

Обработка естественного языка, лекция 7: трансформер

Пожидаев М. С.

22 декабря 2021 г.

Multi-head attention

Разобьём исходное векторное пространство входной последовательности:

1. Векторы *query*, *key* и *value* заменяются k проекциями (пусть $k = 8$).
2. Каждая проекция обрабатывается отдельно своим механизмом Self-attention.
3. Результат склеивается обратно.

Плюсы:

- ▶ обработка становится параллельной;
- ▶ повышается общее качество обработки.

Трансформер

Attention!

Не использует рекуррентные ИНС, (почти) и основан только на механизме Self-attention.

Функция

Трансформер — модель для машинного перевода текста (*transduction model*).

Наконец!

Разные токены во входной последовательности можно обрабатывать параллельно.

1. Входная последовательность (x_1, x_2, \dots, x_n) обрабатывается ячейкой Self-attention и преобразуется в последовательность (z_1, z_2, \dots, z_n) . Каждый x_t представлен в виде word embedding.
2. Каждый вектор z_t затем обрабатывается дополнительной линейной ячейкой, и это можно делать параллельно!

Блок декодера похож на блок кодера, но содержит дополнительную ячейку и имеет следующую структуру:

1. Ячейка Self-attention (как в кодере).
2. Дополнительная ячейка Self-attention с модифицированными связями функционально близкая к вниманию в варианте Бахданау.
3. Линейная ячейка (как в кодере).

Структура трансформера

Трансформер содержит $2k$ блоков:

- ▶ k блоков кодера, каждый из которых включает:
 - ▶ одну ячейку Self-attention;
 - ▶ одну линейную ячейку;
- ▶ k блоков декодера, каждый из которых содержит:
 - ▶ две ячейки Self-attention, одна из которых повторяет назначение внимания Бахданау;
 - ▶ одну линейную ячейку как в кодере;
- ▶ ещё одна линейная ячейка для порождения выходных токенов.

k — параметр модели, обычно равен 6.

Три типа внимания

1. Обычный Self-attention в кодере, в котором векторы , *query*, *key* и *value* вычисляются обычным способом на текущей последовательности.
2. Self-attention в декодере, похожий на ячейку в кодере, но не имеющий доступ к элементам последовательности справа от текущего.
3. Дополнительная ячейка в декодере, в котором *query* берутся с предыдущего шага декодера, а *key* и *value* — с выхода кодера.

Уже было в Seq2seq!

Декодер работает по схеме, напоминающей работу декодера в Seq2seq, в которой совершается несколько шагов, на каждом из которых генерируется новый элемент выходной последовательности.

1. Новый элемент генерируется дополнительной линейной ячейкой.
2. Процесс продолжается, пока не будет получен токен, обозначающий конец сегмента.

Непонятно, где это было!

Self-attention не сохраняет информацию о положении исходного элемента, поэтому добавим дополнительное кодирование (position encoding).

$$PE(pos, 2i) = \sin(pos/10000^{2i/d})$$

$$PE(pos, 2i + 1) = \cos(pos/10000^{2i/d})$$

- ▶ pos — индекс входного токена;
- ▶ $2i$ и $2i + 1$ — индекс компоненты вектора;
- ▶ d — размерность векторного пространства.

BERT (*Bidirectional Encoder Representations from Transformers*) — двунаправленная многофункциональная модель, построенная как кодирующая часть трансформера. Путём дополнительного слоя позволяет решать широкий круг задач, включая:

- ▶ ответы на вопросы (*question answering*);
- ▶ вывод на естественном языке (*NL inference* и др.)

Декодера нет!

Поскольку декодер отсутствует, возникает вопрос, как модель обучать. Использовались две задачи, которые можно решать просто на массиве текста:

1. Скрытие и восстановление 15% слов (*masked language modelling*)/
2. Определение следующего предложения (*next sentence prediction*).

Спасибо за внимание!

Веб-сайт: <https://marigostra.ru/>

E-mail: mSP@luwrain.org