扬州大学
YANGZHOU UNIVERSITY

自 然 语 言 处 理

Natural language Processing

扬州大学信息工程学院（人工智能学院）

第7章

Transformer

01 Transformer

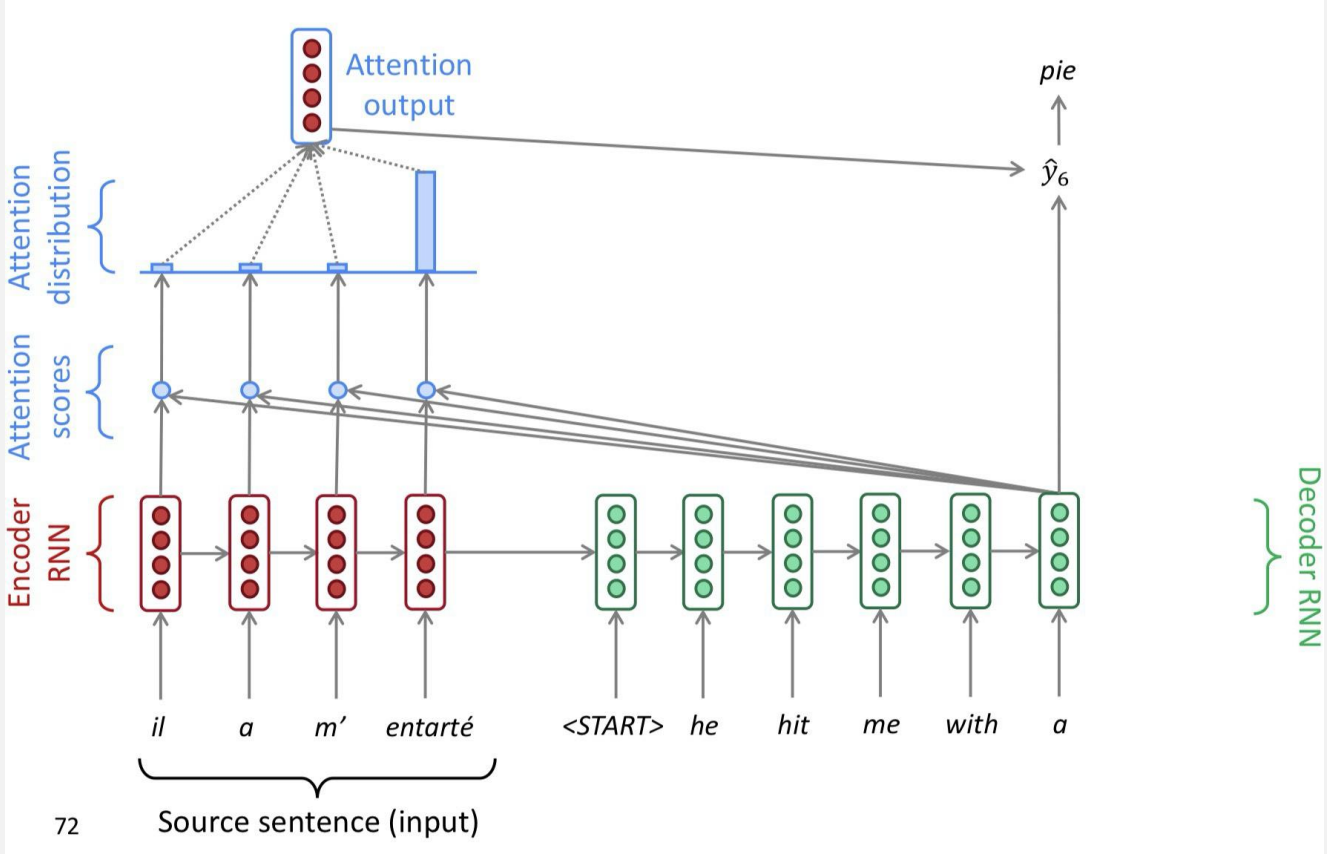
02 自注意力机制

03 预训练语言模型

04 预训练与微调

05 ELMO、GPT、BERT

回顾：Attention



编码器隐藏表示：h^{enc}_1 ，．．．．，h^{enc}_n

解码器隐藏表示：h^{dec}_1 ，．．．．，h^{dec}_n

h^{dec}_t表示在t时间段的解码器隐藏表示

注意力分数计算

$$e^t = [g(h_1^{enc}, h_t^{dec}), \ldots, g(h_n^{enc}, h_t^{dec})] \in \mathbb{R}^n$$

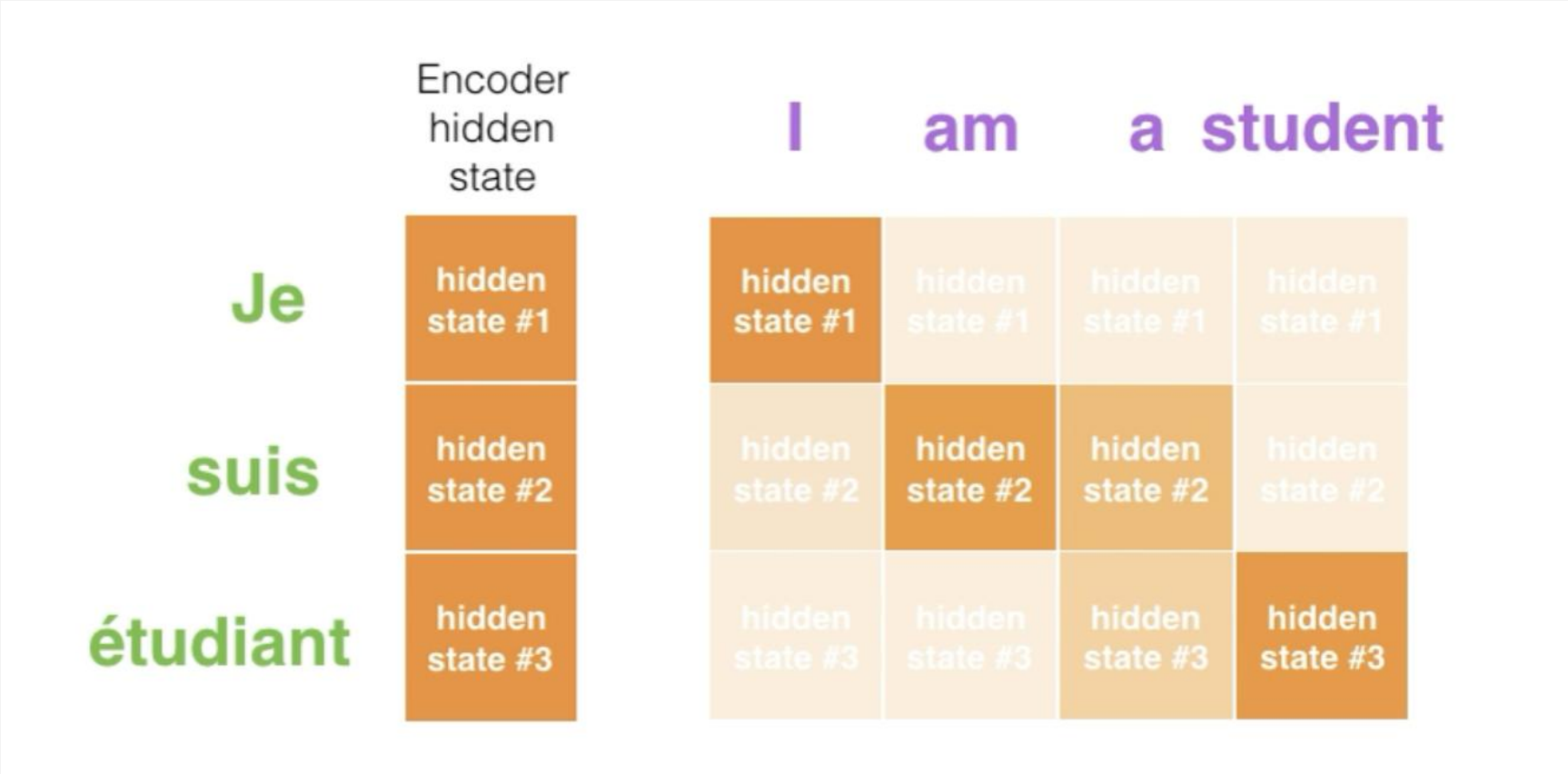
$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^n$$

注意力分布计算

$$a_t = \sum_{i=1}^{n} \alpha_i^t h_i^{enc} \in \mathbb{R}^h$$

t时 a_t 的 h_t^{dec} 注力表示

回顾：Attention



注意力机制解决了“瓶颈”或固定表示问题

- 注意力机制学习了信息对齐的概念

  即，“哪些源词与当前目标词更相关？”

Part.01

Transformer

Transformer（Vaswani et al., 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

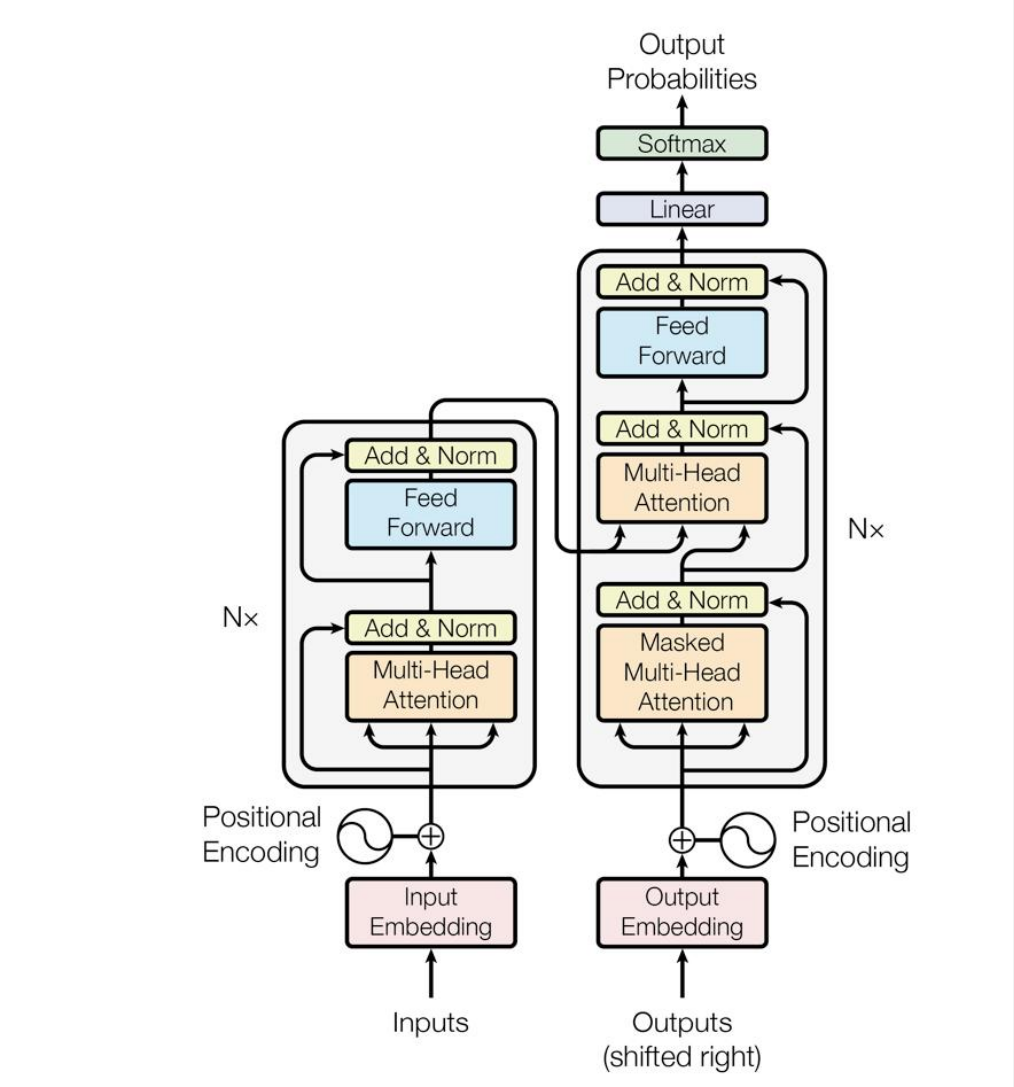
Llion Jones* Google Research llion@google.com

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

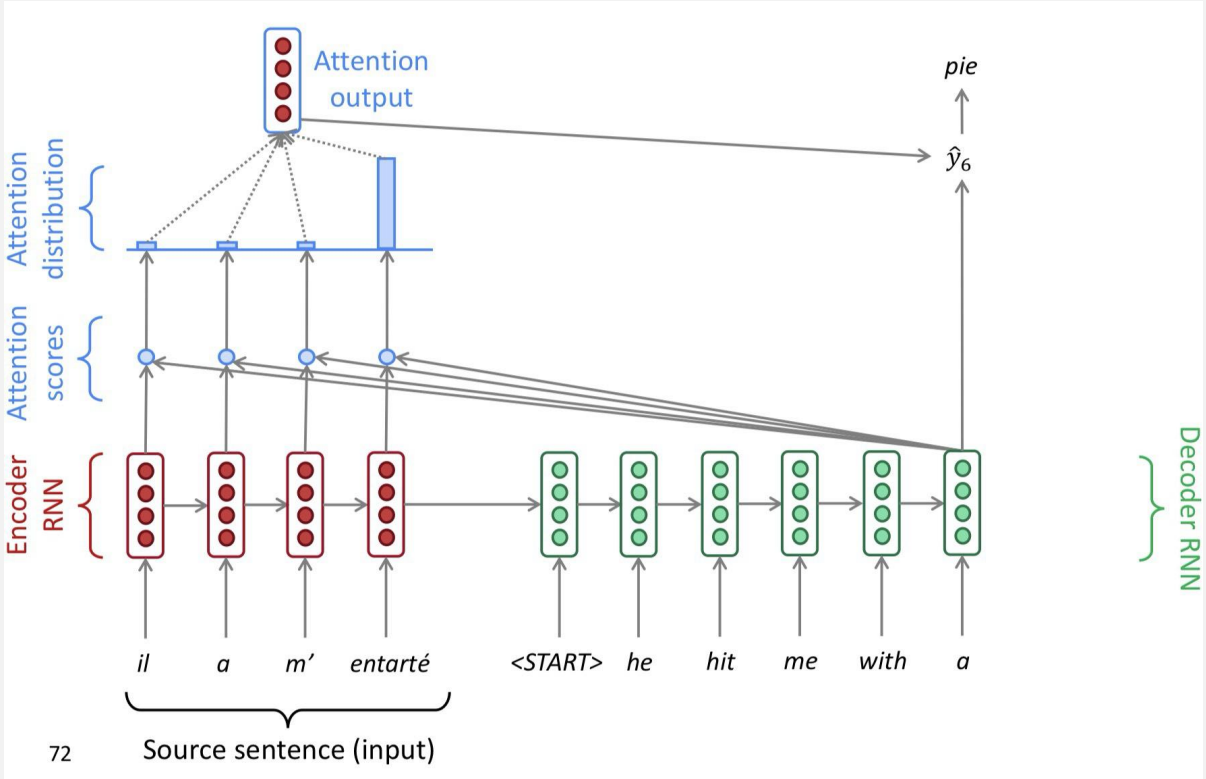
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

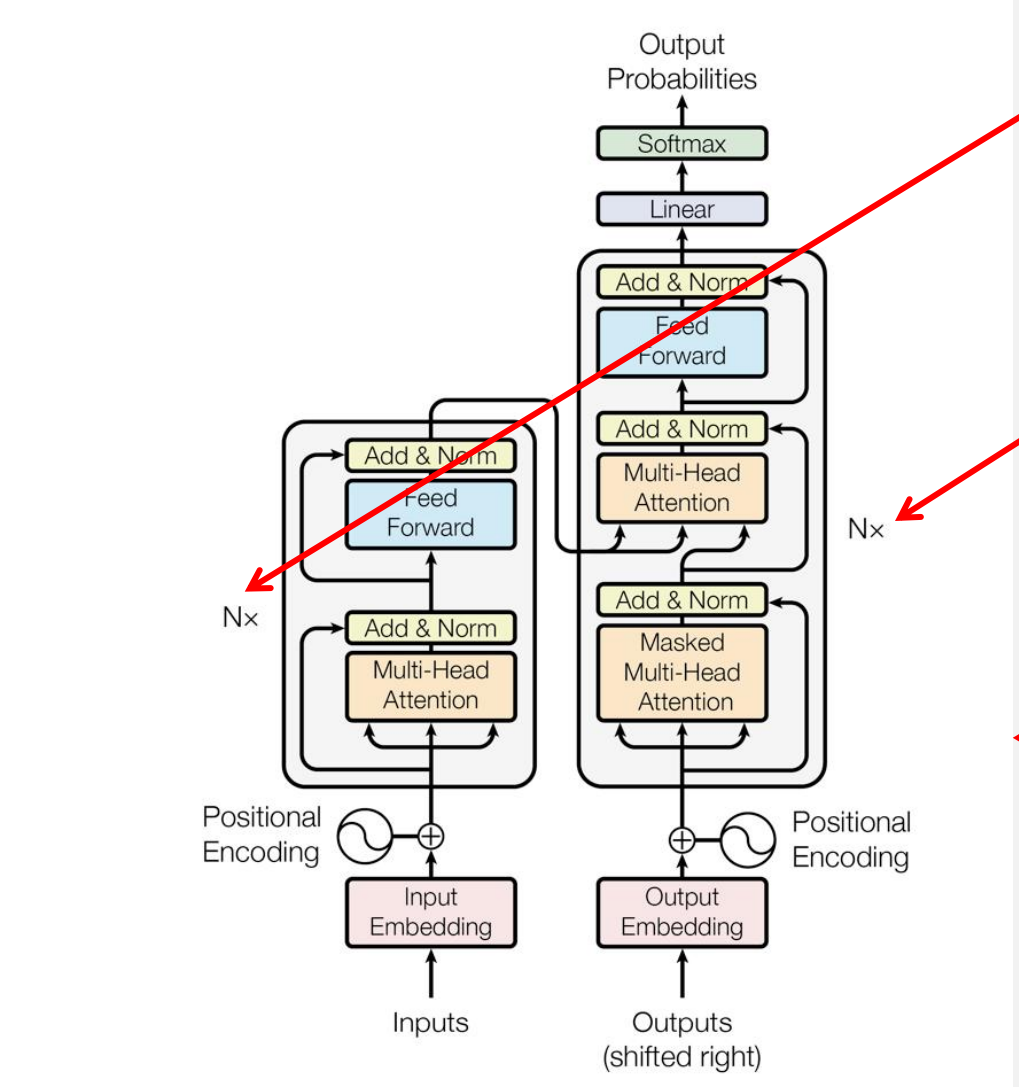
Transformer 编码器解码器



类似于基于RNN+attention的编解码器



Transformer 编码器解码器



Transformer编码器Nx 类似堆叠RNN编码器

例如：BERT，RoBERTa

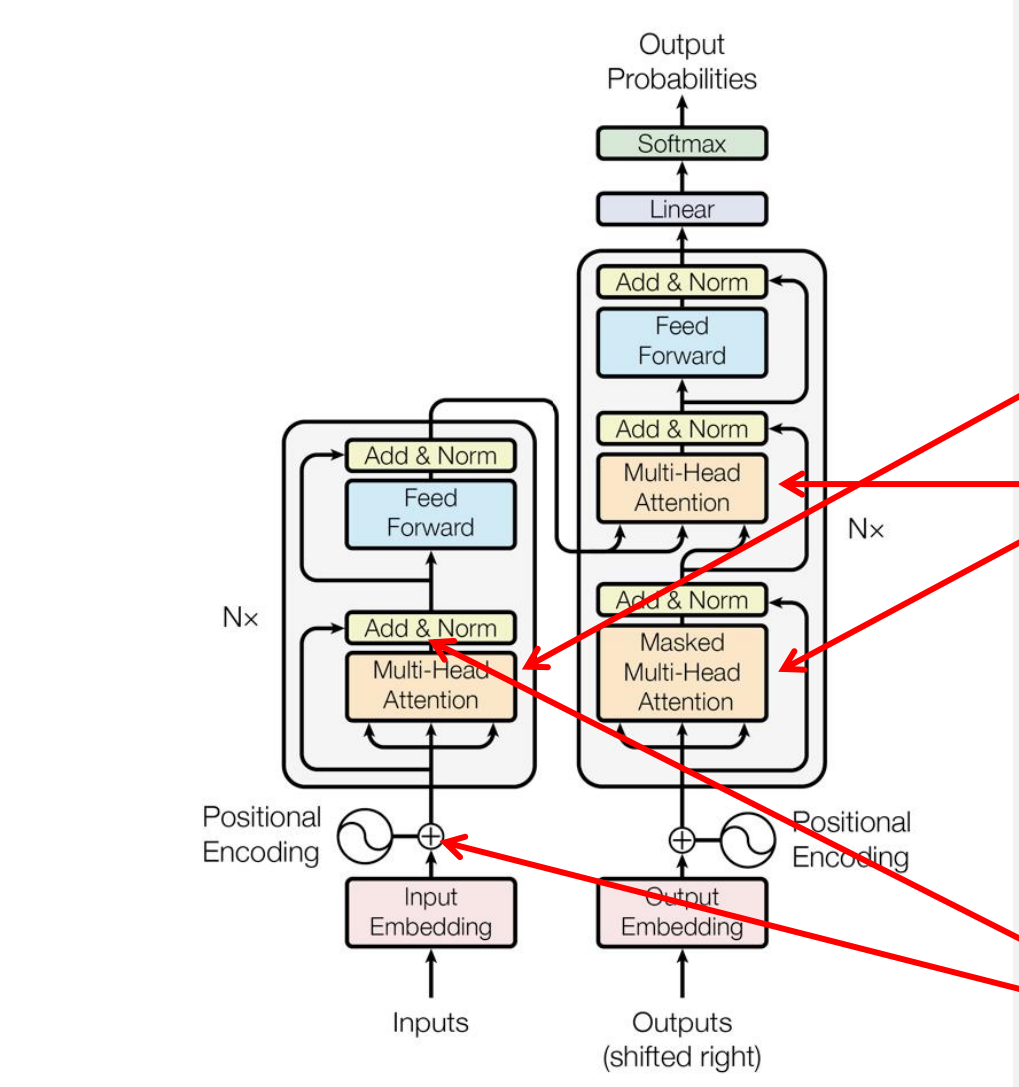
Transformer解码器Nx 类似堆叠RNN解码器

例如：GPT系列，Palm

Transformer编解码器

例如：T5，BART

Transformer 编码器解码器



关键创新点：

1. 自注意力机制；

2. (掩码)多头注意力；

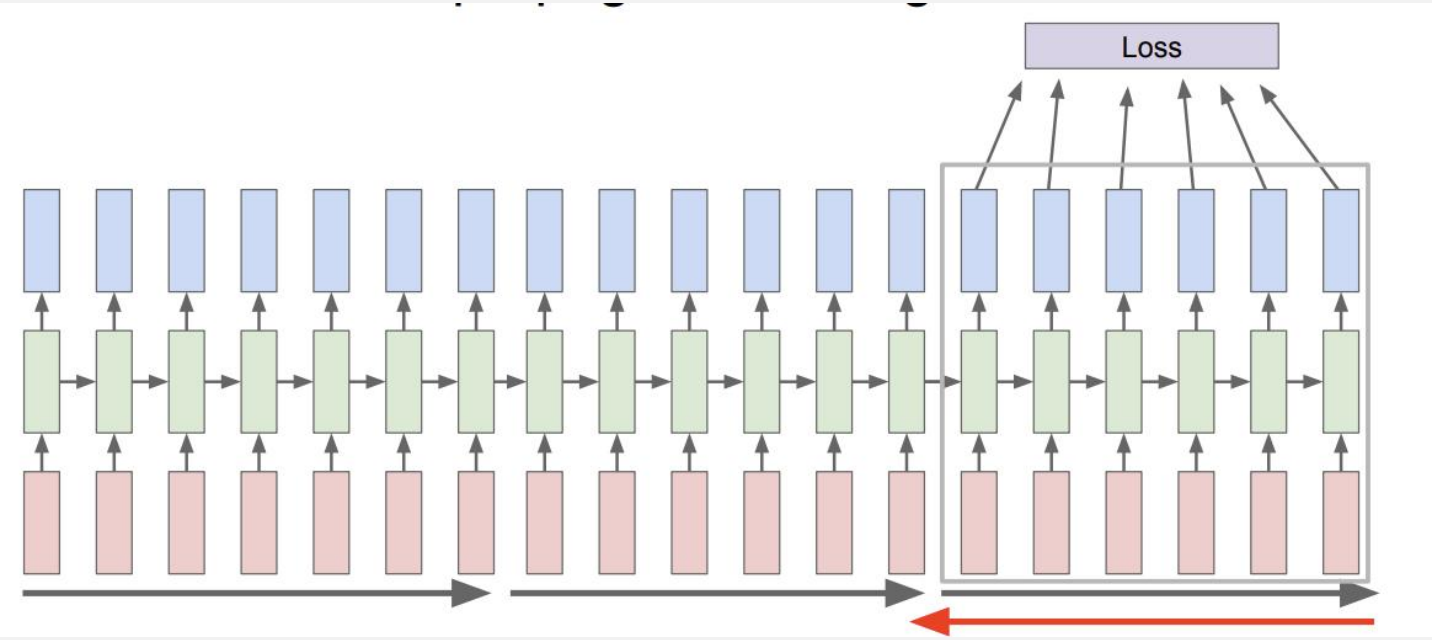
3. Transformer没有RNN类似的循环结构

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \in \mathbb{R}^h$$

4. 结合位置编码+残差链接

为什么不用RNN，其有何缺点？

对长序列编码会导致梯度消失，从而很难捕捉需要长距离依赖的信息



RNN的循环结构不支持并行化处理

- 无法使用并行的高性能计算集群系统

注意力机制

- Assume that we have a set of values $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathbb{R}^{d_v}$ and a query vector $\mathbf{q} \in \mathbb{R}^{d_q}$

- Attention always involves the following steps:
  - Computing the **attention scores** $\mathbf{e} = g(\mathbf{q}, \mathbf{v}_i) \in \mathbb{R}^n$
  - Taking softmax to get **attention distribution** α:

$$\alpha = \mathrm{softmax}(\mathbf{e}) \in \mathbb{R}^n$$

  - Using attention distribution to take **weighted sum** of values:

$$\mathbf{a} = \sum_{i=1}^{n} \alpha_i \mathbf{v}_i \in \mathbb{R}^{d_v}$$

注意力机制

- A more general form: use a set of keys and values $(\mathbf{k}_1, \mathbf{v}_1), \ldots, (\mathbf{k}_n, \mathbf{v}_n), \mathbf{k}_i \in \mathbb{R}^{d_k}, \mathbf{v}_i \in \mathbb{R}^{d_v}$, keys are used to compute the attention scores and values are used to compute the output vector

- Attention always involves the following steps:
  - Computing the **attention scores** $\mathbf{e} = g(\mathbf{q}, \mathbf{k}_i) \in \mathbb{R}^n$
  - Taking softmax to get **attention distribution** α:

$$\alpha = \mathrm{softmax}(\mathbf{e}) \in \mathbb{R}^n$$

  - Using attention distribution to take **weighted sum** of values:

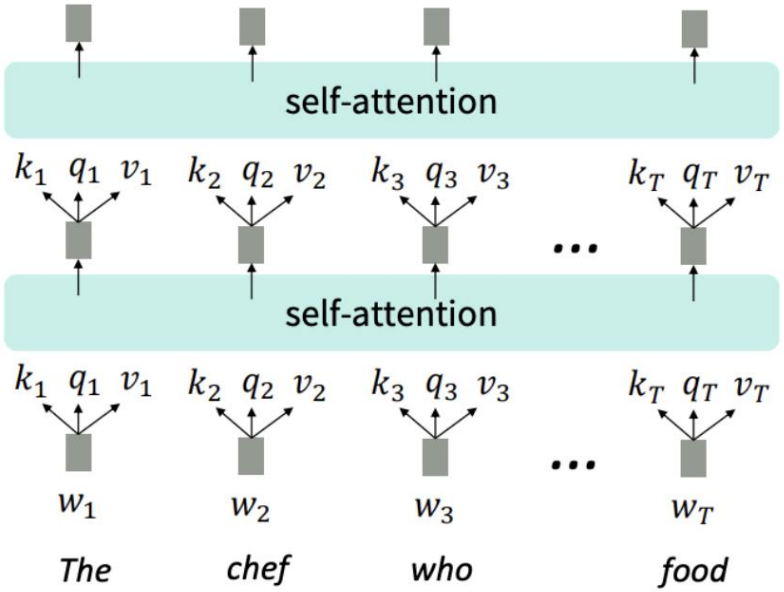
$$\mathbf{a} = \sum_{i=1}^{n} \alpha_i \mathbf{v}_i \in \mathbb{R}^{d_v}$$

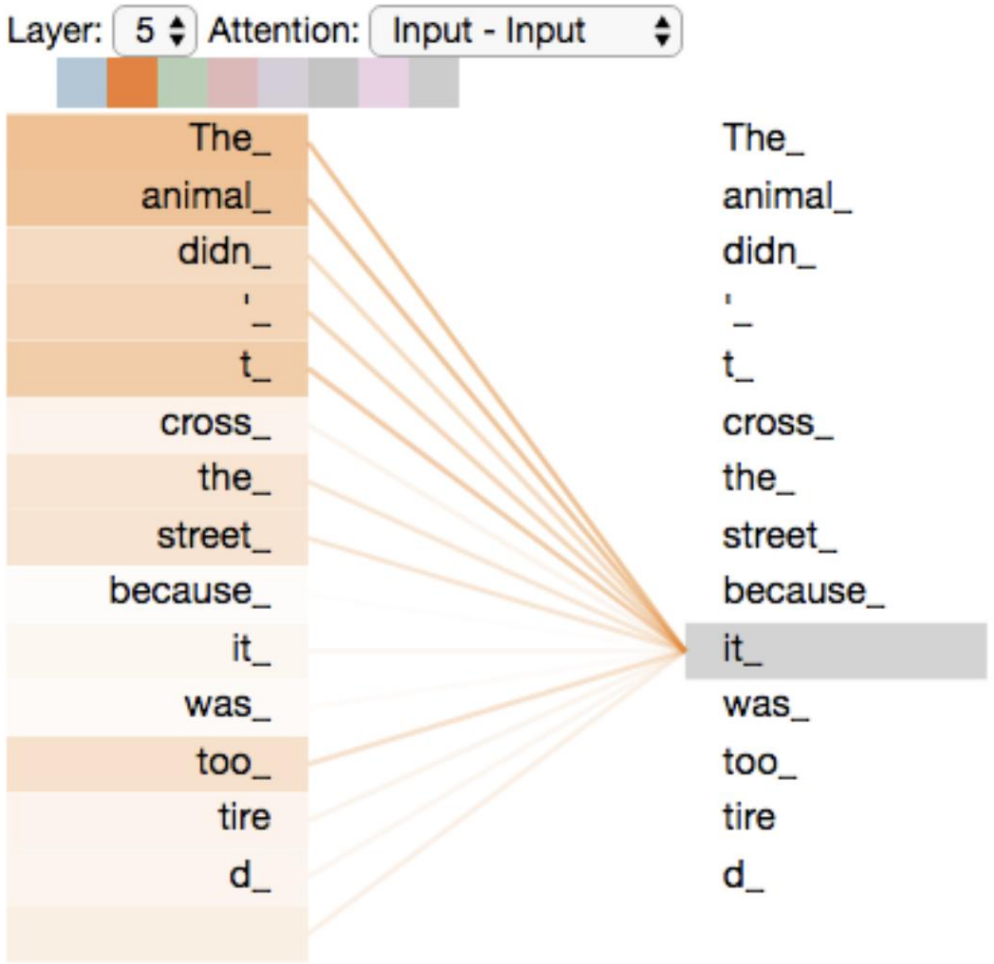
Part.02

自注意力机制

自注意力机制

- In NMT, query = decoder hidden state, keys = values = encoder hidden states
- Self-attention = attention from the sequence to **itself**
- Self-attention: let's use each word in a sequence as the query, and all the other words in the sequence as keys and values.





https://jalammar.github.io/illustrated-transformer/

自注意力机制

A self-attention layer maps a sequence of input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^{d_1}$ to a sequence of *n* vectors: $\mathbf{h}_1, \ldots, \mathbf{h}_n \in \mathbb{R}^{d_2}$

- The same abstraction as RNNs - used as a drop-in replacement for an RNN layer

$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

Self-attention:

$$\mathbf{q}_i = \mathbf{W}^{(q)}\mathbf{x}_i, \quad \mathbf{k}_i = \mathbf{W}^{(k)}\mathbf{x}_i, \quad \mathbf{v}_i = \mathbf{W}^{(v)}\mathbf{x}_i,$$

$$\mathbf{h}_i = \mathbf{W}^{(o)} \sum_{j=1}^{n} \left(\frac{\exp(\mathbf{q}_i \cdot \mathbf{k}_j / \sqrt{d})}{\sum_{j'=1}^{n} \exp(\mathbf{q}_i \cdot \mathbf{k}_{j'} / \sqrt{d})} \mathbf{v}_j \right)$$

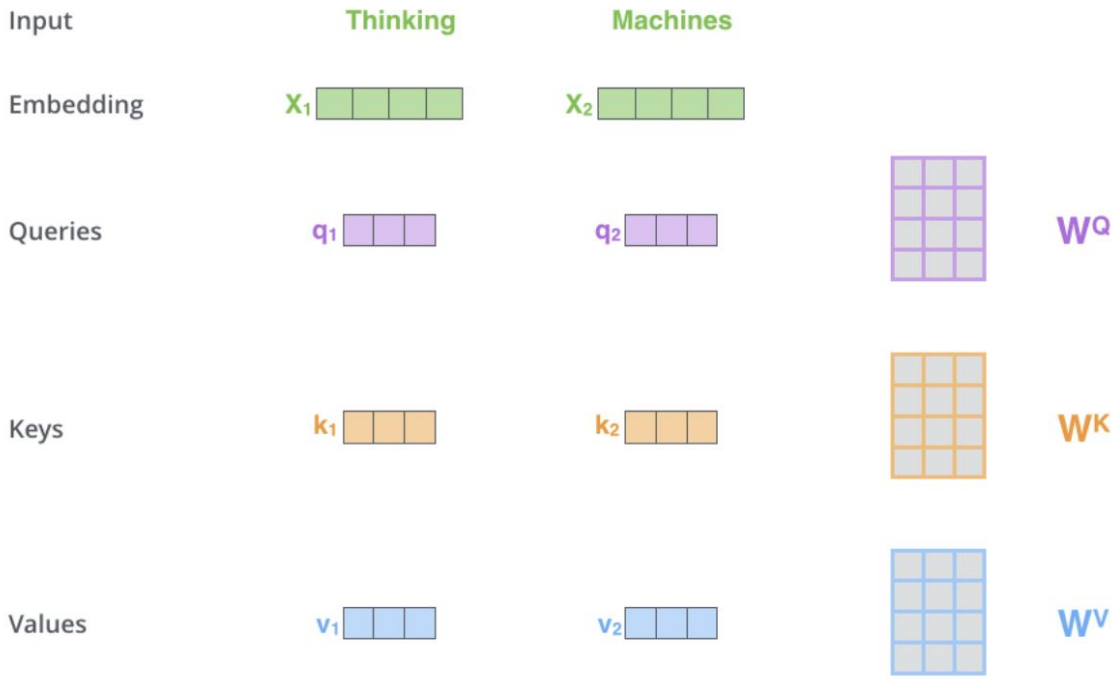
where $\mathbf{W}^{(q)}, \mathbf{W}^{(k)}, \mathbf{W}^{(v)}, \mathbf{W}^{(o)} \in \mathbb{R}^{d \times d}$.

自注意力机制

Step #1: Transform each input vector into three vectors: query, key, and value vectors

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q \in \mathbb{R}^{d_q} \qquad \mathbf{k}_i = \mathbf{x}_i \mathbf{W}^K \in \mathbb{R}^{d_k} \qquad \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^V \in \mathbb{R}^{d_v}$$

$$\mathbf{W}^Q \in \mathbb{R}^{d_1 \times d_q} \qquad \mathbf{W}^K \in \mathbb{R}^{d_1 \times d_k} \qquad \mathbf{W}^V \in \mathbb{R}^{d_1 \times d_v}$$



Note that we use row vectors here;
It is also common to write

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i \in \mathbb{R}^{d_q}$$

for $\mathbf{x}_i$ = a column vector

自注意力机制

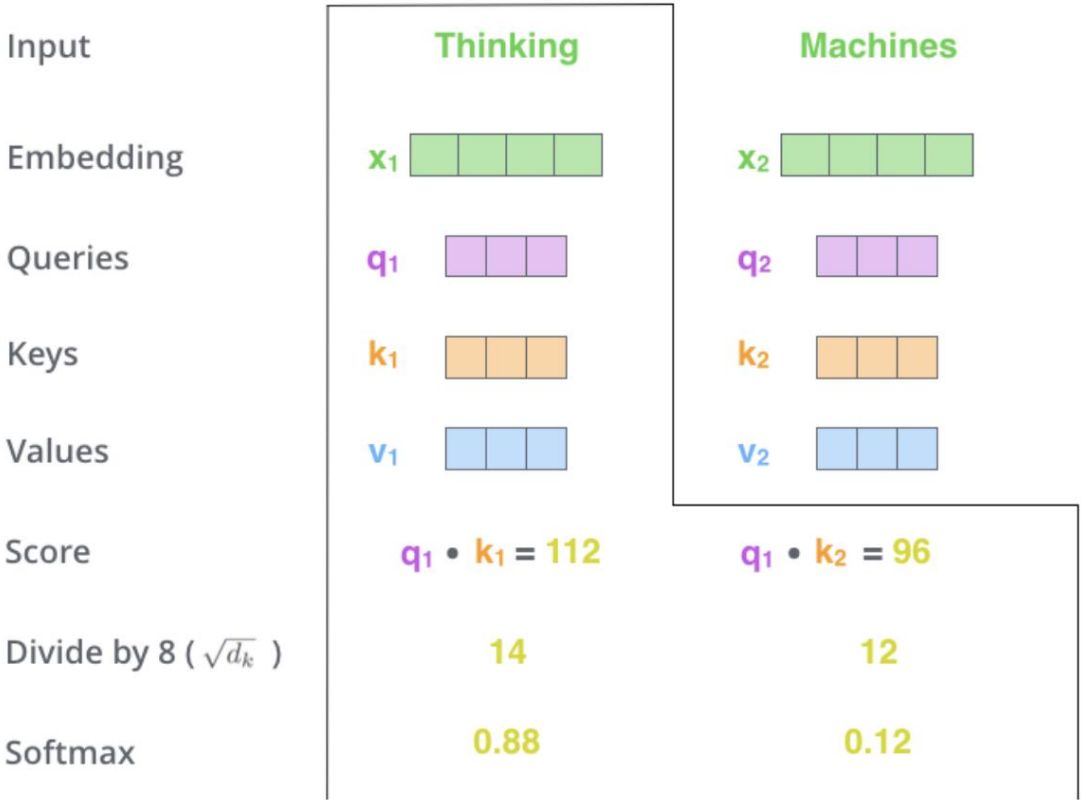
Step #2: Compute pairwise similarities between keys and queries; normalize with softmax

For each $\mathbf{q}_i$, compute attention scores and attention distribution:

$$\alpha_{i,j} = \text{softmax}(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}})$$

aka. "scaled dot product"

It must be $d_q = d_k$ in this case



Q. Why scaled dot product?

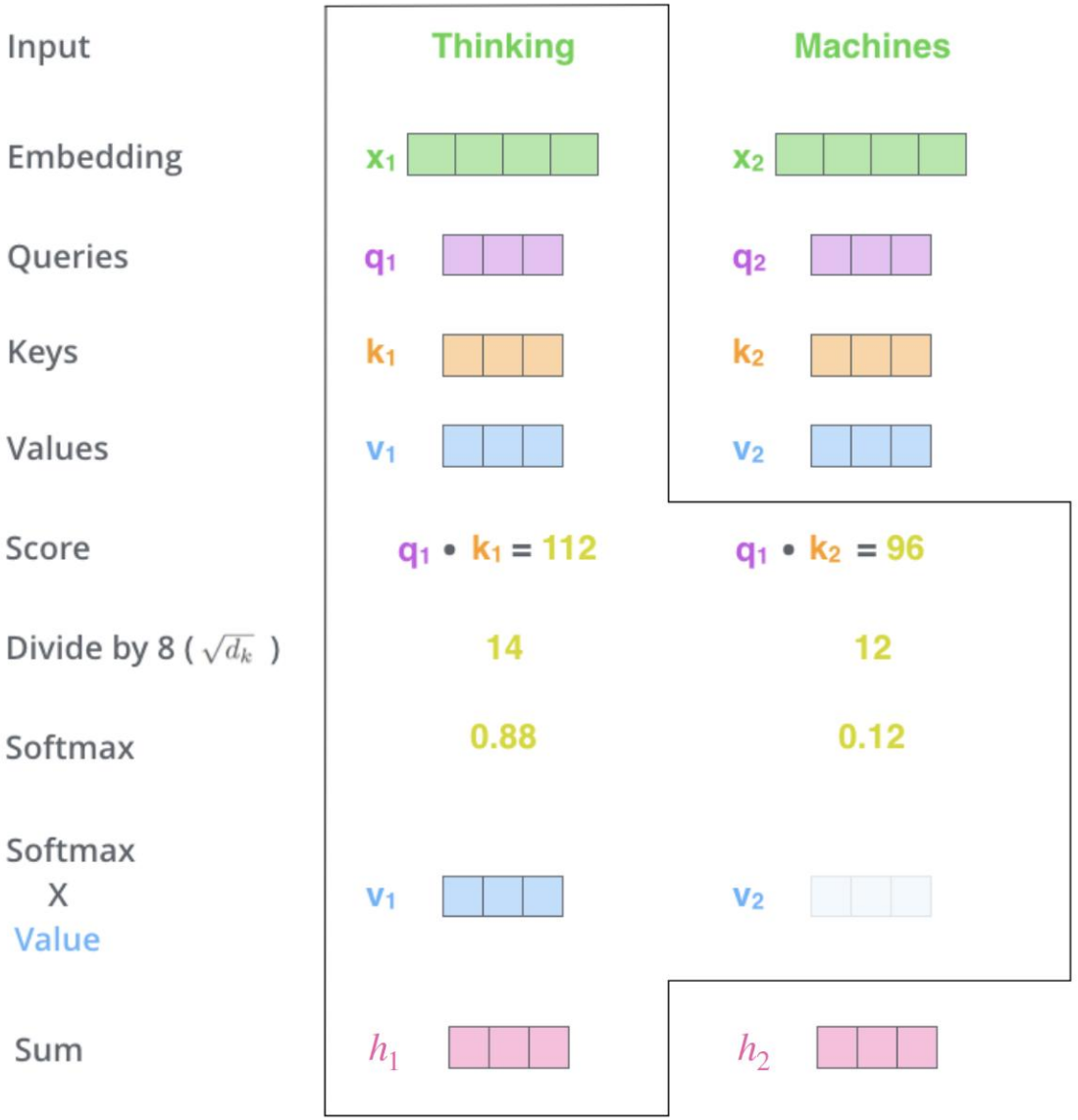
To avoid the dot product to become too large for larger d_k; scaling the dot product by $\frac{1}{\sqrt{d_k}}$ is easier for optimization

自注意力机制

Step #3: Compute output for each input as weighted sum of values

$$\mathbf{h}_i = \sum_{j=1}^{n} \alpha_{i,j} \mathbf{v}_j \in \mathbb{R}^{d_v}$$

$(d_v = d_2)$



自注意力机制

What would be the output vector for the word “Thinking” approximately?

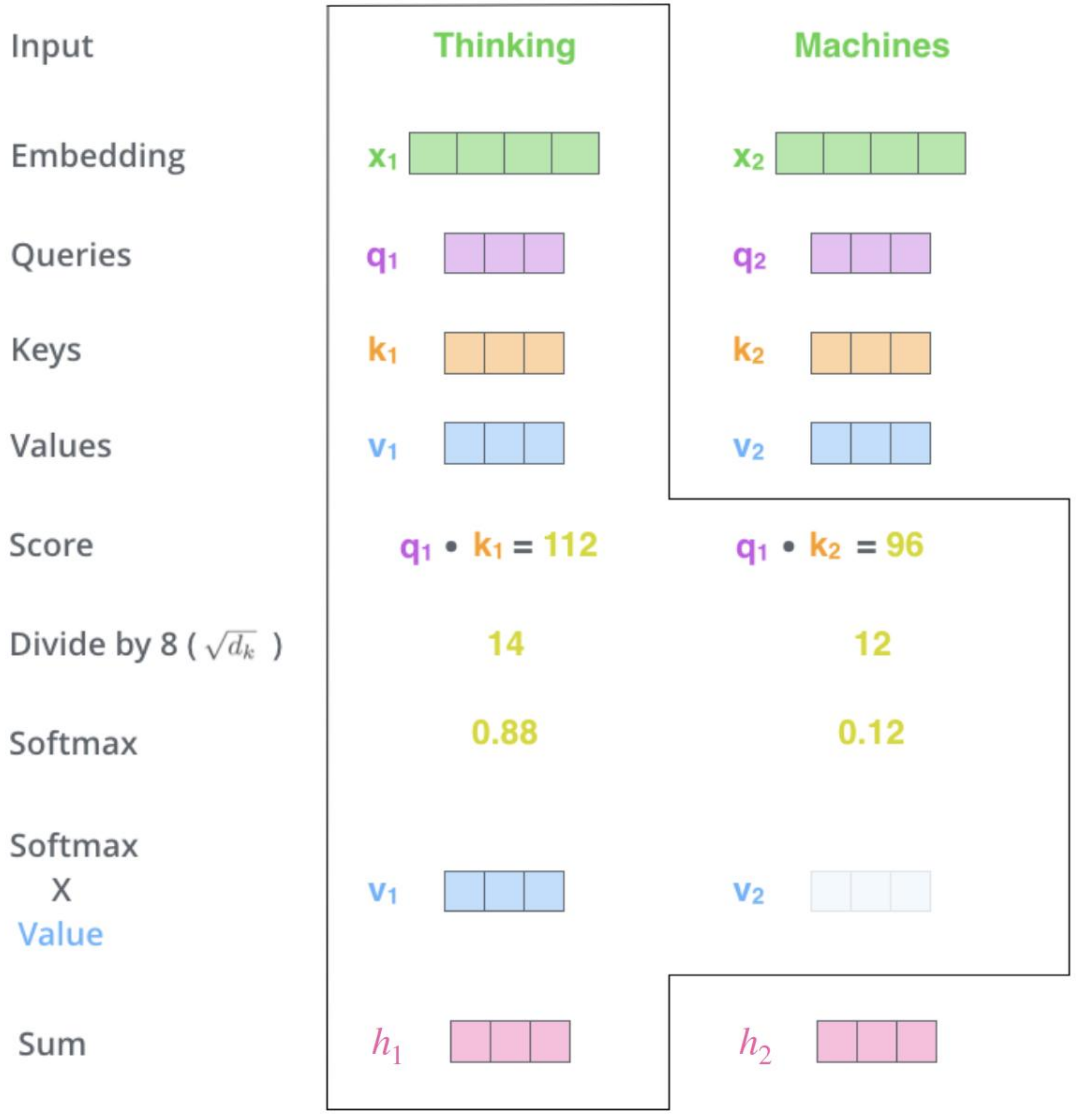
(a) $0.5\mathbf{v}_1 + 0.5\mathbf{v}_2$

(b) $0.54\mathbf{v}_1 + 0.46\mathbf{v}_2$

(c) $0.88\mathbf{v}_1 + 0.12\mathbf{v}_2$

(d) $0.12\mathbf{v}_1 + 0.88\mathbf{v}_2$

(c) is correct.



自注意力机制中的权重矩阵

$X \in \mathbb{R}^{n \times d_1}$ (n = input length)

$Q = XW^Q \quad K = XW^K \quad V = XW^V$

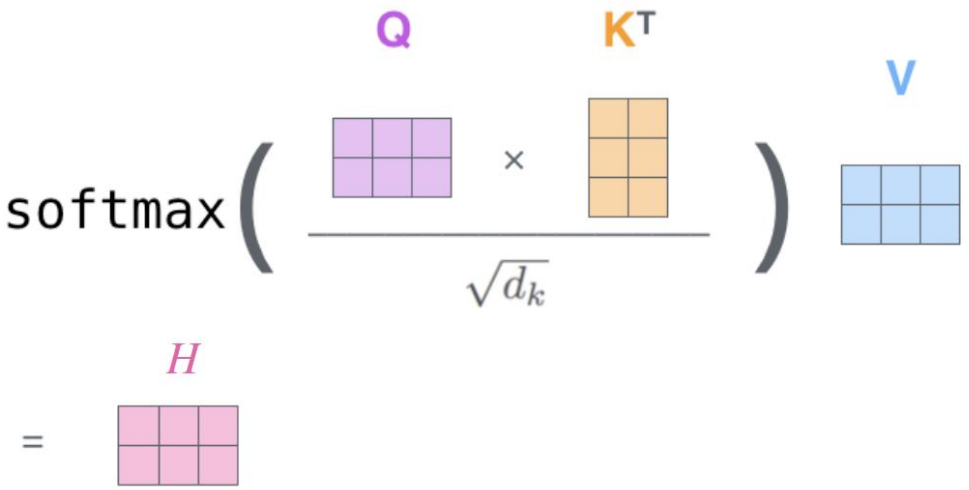
$W^Q \in \mathbb{R}^{d_1 \times d_q}, W^K \in \mathbb{R}^{d_1 \times d_k}, W^V \in \mathbb{R}^{d_1 \times d_v}$

$n \times d_q$ $\quad$ $d_k \times n$

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

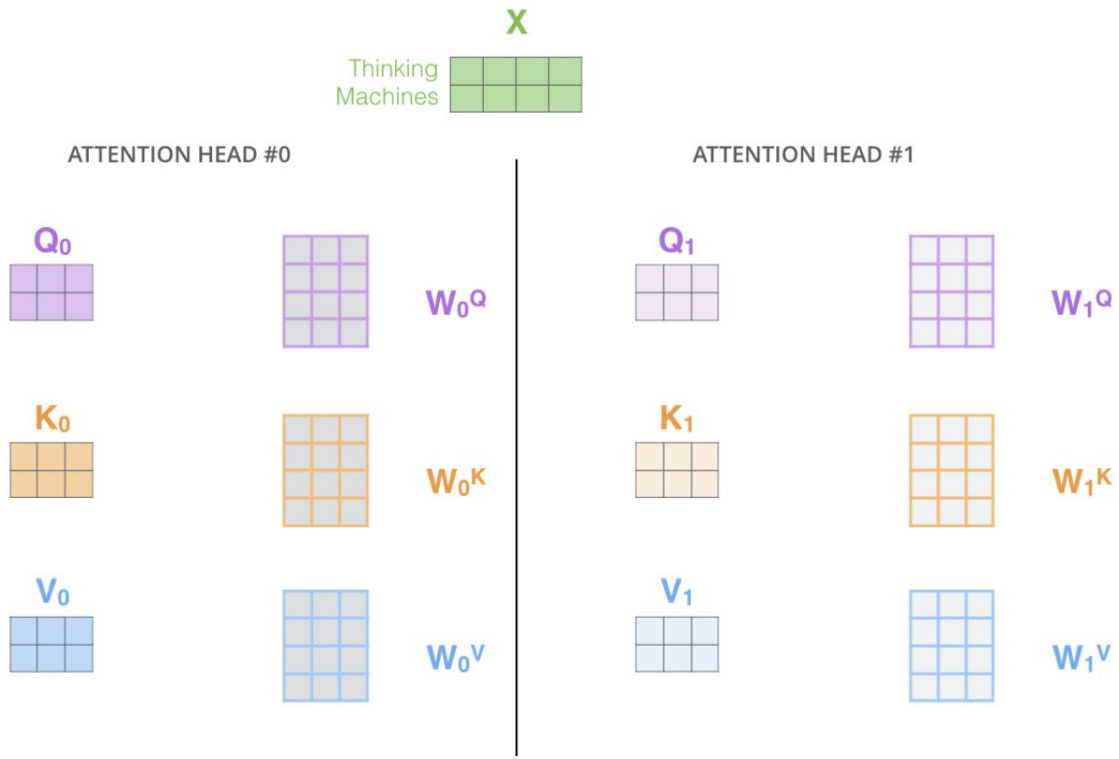
$n \times$

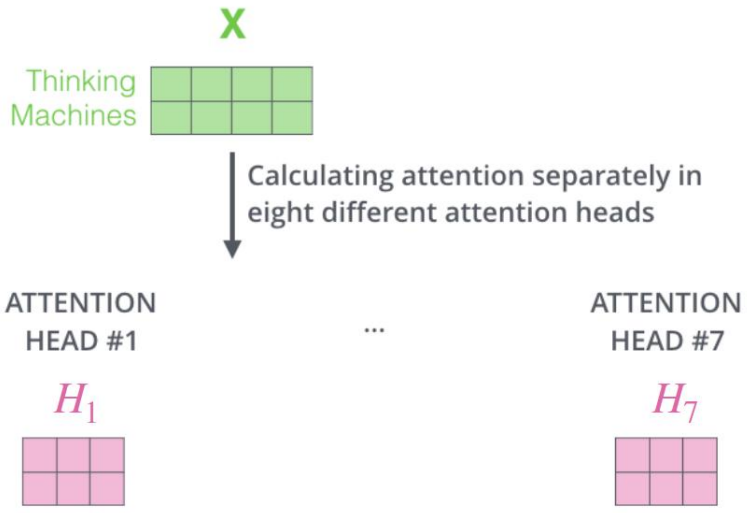
Q: What is this softmax operation?



多头注意力

- It is better to use multiple attention functions instead of one!
  - Each attention function (“head”) can focus on different positions.

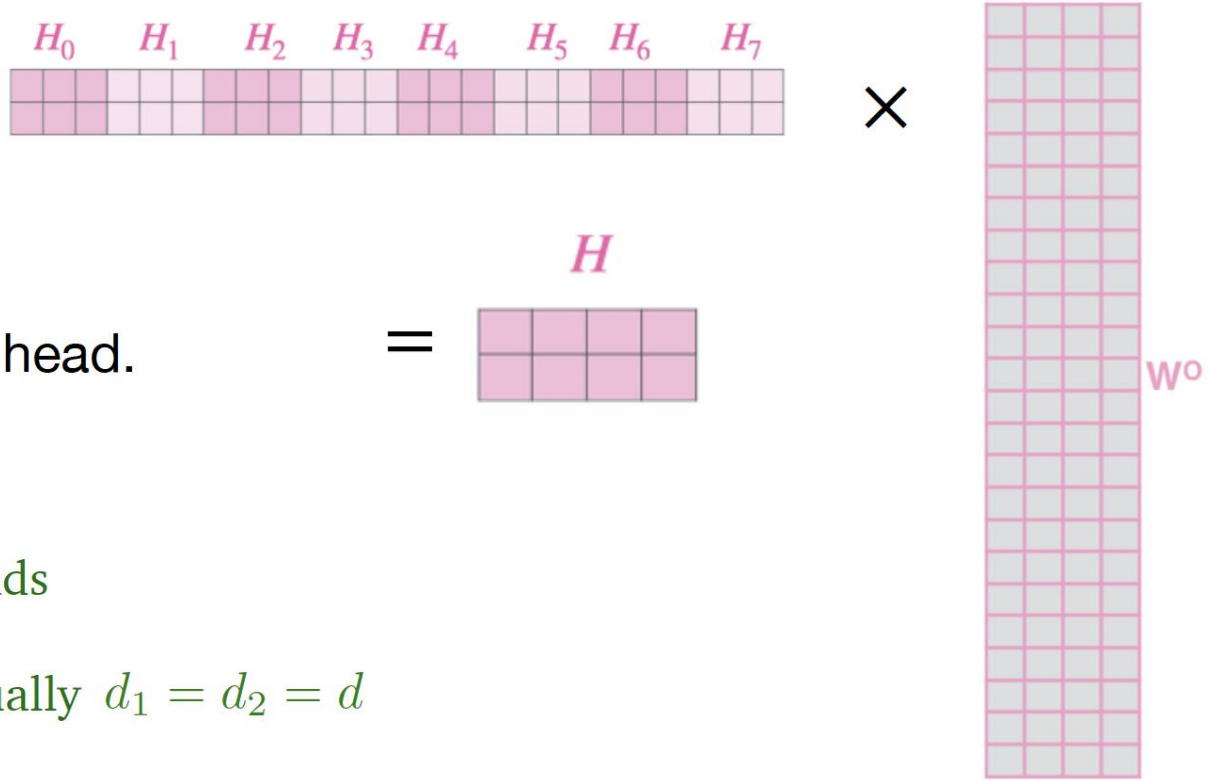




多头注意力

Finally, we just concatenate all the heads and apply an output projection matrix.

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head}_1, ..., \mathrm{head}_\mathrm{h})W^O$$
$$\mathrm{head}_i = \mathrm{Attention}(XW_i^Q, XW_i^K, XW_i^V)$$



- In practice, we use a *reduced* dimension for each head.

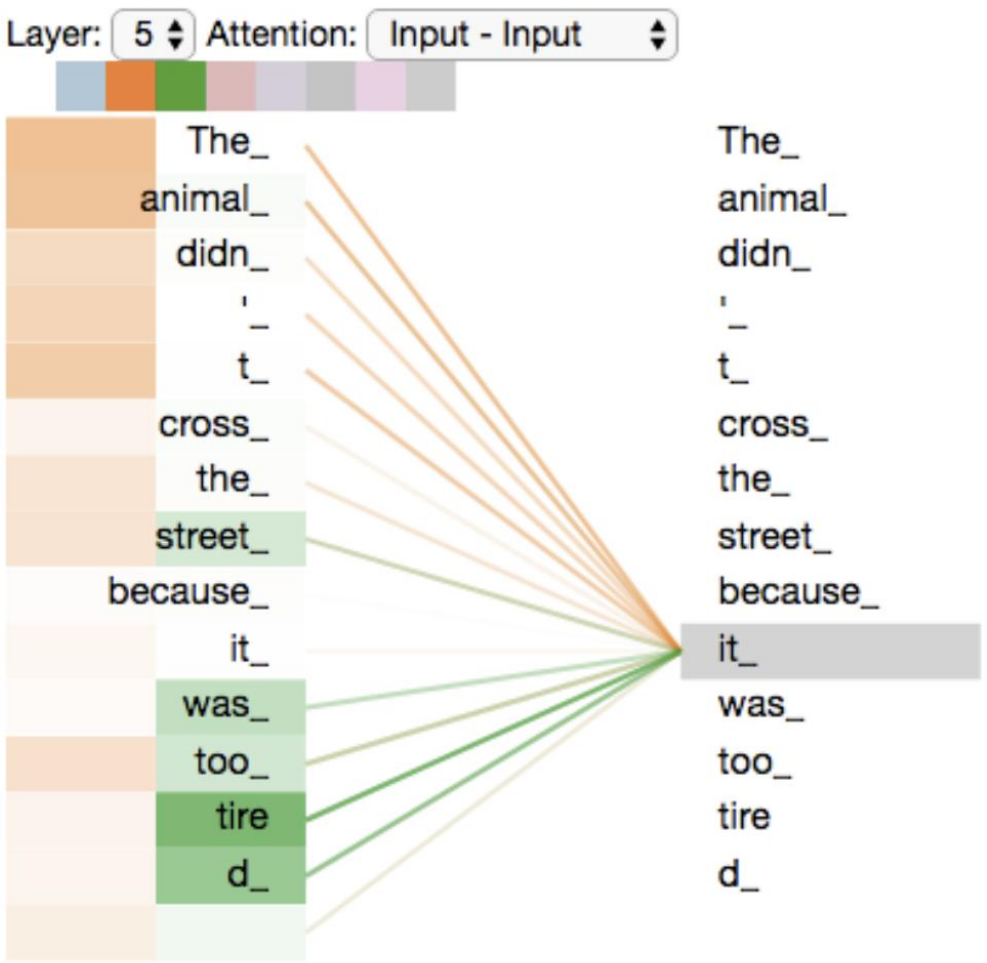
$$W_i^Q \in \mathbb{R}^{d_1 \times d_q}, W_i^K \in \mathbb{R}^{d_1 \times d_k}, W_i^V \in \mathbb{R}^{d_1 \times d_v}$$

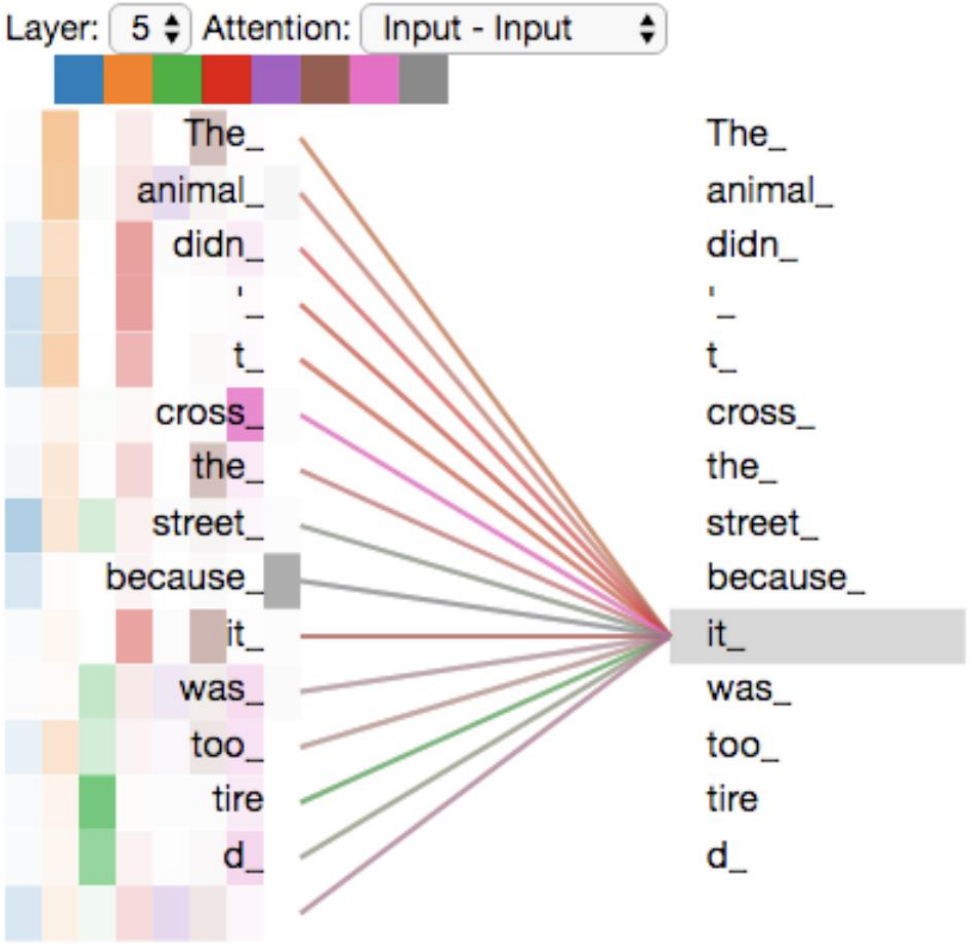
$$d_q = d_k = d_v = d/m \quad d = \text{hidden size}, m = \# \text{ of heads}$$

$$W^O \in \mathbb{R}^{d \times d_2} \quad \text{If we stack multiple layers, usually } d_1 = d_2 = d$$

- The total computational cost is similar to that of single-head attention with full dimensionality.

多头注意力


Layer: 5
Attention: Input - Input
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_


Layer: 5
Attention: Input - Input
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_

Transformer还缺位置信息

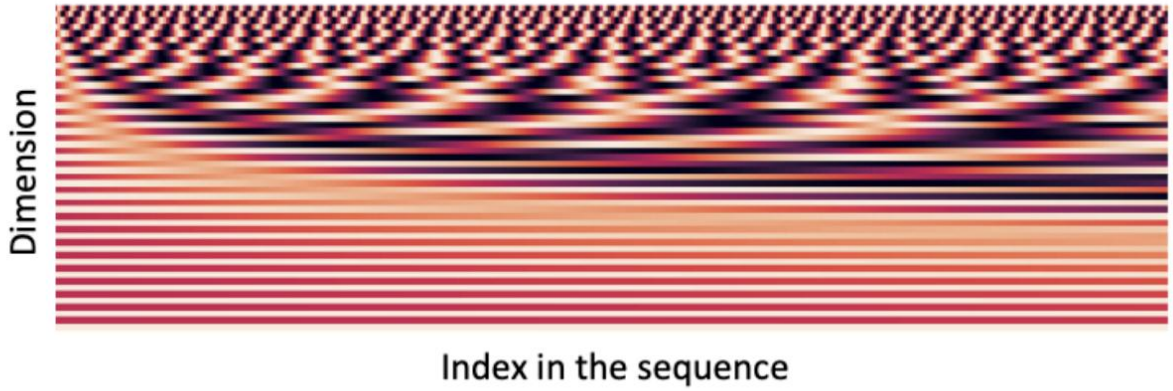
- Unlike RNNs, self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values

- Solution: Add **"positional encoding"** to the input embeddings: $\mathbf{p}_i \in \mathbb{R}^d$ for $i = 1,2,\ldots,n$

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{p}_i$$

- **Sinusoidal position encoding**: sine and cosine functions of different frequencies:

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



  - **Pros**: Periodicity + can extrapolate to longer sequences
  - **Cons**: Not learnable

位置编码

- **Learned absolute position encoding:** let all $\mathbf{p}_i$ be learnable parameters
  - $P \in \mathbb{R}^{d \times L}$ for L = max sequence length

  - **Pros**: each position gets to be learned to fit the data
  - **Cons**: can't extrapolate to indices outside of max sequence length *L*
  - Most systems use this!

Self-Attention with Relative Position Representations

Peter Shaw
Google
petershaw@google.com

Jakob Uszkoreit
Google Brain
usz@google.com

Ashish Vaswani
Google Brain
avaswani@google.com

ROFORMER: ENHANCED TRANSFORMER WITH ROTARY POSITION EMBEDDING

Jianlin Su
Zhuiyi Technology Co., Ltd.
Shenzhen
bojonesu@wezhuiyi.com

Yu Lu
Zhuiyi Technology Co., Ltd.
Shenzhen
julianlu@wezhuiyi.com

Shengfeng Pan
Zhuiyi Technology Co., Ltd.
Shenzhen
nickpan@wezhuiyi.com

Ahmed Murtadha
Zhuiyi Technology Co., Ltd.
Shenzhen
mengjiayi@wezhuiyi.com

Bo Wen
Zhuiyi Technology Co., Ltd.
Shenzhen
brucewen@wezhuiyi.com

Yunfeng Liu
Zhuiyi Technology Co., Ltd.
Shenzhen
glenliu@wezhuiyi.com

Transformer中的激活函数

- There are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages value vectors

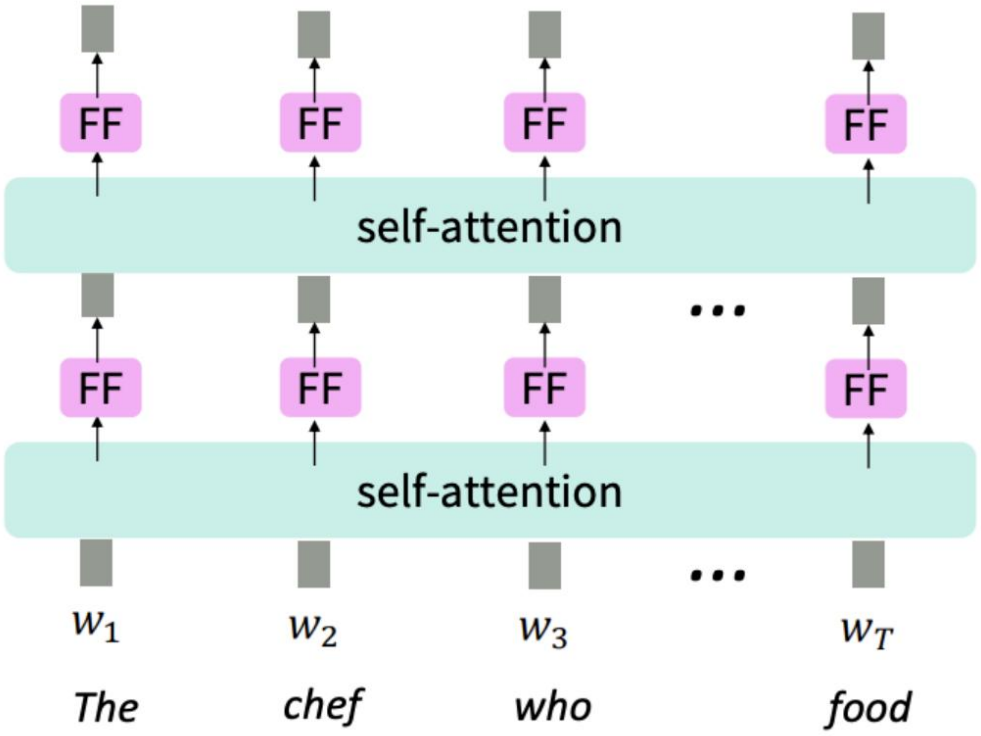
- Simple fix: add a feed-forward network to post-process each output vector

$$\mathrm{FFN}(\mathbf{x}_i) = \mathrm{ReLU}(\mathbf{x}_i \mathbf{W}_{1|} + \mathbf{b}_1)\mathbf{W_2} + \mathbf{b}_2$$

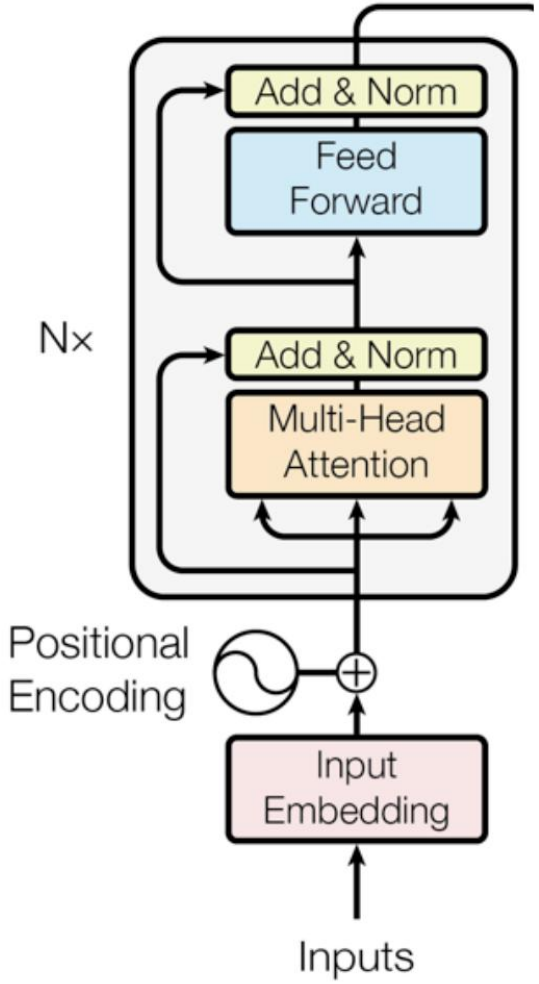
$$\mathbf{W}_1 \in \mathbb{R}^{d \times d_{ff}}, \mathbf{b}_1 \in \mathbb{R}^{d_{ff}}$$

$$\mathbf{W}_2 \in \mathbb{R}^{d_{ff} \times d}, \mathbf{b}_2 \in \mathbb{R}^{d}$$

In practice, they use $d_{ff} = 4d$



Transformer编码器



From the bottom to the top:

- Input embedding
- Positional encoding
- A stack of Transformer encoder layers

Transformer encoder is a stack of N layers, which consists of two sub-layers:

- Multi-head attention layer
- Feed-forward layer

$$\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^{d_1} \longrightarrow \mathbf{h}_1, \ldots, \mathbf{h}_n \in \mathbb{R}^{d_2}$$

残差链接和层归一化

Add & Norm: $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$

Residual connections (He et al., 2016)

Instead of $X^{(i)} = \mathrm{Layer}(X^{(i-1)})$ (i represents the layer)

We let $X^{(i)} = X^{(i-1)} + \mathrm{Layer}(X^{(i-1)})$, so we only need to learn "the residual" from the previous layer

Gradient through the residual connection is 1 - good for propagating information through layers

残差链接和层归一化

Layer normalization (Ba et al., 2016) helps train model faster

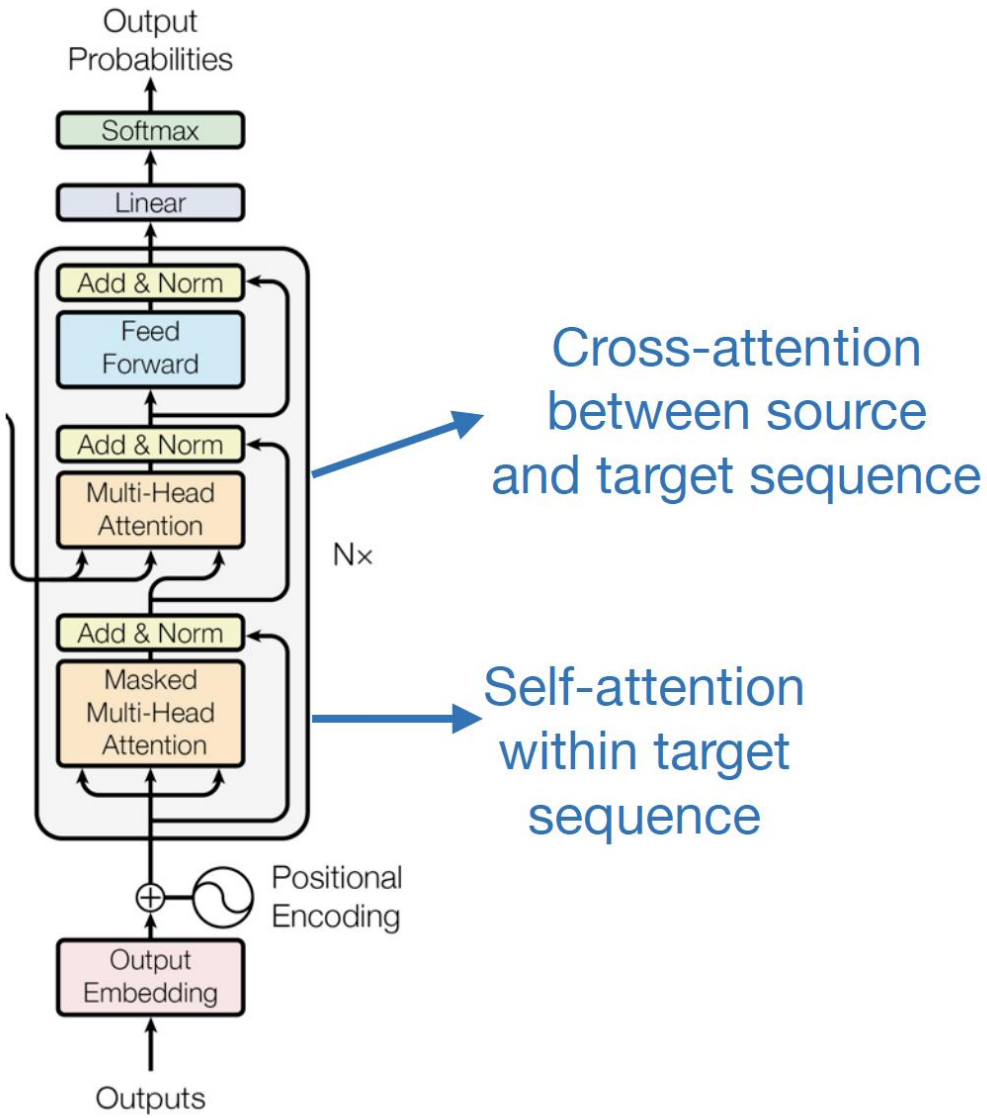
Idea: normalize the hidden vector values to unit mean and stand deviation within each layer

[advanced]

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

$\gamma, \beta \in \mathbb{R}^d$ are learnable parameters

Transformer解码器



From the bottom to the top:

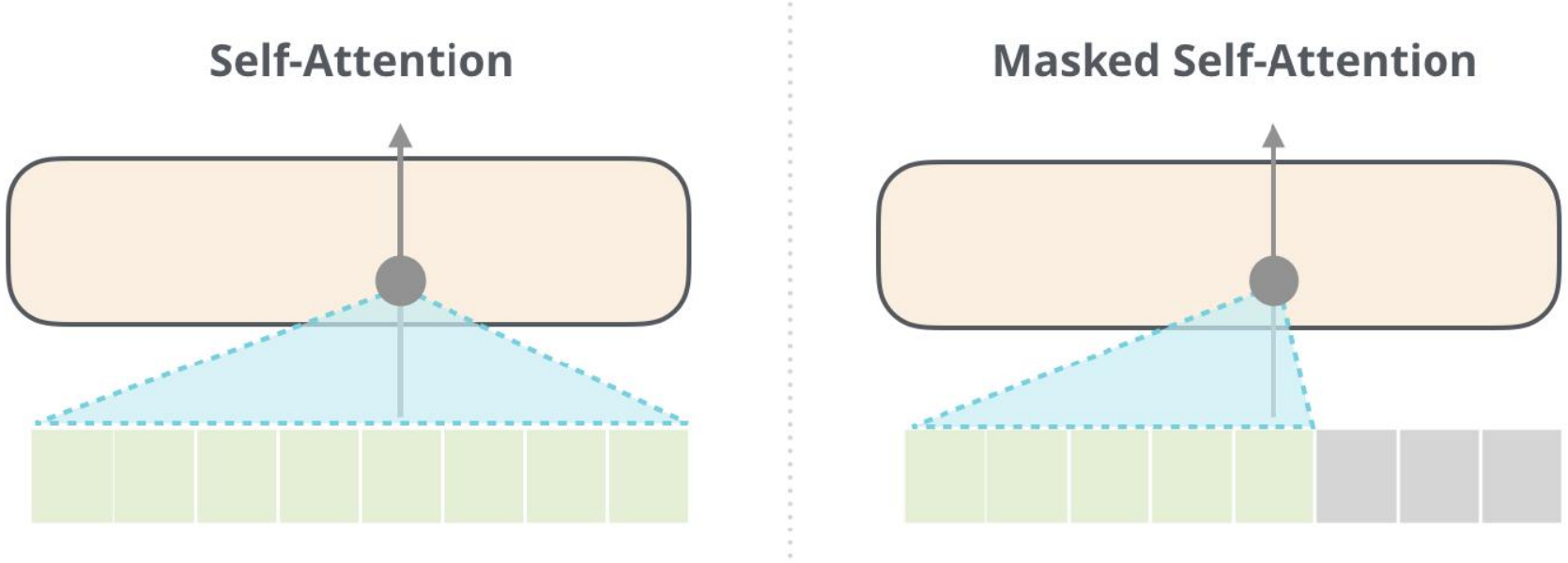
- Output embedding
- Positional encoding
- A stack of Transformer decoder layers
- Linear + softmax

Transformer decoder is a stack of N layers, which consists of three sub-layers:

- Masked multi-head attention
- Multi-head cross-attention
- Feed-forward layer
- (W/ Add & Norm between sub-layers)

掩码注意力机制

- Key: You can't see the future text for the decoder!



- Solution: for every $\mathbf{q}_i$, only attend to $\{(\mathbf{k}_j, \mathbf{v}_j)\}, j \leq i$

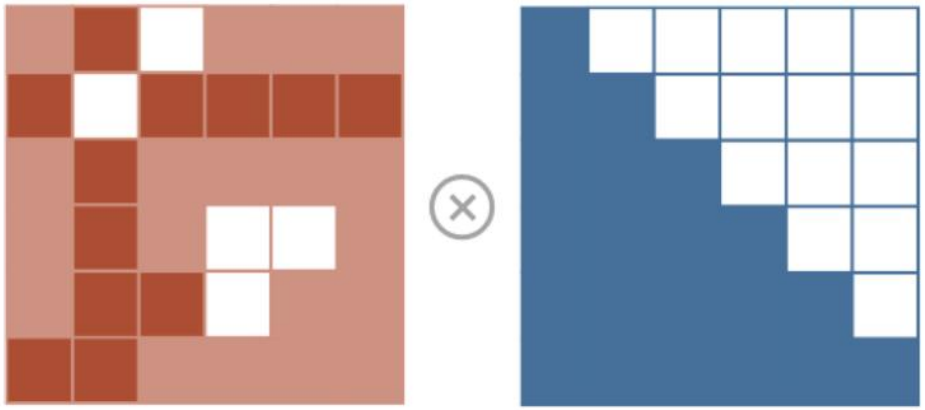
How to implement this? Masking!

掩码多头注意力

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q, \mathbf{k}_i = \mathbf{x}_i \mathbf{W}^K, \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^V$$

$$e_{i,j} = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}, \forall j = 1, \ldots, n$$

$$\alpha_i = \mathrm{softmax}(\mathbf{e}_i)$$

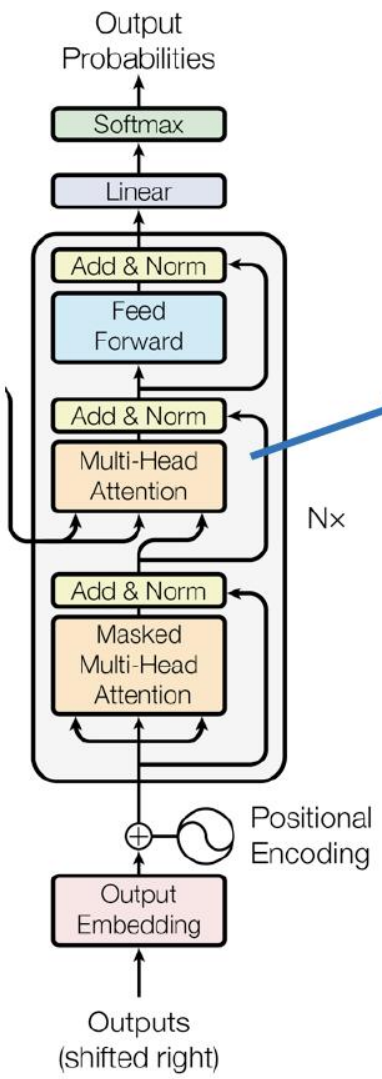


raw attention weights　　mask

Efficient implementation: compute attention as we normally do, mask out attention to future words by setting attention scores to $-\infty$

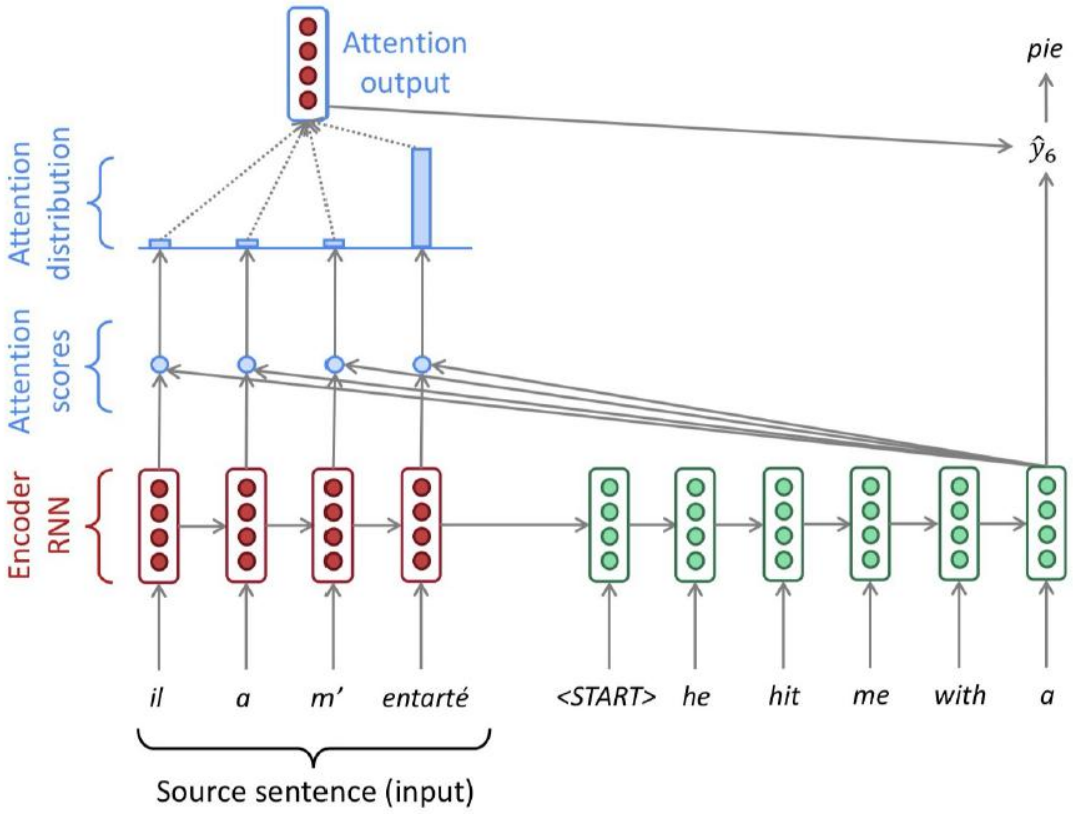
```
dot = torch.bmm(queries, keys.transpose(1, 2))

indices = torch.triu_indices(t, t, offset=1)
dot[:, indices[0], indices[1]] = float('-inf')

dot = F.softmax(dot, dim=2)
```

掩码交叉注意力



Cross-attention between source and target sequence

Similar as the attention we learned in the previous lecture



掩码交叉注意力

Self-attention:

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q, \mathbf{k}_i = \mathbf{x}_i \mathbf{W}^K, \mathbf{v}_i = \mathbf{x}_i \mathbf{W}^V$$

$$e_{i,j} = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}, \forall j = 1, \ldots, n$$

$$\alpha_i = \mathrm{softmax}(\mathbf{e}_i)$$

$$\mathbf{h}_i = \sum_{j=1}^{n} \alpha_{i,j} \mathbf{v}_j$$

Cross-attention:

(always from the top layer)

$\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_m$: hidden states from encoder

$\mathbf{x}_1, \ldots, \mathbf{x}_n$: hidden states from decoder

$$\mathbf{q}_i = \mathbf{x}_i \mathbf{W}^Q \quad i = 1,2,\ldots,n$$

$$\mathbf{k}_j = \tilde{\mathbf{x}}_j \mathbf{W}^K, \mathbf{v}_j = \tilde{\mathbf{x}}_j \mathbf{W}^V \quad \forall j = 1,2,\ldots,m$$

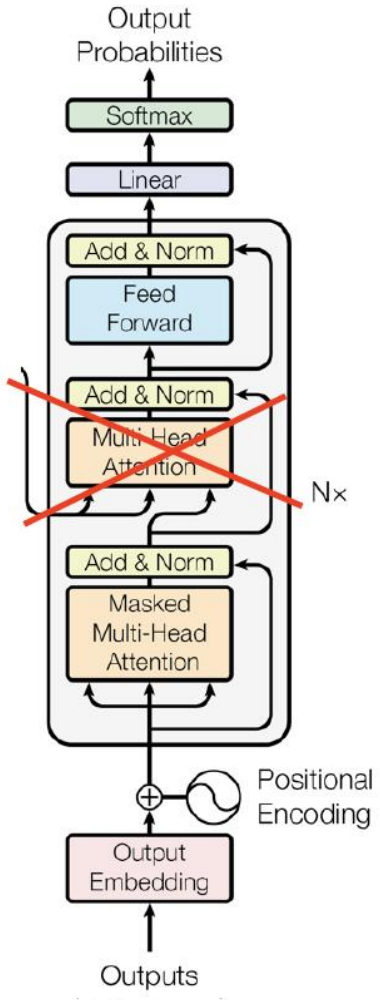
$$e_{i,j} = \frac{\mathbf{q_i} \cdot \mathbf{k}_j}{\sqrt{d_k}}, \forall j = 1, \ldots, m$$

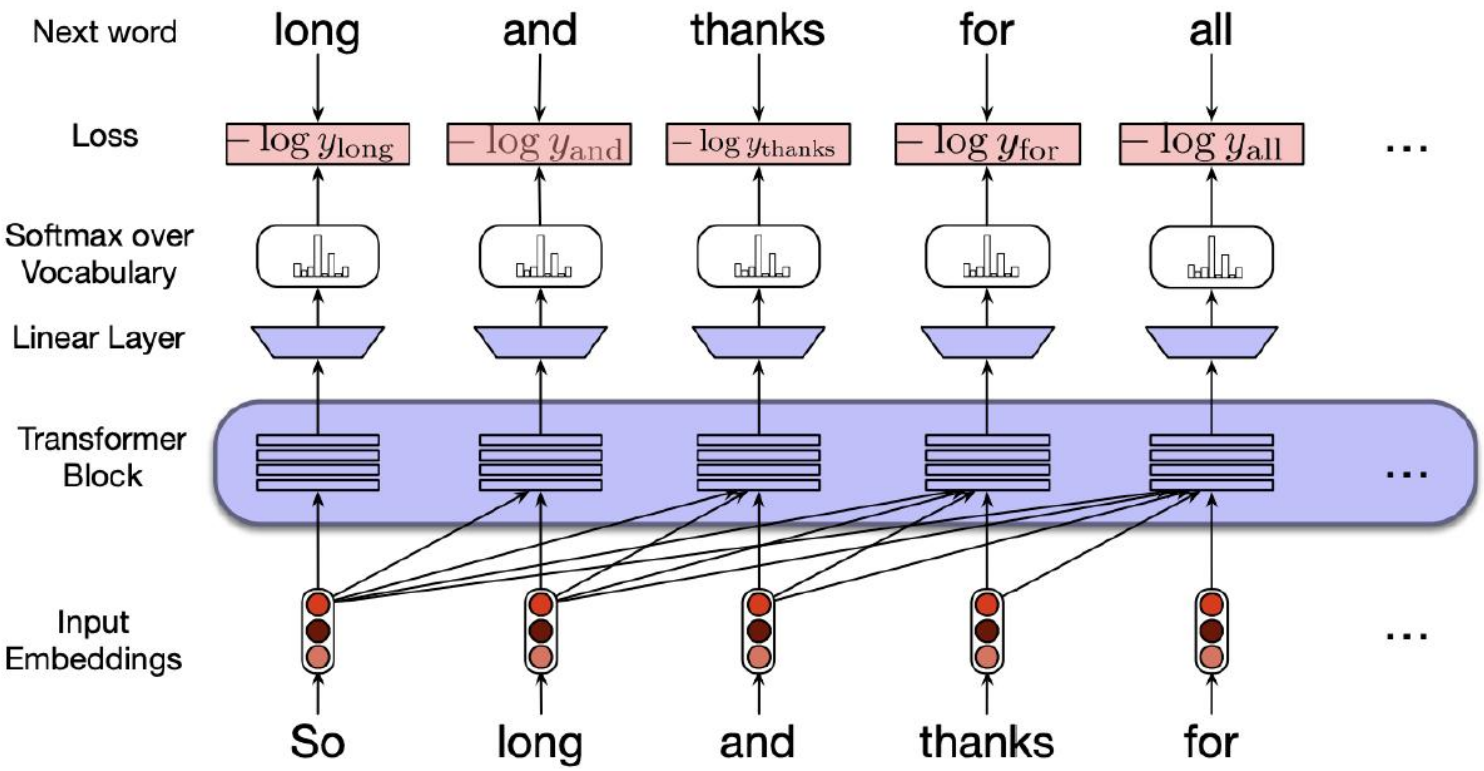
$$\alpha_i = \mathrm{softmax}(\mathbf{e}_i)$$

$$\mathbf{h}_i = \sum_{j=1}^{m} \alpha_{i,j} \mathbf{v}_j$$

Transformer语言模型

- The model architecture of GPT-3, ChatGPT, …



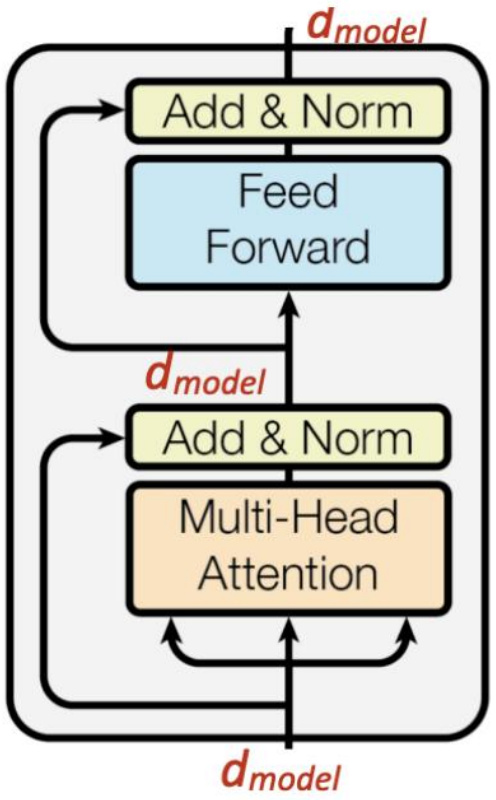


语言模型结构细节

	N	d_{model}	d_{ff}	h	d_k	d_v
base	6	512	2048	8	64	64

- From Vaswani et al.

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}
GPT-3 Small	125M	12	768	12	64
GPT-3 Medium	350M	24	1024	16	64
GPT-3 Large	760M	24	1536	16	96
GPT-3 XL	1.3B	24	2048	24	128
GPT-3 2.7B	2.7B	32	2560	32	80
GPT-3 6.7B	6.7B	32	4096	32	128
GPT-3 13B	13.0B	40	5140	40	128
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128



- From GPT-3; d_{head} is our d_k

Part.03

上下文相关词向量

Word2vec有何缺点？

- One vector for each word type

  (Aka. “Static word embeddings”)

$$v(\text{play}) = \begin{pmatrix} -0.224 \\ 0.130 \\ -0.290 \\ 0.276 \end{pmatrix}$$

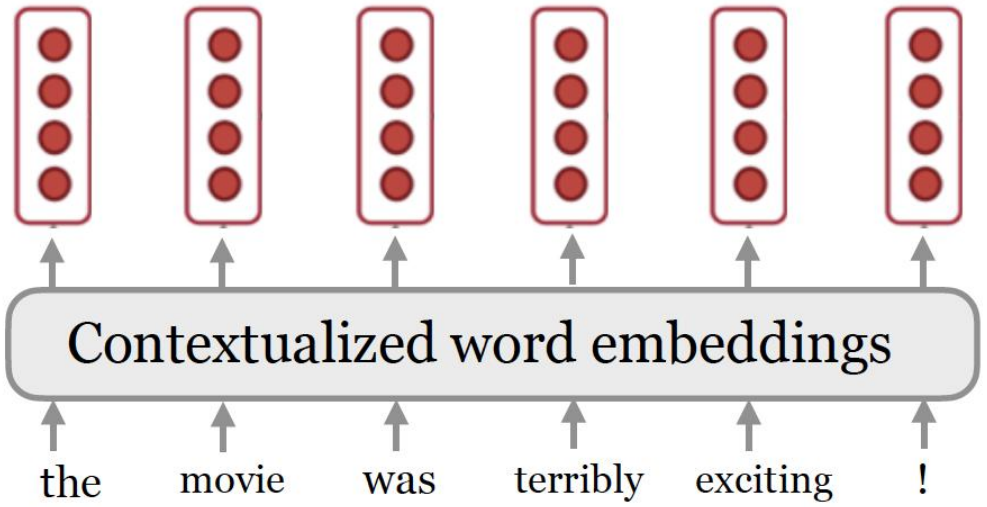
- Complex characteristics of word use: syntax and semantics

- Polysemous words, e.g., bank, mouse

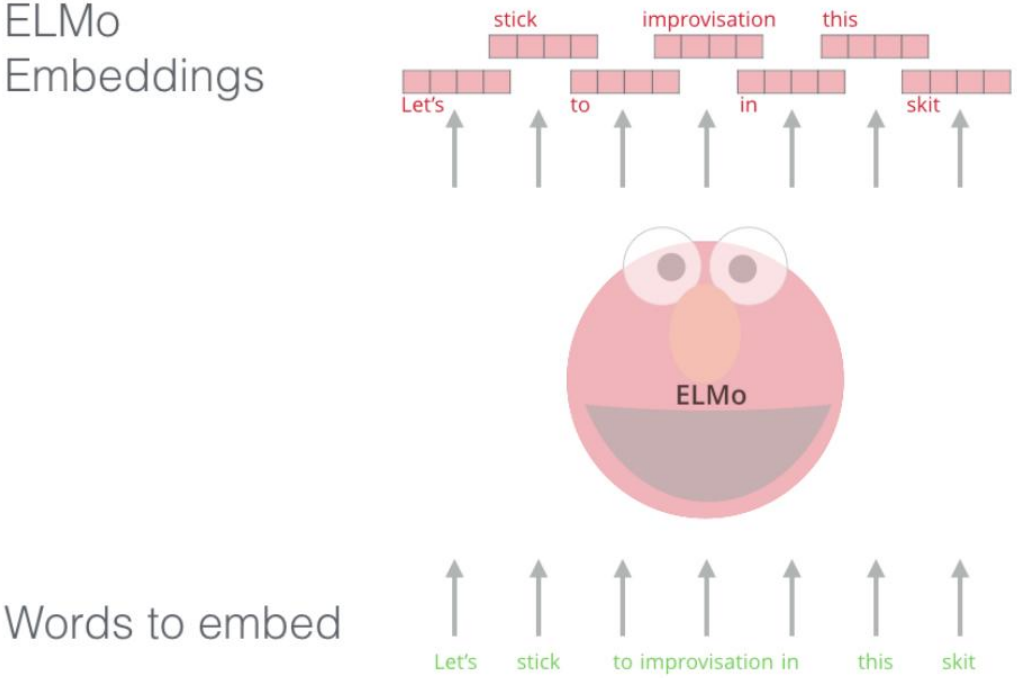
mouse^1 : .... a *mouse* controlling a computer system in 1968.
mouse^2 : .... a quiet animal like a *mouse*
bank^1 : ...a *bank* can hold the investments in a custodial account ...
bank^2 : ...as agriculture burgeons on the east *bank*, the river ...

上下文相关词向量

Let's build a vector for each word conditioned on its **context**!



$$f : (w_1, w_2, \ldots, w_n) \longrightarrow \mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$$



上下文相关词向量

Let's build a vector for each word conditioned on its **context**!

上下文相关词向量

Sent #1: Chico Ruiz made a spectacular **play** on Alusik's grounder {. . . } $v(\textbf{play}) = ?$

Sent #2: Olivia De Havilland signed to do a Broadway **play** for Garson {. . . } $v(\textbf{play}) = ?$

Sent #3: Kieffer was commended for his ability to hit in the clutch , as well as his all-round excellent **play** {. . . } $v(\textbf{play}) = ?$

Sent #4: {. . . } they were actors who had been handed fat roles in a successful **play** {. . . } $v(\textbf{play}) = ?$

Sent #5: Concepts **play** an important role in all aspects of cognition {. . . } $v(\textbf{play}) = ?$

上下文相关词向量

Sent #1: Chico Ruiz made a spectacular **play** on Alusik's grounder {. . . }

Which of the following v(play) is expected to have the most similar vector to the first one?

(A) Olivia De Havilland signed to do a Broadway **play** for Garson {. . . }

(B) Kieffer was commended for his ability to hit in the clutch , as well as his all-round excellent **play** {. . . }

(C) {. . . } they were actors who had been handed fat roles in a successful **play** {. . . }

(D) Concepts **play** an important role in all aspects of cognition {. . . }

(B) is correct.

上下文相关词向量

	Source	Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	Chico Ruiz made a spectacular play on Alusik 's grounder {...}	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his all-round excellent play .
	Olivia De Havilland signed to do a Broadway play for Garson {...}	{...} they were actors who had been handed fat roles in a successful play , and had talent enough to fill the roles competently , with nice understatement .

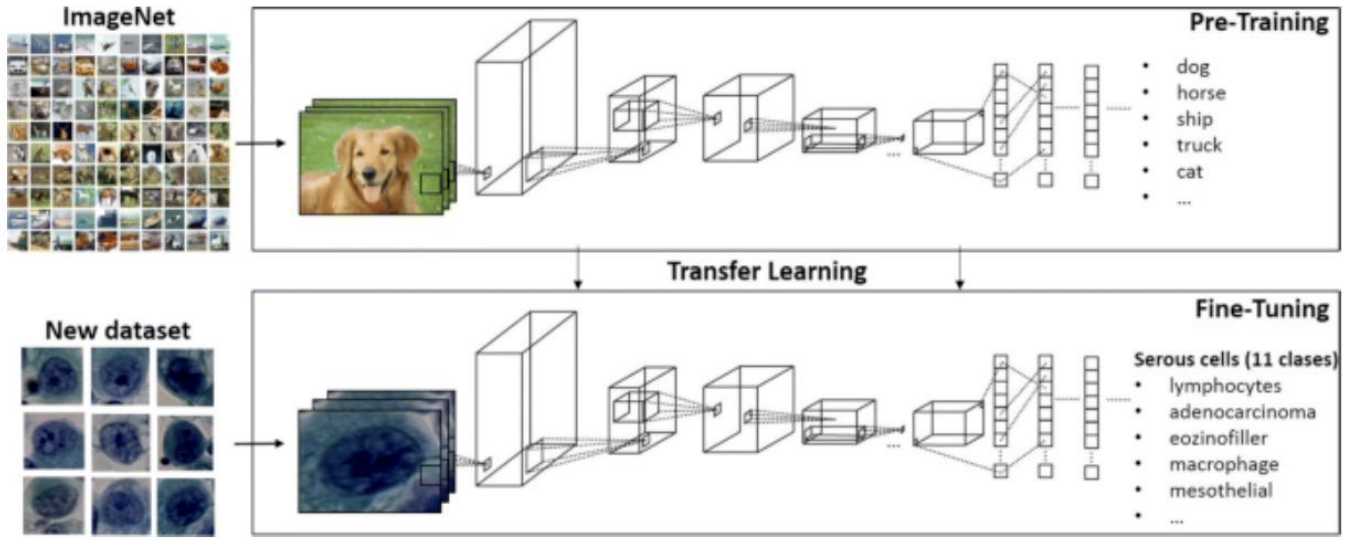
Part.04

预训练模型

什么是预训练/微调

- “Pre-train” a model on a large dataset for task X, then “fine-tune” it on a dataset for task Y
- Key idea: X is somewhat related to Y, so a model that can do X will have some good neural representations for Y as well
- ImageNet pre-training is huge in computer vision: learning generic visual features for recognizing objects

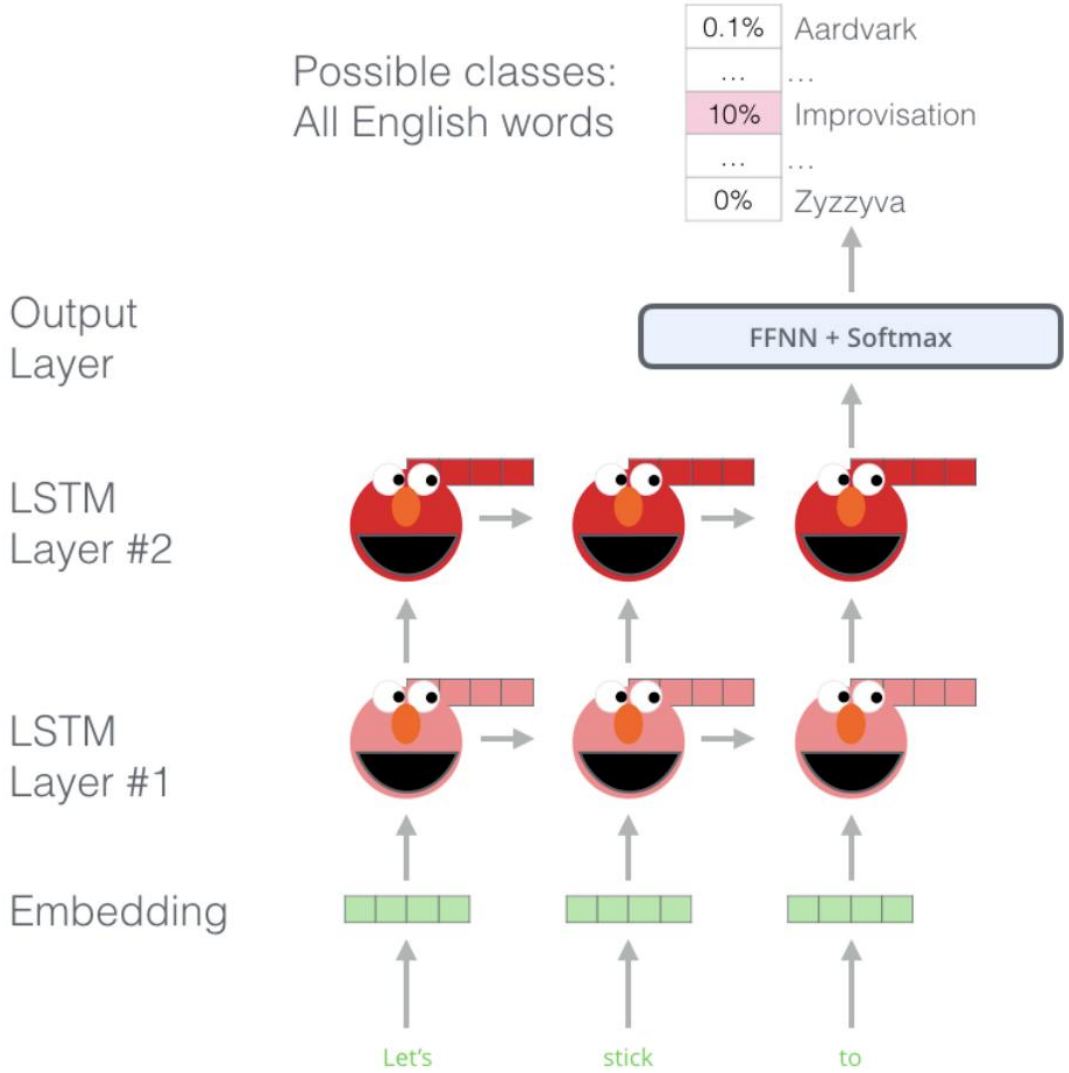
Can we find some task X that can be useful for a wide range of downstream tasks Y?



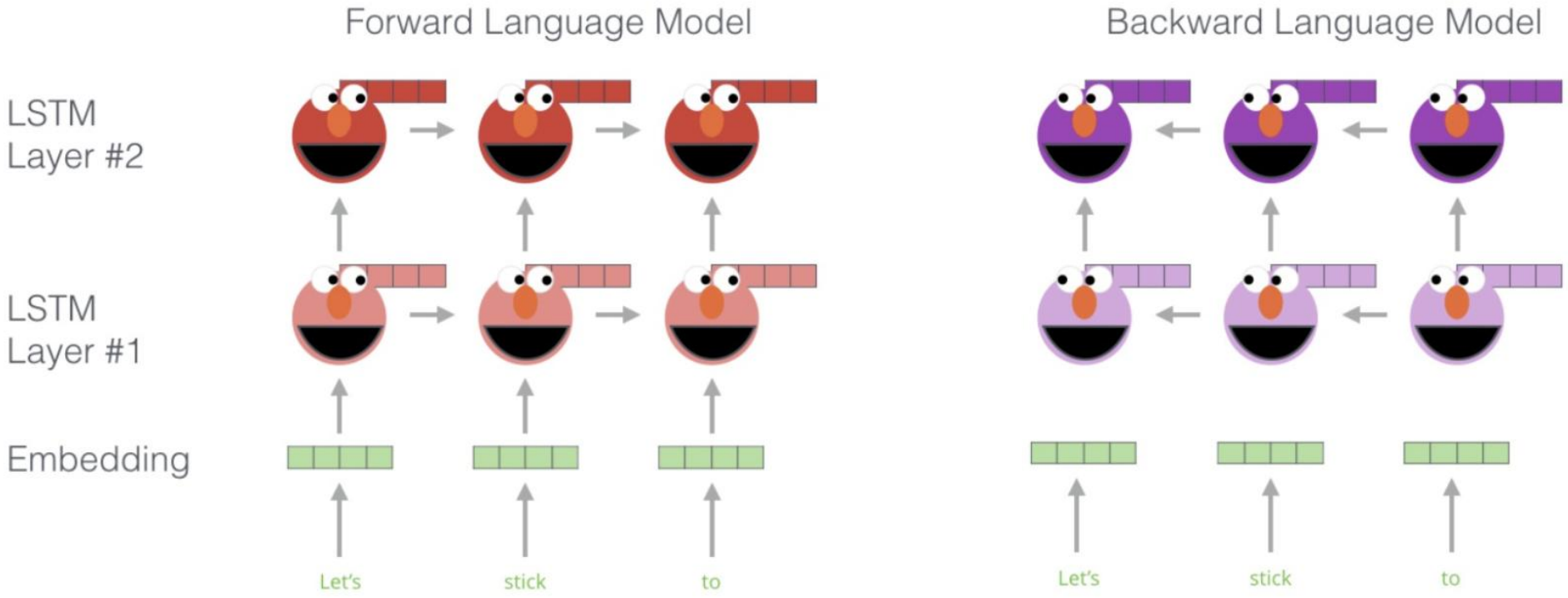
ELMo预训练模型(2018)

The key idea of ELMo:

- Train *two* stacked LSTM-based language models on a **large** corpus
- Use the **hidden states** of the LSTMs for each token to compute a vector representation of each word



ELMo模型架构

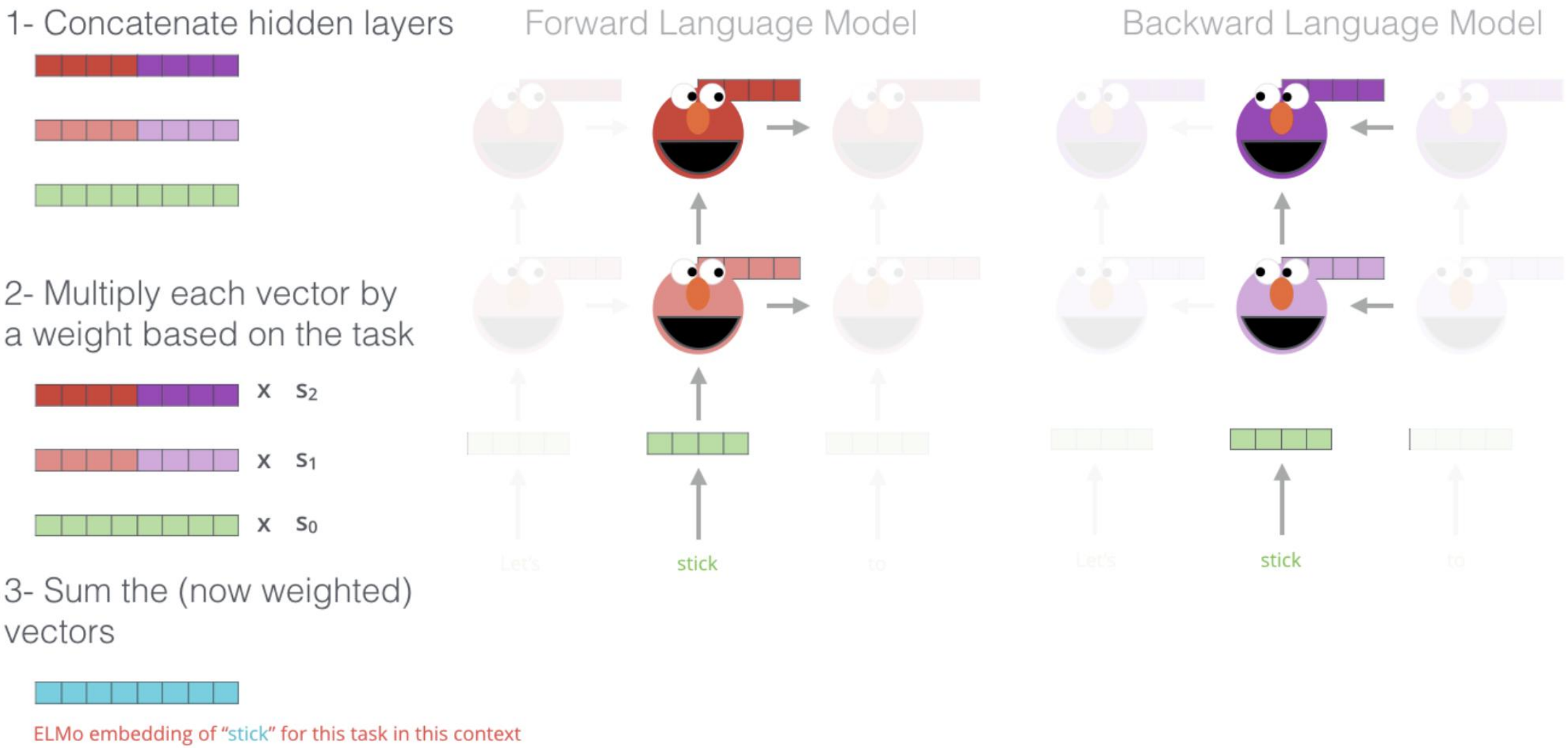


$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}$$

Contextualized word embeddings =
The weighted average of input embeddings + all hidden representations

The weights γ^{task}, s_j^{task} are task-dependent and learned

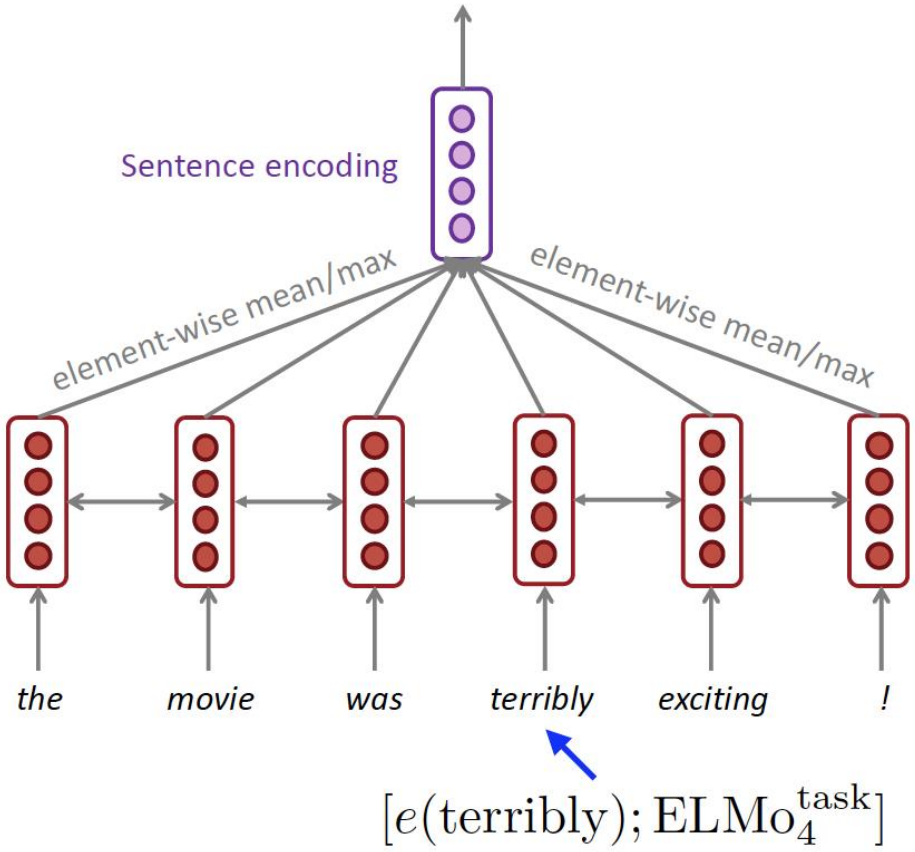
ELMo工作流程

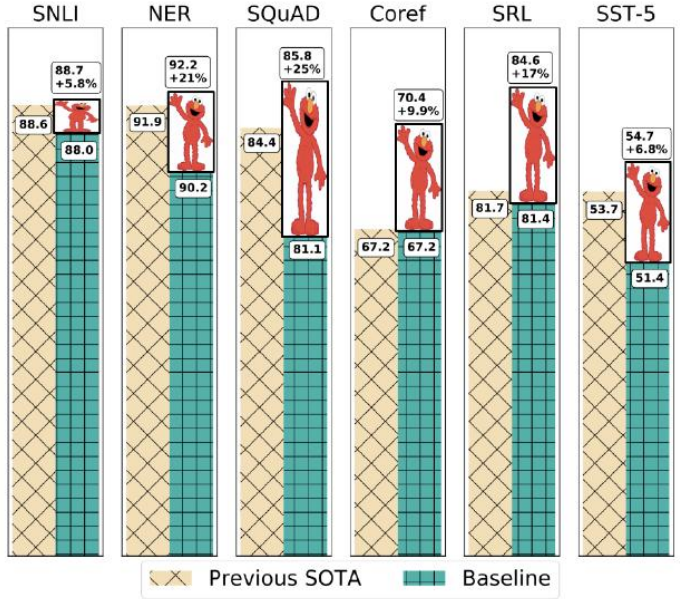

1- Concatenate hidden layers
2- Multiply each vector by a weight based on the task
x s_2
x s_1
x s_0
3- Sum the (now weighted) vectors
ELMo embedding of "stick" for this task in this context
Forward Language Model
Backward Language Model
Let's
stick
to
Let's
stick
to

ELMo预训练及应用

- Data: 10 epochs on 1B Word Benchmark (trained on **single sentences**)
- Training time: 2 weeks on 3 NVIDIA GTX 1080 GPUs

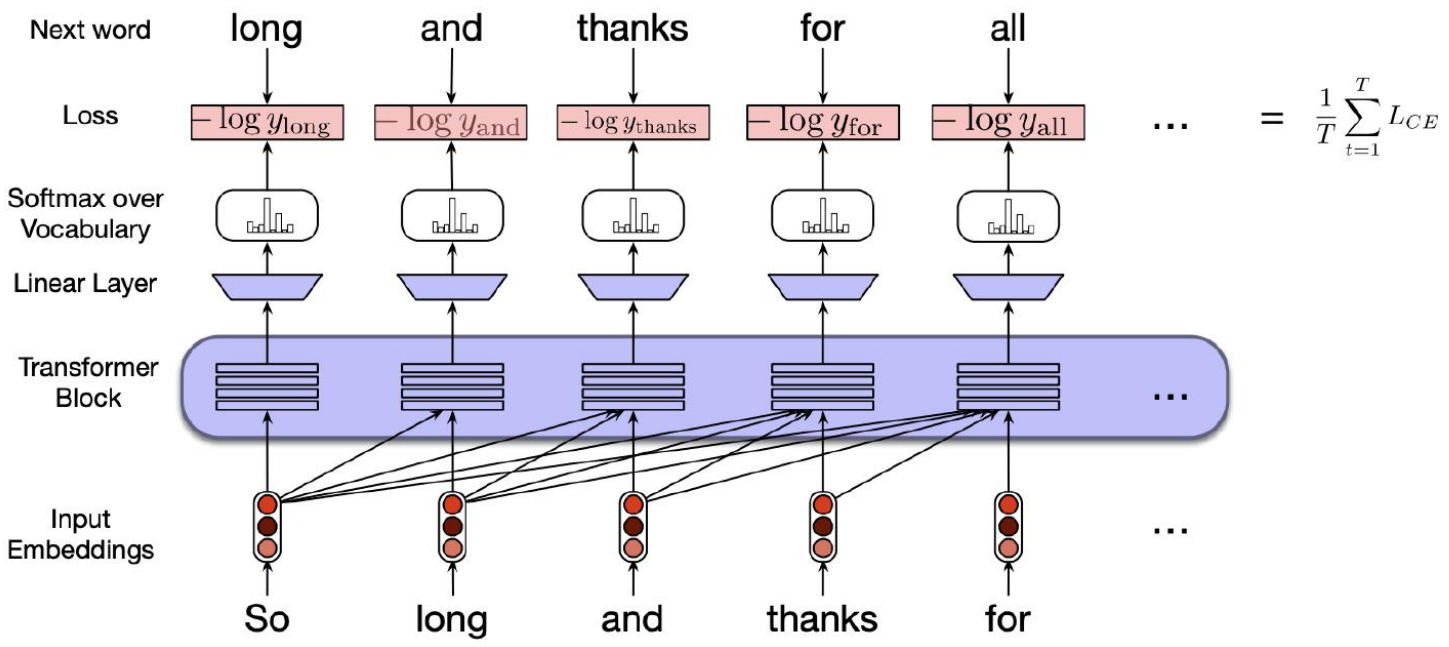
Example use: A BiLSTM model for sentiment classification





生成式预训练（GPT）

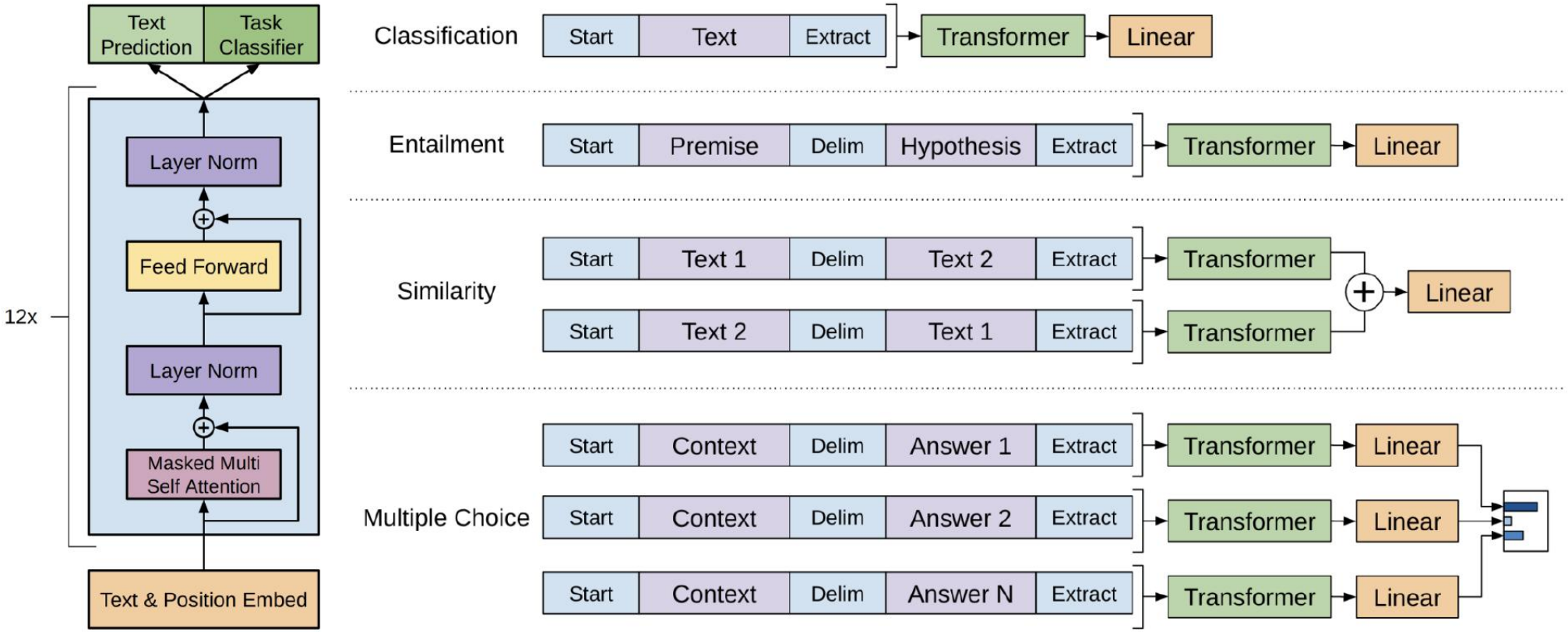
- Use a **Transformer decoder** (unidirectional; left-to-right) instead of LSTMs
- Use **language modeling** as a pre-training objective
- Trained on longer segments of text (**512 BPE tokens**), not just single sentences



(Radford et al, 2018): Improving Language Understanding by Generative Pre-Training

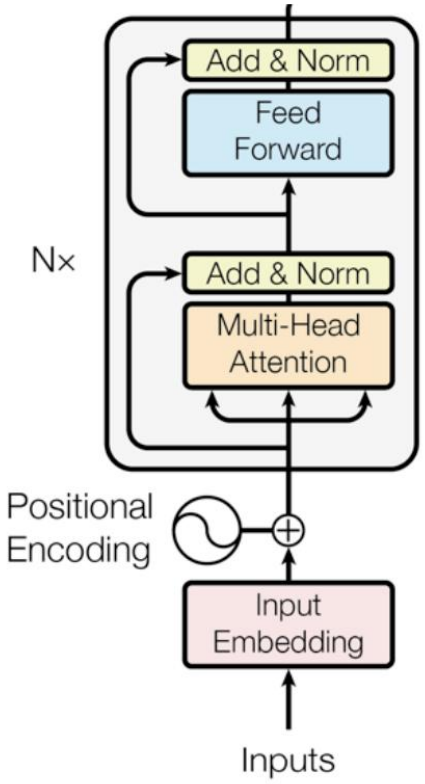
生成式预训练（GPT）

- “Fine-tune” the entire set of model parameters on various downstream tasks



(Radford et al, 2018): Improving Language Understanding by Generative Pre-Training

BERT—双向编码器



- It is a fine-tuning approach based on a deep **bidirectional Transformer encoder** instead of a Transformer decoder
- The key: learn representations based on **bidirectional contexts**

Example #1: we went to the river bank.

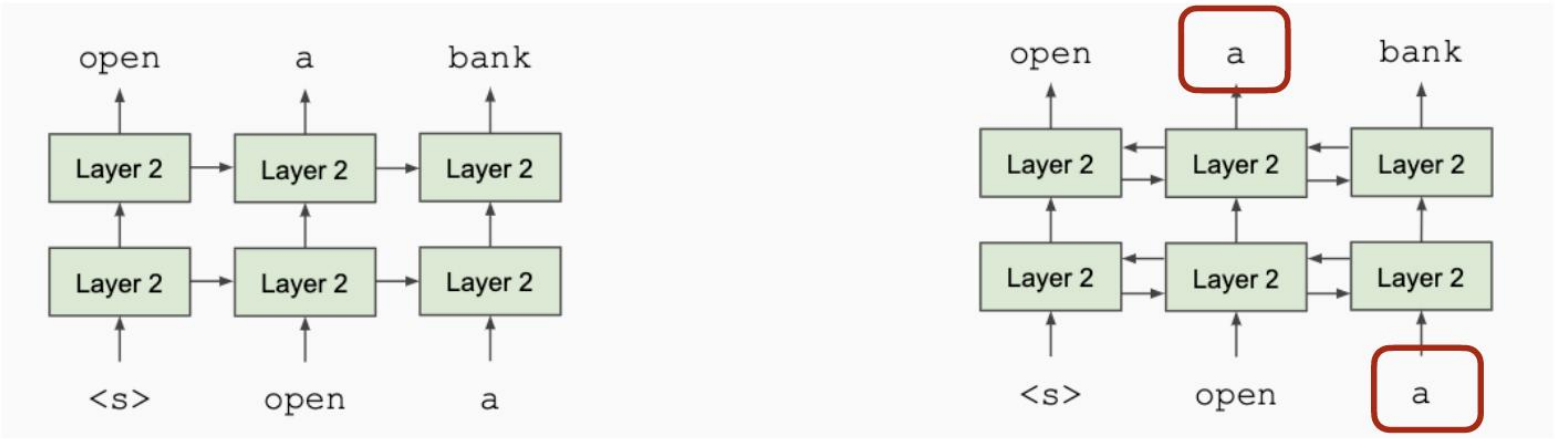
Example #2: I need to go to bank to make a deposit.

- Two new pre-training objectives:
  - **Masked language modeling (MLM)**
  - Next sentence prediction (NSP) - Later work shows that NSP hurts performance though..

(Devlin et al, 2019): BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

掩码语言建模

- Q: Why we can't do language modeling with bidirectional models?



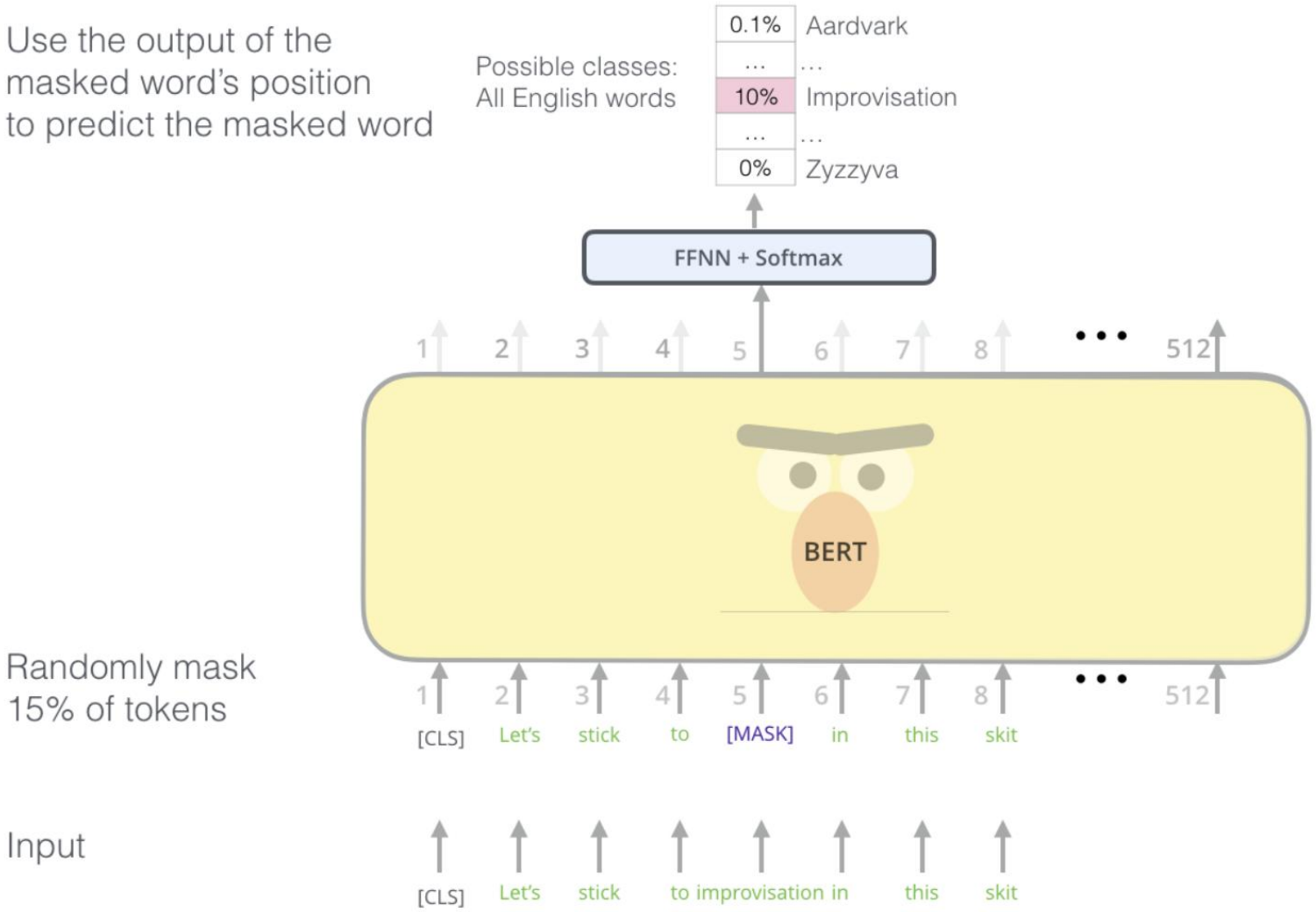
- Solution: Mask out k% of the input words, and then predict the masked words

store gallon

the man went to [MASK] to buy a [MASK] of milk

k = 15% in practice

掩码语言建模



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

掩码语言建模 – 实现

For the 15% predicted words,

- 80% of the time, they replace it with [MASK] token

  went to the store ⟶ went to the [MASK]

- 10% of the time, they replace it with a random word in the vocabulary

  went to the store ⟶ went to the running

- 10% of the time, they keep it unchanged

  went to the store ⟶ went to the store

Why? Because [MASK] tokens are never seen during fine-tuning

(See Table 8 of the paper for an ablation study)

下一句预测任务

- Motivation: many NLP downstream tasks require understanding the relationship between two sentences (natural language inference, paraphrase detection, QA)
- NSP is designed to reduce the gap between pre-training and fine-tuning

[CLS]: a special token always at the beginning

[SEP]: a special token used to separate two segments

Input = [CLS] the man went to [MASK] store [SEP]

he bought a gallon [MASK] milk [SEP]

Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]

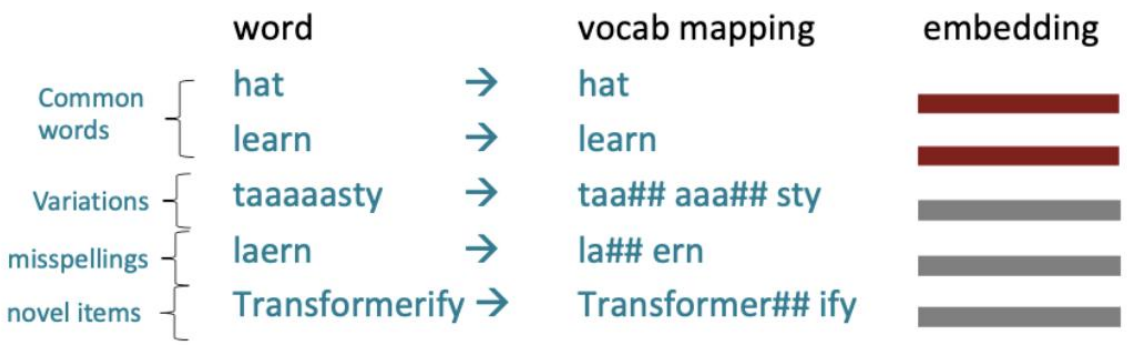
penguin [MASK] are flight ##less birds [SEP]

Label = NotNext

They sample two contiguous segments for 50% of the time and another random segment from the corpus for 50% of the time

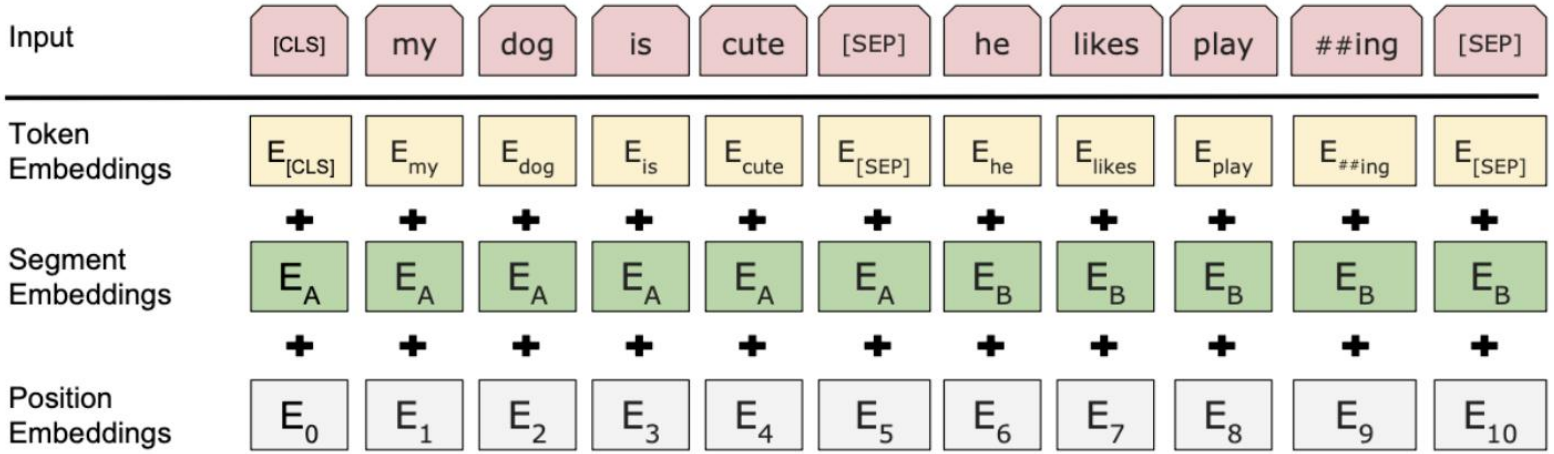
BERT预训练

- Vocabulary size: 30,000 wordpieces (common sub-word units) (Wu et al., 2016)



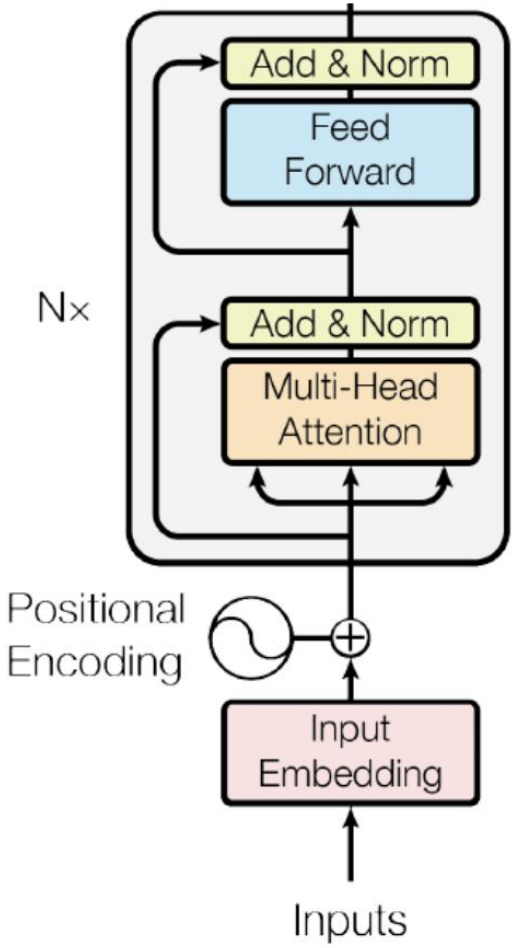
(Image: Stanford CS224N)

- Input embeddings:



Separate two segments

BERT预训练



- BERT-base: 12 layers, 768 hidden size, 12 attention heads, 110M parameters

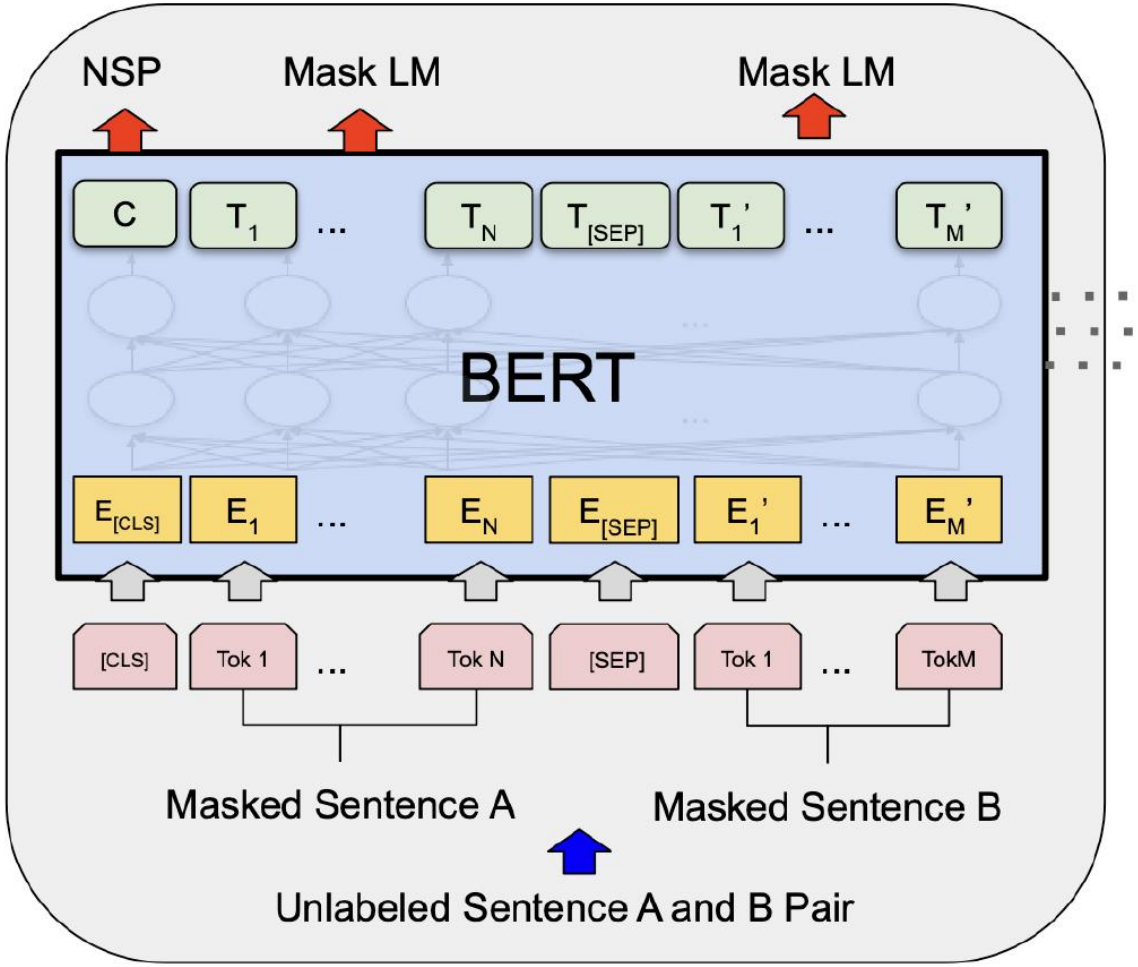
Same as OpenAI GPT

- BERT-large: 24 layers, 1024 hidden size, 16 attention heads, 340M parameters

OpenAI GPT was trained on BooksCorpus only!

- Training corpus: Wikipedia (2.5B) + BooksCorpus (0.8B)
- Max sequence size: 512 wordpiece tokens (roughly 256 and 256 for two non-contiguous sequences)
- Trained for 1M steps, batch size 128k

BERT预训练



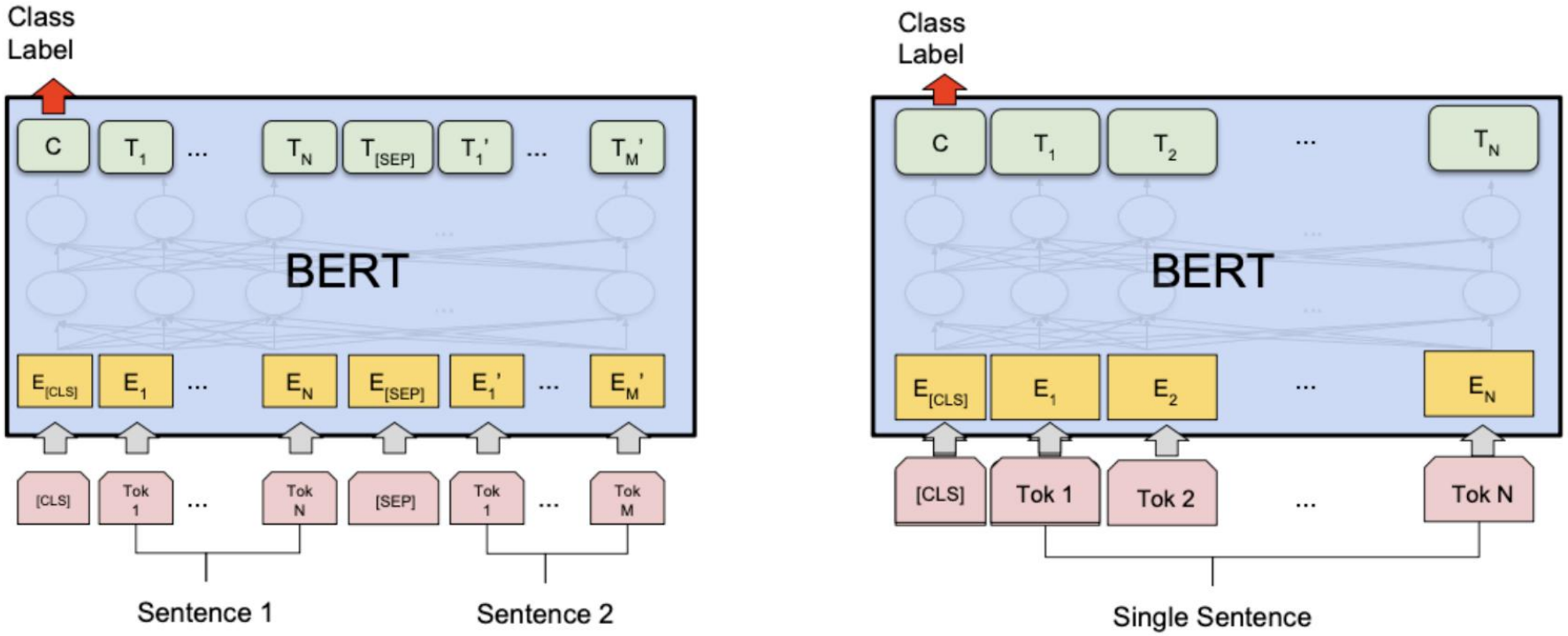
Pre-training

- MLM and NSP are trained together
- [CLS] is pre-trained for NSP
- Other token representations are trained for MLM

BERT微调

"Pretrain once, finetune many times."

sentence-level tasks



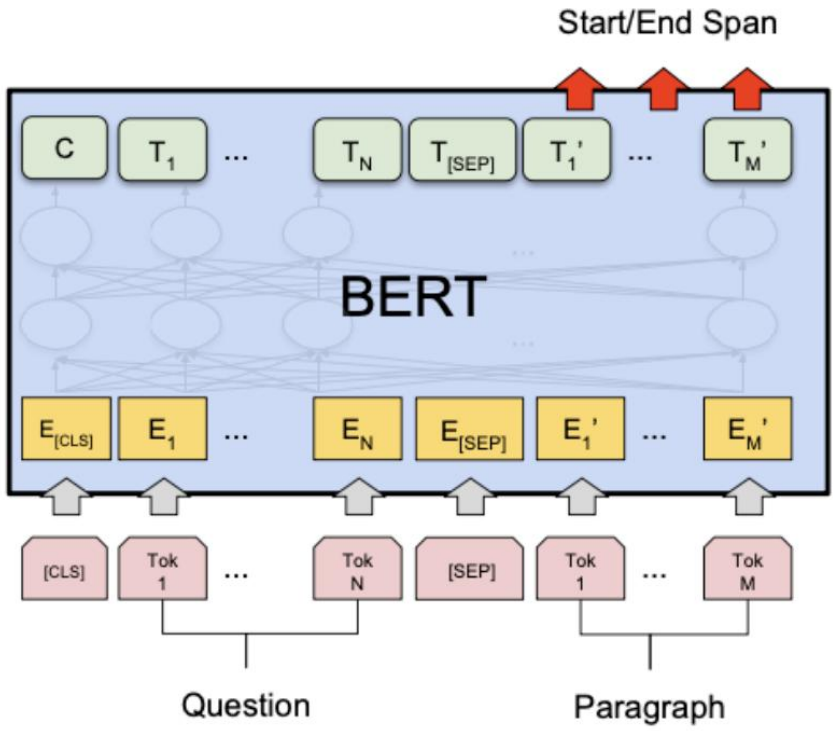
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

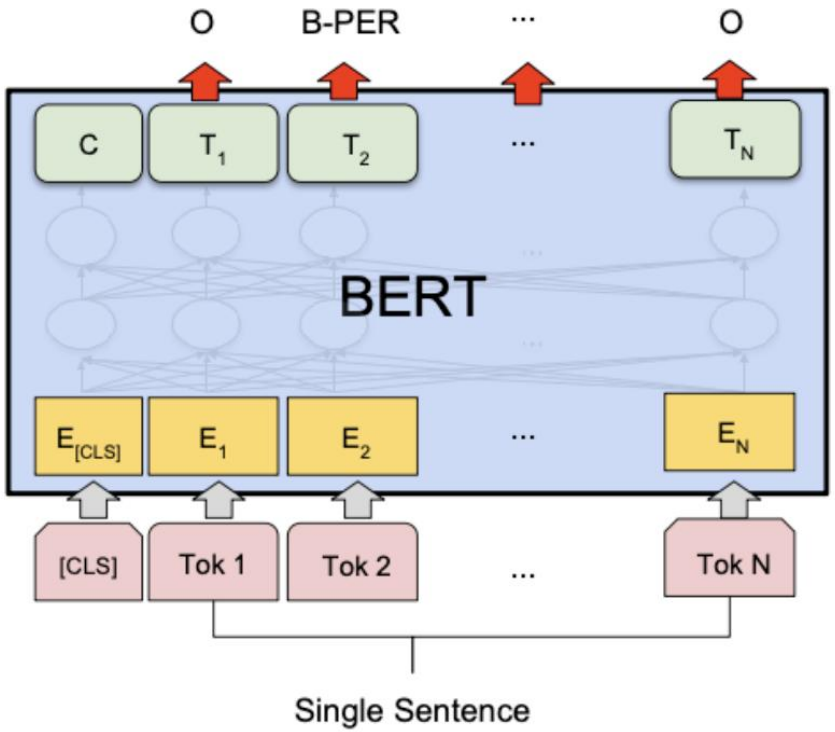
BERT微调

"Pretrain once, finetune many times."

token-level tasks



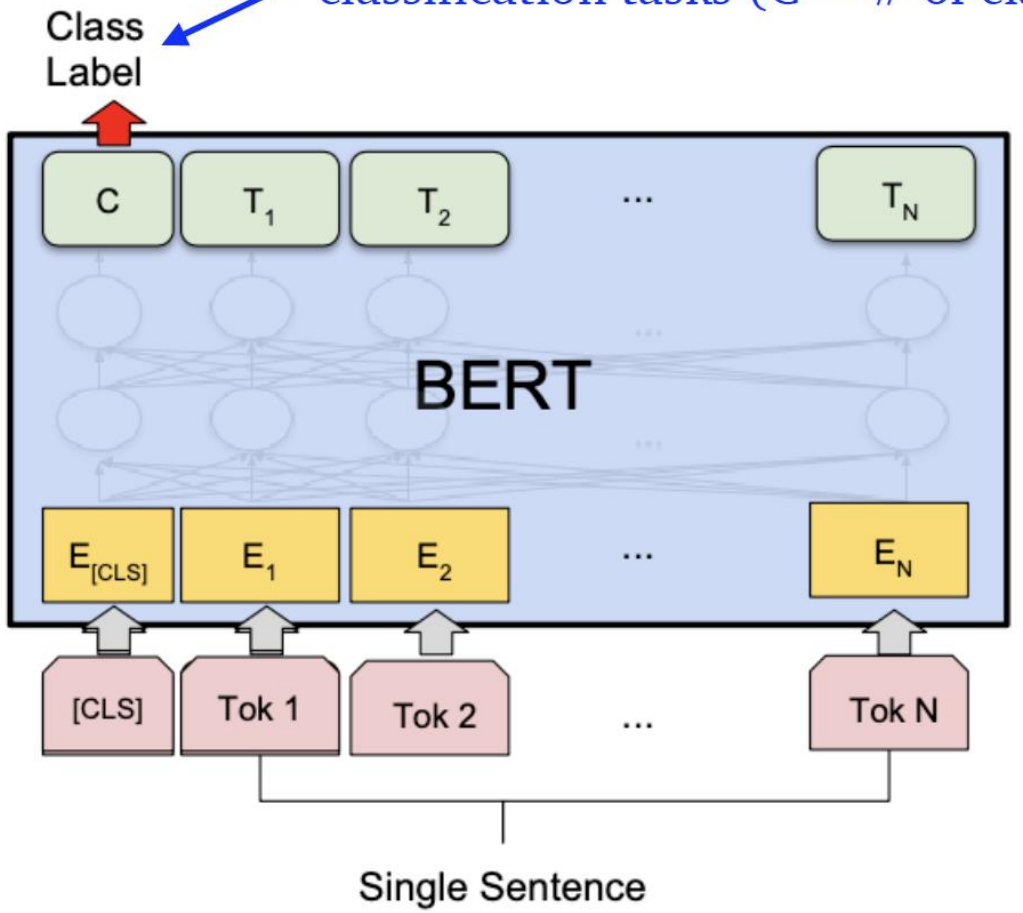
(c) Question Answering Tasks:
SQuAD v1.1



(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

BERT情感分类

We just need to introduce $C \times h$ parameters for classification tasks (C = # of classes, h = hidden size)!



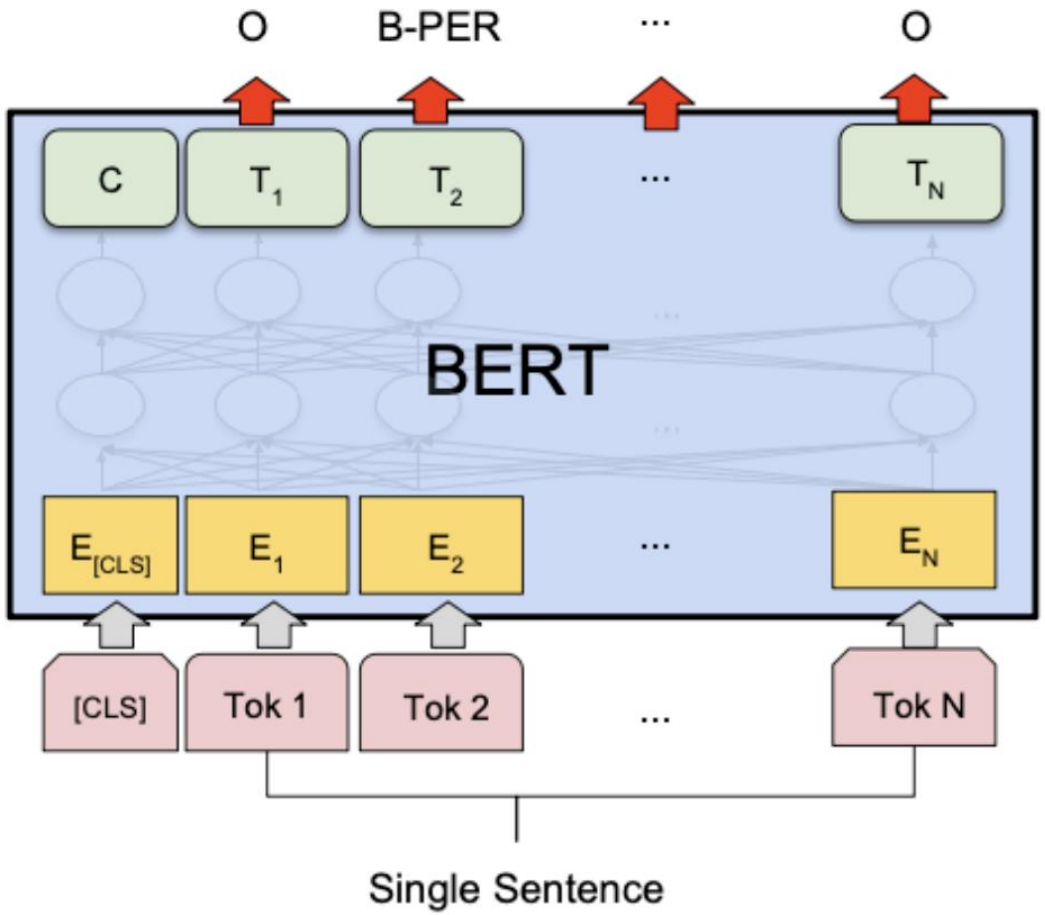
$$P(y = k) = softmax_k(\mathbf{W}_o \mathbf{h}_{[CLS]})$$

$$\mathbf{W}_o \in \mathbb{R}^{C \times h}$$

All the parameters will be learned together (original BERT parameters + new classifier parameters)

BERT命名实体识别

We just need to introduce $C \times h$ parameters for classification tasks (C = # of classes, h = hidden size)!



$$P(y_i = k) = softmax_k(\mathbf{W}_o \mathbf{h}_i)$$

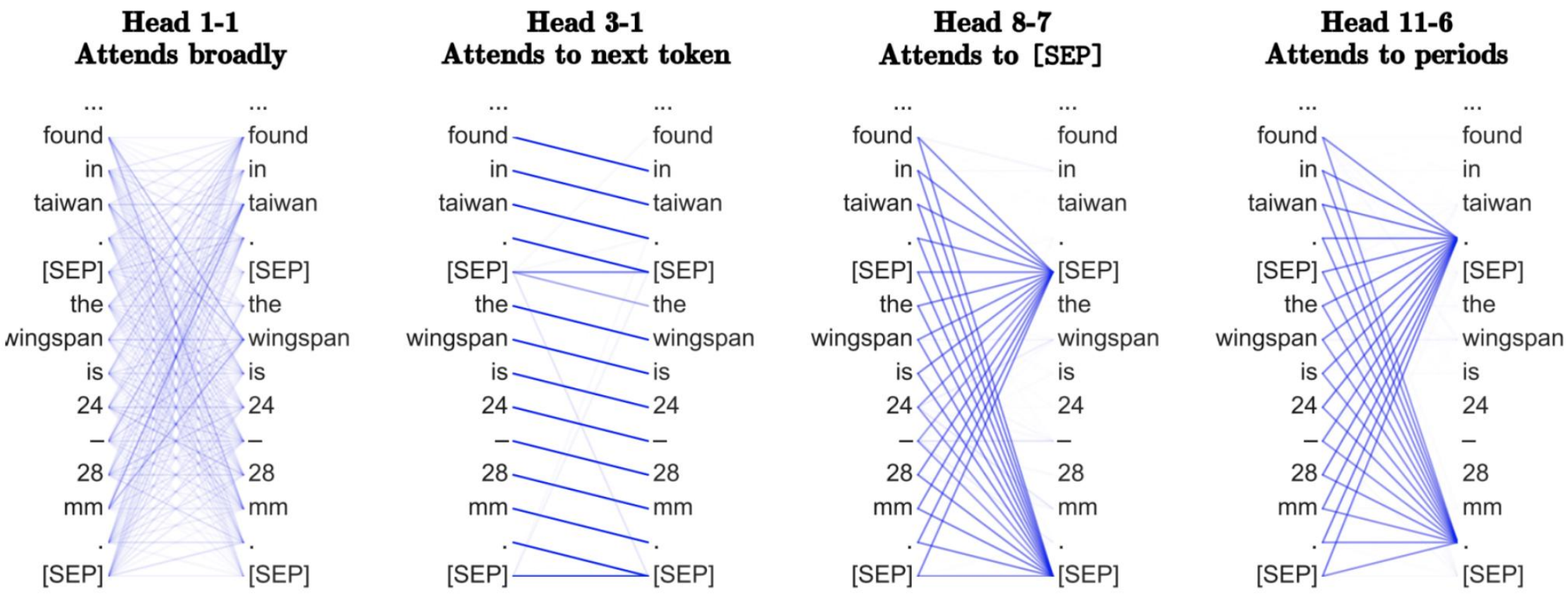
$$\mathbf{W}_o \in \mathbb{R}^{C \times h}$$

BERT性能验证

System	MNLI-(m/mm) 392k	QQP 363k
Pre-OpenAI SOTA	80.6/80.1	66.1
BiLSTM+ELMo+Attn	76.4/76.1	64.8
OpenAI GPT	82.1/81.4	70.3
$BERT_{BASE}$	84.6/83.4	71.2
$BERT_{LARGE}$	**86.7/85.9**	**72.1**

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$BERT_{BASE}$ (Single)	80.8	88.5	-	-
$BERT_{LARGE}$ (Single)	84.1	90.9	-	-
$BERT_{LARGE}$ (Ensemble)	85.8	91.8	-	-
$BERT_{LARGE}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$BERT_{LARGE}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

BERT学到了什么？


Head 1-1
Attends broadly
Head 3-1
Attends to next token
Head 8-7
Attends to [SEP]
Head 11-6
Attends to periods
... found in taiwan . [SEP] the wingspan is 24 – 28 mm . [SEP]

揚州大學
YANGZHOU UNIVERSITY

谢 谢

信息工程学院
（人工智能学院）