Математические методы анализа текстов

Нейросетевой машинный перевод. Архитектура Transformer

Мурат Апишев (mel-lain@yandex.ru)

Сентябрь, 2020

Задача машинного перевода

- Задача перевода – массовая, её было бы здорово решать автоматически
- Идея машинного перевода зародилась ещё в 1947 году и к середине 60-х уже появились первые системы
- Несмотря на сложности, область развивается до сих пор, особенно сильный рост качества произошёл в последние годы
- Изначально задача решалась статистическими методами (IBM Model)
- В 2013 году появилась первая полностью нейросетевая модель, с 2016 нейросетевой машинный перевод (NMT) стал индустриальным стандартом
- Современный машинный перевод хорош в ситуациях, где тексты формализованы или же достаточно грубого перевода
- С художественной литературой до сих пор всё плохо

Оценка качества машинного перевода

- Экспертная оценка — даём исходное предложение и перевод модели специалистам, просим оценить по шкале
  - Оценка очень точная
  - Получать её дорого и медленно
- Сравнение с правильным ответом — на тестовом корпусе сравним полученный ответ с одним из возможных переводов
  - Оценка получается быстро (если есть готовый корпус)
  - Корпус размечать сложно, обычно тексты одного жанра, не содержать сленг и неологизмы
- Для автоматической оценки нужно уметь сравнивать две последовательности токенов

Метрика BLEU

- BLEU (bilingual evaluation understudy) — метод сравнения автоматически переведённого предложения с «золотым стандартом»
- Рассмотрим на примере:

  Правильный ответ: *E-mail was sent on Tuesday*

  Ответ системы: *The letter was sent on Tuesday*

- Посчитаем для заданного N (обычно 3-4) число N-грамм в ответе системы, которые присутствуют в правильном ответе (аналог точности):

  $N = 1 \Rightarrow 4/6 \qquad N = 2 \Rightarrow 3/5 \qquad N = 3 \Rightarrow 2/4 \qquad N = 4 \Rightarrow 1/3$

- Теперь посчитаем среднее геометрическое по всем N:

$$\text{score} = \sqrt[4]{4/6 \cdot 3/5 \cdot 2/4 \cdot 1/3}$$

- Вместо подсчёта полноты вводится штраф за краткость (brevity penalty):

$$BP = \min(1, 6/5)$$

- Итоговая значение метрики BLEU: $BP \cdot score \approx 0.5081$

Метрика WER

- WER (word error rate) — минимальное число операций, нужное для преобразования полученного перевода в правильный
- Допустимые операции: замена, вставка, удаление слова
- Значение рассчитывается по формуле

$$WER = \frac{\#\text{insetions} + \#\text{deletions} + \#\text{replacements}}{\#\text{words in translated sentence}}$$

- Особенности метрик BLEU и WER:
  - Легко считаются
  - Неплохо коррелируют с экспертными оценками
  - Оперируют короткими фрагментами, не оценивают общую корректность
  - Не позволяют оценить жанровую специфику
  - Не дифференцируемы

Задача seq2seq для машинного перевода

- $\mathbf{x}_{source} = x_i,\ i \in [1, n],\ x_i \in V_{source}$ — предложение на исходном языке
- $\mathbf{y}_{target} = y_i,\ i \in [1, m],\ y_i \in V_{target}$ — предложение на целевом языке
- $\mathbf{C}$ — параллельный обучающий корпус из пар предложений $\mathbf{x}, \mathbf{y}$
- Решается задача максимизации лог-правдоподобия модели с параметрами θ:

$$\mathcal{L}_\theta = \sum_{\mathbf{x},\mathbf{y} \in \mathbf{C}} \log p(\mathbf{y}|\mathbf{x}; \theta) \to \max_\theta$$

- Предсказываем последовательно каждый целевой токен перевода в зависимости от предыдущих:

$$p(\mathbf{y}|\mathbf{x}; \theta) = \prod_{j=1}^{m} p(\mathbf{y}_j|\mathbf{y}_{<j}, \mathbf{x}; \theta)$$

- Модель можно параметризовать нейронной сетью

Кодировщик-декодировщик

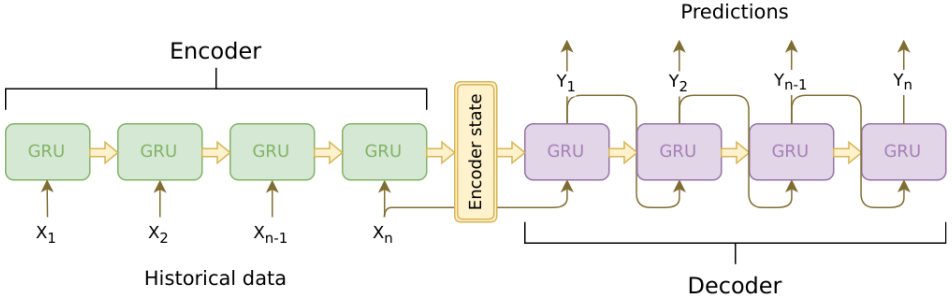
- Показывает хорошие результаты в задачах машинного перевода, суммаризации и генерации подписей к картинкам
- Кодировщик получает на вход последовательность входных элементов $\mathbf{x}_{source}$ и генерирует числовой вектор контекста h_n
- На базе этого вектора декодировщик генерирует выходную последовательность $\hat{\mathbf{y}}_{target}$
- Работа завершается после генерации завершающего токена (<EOS>)
- Функция потерь — отрицательный логарифм правдоподобия (NLL Loss):

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \sum_j y_j \log \hat{y}_j$$

- Архитектуры кодировщика и декодировщика могут быть различными

Кодировщик-декодировщик

- Можно обе сети взять рекурентными (RNN, LSTM, GRU)
- Входные слова x_i кодируются эмбеддингами (word2vec или GloVe)
- Вход декодировщика: вектор итоговый скрытого состояния кодировщика и вектор старта фразы (выделенный или последний из входа)
- На каждом шаге передаётся вектор последнего сгенерированного токена и скрытое состояние декодировщика



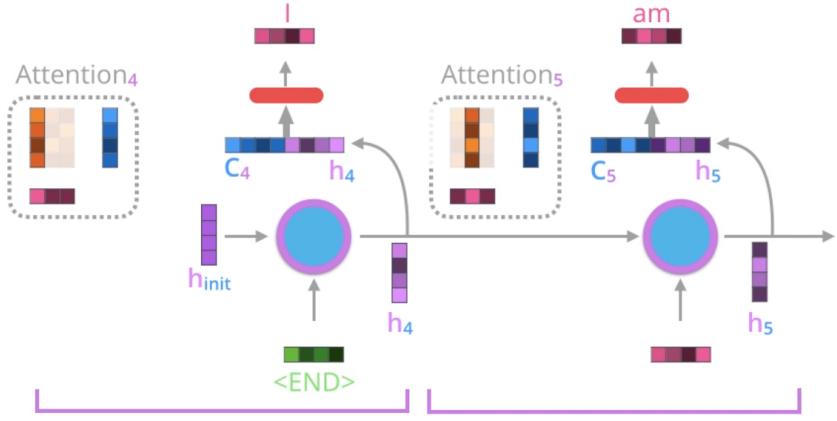
Механизм внимания

- Узким местом описанного подхода является вектор контекста h_n (итоговое состояние RNN-кодировщика)
- Очевидно, что последние слова входной фразы будут оказывать на него большее влияние, чем первые
- Это позволяет переводить только короткие фразы
- Одно из возможных решений — механизм внимания (attention):
  - при ручном переводе слова в предложении мы смотрим не только на само слово, но и на релевантный контекст
  - в то же время, мы игнорируем те части предложения, которые к текущему переводимому слов не относятся
- Кодировщик передаёт декодировщику не последнее значение своего вектора состояния h_n, а все $h_i,\ i \in [1, n]$
- Каждый вектор в наибольшей степени отражает влияние того слова, при обработке которого он был получен

Декодирование с вниманием

При генерации очередного слова декодировщик:

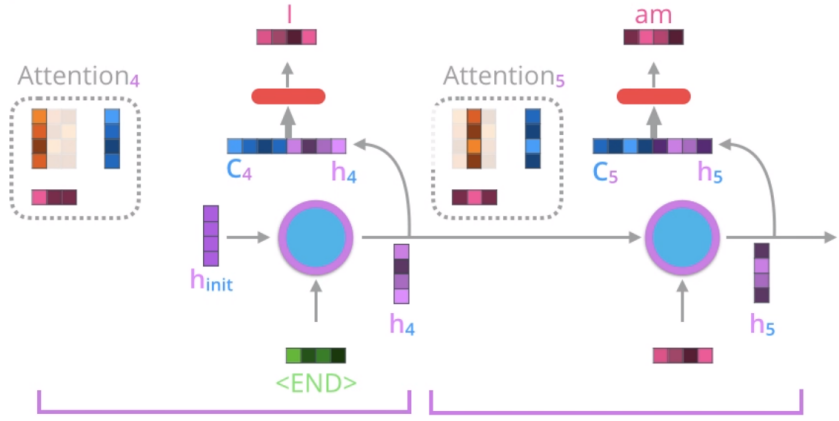
- получает на вход последний вектор своего состояния h_{j-1}, $j \in [n+1, m]$ и вектор последнего сгенерированного слова (или метки старта) $\hat{y}_{j-1}$
- выдаёт новый вектор своего состояния h_j и выходной вектор



Декодирование с вниманием

При генерации очередного слова декодировщик:

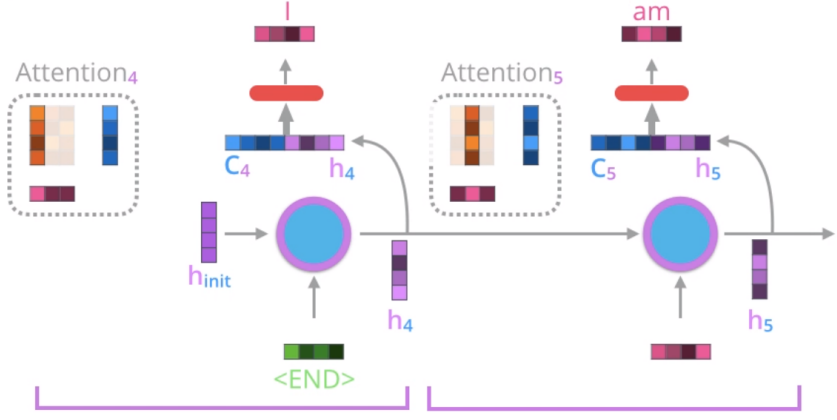
- считает для каждого вектора состояния кодировщика $h_i,\ i \in [1, n]$ вес α_i^j, отражающий его важность при генерации текущего j-го слова
- конкатенирует вектор $\sum_{i=1}^{n} \alpha_i^j h_i$ с новым вектором своего состояния h_j



Декодирование с вниманием

При генерации очередного слова декодировщик:

- подаёт результат конкатенации в общую для модели полносвязную сеть
- результат проходит через Softmax и генерируется слово перевода y_j



Подсчёт весов внимания

- Вес α_{ij} вектора состояния кодировщика h_i при генерации слова j зависит от h_i и вектора скрытого состояния декодировщика h_j:

$$a_{ij} = \frac{\exp(\mathrm{sim}(h_i, h_j))}{\sum_k \exp(\mathrm{sim}(h_k, h_j))}$$

- Считать функцию близости sim можно по-разному:
  - Скалярное произведение:

$$\mathrm{sim}(h_i, h_j) = h_i^T h_j$$

  - Аддитивное внимание:

$$\mathrm{sim}(h_i, h_j) = w^T \tanh(W_{h_i} h_i + W_{h_j} h_j)$$

  - Мультипликативное внимание:

$$\mathrm{sim}(h_i, h_j) = h_i^T W h_j$$

- Параметры весовых функций (при их наличии) обучаются вместе с основной сетью

Вариации работы с вниманием

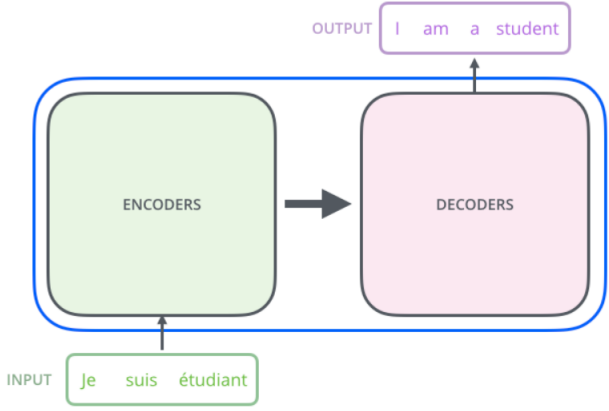
- Использовать внимание до RNN:
  - В примере RNN выдаёт текущее состояние, после чего используется внимание + полносвязная сеть
  - Можно вектор внимания посчитать до RNN на основе состояния с предыдущего шага и выходы RNN использовать напрямую
- Считать веса для внимания на основе разных векторов:
  - В примере «вектор декодировщика + векторы кодировщика»
  - Можно «вектор декодировщика + векторы входных слов»
- Изменять множество рассматриваемых векторов состояния кодировщика:
  - Global Attention — как в примере, работаем на каждом шаге со всеми векторами (качественнее)
  - Local Attention — предсказываем позицию слова и работаем с векторами из фиксированного окна (быстрее)

Архитектура Transformer

- До последнего времени основой для Seq2seq моделей служили рекурентные сети
- Основная проблема в их использовании — большие затраты времени и вычислительных ресурсов на обучение
- В 2017 году была предложена архитектура Transformer, которая полностью откзаывается от рекурентных слоёв в кодировщике и декодировщике
- Вместо этого предлагается использовать новый тип слоя — Multi-head self-attention, работающий исключительно на основе механизма внимания
- Transformer превзошёл имеющиеся на тот момент архитектуры на основе LSTM и GRU как по качеству решения (в т.ч. и в переводе), так и по скорости обучения

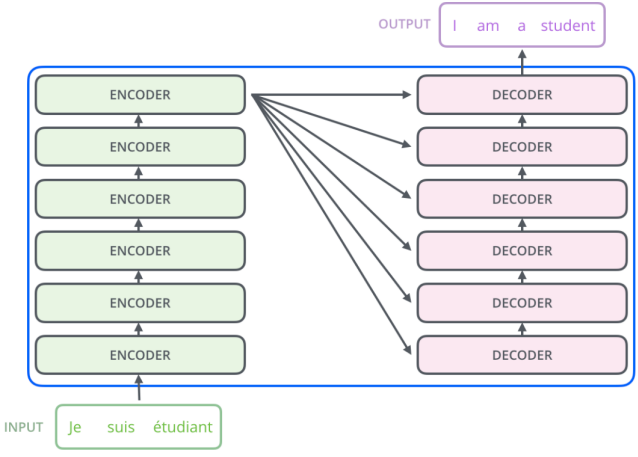
Transformer сверху вниз

- По указанной ссылке находится одно из наиболее подробных и доступных объяснений Transformer, будем следовать ему
- Верхнеуровнево это всё тот же кодировщик-декодировщик



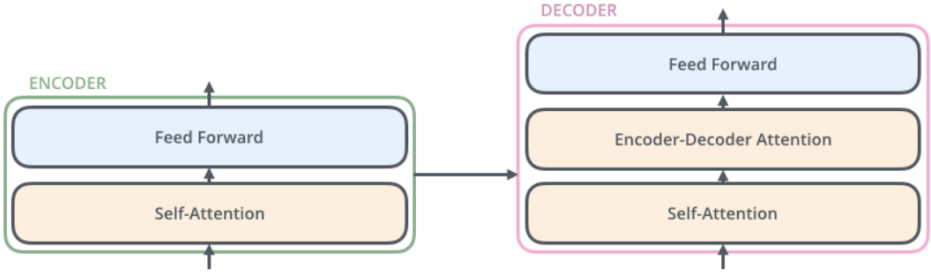
Transformer сверху вниз

- Кодировщик и декодировщик состоят из своих наборов одинаковых блоков, блоки стекаются друг за другом
- В оригинальной статье блоков 6, но это не принципиально
- Веса у каждого блока свои (т.е. неразделяемые)



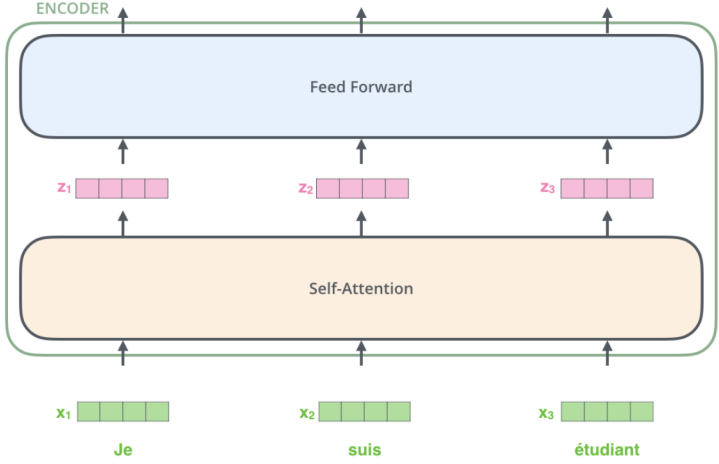
Transformer сверху вниз

- Первый слой кодировщика — self-attention, который кодирует каждое слово последовательности с учётом остальных (рассмотрим далее)
- Затем выход self-attention для каждого элемента последовательности проходит через одну и ту же полносвязную сеть
- Декодер дополнительно к этим слоям имеет слой обычного внимания для работы с последовательностью выходов кодировщика

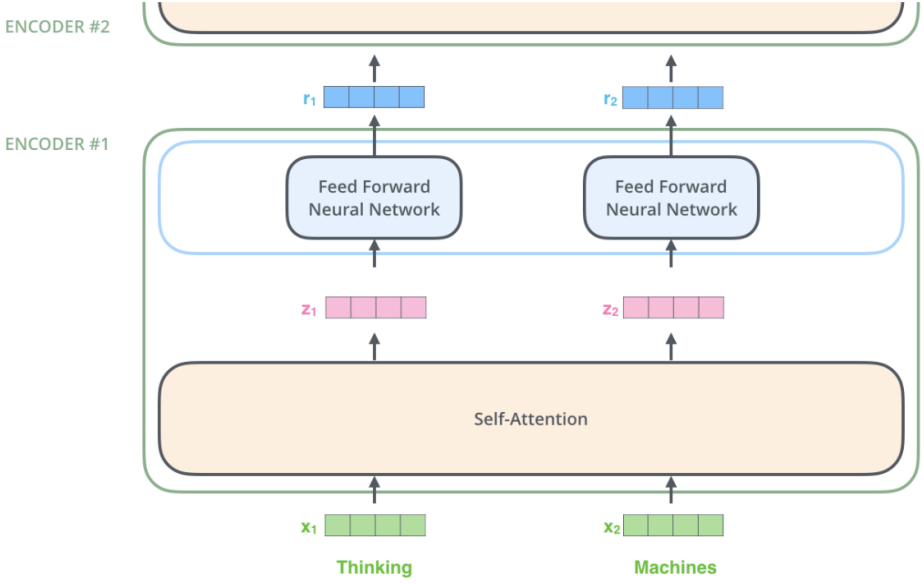


Transformer сверху вниз

- На вход первого кодировщика приходят эмбеддинги слов, остальные получают выходы предшественников
- Слова последовательности обрабатываются взаимнозависимо в слое self-attention и независимо в полносвязном, всё параллелится

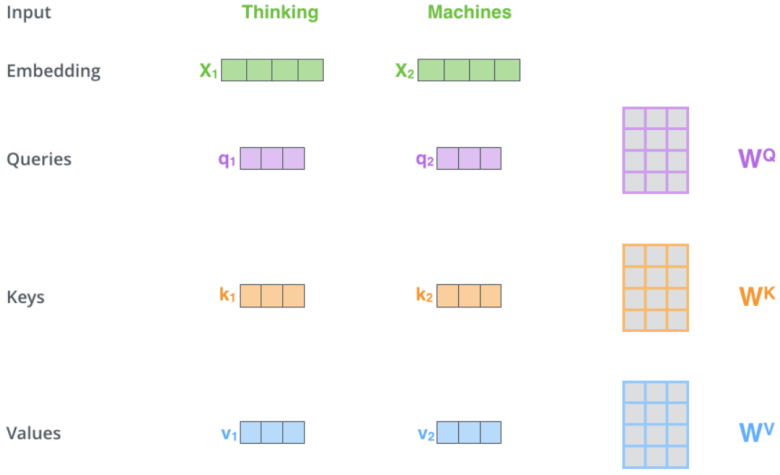


Transformer сверху вниз



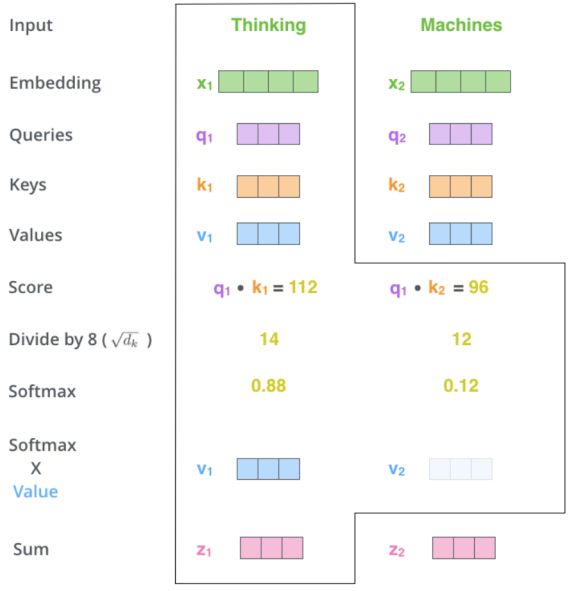
Слой self-attention

- Для каждого входного вектора считаются три новых: `Key`, `Value` и `Query`
- Матрицы преобразований обучаются вместе с сетью

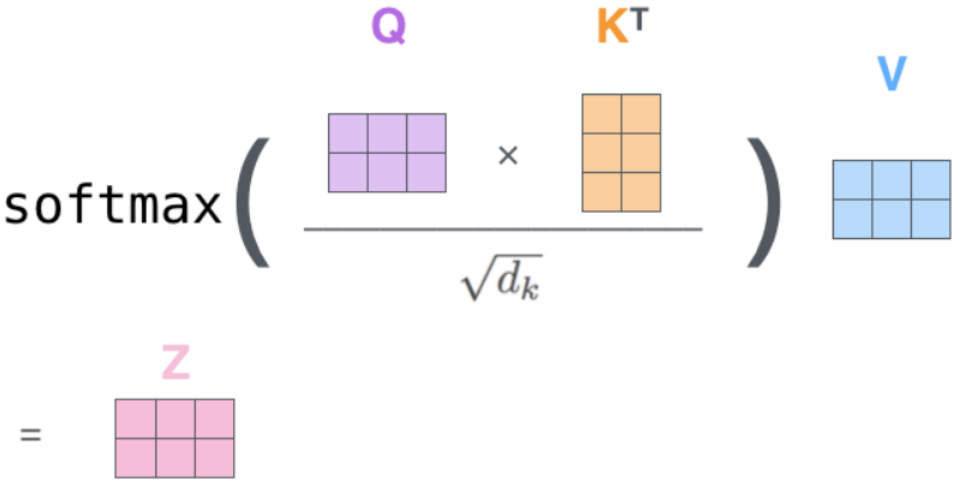


Слой self-attention

- Как и в обычном внимания, нужно сгенерировать вектор для слова с учётом всей последовательности
- Обработка текущего слова:
  - `Query` для текущего слова скалярно умножаем на векторы `Key` всех входных слов, получаем веса
  - делим все веса на некоторую константу (для стабильности градиентов), пропускаем через Softmax
  - складываем все векторы `Value` с полученными весами — получаем итоговый вектор для слова в контексте последовательности

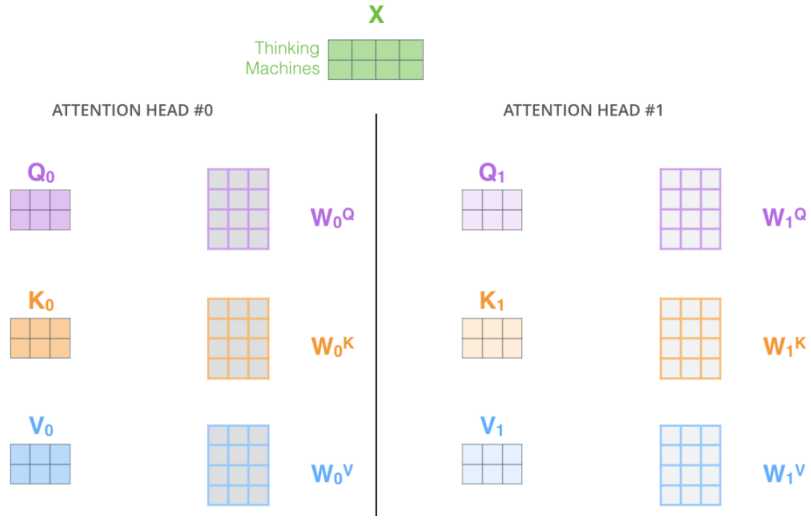


Self-attention в матричном виде



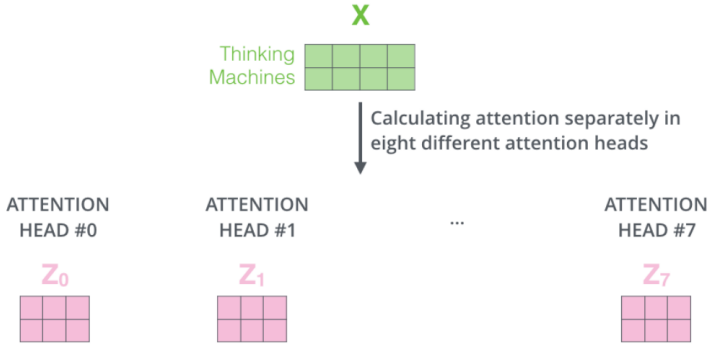
Multi-head self-attention

- Идея: для каждого слова вычислять параллельно сразу несколько векторов self-attention



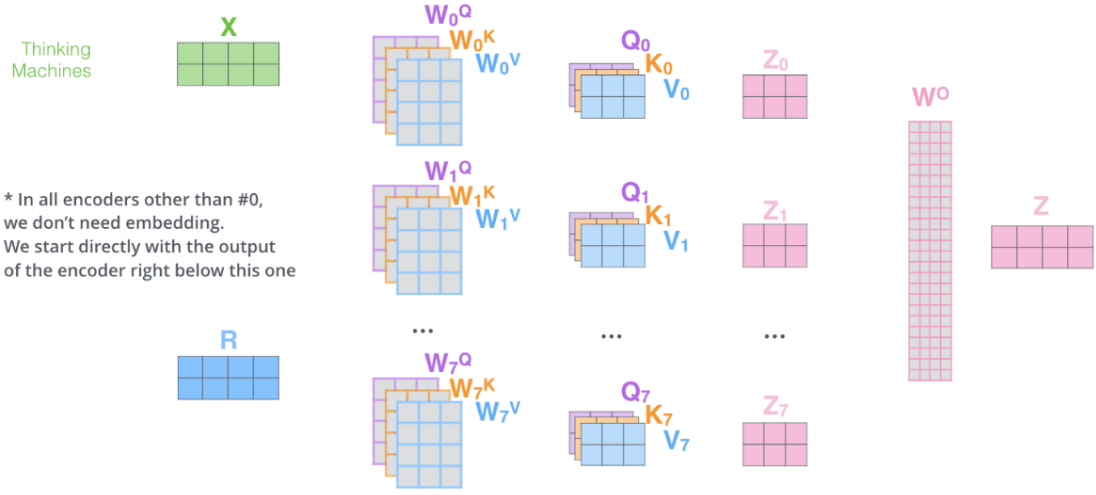
Multi-head self-attention

- Эксперименты показывают, что матрицы весов, инициализированные по-разному, выделяют различные аспекты слова в последовательности
- На выходе получается несколько матриц векторов для одной входной последовательности токенов
- Перед подачей в полносвязную сеть они конкатенируются и умножаются на промежуточную весовую матрицу (W_0) для сохранения размерности



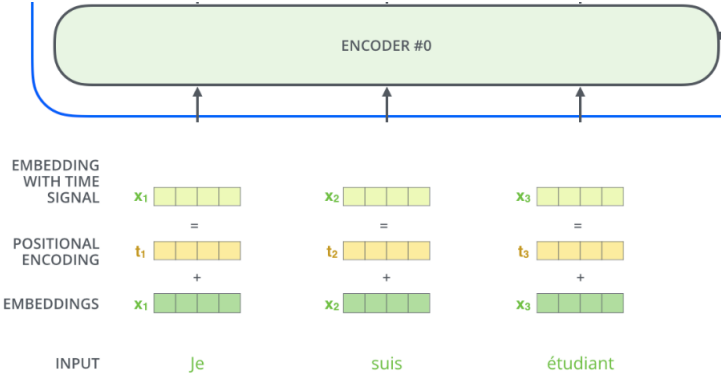
Общая схема Multi-head self-attention

На входе первого слоя эмбеддинги X, далее — предыдущие выходы R



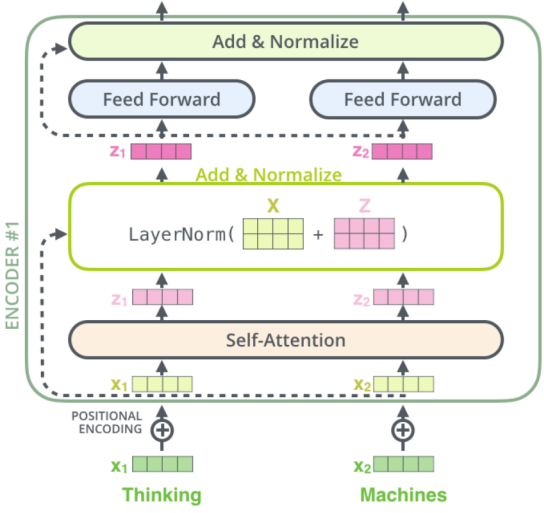
Positional encoding

- Позиционное кодирование (positional encoding) – способ передачи информации о взаимном расстоянии между словами в последовательности через их векторные представления
- Для этого к вектору слова прибавляется вектор из набора значений синусов и косинусов с разными периодами от номера позиции слова в последовательности



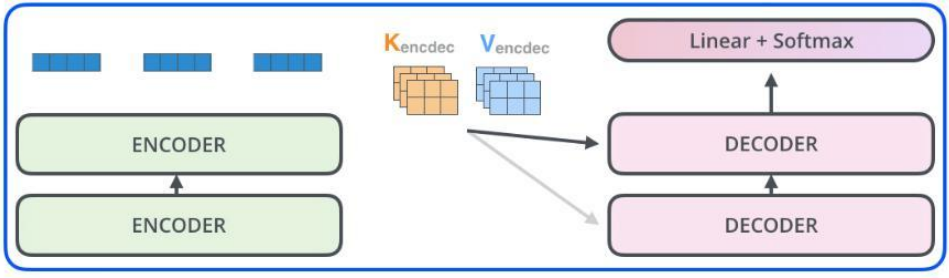
Детали устройства кодировщика

- Для борьбы с затуханием градиента добавляются residual connections
- Для регуляризации используется Layer normalization



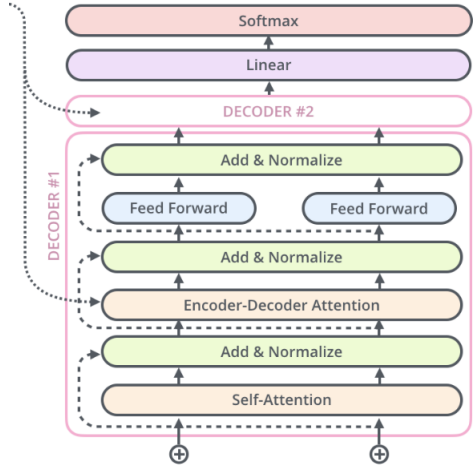
Сеть-декодировщик

- Сеть-декодировщик состоит из последовательных блоков-декодировщиков
- Выходы последнего кодировщика преобразовываются обучаемыми весовыми матрицами в набор матриц `Key` и `Value`
- Эти матрицы передаются в каждый из блоков-декодировщиков



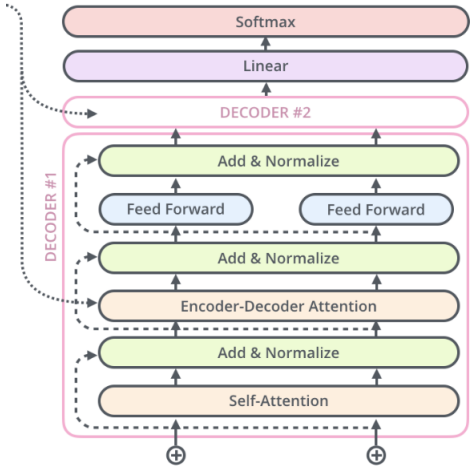
Блок-декодировщик

- Блок декодирования похож на блок кодирования
- В начале его идёт слой self-attention, на входе — токены генерируемой последовательности
  - При обучении используется teacher forcing — на вход подаётся вся правильная целевая последовательность
  - При выводе подаются уже сгенерированные ранее токены
- При обучении токены, которые при генерации текущего токена ещё неизвестны, маскируются
- Они не участвуют в self-attention для текущего токена



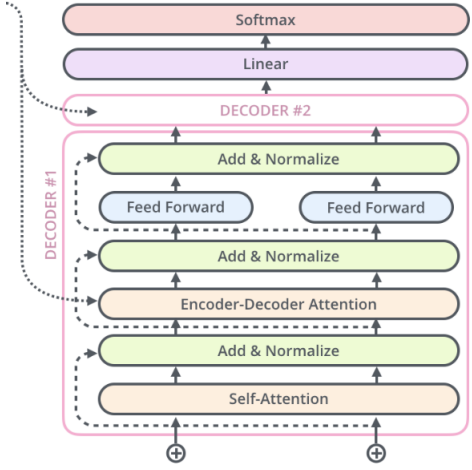
Блок-декодировщик

- Выходы первого слоя Multi-head self-attention идут во второй — Encoder-Decoder Attention
- Он соединяет информацию от первого слоя и кодировщика
  - Выходы кодировщика — набор матриц `Key` и `Value`
  - Выхода первого слоя блока декодировщика — матрица Query
- Снова используется Multi-head self-attention, выходы слоя подаются в полносвязный слой
- На выходе блока — набор векторов, соответствующих токенам входной последовательности



Сеть-декодировщик

- Последний блок выдал векторы
- При обучении:
  - каждый из них пропускается через полносвязный слой и Softmax
  - все векторы обрабатываются одновременно
  - максимизируется вероятность ожидаемого слова
- При выводе:
  - всё в декодировщике считается только вектора предсказываемого токена
  - предсказание можно делать с помощью argmax, Beam Search или сэмплирования



Проблема OOV-слов

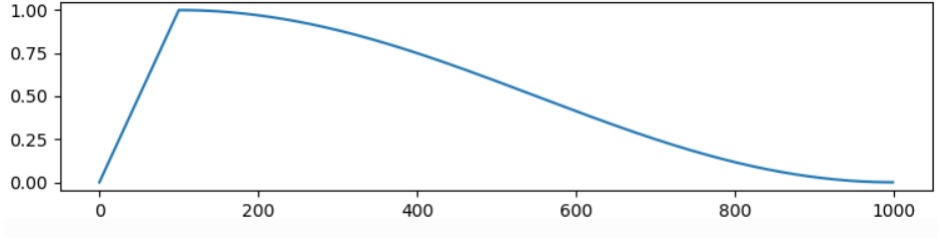
- Softmax на последнем слое — существенное ограничение
- Добавление нового генерируемого слова требует полного переобучения последнего полносвязного слоя
- Можно добавить специальный токен <UNK> для незнакомых слов и переводить их по обычному словарю
- Можно строить модели не на словах, а на уровне символов или символьных последовательностей
- Архитектура модели не меняется, предсказываются фрагменты слов с символом/токеном конца слова
- Символьных последовательностей может быть очень много, один из методов сокращения их числа — Byte-pair encoding (BPE)

Byte-pair encoding

- Предварительно текст разбивается на токены-слова
- Далее каждое слово разбивается на символы, общее число уникальных символов — базовый словарь
- Выполняется N операций слияния, на каждом шаге наиболее частая пара токенов-фрагментов (символов или их последовательностей) сливается в один
- Число N позволяет контролировать объём итогового словаря
- Векторные представления фрагментов можно предобучать заранее, обычно они обучаются вместе с моделью
- Для того, чтобы работать со всеми символами из unicode без использования огромного словаря применяется Byte-level BPE

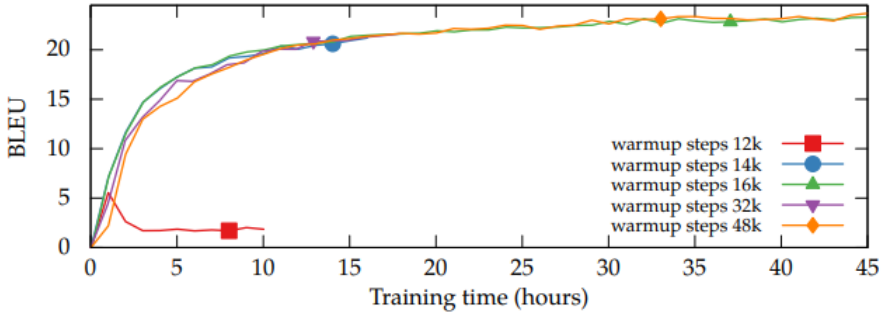
Warm-up learning rate

- Одна из частых проблем при обучении Transformer — расхождение (divergence): кривая обучения нормально растёт, но в некоторый момент резко падает в ноль и не больше поднимается
- Для борьбы с этим обычно применяется warm-up learning rate:
  - на первых батчах используется небольшой lr
  - затем он линейно растёт заданное число батчей (warm-up period)
  - с некоторого момента lr начинает убывать



Методы борьбы с расхождением

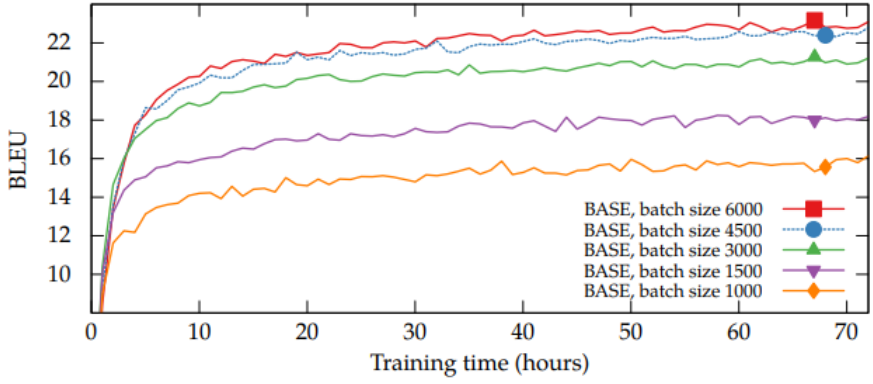
- Более длинный (warm-up period) повышает шансы избежать расхождения



- При этом расширяется диапазон используемых значений learning rate, что ускоряет сходимость модели
- Если расхождение всё равно происходит, стоит уменьшать learning rate
- Ещё вариант — gradient clipping и/или gradient scaling

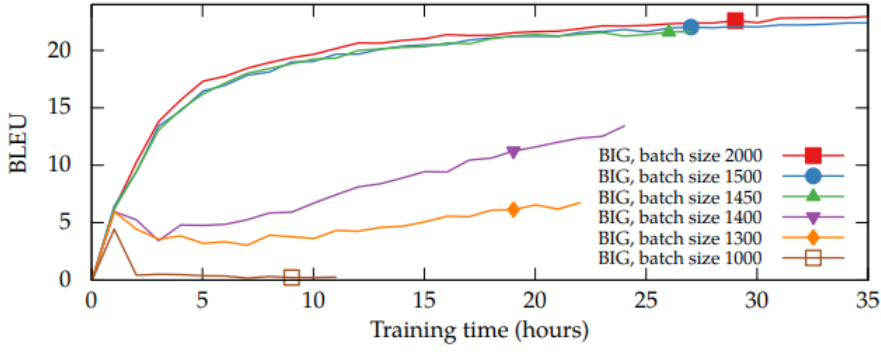
Влияние частоты обновления весов

- Между шагами обновления весов модели должно быть обработано достаточно обучающих примеров
- Иначе для моделей поменьше ухудшится сходимость:



Влияние частоты обновления весов

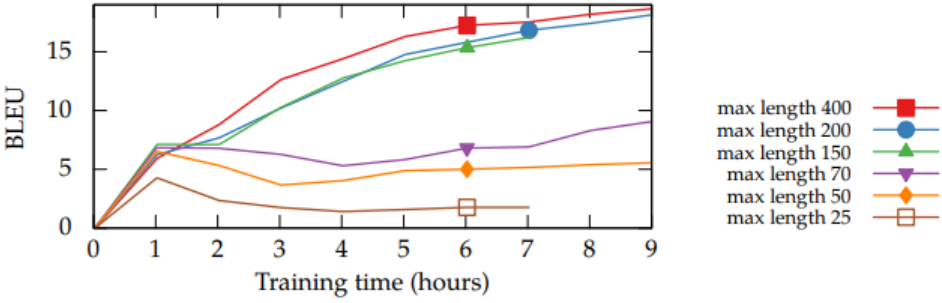
- Между шагами обновления весов модели должно быть обработано достаточно обучающих примеров
- Иначе для больших моделей может произойти расхождение:



- Если расхождения у большой модели не произошло, то дальнейший рост числа объектов между обновлениями несущественен

Влияние длин предложений

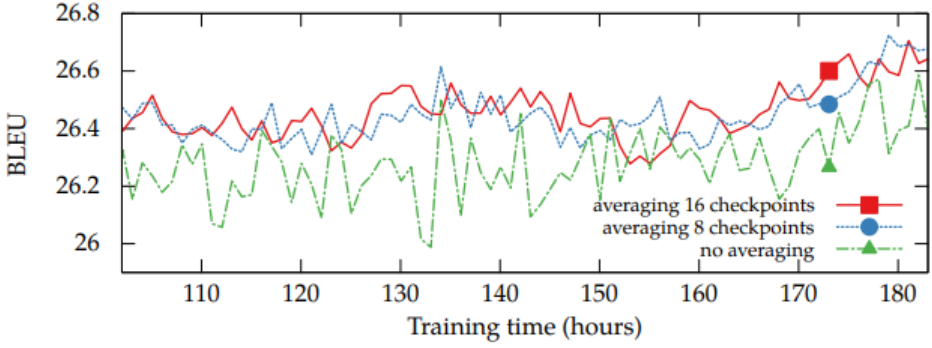
- Максимальную длину предложений в батчах стоит брать побольше
- Иначе модель будет хуже:



- Кроме того, она не сможет генерировать длинные переводы

Усреднение чекпойнтов

- Большой Transformer обучается долго (дни, недели), полезно периодически сохранять состояние модели
- Если вместо последней версии модели взять усреднение весов нескольких последних версий, можно бесплатно получить небольшое улучшение качества:



Неавторегрессионный машинный перевод

- До сих пор рассматривался авторегрессионный подход — генерация текущего токена зависит от предыдущих:

$$p(\mathbf{y}|\mathbf{x};\theta) = \prod_{j=1}^{m} p(\mathbf{y}_j|\mathbf{y}_{<j},\mathbf{x};\theta)$$

- Неавторегрессионный перевод предполагает параллельную генерацию всех токенов целевой последовательности
- Наивный вариант — убрать из модели зависимости между токенами:

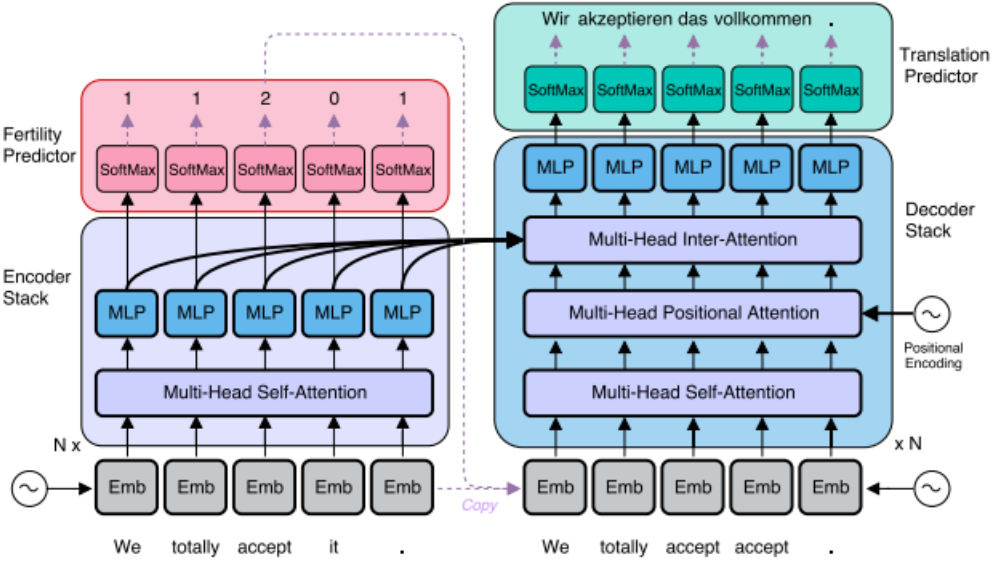
$$p(\mathbf{y}|\mathbf{x};\theta) = p(m|\mathbf{x};\theta)\prod_{j=1}^{m} p(\mathbf{y}_j|\mathbf{x};\theta)$$

- Работать будет плохо: перевод фразы несколькими переводчиками по одному слову независимо друг от друга

Non-Autoregressive Transformer

- В основе модель Transformer, сеть-кодировщик остаётся без изменений
- Сеть декодировщик получает на вход исходную последовательность
- Но число выходов должно быть равно числу входов
- Сопоставим каждому токену входной последовательности целое число fertility — сколько токенам выходной последовательности он должен соответствовать
- Будем предсказывать эти числа на выходе кодировщика
- Каждый токен исходной последовательности будем подавать на вход декодеровщика N раз (его значение fertility)
- В блок-декодировщика между двумя слоями self-attention добавляется + один:
  - Value — выходы первого слоя внимания
  - Key и Query — позиционные эмбеддинги входной последовательности
- Обычного маскирования нет, но есть маскирование токена от самого себя

Non-Autoregressive Transformer



Данные для машинного перевода

- Обычно данные для NMT представляют собой наборы пар фрагментов на разных языках:
  - пары предложений-переводов с выравниванием по словам
  - просто пары предложений-переводов
  - пары абзацев/документов-переводов
- Примеры популярных мультиязычных корпусов (parallel corpus):
  - Europarl: параллельные предложения на 21 языке
  - Wikipedia: параллельные предложения на 20 языках
  - Global Voices: параллельные тексты на 57 языках
- На самом деле их много, в том числе для локальных языков

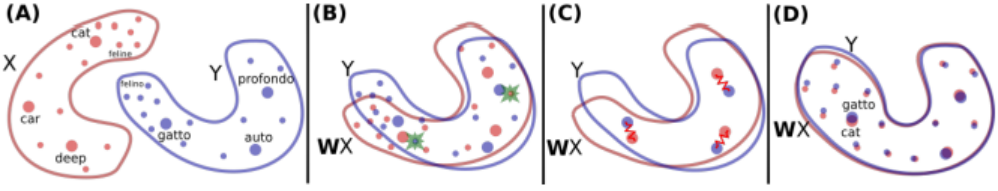
Машинный перевод без учителя

- Параллельные данные можно сгенерировать, для этого нужны:
  - Алгоритм перевода между парой языков для некоторого числа популярных слов
  - Языковая модель для целевого языка
  - Векторные представления слов для обоих языков
- Основная идея:
  - Составляем автоматический словарь между двумя языками
  - Для исходной реплики генерируем варианты переводов по токенам
  - С помощью языковой модели выбираем наиболее вероятный перевод на целевом языке
- Почему словарь автоматический, а не статический:
  - Большой словарь для произвольной пары языков сложно построить
  - Его сложно держать в полностью актуальном состоянии
  - Он неустойчив к опечаткам или небольшим изменениям форм слов

Построение автоматического словаря

Обучаются поворот пространства векторов слов целевого языка и его наложение на пространство векторов исходного языка:

- Построение векторных пространств отдельно для обоих языков
- Выбор опорных точек — пар слов с известным переводом
- Поворот и растяжение пространства для совпадения опорных точек
- Растяжение плотных областей вокруг частых слов



Итоги занятия

- Машинный перевод — одна из флагманских задач обработки текстов, многие сильные методы NLP появились в процессе её решения
- Текущим стандартом является архитектура seq2seq на основе рекурентных сетей или Transformer
- Качество работы с длинными последовательностями существенно растёт при использовании механизмов внимания
- Модель Transformer основана на новом типе слоя — Multi-head self-attention, который является развитием механизма внимания
- Transformer сейчас одна из наиболее сильных и универсальных моделей, но обучается сложно, важны технические детали
- Перевод может авторегрессионным и неавторегрессионым, второй быстрее при выводе, модели в обоих случаях похожи
- Для обучения нужны параллельные корпуса, иногда их можно генерировать автоматическими методами