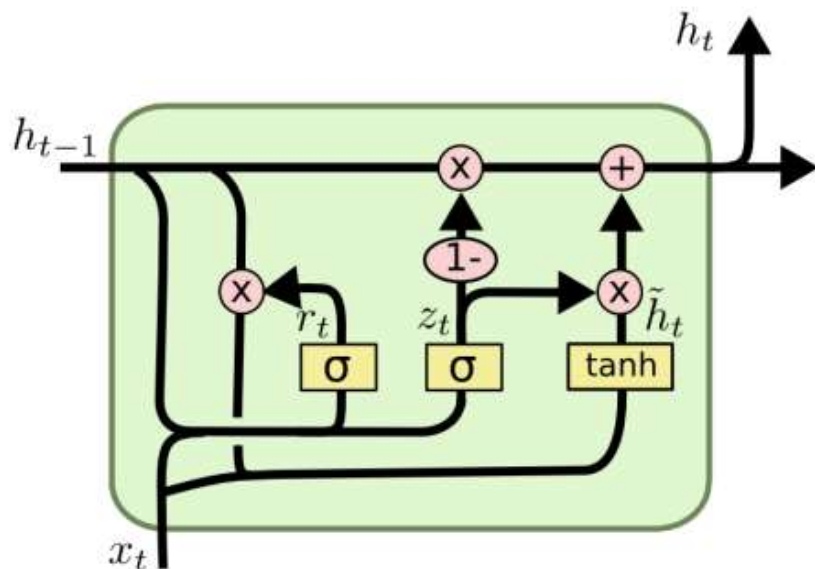
# Gated Recurrent Unit (GRU)

GRU является логическим упрощением LSTM с сохранением всех достоинств оригинальной архитектуры.

Используется только состояние  $h_t$ , вектор  $C_t$  не вводится.

Используется update gate – обобщение input gate и forget gate

Reset gate – решает, какую часть памяти нужно перенести дальше с прошлого шага



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z)$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Update gate – будем забывать только те значения, которые собираемся обновить (forget gate + input gate):

$$z_t = \sigma(h_{t-1}W_1^z + x_tW_2^z + b_z)$$

Reset gate – определяет, какую долю информации из  $h_{t-1}$  с прошлого шага надо «сбросить» и инициализировать заново:

$$r_t = \sigma(h_{t-1}W_1^r + x_tW_2^z + b_r)$$

Вычисляем потенциальное обновление для скрытого состояния:

$$\widetilde{h}_t = \tanh((r_t \odot h_{t-1})W_1^h + x_tW_2^h + b_h)$$

Решаем, что из старого забыть, а что из нового добавить:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \widetilde{h}_t$$

# An Empirical Exploration of Recurrent Network Architectures

- LSTM и GRU придуманы эмпирически, неясно являются ли эти архитектуры оптимальными
- В ходе исследования, проведенного Rafal Jozefowicz, Ilya Sutskever (Google Inc.) и Wojciech Zaremba (NY University, Facebook) была проведена оценка более десяти тысяч различных архитектур RNN и LSTM. Были найдены некоторые архитектуры (ячейки), которые превосходят и LSTM и GRU на некоторых, но не на всех задачах

Arch.	5M-tst	10M-v	20M-v	20M-tst
Tanh	4.811	4.729	4.635	4.582 (97.7)
LSTM	4.699	4.511	4.437	4.399 (81.4)
LSTM-f	4.785	4.752	4.658	4.606 (100.8)
LSTM-i	4.755	4.558	4.480	4.444 (85.1)
LSTM-o	4.708	4.496	4.447	4.411 (82.3)
LSTM-b	4.698	4.437	4.423	<b>4.380 (79.83)</b>
GRU	4.684	4.554	4.559	4.519 (91.7)
MUT1	4.699	4.605	4.594	4.550 (94.6)
MUT2	4.707	4.539	4.538	4.503 (90.2)
MUT3	4.692	4.523	4.530	4.494 (89.47)

MUT1:

$$\begin{aligned}
 z &= \text{sigm}(W_{xz}x_t + b_z) \\
 r &= \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\
 h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + \tanh(x_t) + b_h) \odot z \\
 &\quad + h_t \odot (1 - z)
 \end{aligned}$$

MUT2:

$$\begin{aligned}
 z &= \text{sigm}(W_{xz}x_t + W_{hz}h_t + b_z) \\
 r &= \text{sigm}(x_t + W_{hr}h_t + b_r) \\
 h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\
 &\quad + h_t \odot (1 - z)
 \end{aligned}$$

MUT3:

$$\begin{aligned}
 z &= \text{sigm}(W_{xz}x_t + W_{hz} \tanh(h_t) + b_z) \\
 r &= \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \\
 h_{t+1} &= \tanh(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z \\
 &\quad + h_t \odot (1 - z)
 \end{aligned}$$

## Достоинства RNN архитектур:

- Способны учитывать контекст и последовательность данных
- Способны обрабатывать ввод любой длины
- Эффективны на задачах с временными зависимостями

## Недостатки RNN архитектур:

- Вычислительная сложность, взрыв градиента
- Ограничения на долгосрочное запоминание, сложно получить информацию со всей последовательности

# Модели внимания и трансформеры

 Toplyne



# Transformers

**RECURRENT NEURAL  
NETWORKS**



- Текст (документ) – последовательность токенов
- Токенизация – представление текста в виде последовательности
- Токен – атомарный элемент последовательности (слово, символ, морфема (значимая часть слова) и т.д.)

*Машинное обучение – это весело*



Машинное обучение – это весело

text → features → ML model →  $P(y|x)$



Как представить текст  
в виде, который могла  
бы понять модель?

# Представление текста

Необходимо кодировать текстовые данные – преобразовать их в вектор, прежде чем подавать на вход нейронной сети.

Существует два подхода:

1. Векторизовать текст целиком, превращая его в один вектор
2. Векторизовать отдельные структурные единицы, превращая текст в последовательность векторов (embedding)

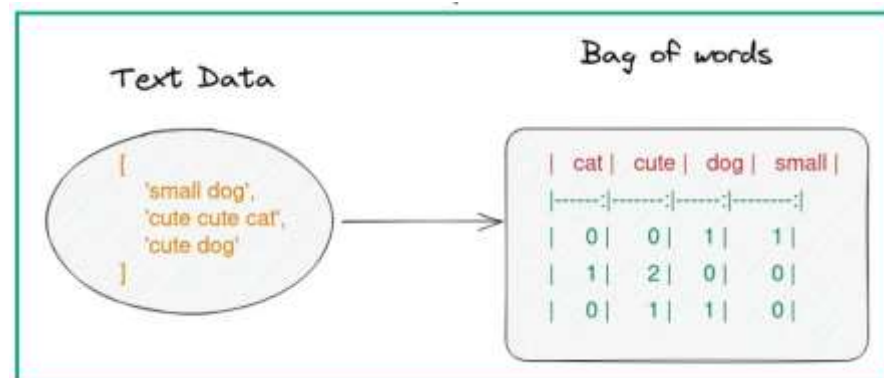
- Bag-of-Words (мешок слов) – текст представляется в виде вектора частот встречаемости каждого токена, кроме элементов заранее заданного списка «стоп-слов», в которые обычно включают самые вездесущие токены: личные местоимения, артикли и так далее
- Представление мешка слов – таблица с числами, в которой столбцы – уникальные слова, а в ячейках находится число вхождений слова в документ. В каждой строке получается набор чисел (вектор), характеризующий состав документа
- Не учитывается порядок слов – два предложения могут называться одинаковыми, если содержат один и тот же набор слов

## The Bag of Words Representation

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



it 6  
I 5  
the 4  
to 3  
and 3  
seen 2  
yet 1  
would 1  
whimsical 1  
times 1  
sweet 1  
satirical 1  
adventure 1  
genre 1  
fairy 1  
humor 1  
have 1  
great 1  
...



*Заходит гелий в бар и заказывает пиво. Бармен: "Извините, но мы не обслуживаем благородных" Гелий не реагирует.*

заходит	1
гелий	2
в	1
бар	1
и	1
заказывает	1
пиво	1
бармен	1
извините	1
но	1
мы	1
не	2
обслуживаем	1
благородных	1
реагирует	1
кот	0
собака	0
мышь	0

- TF-IDF (Term Frequency - Inverse Document Frequency) – статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документа коллекции
- TF (Term Frequency, частота термина) – обозначает, насколько часто определенное слово появляется в данном документе (частота вхождения токена в документ).
- TF измеряет важность слова в контексте отдельного документа

$$TF(t, d) = \frac{n_t}{\sum_k n_k}$$

где  $n_t$  – число вхождений токена  $t$  в документ,  
а в знаменателе стоит общее число слов в данном документе

- IDF (Inverse document frequency) – измеряет, насколько уникально слово является по всей коллекции документов. Слова, которые в большинстве документов, имеют низкое IDF, так как не вносят большой информационной ценности

$$IDF(t, D) = \log \frac{|D|}{|\{d_i \in D | t \in d_i\}|}$$

где  $\{d_i \in D | t \in d_i\}$  – число документов в текстовой коллекции  $D$ , в которых встречается слово  $t$

- Этот множитель штрафует компоненты, отвечающие слишком распространенным токенам и повышает вес специфических для отдельных текстов (и, вероятно, информативных) слов

Мера TF-IDF часто используется в задача анализа текстов и информационного поиска

Преимущества TD-IDF:

- Учет важности слов – учитывает как частоту слова в документе, так и его общую редкость по всей коллекции. Выделяет ключевые слова, которые часто встречаются в данном документе, но не слишком распространены в остальных
- Устранение шума – слова, которые встречаются в большинстве документов - «стоп-слова» - имеют низкий общий вес TF-IDF

Недостатки TF-IDF:

- Отсутствие семантической информации – не учитывает семантические связи между словами
- Чувствительность к длине документа – длинные документы могут иметь более высокие значения TF, даже если ключевые слова встречаются реже

# Информативные векторные представления – эмбединги (*words embeddings* – для слов)

- Мотивация использовать контекст:

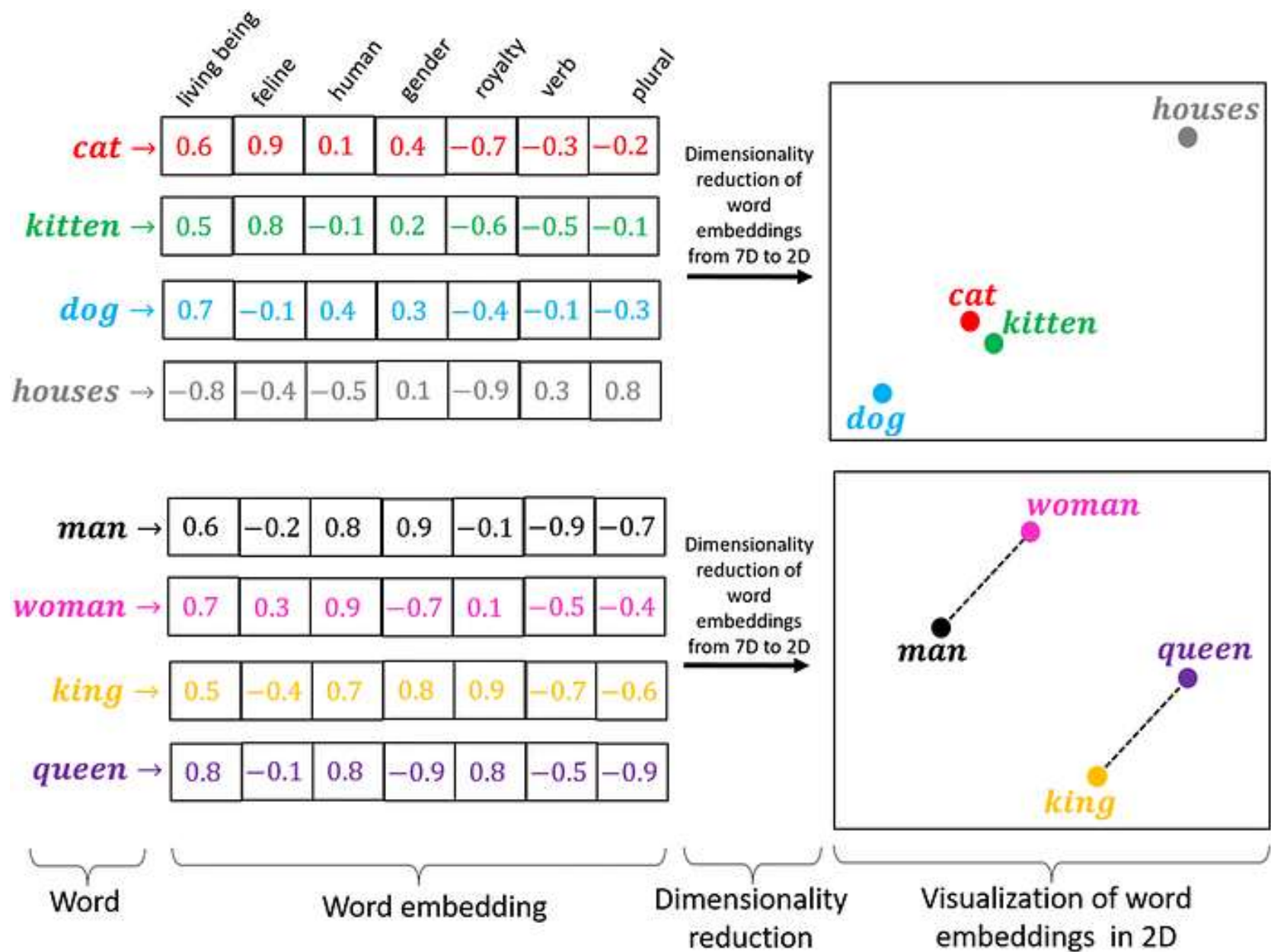
Допустим, что одно и то же слово будет представлено одним и тем же вектором во всех текстах и в любых позициях. Как заключить в векторе его смысл (информацию)?

Использование контекста позволяет понять смысл слова по словам, встречающимся рядом с ним

*«Я не успел к началу лекции, потому что опоздал на...»*

1. *Автобус – 0.55*
2. *Трамвай – 0.35*
3. *Такси – 0.05*
4. *Велосипед – 0.04*
5. *Самолет – 0.01*
6. *...*



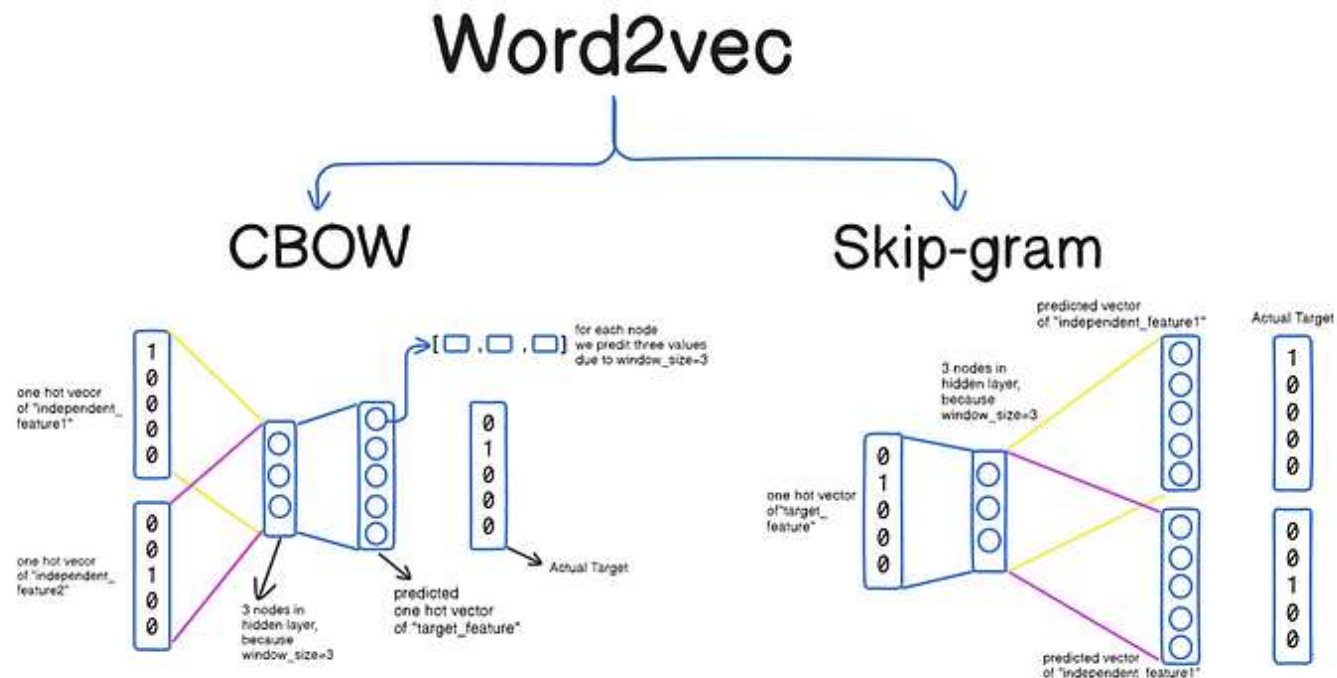


# Word2vec

Реализация подхода с использованием контекста – word2vec

Предложен Томашем Миколовым в 2013 году в статье  
«Efficient Estimation of Word Representations in Vector Space»

Основная идея – упаковать информацию о контексте слов в вектора.  
Будем обучать вектора, пытаюсь предсказать контекст, в котором встречается слово

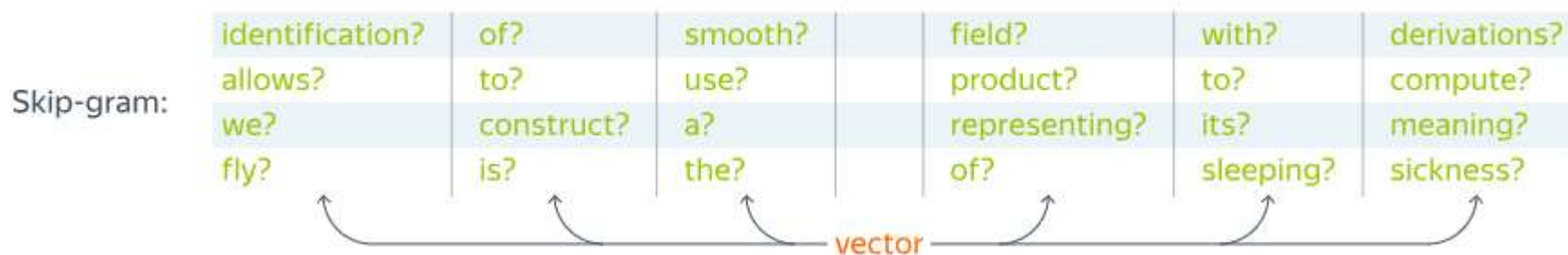


Предложено две стратегии – CBOW и Skip-gram

1. CBOW (continuous bag-of-words) – модель учится предсказывать данное (центральное) слово по контексту. Например, по двум словам перед данным и по двум словам после него.



2. Skip-gram – модель учится предсказывать контекст.  
Например, каждого из двух соседей слева и справа.



- Для каждого слова w обучается два эмбединга:  $v_u$  и  $v_w$ . Первый используется, когда w является центральным, второе – когда оно рассматривается как часть контекста.

В модели CBOW при фиксированном контексте «*context*» вычисляются логиты

$$\text{logits}_u = \left\langle \sum_{w \in \text{context}} v_w, v_u \right\rangle$$

После чего «вероятности» всевозможных слов и быть центральным словом для «*context*» вычисляются как  $\text{softmax}(\text{logits})$

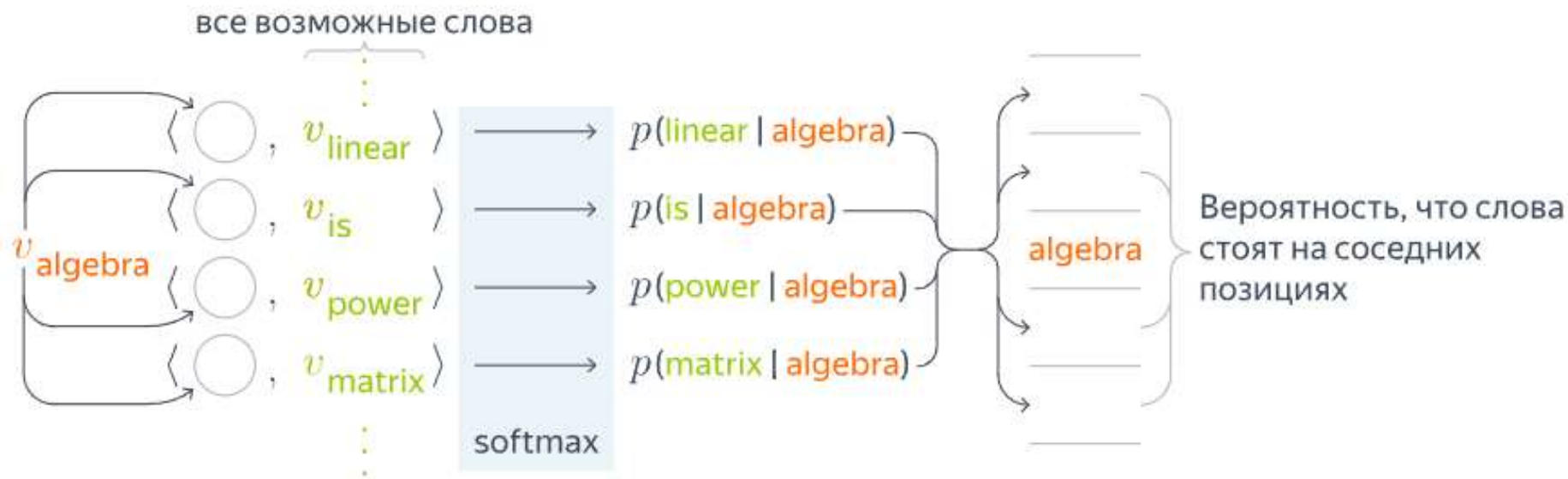
Модель учится с помощью SGD на кросс-энтропию полученного распределения с истинным распределением центральных слов

CBOW:



В модели Skip-gram по данному центральному слову  $u$  для каждой позиции контекста «context» независимо предсказывается распределение вероятностей. В качестве функции потерь выступает сумма кросс-энтропий распределений слов контекста с их истинными распределениями

Skip-gram:



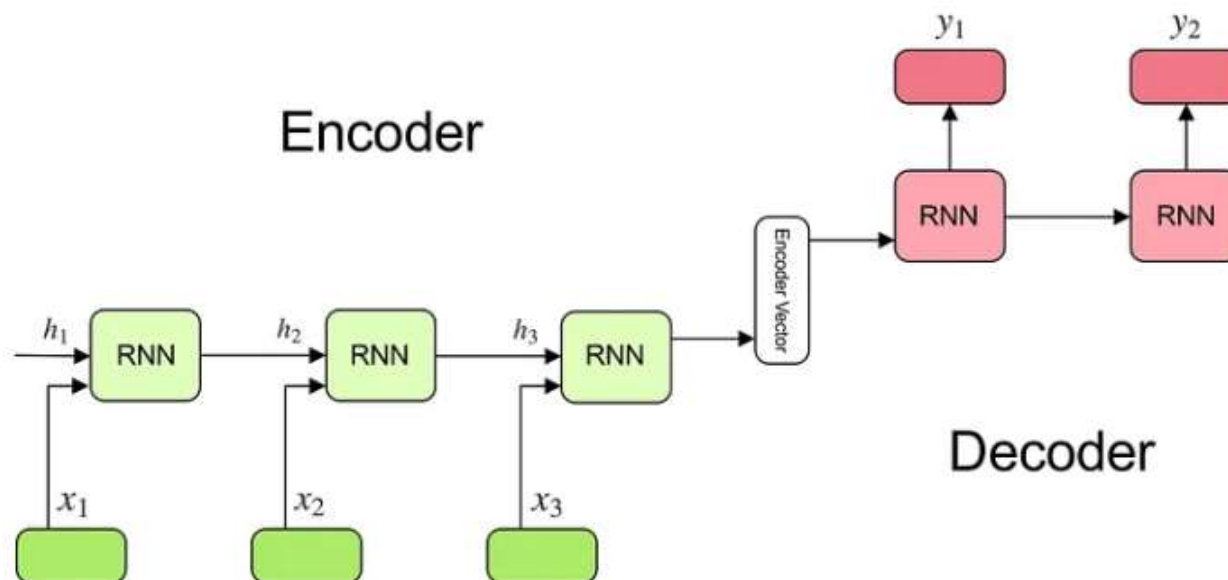
- Примеры. Возьмём несколько слов и посмотрим, как выглядят топ-10 слов, ближайших к ним в пространстве эмбедингов (обученных на одном из датасетов Quora Questions с помощью word2vec):
  1. **quantum**: electrodynamics, computation, relativity, theory, equations, theoretical, particle, mathematical, mechanics, physics;
  2. **personality**: personalities, traits, character, persona, temperament, demeanor, narcissistic, trait, antisocial, charisma;
  3. **triangle**: triangles, equilateral, isosceles, rectangle, circle, choke, quadrilateral, hypotenuse, bordered, polygon;
  4. **art**: arts, museum, paintings, painting, gallery, sculpture, photography, contemporary, exhibition, artist.
- Размерность эмбединга в каждой из архитектур — это гиперпараметр и подбирается эмпирически. В оригинальной статье предлагается взять размерность эмбединга 300. Полученные представления центральных слов могут дальше использоваться в качестве эмбедингов слов, которые сохраняют семантическую связь слов друг с другом.

# Sequence-to-sequence

Рекуррентная сеть для синтеза последовательностей (seq2sec) – архитектура, предназначенная для задач, в которых требуется преобразование одной последовательности в другую (машинный перевод, суммаризация текста, генерация описаний и т.д.). Состоит из энкодера и декодера.

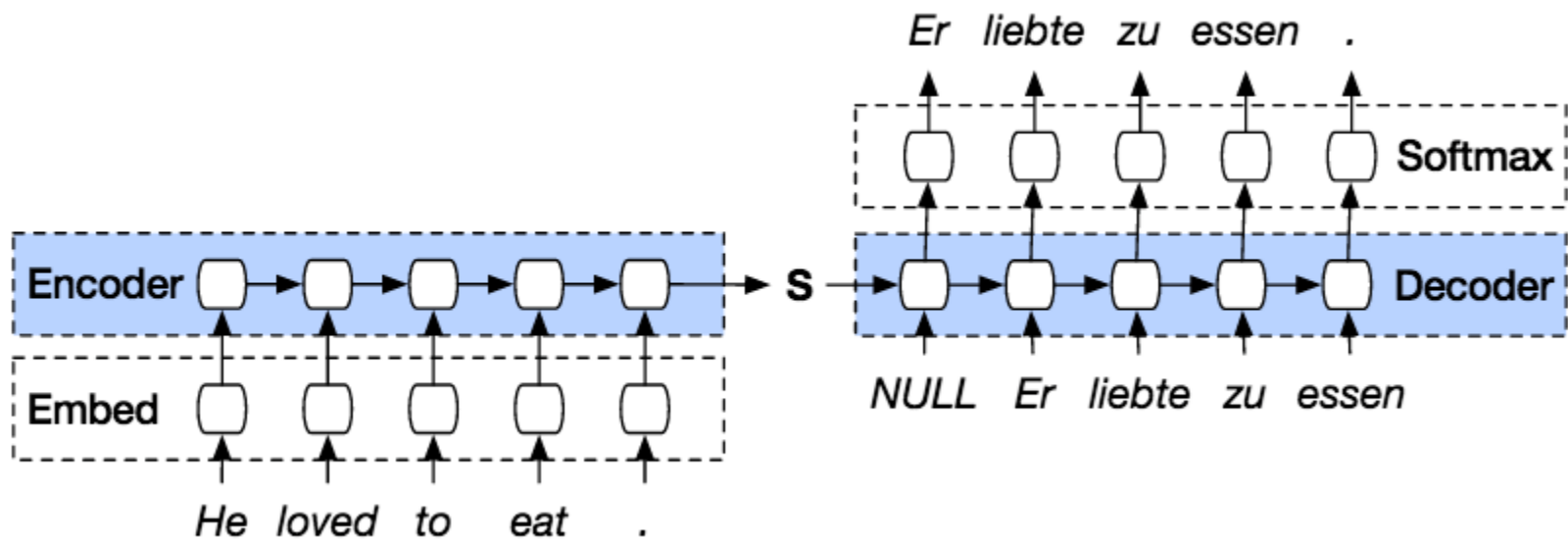
Энкодер (encoder) – кодирование информации об исходной последовательности в контекстный вектор (context vector, encoded vector)

Декодер (decoder) – преобразование закодированной энкодером информации в новую последовательность



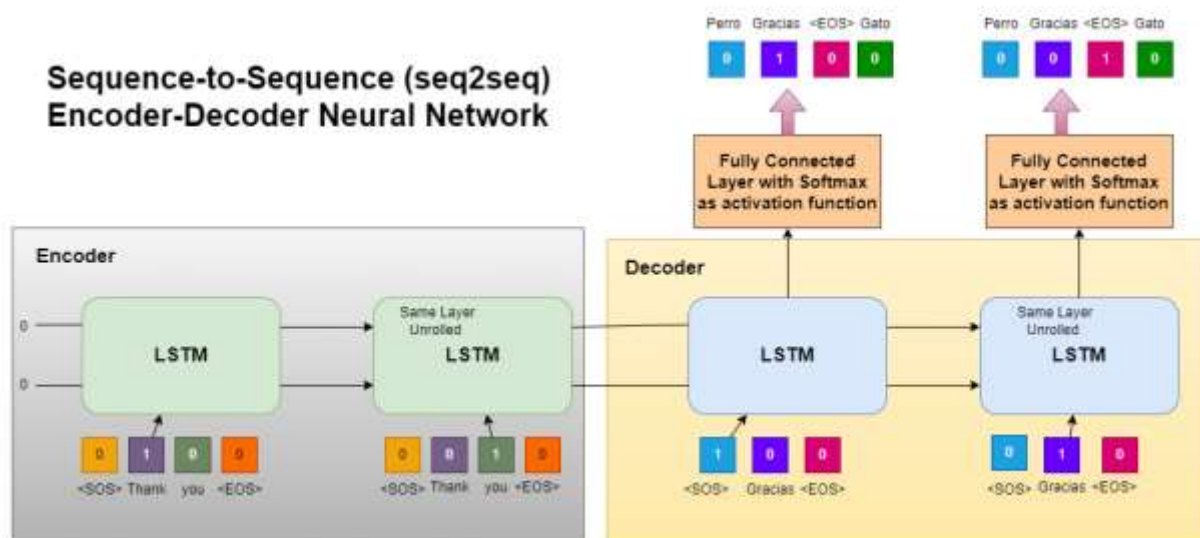


- Входные текстовые данные преобразуются в числовые представления с помощью эмбеддинга, который преобразует каждый токен во входной последовательности в вектор фиксированной размерности



- Encoder – преобразует входную последовательность в контекстный вектор, содержащий обобщённое представление всей входной информации
- Основу энкодера RNN, обычно реализованные с использованием LSTM или GRU. Эти сети обрабатывают входную последовательность пошагово:
  1. На каждом шаге RNN принимает эмбединговое представление текущего токена и скрытое состояние от предыдущего шага
  2. Выход каждого шага включает новое скрытое состояние, которое передаётся на следующий шаг вместе со следующим токеном
- В конце последовательности RNN генерирует контекстный вектор, который является финальным скрытым состоянием. Этот вектор обобщает всю информацию из входной последовательности и передаётся в декодер для дальнейшей генерации выходной последовательности. Контекстный вектор — это своего рода сжатая версия входной последовательности, включающая в себя её смысл
- Для улучшения качества представления входной последовательности часто используются двунаправленные RNN. В этом случае два RNN работают параллельно: один — слева направо, другой — справа налево. Их состояния объединяются на каждом шаге, что позволяет учитывать как предшествующие, так и последующие слова для каждого токена в последовательности

- Decoder – декодер генерирует данные на основе предыдущих предсказаний и контекстного вектора, предоставленного энкодером
- Подобно энкодеру, декодер принимает токены, которые сначала преобразуются в числовые представления с помощью эмбединга. На вход подаются не только реальные данные, но и предсказанные токены на предыдущих шагах. Реализован с использованием LSTM или GRU, на каждом шаге принимает:
  1. Контекстный вектор от энкодера
  2. Предыдущий предсказанный токен (начальный токен для первого шага)
  3. Скрытое состояние от предыдущего шага декодера. Этот выходной вектор затем преобразуется в вероятности через слой Softmax, который указывает на вероятность каждого возможного токена в выходной последовательности



# Attention Is All You Need

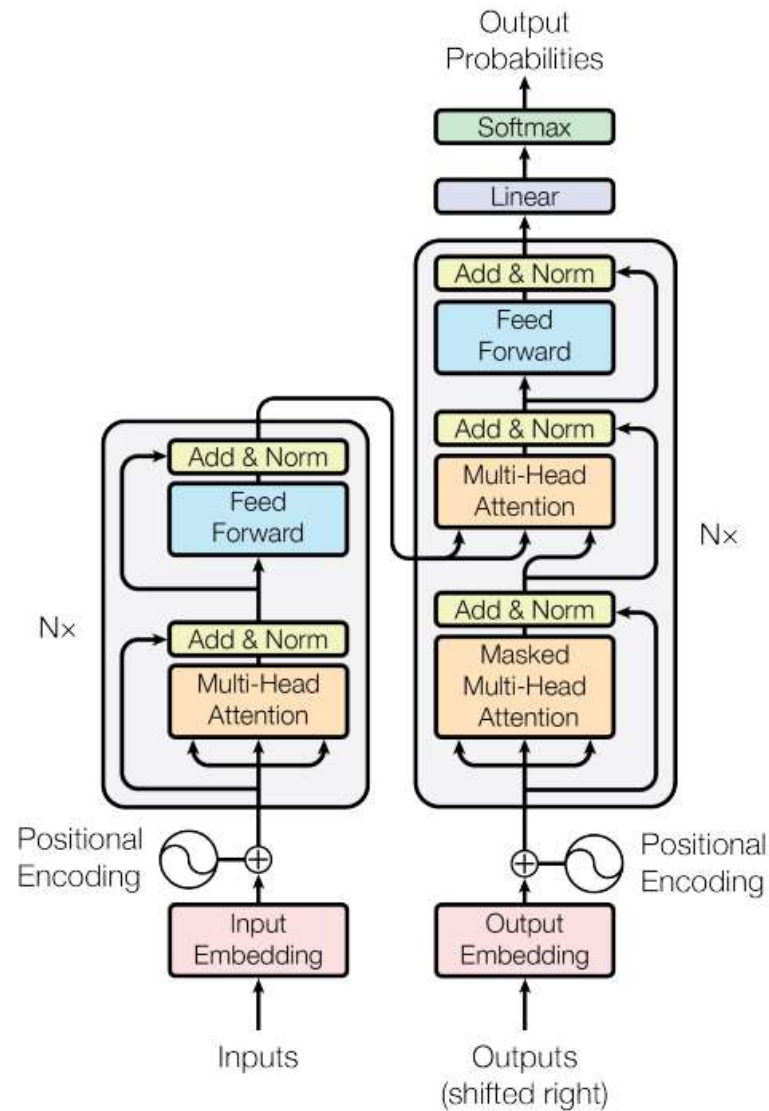
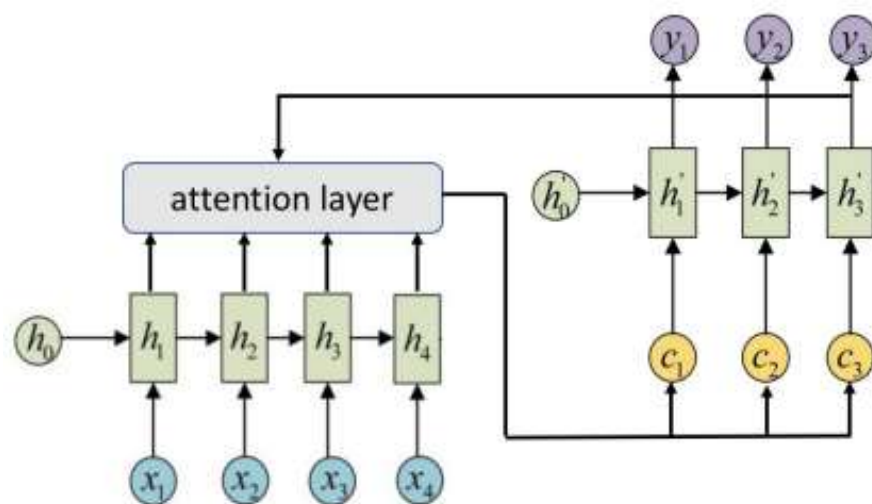
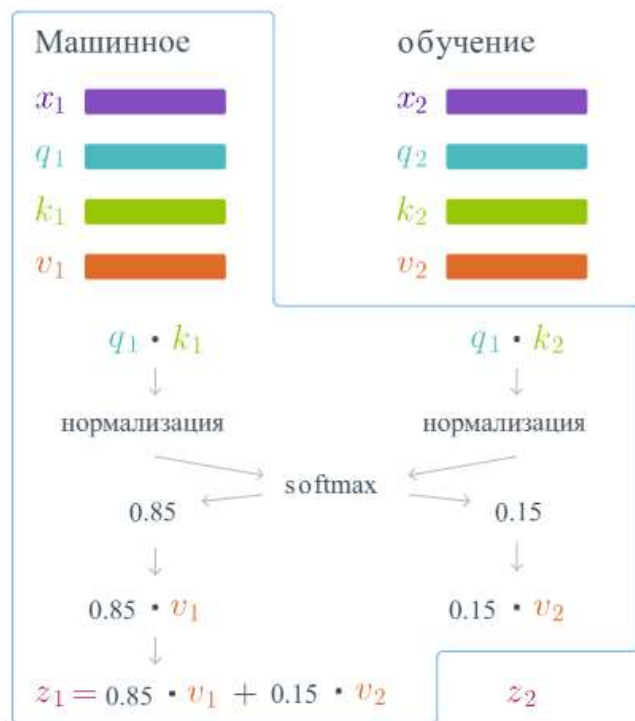


Figure 1: The Transformer - model architecture.

# Механизм внимания (attention)

- Предоставление декодеру информации обо всех токенах исходного предложения на каждом шаге генерации для указания, на какое слово обратить внимание. На каждом шаге декодера считаются attention scores, получаем N значений, указывающих, насколько каждый из токенов с номерами (0 ... n) из исходного состояния важен для генерации токена i из выходной последовательности.



# Механизм внутреннего внимания (self-attention)

- Используется, чтобы посмотреть на другие слова во входной последовательности во время кодирования конкретного слова
- Из исходного вектора (эмбединга каждого токена) формируется три вектора – Query (запрос), Key (ключ), Value (значение). Они получаются с помощью умножения входного вектора на матрицы  $W_Q$ ,  $W_K$ ,  $W_V$ , веса которых учатся вместе со всеми остальными параметрами модели с помощью обратного распространения ошибки
- Выделение трех абстракций нужно, чтобы разграничить эмбединги, задающие направление внимания (query, key) и смысловую часть токена (value). Вектор query задает модальность «начальной точки» механизма внутреннего внимания (от какого токена направлено внимание), вектор key – модальность «конечной точки» (к какому токеноу направлено внимание). Один и тот же токен может выступать как «начальной», так и «конечной» точкой направления внимания – self-attention вычисляется между всеми токенами в выбранном фрагменте текста

# Механизм внутреннего внимания (self-attention)

- Каждый токен фиксируется по очереди – он становится query и просчитывается его степень связанности со всеми оставшимися токенами. Для этого поочередно key-вектора всех токенов скалярно умножаются на query-вектор текущего токена. Полученные числа показывают, насколько важны остальные токены при кодировании query токена в конкретной позиции
- Полученные числа нормализуются и пропускаются через Softmax для получения распределения вероятностей. Затем подсчитывается взвешенная сумма value векторов, где в качестве весов используются полученные на предыдущем шаге вероятности. Полученный вектор и будет выходом внутреннего слоя внимания для одного токена
- На практике используются матричные вычисления вместо вычисления значений  $Q, K, V$  для отдельных токенов
- Обычно параллельно используется несколько self-attention блоков – «Multi-head self-attention»

# Механизм внутреннего внимания (self-attention)

$$\text{self\_attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Q, K, V – матрицы запросов, ключей и значений (query, key, values)

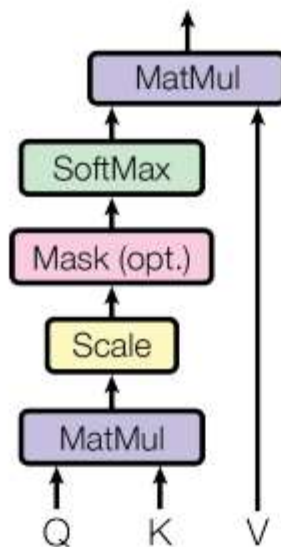
$\sqrt{d_k}$  - нормировочная константа – корень из размерности ключей и значений

The diagram illustrates the self-attention calculation in matrix form. It shows the formula:  $\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$ . The matrices are represented as grids of colored squares: Q is a 2x3 purple grid, K<sup>T</sup> is a 3x2 orange grid, V is a 2x3 blue grid, and Z is a 2x3 pink grid. The result Z is shown below the main equation.

The self-attention calculation in matrix form



### Scaled Dot-Product Attention



### Multi-Head Attention

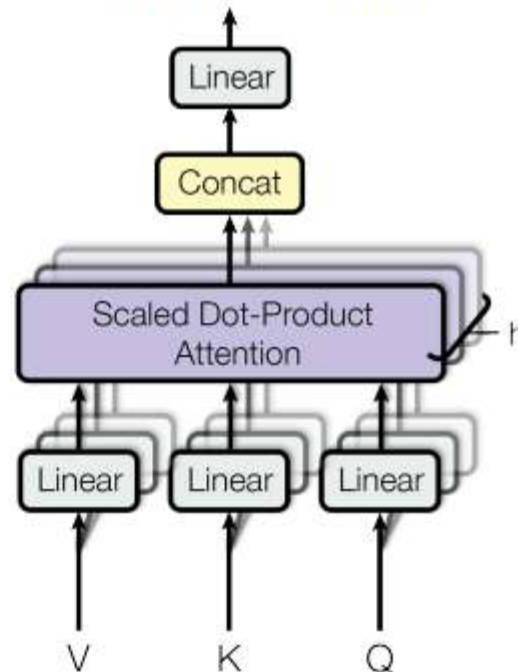
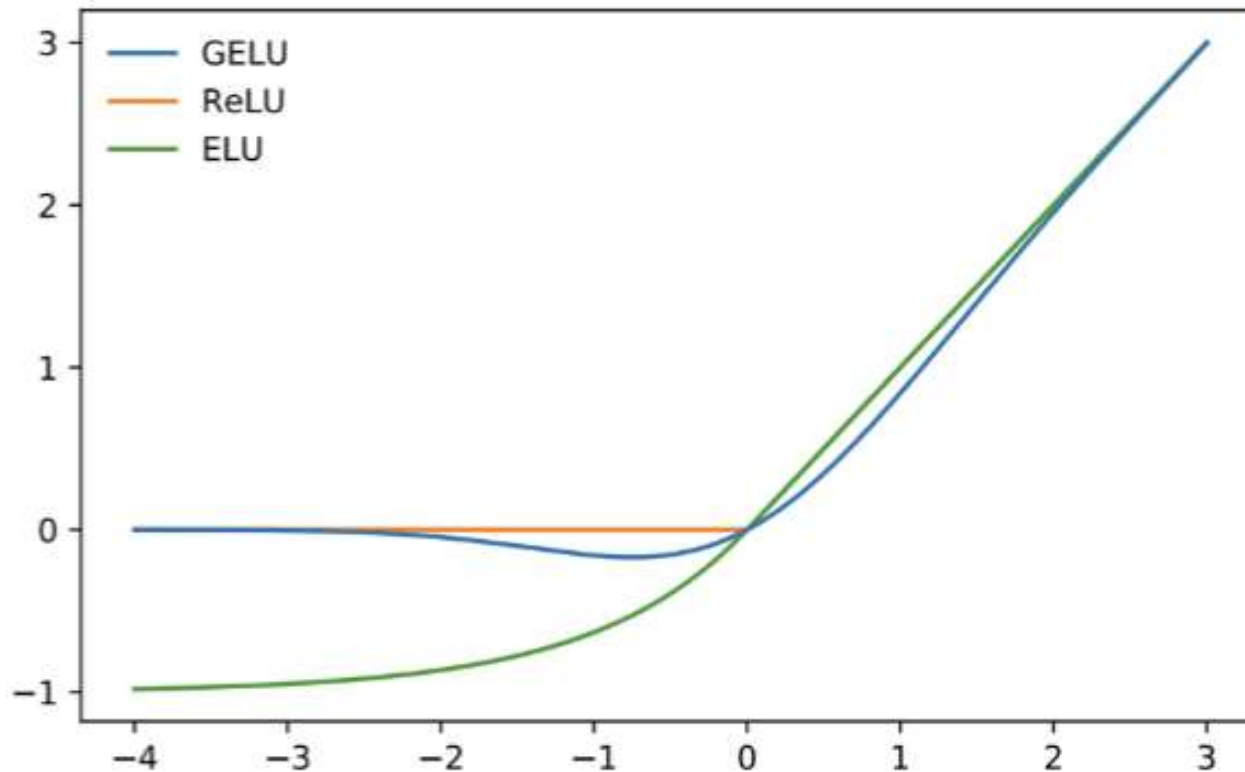


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

# Полносвязный слой

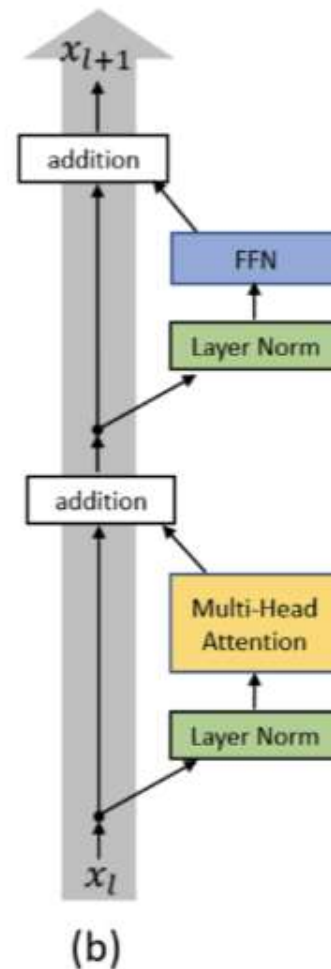
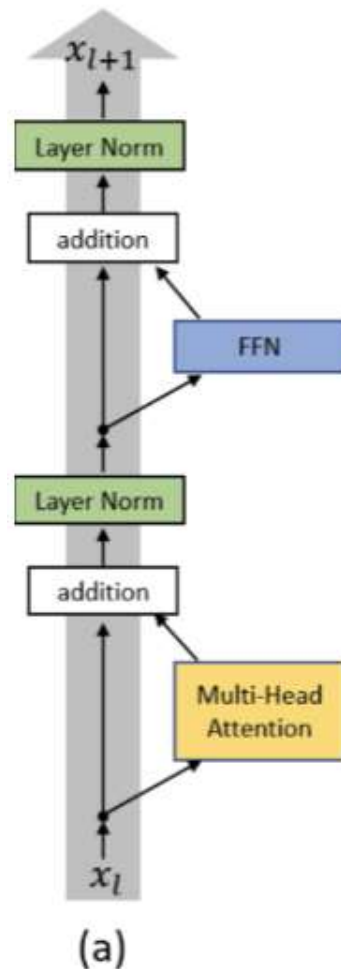
- Feed-forward network (FFN) – два полносвязных слоя, применяемых независимо к каждому элементу входной последовательности

$$FFN(x) = GELU(xW_1 + b_1)W_2 + b_2$$



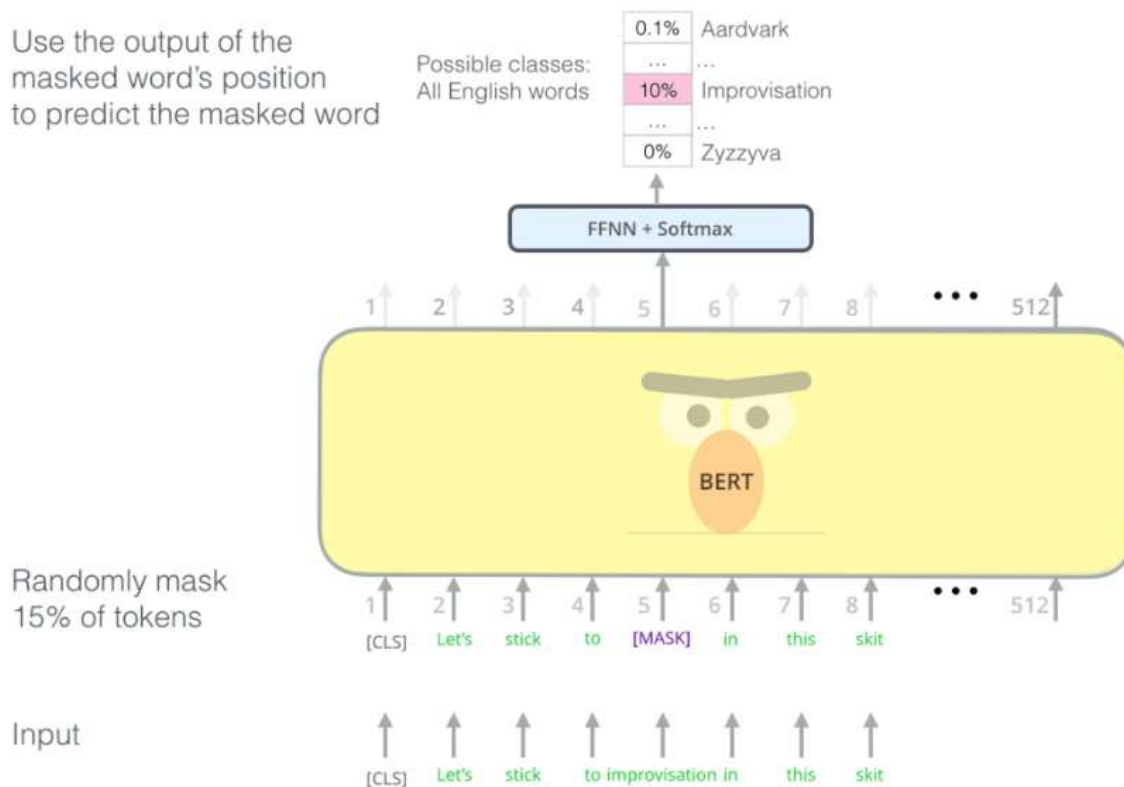
# Нормализация

- Layer normalization – нормализация применяется после остаточной связи – PostLN (a) либо ко входу residual-ветки – PreLN (b)



# BERT

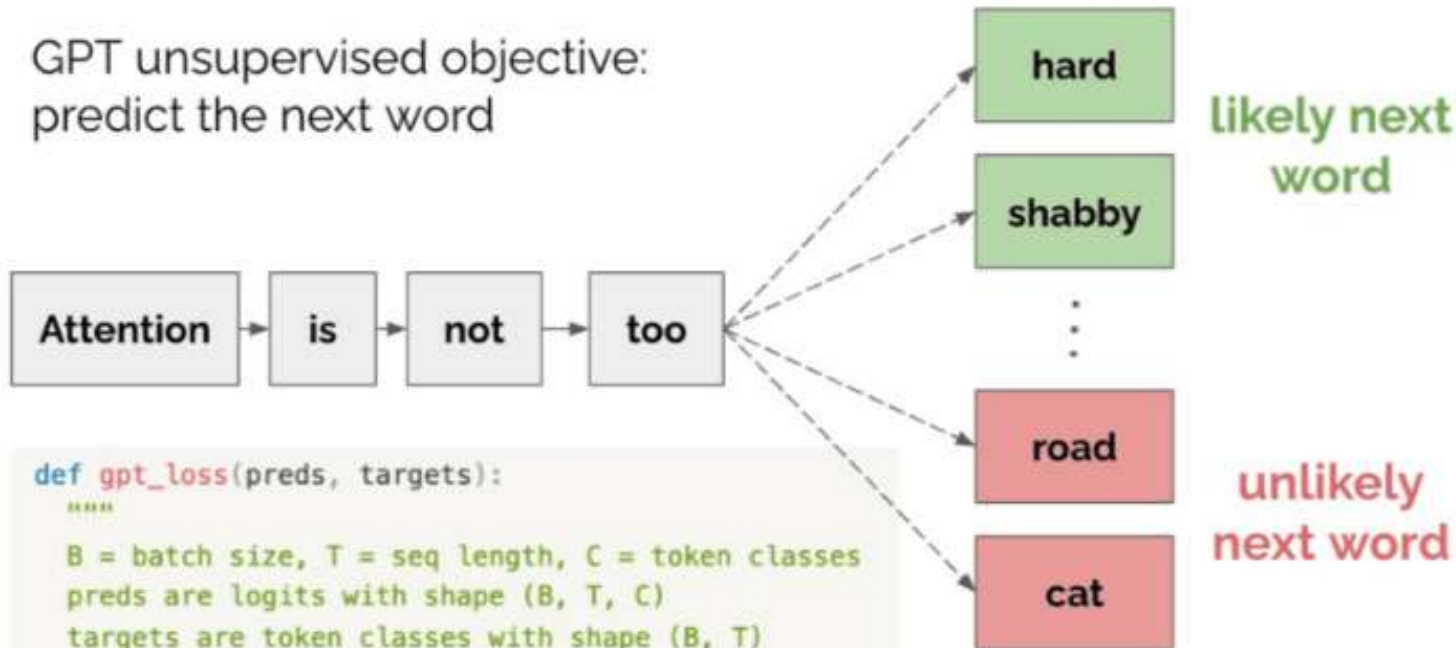
- Bidirectional Encoder Representations from Transformer – двунаправленный механизм внимания, при обработке входной последовательности все токены могут использовать информацию друг о друге
- BERT не учится генерировать тексты с нуля, первая его задача это предсказание случайно замаскированных слов по оставшимся (masked language modeling), вторая – предсказание по паре текстовых фрагментов, следуют они друг за другом или нет (next sentence prediction)



# GPT

- Generative Pretrained Transformer – языковая модель, реализованная в виде последовательности слоев декодера трансформера
- В качестве задачи при обучении выступает предсказание следующего токена (многоклассовая классификация по словарю)

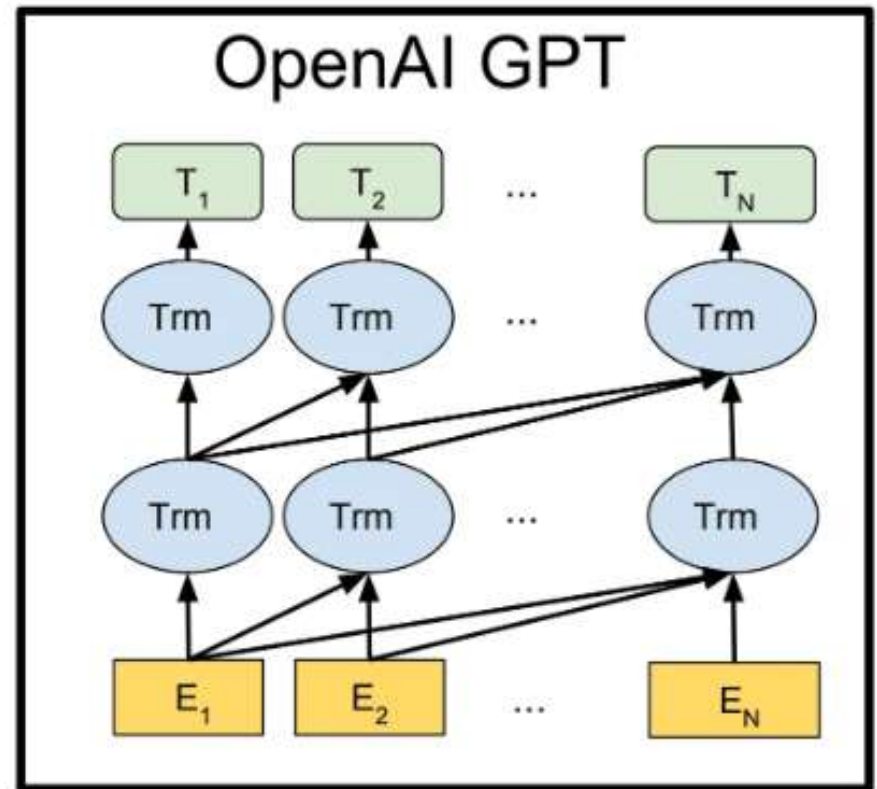
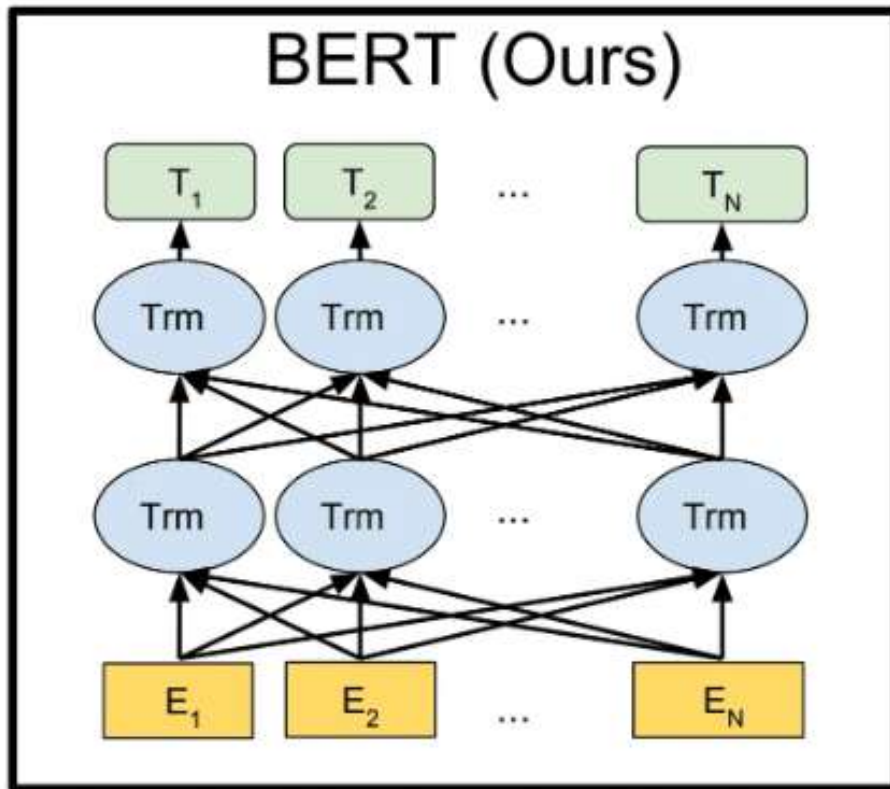
GPT unsupervised objective:  
predict the next word



```
def gpt_loss(preds, targets):  
    """  
    B = batch size, T = seq length, C = token classes  
    preds are logits with shape (B, T, C)  
    targets are token classes with shape (B, T)  
    """  
    preds = preds.view(B*T, C)  
    targets = targets.view(B*T)  
    return F.cross_entropy(preds, targets)
```

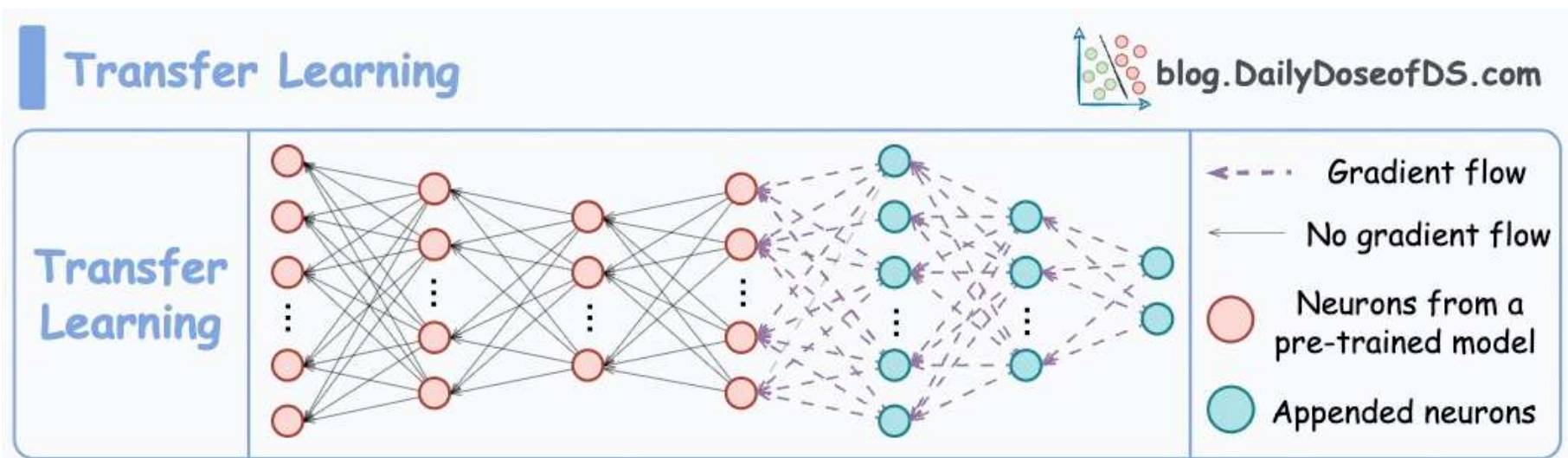
# BERT vs GPT

- Ключевое отличие – использование разных видов внимания



# Transfer learning

- Перенос обучения (transfer learning) – метод обучения, при котором модель, обученная для одной задачи (связанная задача), переиспользуется для схожей задачи (целевая задача)
- Целевая задача содержит мало данных. Например, классификация специфических изображений. Связанная содержит много данных, есть обученная модель которая хорошо решает связанную задачу.
- Заменяем от 1 до N последних слоев, обучаем только эти слои (веса в остальных слоях не изменяются)





# Fine-tuning

- Fine tuning – метод дополнительного обучения сети, при котором обученная на схожей задаче модель обучается на новом наборе данных. При этом веса не инициализируются случайно, а используются уже оптимизированные в процессе предыдущего обучения веса нейронной сети
- Можно добавить несколько слоев на выходе нейронной сети
- Обучаются все слои

