Лекция 7: Механизм внимания (attention)

Автор: Сергей Вячеславович Макрушин e-mail: SVMakrushin@fa.ru (mailto:SVMakrushin@fa.ru)

Финансовый университет, 2021 г.

При подготовке лекции использованы материалы:

- https://dlcourse.ai/ (https://dlcourse.ai/)
- https://neerc.ifmo.ru/wiki/index.php?title=Механизм_внимания (https://neerc.ifmo.ru/wiki/index.php?title=%D0%9C%D0%B5%D1%85%D0%B0%D0%BD%D0%B8%D0%B7%D0%BC_%D0%B2%D0%BD

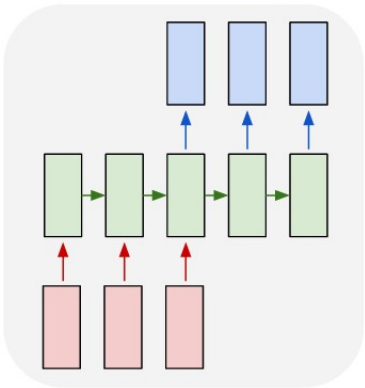
v 0.1 02.05.21

Введение в механизм внимания

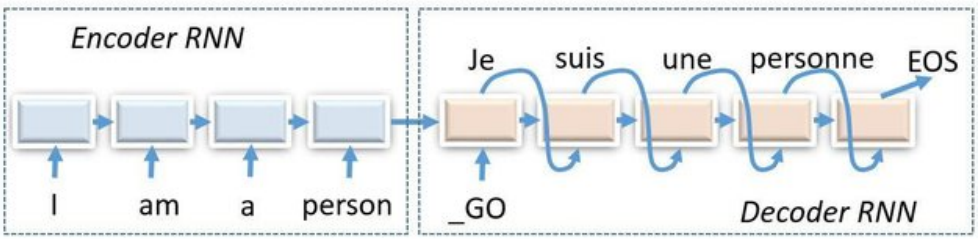
Изначально механизм внимания был представлен в контексте рекуррентных Seq2seq[1] сетей [2] для "обращения внимания" блоков декодеров на скрытые состояния RNN для любой итерации энкодера, а не только последней.

После успеха этой методики в **машинном переводе** последовали ее внедрения в:

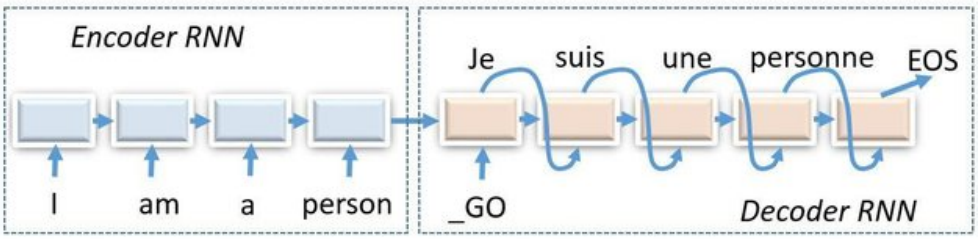
- других задачах обработки естественного языка
- задачах генерации описания изображения (в применении к CNN)
- в порождающих состязательных сетях (GAN)



Рекуррентные сети Seq2seq (приниципиальная схема)



Пример использования RNN для задачи машинного перевода



Пример использования RNN для задачи машинного перевода

Специфика:

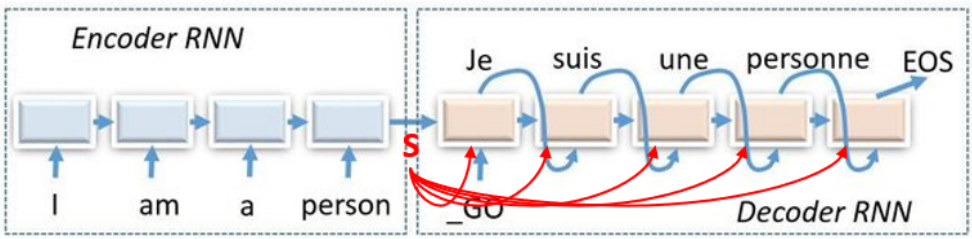
- Для энкодера и декодера используются различные веса сети (RNN модуля)
- Скрытое состояние энкодера на последнем шаге обработки предложения (последовательности) является ключевым т.к. по сути кодирует все предложение. Затем декодер использует именно это состояние для того, чтобы сгенерировать перевод предложения на другом языке.
- Результат сэмплинга декодера на предыдущем шаге передается на следующий блок декодера в добавок к внутреннему состянию RNN декодера.

Проблемы:

- Расстояние между местом кодирования слова и местом его декодирования большое.
- Последовательность слов в исходном и целевом предложении часто должна быть разная.
- Часто количество слов в исходном и целевом предложении различается.

Приемы, направленные на преодоление проблемы :

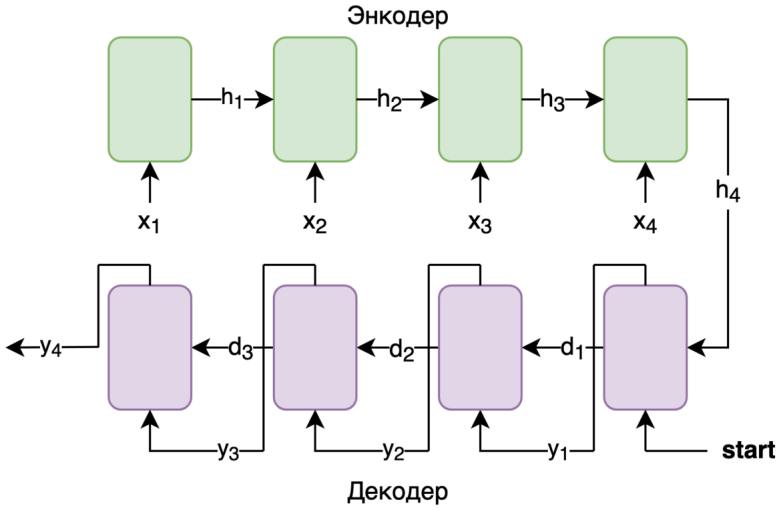
- Для языков, в которых последовательность слов в эквивалентных передложениях примерно одинакова, может быть полезно декодеру **гененрировать предложение в обратном порядке**. Это дает возможность иметь небольшую последовательность преобразований (*"небольшое расстояние"*) от энкодинга слов (последних слов в исходном предложении) до места их декодирования.
- Может быть полезно скрытое состояние энкодера на последнем шаге передавать на вход каждому блоку декодера, чтобы каждый декодер имел прямой доступ к энкодингу всего предложения (этот подход тоже сокращает сложность передачи информации об исходном предложение "сквозь" последовательность в декодере):



RNN в машинном переводе: использование ключевого состояния энкодера

Базовая архитектура Seq2seq

Seq2seq состоит из двух рекуррентных нейронных сетей (RNN): **Энкодера** и **Декодера**.



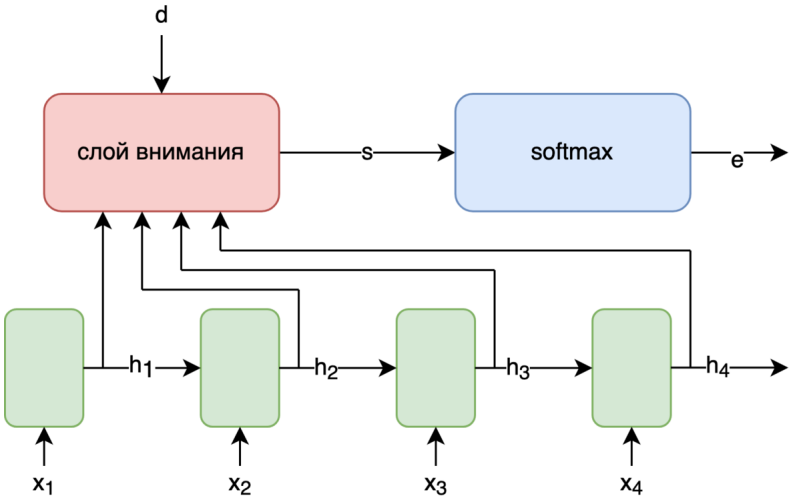
Рекуррентные сети Seq2seq

- **Энкодер** - принимает последовательность (например: предложение на языке *A*) и сжимает его в вектор скрытого состояния.
  - x_i слова в предложении на языке *A*
  - h_i скрытое состояние энкодера
  - **блоки энкодера** (зеленый): получают на вход x_i и передают скрытое состояние h_i на следующую итерацию
- **Декодер** - выдает слово на языке *B*, принимает последнее скрытое состояние энкодера (для первого слова) / предыдущее скрытое состояние декодера (последующие слова) и предыдущее предсказанное слово.
  - d_i скрытое состояние декодера
  - y_i слова в предложении на языке *B*
  - Блоки декодера (фиолетовый) блоки декодера получающие на вход y_{i-1} или специальный токен `<start>` (в случае первой итерации) и возвращаюшие на следующую итерацию:
    - y_i - слова в предложении на языке *B*
    - d_i - скрытое состояние декодера
    - Перевод считается завершенным при возвращении y_i, равного специальному токену `<end>`

Применение механизма внимания для Seq2seq: базовый принцип

- Использование механизма внимания позволяет решать задачу нахождения закономерности между словами находящимися **на большом расстоянии друг от друга**.
- LSTM, GRU и аналогичные блоки используются для улучшения передачи информации на большое количество итераций по сравнению с базовыми RNN, несмотря на это сохраняется проблема: влияние предыдущих состояний на текущее уменьшается экспоненциально от расстояния между словами
  - В классическом применении RNN результатом является только последнее скрытое состояние h_m, где m - длина последовательности входных данных.
- **Механизм внимания** улучшает этот показатель до **линейного**.
  - Использование механизма внимания позволяет использовать информацию, полученную не только из последнего скрытого состояния, но и из любого скрытого состояния h_t для любого $t \in 1 \dots m$.

Устройство слоя механизма внимания

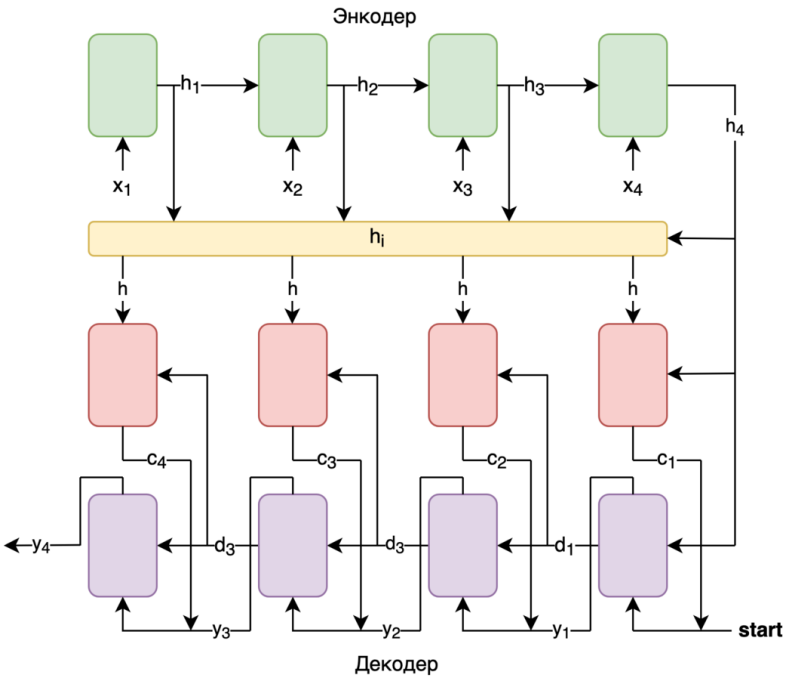


Обобщенный механизм внимания в RNN

1. Слой механизма внимания представляет собой обычную, чаще всего однослойную, нейронную сеть на вход которой подаются $h_t, t = 1 \dots m$ (скрытые состояния на последовательных шагах RNN), а также вектор d в котором содержится некий контекст, зависящий от конкретной задачи.
    - В случае *Seq2seq* сетей вектором d будет скрытое состояние предыдущей итерации декодера d_{i-1} .
2. Выходом данного слоя будет является вектор s (от *score*) - оценки на основании которых на скрытое состояние h_i будет "обращено внимание".
3. Для нормализации значений s используется $softmax$: $e = softmax(s)$
    - $\forall s: \ \sum_{i=1}^{n} softmax(e)_i = 1$
    - $\forall s, \ i: \ softmax(e)_i >= 0$
4. Результатом работы слоя внимания является c (**context vector**) который, содержит в себе информацию обо всех скрытых состояниях h_i пропорционально нормализованному вниманию e_i :
    - $\mathrm{c} = \sum_{i=1}^{m} e_i h_i$

Применение механизма внимания к базовой Seq2seq архитектуре

При добавлении слоя механизма внимания в базовую архитектуру *Seq2seq* между RNN *Энкодером* и *Декодером* получится следующая схема:



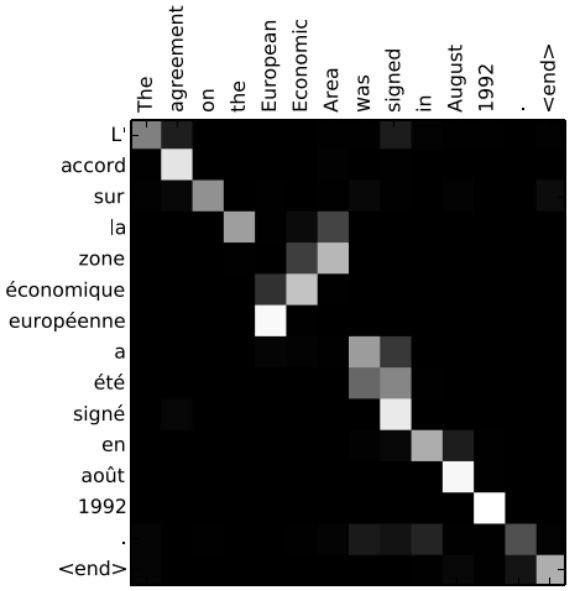
Пример схемы работы Seq2seq сети с механизмом внимания

- x_i, h_i, d_i, y_i имеют те же назначения, что и в базовом варианте без механизма внимания.

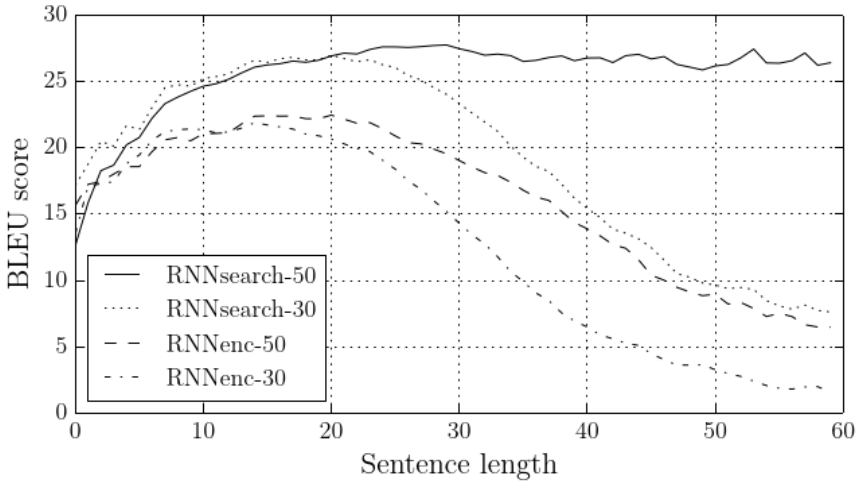
- *Агрегатор скрытых состояний энкодера (желтый)* - агрегирует всю последовательность векторов $h = [h_1, h_2, h_3, h_4]$.
- *Блоки механизма внимания (красный)* - принимают h и d_{i-1} (*контекст для выбора параметров* для блока i-го блока декодера (сиреневый цвет)), возвращает c_i - контекст для i-го блока декодера
- *Блоки декодера (фиолетовый)* - по сравнению с обычной *Seq2seq* сетью меняются входные данные. Теперь, вместо y_{i-1} на i итерации на вход подается: конкатенация y_{i-1} и c_i .

При помощи механизма внимания достигается **"фокусирование" декодера** на определенных скрытых состояниях энкодера, на любом из шагов энкодера.

- В случаях машинного перевода эта возможность помогает декодеру предсказывать на какие скрытые состояния при исходных определенных словах на языке *A* необходимо обратить больше внимания при генерации очередного слова перевода на язык *B*.
  - При этом расстояние (в шагах последовательности) между генерируемым словом и влияющими на него словами из исходной последовательности никак не влияют на возможность использование скрытых состояний, сформированных под их воздействием.



Пример весов механизма внимания при решении задачи машинного перевода

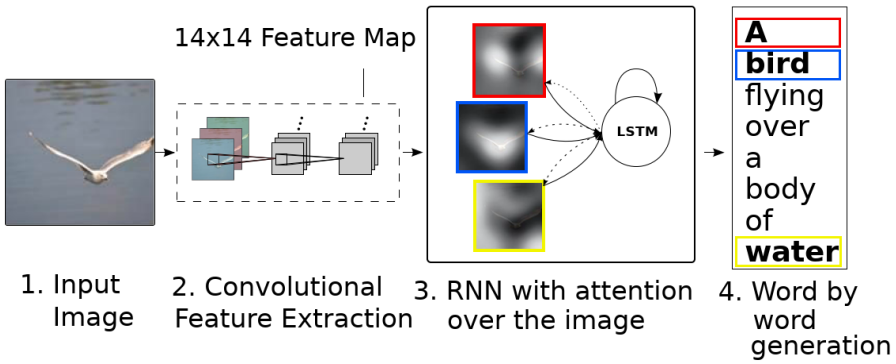


Эффект использования механизма внимания при решении задачи машинного перевода для предложений разной длины

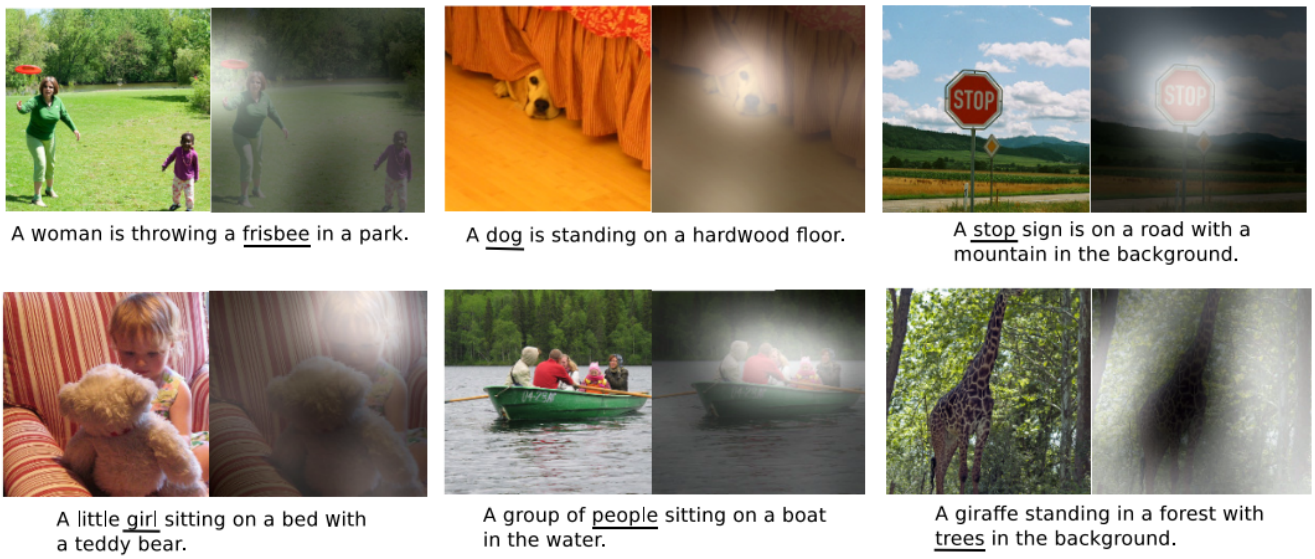
Использование механизмов внимания для других задач

Задача *Image Captioning* : на вход подается изображение, а на выходе создается текстовое предложение, описывающее визуальное содержание картинки.

- Классическая нейростевая модель для решения задачи Image Captioning состоит из:
  - сверточной нейронной сети (CNN) для извлечения визуальных характеристик из картинки
  - и рекуррентной нейронной сети (RNN) для перевода результата работы сверточной сети в текст.
- **Сверточный модуль внимания** (convolutional block attention module) — модуль внимания для сверточных нейросетей. Применяется для задач детектирования обьектов на изображениях и классификации с входными данными больших размерностей.



Логика работы архитектуры, использующей attention model для задачи Image Captioning

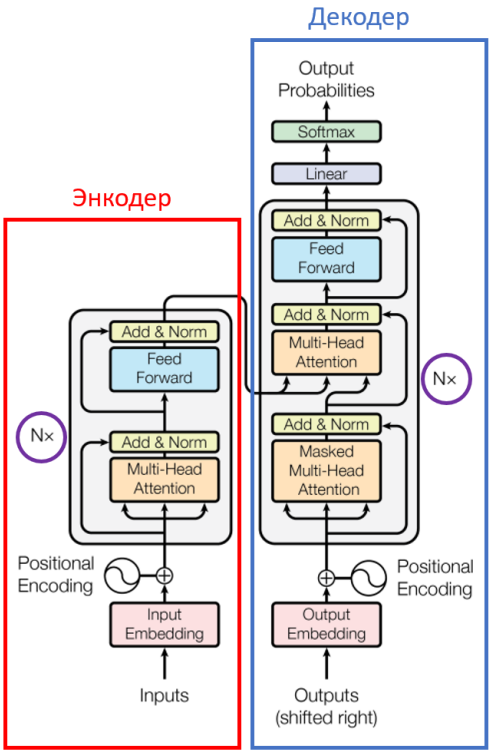


Пример использования attention model для генерации существительных в описании картинок

Transformer

Архитектура Transformer

- Рассматриваем задачу машинного перевода *Seq2seq*
- новая архитектура для решения этой задачи, которая не является ни RNN, ни CNN



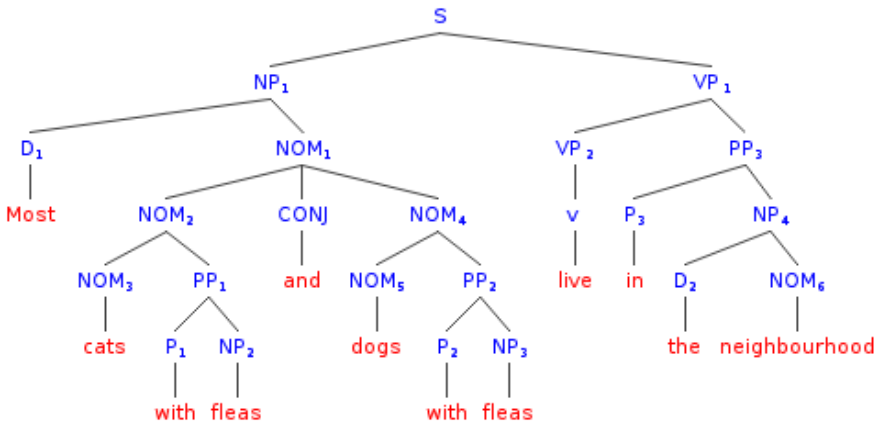
Принципиальная архитектуры модели Transformer из статьи "Attention Is All You Need"*

*https://arxiv.org/abs/1706.03762 (https://arxiv.org/abs/1706.03762)

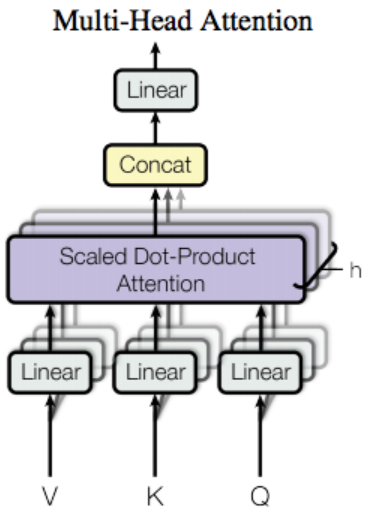
Encoder, Multi-head attention

Multi-head attention - слой, который дает возможность каждому входному вектору (эмбеддингу слова) взаимодействовать с любыми другими словами предложения через attention mechanism, вместо:

- последовательной передачи hidden state, как в RNN
- свертки эмбеддингов соседних слов, как в CNN



Пример иерархическая структуры предложения



Организация Multi-head attention

В Multi-head attention:

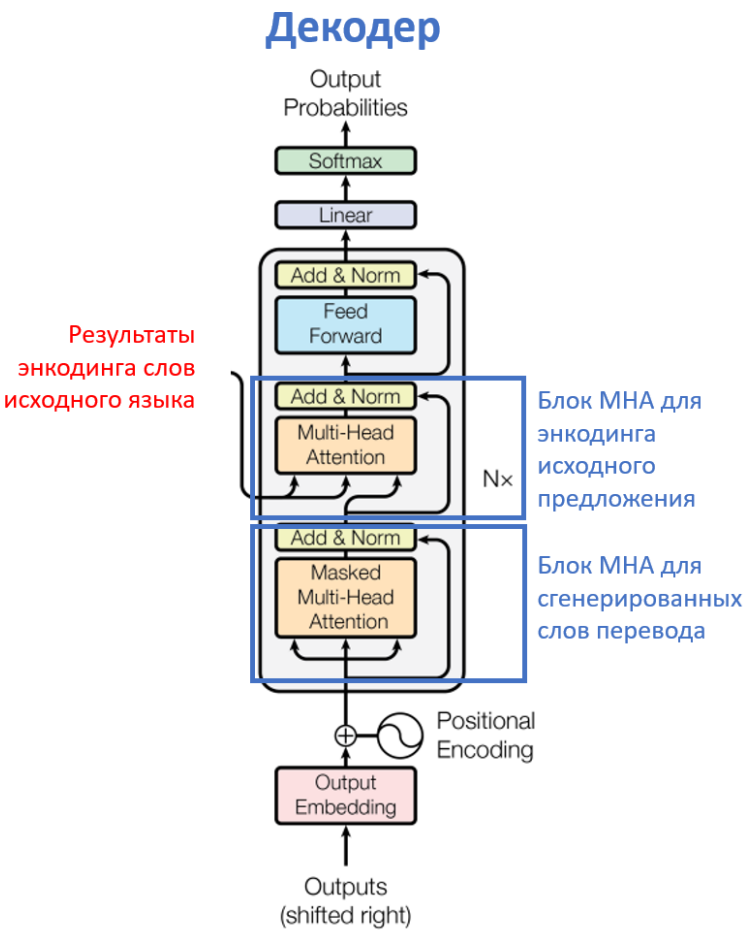
1. на вход даются вектора Query (Q), и несколько пар Key (K) и Value (V) (на практике, Key и Value это всегда один и тот же вектор)
2. каждый из них преобразуется обучаемым линейным преобразованием
3. вычисляется скалярное произведение Q со всеми K по очереди
4. результат этих скалярных произведений прходит через softmax
5. с полученными весами все вектора V суммируются в единый вектор
6. **Ключевое отличие от классического attention**: результат всех этих параллельных attention'ов (на картинке их h) конкатенируется, прогоняется через обучаемое линейное преобразование и идет на выход (получается вектор того же размера, что и каждый из входов)

Особенности:

- Q: в чем смысл множества одинаковых attention слоев?
  - Логика стандартного attention: сеть пытается выдать соответствие одного слова другому в предложении, если они близки чем-то
  - Не обучатся ли разныеслои Multi-head attention в точности одному и тому же?
    - Нет. Технически, разных начальных случайных весов хватает, чтобы толкать разные слои в разные стороны.
    - Разные слои позволяют сети обращать внимание на различные аспекты слов (например семантические (смысловые) и грамматические аспекты слов)
- Так как на выход такой блок выдает вектор того же размера, что и был на входе, то этот блок можно вставлять в сеть несколько раз, добавляя сети глубину (в рассмотренной работе этот блок использовался с глубиной 6).
- Одной из фич каждого слова является **positional encoding** — т.е. его позиция в предложении. Например, в процессе обработки слова это позволяет легко "обращать внимание" на соседние слова, если они важны.

Пример работы архитектуры Transformer

Декодер



Организация декодера

- декодер запускается по слову за раз, получает на вход:
  - вектора всех исходных слов, полученные при энкодинге
  - все прошлые слова и должен выдать следующее (на первой итерации получает специальный токен).
- соответственно, декодер состоит из двух блоков:
  - первый — работает с векторами предыдущих декодированных слов, аналогичный использованному в процессе encoding (но может обращаться не ко всем словам, а только к

использованному в процессе encoding (не может обращаться ни ко всем словам, а только к уже декодированным).
    - второй — работает с выходом энкодера. В этом случае Query — это вектор входа в декодере, а пары Key/Value — это финальные эмбеддинги энкодера
- все это снова повторяется 6 раз (глубокое обучение!), где выход предыдущего блока идет на вход следующему
- в конце сети стоит softmax, который выдает вероятности слов. Сэмплирование из него и есть результат, то есть следующее слово в предложении.
- Результат семплирования подается на вход следующему запуску декодера и процесс повторяется, пока декодер не выдаст токен `<EOS>`.

Преимущества архитектуры Transformer

(над RNN и CNN)

- Все слова в предложении одинаково доступны при кодировании/декодировании (расстояние не важно)
- Может эффективно выполняться параллельно
- Эффективная глубина сети меньше по сравнению с RNN (RNN "разматывается" вдоль последовательности), в Transformer глубина графа не зависит от длины последовательности

Bert

BERT

Bidirectional Encoder Representations from Transformers (BERT) - языковая модель, основанная на архитектуре трансформер, предназначенная для предобучения языковых представлений с целью их последующего применения в широком спектре задач обработки естественного языка. BERT был создан и опубликован в 2018 году Якобом Девлином и его коллегами из Google.

- BERT является автокодировщиком
- BERT использует трансформер-архитектуру
    - В каждом слое кодировщика применяется двустороннее внимание, что позволяет учитывать контекст с обеих сторон от рассматриваемого токена
- В каждом слое кодировщика применяется двустороннее внимание

Языковое моделирование (language modelling, LM) - использование различных статистических и вероятностных методов для определения вероятности того, что данная последовательность слов встречается в предложении.

- Для обеспечения достоверности предсказаний языковые модели строятся на основе анализа массивов текстовых данных.
- Языковые модели используются в приложениях обработки естественного языка (natural language processing, NLP). В частности:
    - Задачах машинного перевода.
    - Задачах ответа на вопросы (question answering).
    - Других задачах, особенно, требующих генерацию текста.

Автокодировщик (autoencoder) — специальная архитектура искусственных нейронных сетей, позволяющая применять **обучение без учителя** с использованием метода обратного распространения ошибки.

Автокодировщик состоит из двух частей:

- **энкодера** g, он переводит входной сигнал в его представление (код): $h = g(x)$
- **декодера** f, он восстанавливает сигнал по его коду: $x = f(h)$

При этом семейства функций энкодера g и декодера f определенным образом ограничены, чтобы автоэнкодер был вынужден отбирать наиболее важные свойства сигнала.

Автокодировщик, изменяя (обучая) f и g, стремится выучить тождественную функцию $x = f(g(x))$, минимизируя заданный функционал ошибки: $L(x, f(g(x)))$.

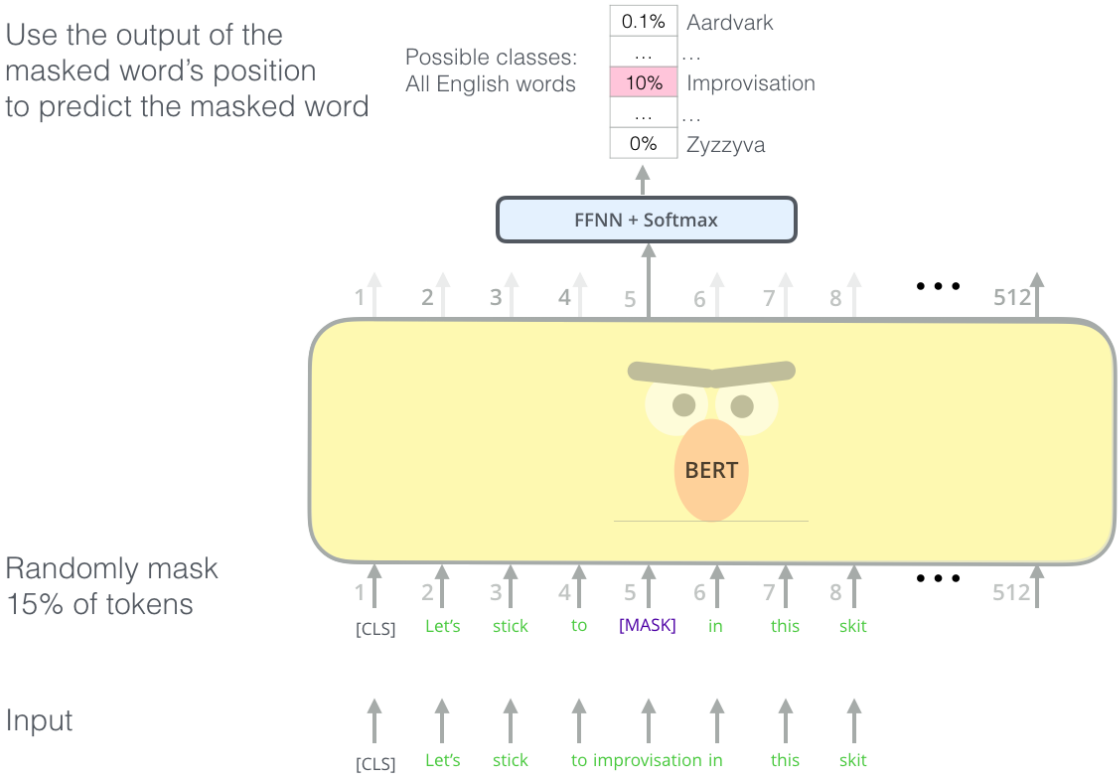
Автокодировщик можно использовать:

- для предобучения, например, когда стоит задача классификации, а размеченных пар слишком мало
- понижения размерности в данных, например для последующей визуализации
- когда надо научиться различать полезные свойства входного сигнала

Обучение модели BERT

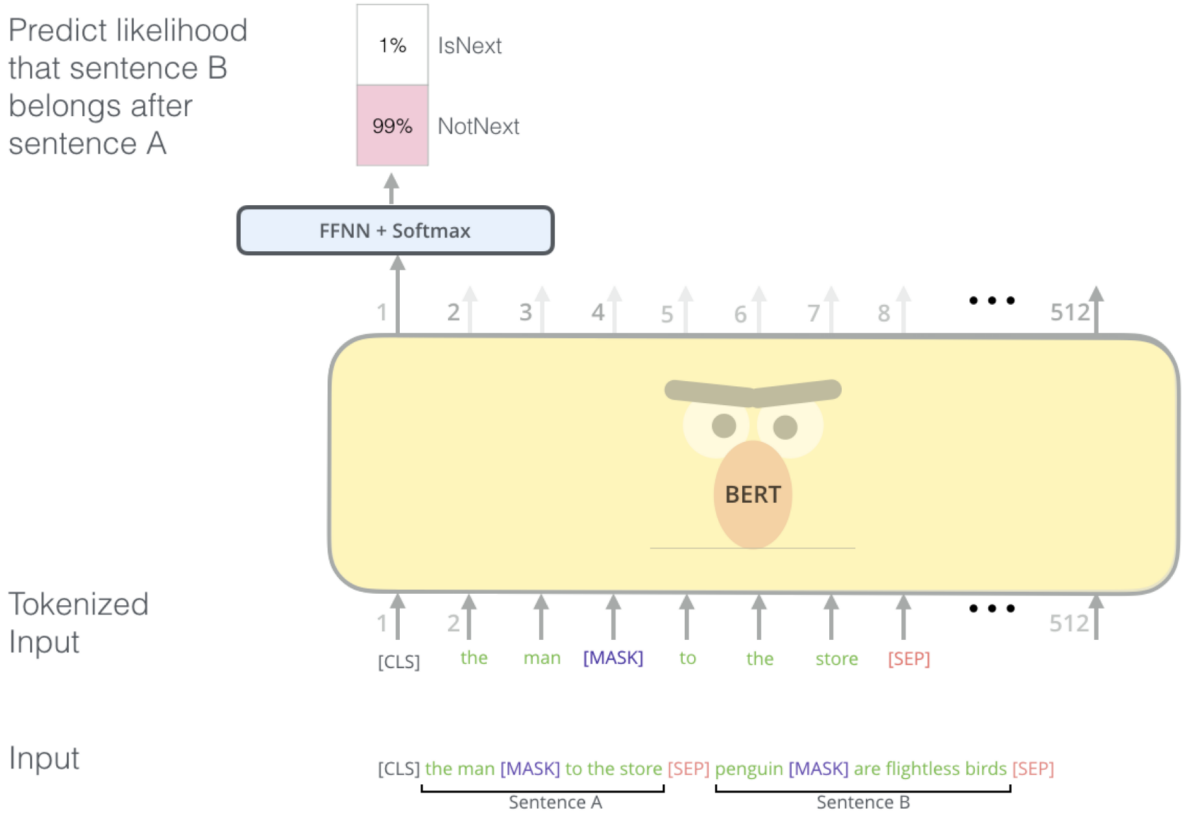
BERT-модель **предварительно обучена без учителя** на на двух NLP-задачах:

- генерации пропущенного токена (masked language modeling)
  - На вход BERT подаются токенизированные пары предложений, в которых **некоторые токены скрыты**
  - Модель обучается генерировать пропущенные токены (слова или отдельные токены)

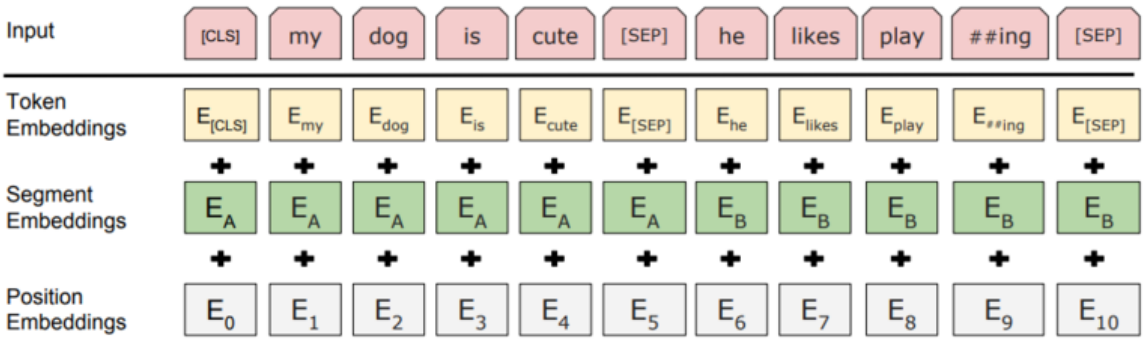


Логика задачи генерации пропущенного токена

- предсказание следующего предложения:
  - Задача же предсказания следующего предложения есть задача бинарной классификации — является ли второе предложение продолжением первого
  - Благодаря этой задаче сеть обучается различать связь между предложениями в тексте



Логика задачи генерации пропущенного токена



Представление входных данных модели

- Токенизация в BERT:
  - Токенами служат слова, доступные в словаре, или их составные части
  - Если слово отсутствует в словаре, оно разбивается на части, которые в словаре присутствуют
  - Словарь является составляющей модели
    - Например в BERT-Base используется словарь примерно из 30K слов.
- В нейронной сети BERT **токены кодируются своими эмбеддингами**, а именно: соединяются представления самого токена (предобученные), номера его предложения, а также позиции токена внутри своего предложения.
- Предобучение проведено на больших корпусах текстов (есть многоязычные варианты модели BERT).
  - В частности, многоязыковая модель BERT-Base, поддерживающая 104 языка, состоит из 12 слоев, 768 узлов, 12 выходов и 110M параметров
- Предобученные модели можно подстраивать под решение конкретных NLP-задач:
  - Дообучаем модель на данных, специфичных задаче: знание языка уже получено на этапе предобучения языковой модели, необходима лишь коррекция сети

Данные и оценка качества

Исходные данные: Предобучение ведётся на текстовых данных корпуса BooksCorpus (800 млн. слов), а также на текстах англоязычной Википедии (2.5 млрд. слов).

Качество модели авторы оценивают на популярном для обучения моделей обработки естественного языка наборе задач GLUE (https://gluebenchmark.com/tasks (https://gluebenchmark.com/tasks))

Гиперпараметры модели

Гиперпараметрами модели являются:

- H — размерность скрытого пространства кодировщика
- L — количество слоёв-кодировщиков
- A — количество голов в механизме внимания.

В репозитории Google Research доступны для загрузки и использования несколько вариантов обученной сети в формате контрольных точек обучения модели популярного фреймворка TensorFlow[5]. В таблице в репозитории приведено соответствие параметров L и H и моделей. Использование моделей с малыми значениями гиперпараметров на устройствах с меньшей вычислительной мощностью позволяет сохранять баланс между производительностью и потреблением ресурсов. Также представлены модели с различным типом скрытия токенов при обучении, доступны два варианта: скрытие слова целиком (англ. whole word masking) или скрытие составных частей слов (англ. WordPiece masking).

Также модель доступна для использования с помощью популярной библиотеки PyTorch.

Точная настройка (Fine-tuning)

Этот этап обучения зависит от задачи, и выход сети, полученной на этапе предобучения, может использоваться как вход для решаемой задачи.

Пример: если решаем задачу построения вопросно-ответной системы, можем использовать в качестве ответа последовательность токенов, следующую за разделителем предложений. В общем случае дообучаем модель на данных, специфичных задаче: знание языка уже получено на этапе предобучения, необходима лишь коррекция сети.

Интерпретация этапа fine-tuning — обучение решению конкретной задачи при уже имеющейся общей модели языка.

Пример использования

Пример предказания пропущенного токена при помощи BERT в составе PyTorch. Скрытый токен — первое слово второго предложения:

In [ ]:

```
# Загрузка токенизатора и входные данные
tokenizer = torch.hub.load('huggingface/pytorch-transformers', 'tokenizer', 'bert-base-c
text_1 = "Who was Jim Henson ?"
text_2 = "Jim Henson was a puppeteer"

# Токенизация ввода, также добавляются специальные токены начала и конца предложения.
indexed_tokens = tokenizer.encode(text_1, text_2, add_special_tokens=True)
segments_ids = [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]

# Конвертирвание ввода в формат тензоров PyTorch
segments_tensors = torch.tensor([segments_ids])
tokens_tensor = torch.tensor([indexed_tokens])
encoded_layers, _ = model(tokens_tensor, token_type_ids=segments_tensors)

# Выбираем токен, который будет скрыт и позднее предсказан моделью
masked_index = 8
indexed_tokens[masked_index] = tokenizer.mask_token_id
tokens_tensor = torch.tensor([indexed_tokens])

# Загрузка модели
masked_lm_model = torch.hub.load('huggingface/pytorch-transformers', 'modelWithLMHead',
predictions = masked_lm_model(tokens_tensor, token_type_ids=segments_tensors)

# Предсказание скрытого токена
predicted_index = torch.argmax(predictions[0][0], dim=1)[masked_index].item()
predicted_token = tokenizer.convert_ids_to_tokens([predicted_index])[0]
assert predicted_token == 'Jim'
```