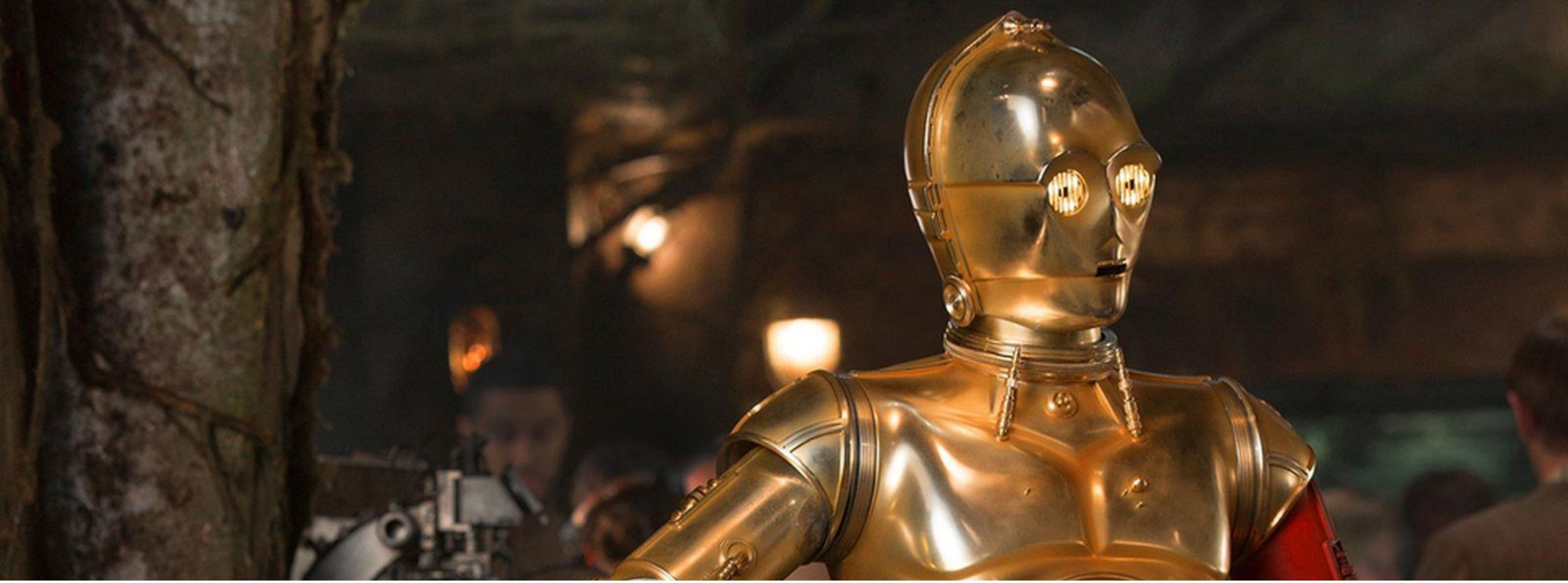


Задача преобразования последовательности. Машинный перевод. Трансформеры.

Попов Артём, OzonMasters, осень 2022

Natural Language Processing

Часть 2

Задача преобразования последовательности

Дано множество пар последовательностей (x, y):

- $x = (x_1, \dots, x_n), x_i \in X$ – входная последовательность
- $y = (y_1, \dots, y_m), y_i \in Y$ – выходная последовательность

Необходимо по входной последовательности предсказать элементы выходной последовательности.

1. Длины x и y не совпадают
2. Нет никаких известных связей между элементами x и y

Критерии качества: BLEU, WER

Архитектура кодировщик-декодировщик (encoder-decoder)

Задача решается методом максимизации правдоподобия:

$$\log p_\theta(y|x) = \log \prod_{i=1}^{m} p_\theta(y_i|x, y_{<i}) = \sum_{i=1}^{m} \log p_\theta(y_i|x, y_{<i}) \to \max_\theta$$

Кодировщик получает на вход последовательность входных элементов x и кодирует информации в последовательности векторов H_{enc}

Декодировщик по уже сгенерированным токенам и вектору контекста итеративно генерирует следующие токены

Sutskever et al (NIPS 2014); Sequence to Sequence Learning with Neural Networks

Cho et al (EMNLP 2014); Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation

Разница в обучении и применении

Обучение
По последовательности $[x_1, \dots, x_n, y_0, \dots, y_m]$ восстанавливаем последовательность $[y_1, \dots, y_{m+1}]$

Применении
По последовательности $[x_1, \dots, x_n, y_0, \hat{y}_1, \dots, \hat{y}_j]$ предсказываем следующий токен $\hat{y}_{j+1}$, пока не получим $< END >$ или не превысим заданное максимальное число токенов

Кодировщик-декодировщик на основе RNN

RNN-RNN

- RNN-энкодер кодирует входную последовательность в один вектор h_n.
- RNN-декодер использует h_n при пересчёте внутренних состояний.

RNN-RNN + attention

- RNN-энкодер кодирует входную последовательность в последовательность векторов H.
- На основе механизма внимания на каждом шаге RNN-декодера вычисляется вектор контекста c_j
- RNN-декодер использует c_j при пересчёте внутренних состояний

Модель трансформер (transformer)

Идея: в каждой позиции хотим давать сети информацию обо всех элементах последовательности.

Составляющие трансформера:

- механизм self-attention
- позиционные представления (positional encoding)
- нормализация

На прошлой лекции рассмотрели устройство кодировщика.

[Vaswani et al (2017); Attention is all you need]

Механизм self-attention

1. Для каждого элемента входа $(x_1, \dots, x_n)$, $x_i \in \mathbb{R}^d$ строим эмбеддинги запроса, ключа и значения, $W_k, W_q, W_v \in \mathbb{R}^{d \times m}$ – обучаемые параметры:

$$q_i = W_q x_i, \qquad k_i = W_k x_i, \qquad v_i = W_v x_i$$

2. Считаем близости между “запросами” и “ключами”

$$sim(q_i, k_j) = \frac{\langle q_i, k_j \rangle}{\sqrt{m}}, \qquad \alpha_{ij} = \underset{j \in \{1,\dots,n\}}{softmax}\, sim(q_i, k_j) = \frac{exp\left(sim(q_i, k_j)\right)}{\sum_{s=1}^{n} exp\left(sim(q_i, k_s)\right)}$$

3. Вычисляем выпуклую комбинацию значений v

$$c_i = \sum_{j=1}^{n} \alpha_{ij} v_j\,, \qquad c = (c_1, \dots, c_n) = SA\left(x; W_q, W_k, W_v\right), \qquad c_i \in \mathbb{R}^m$$

Алгоритм работы multi-head self-attention (MHSA)

Последовательность $(x_1, \dots, x_n),\ x_i \in \mathbb{R}^d$ преобразуем в последовательность $(y_1, \dots, y_n),\ y_i \in \mathbb{R}^d$.

Параметры слоя ($\boldsymbol{\theta}$): N преобразований $W_k^j, W_q^j, W_v^j \in \mathbb{R}^{d \times m}$, линейное преобразование $W \in R^{Nm \times \mathrm{d}}$

1. Вычисляем по всем наборам параметров self-attention
   $$c^j = SA\left(x; W_k^j, W_q^j, W_v^j\right)$$

2. Конкатенируем и пропускаем через ещё один слой
   $$y_i = MHSA(x;\theta)_i = \left[c_i^1, \dots, c_i^N\right] W$$

Позиционные представления

Проблема. Механизм self-attention никак не учитывает порядок элементов в последовательности.

Решение. Добавить информацию о порядке при помощи позиционных эмбеддингов

$$x_i^{new} = x_i + p_i$$

Способы реализации

- фиксированные позиционные эмбеддинги
- обучающиеся позиционные эмбеддинги
- относительные позицонные эмбеддинги (relative encoding)

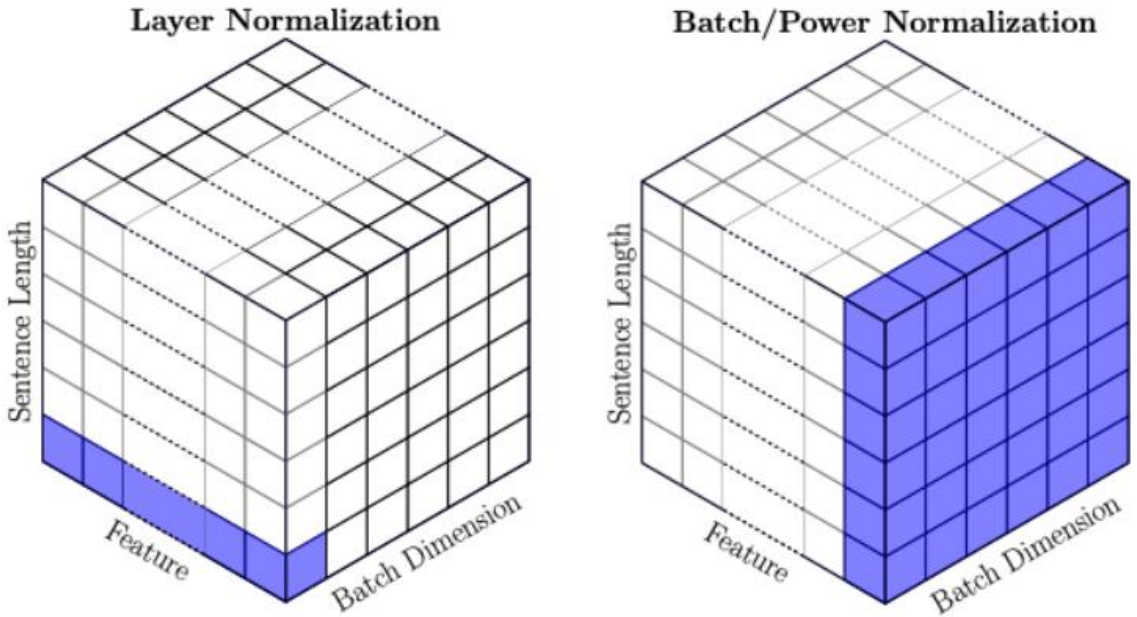
Layer normalization (нормализация слоя)

$$LN(x;\mu,\sigma) = \left\{\sigma_j \frac{x^j - \mu_x}{\sigma_x} + \mu_j\right\}_{j=\overline{1..d}}$$

$$\mu_x = \frac{1}{d}\sum_{j=1}^{d} x_j$$

$$\sigma_x^2 = \frac{1}{d}\sum_{j=1}^{d}\left(x_j - \mu_x\right)^2$$

$$x, \mu, \sigma \in \mathbb{R}^d$$



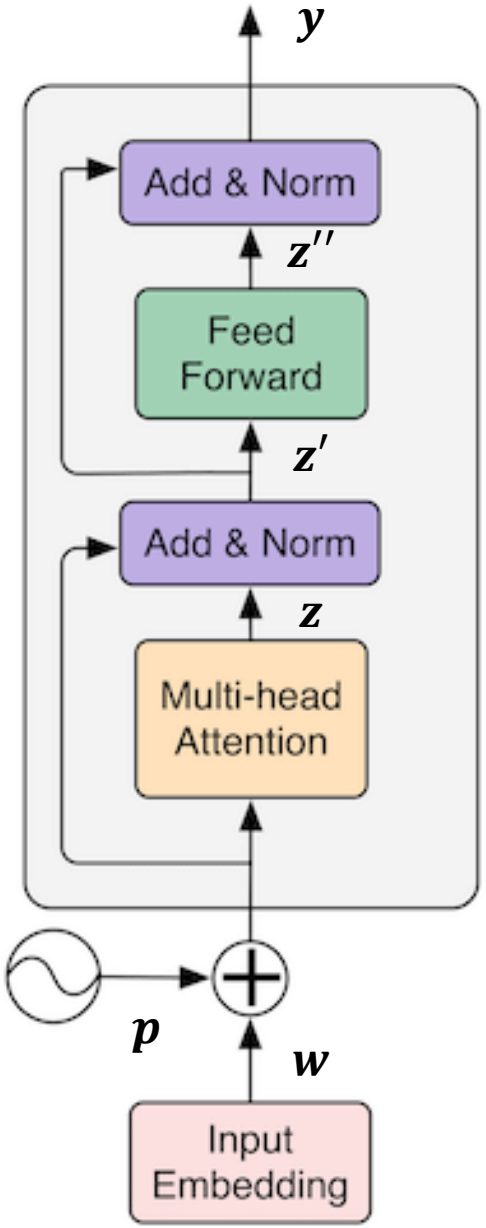
Почему LN, а не BN?

Для распараллеливания по элементам последовательности.

Энкодер трансформера

1. Перед первым слоём складываем представления токенов и позиций
   $x_i = Emb(w_i) + p_i$
2. Применяем MHSA, l – номер слоя
   $\mathrm{z} = MHSA(x; \theta_l)$
3. Residual связи + нормализация слоя
   $z_i' = LN(z_i + x_i; \mu_l^1, \sigma_l^1)$
4. Дополнительные Feed-Forward слои
   $z_i'' = RELU(z_i' V_1 + b_1)V_2 + b_2$
5. Residual связи + нормализация слоя
   $y_i = LN(z_i'' + z_i'; \mu_l^2, \sigma_l^2)$

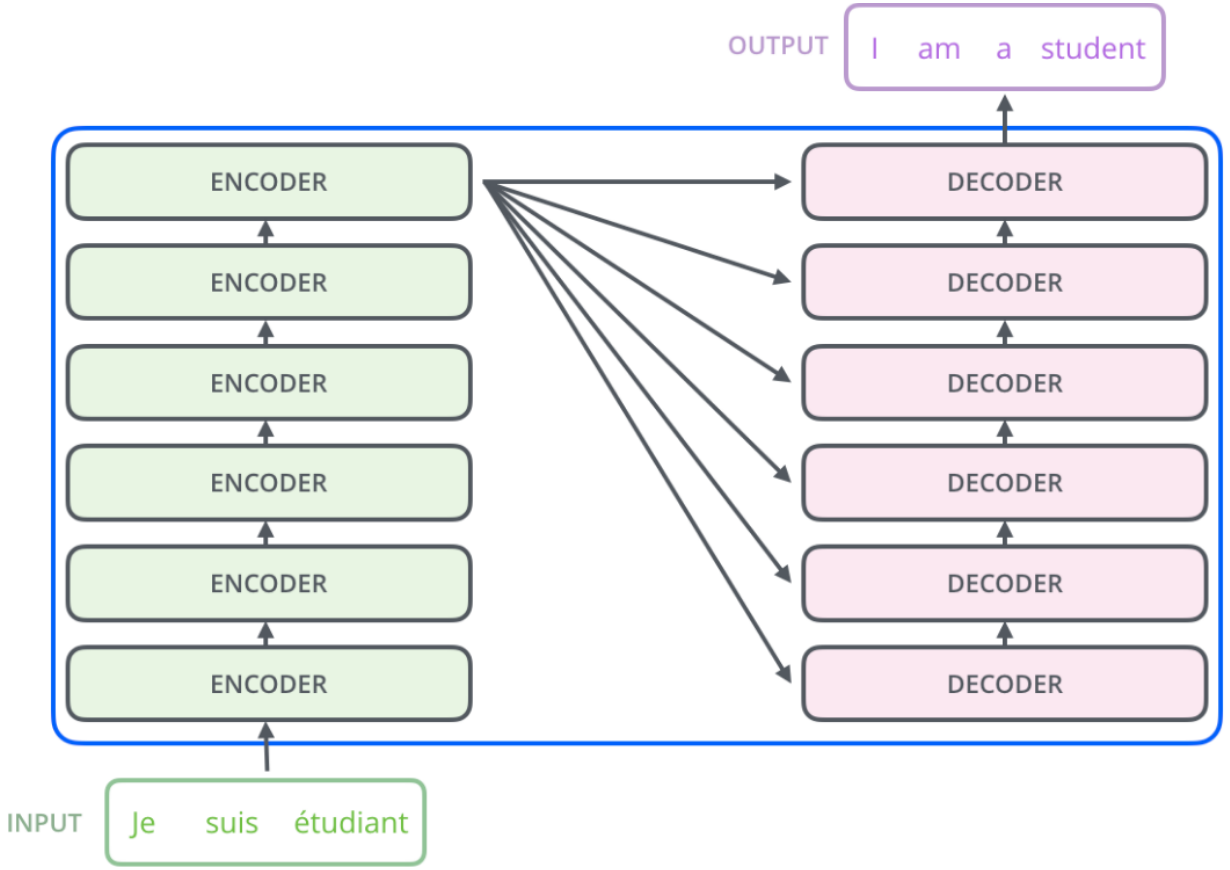
На остальных слоях повторяем шаги 2-6.



Теперь мы готовы перейти к кодировщику-декодировщику!

Декодировщик трансформера: связь с кодировщиком

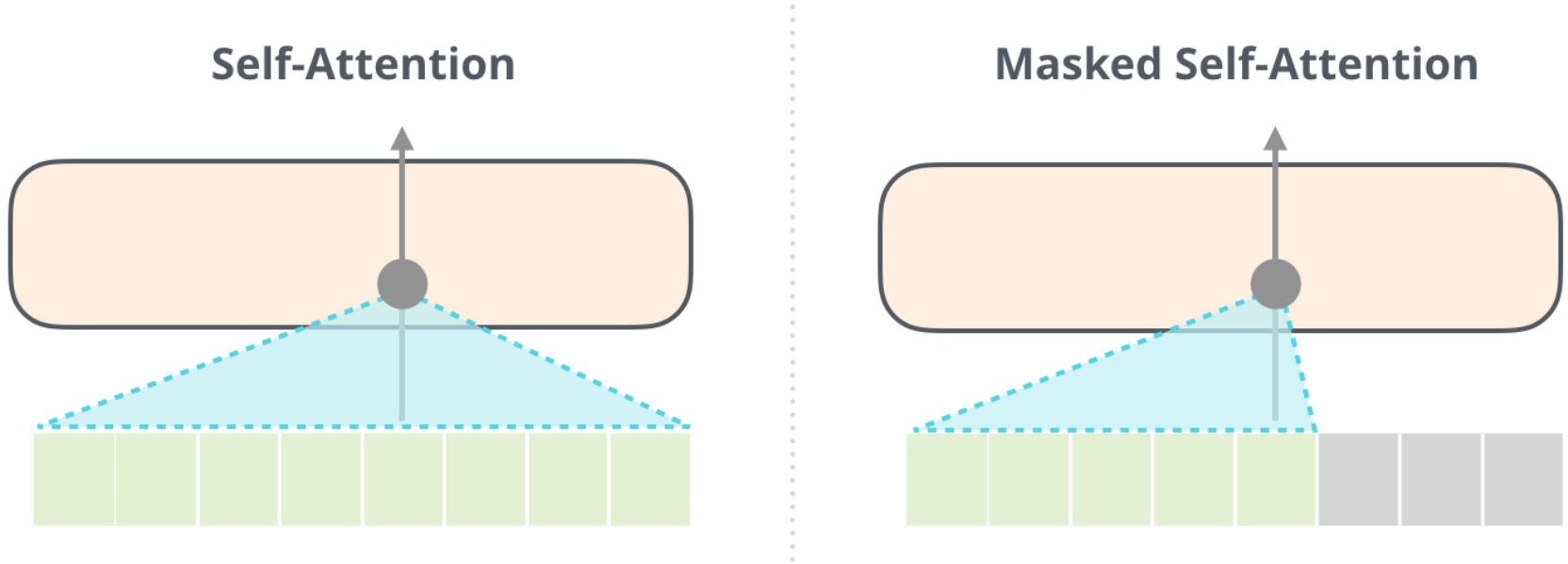
Кодировщик и декодировщик состоят из своих наборов одинаковых блоков, блоки стекаются друг за другом.
По-умолчанию, веса у каждого блока свои (неразделяемые).



Masked self-attention

Внимание в декодировщике трансформера учитывает только предыдущие токены:

$$\alpha_{ij} = \underset{j \in \{1,\ldots,i\}}{softmax}\ sim(q_i, k_j)$$

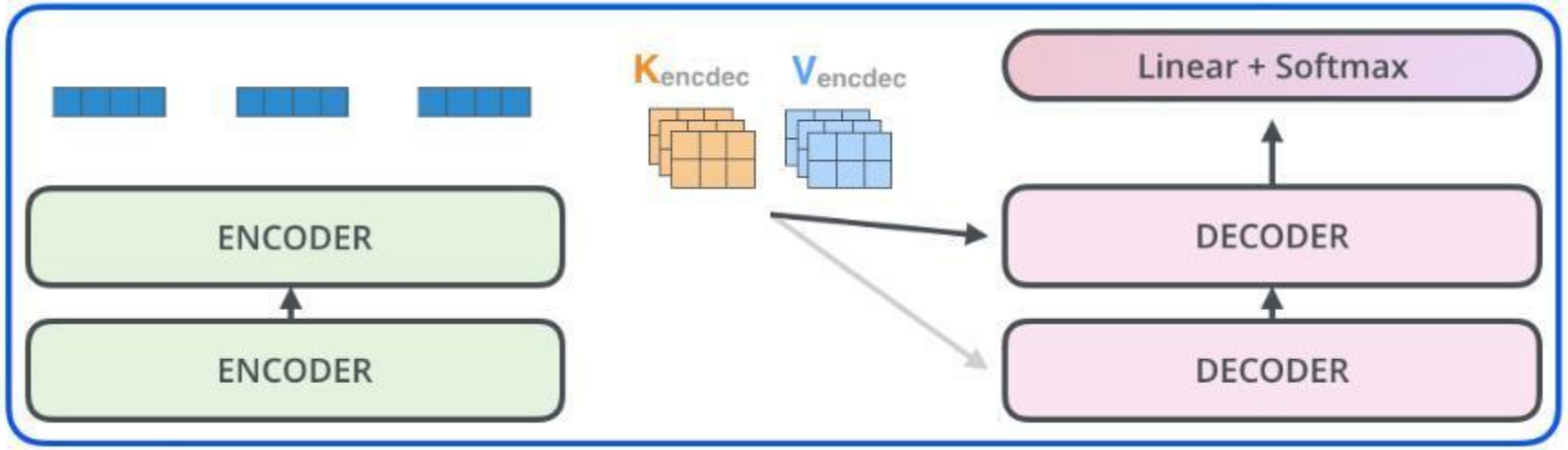


Декодировщик: связь с кодировщиком

Декодировщик состоит из последовательных блоков декодировщиков

Выходы кодировщика преобразовываются обучаемыми весовыми матрицами в набор матриц Key и Value

Эти матрицы передаются в каждый из блоков-декодировщиков



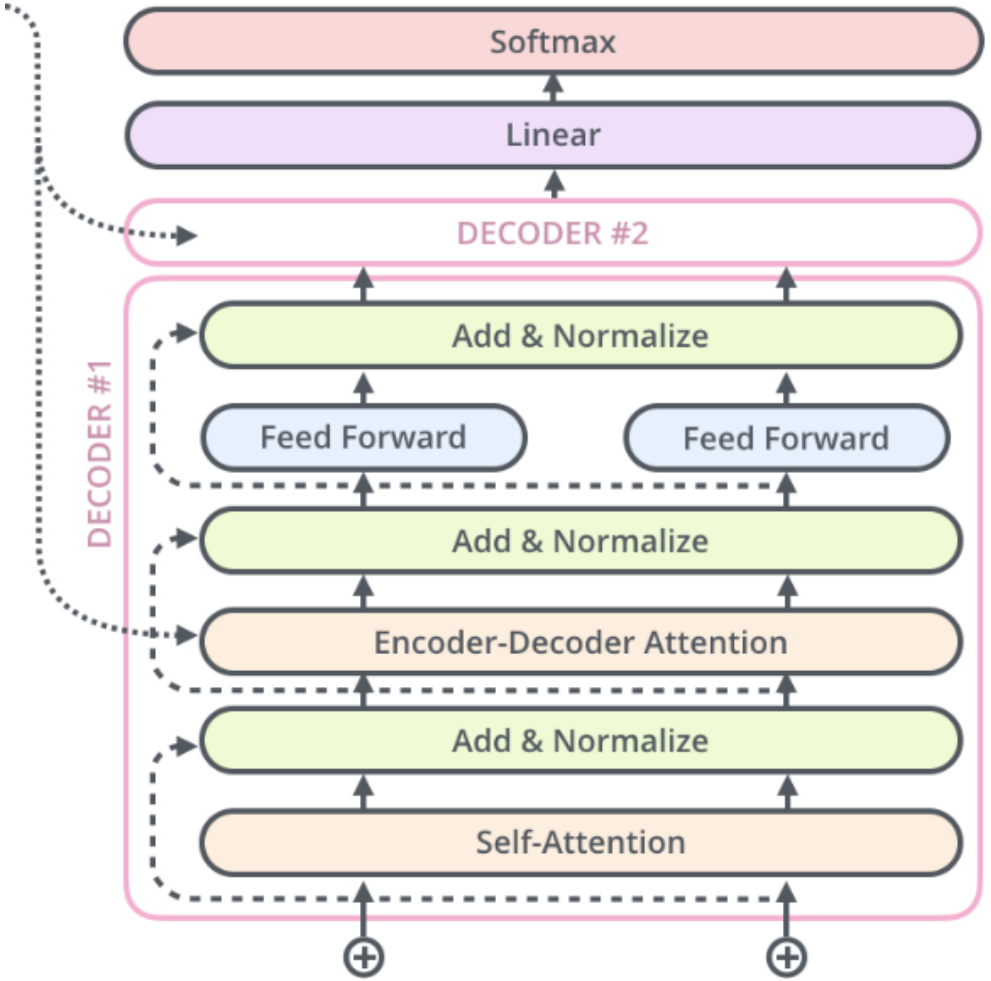
Архитектура декодировщика

Выходы первого слоя MHSA идут во второй – Encoder-Decoder Attention

Encoder-Decoder Attention — MHSA выходов первого слоя по выходам кодировщика:

- Key и Value — выходы кодировщика
- Query — первый слой декодировщика

На выходе блока — набор векторов, соответствующих токенам входной последовательности.



Обучение трансформеров: warmup

Проблема 1. В первые несколько итераций сеть адаптируется к данным, а только потом начинает обучаться.

Проблема 2. На первых эпохах сложная сеть переобучается под простые объекты в данных.

Решение. Использование warmup scheduler (изменение темпа обучения с разогревом).

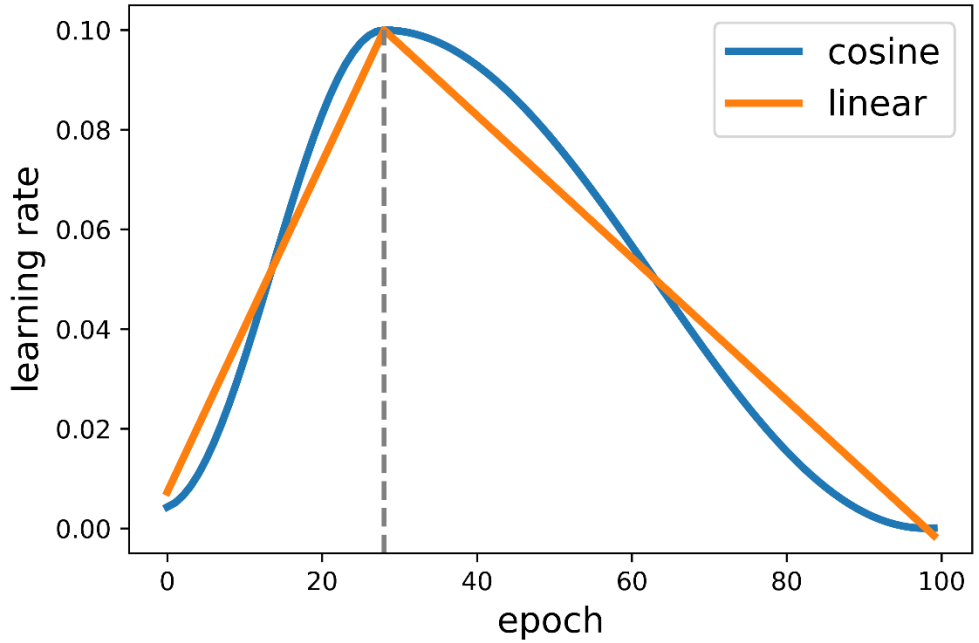
Warmup sheduler: основные стратегии

Два основных типа:

- Cosine annealing
- Linear annealing

Важные параметры:

- доля разогрева (pct)
- общее число эпох
- первый, максимальный и последний темп обучения



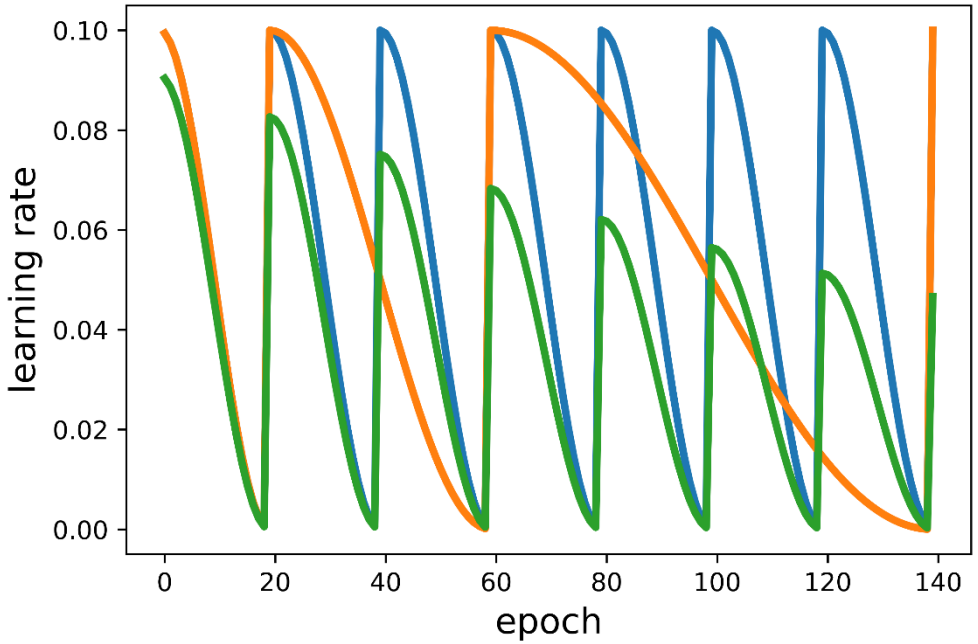
Pytorch: OneCycleLR

Smith et al. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates

Warmup: циклические стратегии

Стратегии могут быть циклическими:

- Длина циклов может увеличиваться
- Максимальный темп обучения может уменьшаться
- К первой точке нового цикла может не быть плавного перехода



Зачем это может быть надо?

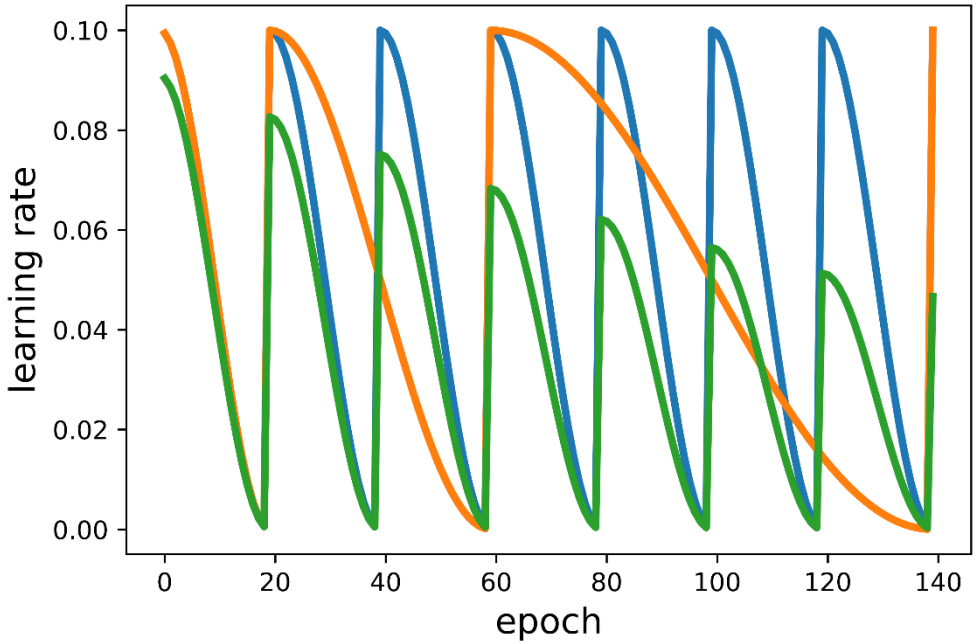
Smith. Cyclical Learning Rates for Training Neural Networks

Loshchilov et al. SGDR: Stochastic Gradient Descent with Warm Restarts

Warmup: циклические стратегии

Стратегии могут быть циклическими:

- Длина циклов может увеличиваться
- Максимальный темп обучения может уменьшаться
- К первой точке нового цикла может не быть плавного перехода



Зачем это может быть надо?

Можем ансамблировать модели из нижних точек графиков.

Smith. Cyclical Learning Rates for Training Neural Networks

Loshchilov et al. SGDR: Stochastic Gradient Descent with Warm Restarts

Что ещё следует помнить про трансформеры?

- Обычно, мы работаем с subword-токенизацией (токенизация по буквенным n-граммам)
- Кодировщик и декодировщик не обязаны быть одного размера
- При обучении можно использовать Adam, обычно используют warm-up расписание для темпа обучения
- Сложность трансформера квадратичная как по длине последовательности, так и по размеру внутреннего слоя.
- Позиционные эмбеддинги могут быть фиксированными или обучаемыми. Также, их можно сделать относительными.

Что ещё следует помнить про трансформеры?

- Трансформеры можно применять во всех задачах, которые мы обсуждали до этого: классификация и разметка
- Трансформеры обычно не требуют детального подбора параметров обучения и архитектуры (в отличии от RNN)
- Самые важные параметры – размер скрытого слоя и количество слоёв, затем размер словаря и только потом количество голов
- Трансформеры можно использовать вместе с CRF
- Трансформеры могут содержать очень много attention блоков (BERT – 12 блоков, T5 – 14 блоков, GTP3 – 96 блоков)

Subword токенизация

- softmax по всем словам – дорогая операция
- не можем использовать лемматизацию/стемминг, так как хотим генерировать естественный язык
- есть языки, в которых слова могут быть очень длинными (например, немецкий язык)

Решение: вместо токенизация предложения на слова, будем делить предложение на буквенные n-граммы

Алгоритм BPE (byte pair encoding)

- Исходный словарь токенов – множество символов корпуса
- Исходный набор правил – пустое множество
- Каждое слово в корпусе – последовательность токенов

На каждой итерации **обучения**:

1. Добавляем в словарь токенов самую часто встречающуюся пару a, b
2. Добавляем в набор правил новое правило $\{a\ b\ -> \ ab\}$
3. Обновляем корпус исходя из нового правила
4. Можно ограничивать максимальный размер токена в символах или общее число токенов

Применение: последовательное применение полученных правил

Sennrich et al (ACL 2016); Neural Machine Translation of Rare Words with Subword Units

BPE на примере

Проведём 5 итераций обучения алгоритма на предложении «she sells seashells by the seashore»:

1. s h e | s e l l s | s e a s h e l l s | b y | t h e | s e a s h o r e

BPE на примере

Проведём 5 итераций обучения алгоритма на предложении «she sells seashells by the seashore»:

1. s h e | s e l l s | s e a s h e l l s | b y | t h e | s e a s h o r e

$$\{s\ h \rightarrow sh\}$$

2. sh e | s e l l s | s e a sh e l l s | b y | t h e | s e a sh o r e

BPE на примере

Проведём 5 итераций обучения алгоритма на предложении «she sells seashells by the seashore»:

1. s h e | s e l l s | s e a s h e l l s | b y | t h e | s e a s h o r e

$$\{s\ h \rightarrow sh\}$$

2. sh e | s e l l s | s e a sh e l l s | b y | t h e | s e a sh o r e

$$\{s\ h \rightarrow sh, s\ e \rightarrow se\}$$

3. sh e | se l l s | se a sh e l l s | b y | t h e | se a sh o r e

BPE на примере

Проведём 5 итераций обучения алгоритма на предложении «she sells seashells by the seashore»:

1. s h e | s e l l s | s e a s h e l l s | b y | t h e | s e a s h o r e

$$\{s\ h \rightarrow sh\}$$

2. sh e | s e l l s | s e a sh e l l s | b y | t h e | s e a sh o r e

$$\{s\ h \rightarrow sh, s\ e \rightarrow se\}$$

3. sh e | se l l s | se a sh e l l s | b y | t h e | se a sh o r e

$$\{s\ h \rightarrow sh, s\ e \rightarrow se, l\ l \rightarrow ll\}$$

4. sh e | se ll s | se a sh e ll s | b y | t h e | se a sh o r e

BPE на примере

Проведём 5 итераций обучения алгоритма на предложении «she sells seashells by the seashore»:

1. s h e | s e l l s | s e a s h e l l s | b y | t h e | s e a s h o r e

$$\{s\ h\ \rightarrow\ sh\}$$

2. sh e | s e l l s | s e a sh e l l s | b y | t h e | s e a sh o r e

$$\{s\ h \rightarrow sh, s\ e \rightarrow se\}$$

3. sh e | se l l s | se a sh e l l s | b y | t h e | se a sh o r e

$$\{s\ h \rightarrow sh, s\ e \rightarrow se, l\ l \rightarrow ll\}$$

4. sh e | se ll s | se a sh e ll s | b y | t h e | se a sh o r e

$$\{s\ h \rightarrow sh, s\ e \rightarrow se, l\ l \rightarrow ll, se\ a\ \rightarrow se\}$$

5. sh e | se ll s | sea sh e ll s | b y | t h e | sea sh o r e

Subword токенизация: префиксы

Необходимо понимать какие BPE токены начинают новые слова, а какие находятся внутри слова. Для этого используют специальные префиксы:

- Sentencepiece

  [I, like, fishing] -> [_I, _li, ke, _fish, ing]

- Wordpiece

  [I, like, fishing] -> [I, li, ##ke, fish, ##ing]

Также можно считать пробельные символы отдельным токеном (используется для кодирования текстов Python программ).

https://github.com/google/sentencepiece

Subword токенизация: разновидности

Wordpiece

Вместо выбора по частоте $a, b = arg\max_{a,b} p(a, b) = arg\max_{a,b} n_{ab}$

выбираем по правдоподобию $a, b = arg\max_{a,b} p(b|a) = arg\max_{a,b} \frac{n_{ab}}{n_a}$

BPE + байты

Слова представляются как последовательность байтов и BPE применяется к байтам

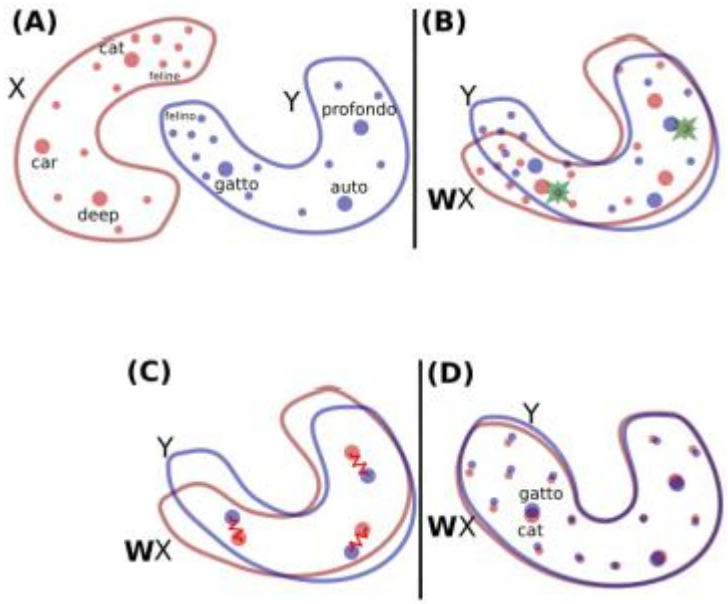
BPE dropout

На этапе применения случайно пропускаются некоторые правила (для улучшения работы с новыми словами).

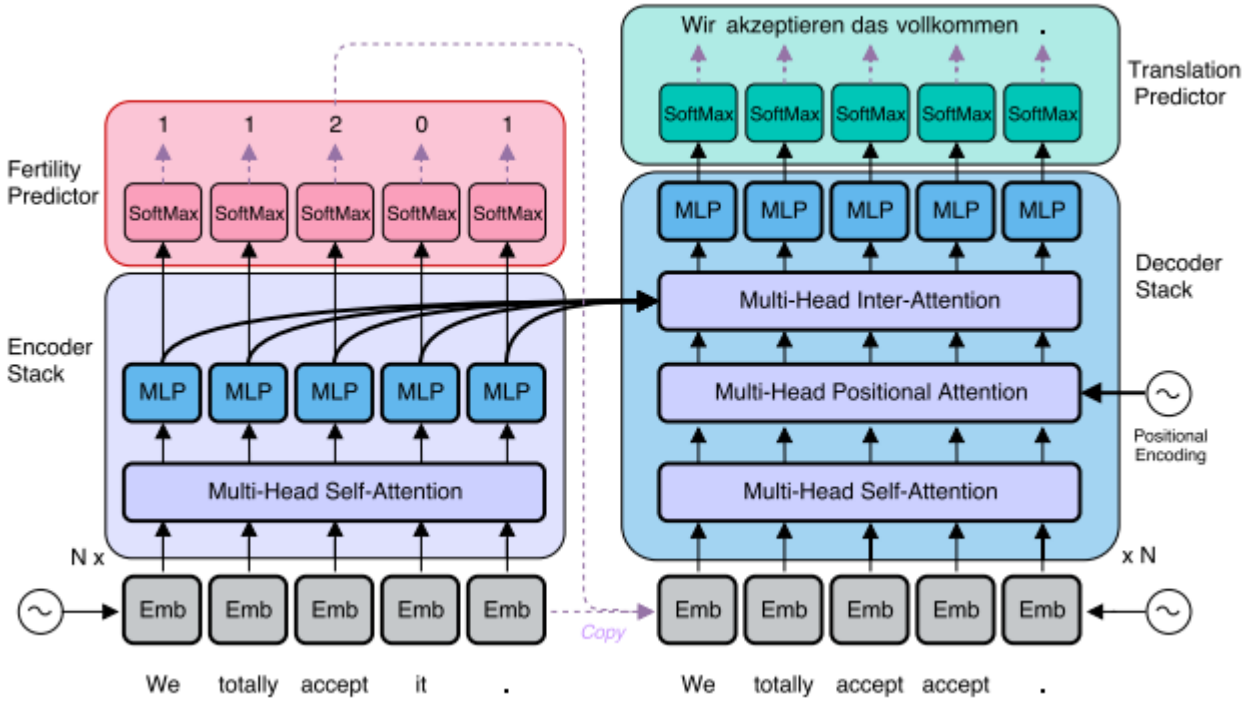
Provilkov et al (ACL 2020); BPE-Dropout: Simple and Effective Subword Regularization

А что ещё можно узнать про машинный перевод?

Машинный перевод без учителя



Неавторегрессионный перевод



[Gu et al (ICLR 2018); Non-Autoregressive Neural Machine Translation]

[Conneau et al (ICLR 2018); Word Translation Without Parallel Data]