Deep learning in NLP

Part 3: Transformers, Bert & GPT

Zeinab Rahimi, Feb 2022

Transformers, 2017

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

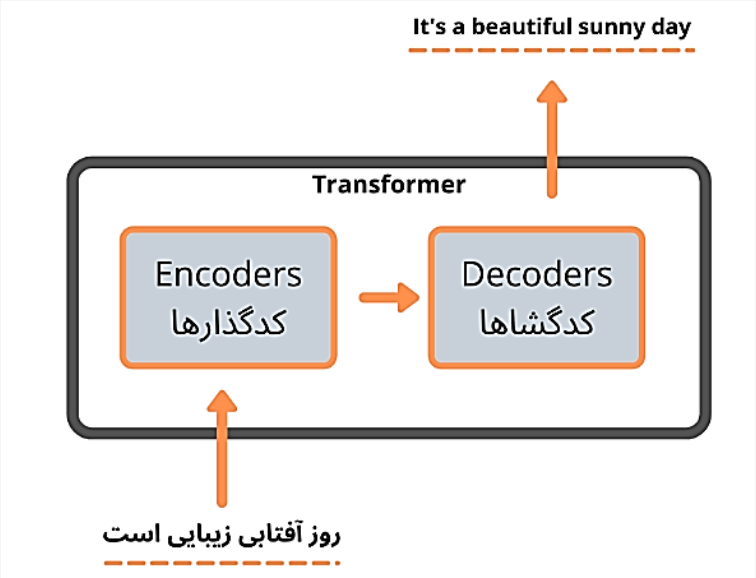
Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

ec 2017

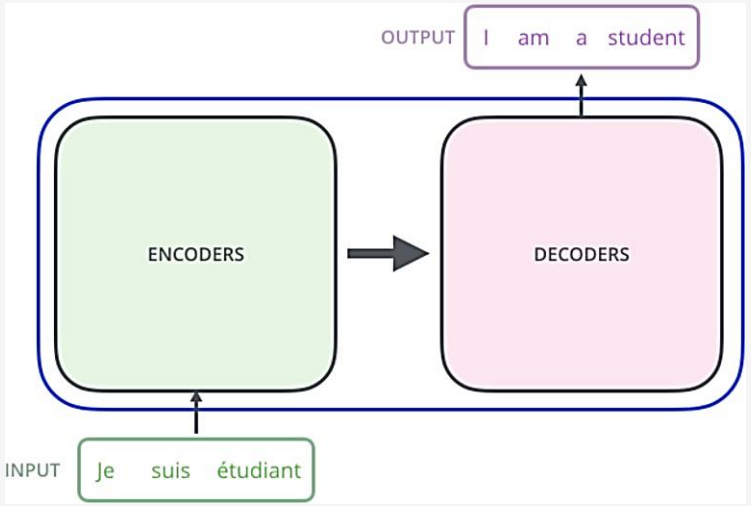
Transformers

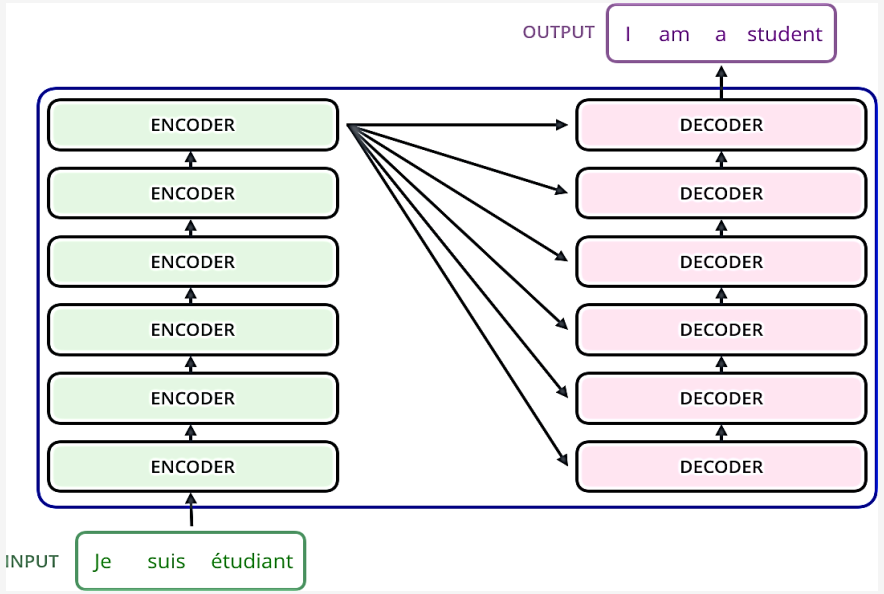


- استفاده از مکانیزم توجه و یادگیری رابطهٔ بین کلمات
- در ساده‌ترین شکل، ترنسفورمر شامل دو مکانیزم جداگانه:
  - یک کدگذار (Encoder) برای خواندن متن ورودی
  - یک کدگشا (Decoder) برای بیان پیش‌بینی محتمل را برای تسک مشخص‌شده
- لایهٔ انکودر ترنسفورمرها دنباله‌ای از کلمات ورودی را به صورت یکجا می‌خواند.
  - برعکس مدل‌های قبلی (RNN و LSTM) که متن ورودی را به ترتیب از چپ به راست یا از راست به چپ می‌خواندند.

Transformers

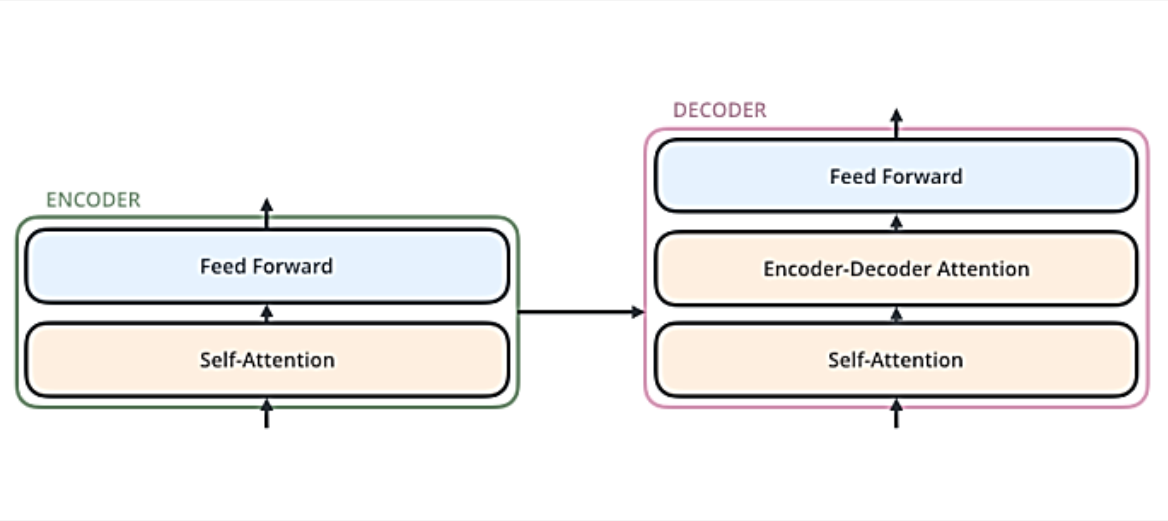
- ناکارآمدی زیربنایی شبکه‌های بازگشتی
  - در دنباله های طولانی و موازی سازی
- معرفی توسط گوگل در سال ۲۰۱۷
- یک بلاک ترنسفورمر: مجموعه ای از انکودها و دیکودرها
- بررسی با یک تسک ترجمه ماشینی
- تعداد لایه ها: هایپرپارامتر





http://jalammar.github.io/illustrated-bert/

Transformers

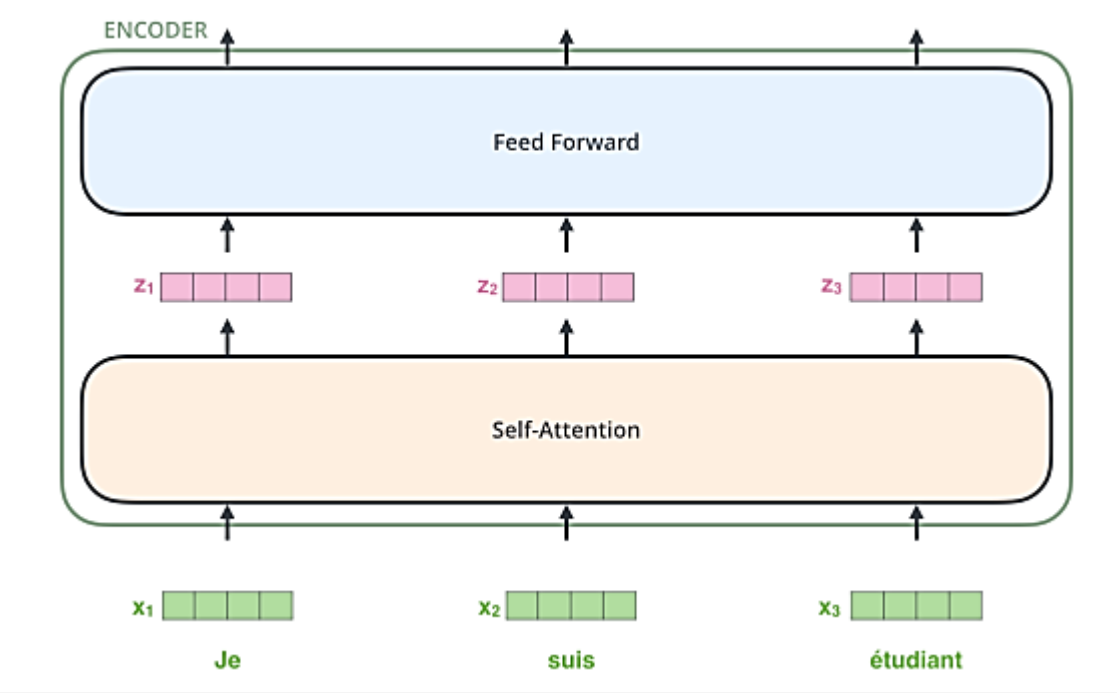


- **انکودر**
  - متن ورودی را پردازش می‌کند، قسمت‌های مهم آن را جست‌وجو می‌کند و بر اساس میزان ارتباط هر کلمه با سایر کلمات تشکیل‌دهنده جمله، برای تک تک آن‌ها یک تعبیه ایجاد می‌کند.
  - یک لایه Self-Attention
  - یک لایه Feed Forward
- **دیکودر**
  - مشابه انکودر+ یک لایه Encoder-decoder attention
  - دیکدر: خروجیِ انکدر که یک تعبیه است، دریافت می‌کند و مجدداً آن را به یک خروجی متنی تبدیل می‌کند؛

    به بیان دیگر، نسخه ترجمه‌شده ورودی متنی را خروجی می‌دهد.

http://jalammar.github.io/illustrated-bert/

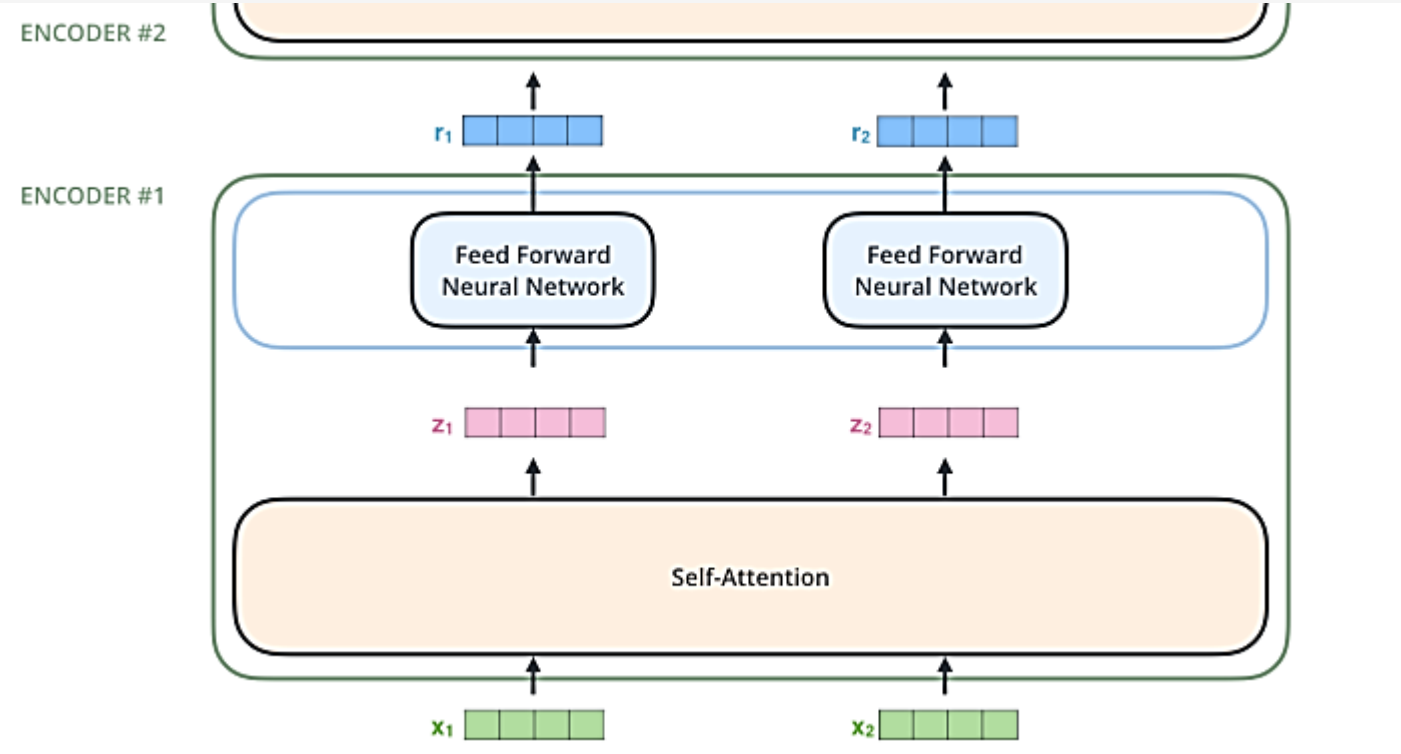
Transformers: بخش Encoder



- تبدیل ورودی به بردار جاسازی
- انکودر بابت هر ورودی یک بردار Z تولید می کند: ورودی لایه FF

http://jalammar.github.io/illustrated-bert/

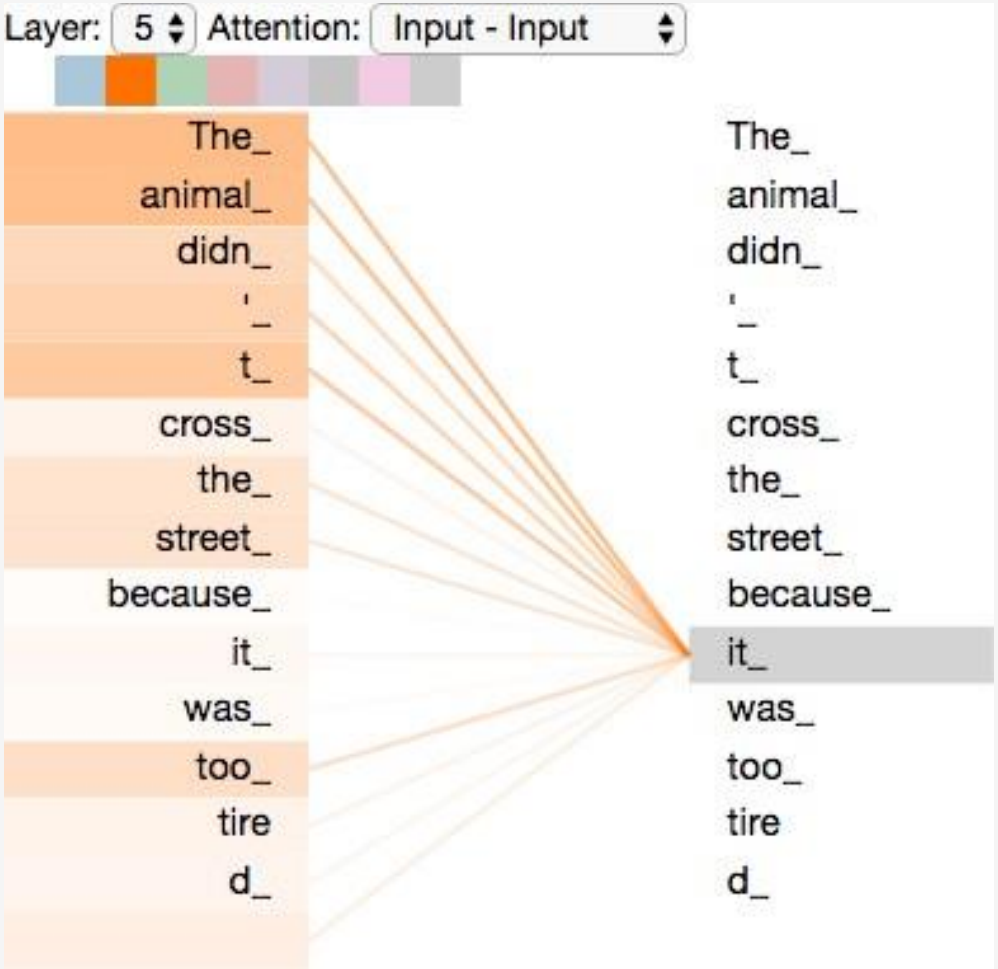
Transformers



- نمایش لایه self-attention به صورت یکپارچه:
  - در نظر گرفتن وابستگی seq ورودی
- لایه FF جدا به ازای هر کلمه

http://jalammar.github.io/illustrated-bert/

Self Attention



- به بخش‌های مختلف جمله نگاه می‌کند و تلاش می‌کند اطلاعات معنایی و نحوی بیشتری یاد بگیرد.
- پیاده سازی تاریخچه کلمات قبل و بعد که در بازگشتی ها بود.
- مشخص کردن وزن برای هر کلمه

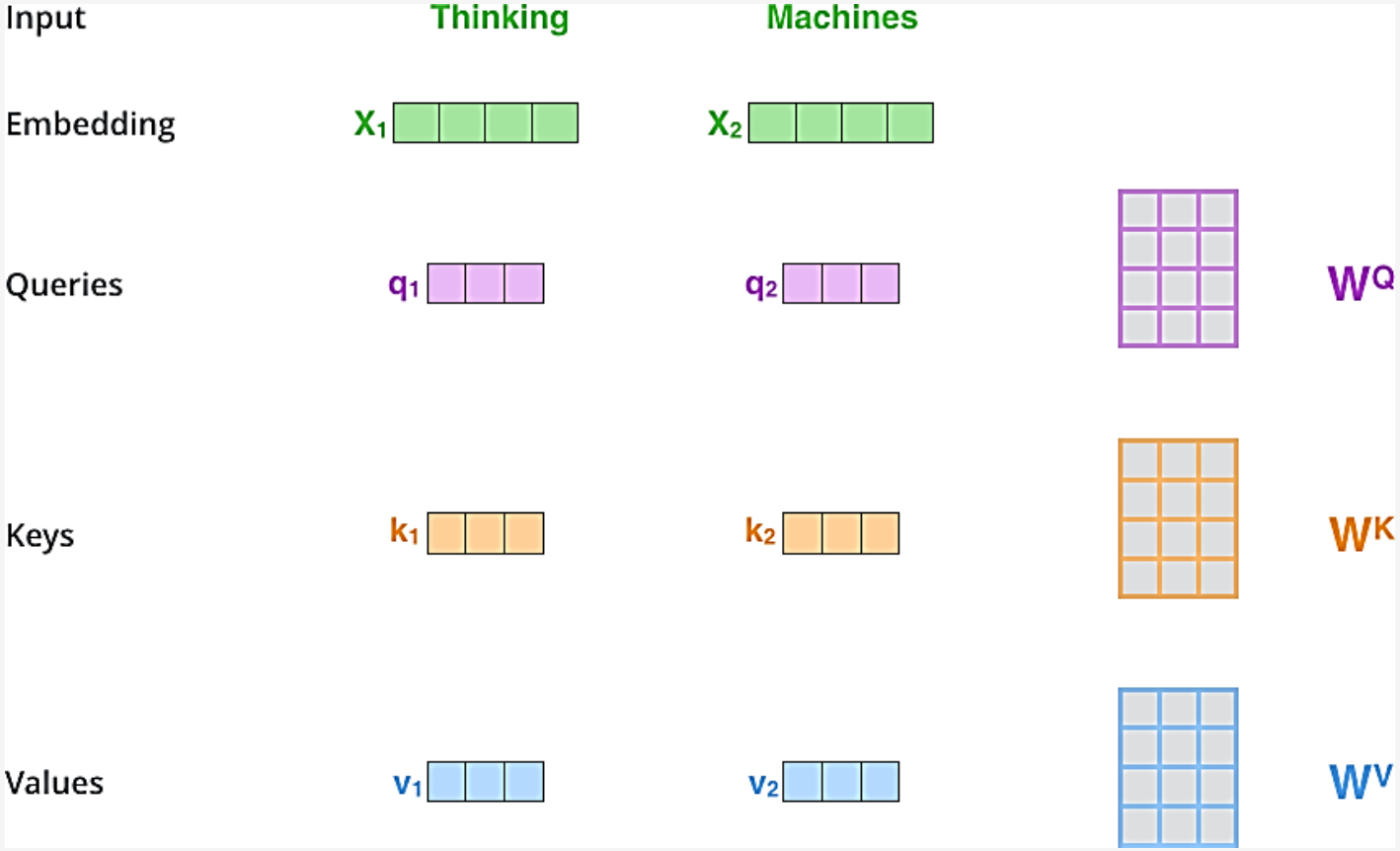
The animal didn't cross the street because it was too tired.
The animal didn't cross the street because it was too wide.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

http://jalammar.github.io/illustrated-bert/

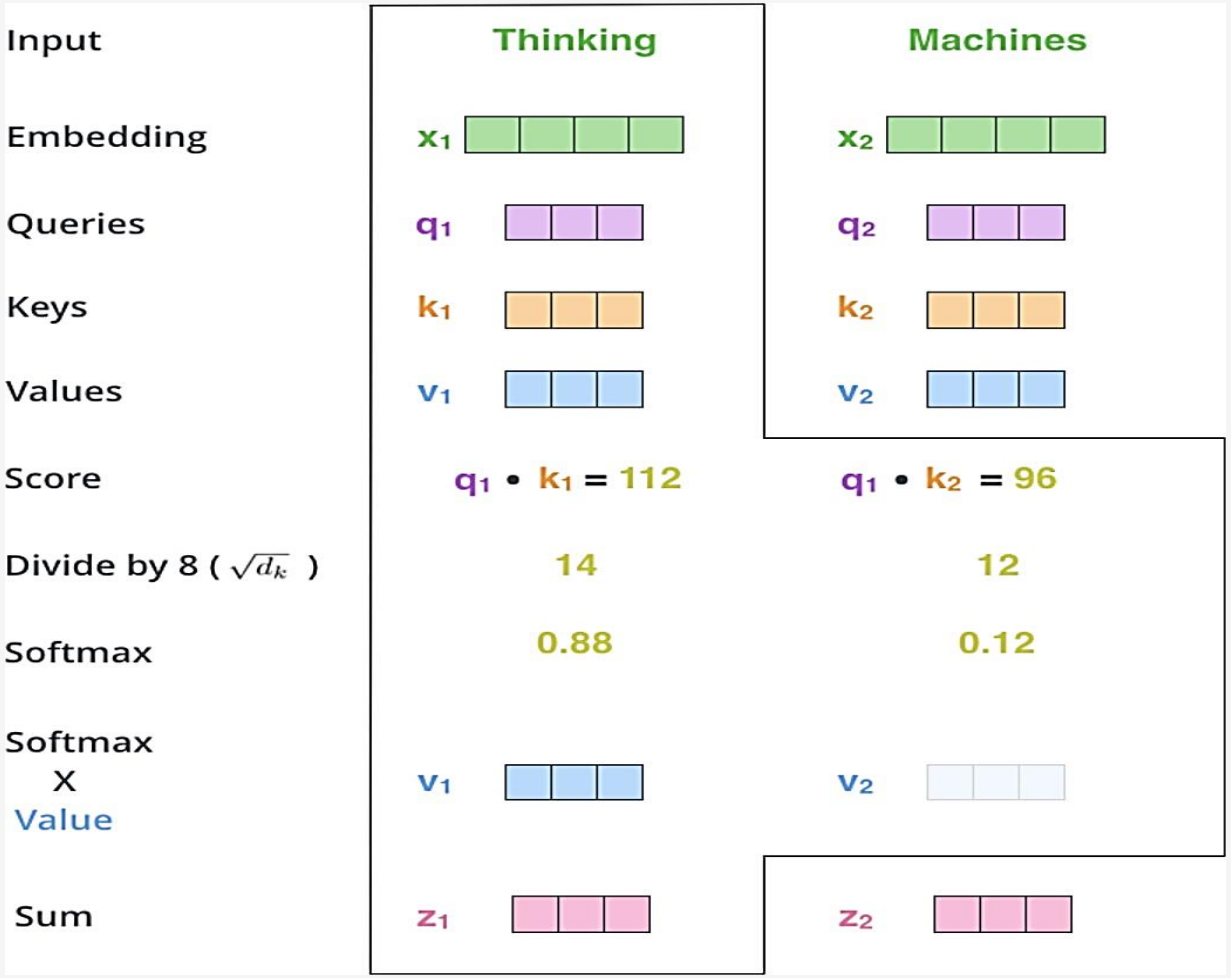
Equation in (Vaswani et al., 2017)

Self Attention



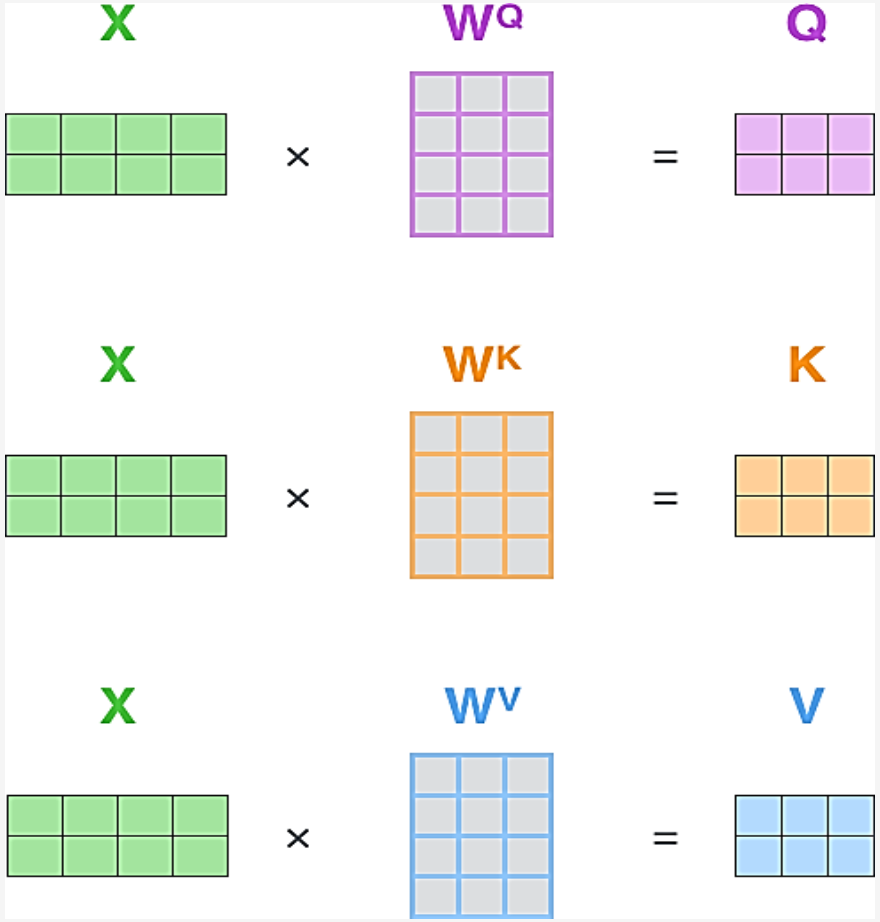
- در هر مرحله ۳ بردار Q، K و V تشکیل می دهیم. چطور؟
- ۳ ماتریس وزن داریم، ضرب در بردار کلمه

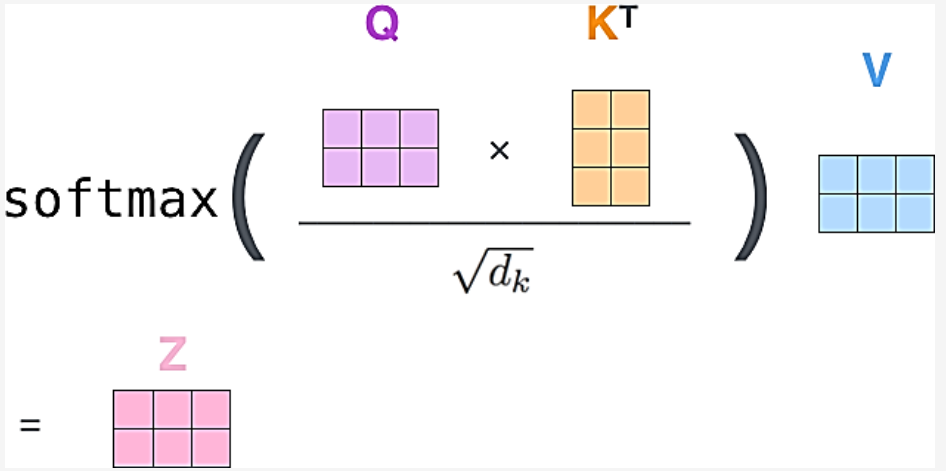
Self Attention



- بعد از ورود هر کلمه به شبکه باید رتبه بندی باید داشته باشیم.
  - چطور امتیاز بدهیم؟
    - ماتریس q کلمه ضرب در k همه کلمات
    - با تقسیم بر رادیکال بُعد k نرمالسازی
    - عبور از softmax (بازه ۰ و ۱ برای شناسایی مهم ها)
    - ضرب در بردار v کلمات دنباله
    - z به دست آمد.

Self Attention

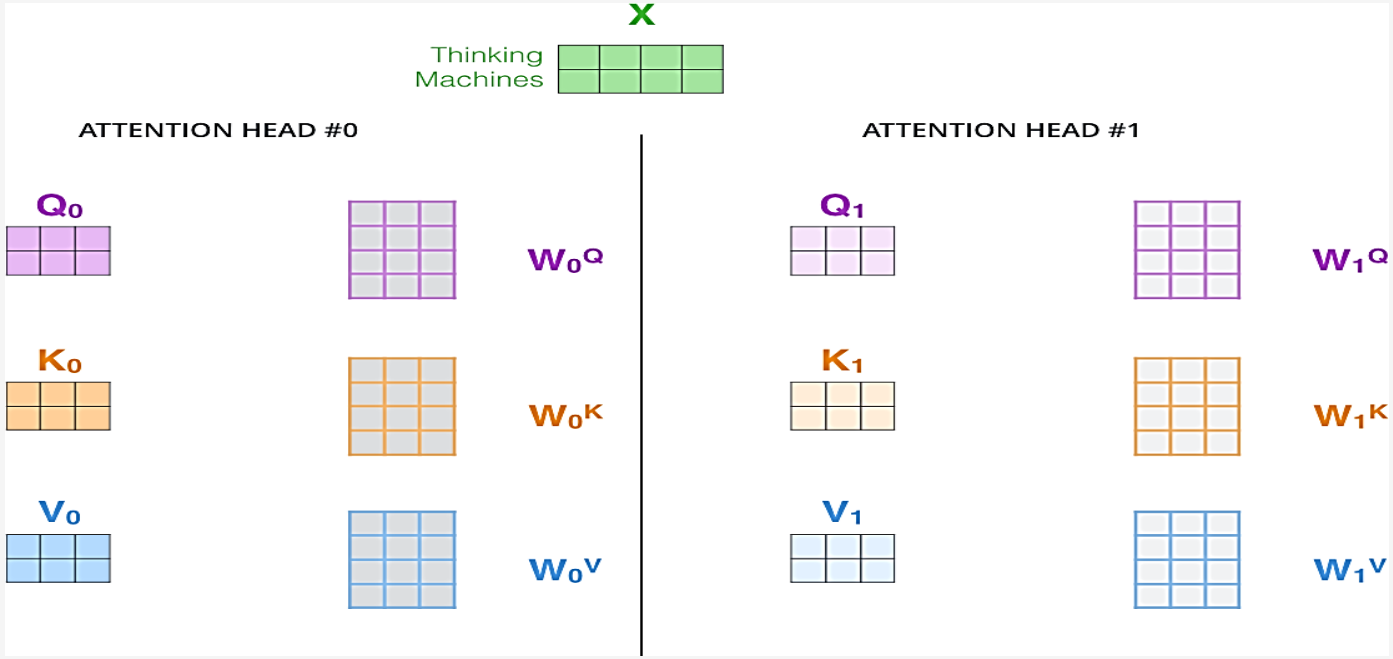


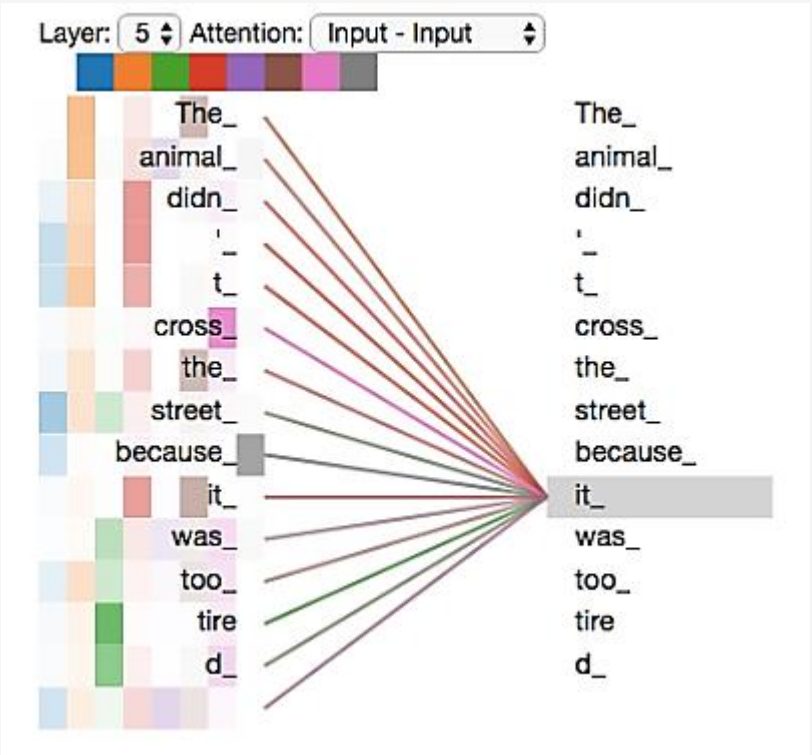


http://jalammar.github.io/illustrated-bert/

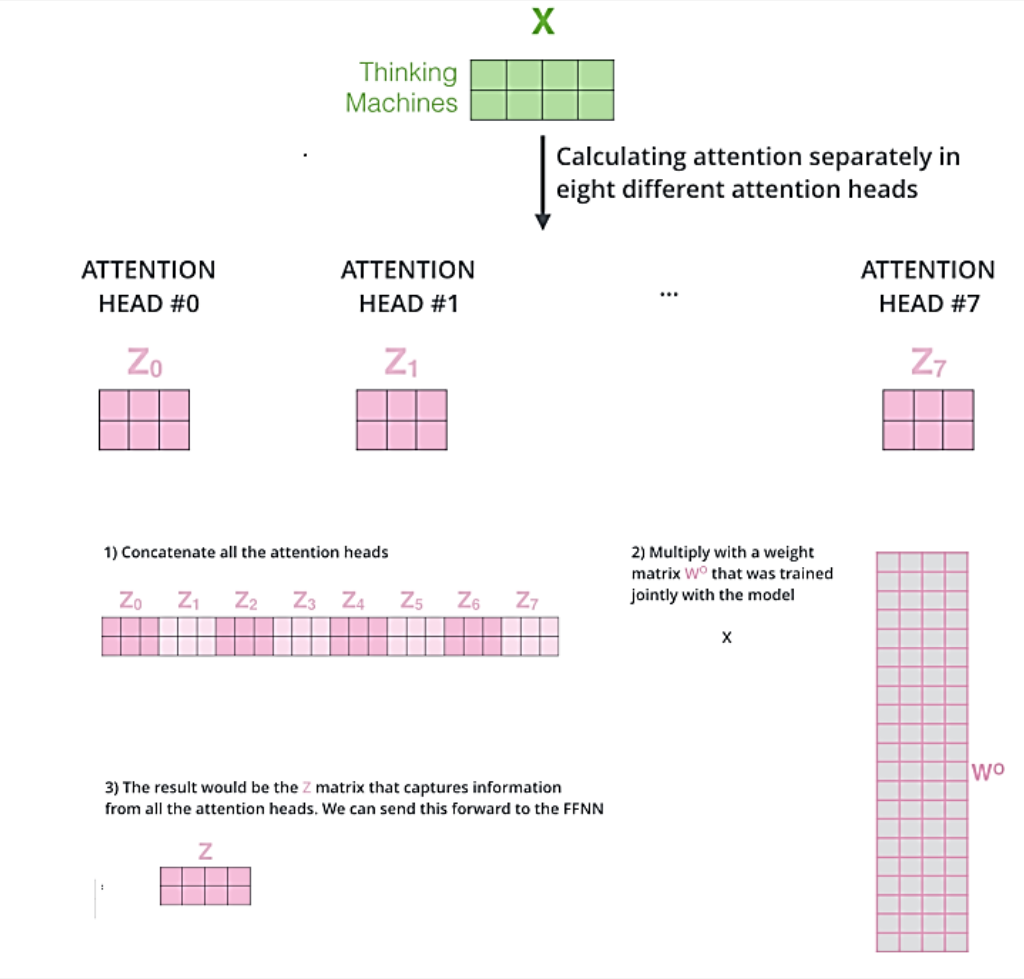
Multi Head Self Attention

- همزمان به چند جنبه اطلاعاتی و چند فضای بازنمایی توجه دارد.
- امکان موازی انجام دادن



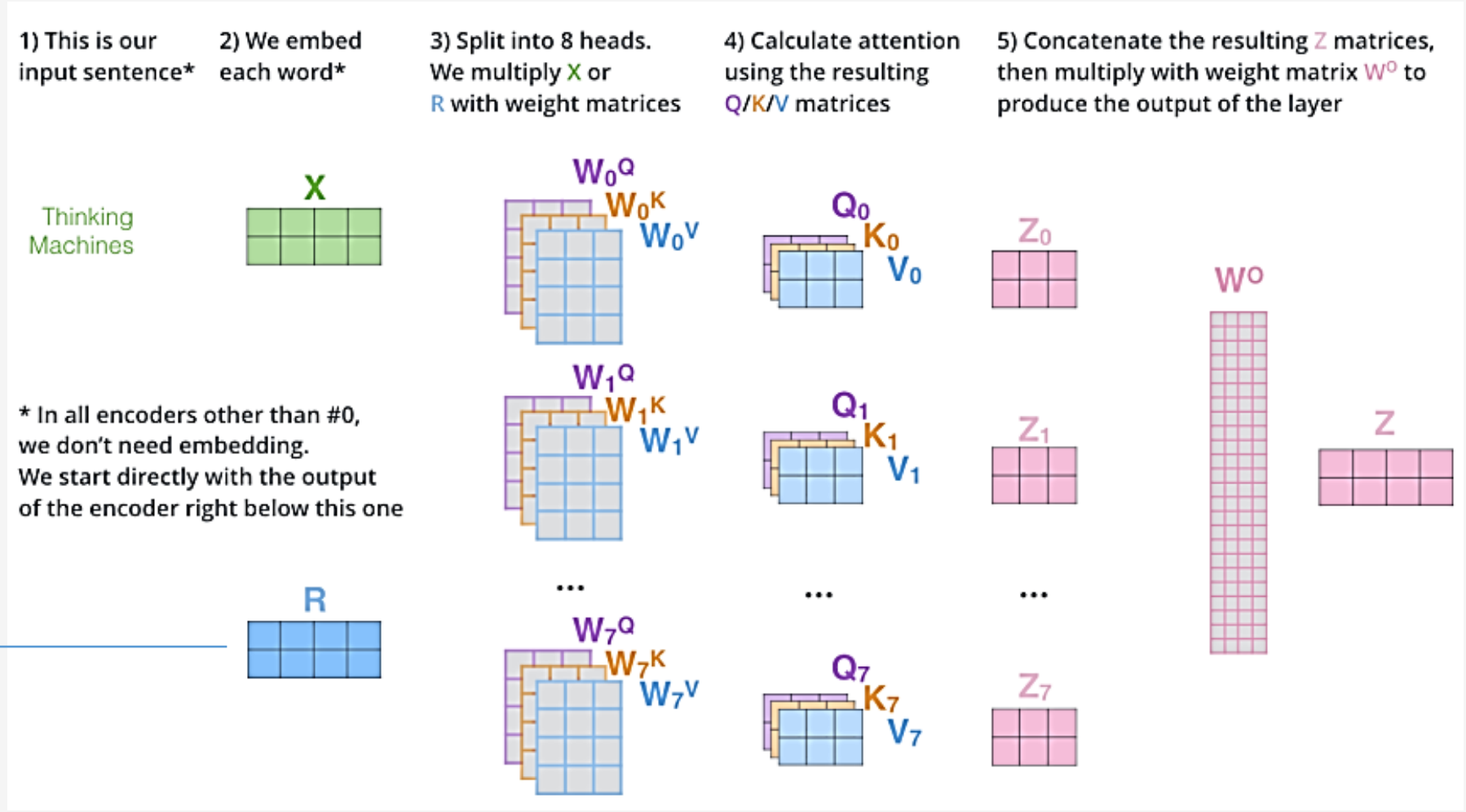


Multi Head Self Attention



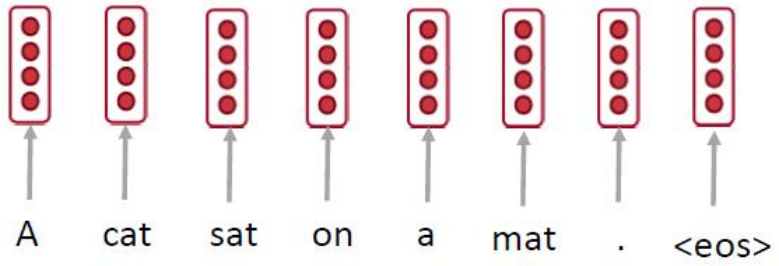
- تکرار همان کارها به ازای هر هد و بدست آوردن چند Z
- تبدیل چند Z به یک Z مثل قبل
- وصل کردن Z ها به هم و ضرب در ماتریس مناسب

Multi Head Self Attention

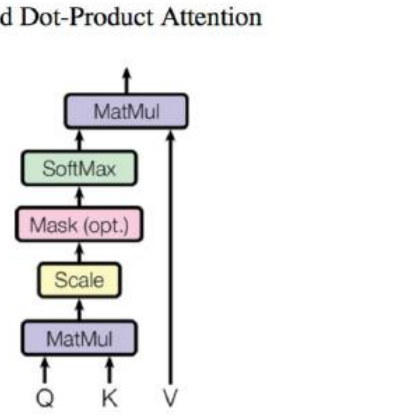


خروجی لایه قبل

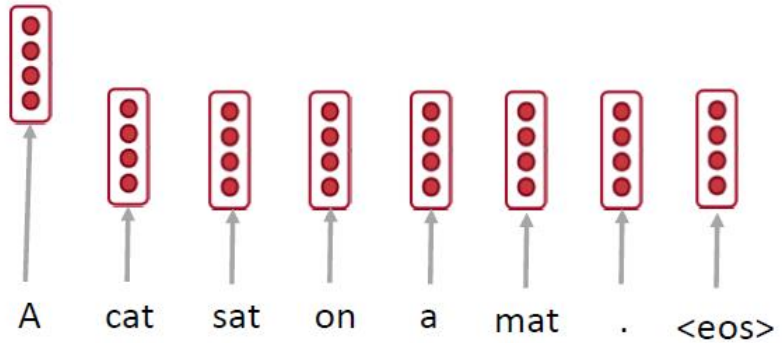
Self-attention: A Running Example



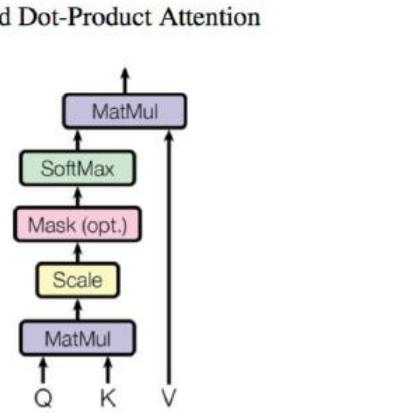
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



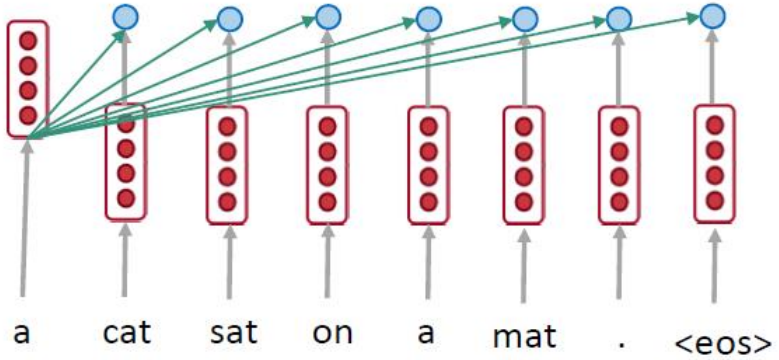
Self-attention: A Running Example



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

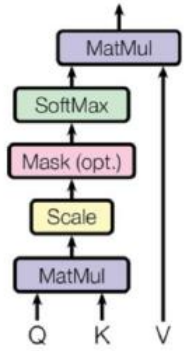


Self-attention: A Running Example

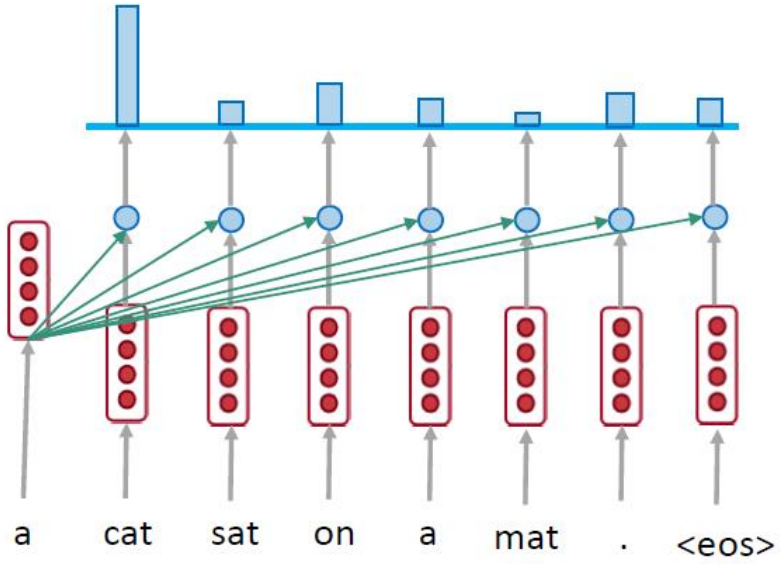


$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

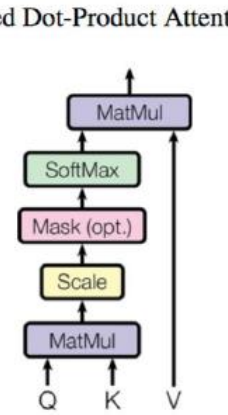
Scaled Dot-Product Attention



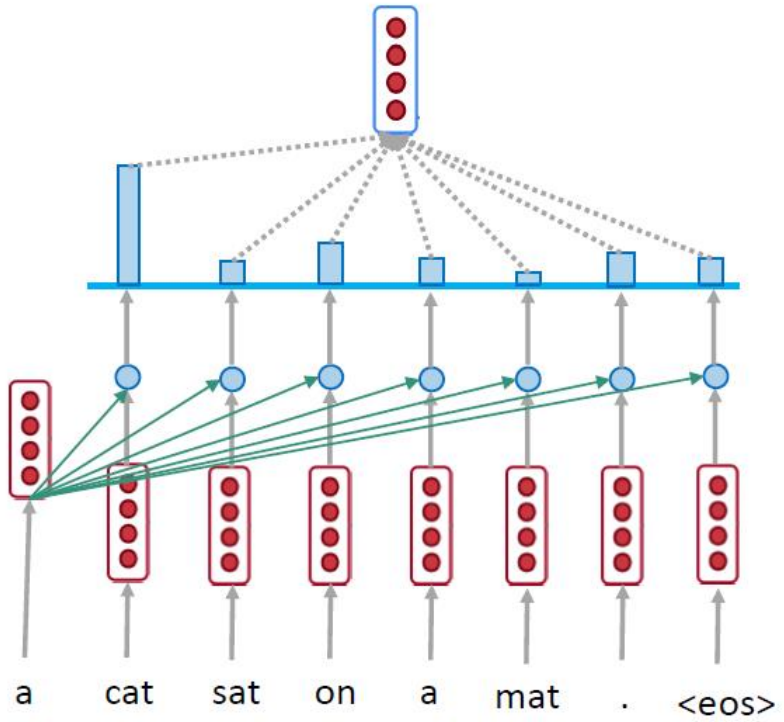
Self-attention: A Running Example



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

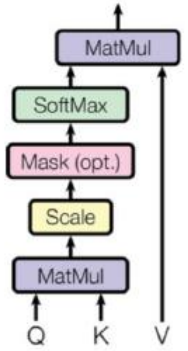


Self-attention: A Running Example

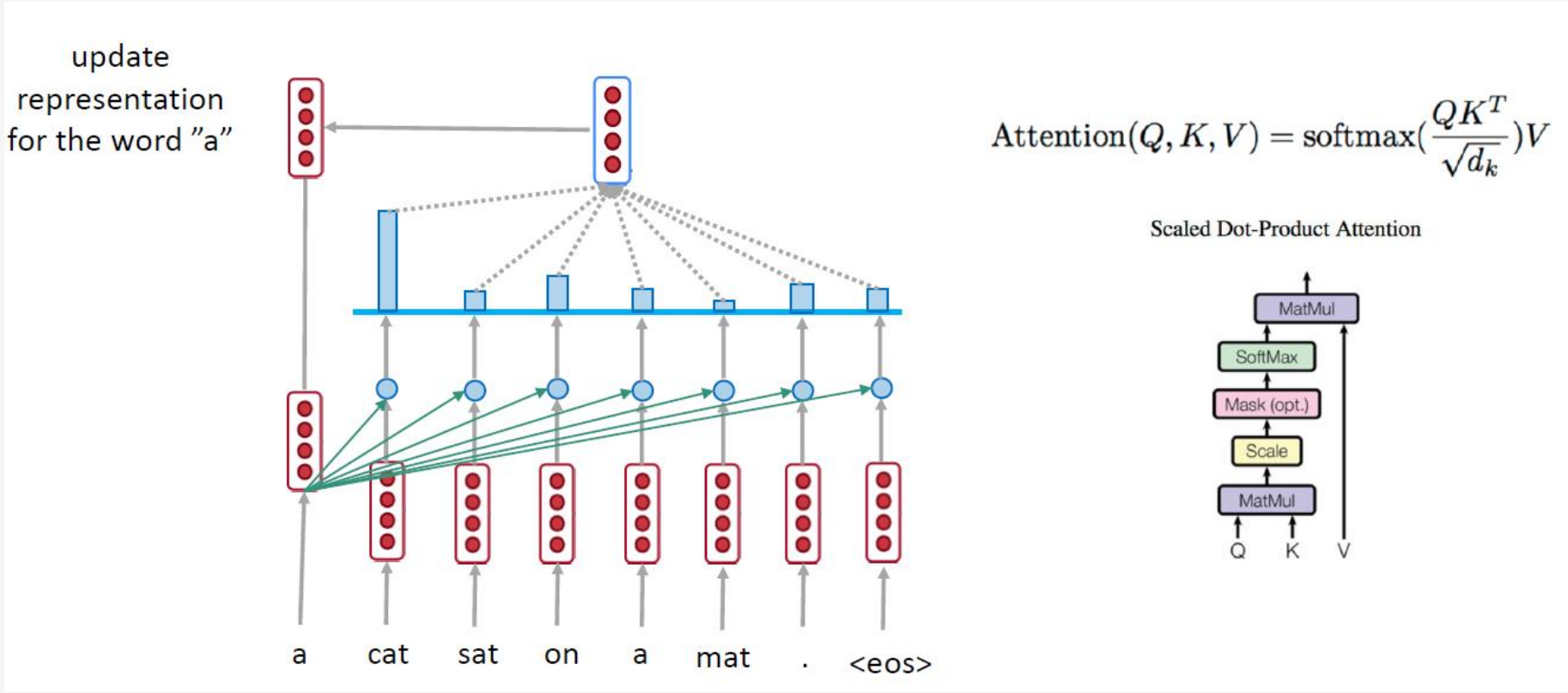


$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Scaled Dot-Product Attention



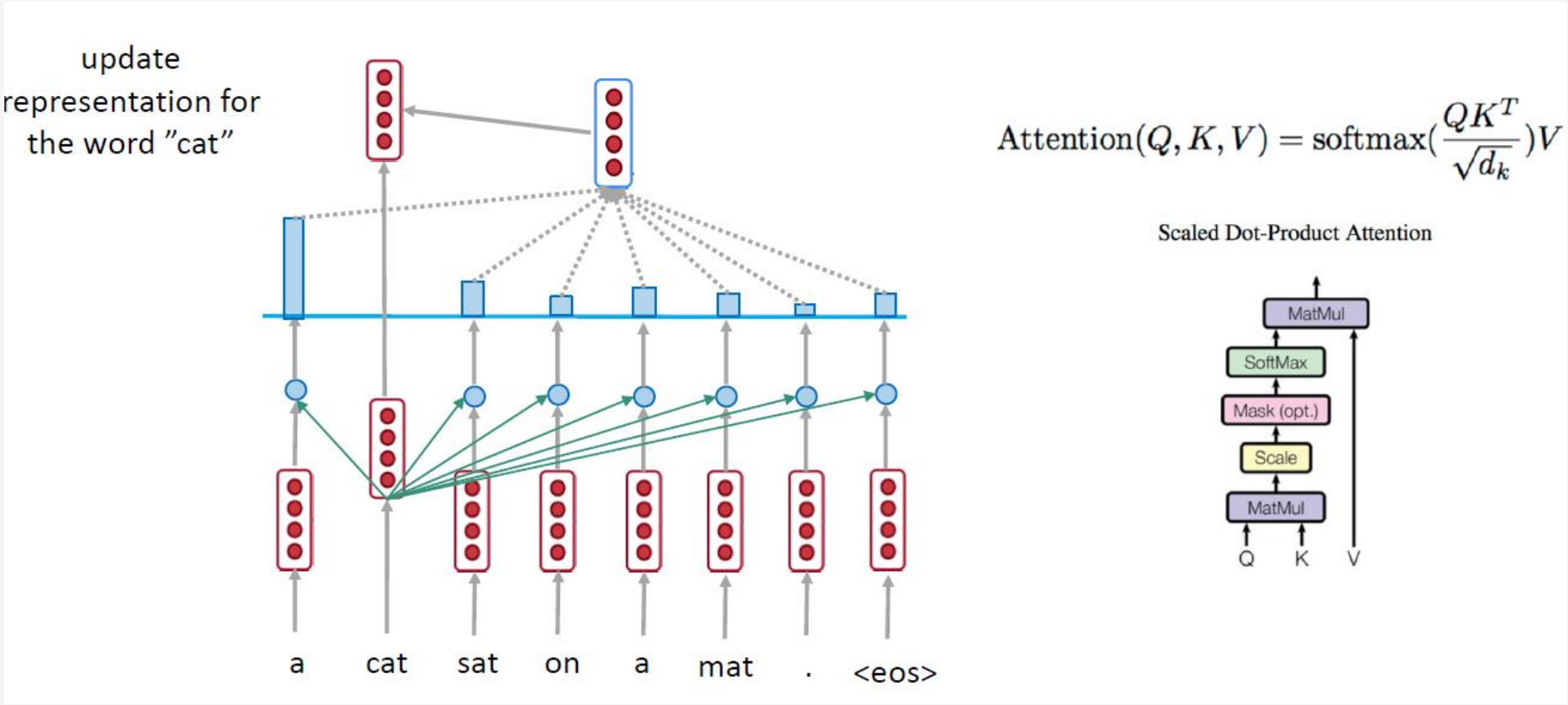
Self-attention: A Running Example



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Scaled Dot-Product Attention

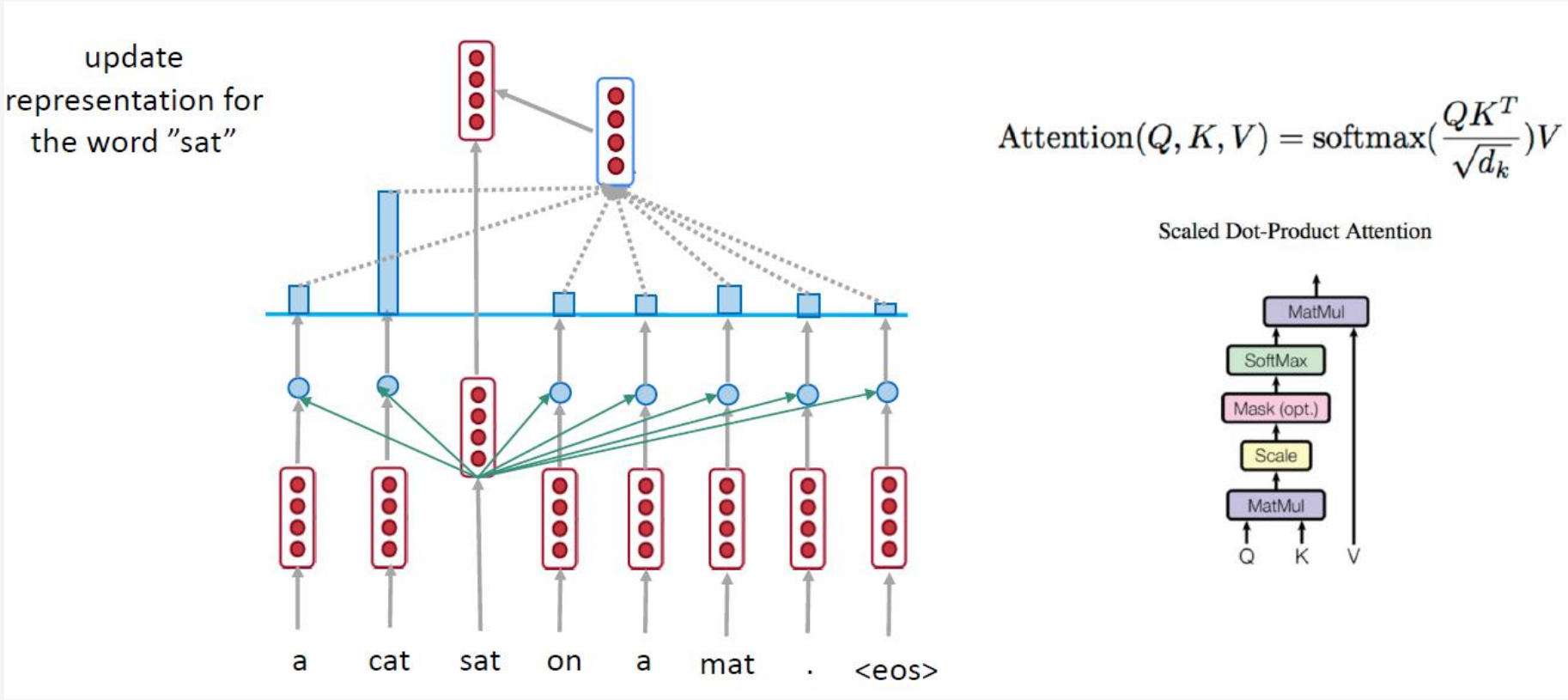
Self-attention: A Running Example



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Scaled Dot-Product Attention

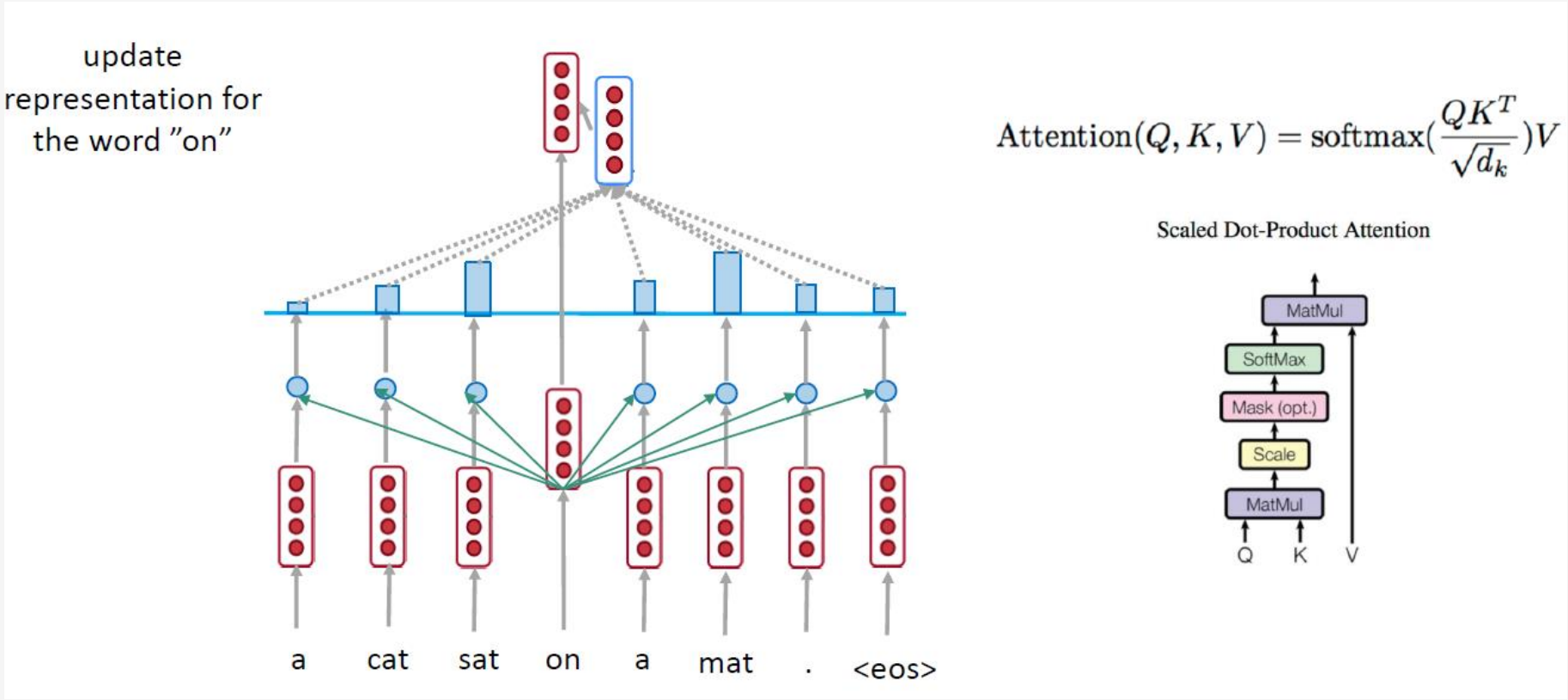
Self-attention: A Running Example



$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Scaled Dot-Product Attention

Self-attention: A Running Example

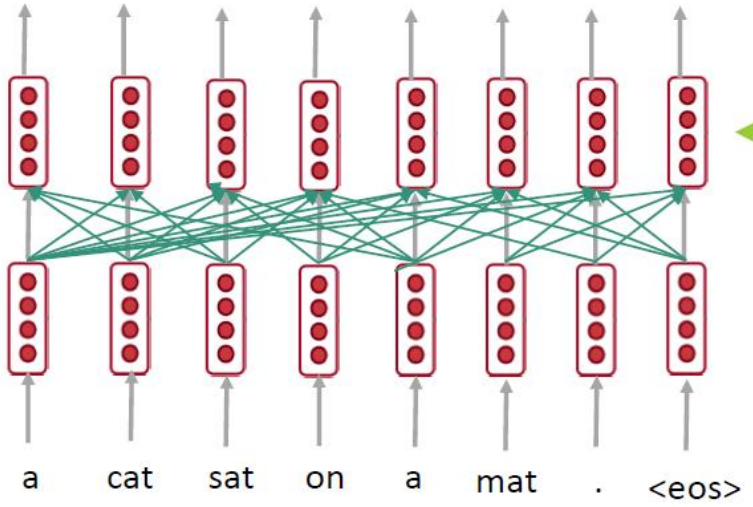


$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Scaled Dot-Product Attention

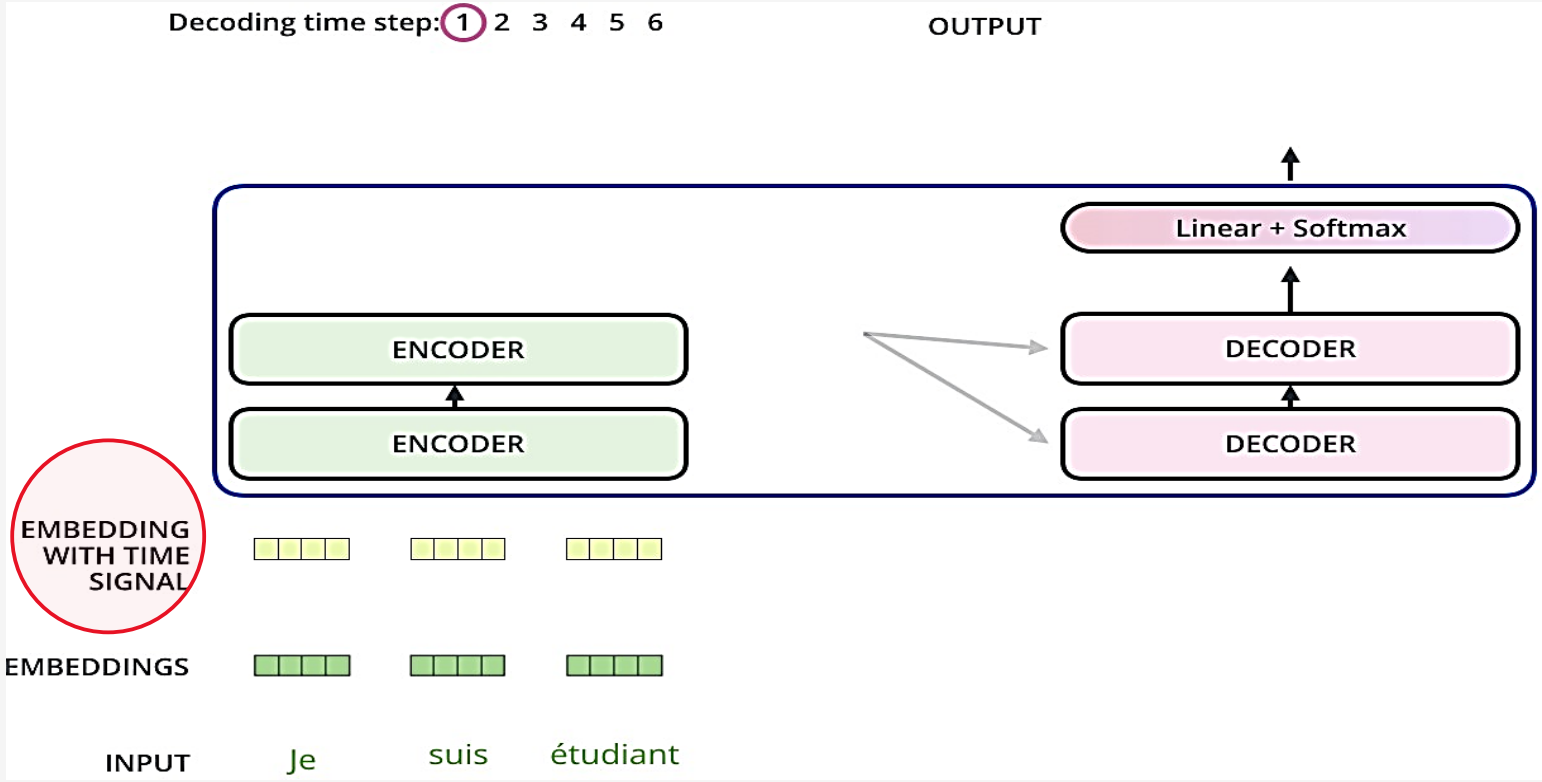
Self-attention: A Running Example

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



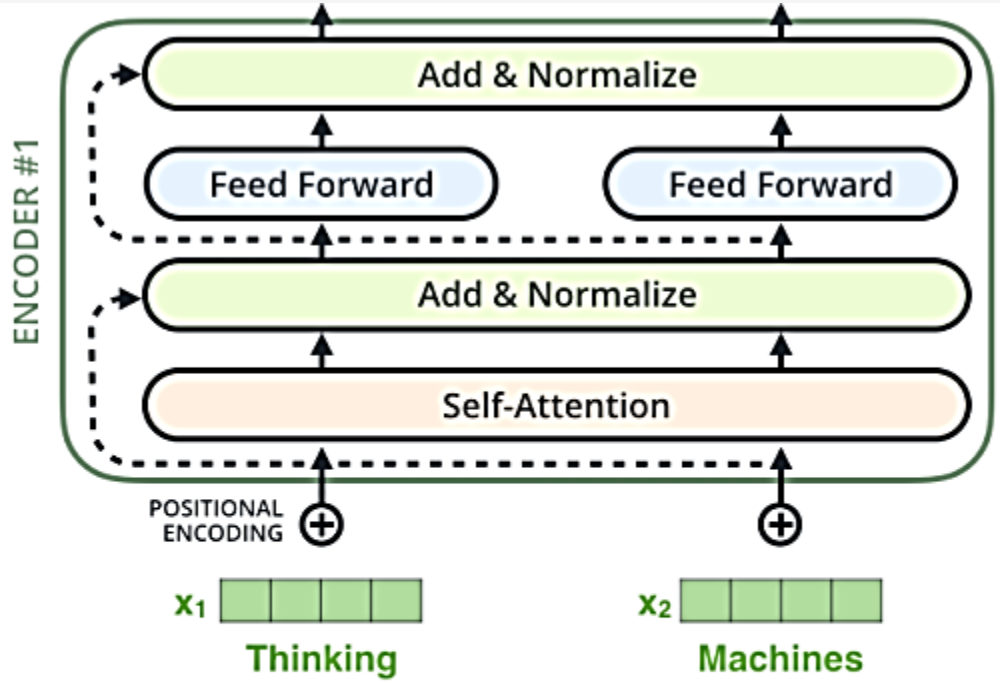
Updated representations
for each word
(one layer)

پیاده سازی توالی

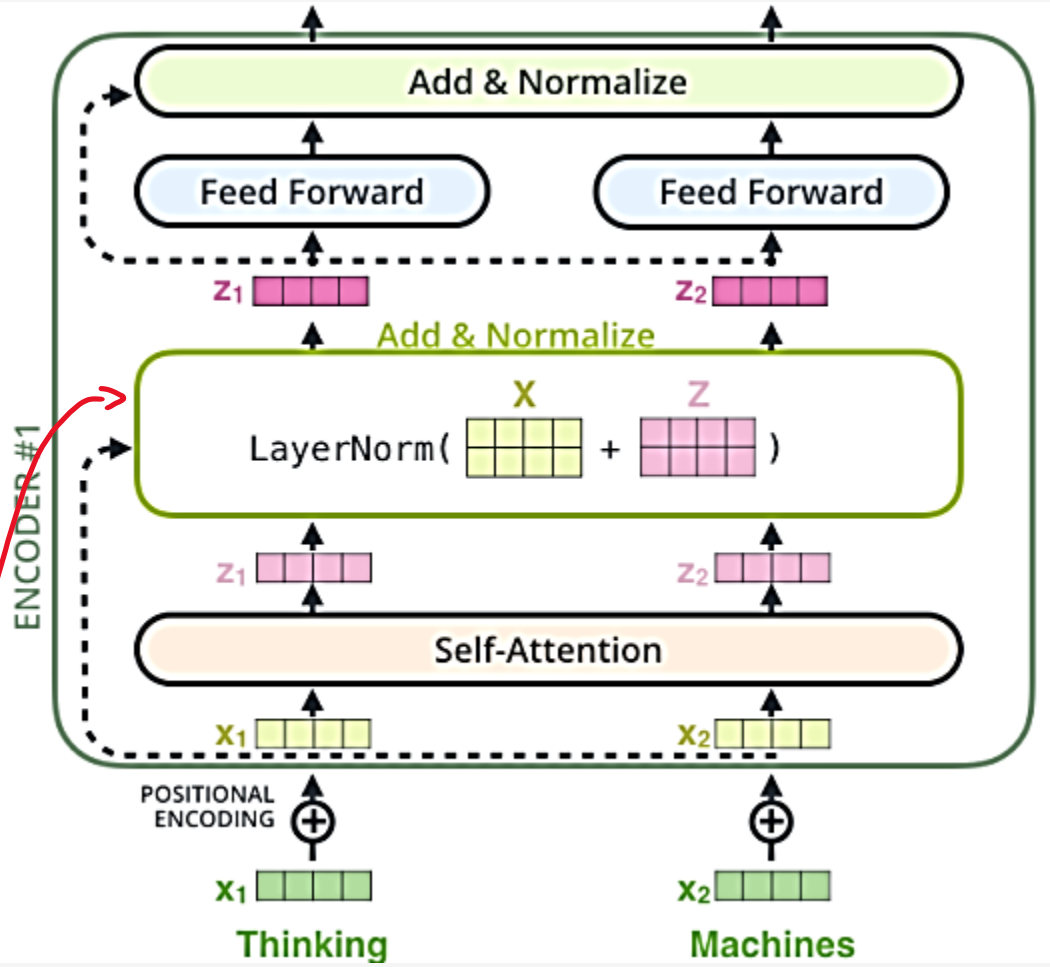


http://jalammar.github.io/illustrated-bert/

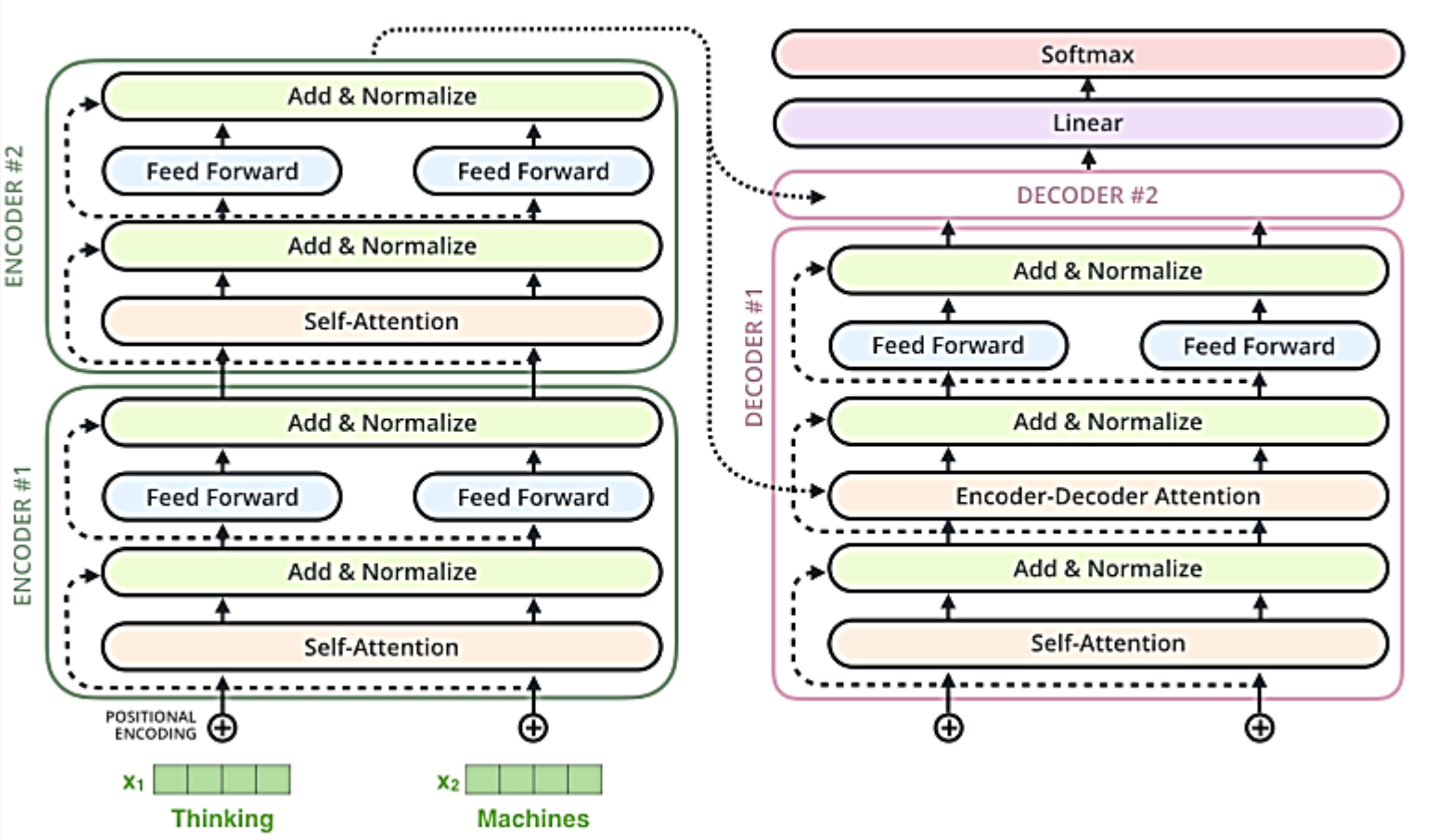
Transformers: بخش Add & Normalize



- برای افزایش سرعت آموزش
- عبور ماتریس جمع X و Z از یک تابع نرمال

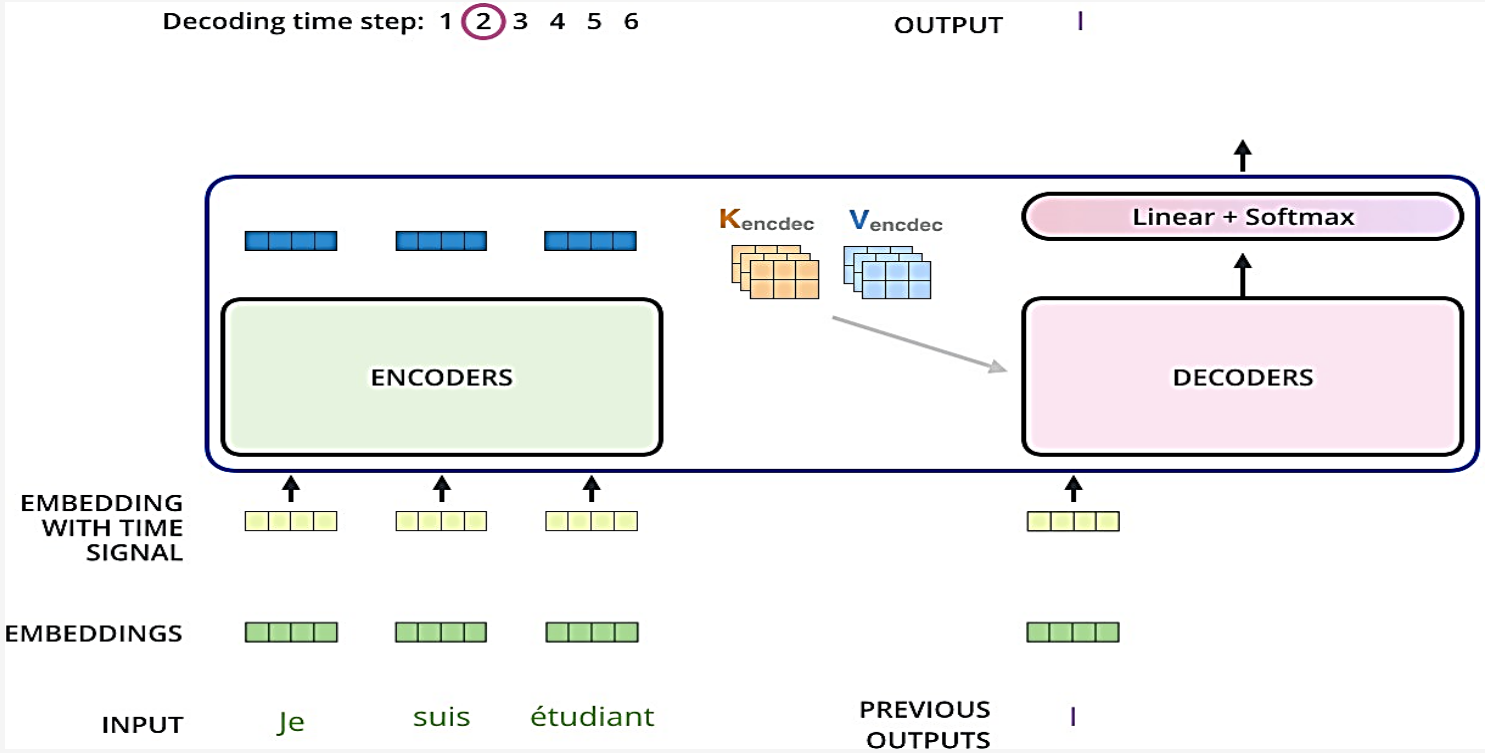


Transformers :بخش Add & Normalize



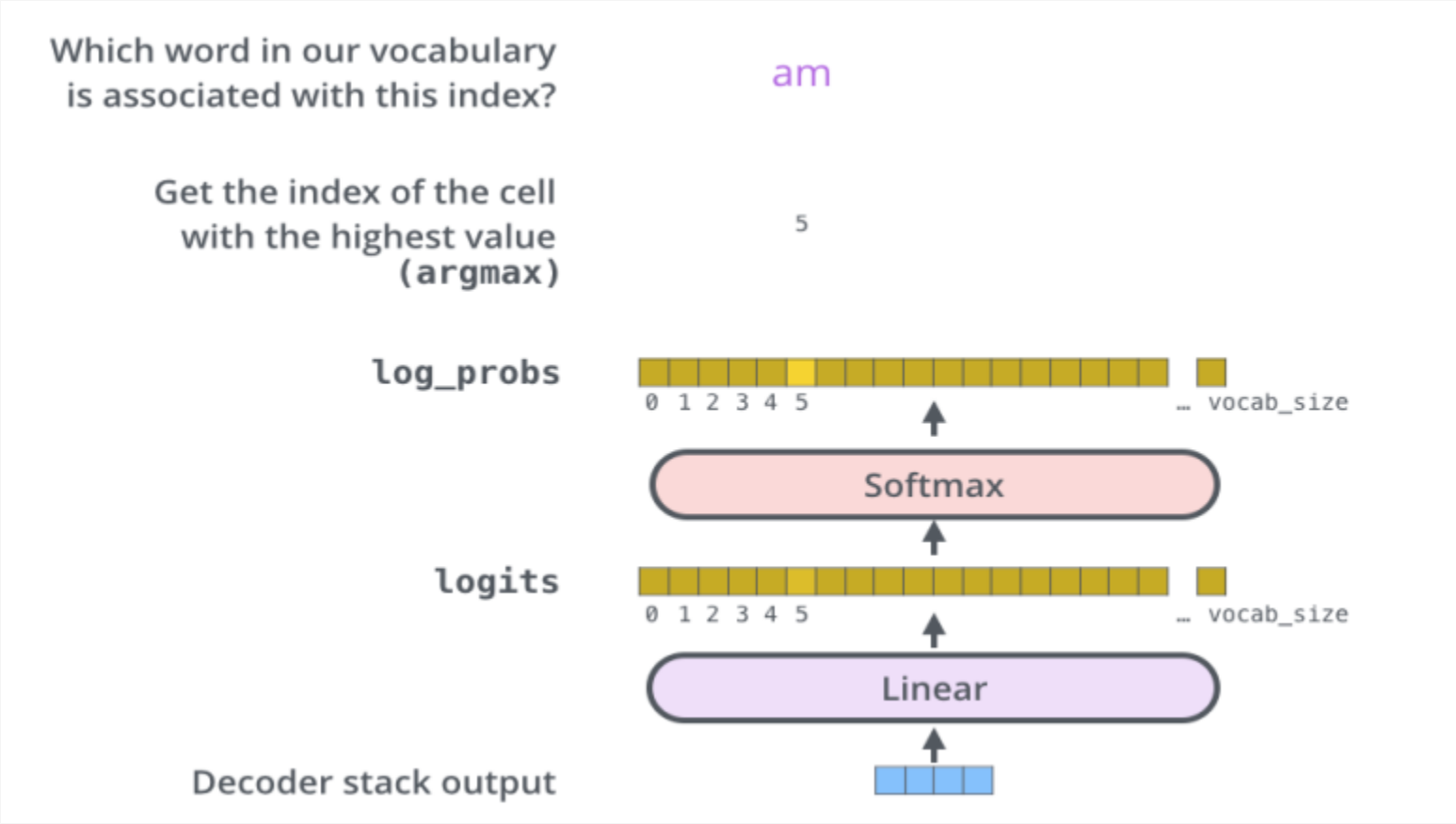
Transformers: بخش Decoder

- تا اینجا دیدیم ورودی بعد از لایه توجه نرمال شده و به FF می رود.
- خروجی FF ورودی دیکودر است.
- دیکودر مشابه انکودر است، فقط یک لایه encoder-decoder attention بیشتر دارد.
  - همان Multi head attention، فقط ماتریس های key و Value را از انکودر می گیرد.

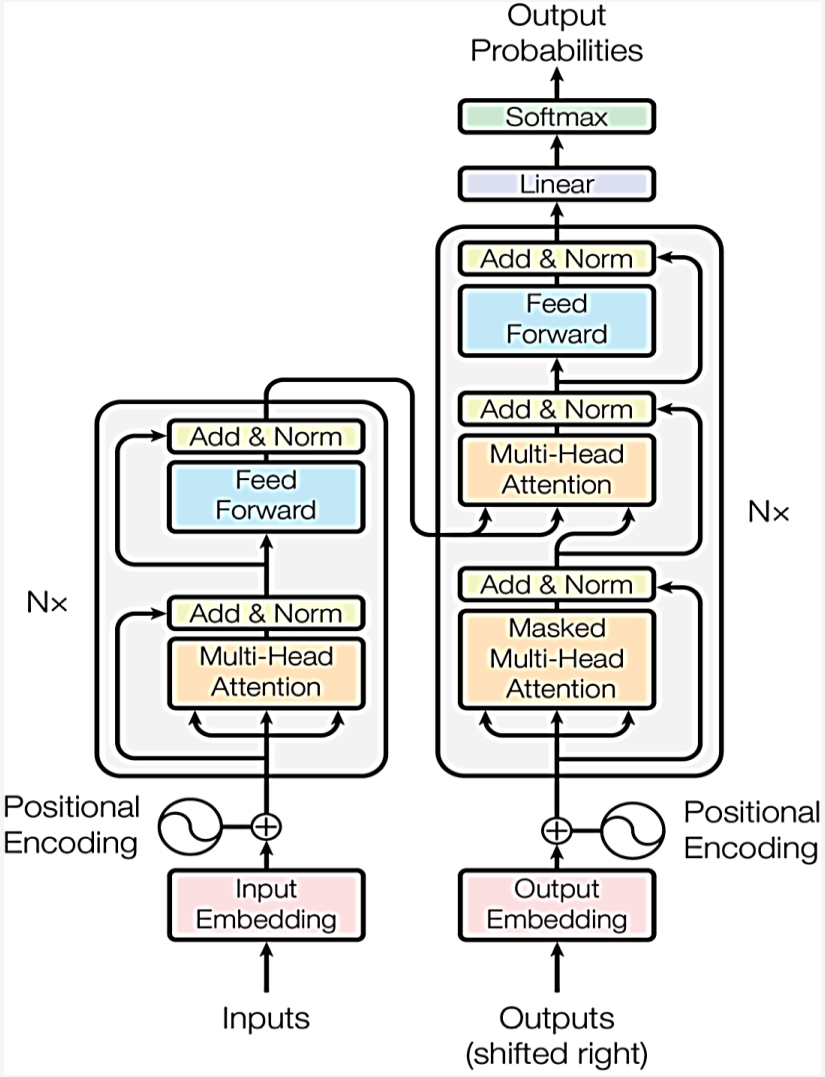


Transformers: لایه softmax

- مشابه قبلی ها



Transformers: معماری کلی

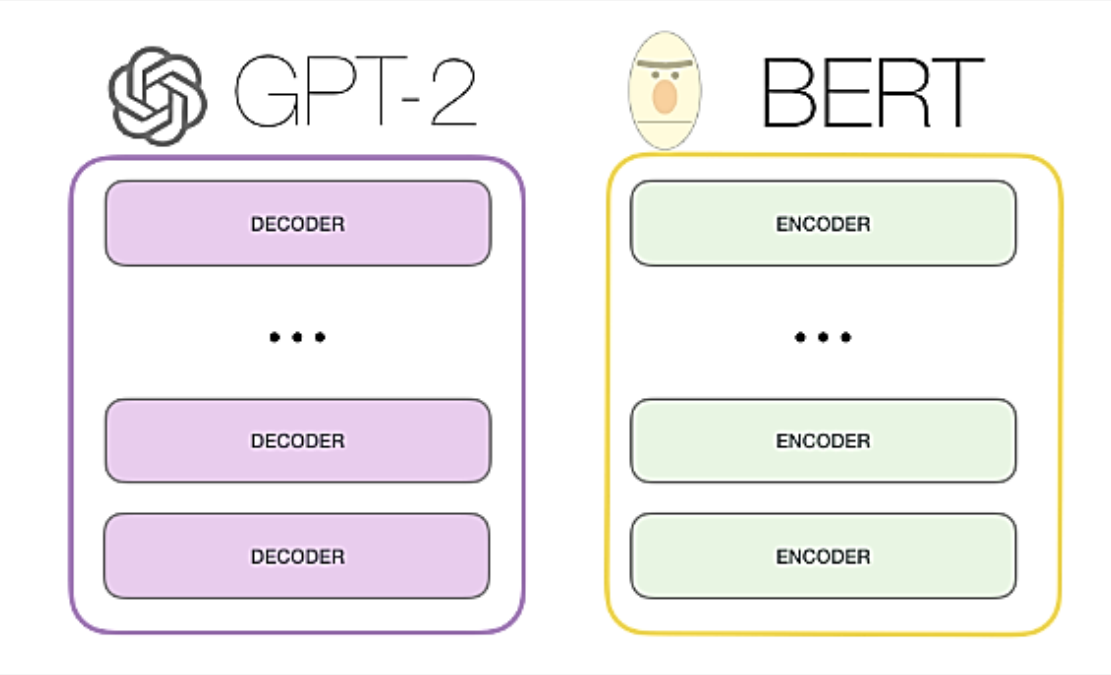


(Vaswani *et al.*, 2017)

Transformer Networks Visualization

https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

GPT و BERT



- یک ترنسفورمر از پشته انکودر برای مدل سازی ورودی استفاده می کند و از پشته دیکودر برای مدل سازی خروجی (با استفاده از اطلاعات ورودی از سمت رمزگذار)
- اگر ورودی نداریم، فقط می‌خواهیم «کلمه بعدی» را مدل کنیم، می‌توانیم سمت رمزگذار یک ترنسفورمر را حذف کنیم و «کلمه بعدی» را یکی یکی خروجی بدهیم. این به ما GPT می دهد.
- اگر ما فقط علاقه مند به آموزش یک مدل زبانی برای ورودی برای برخی از تسک های دیگر هستیم، به رمزگشای ترنسفورمر نیاز نداریم که به ما BERT می دهد.