

# Интро в современную автоматическую обработку текстов (NLP)

для тех, кто в этом (почти) ничего не понимает, но всегда хотел  
разобраться

# Многообразие задач NLP

## Технические задачи:

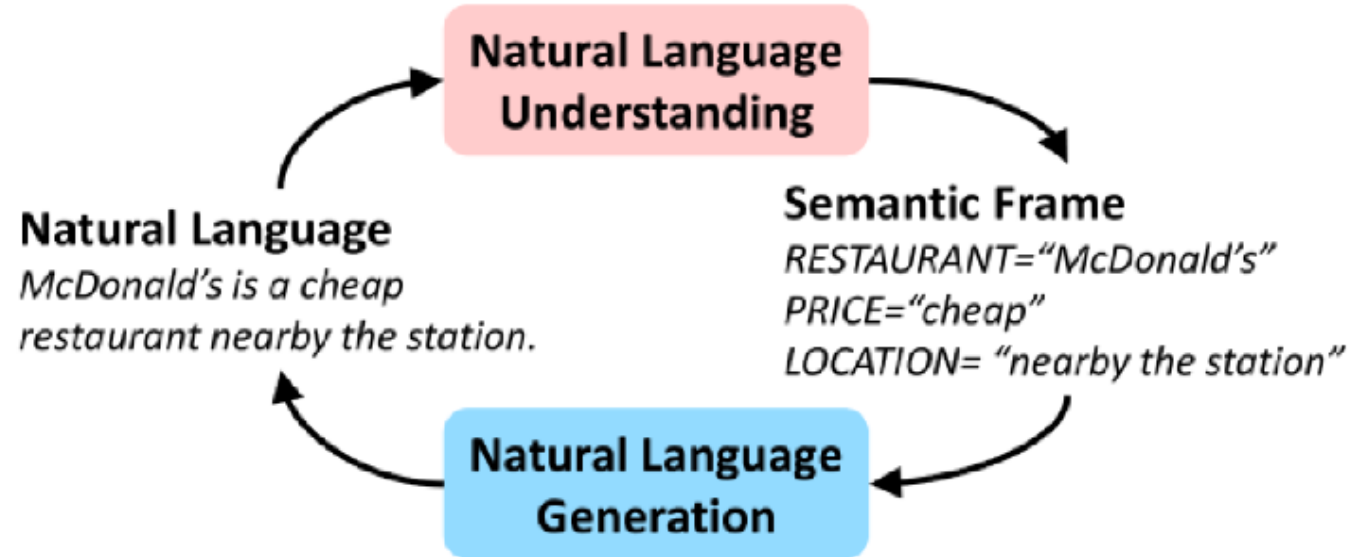
- ▶ Классификация
- ▶ POS-теггинг
- ▶ Извлечение сущностей
- ▶ Разрешение кореференции
- ▶ Лемматизация
- ▶ Сегментация
- ▶ Разметка семантических ролей
- ▶ Векторизация
- ▶ Машинный перевод
- ▶ Суммаризация
- ▶ ...

## Продуктовые задачи:

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

# Структура NLP

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



- ▶ Смежные области — распознавание (ASR) и генерация (TTS) речи




# Особенности обработки естественного языка

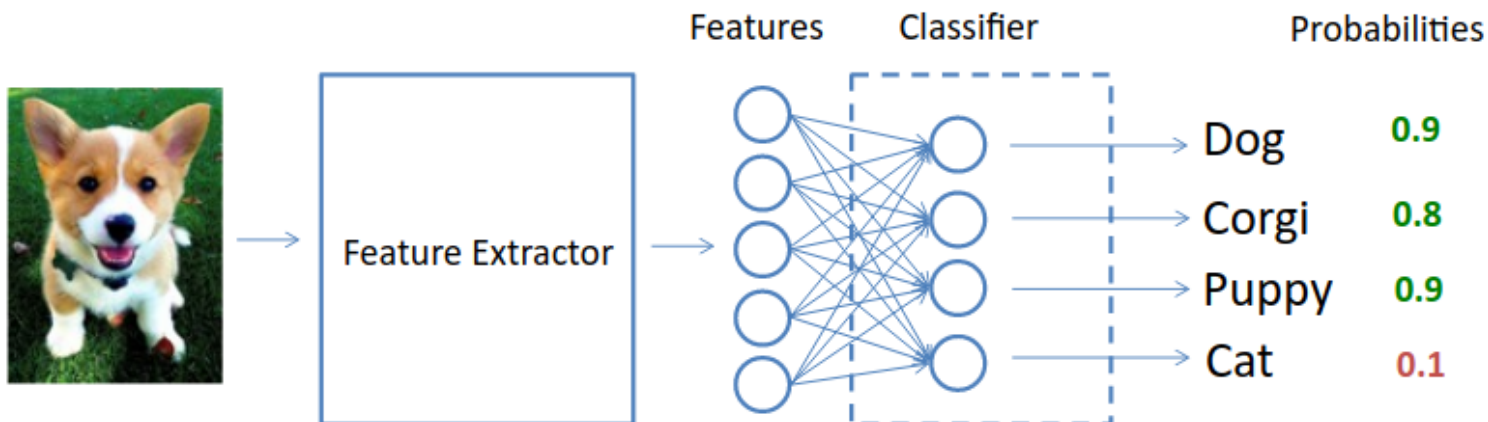
- ▶ Базовая структурная единица языка — слово
  - ▶ Даже вне контекста оно несёт полезную информацию
- ▶ В NLP обычно большие разреженные признаковые пространства
- ▶ Текст без дополнительной разметки имеет внутреннюю структуру, определяемую языком на разных уровнях:
  - ▶ текст (порядок реплик)
  - ▶ предложение (синтаксис)
  - ▶ слова и словосочетания (морфология, синтаксис)
- ▶ Наличие огромных массивов сырых текстов и структуры в них позволяет обучать **большие общезыковые модели**
  - ▶ Такие модели легко дообучать для решения конкретной задачи даже на небольшом объёме данных
- ▶ Существует много лингвистических ресурсов, которые помогают в различных задачах обработки текстов

# Как будет устроена беседа сегодня

- Представление любых данных для автоматической обработки
- Общие принципы классификации
- Минимум математики, линейные модели
- Представление текстовых данных для автоматической обработки
- Рассказ про Classic ML
- Более сложные подходы к обработке, рассказ про DL
- Главная задача NLP
- Рекуррентные нейросети, механизм внимания
- Трансформеры и иже с ними

# Классификация чего угодно

Что угодно   
Удобное для компьютера **информативное** представление (**признаки**)   
Умный чёрный ящик (**модель**)   
Метка класса



## ВОПРОСЫ:

- что за признаки, и где их взять?
- что за модель, и где ее взять,

# На пути от "чего угодно" к признакам

- Пусть наше "что-то" сейчас - точки на листе бумаги
- Пусть наша задача - понять, в каком углу листа находится точка



# На пути от "чего угодно" к признакам

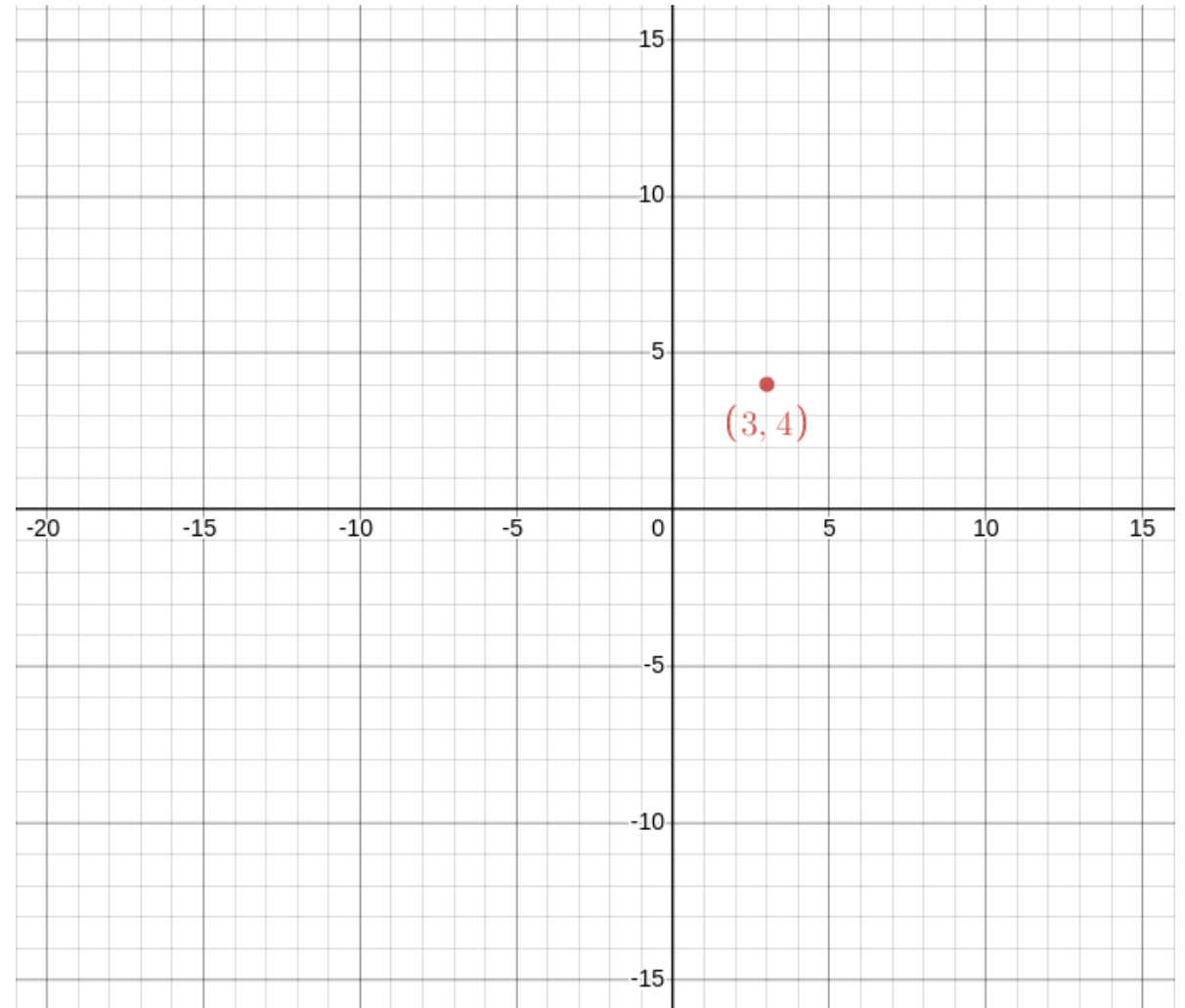
- Компьютер понимает только числовую информацию
- Надо описать точку **информативными** числами
- Информативными с точки зрения решаемой задачи





# На пути от "чего угодно" к признакам

- Нарисуем на листке оси координат
- Выставим шаг в 1 см
- Сопоставим точке её координаты - 2 числа
- Это двумерный **вектор**
- Каждая ось - **признак**
- Каждый элемент вектора - **значение признака**

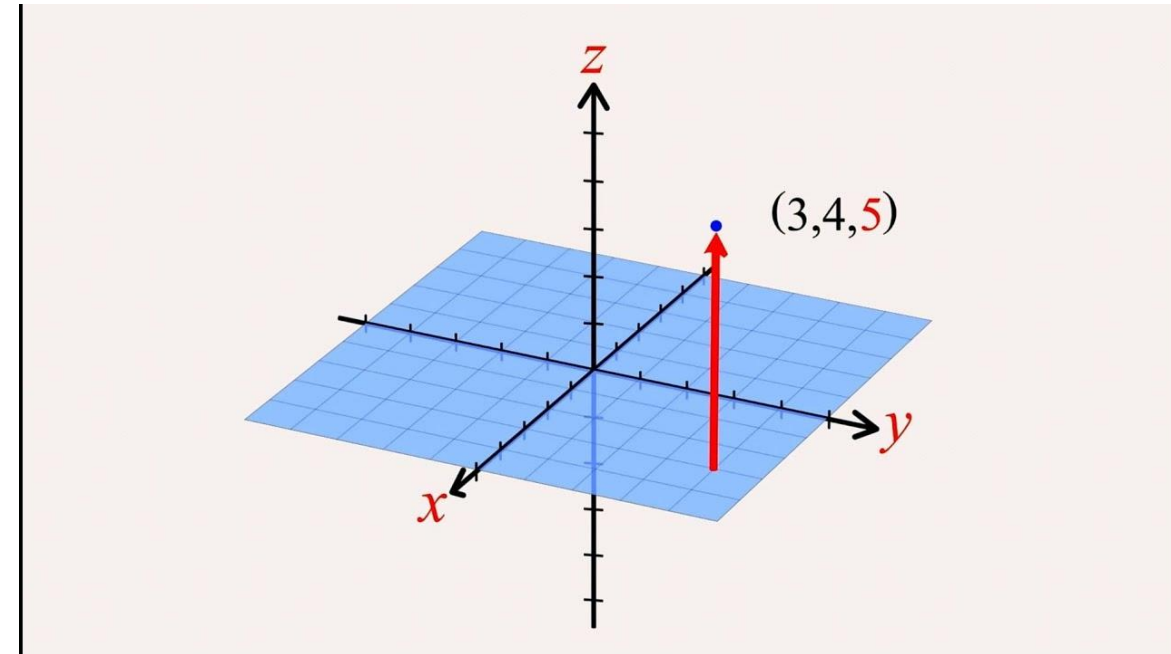


# На пути от "чего угодно" к признакам

- Компьютеру очень удобно работать с вектором
- Можем обучить модель (не важно, как и какую) отделять друг от друга точки на листе бумаги по их координатам
- Реальная задача: определить, болен ли человек, по его температуре и весу
- Сведем эту задачу к предыдущей: пусть
  - ось  $X$  - температура
  - ось  $Y$  - вес
- Каждый наблюдаемый человек будет представлен точкой-вектором из двух чисел

# На пути от "чего угодно" к признакам

- Для двух признаков понятно?
- Для трёх и более все аналогично!
- Можем добавить признак "рост"
- И все люди-объекты-точки будут описываться векторами из трёх чисел



# На пути от "чего угодно" к признакам

- **Итог:** для описания любого объекта реального мира на понятном для компьютера языке нужно сделать две вещи:
  - Придумать набор из  $N$  признаков, которыми можно описать этот объект
  - Для каждого объекта измерить и зафиксировать значения этих признаков

# На пути от "чего угодно" к признакам

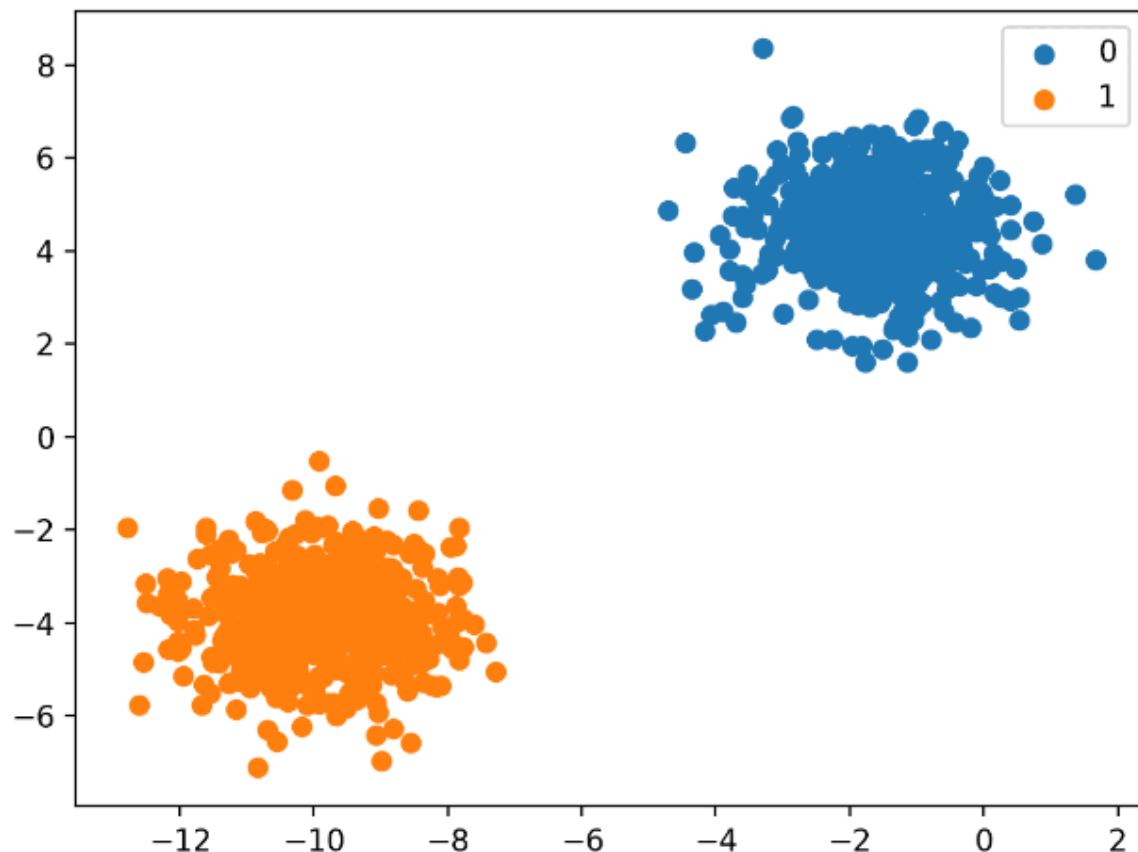
- Это задача feature engineering - придумать такие признаки, чтобы они были полезными для нашей задачи, и их было реалистично посчитать
- Пример малополезного признака для задачи определения болезни - цвет автомобиля человека
- Пример признака, значения которого сложно посчитать - процент мутировавших клеток

# Теперь про модель

- Объект в задаче классификации - это просто набор чисел, вектор
- Раз компьютеру удобно работать с числами, логично, что модель представляет собой что-то похожее
- Это тоже какой-то набор чисел и какие-то правила, по которым эти числа взаимодействуют с вектором объекта так, что на выходе получается вердикт - относится объект к какому-то классу, или нет
- Вернемся к задаче с определением наличия болезни (т.е. 2 класса - болен или нет)

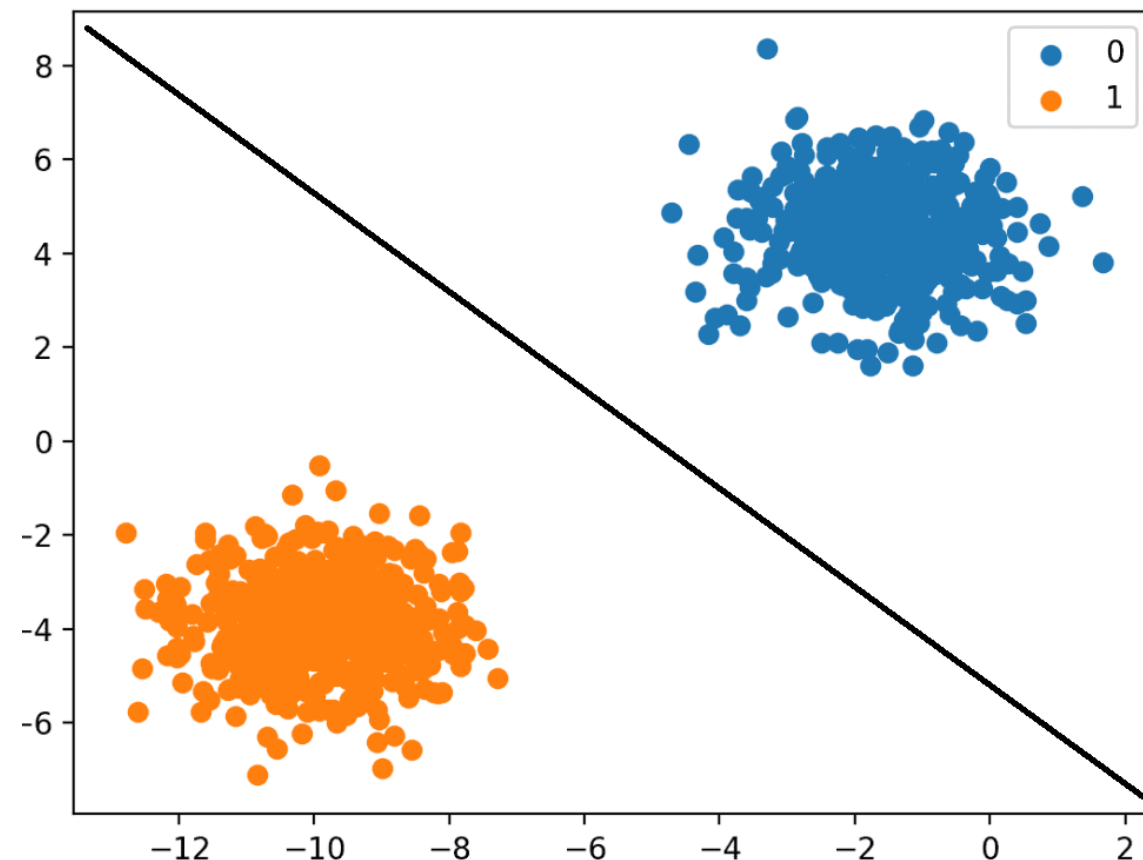
# Теперь про модель

- Вернемся к задаче с определением наличия болезни (т.е. 2 класса - болен или нет)
- Для наглядности оставим 2 исходных признака: температуру и вес
- Какая простая линия могла бы разделить эти два облака точек?



# Теперь про модель

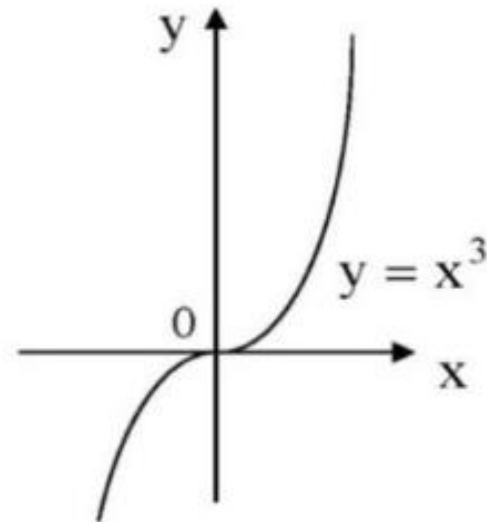
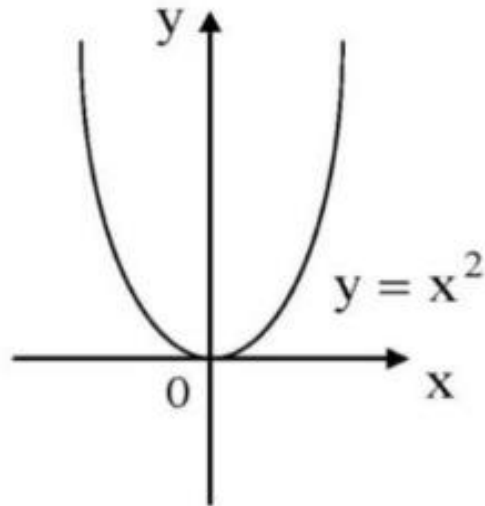
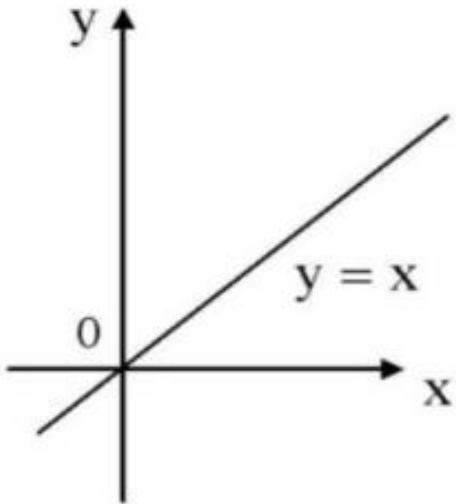
- Прямая - это проще всего (из адекватных решения)
- Всех, кто попал в одну сторону, считаем больными, кто в другую - здоровыми
- Для описания прямой нужны функции





# Немного математики

- Вспомним, что такое функция (7 класс)  $y = f(x)$
- В более общем случае это  $f(x, y) = 0$
- Это закон, который описывает все точки с координатами, которые делают это равенство верным
- Вспомним самые простые примеры:



# Немного математики

- Прямая описывается уравнением  $A \cdot x + B \cdot y + C = 0$
- Если мы выберем какие-то конкретные  $A$ ,  $B$  и  $C$  (например, 2, 3 и 4), то получим прямую
- Любая точка на этой прямой при подстановке в левую часть уравнения даст 0
- Для всех остальных точек  $(x, y)$ 
  - Если  $2 \cdot x + 3 \cdot y + 4$  больше 0, то точка попадёт по одну сторону прямой
  - Если меньше - по другую

# Теперь про модель

- Если знать подходящие  $A$ ,  $B$  и  $C$ , то можно легко отделить ею больных и здоровых людей
- Мы получили модель!
  - Параметры (**веса**): три числа  $A$ ,  $B$  и  $C$
  - Правило классификации: попадает ниже прямой - здоров, выше - болен
- Эта базовая и очень эффективная модель ML - **линейная модель**

# Поприветствуем скалярное произведение

- Векторы, как и числа, можно умножать
- Мы уже это делали:  $A \cdot x + B \cdot y$
- Это произведение двух векторов:  $a = (A, B)$  и  $b = (x, y)$
- Для сокращения эту операцию называли скалярным произведением и обозначили  $\langle a, b \rangle$
  
- Линейная модель - это тоже вектор! (+ свободный член  $C$ )
- СП удобно использовать для нотации (представим, что признаков не 2, а 20)

# Как модель узнает, что ей нужно делать

- Решили использовать линейную модель для задачи определения больного по температуре и весу
- Коэффициенты  $A$ ,  $B$  и  $C$  нужно подобрать по данным автоматически и не перебором
- Можно "показывать" модели правильные примеры, т.е. примеры векторов больных и здоровых людей
- При этом модели нужно как-то давать понять, правильно ли она принимает решение или нет

# Обучение модели

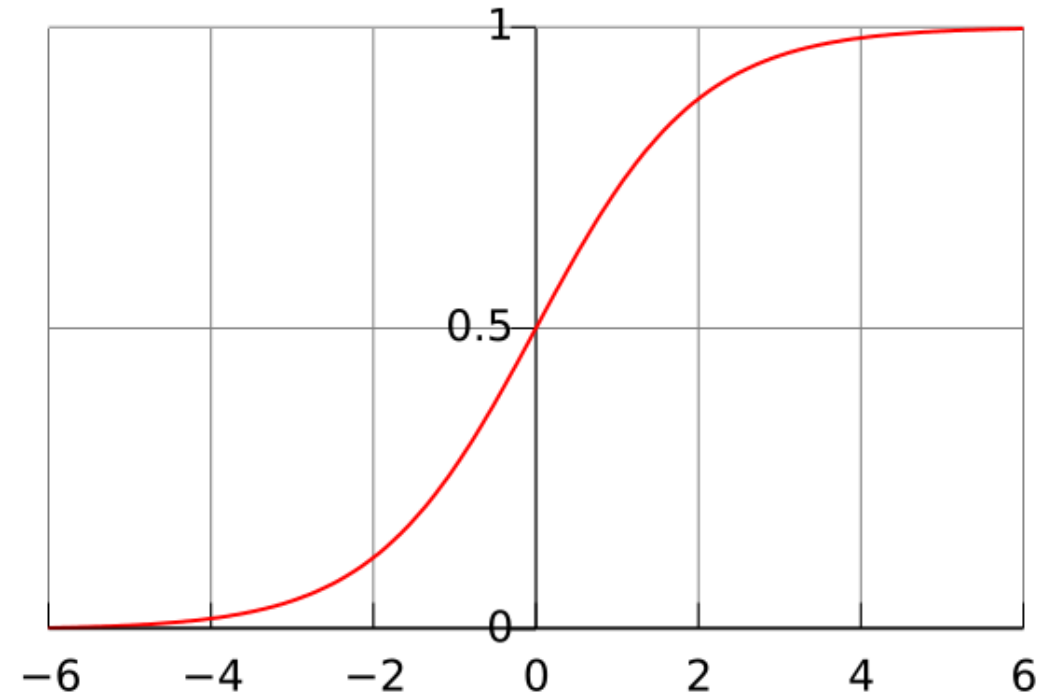
- "Правильность" должна быть представлена числом (иначе компьютер не поймет)
- Считаем, что для больных людей модель должна выдавать число  $> 0$  на выходе, а для здоровых  $< 0$
- Предположим, что на этапе обучения мы показываем модели пример здорового человека
  - если ответ модели отрицательный - ничего не делаем
  - если положительный - требуем от модели подкорректировать свои веса (A, B) и C, чтобы в следующий раз не ошибаться

# Обучение модели

- Процессом изменения весов модели должен кто-то управлять
- Этим занимается выбранный **метод оптимизации**
- Неважно, что это за методы, и как они работают
- Достаточно того, что они умеют по ошибке модели корректировать её веса, чтобы она ошибалась меньше
- Вопрос в том, как адекватно посчитать ошибку
- Для этого вводят **функцию потерь**, которая считает ошибку на основании выхода классификатора и правильного ответа

# Модель линейной регрессии

- Если для линейной модели использовать т.н. логистическую функцию потерь, то мы получим модель **логистической регрессии**
- Получается линейная модель, результат которой подаётся в функцию сигмоиды
- Формула неважна, эта функция переводит любое число в отрезок от 0 до 1
- Результат модели - вероятность того, что человек болен





# Обучение модели

- Теперь для обучения достаточно взять много примеров больных и здоровых людей (точнее, значения их температур и весов) и запустить алгоритм оптимизации
- В результате будут настроены веса  $A$ ,  $B$  и  $C$
- Дальше по правилу сигмоида от  $\langle (A, B), (x, y) \rangle + C$  можно получать предсказание для нового объекта  $(x, y)$

# Промежуточный итог

- Разобрали на пальцах общую концепцию линейных моделей и их обучения
- **Это важно:** на основе векторов, скалярного произведения и линейных моделей строится все, что будем разбирать дальше
- Можно вернуться к текстам и снова вспомнить о получении признаков

# Какие признаки у текста

- Придумать можно очень много
- Но мы попробуем максимальное общие, исходя из того, что знаем только слова текста, больше ничего
- В задаче анализа тональности (плохая или хорошая) признаком может быть наличие в тексте слова "хорошо"

Номер текста	Содержит «хорошо»
1	1
...	...
N	0

# Какие признаки у текста

- Но одного слова мало, наличие всех слов является хорошими признаками:

Номер текста	Содержит «абрикос»	...	Содержит «яблоко»
1	0	...	1
...	...	...	...
N	1	...	0

# Какие признаки у текста

- Еще полезно посмотреть на то, сколько раз слово встретилось в тексте:

Номер текста	Вхождений «абрикос»	...	Вхождений «яблоко»
1	0	...	23
...	...	...	...
N	2	...	0

- Такое представление текста называется "**мешком слов**"

# А почему только слов

- Со словами есть проблема: "яблоко" - "яблоки" - "яблок"
- Для борьбы с этим можно вместо слов (или с ними) использовать
  - Леммы ("яблоко")
  - Стеммы ("яблок")
  - Символьные N-граммы ("ябл", "бло", ...)
  - Словарные N-граммы ("вкусное яблоко", "яблоко раздора", ...)

# Собираем все вместе

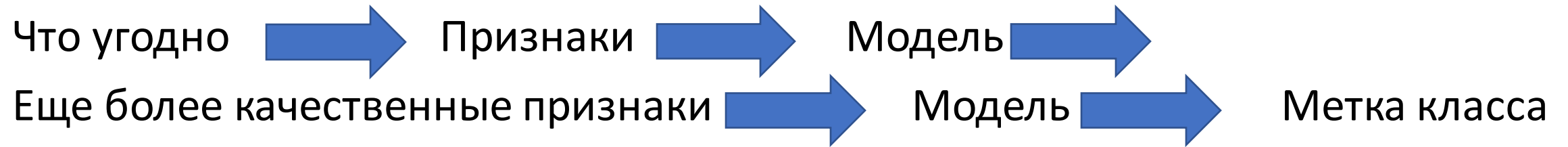
- Извлекаем из текстов значения всех указанных выше признаков
- Обучаем на них логистическую регрессию
- Получаем **Classic ML:**)

# Не мешком единым живем

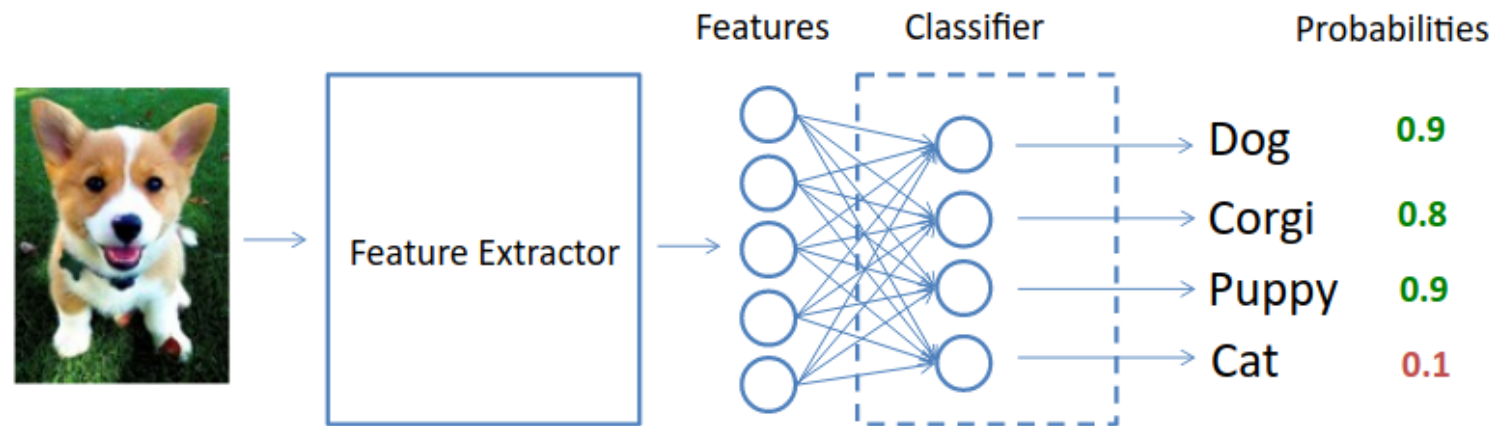
- Мешок признаков удобный, легко считается и довольно эффективный, но в сложных случаях он пасует
- Частота встречаемости слова в тексте сама по себе тоже не всегда является адекватным признаком
- Что можно сделать?
  - Для применения классификатора текст должен быть превращён в вектор
  - Но получать вектор необязательно на основании вручную выбранных признаков
  - Можно получить его, обучив другую модель



# Классификация чего угодно



- Feature Extractor тоже может быть модель, и даже линейной моделью!



# Сперва на примере все того же мешка

- Никто не мешает соединять последовательно много линейных моделей
- Пусть в задаче анализа тональности с "мешком слов" 50К уникальных слов-признаков
- Выучим на этих признаках 100 линейных моделей
- Их векторы весов можно положить друг на друга и получится прямоугольная таблица - **матрица**

Пример для 3 признаков на входе и 2 на выходе

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix} = \begin{bmatrix} (1)(7)+(2)(8)+(3)(9) \\ (4)(7)+(5)(8)+(6)(9) \end{bmatrix} = \begin{bmatrix} 7+16+27 \\ 28+40+54 \end{bmatrix} = \begin{bmatrix} 50 \\ 122 \end{bmatrix}$$

$2 \times 3$        $3 \times 1$        $2 \times 1$        $2 \times 1$        $2 \times 1$

columns on 1st = rows on 2nd

The number of rows in the 1st matrix and the number of columns in the 2nd matrix, make the dimensions of the final matrix

# Учим больше весов

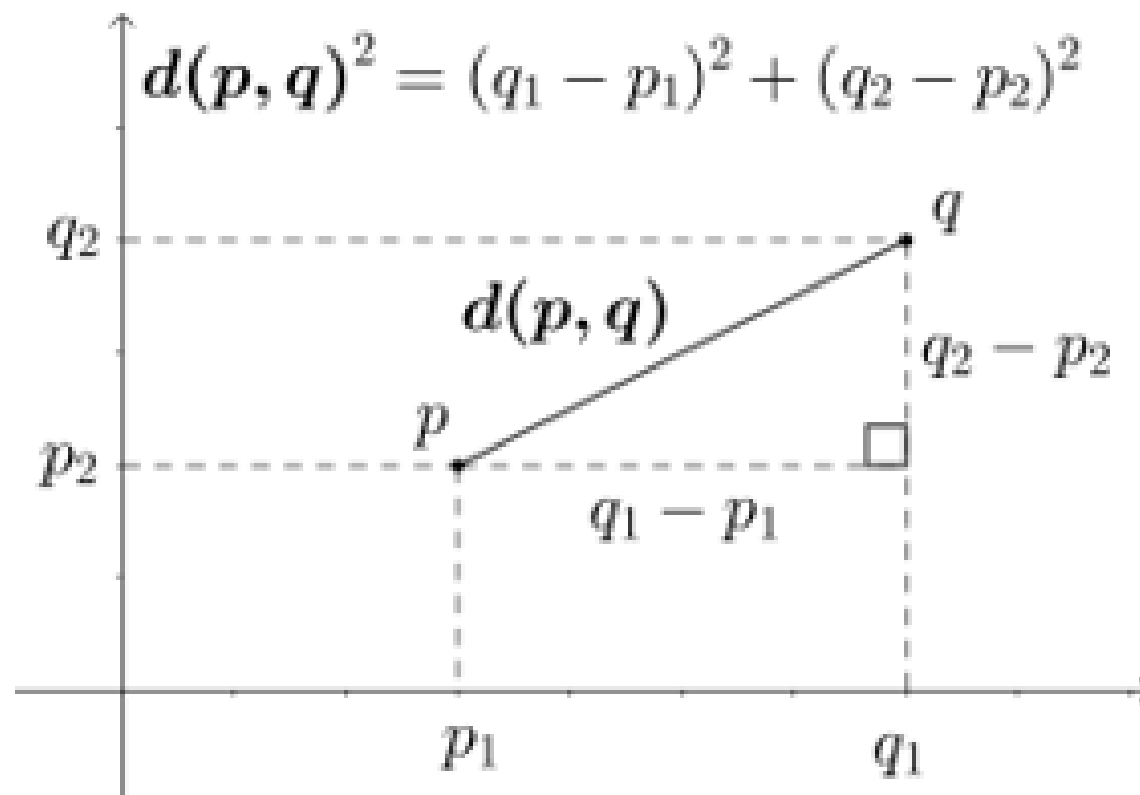
- Каждая модель даст на выходе одно число
- Получаем вектор из 100 чисел - новые признаки
- Признаки неинтерпретируемые
- Их фиксированное небольшое количество
- Но они содержат много полезной информации о тексте
- Дальше на этих признаках можно учить линейный классификатор
- Тут ремарка: чтобы это работало, после первого линейного слоя нужно добавить какое-то нелинейное преобразование (ReLU, tanh)
- Получилась полносвязная нейронная сеть!

# Сжатые векторные представления слов

- Можно до получения вектора текста сперва получить хорошие векторы для каждого слова и потом использовать их
- **Поверим:** на основе описанной выше модели можно обучить заранее векторы, которые будут хорошо отражать семантическую близость
- Это значит, что векторы слов, близких по смыслу, будут близки (и наоборот)

# Что значит "близость векторов"

- Между векторами можно считать расстояние (как между точками на листке)
- Вспомним школьную геометрию, **евклидово** расстояние
- Для многомерного случая все аналогично
- В текстах чаще используется другое расстояние (**косинусное**) но суть абсолютно та же

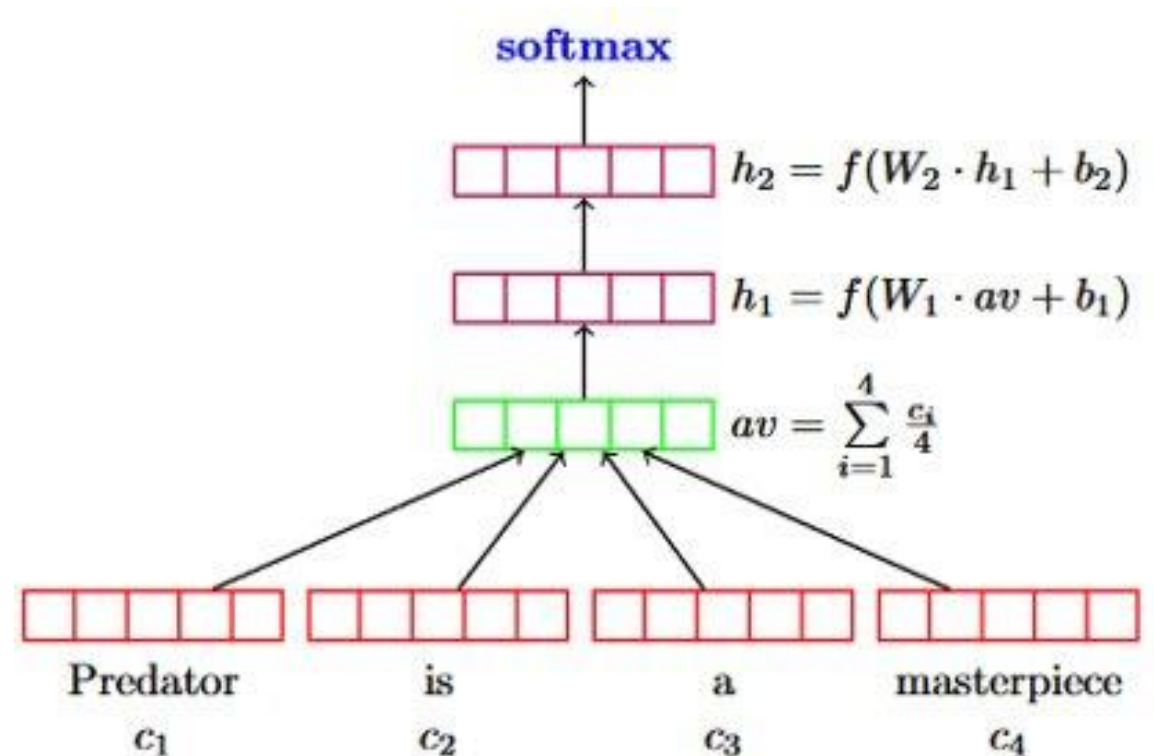


# Сжатые векторные представления слов

- **Вывод:** есть модель, которая
  - получает на вход слово (прямо в виде строки)
  - возвращает для него вектор фиксированного размера (200, 300, 500)
- Если мы измерим расстояние между векторами слов "торт", "кекс" и "интернет", то
  - "торт" и "кекс" будут близки
  - от "интернет" оба будут далеки
- Такие модели есть не только для слов, но и для словарных и символьных N-грамм

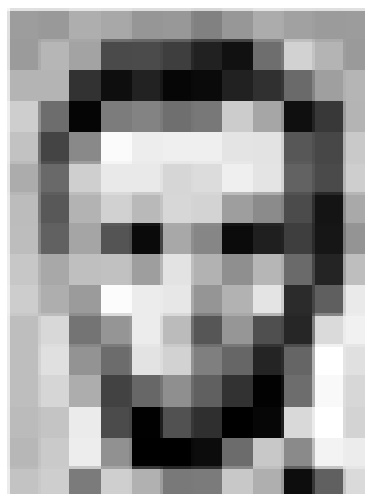
# Как использовать векторы слов

- Можно просто сложить векторы всех слов текста и учить классификатор на них
- Но это не лучшее решение, опять нет никакого учета порядка



# Как использовать векторы слов

- Один способ пришел из компьютерного зрения
- Вспомним, что такое BMP картинка (для простоты - серая)
- Это таблица (матрица) пикселей, где в ячейках записана интенсивность серого цвета
- В CV для работы с такими данными используются свёрточные нейронные сети (CNN)



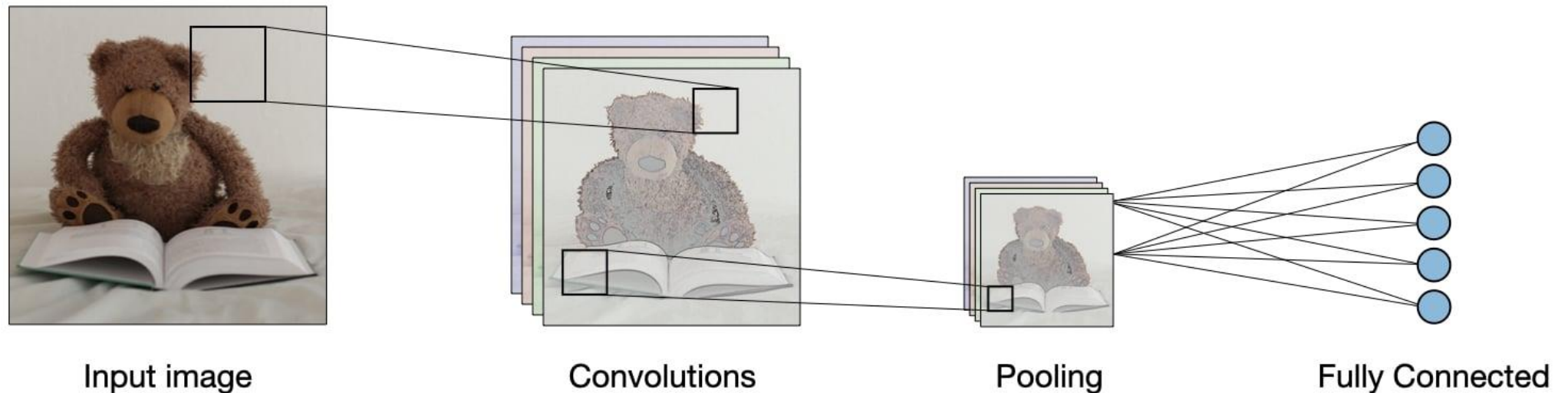
187	183	174	166	150	143	129	125	112	703	108	108
188	183	183	71	71	81	23	17	112	210	180	184
185	188	83	14	24	9	70	23	88	100	100	181
204	108	8	124	121	111	123	204	165	15	86	180
184	88	107	83	87	89	208	228	827	71	281	
158	188	267	88	88	214	204	268	128	84	14	268
188	88	178	200	188	218	271	188	180	78	20	188
188	87	188	88	82	188	124	18	88	82	23	188
188	188	188	188	188	237	178	183	182	100	26	182
205	174	155	152	136	87	148	178	288	49	85	284
188	214	116	140	88	187	88	150	78	88	218	241
188	204	187	188	227	218	127	182	88	101	288	224
188	214	178	88	122	182	88	88	2	108	248	218
187	188	228	71	81	87	8	8	217	288	278	
183	228	227	145	8	8	78	108	100	138	243	228
188	224	123	267	177	121	123	288	178	118	88	218

187	183	174	166	150	143	129	125	112	703	108	108
188	183	183	71	71	81	23	17	112	210	180	184
185	188	83	14	24	9	70	23	88	100	100	181
204	108	8	124	121	111	123	204	165	15	86	180
184	88	107	83	87	89	208	228	827	71	281	
158	188	267	88	88	214	204	268	128	84	14	268
188	88	178	200	188	218	271	188	180	78	20	188
188	87	188	88	82	188	124	18	88	82	23	188
188	188	188	188	188	237	178	183	182	100	26	182
205	174	155	152	136	87	148	178	288	49	85	284
188	214	116	140	88	187	88	150	78	88	218	241
188	204	187	188	227	218	127	182	88	101	288	224
188	214	178	88	122	182	88	88	2	108	248	218
187	188	228	71	81	87	8	8	217	288	278	
183	228	227	145	8	8	78	108	100	138	243	228
188	224	123	267	177	121	123	288	178	118	88	218



# Сверточная нейросеть

- **Вход:** матрица (первичные признаки)
- Маленькая модель просматривает её по квадратам
- Она выдаёт для каждой области одно число
- Оно агрегирует информацию из этой области
- **Выход:** новая матрица (новые признаки)



# Как использовать векторы слов

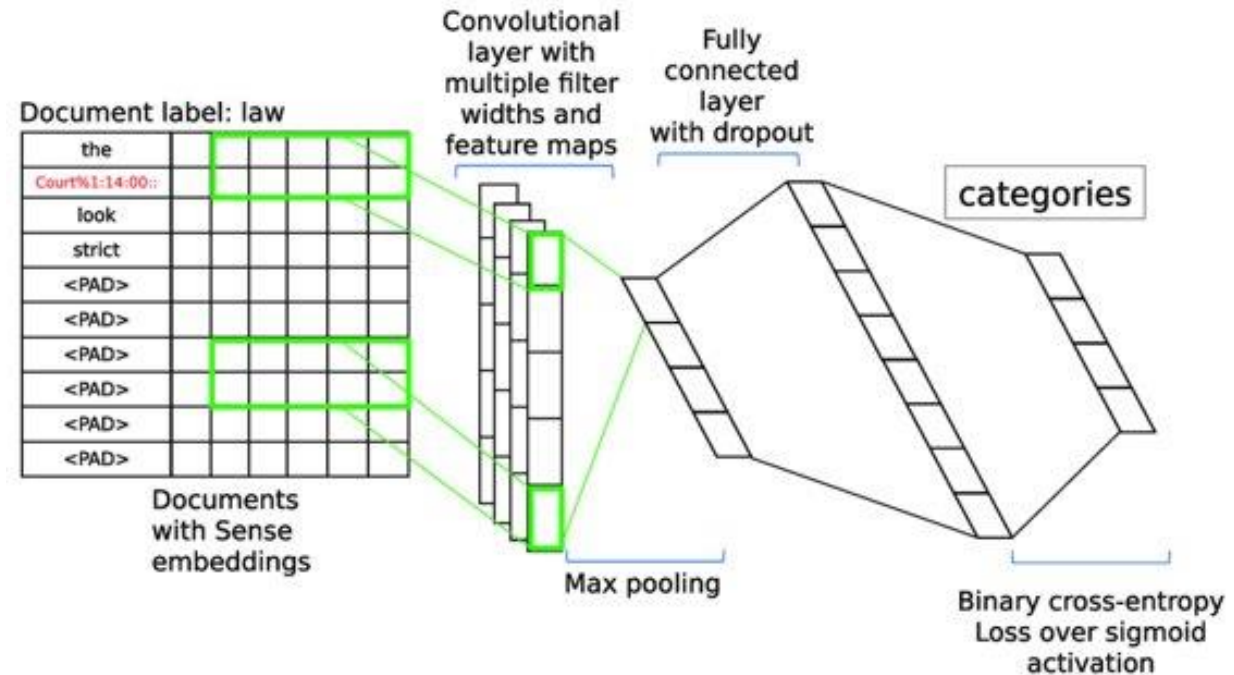
- У слов входного текста есть обученные заранее векторы (но можно учить и по ходу дела!)
- Составим из этих векторов матрицу по порядку слов в тексте
- Будем просматривать эту матрицу так же, как картинку, но с прямоугольной моделью (по горизонтали смотрим сразу на все)
- Проходим вниз по всему тексту и получаем вектор значений - новые признаки





# Сверточная нейросеть

- Все маленькие модели достают признаки, но их много
- Ещё они разных размеров
- Зато моделей фиксированное количество
- Возьмём из вектора каждой только самое большое значение и будем считать его признаком
- Получим один вектор таких значений
- И на этом векторе признаков выучим линейный классификатор!



# Просим любить и жаловать - DL (в JAICP)

- Есть много опущенных деталей, но суть примерно такая
- Берем предобученные векторы фрагментов слов
- Для входного текста составляем матрицу из векторов этих слов в порядке их следования в тексте
- Обучаем много маленьких моделей, которые извлекают свои векторы промежуточных признаков
- Выбираем из каждого вектора максимальное значение (пулинг) и составляем из них итоговый вектор признаков
- Подаём его в логистическую регрессию, она учится предсказывать класс
- Все параметры (веса) учатся одновременно
- По сути - много линейных моделей и несколько нелинейных операций (пулинг и сигмоида)

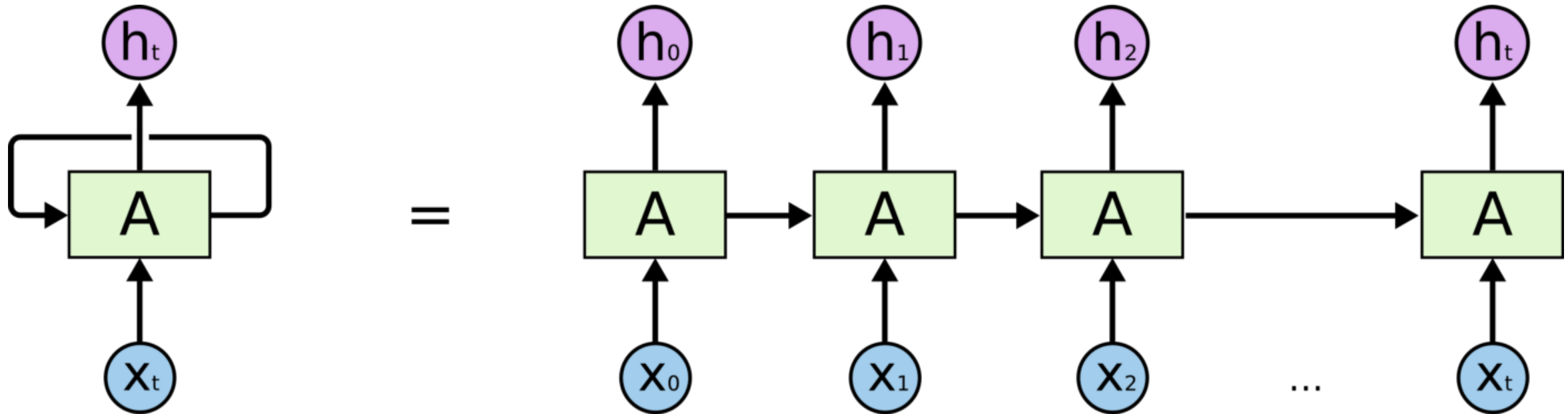
# Главная задача NLP

- Формально такого титула нет
- Но по факту это машинный перевод
- Задачу пытаются решить с 50-х годов, и самые крутые идеи в мире NLP пришли именно из попыток улучшить перевод

# Рекуррентная нейросеть

- Со словами (= векторами) можно работать иначе
- В линейной модели с мешком слов они суммируются
- В CNN обрабатываются наборы из нескольких слов, после вся информация собирается в вектор
- В рекуррентных сетях (RNN) предлагается формировать вектор признаков документа, обрабатывая слова одно за другим
- При этом параметры (веса) модели используются одни и те же

# Общий вид RNN



- Внизу - векторы слов текста (можно брать готовые, можно учить)
- Сверху - выходы сети (результаты) для слов текста
- Справа - вектор состояния обработки текста

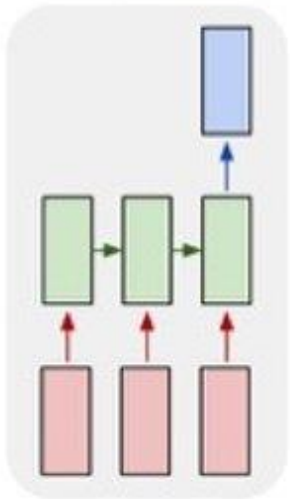


# Рекуррентная нейросеть

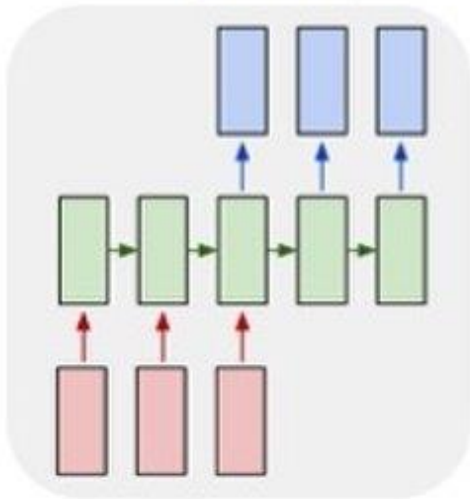
- Детали устройства особо значения не имеют
- Это могут быть две линейных модели и нелинейные функции для них
- Вектор состояния - это набор чисел, в которых нейросеть пытается собрать информацию про все увиденные слова текста и их порядок
- На выход после всех вычислений идут два вектора:
  - **Вверх**: вектор ответа модели для данного слова (с учетом состояния)
  - **Вправо**: обновлённый вектор состояния для обработки следующего слова
- Для простоты опустим генерацию слова из выходного вектора

# Примеры задач для RNN

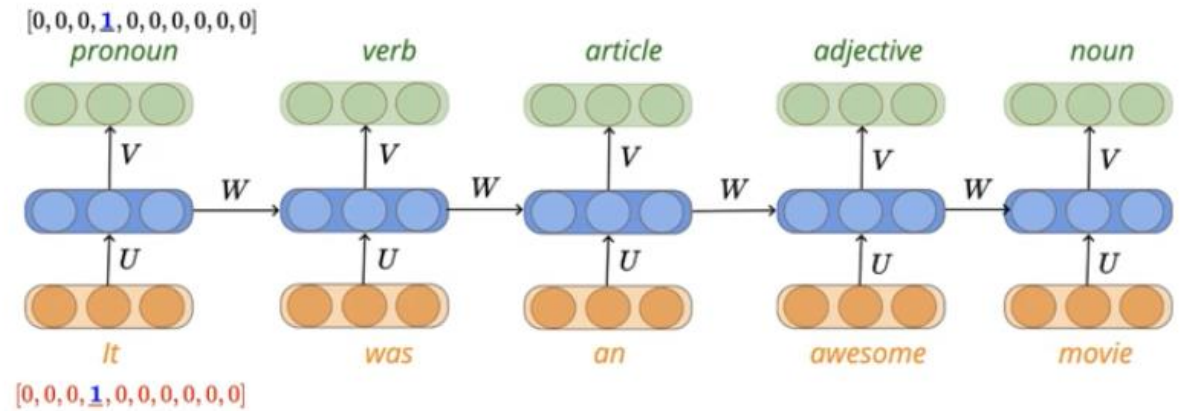
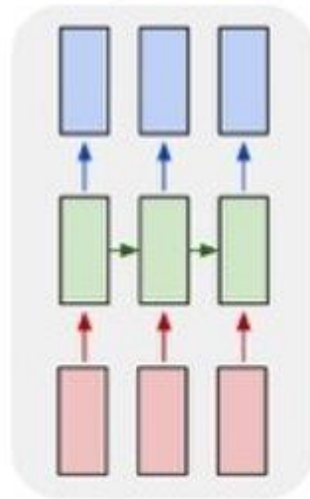
many to one



many to many

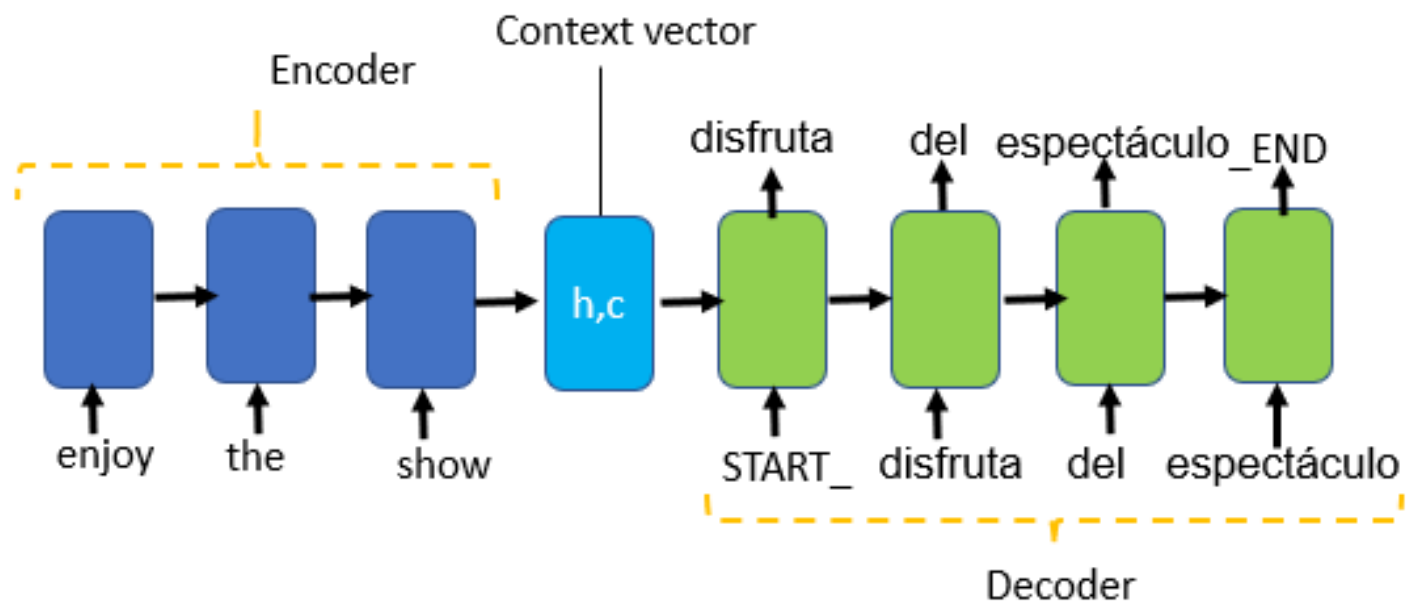


many to many



# Нейросетевой машинный перевод (NMT)

- Рекуррентные сети стали основой для подхода encoder-decoder
- RNN-кодировщик собирает по входному тексту вектор состояния
- RNN-декодировщик по этому вектору генерирует выходной текст
- Обучение: пары фраз на исходном и целевом языках



# Мир не идеален

- **Проблемы:**

1. Обычная RNN плохо запоминает длинные тексты
2. Последни слова сильнее влияю на вектор состояния

- **Решения:**

- Более совершенные архитектуры (LSTM, GRU, MGU)
- Механизм внимания - хорошо при генерации слова смотреть не на последние слова входа, а на важные для текущего слова

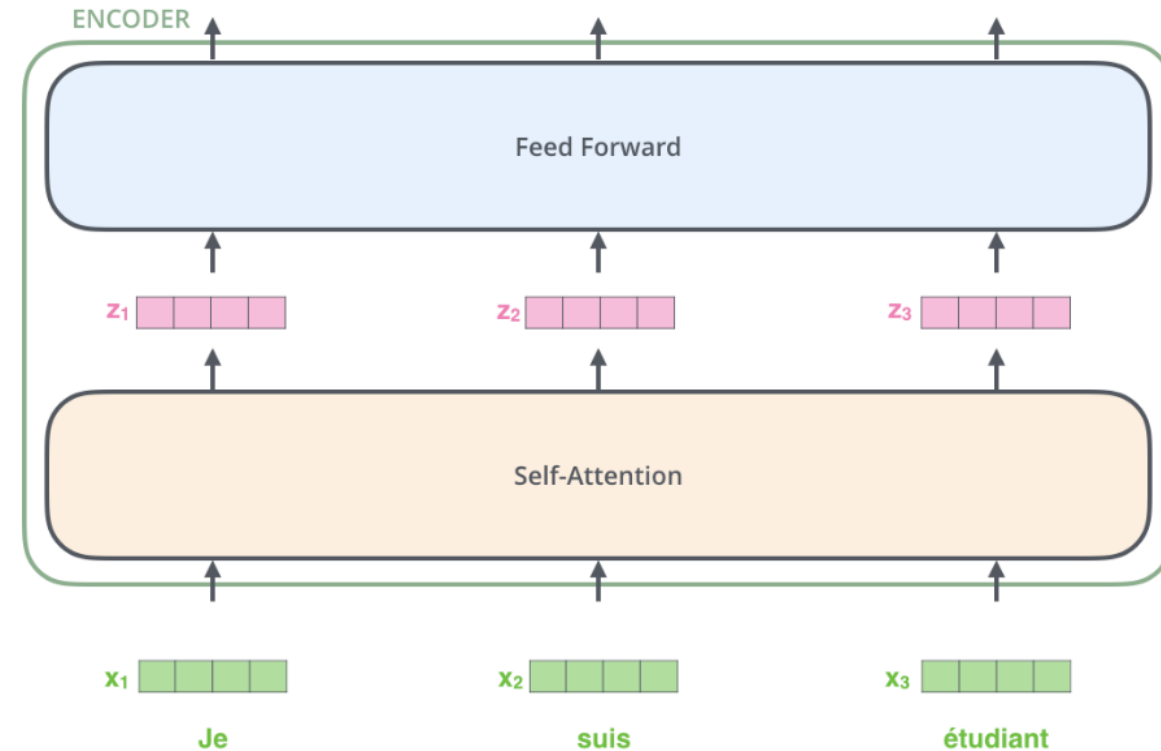


# От RNN к Transformer

- Зачем вообще нужна рекуррентность?
- Почему на этапе кодирования не обработать все слова текста сразу?
- Почему на этапе декодирования не учитывать все сгенерированные слова?
- Почему последовательность просматривается только в одном направлении (есть еще двунаправленные RNN, но и это не идеал)
- Вектор состояния для каждого слова в encoder - это новые признаки
- Хочется получить векторы признаков сразу для всех слов текста, и так, чтобы каждый обращал внимание на все остальные слова

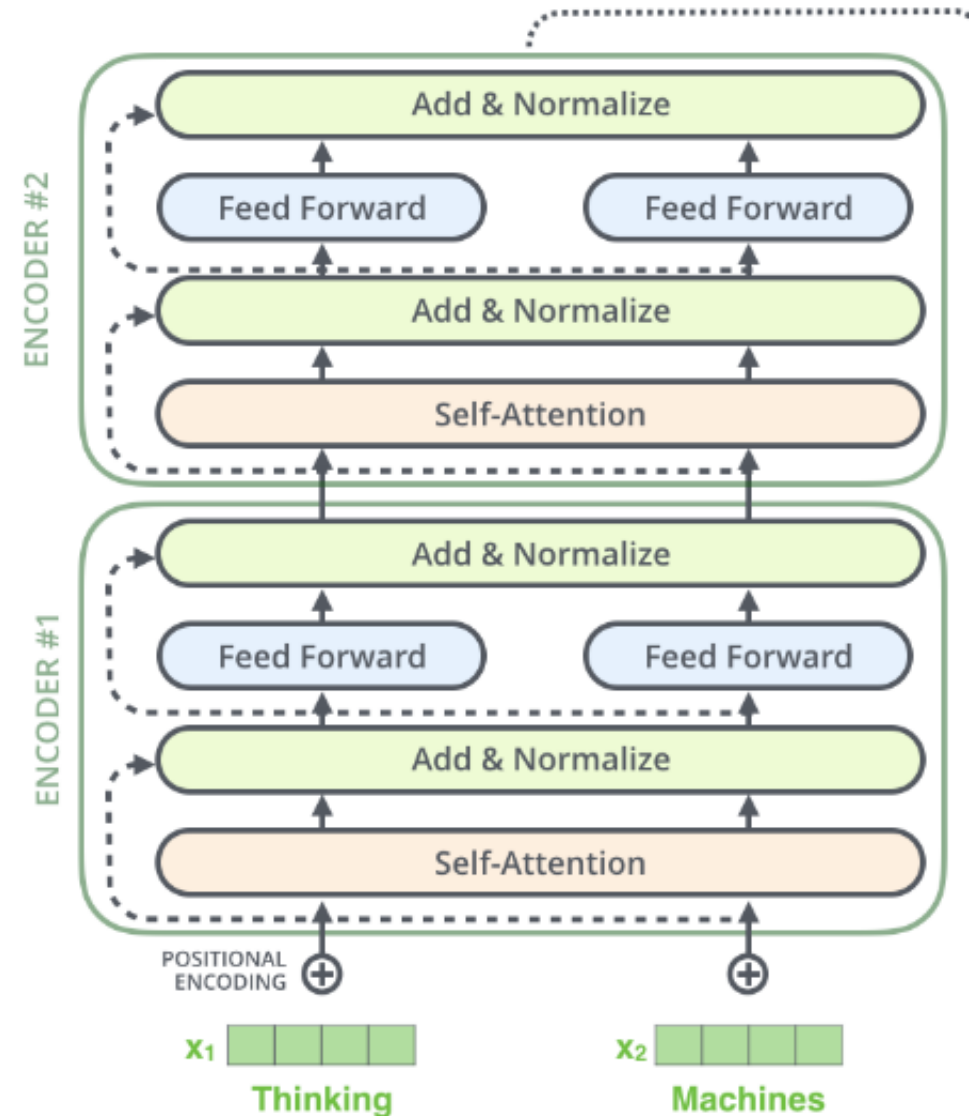
# Задачу решает Self-Attention

- Это механизм, который
  - для каждого входного слова смотрит на все остальные слова
  - решает, какие из них важнее для этого слова
  - генерирует новый вектор признаков для этого слова
- Вектор имеет такой же размер, что и входной
- Можно такие блоки ставить один за другим



# Кодировщик Transformer

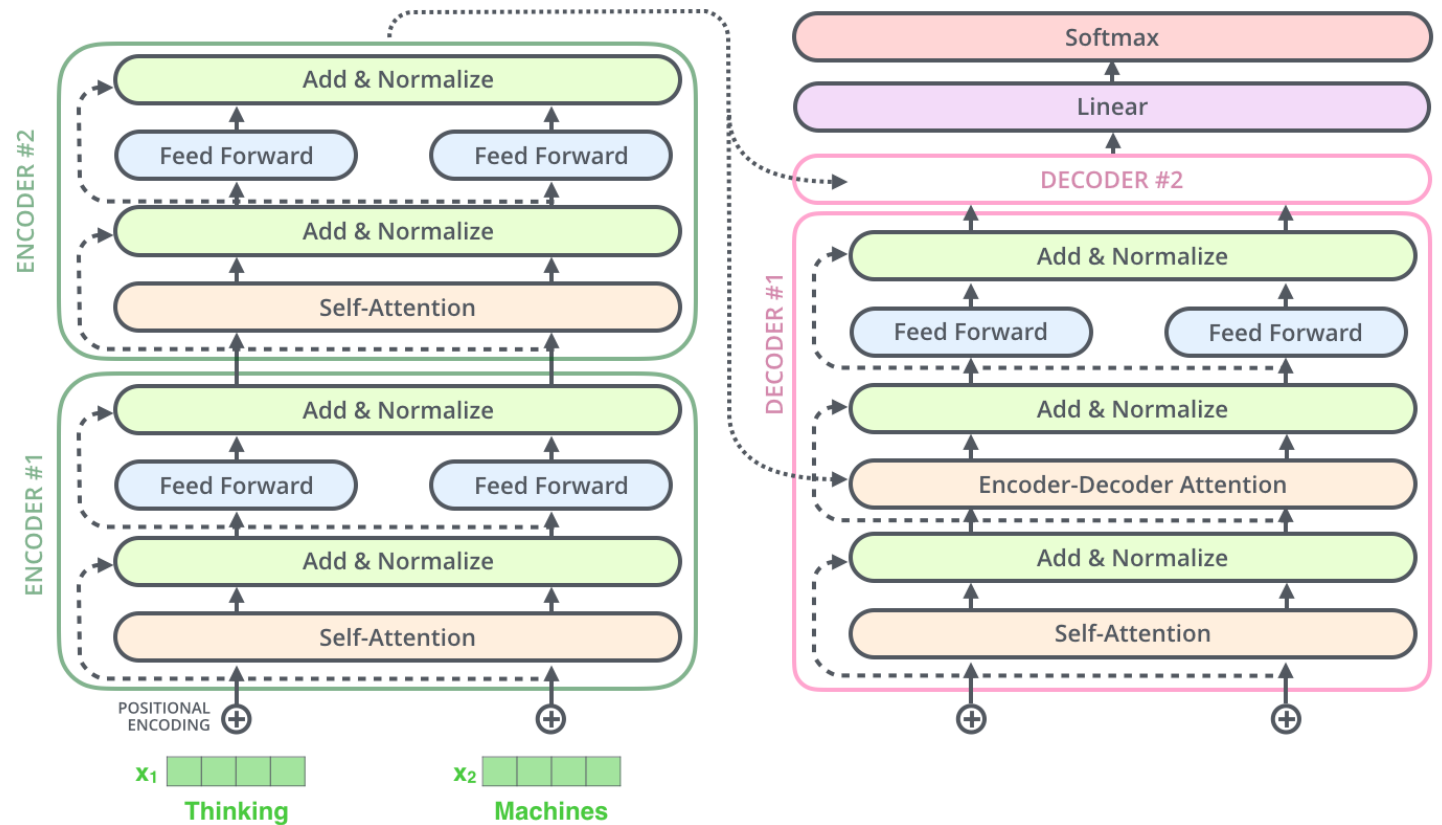
- Ставим блоки с self-attention один за другим
- На каждом уровне генерируем всё более хорошие векторы признаков для слов текста
- Опускаем детали, если захочется - обсудим в конце
- **Важно:**
  - Multihead self-attention
  - Линейная модель с нелинейной функций в конце





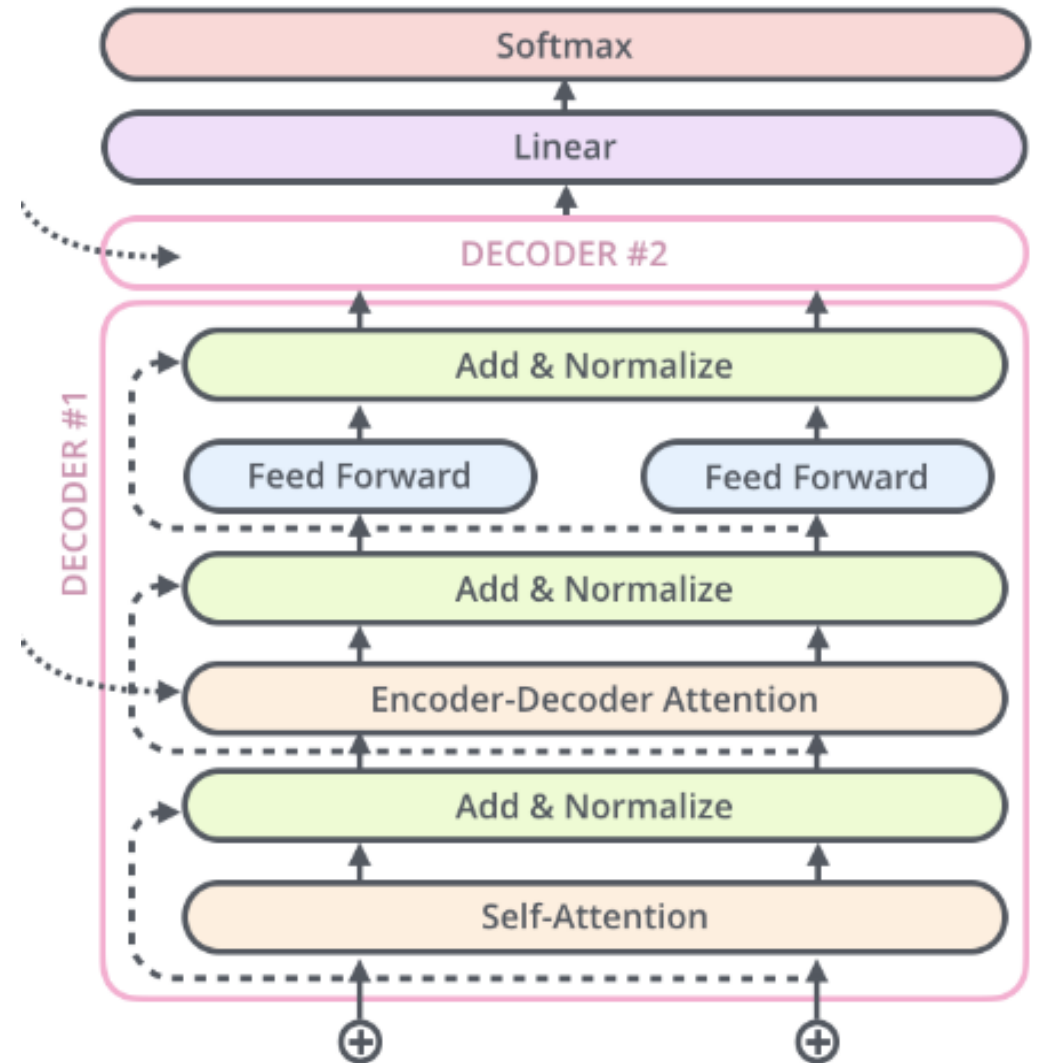
# Встречаем архитектуру Transformer

- Тоже encoder-decoder
- Блок self-attention - основная часть, из него конструируется все
- Опускаем детали, если захочется - обсудим в конце
- **Важно:**
  - Multihead self-attention
  - Линейная модель с нелинейной функций в конце



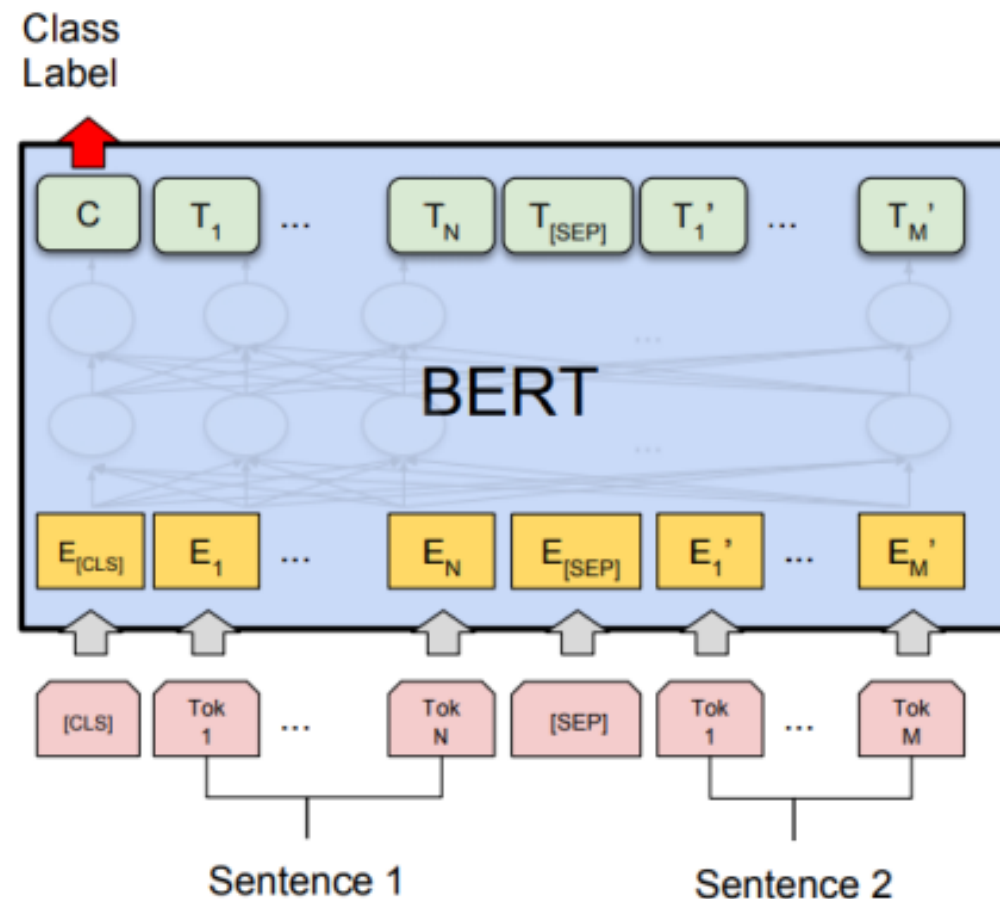
# Декодировщик Transformer

- Получает от encoder векторы слов
- Имеет два внутренних self-attention
- Один для уже сгенерированных слов (такой же, как в encoder)
- Второй - для векторов encoder, чтобы обращать внимание на них
- Опускаем детали, если захочется - обсудим в конце



# Встречаем: BERT

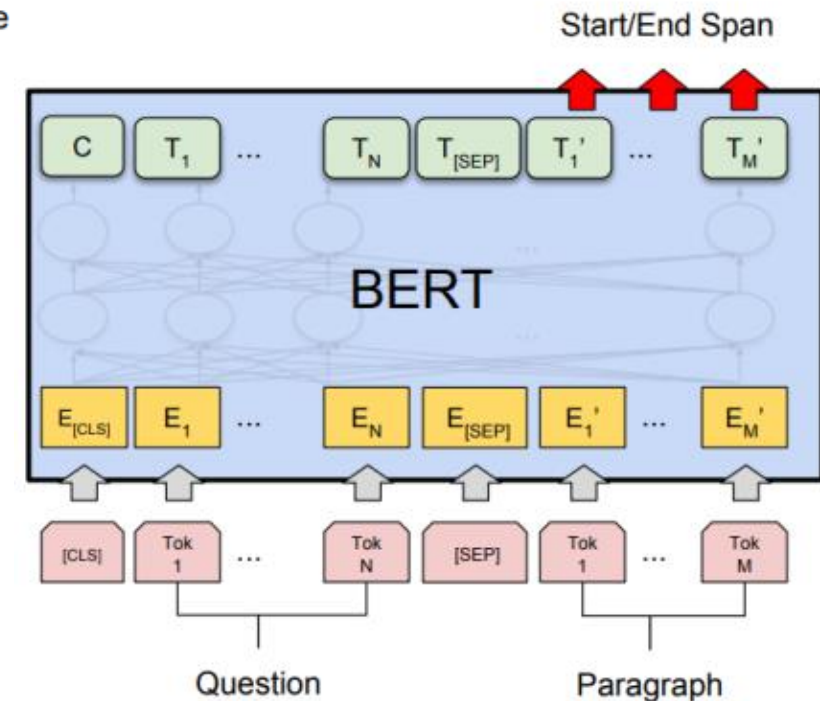
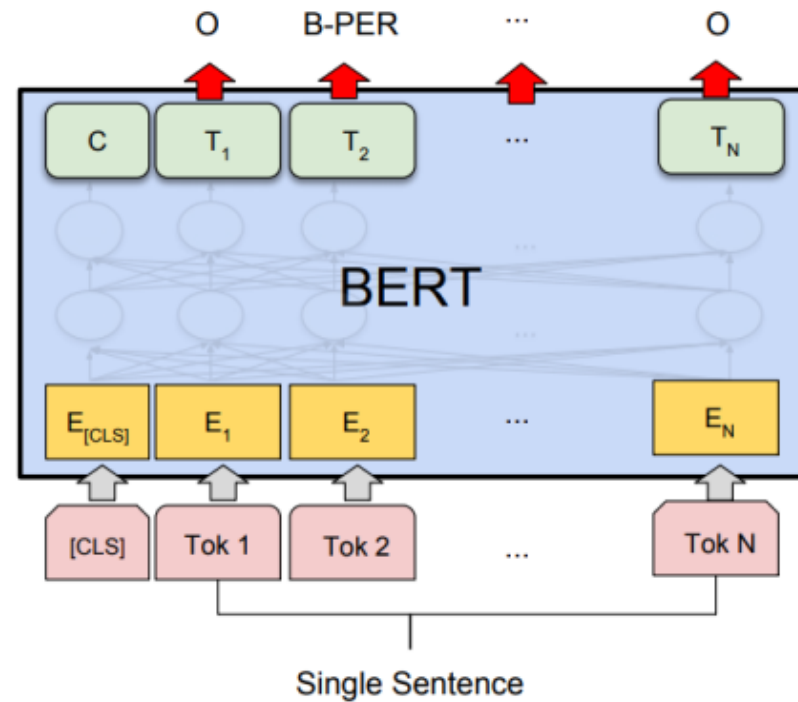
- Модель BERT основана на кодировщике трансформера
- Эту модель учили заполнять пропуски в текстах и определять последовательность двух текстов
- Научившись решать эти задачи, она научилась генерировать хорошие признаки
- С такими признаками проще решать другие задачи (**Transfer learning**), в т.ч. классификацию
- Можно дообучать, можно брать признаки и учить другую модель



# Встречаем: BERT

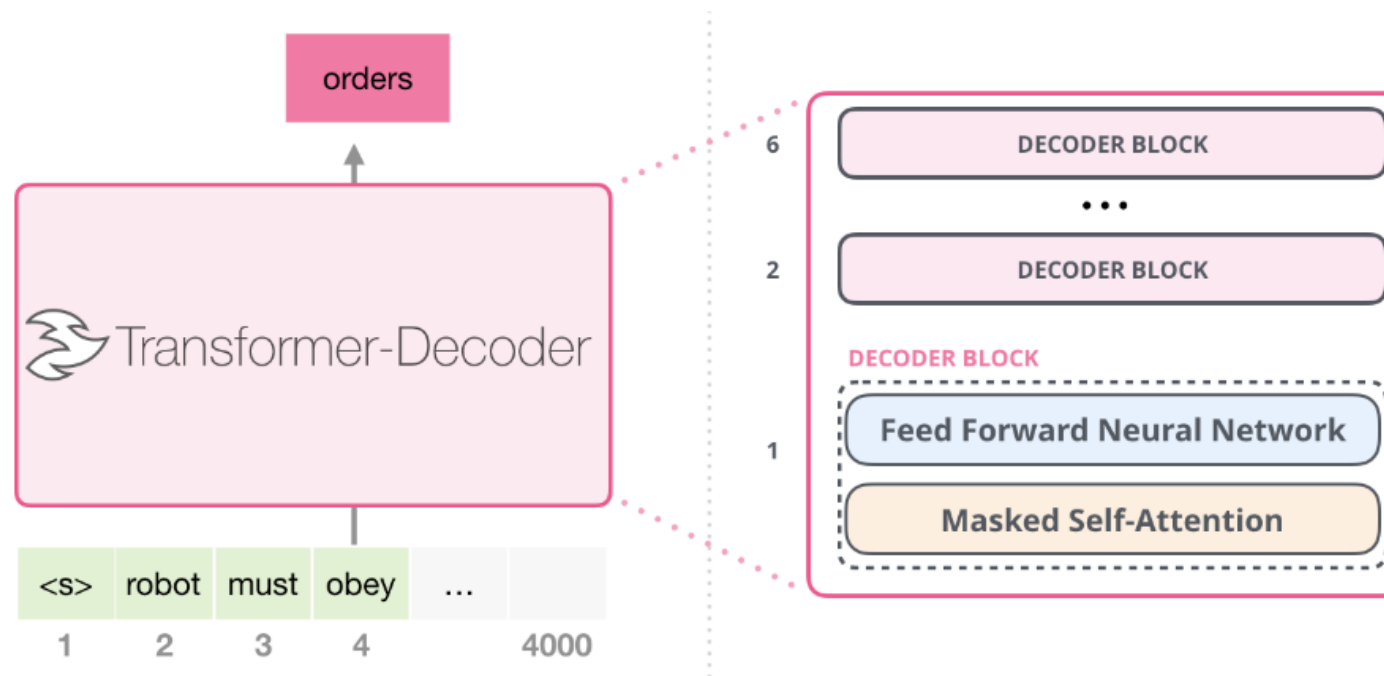
- На архитектуру BERT напрямую хорошо ложатся разные задачи NLP:

- Классификация
- Выделение именованных сущностей (NER)
- Разметка частей речи (POS-теггинг)
- Предсказание спана ответа



# Встречаем: GPT

- Модели GPT-2 и GPT-3 основаны на декодирующей трансформера
- Из них удалён блок self-attention, связанный с кодировщиком (его нет)
- Модели учатся предсказывать слова текста по предыдущим словам
- Часто используются для создания диалоговых генеративных моделей
- Для задач типа парафразы и суммаризации тоже, но там чаще обучают полный трансформер, он стабильнее



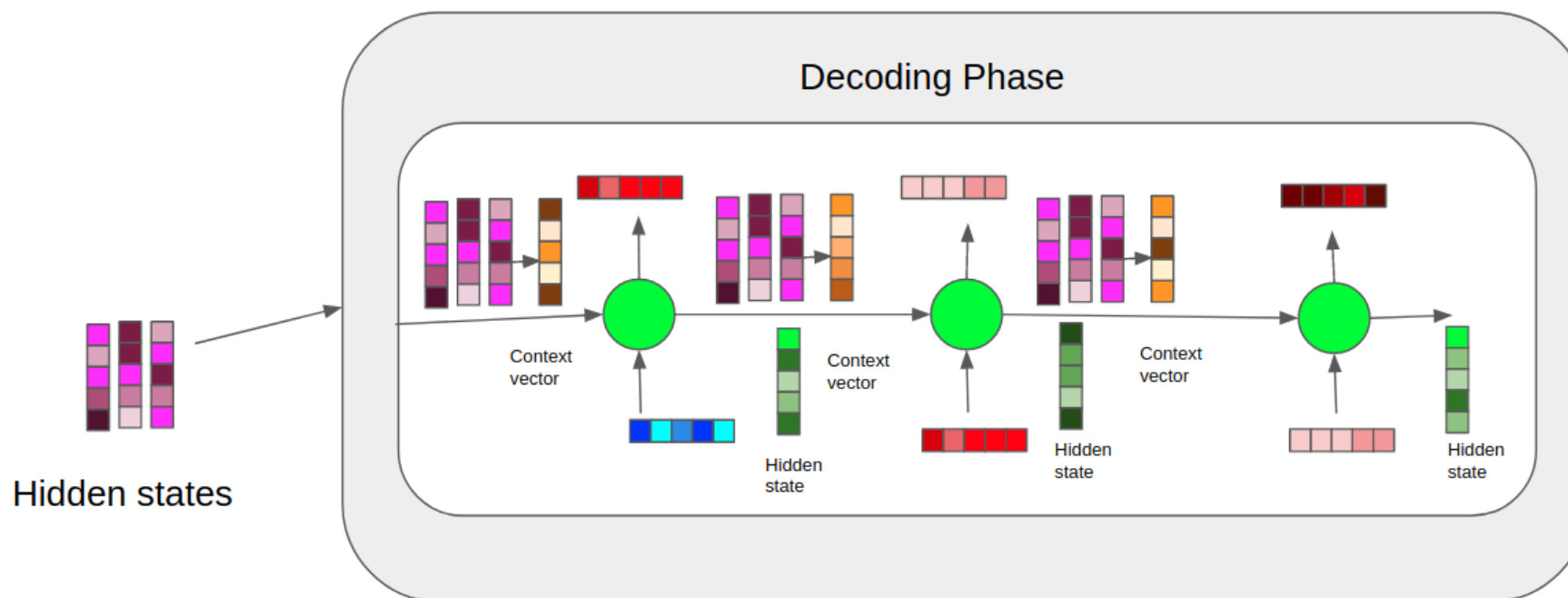
# Итоги рассказа

- Для любых данных, в т.ч. текстов, важно уметь придумывать и считать хорошие признаки
- Моделей, чтобы их обрабатывать, много, но почти все основаны на линейных операциях с нелинейными функциями после
- Мир NLP за 10 лет совершил скачок от традиционных лингвистических и ML подходов к глубоким нейросетям
- Особенно важна возможность Transfer learning
- При этом традиционные подходы все ещё актуально и часто используются вместе с новыми моделями или как идеи для их улучшения

Технические детали

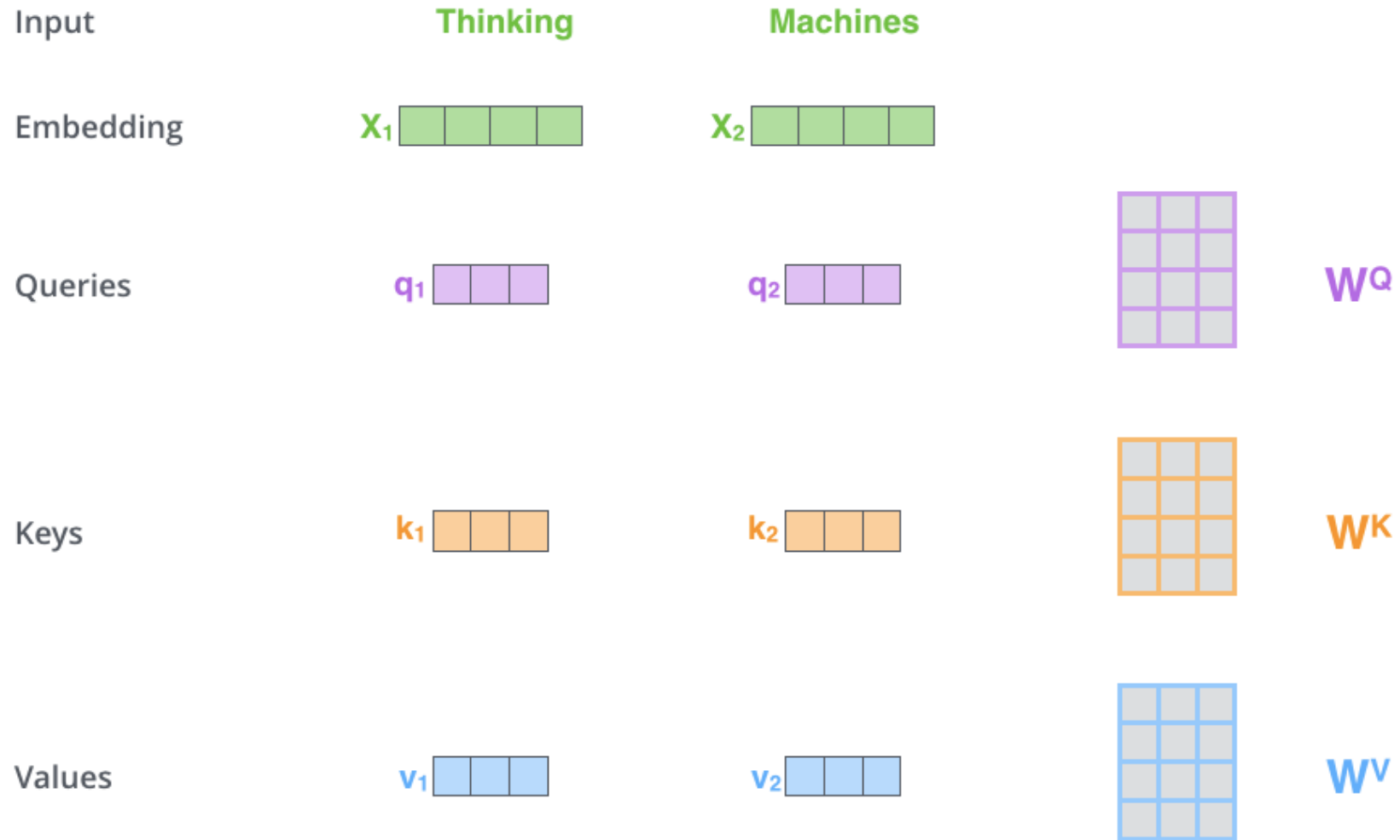
# Механизм внимания в RNN

- Текущей вектор состояния decoder умножается на векторы encoder (можно иначе!)
- Полученные числа нормализуются в 1 - это важности слов входа
- Все векторы encoder складываются с этими весами в один вектор (информация от входа)
- Для генерации этого слова используется **ЭТОТ ВЕКТОР**, **вектор состояния декодировщика** и **вектор предыдущего сгенерированного слова**

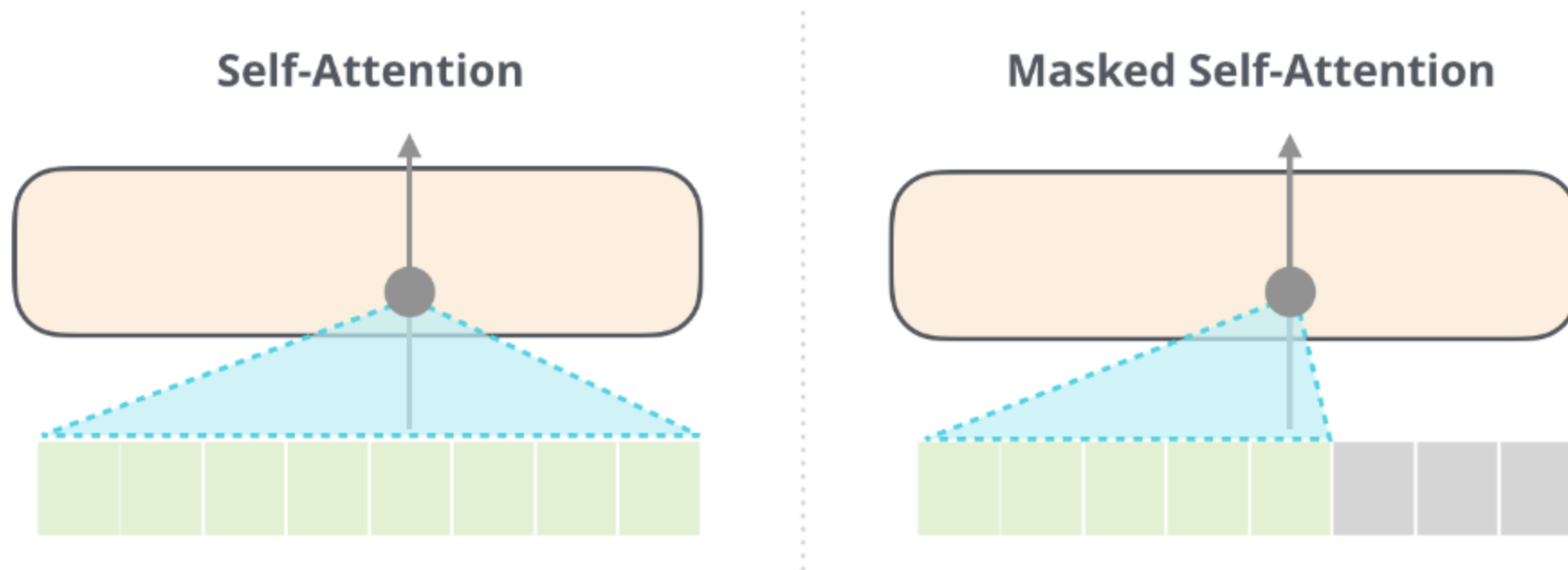




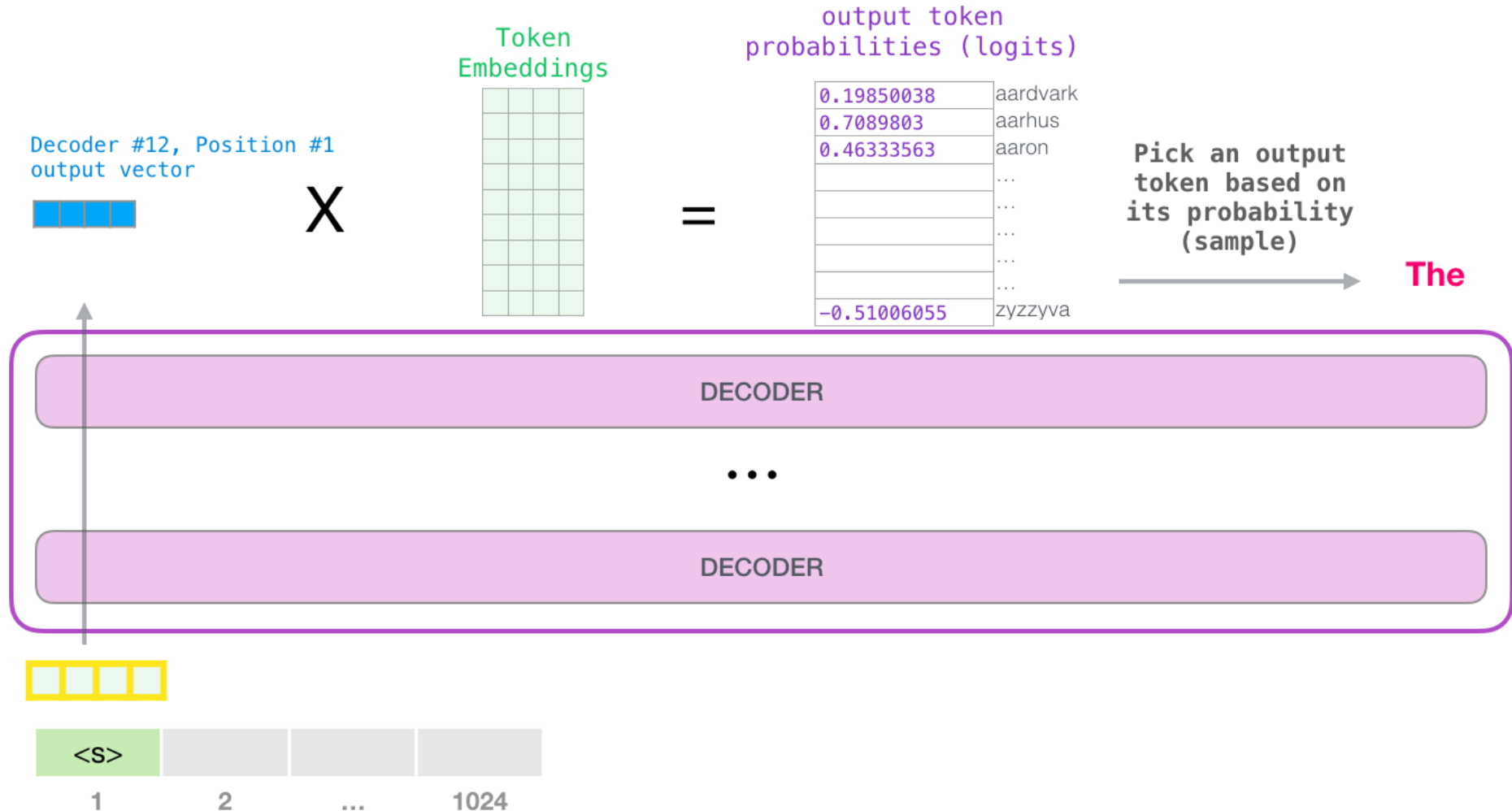
# Механизм Self-Attention



# Маскированный Self-Attention



# Генерация слов по выходу сети



# Генерация слов по выходу сети

Which word in our vocabulary is associated with this index?

am

Get the index of the cell with the highest value (argmax)

5

log\_probs



Softmax



logits



Linear



Decoder stack output

