北京航空航天大學
BEIHANG UNIVERSITY

计算语言学理论与实践

人工智能研究院

主讲教师 沙磊

Transformers PLMs

Contents

- Transformers
  - Impact of Transformers on NLP (and ML more broadly)
  - From Recurrence (RNNs) to Attention-Based NLP Models
  - Understanding the Transformer Model
  - Drawbacks and Variants of Transformers
- Pretraining Language Models(PLMs)
  - Subword modeling
  - Motivating model pretraining from word embeddings
  - Model pretraining three ways
    - Decoders
    - Encoders
    - Encoder-Decoders
  - Very large models and in-context learning

Contents

- Transformers
  - Impact of Transformers on NLP (and ML more broadly)
  - From Recurrence (RNNs) to Attention-Based NLP Models
  - Understanding the Transformer Model
  - Drawbacks and Variants of Transformers
- Pretraining Language Models(PLMs)
  - Subword modeling
  - Motivating model pretraining from word embeddings
  - Model pretraining three ways
    - Decoders
    - Encoders
    - Encoder-Decoders
  - Very large models and in-context learning

Transformers: Is Attention All We Need?

Spoiler: Not Quite!

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

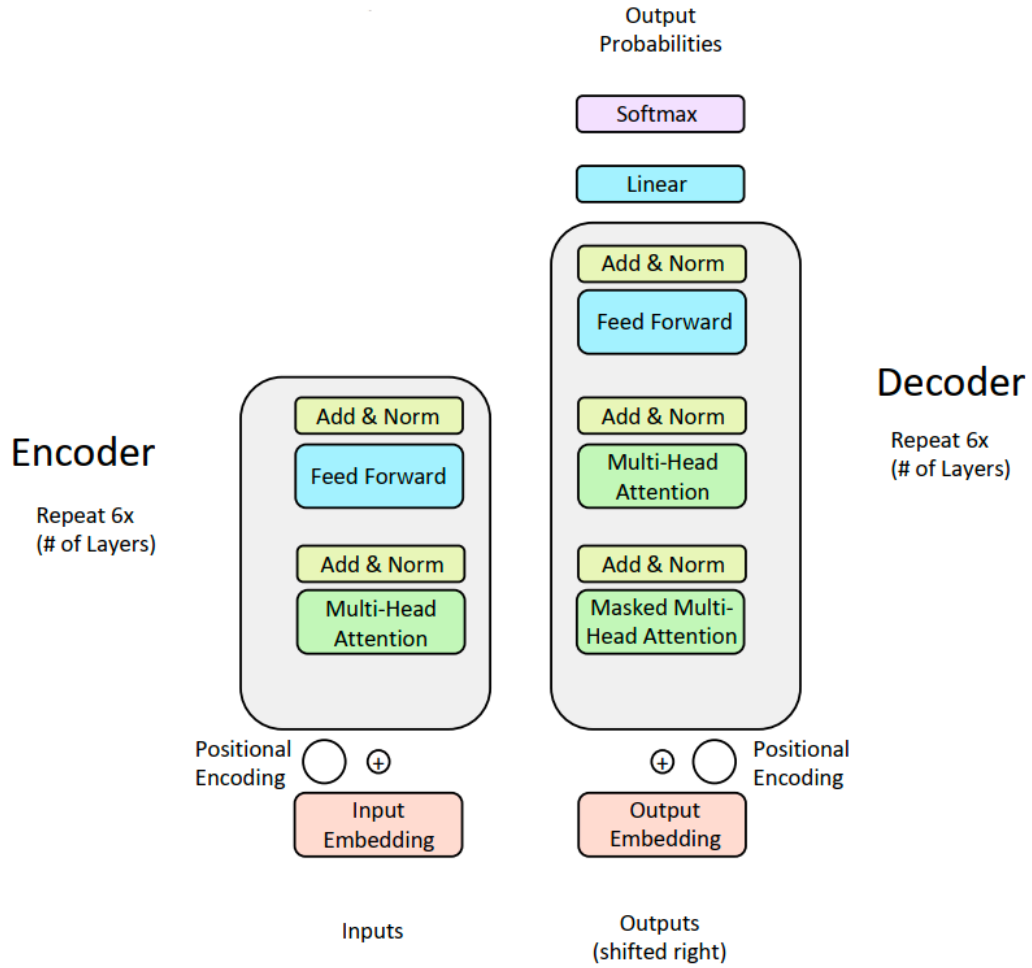
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Transformers Have Revolutionized the Field of NLP

Courtesy of Paramount Pictures



Great Results with Transformers: Machine Translation

- First, Machine Translation results from the original Transformers paper!

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Great Results with Transformers: Document Generation

- Next, document generation!
- (For perplexity, lower is better; for ROUGE-L, higher is better.)

Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

The old standard from last week!

Transformers dominating across the board.

Preview: Great Results with (Pre-Trained) Transformers

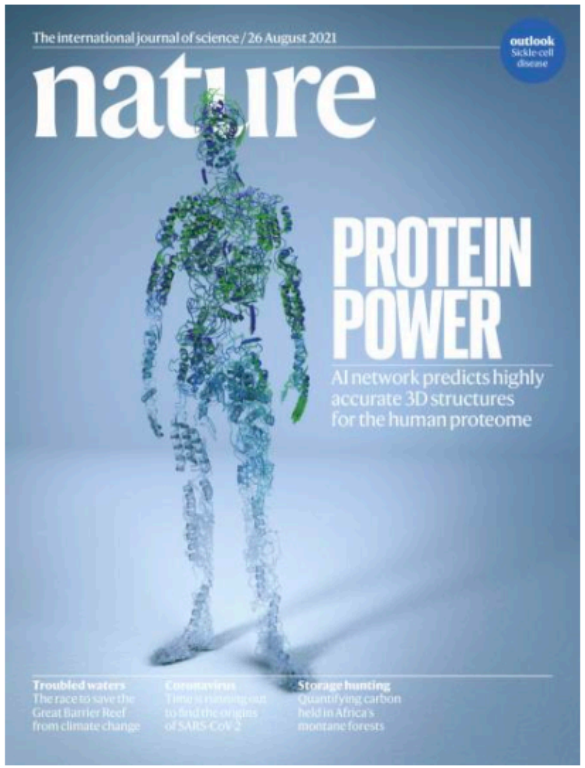
- Transformers可以用来预训练
- Transformer的并行性使得预训练很高效

All top models are Transformer (and pretraining)-based

	Rank	Name	Model	URL	Score
	1	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.8
	2	HFL iFLYTEK	MacALBERT + DKM		90.7
+	3	Alibaba DAMO NLP	StructBERT + TAPT		90.6
+	4	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6
	5	ERNIE Team - Baidu	ERNIE		90.4
	6	T5 Team - Google	T5		90.3

Transformers Even Show Promise Outside of NLP

Protein Folding



[Jumper et al. 2021] aka AlphaFold2!

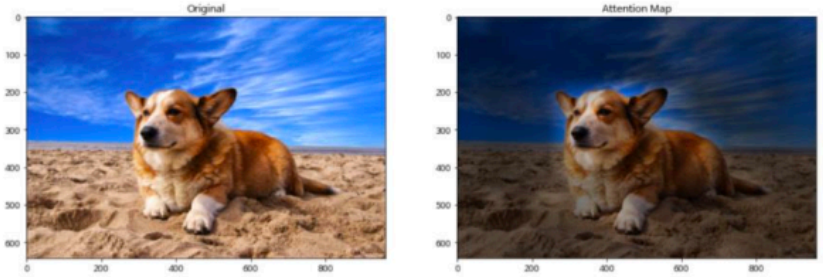
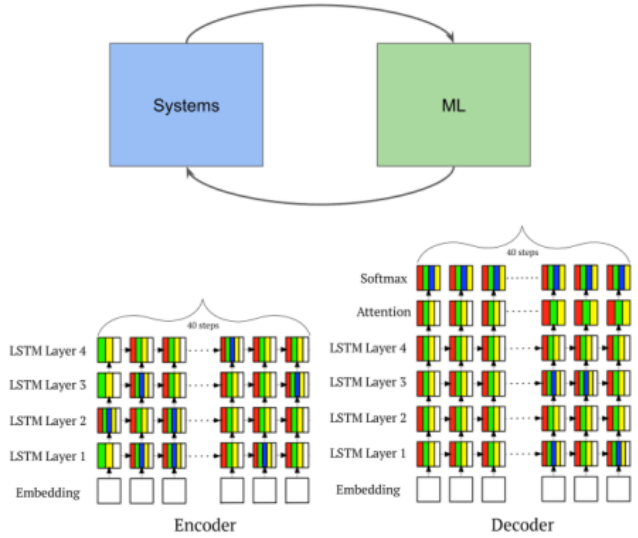


Image Classification

[Dosovitskiy et al. 2020]: Vision Transformer (ViT) outperforms ResNet-based baselines with substantially less compute.

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k



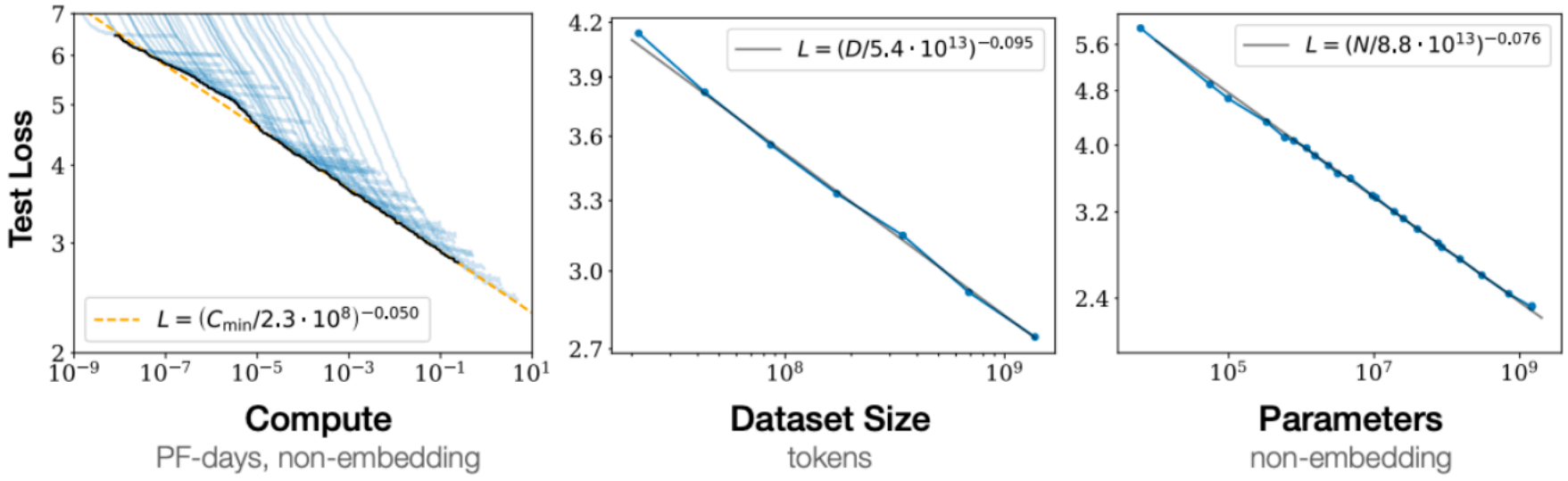
ML for Systems

[Zhou et al. 2020]: A Transformer-based compiler model (GO-one) speeds up a Transformer model!

Model (#devices)	GO-one (s)	HP (s)	METIS (s)	HDP (s)	Run time speed up over HP / HDP	Search speed up over HDP
2-layer RNNLM (2)	0.173	0.192	0.355	0.191	9.9% / 9.4%	2.95x
4-layer RNNLM (4)	0.210	0.239	0.503	0.251	13.8% / 16.3%	1.76x
8-layer RNNLM (8)	0.320	0.332	OOM	0.764	3.8% / 58.1%	27.8x
2-layer GNMT (2)	0.301	0.384	0.344	0.327	27.6% / 14.3%	30x
4-layer GNMT (4)	0.350	0.469	0.466	0.432	34% / 23.4%	58.8x
	0.440	0.562	OOM	0.693	21.7% / 36.5%	7.35x
2-layer Transformer-XL (2)	0.223	0.268	0.37	0.262	20.1% / 17.4%	40x
4-layer Transformer-XL (4)	0.230	0.27	OOM	0.259	17.4% / 12.6%	26.7x
8-layer Transformer-XL (8)	0.350	0.46	OOM	0.425	23.9% / 16.7%	16.7x
	0.229	0.312	OOM	0.301	26.6% / 23.9%	13.5x
Inception (2) b64	0.423	0.731	OOM	0.498	42.1% / 29.3%	21.0x
AmoebaNet (4)	0.394	0.44	0.426	0.418	26.1% / 6.1%	58.8x
2-stack 18-layer WaveNet (2)	0.317	0.376	OOM	0.354	18.6% / 11.7%	6.67x
4-stack 36-layer WaveNet (4)	0.659	0.988	OOM	0.721	50% / 9.4%	20x
GEOMEAN	-	-	-	-	**20.5% / 18.2%**	**15x**

Scaling Laws: Are Transformers All We Need?

- 有了Transformers，语言模型的效果可以跟随模型大小，训练数据规模以及计算资源量的增加而提升
- 如果我们在不改变架构的情况下持续增大这些模型，有无可能最终超越人类?



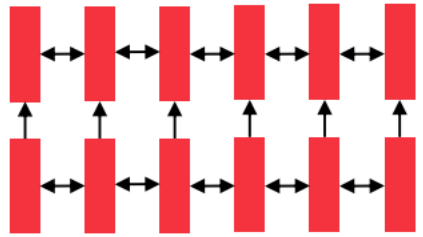
[Kaplan et al., 2020]

Contents

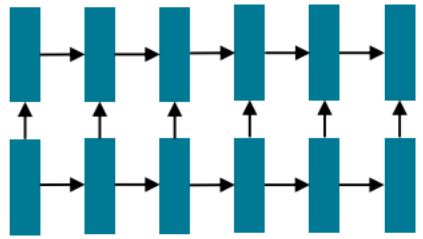
- Transformers
  - Impact of Transformers on NLP (and ML more broadly)
  - From Recurrence (RNNs) to Attention-Based NLP Models
  - Understanding the Transformer Model
  - Drawbacks and Variants of Transformers
- Pretraining Language Models(PLMs)
  - Subword modeling
  - Motivating model pretraining from word embeddings
  - Model pretraining three ways
    - Decoders
    - Encoders
    - Encoder-Decoders
  - Very large models and in-context learning

Recurrent models for (most) NLP!

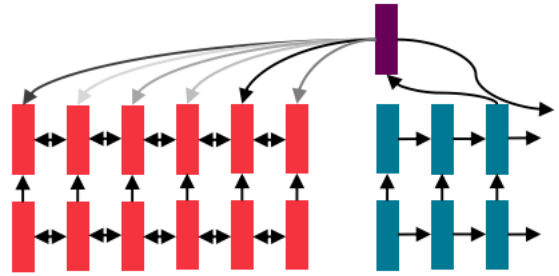
- Since 2016, the de facto strategy in NLP is to **encode** sentences with a bidirectional LSTM: (for example, the source sentence in a translation)



- Define your output (parse, sentence, summary) as a sequence, and use an LSTM to generate it.

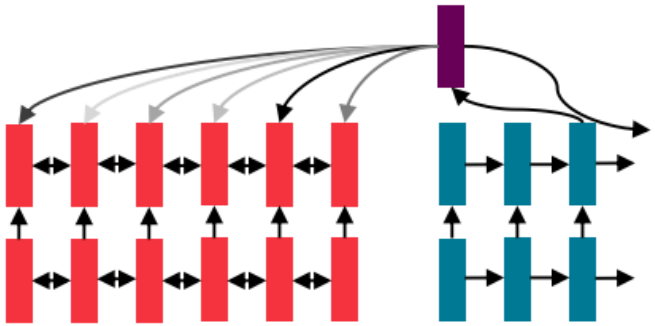


- Use attention to allow flexible access to memory



Today: Same goals, different building blocks

- We have learned about sequence-to-sequence problems and encoder-decoder models.
- Today, we're not trying to motivate entirely new ways of looking at problems (like Machine Translation)
- Instead, we're trying to find the best building blocks to plug into our models and enable broad progress.



Lots of trial and error

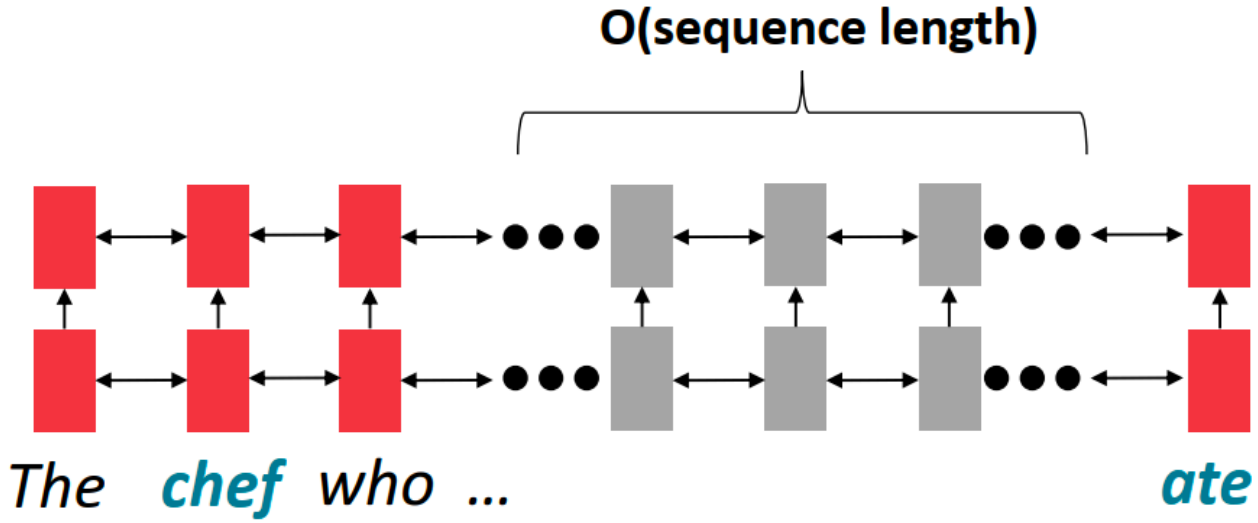
2014-2017ish Recurrence

2021 ?????

Issues with recurrent models: Linear interaction distance

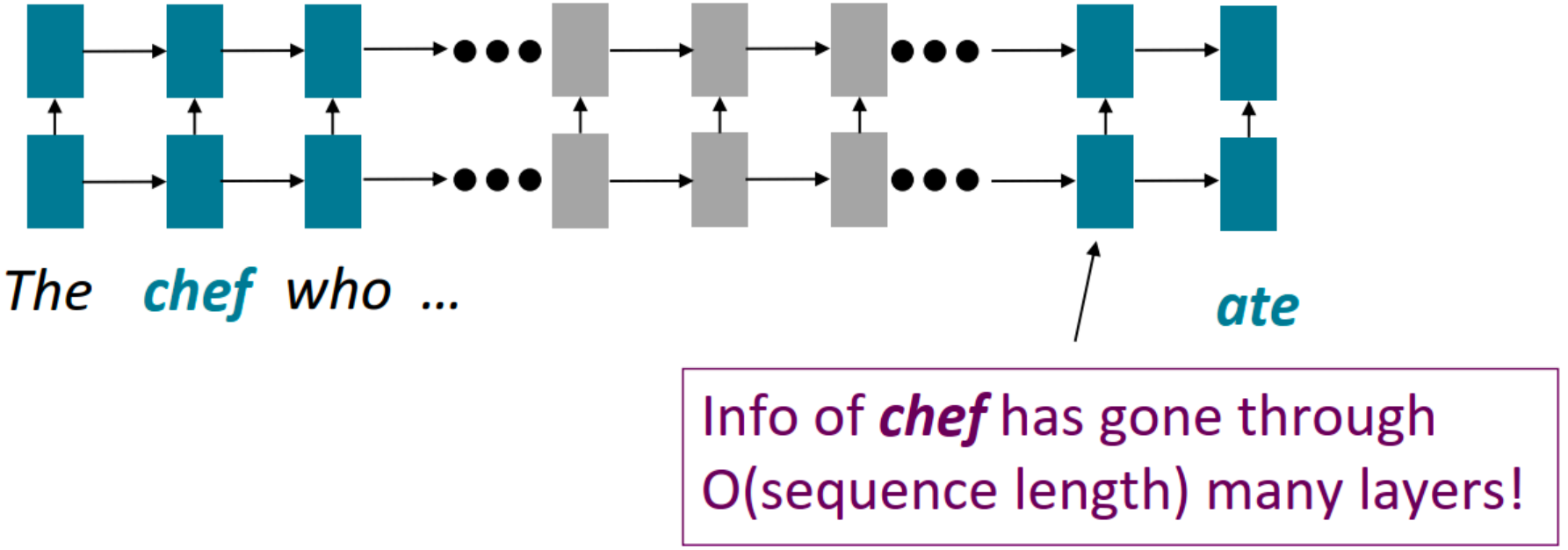
- RNNs are unrolled "left-to-right".
- It encodes linear locality: a useful heuristic!
  - Nearby words often affect each other's meanings

- Problem: RNNs take **O(sequence length)** steps for distant word pairs to interact.



Issues with recurrent models: Linear interaction distance

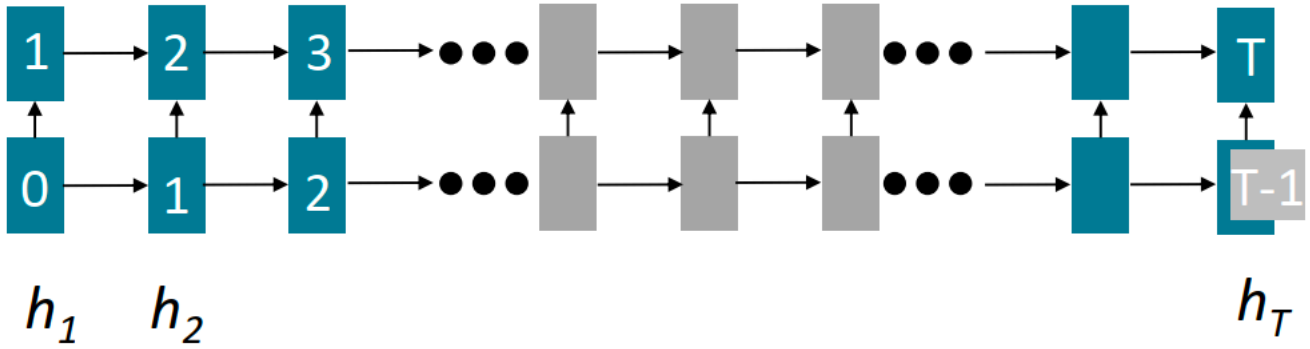
- 远距离的词的交互需要O(sequence length) 步，意味着：
  - 长距离依存关系很难学习（梯度消失）
  - 词的线性顺序被模型限定



Issues with recurrent models: Lack of parallelizability

- 前向与后向计算均无法并行实现
  - GPU（或TPU）可以同时做很多不相关的计算
  - 但未来的RNN隐状态无法在之前的RNN隐状态计算完成之前进行计算
  - 对于大数据集训练很不利
  - 序列长度越长问题越大。Why?

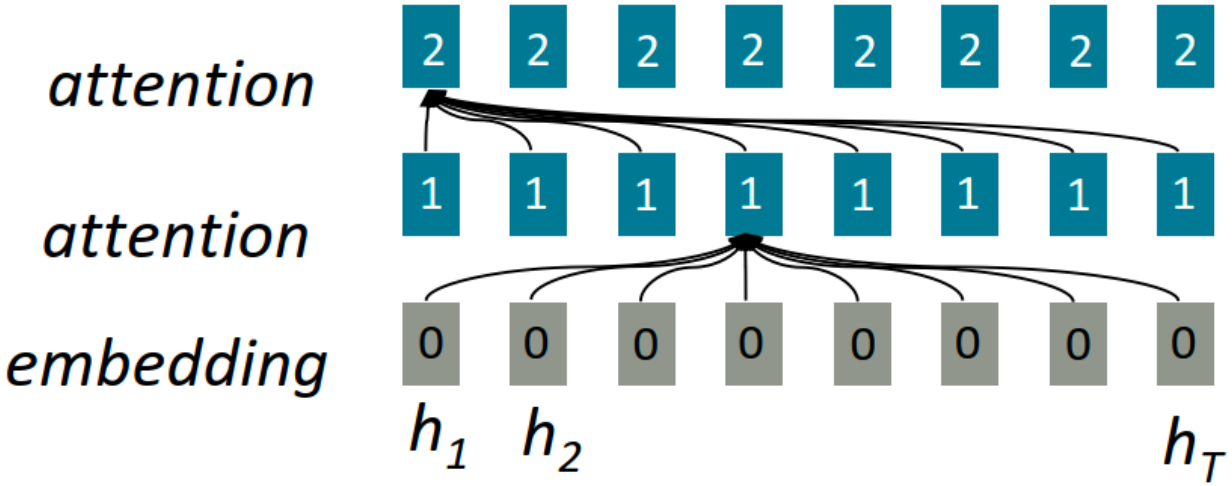
因为batch 就得相应调小



Numbers indicate min # of steps before a state can be computed

If not recurrence, then what? **How about (self) attention?**

- 回忆：Attention：将每个词的表示看做query，从一组value中检索信息
  - 我们学过从decoder到encoder的attention
  - **Self-attention**是encoder-encoder或decoder-decoder的attention，每一个词都要关注到其他的词



All words attend to all words in previous layer; most arrows here are omitted

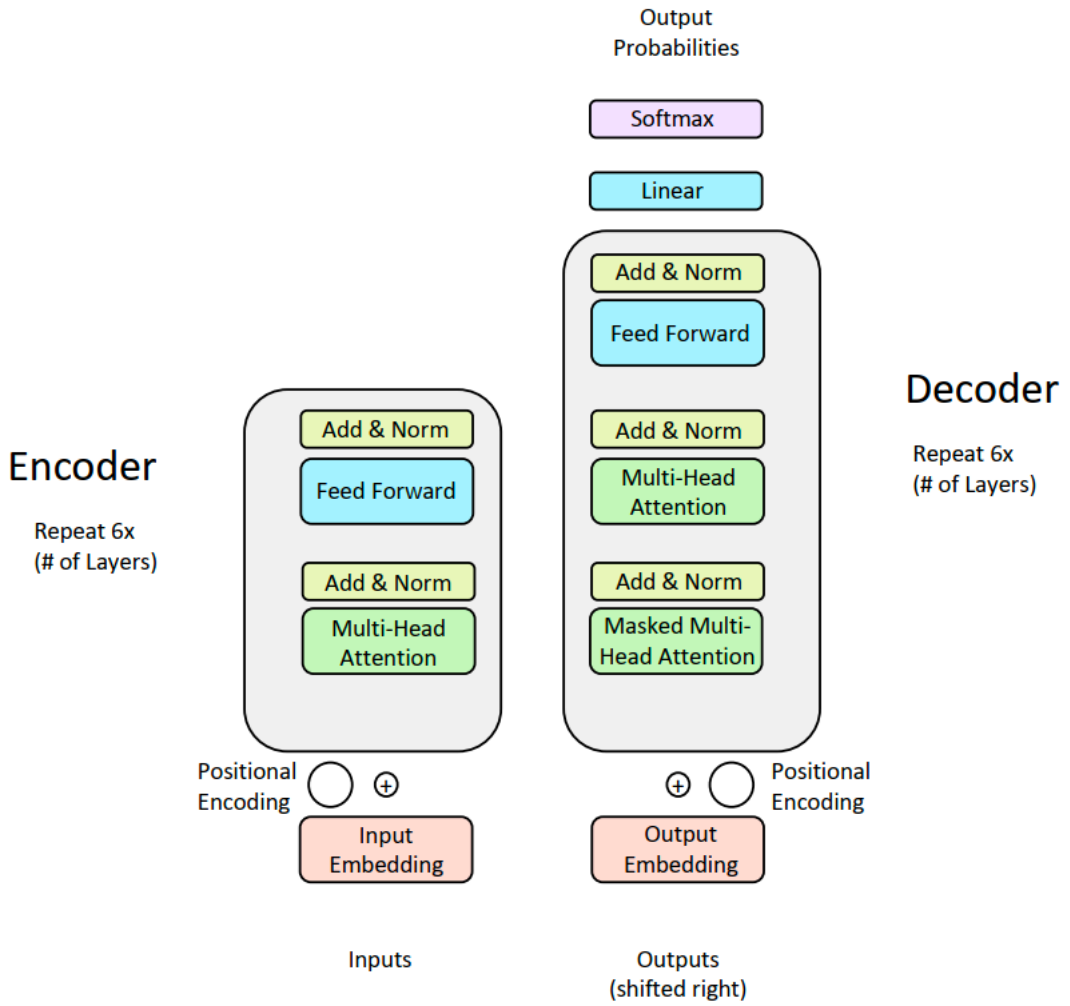
Maximum interact distance O(1)!

Contents

- Transformers
  - Impact of Transformers on NLP (and ML more broadly)
  - From Recurrence (RNNs) to Attention-Based NLP Models
  - Understanding the Transformer Model
  - Drawbacks and Variants of Transformers
- Pretraining Language Models(PLMs)
  - Subword modeling
  - Motivating model pretraining from word embeddings
  - Model pretraining three ways
    - Decoders
    - Encoders
    - Encoder-Decoders
  - Very large models and in-context learning

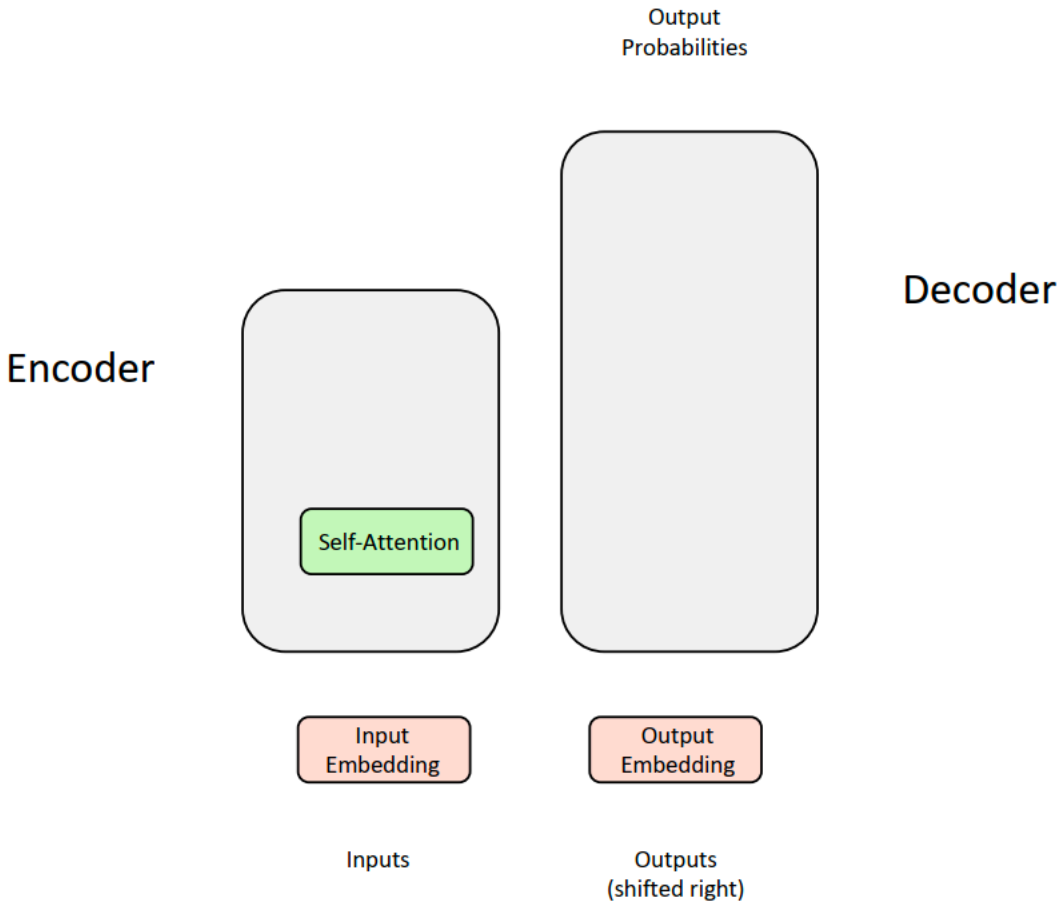
The Transformer Encoder-Decoder [Vaswani et al., 2017]

- In this section, you will learn exactly how the Transformer architecture works:
  - First, we will talk about the Encoder!
  - Next, we will go through the Decoder (which is quite similar)!



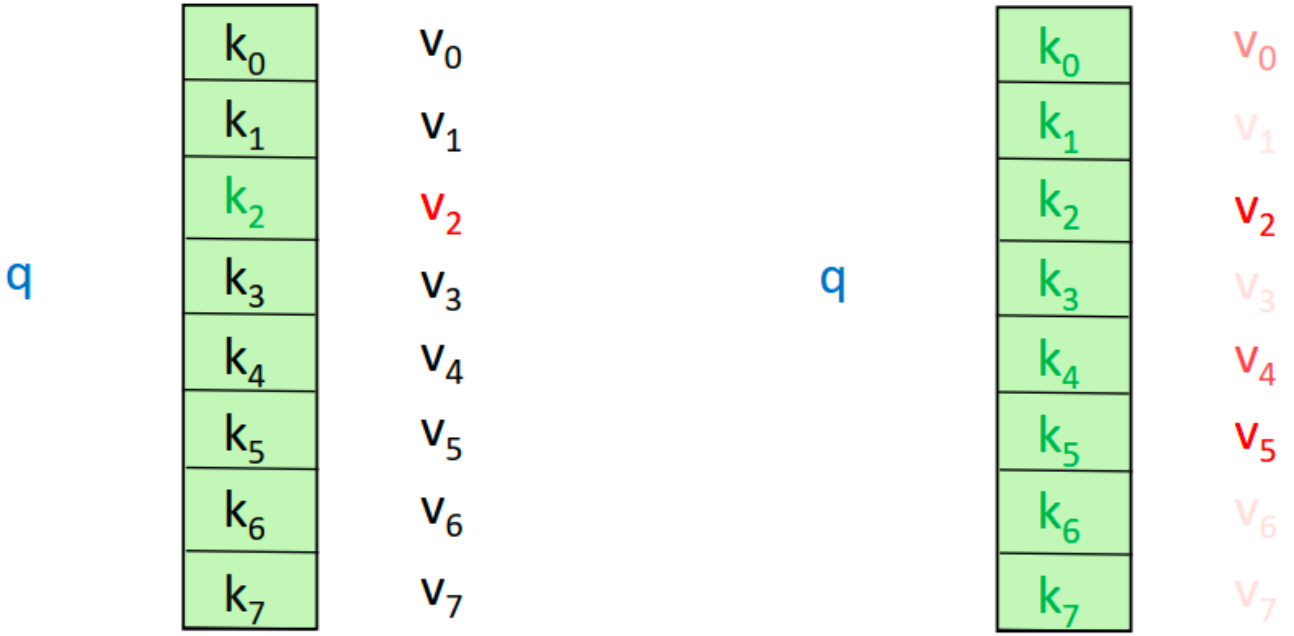
Encoder: Self-Attention

- Self-Attention is the core building block of Transformer, so let's first focus on that!



Intuition for Attention Mechanism

- Let's think of attention as a "fuzzy" or approximate hashtable:
  - To look up a value, we compare a query against keys in a table.
  - In a hashtable (shown on the bottom left):
    - Each query (hash) maps to exactly one key-value pair.
  - In (self-)attention (shown on the bottom right):
    - Each query matches each key to varying degrees.
    - We return a sum of values weighted by the query-key match



Recipe for Self-Attention in the Transformer Encoder

- Step 1: For each word , calculate its query, key, and value.

$$q_i = W^Q x_i \quad k_i = W^K x_i \quad v_i = W^V x_i$$

- Step 2: Calculate attention score between query and keys.

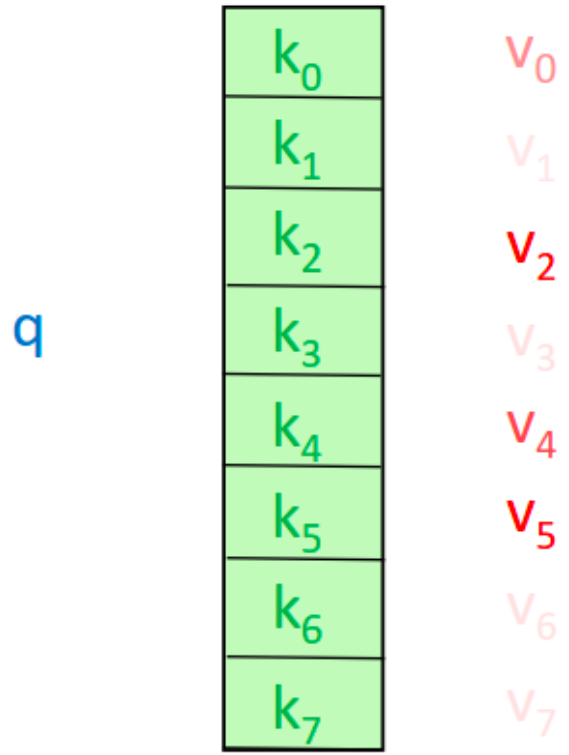
$$e_{ij} = q_i \cdot k_j$$

- Step 3: Take the softmax to normalize attention scores.

$$\alpha_{ij} = softmax(e_{ij}) = \frac{exp(e_{ij})}{\sum_k exp(e_{ik})}$$

- Step 4: Take a weighted sum of values

$$Output_i = \sum_j \alpha_{ij} v_j$$



Recipe for (Vectorized) Self-Attention in the Transformer Encoder

- Step 1: With embeddings stacked in X, calculate queries, keys, and values.

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

- Step 2: Calculate attention scores between query and keys.

$$E = QK^T$$

- Step 3: Take the softmax to normalize attention scores.

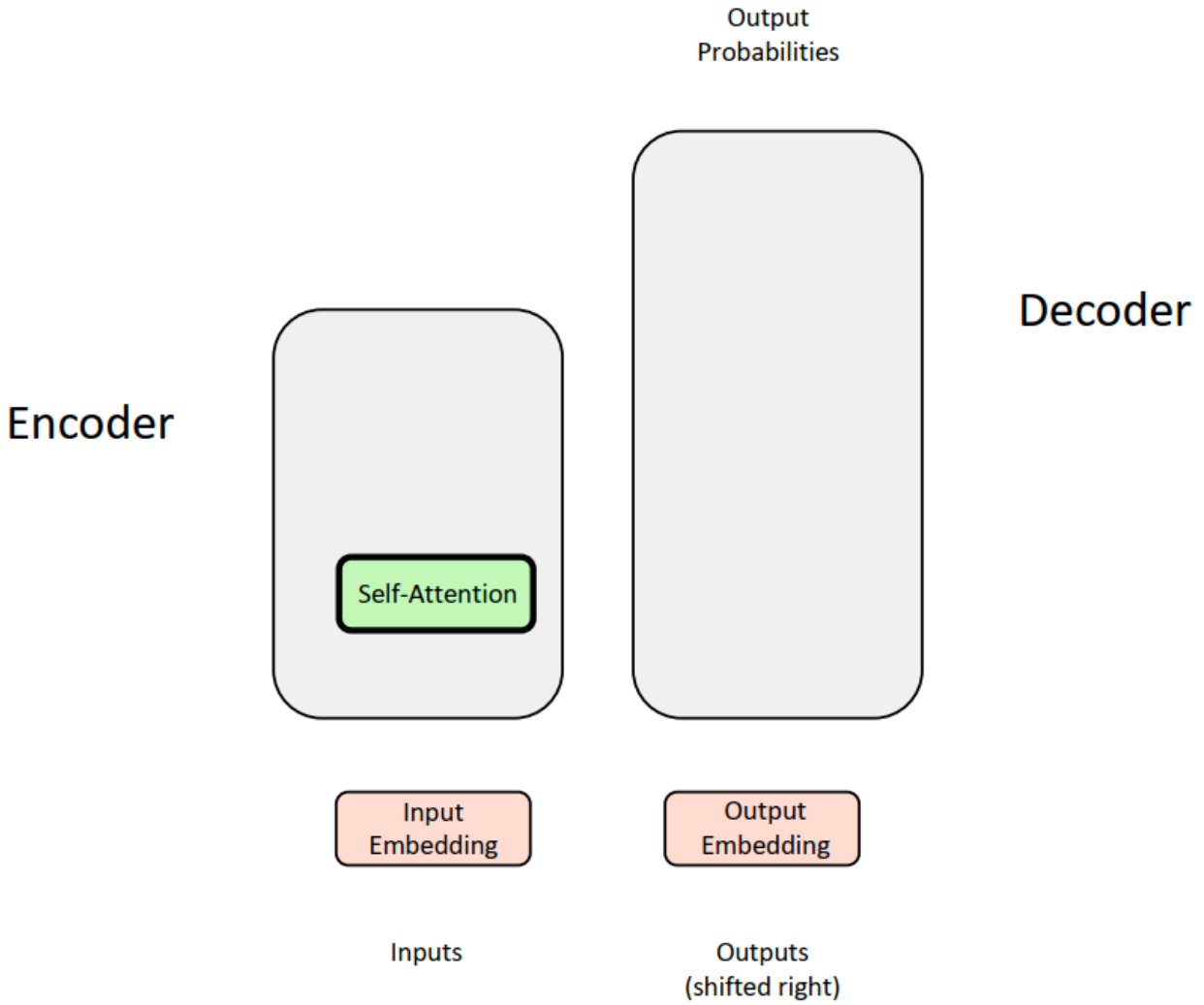
$$A = softmax(E)$$

- Step 4: Take a weighted sum of values

$$Output = AV$$

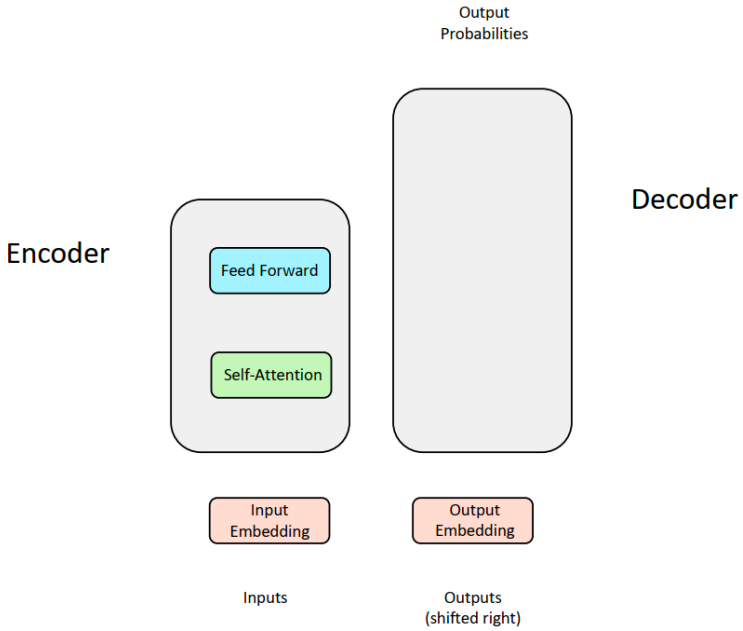
$$Output = softmax\left(QK^T\right)V$$

What We Have So Far: (Encoder) Self-Attention!



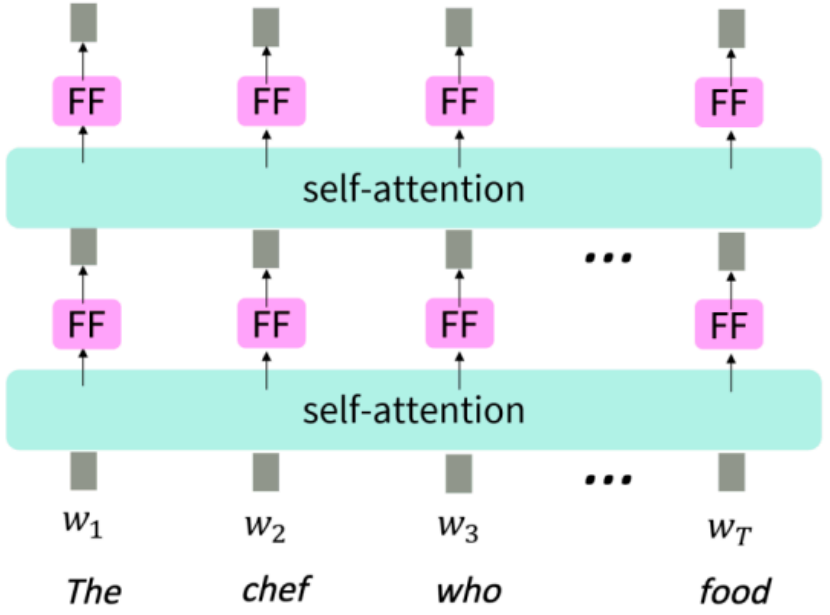
But attention isn't quite all you need!

- 问题：
  - self-attention就是value向量的重新加权平均
  - 没有任何非线性层
- 简单的修复方式
  - Self-attention后加一个feedforward层，加入非线性激活函数，以及额外的表达能力

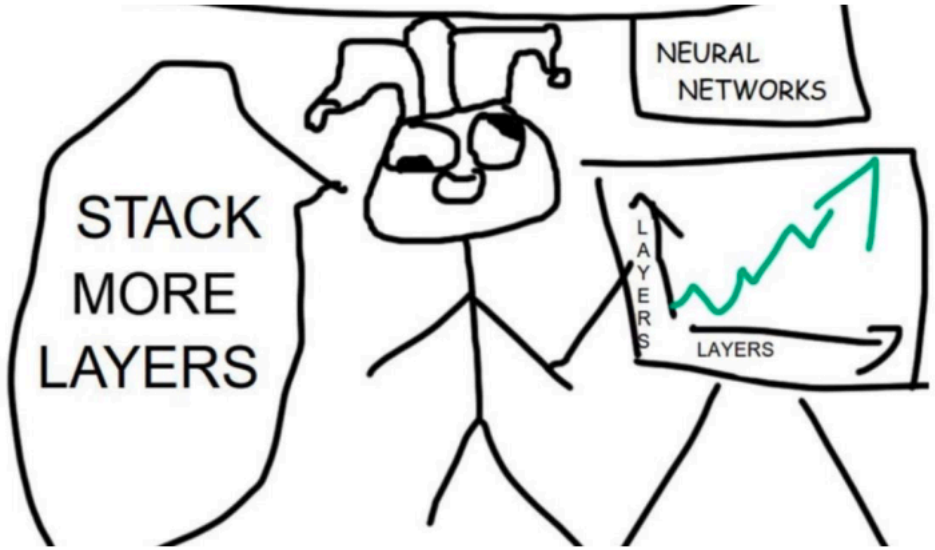


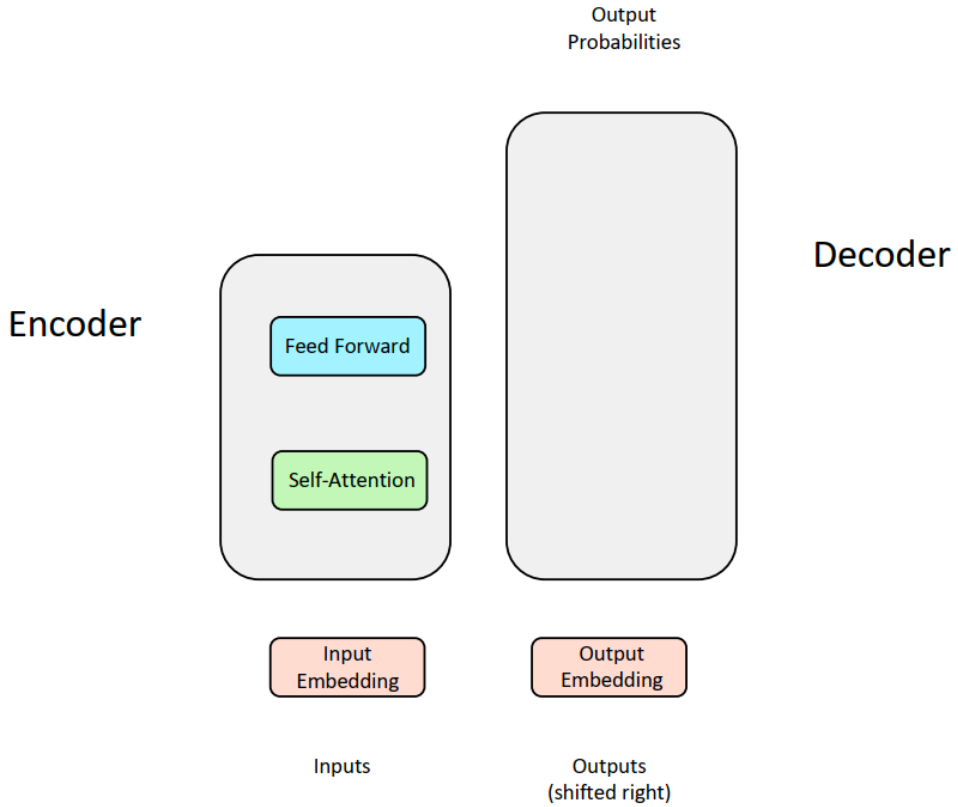
Equation for Feed Forward Layer

$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$



But how do we make this work for deep networks?



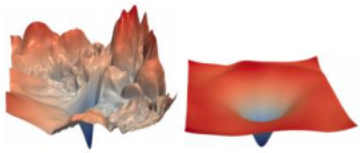


- Training Trick #1: Residual Connections
- Training Trick #2: LayerNorm
- Training Trick #3: Scaled Dot Product Attention

Training Trick #1: Residual Connections [He et al., 2016]

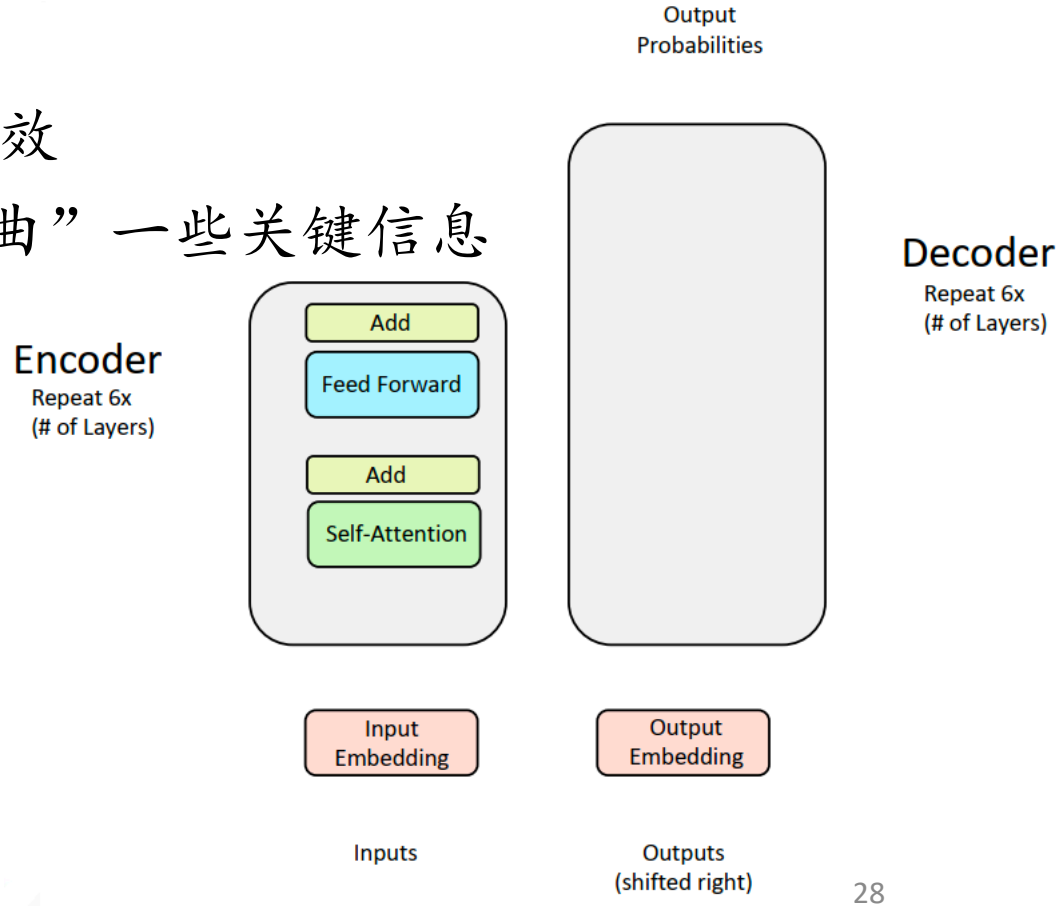
- Residual Connection：CV中的简单有效的方法
- 深度神经网络很难学习到identity function
- 直接将“raw” embeddings传到下一层比较有效
- 可以避免网络“遗忘”或经过很多层后“扭曲”一些关键信息

Residual connections are also thought to smooth the loss landscape and make training easier!



[no residuals] [residuals]

[Loss landscape visualization, Li et al., 2018, on a ResNet]

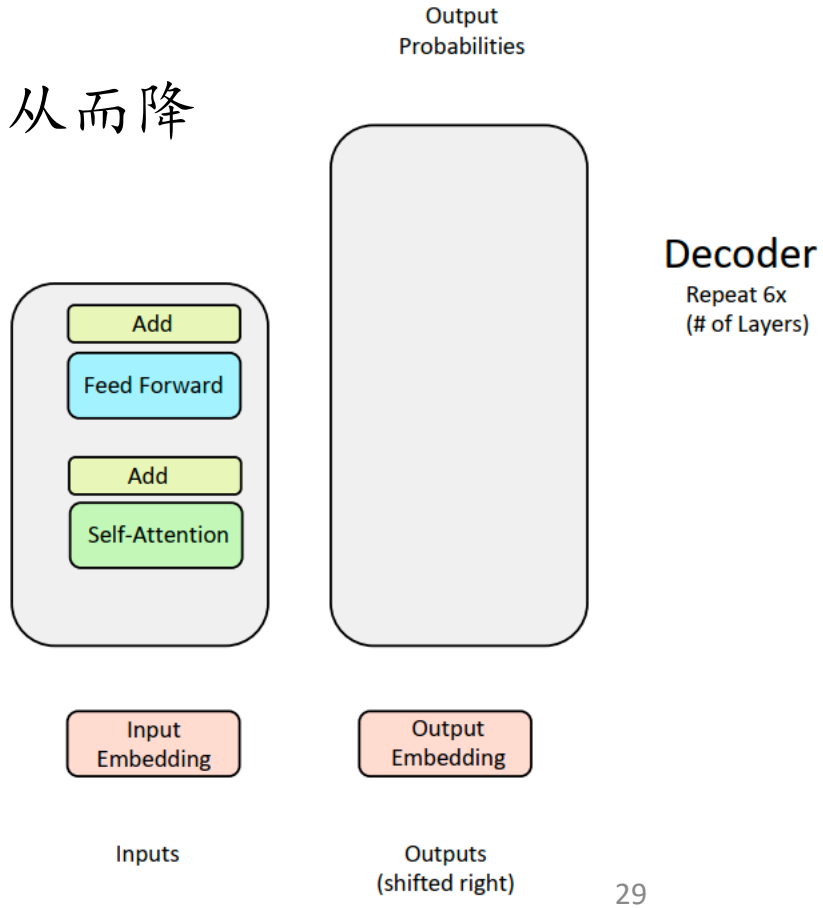


Training Trick #2: Layer Normalization [Ba et al., 2016]

- 问题：由于下游层的输出会不断飘移，所以上层的参数会难以训练
- 解决方案：每层之内归一化为均值为0，标准差为1。从而降低非新信息导致的偏移。

Mean: $\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l$ Standard Deviation: $\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$

$$x^{\ell\prime} = \frac{x^\ell - \mu^\ell}{\sigma^\ell + \epsilon}$$

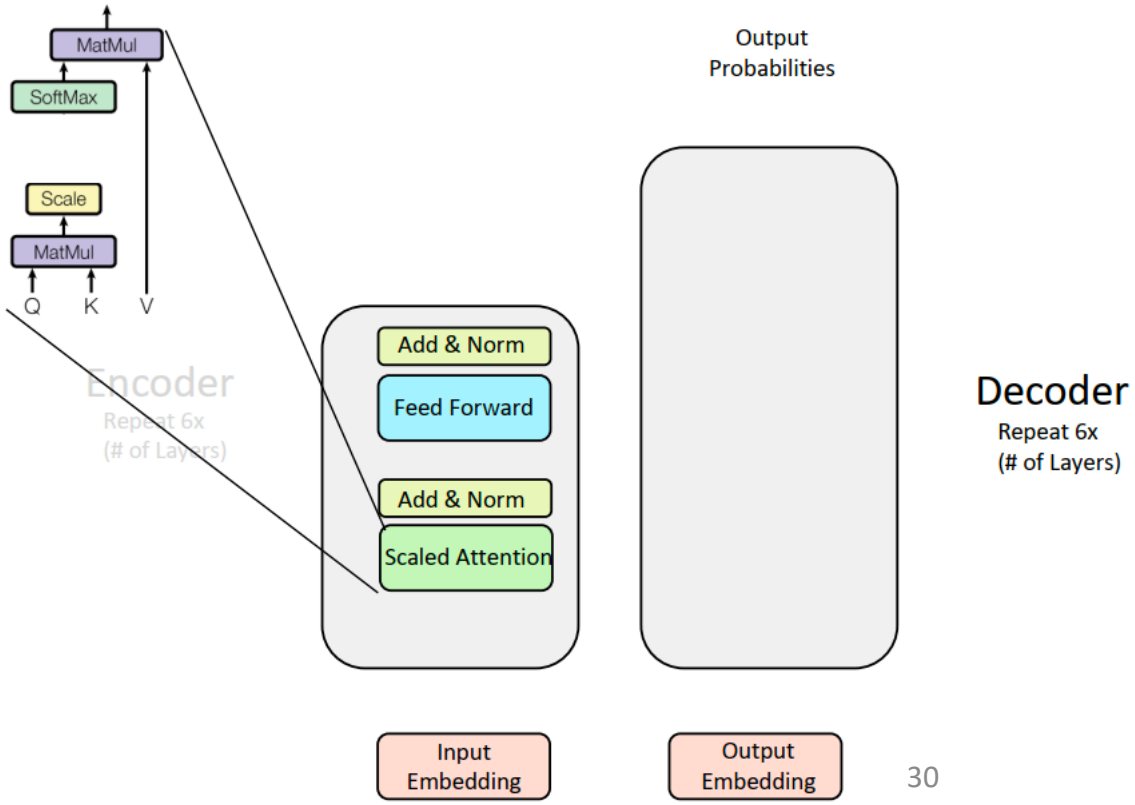


Training Trick #3: Scaled Dot Product Attention

- LayerNorm之后，向量的均值方差被归为了0和1
- 然而，点积会给出极端值，向量越长，越容易出现。方差会因此

Quick Statistics Review:

- Mean of sum = sum of means = $d_k * 0 = 0$
- Variance of sum = sum of variances = $d_k * 1 = d_k$
- To set the variance to 1, simply divide by $\sqrt{d_k}$!



Updated Self-Attention Equation:

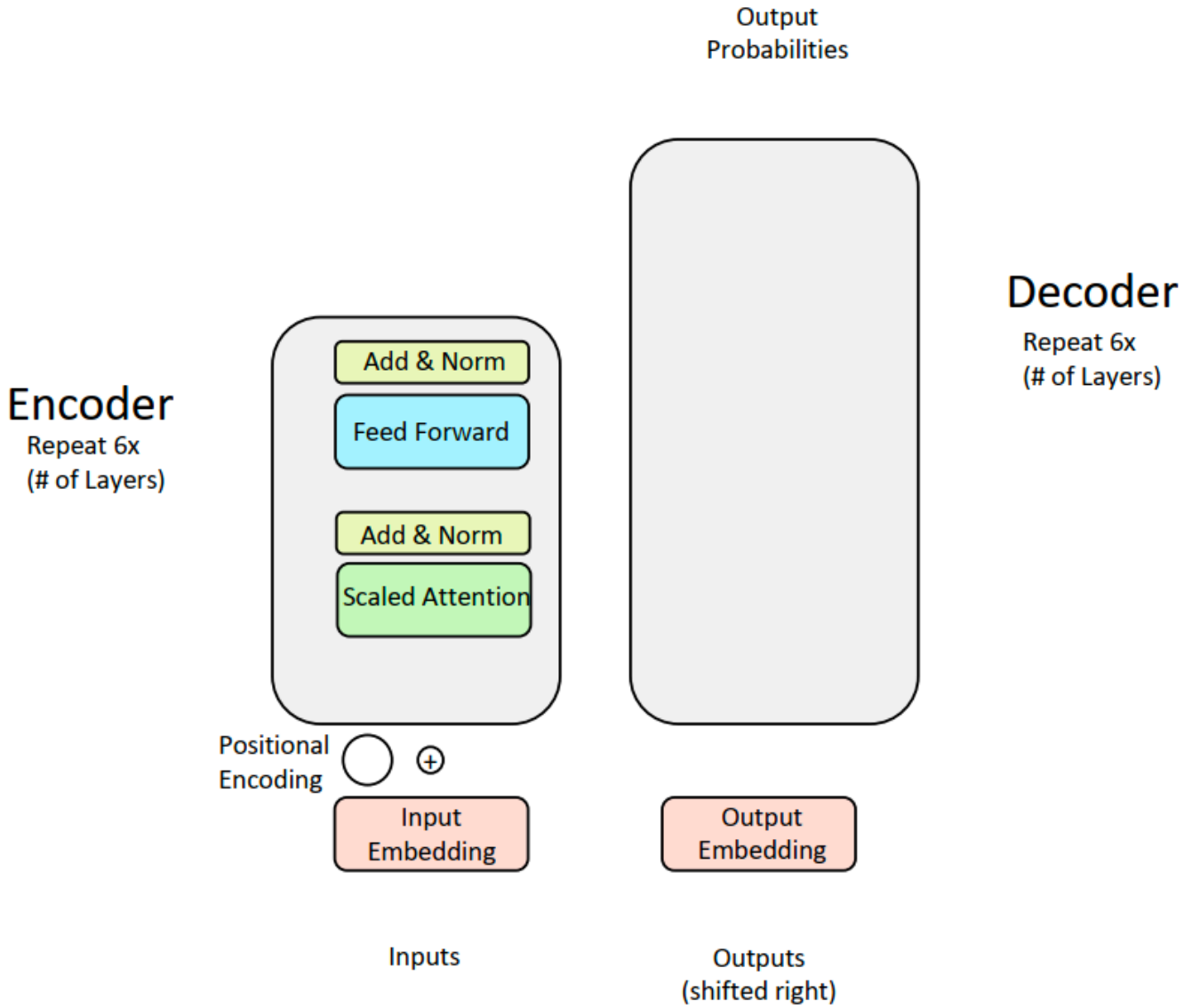
$$Output = softmax\left(QK^T / \sqrt{d_k}\right)V$$

Major issue!

- Encoder我们介绍的差不多了，但现在还有一个重要的问题，有人发现吗？
- 考虑这个句子
  - "Man eats small dinosaur."
- 用Transformer的话，词语顺序并不影响神经网络计算
- 词语顺序在很多语言中都很重要。所以此处问题很大。

$$Output = softmax\left(QK^T / \sqrt{d_k}\right)V$$

Solution: Inject Order Information through Positional Encodings!



Fixing the first self-attention problem: sequence order

- 由于self-attention无法利用到顺序信息，我们需要把顺序建模进keys, queries, and values中
- 考虑把index表示成向量

$$p_i \in \mathbb{R}^d, \text{ for } i \in \{1,2,\ldots,T\} \text{ are position vectors}$$

- 暂时先不考虑 p_i 怎么算的
- 融合进self-attention很容易：简单的加在一起
- 令 $\tilde{v}_i\, \tilde{k}_i, \tilde{q}_i$ 是旧的values, keys, and queries

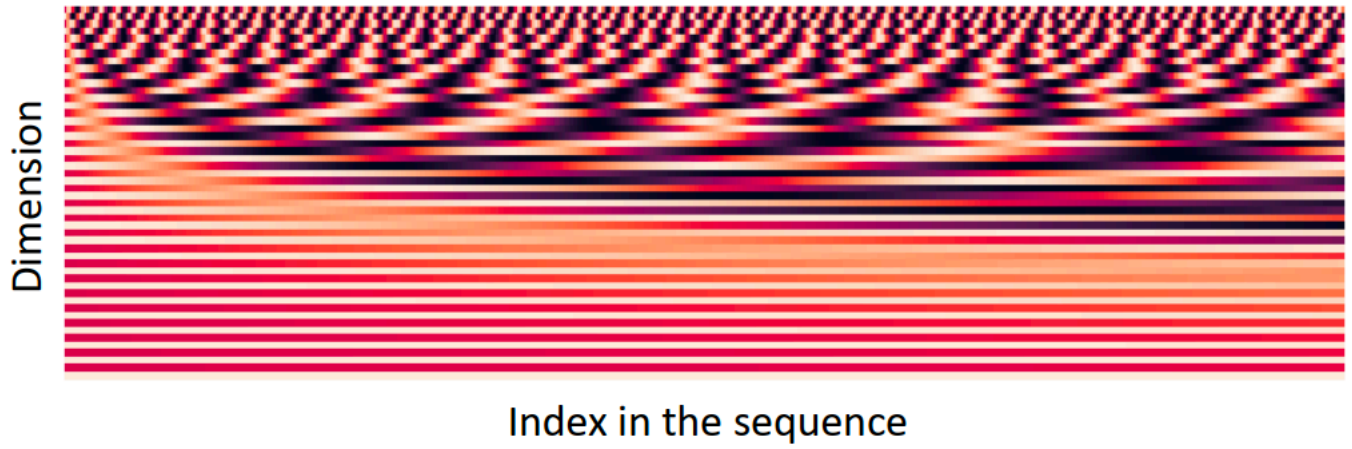
$$v_i = \tilde{v}_i + p_i$$
$$q_i = \tilde{q}_i + p_i$$
$$k_i = \tilde{k}_i + p_i$$

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add…

Position representation vectors through sinusoids

- Sinusoidal position representations: concatenate sinusoidal functions of varying periods:

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



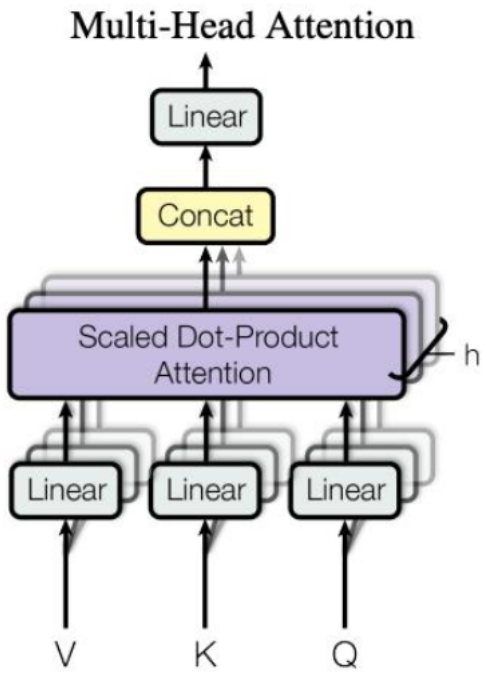
- 优点：（1）周期性暗指绝对位置不是很重要（2）可以扩展到长序列
- 缺点：（1）不可学习（2）“扩展”不一定很有用

Position representation vectors learned from scratch

- 学习一个绝对的位置表示：令所有的 p_i 可学习
  - 学习一个矩阵 $p \in \mathbb{R}^{d \times T}$，令 p_i 是该矩阵的某一列
- 优点：
  - 灵活，每个位置的表示可以从数据中得到训练
- 缺点：
  - 无法扩展，超出1~T的范围无法表示
- 目前很多模型用的是这种方法
- 其他灵活的位置向量方法
  - Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]

Multi-Headed Self-Attention: k heads are better than 1!

- **High-Level Idea**: Let's perform self-attention multiple times in parallel and combine the results



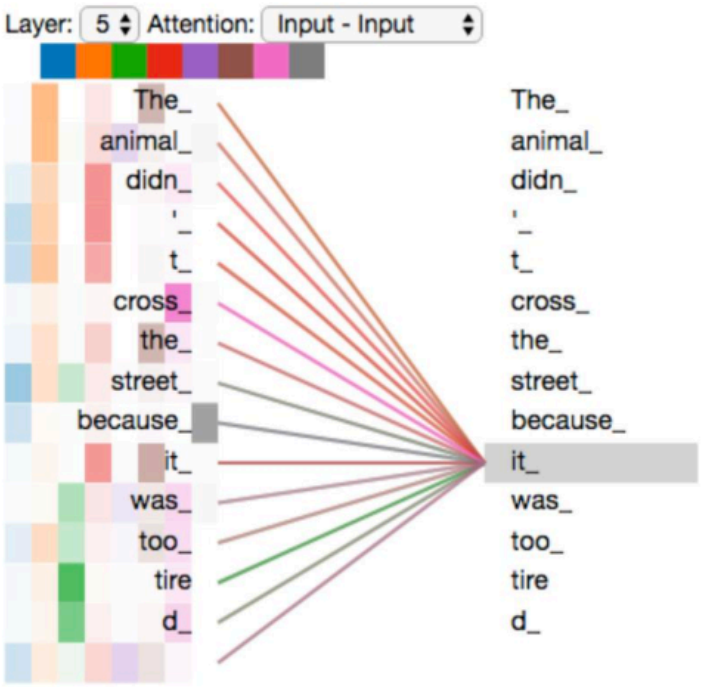
[Vaswani et al. 2017]

Wizards of the Coast, Artist: Todd Lockwood

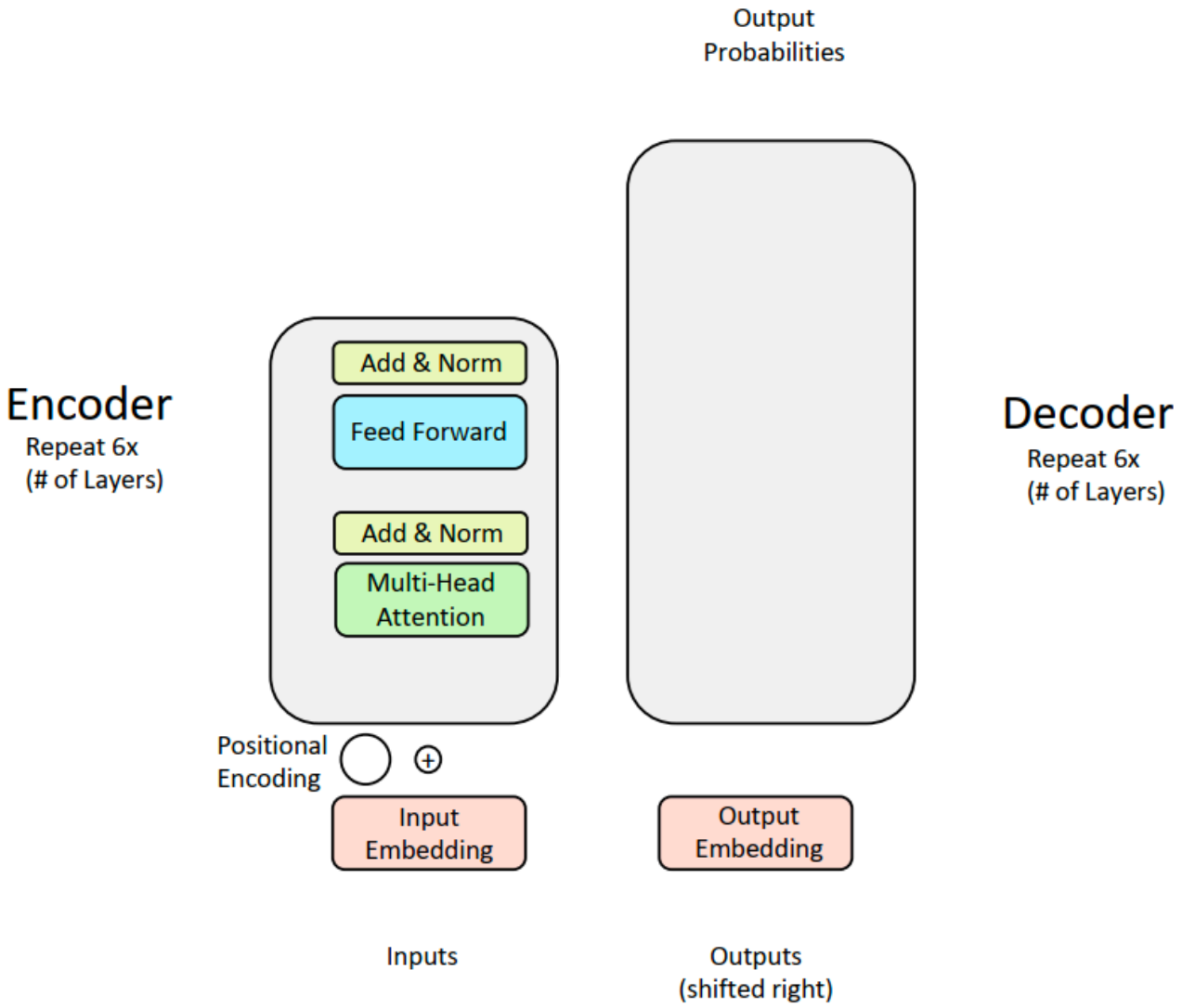
The Transformer Encoder: **Multi-headed Self-Attention**

- 如果我们想同时关注句子中的多个位置呢？
  - 对于词i，self-attention关注的是 $x_i^\top Q^\top K x_j$ 最大的点。如果我们由于其他原因还想关注别的点呢？
- 通过多组Q,K,V来定义多个注意力头
- 令 $Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times \frac{d}{h}}$，h是注意力头的数量，l=1…h
- 每个注意力头的计算过程都是一样的
- 最后将每个头的计算结果拼接起来

$$\text{output} = Y[\text{output}_1; \ldots; \text{output}_h], \text{where } Y \in \mathbb{R}^{d \times d}$$



Yay, we've completed the Encoder! Time for the Decoder...



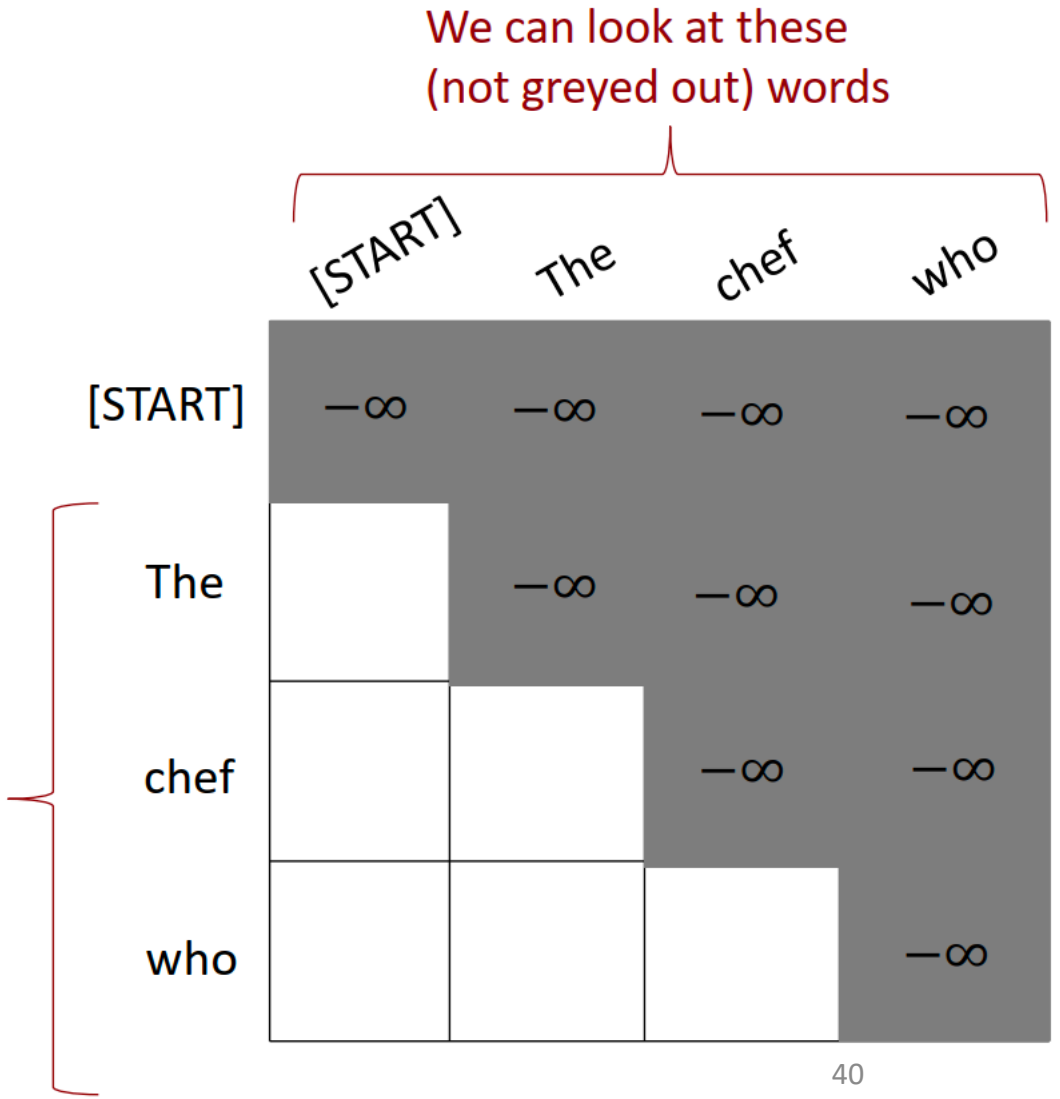
Decoder: Masked Multi-Head Self-Attention

- 问题：
  - Self-attention 可以同时看到所有信息
  - Decode阶段会不会直接看到答案？
- 解答：
  - 将Multi-Head Attention 的未来信息mask掉

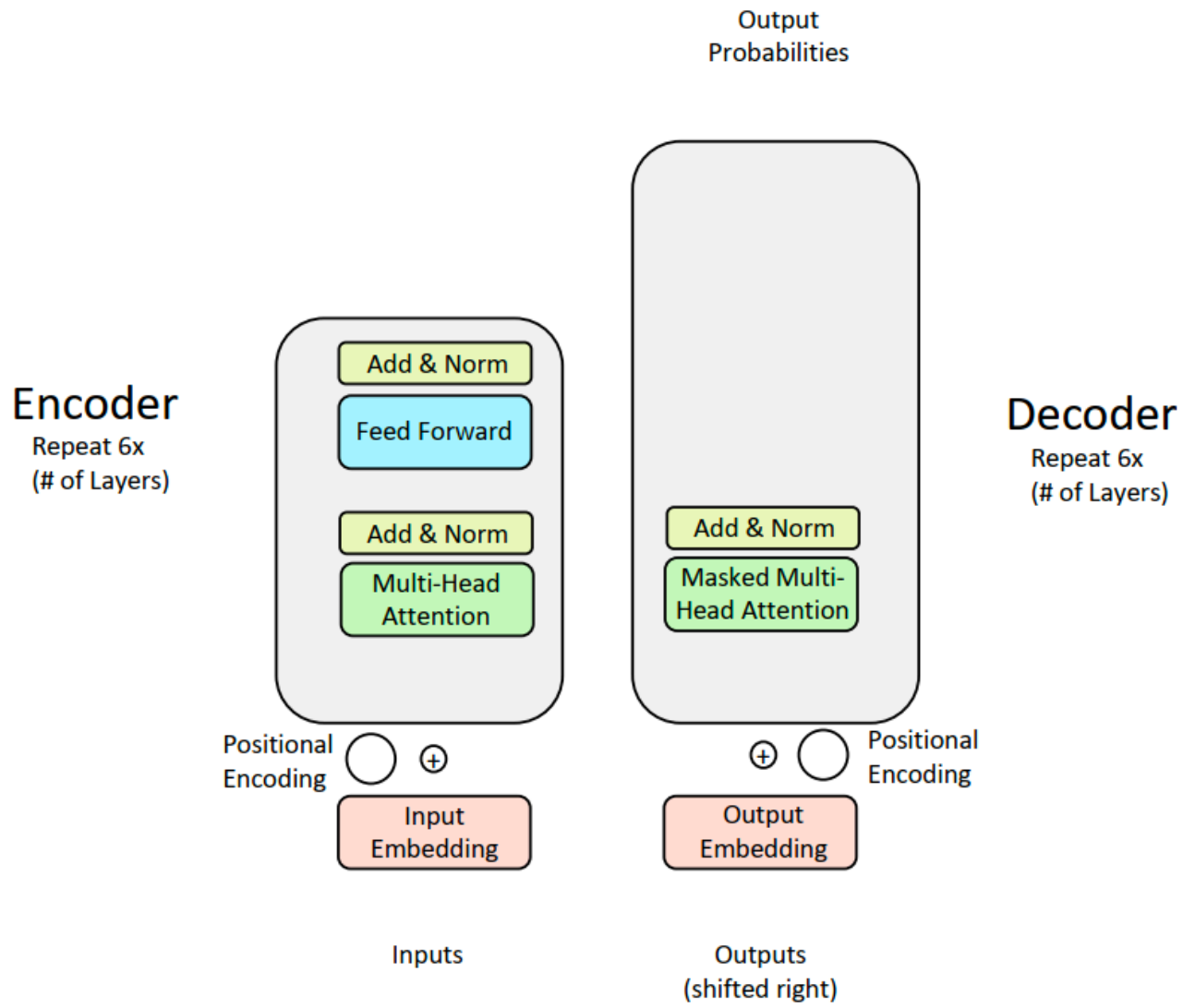
Masking the future in self-attention

- To use self-attention in decoders, we need to ensure we can't peek at the future.
- To enable parallelization, we mask out attention to future words by setting attention scores to $-\infty$

$$e_{ij} = \begin{cases} q_i^\top k_j, j < i \\ -\infty, j \geq i \end{cases}$$



Decoder: Masked Multi-Headed Self-Attention



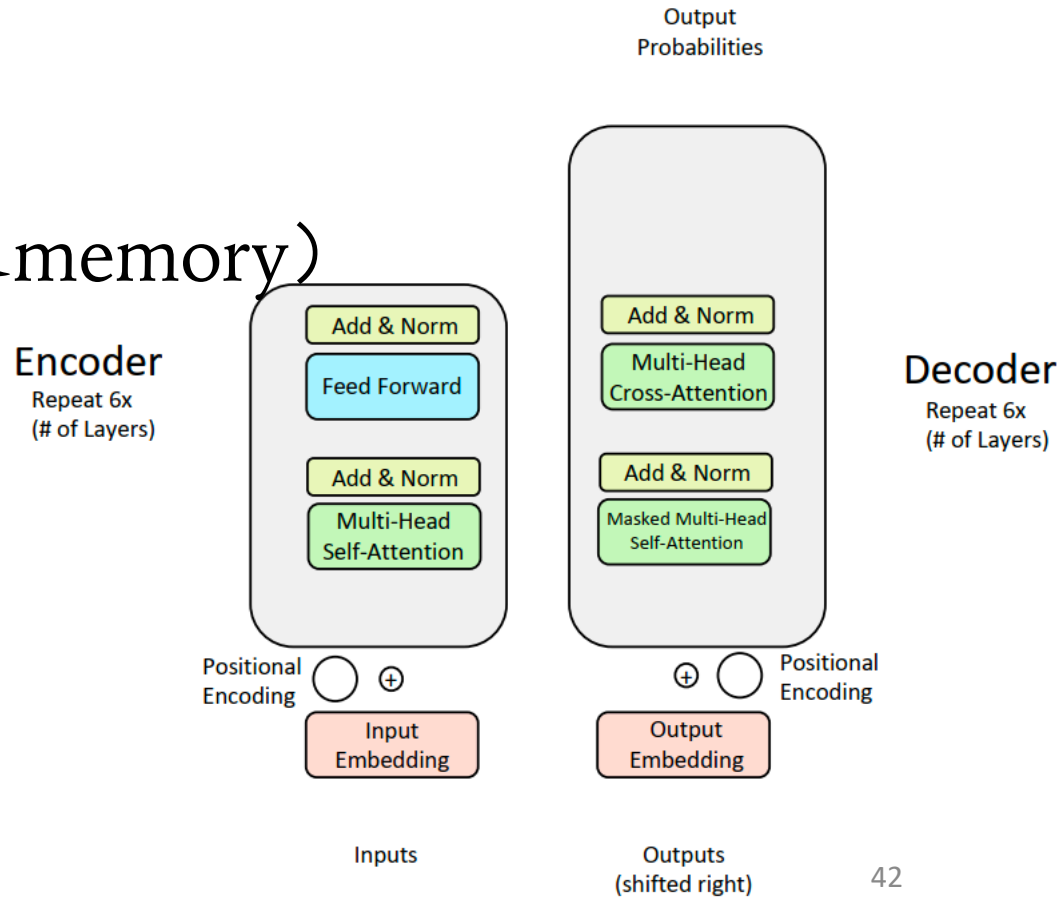
Encoder-Decoder Attention

- Self-attention中， keys, queries, values都是同源的
- 令 $h_1, \ldots, h_T$ 是encoder的输出向量
- 令 $z_1, \ldots, z_T$ 是decoder的输入向量
- Key和value是从encoder中提取的（像memory）

$$k_i = Kh_i, v_i = Vh_i$$

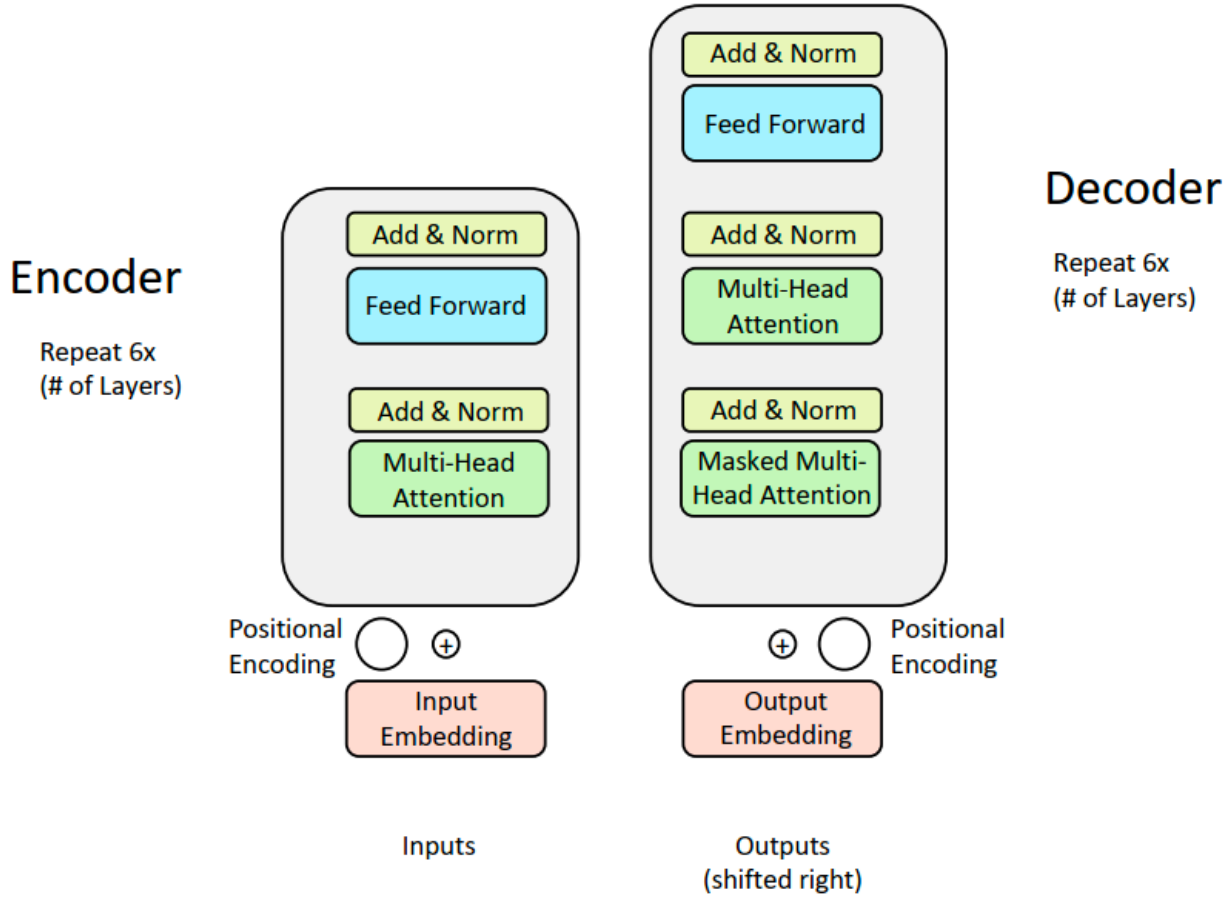
- Query是从decoder中来的

$$q_i = Qz_i$$



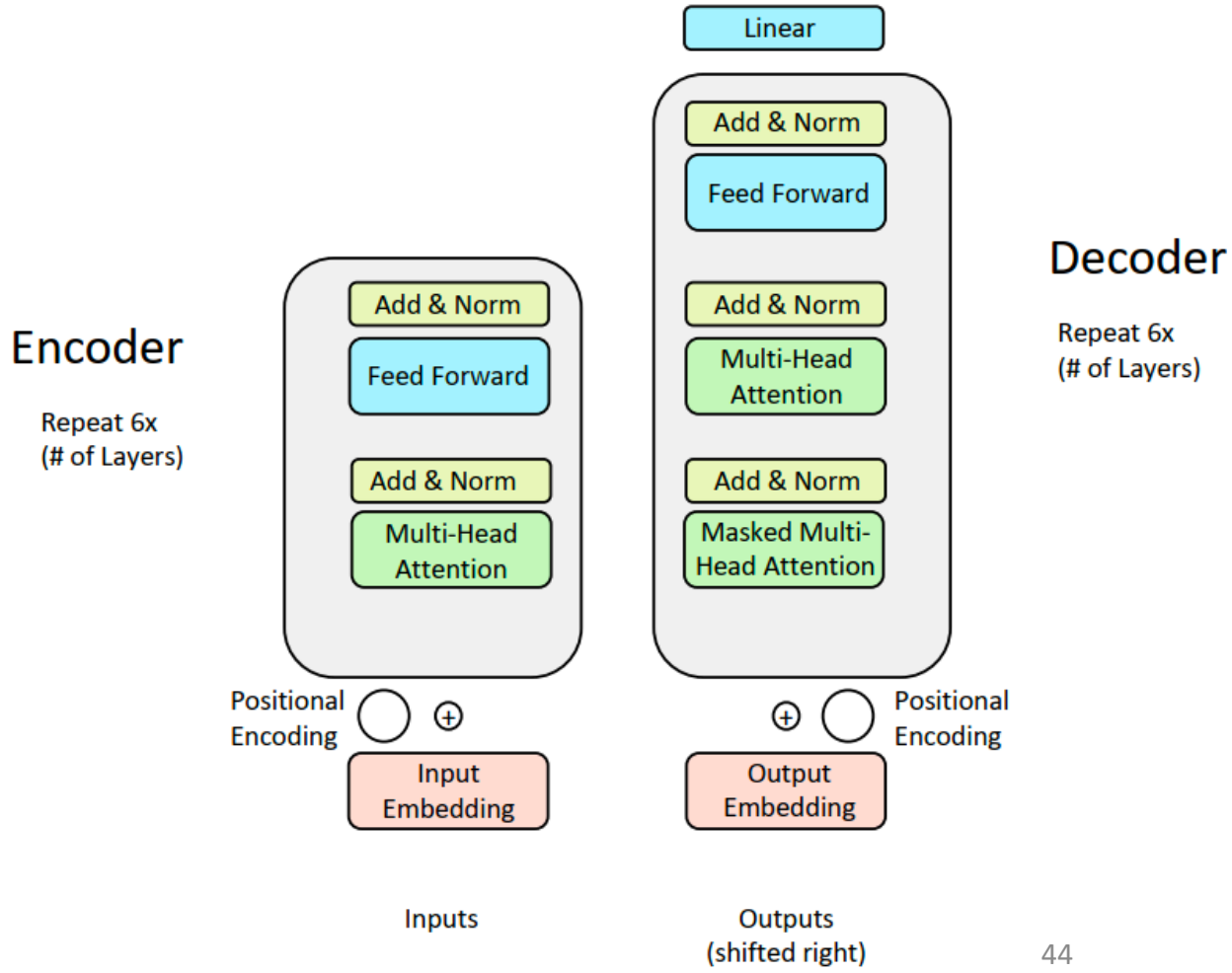
Decoder: Finishing touches!

- Add a feed forward layer (with residual connections and layer norm)



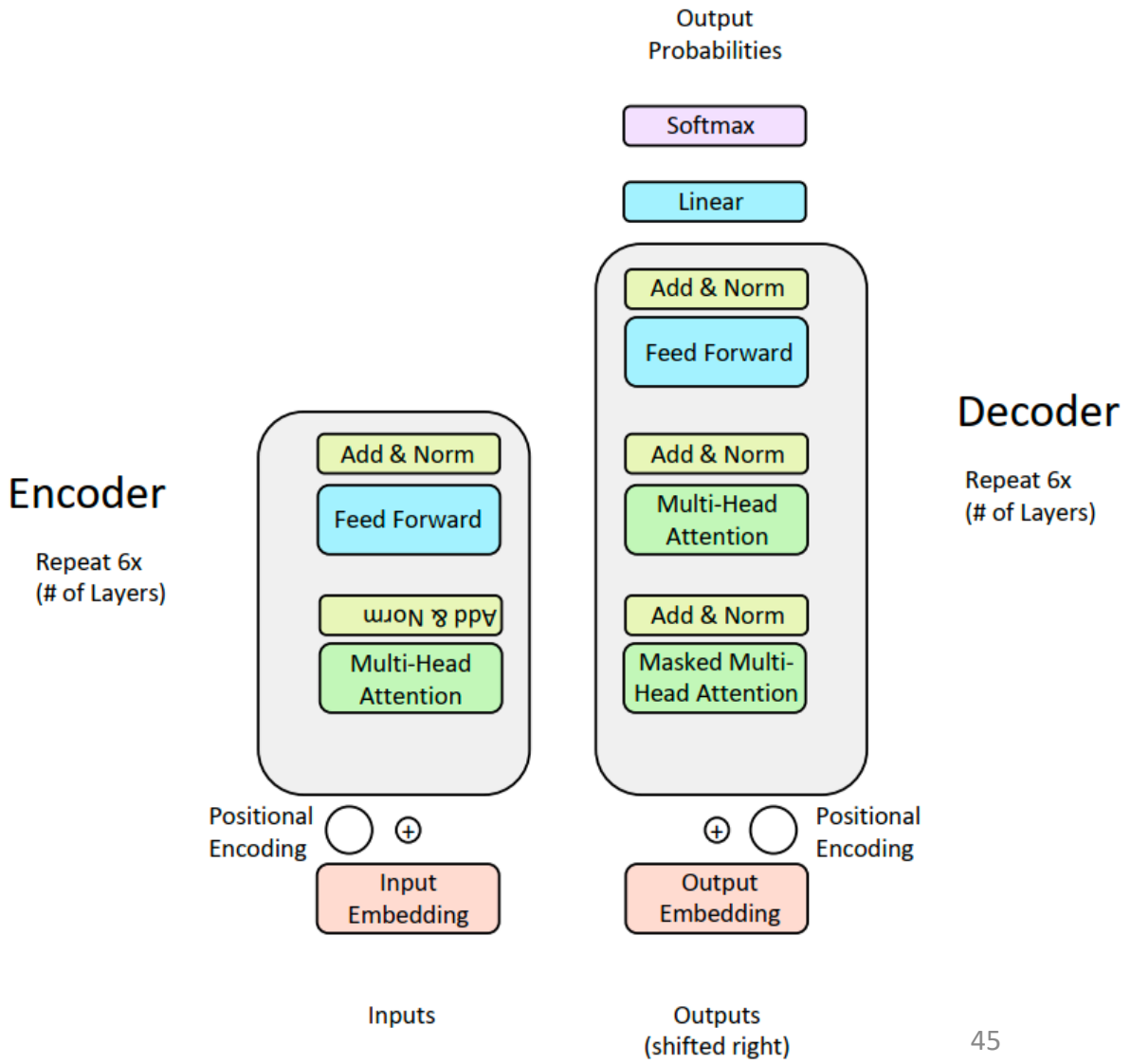
Decoder: Finishing touches!

- Add a feed forward layer (with residual connections and layer norm)
- Add a final linear layer to project the embeddings into a much longer vector of length vocab size (logits)

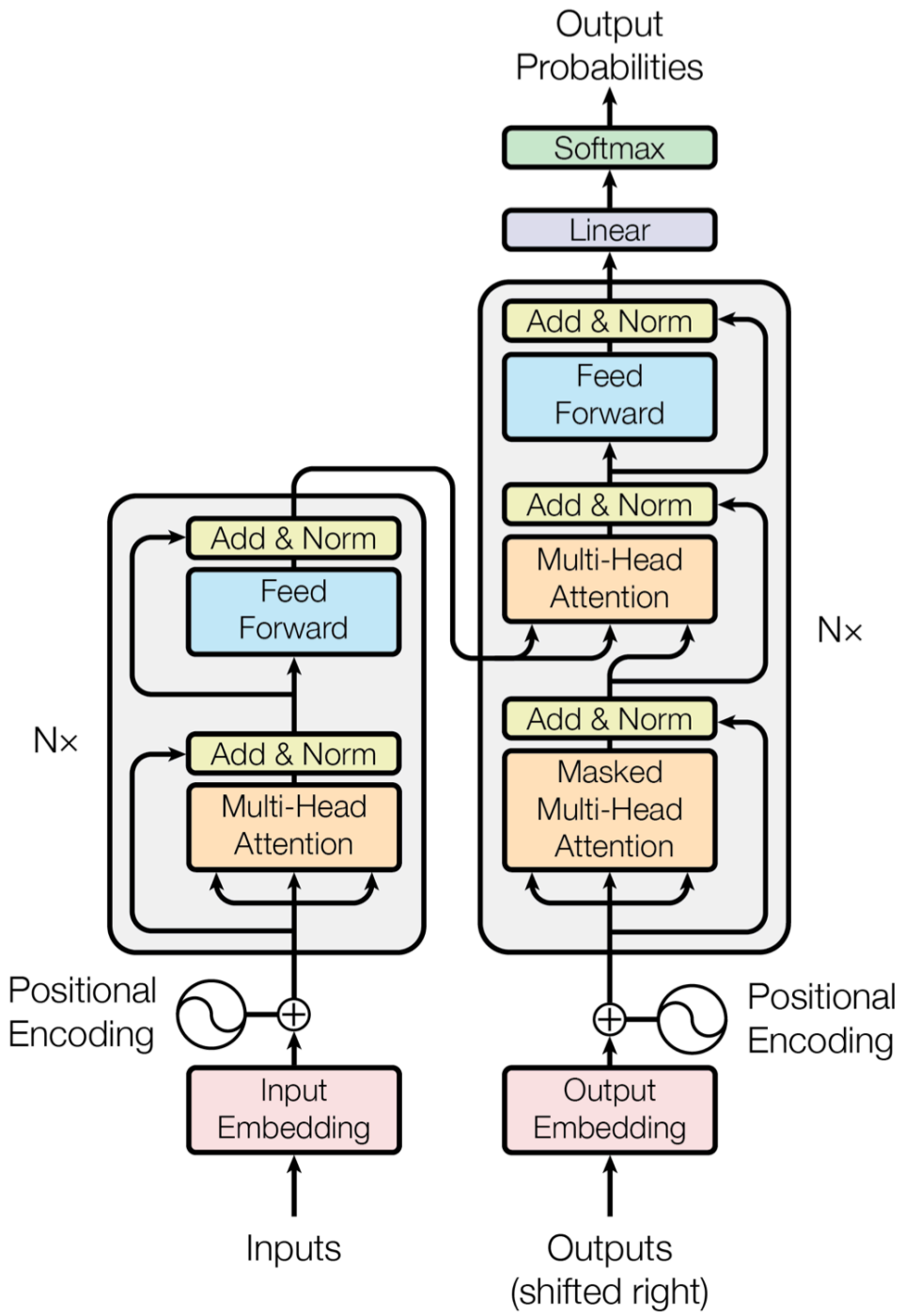


Decoder: Finishing touches!

- Add a feed forward layer (with residual connections and layer norm)
- Add a final linear layer to project the embeddings into a much longer vector of length vocab size (logits)
- Add a final softmax to generate an probability distribution of possible next words!



Recap of Transformer Architecture



Contents

- Transformers
  - Impact of Transformers on NLP (and ML more broadly)
  - From Recurrence (RNNs) to Attention-Based NLP Models
  - Understanding the Transformer Model
  - Drawbacks and Variants of Transformers
- Pretraining Language Models(PLMs)
  - Subword modeling
  - Motivating model pretraining from word embeddings
  - Model pretraining three ways
    - Decoders
    - Encoders
    - Encoder-Decoders
  - Very large models and in-context learning

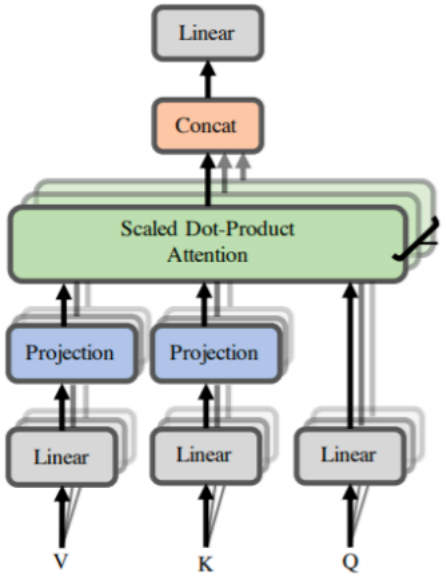
What would we like to fix about the Transformer?

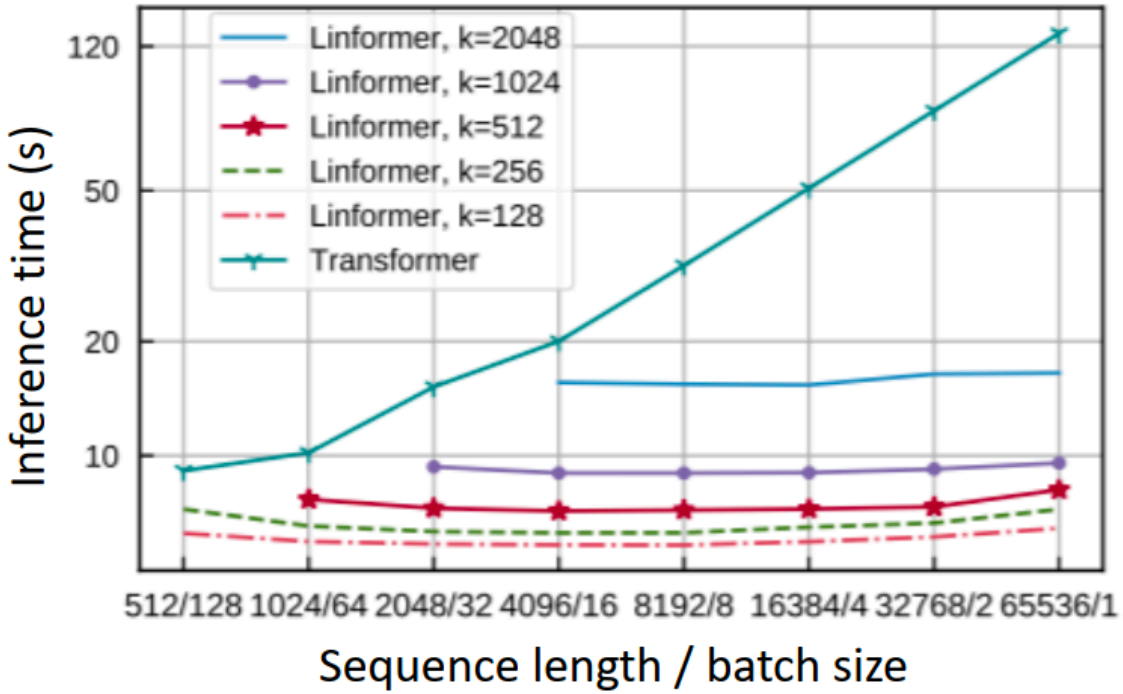
- Self-attention的计算是平方复杂度的
  - Recurrent模型仅仅是线性复杂度
- 位置表示
  - 绝对位置的可学习的向量是我们的最好选择吗？
  - Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]

Recent work on improving on quadratic self-attention cost

- 很多目前的工作都要解决self-attention的$O(T^2)$计算量问题
- 比如，Linformer [Wang et al., 2020]

Key idea: map the sequence length dimension to a lower-dimensional space for values, keys

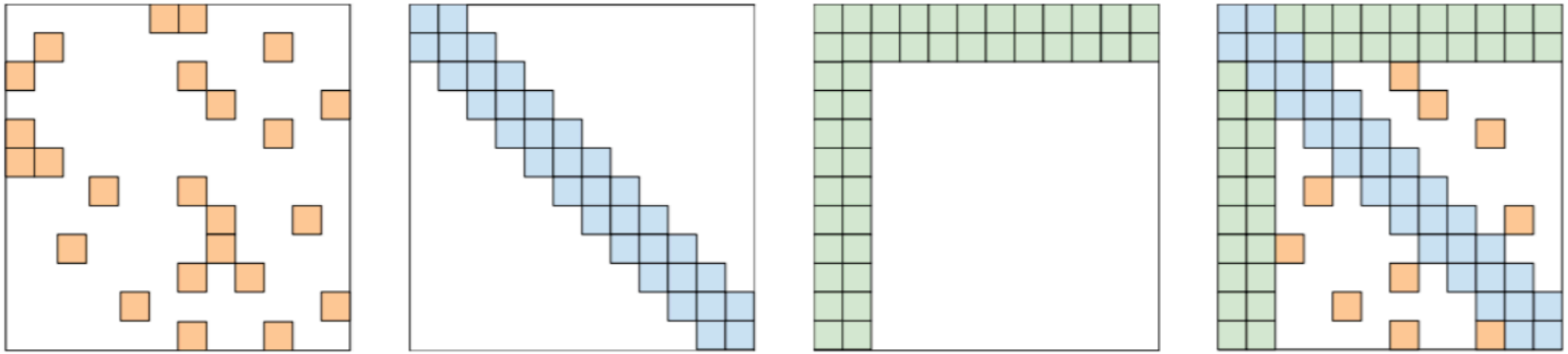




Recent work on improving on quadratic self-attention cost

- 很多目前的工作都要解决self-attention的$O(T^2)$计算量问题
- 比如，BigBird [Zaheer et al., 2021]

Key idea: replace all-pairs interactions with a family of other interactions, **like local windows**, **looking at everything**, and **random interactions**.



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

Do Transformer Modifications Transfer?

- "Surprisingly, we find that most modifications do not meaningfully improve performance."

Model	Params	Ops	Step/s	Early loss	Final loss	SGLUE	XSum	WebQ	WMT EnDe
Vanilla Transformer	223M	11.1T	3.50	2.182 ± 0.005	1.838	71.66	17.78	23.02	26.62
GeLU	223M	11.1T	3.58	2.179 ± 0.003	1.838	**75.79**	**17.86**	**25.13**	26.47
Swish	223M	11.1T	3.62	2.186 ± 0.003	1.847	**73.77**	17.74	**24.34**	**26.75**
ELU	223M	11.1T	3.56	2.270 ± 0.007	1.932	67.83	16.73	23.02	26.08
GLU	223M	11.1T	3.59	2.174 ± 0.003	**1.814**	**74.20**	**17.42**	24.34	**27.12**
GeGLU	223M	11.1T	3.55	2.130 ± 0.006	**1.792**	**75.96**	**18.27**	**24.87**	**26.87**
ReGLU	223M	11.1T	3.57	2.145 ± 0.004	**1.803**	**76.17**	**18.36**	**24.87**	**27.02**
SeLU	223M	11.1T	3.55	2.315 ± 0.004	1.948	68.76	16.76	22.75	25.99
SwiGLU	223M	11.1T	3.53	2.127 ± 0.003	**1.789**	**76.00**	**18.20**	**24.34**	**27.02**
LiGLU	223M	11.1T	3.59	2.149 ± 0.005	**1.798**	**75.34**	**17.97**	**24.34**	26.53
Sigmoid	223M	11.1T	3.63	2.291 ± 0.019	1.867	**74.31**	17.51	23.02	26.30
Softplus	223M	11.1T	3.47	2.207 ± 0.011	1.850	**72.45**	17.65	**24.34**	**26.89**
RMS Norm	223M	11.1T	3.68	2.167 ± 0.008	**1.821**	**75.45**	**17.94**	**24.07**	**27.14**
Rezero	223M	11.1T	3.51	2.262 ± 0.003	1.939	61.69	15.64	20.90	26.37
Rezero + LayerNorm	223M	11.1T	3.26	2.223 ± 0.006	1.858	70.42	17.58	23.02	26.29
Rezero + RMS Norm	223M	11.1T	3.34	2.221 ± 0.009	1.875	70.33	17.32	23.02	26.19
Fixup	223M	11.1T	2.95	2.382 ± 0.012	2.067	58.56	14.42	23.02	26.31
24 layers, $d_{ff} = 1536$, $H = 6$	224M	11.1T	3.33	2.200 ± 0.007	1.843	**74.89**	17.75	**25.13**	**26.89**
18 layers, $d_{ff} = 2048$, $H = 8$	223M	11.1T	3.38	2.185 ± 0.005	**1.831**	**76.45**	16.83	**24.34**	**27.10**
8 layers, $d_{ff} = 4608$, $H = 18$	223M	11.1T	3.69	2.190 ± 0.005	1.847	**74.58**	17.69	**23.28**	**26.85**
6 layers, $d_{ff} = 6144$, $H = 24$	223M	11.1T	3.70	2.201 ± 0.010	1.857	**73.55**	17.59	**24.60**	**26.66**
Block sharing	65M	11.1T	3.91	2.497 ± 0.037	2.164	64.50	14.53	21.96	25.48
+ Factorized embeddings	45M	9.4T	4.21	2.631 ± 0.305	2.183	60.84	14.00	19.84	25.27
+ Factorized & shared embeddings	20M	9.1T	4.37	2.907 ± 0.313	2.385	53.95	11.37	19.84	25.19
Encoder only block sharing	170M	11.1T	3.68	2.298 ± 0.023	1.929	69.60	16.23	23.02	26.23
Decoder only block sharing	144M	11.1T	3.70	2.352 ± 0.029	2.082	67.93	16.13	**23.81**	26.08
Factorized Embedding	227M	9.4T	3.80	2.208 ± 0.006	1.855	70.41	15.92	22.75	26.50
Factorized & shared embeddings	202M	9.1T	3.92	2.320 ± 0.010	1.952	68.69	16.33	22.22	26.44
Tied encoder/decoder input embeddings	248M	11.1T	3.55	2.192 ± 0.002	1.840	**71.70**	17.72	**24.34**	26.49
Tied decoder input and output embeddings	248M	11.1T	3.57	2.187 ± 0.007	**1.827**	**74.86**	17.74	**24.87**	**26.67**
Untied embeddings	273M	11.1T	3.53	2.195 ± 0.005	**1.834**	**72.99**	17.58	**23.28**	26.48
Adaptive input embeddings	204M	9.2T	3.55	2.250 ± 0.002	1.899	66.57	16.21	**24.07**	**26.66**
Adaptive softmax	204M	9.2T	3.60	2.364 ± 0.005	1.982	**72.91**	16.67	21.16	25.56
Adaptive softmax without projection	223M	10.8T	3.43	2.229 ± 0.009	1.914	**71.82**	17.10	23.02	25.72
Mixture of softmaxes	232M	16.3T	2.24	2.227 ± 0.017	**1.821**	**76.77**	17.62	22.75	**26.82**
Transparent attention	223M	11.1T	3.33	2.181 ± 0.014	1.874	54.31	10.40	21.16	**26.80**
Dynamic convolution	257M	11.8T	2.65	2.403 ± 0.009	2.047	58.30	12.67	21.16	17.03
Lightweight convolution	224M	10.4T	4.07	2.370 ± 0.010	1.989	63.07	14.86	23.02	24.73
Evolved Transformer	217M	9.9T	3.09	2.220 ± 0.003	1.863	**73.67**	10.76	**24.07**	26.58
Synthesizer (dense)	224M	11.4T	3.47	2.334 ± 0.021	1.962	61.03	14.27	16.14	**26.63**
Synthesizer (dense plus)	243M	12.6T	3.22	2.191 ± 0.010	1.840	**73.98**	16.96	**23.81**	**26.71**
Synthesizer (dense plus alpha)	243M	12.6T	3.01	2.180 ± 0.007	**1.828**	**74.25**	17.02	**23.28**	26.61
Synthesizer (factorized)	207M	10.1T	3.94	2.341 ± 0.017	1.968	62.78	15.39	**23.55**	26.42
Synthesizer (random)	254M	10.1T	4.08	2.326 ± 0.012	2.009	54.27	10.35	19.56	26.44
Synthesizer (random plus)	292M	12.0T	3.63	2.189 ± 0.004	1.842	**73.32**	17.04	**24.87**	26.43
Synthesizer (random plus alpha)	292M	12.0T	3.42	2.186 ± 0.007	**1.828**	**75.24**	17.08	**24.08**	26.39
Universal Transformer	84M	40.0T	0.88	2.406 ± 0.036	2.053	70.13	14.09	19.05	23.91
Mixture of experts	648M	11.7T	3.20	2.148 ± 0.006	**1.785**	**74.55**	**18.13**	**24.08**	**26.94**
Switch Transformer	1100M	11.7T	3.18	2.135 ± 0.007	**1.758**	**75.38**	**18.02**	**26.19**	**26.81**
Funnel Transformer	223M	1.9T	4.30	2.288 ± 0.008	1.918	67.34	16.26	22.75	23.20
Weighted Transformer	280M	71.0T	0.59	2.378 ± 0.021	1.989	69.04	16.98	23.02	26.30
Product key memory	421M	386.6T	0.25	2.155 ± 0.003	**1.798**	**75.16**	17.04	**23.55**	**26.73**

Do Transformer Modifications Transfer Across Implementations and Applications?

Sharan Narang* Hyung Won Chung Yi Tay William Fedus

Thibault Fevry† Michael Matena† Karishma Malkan† Noah Fiedel

Noam Shazeer Zhenzhong Lan† Yanqi Zhou Wei Li

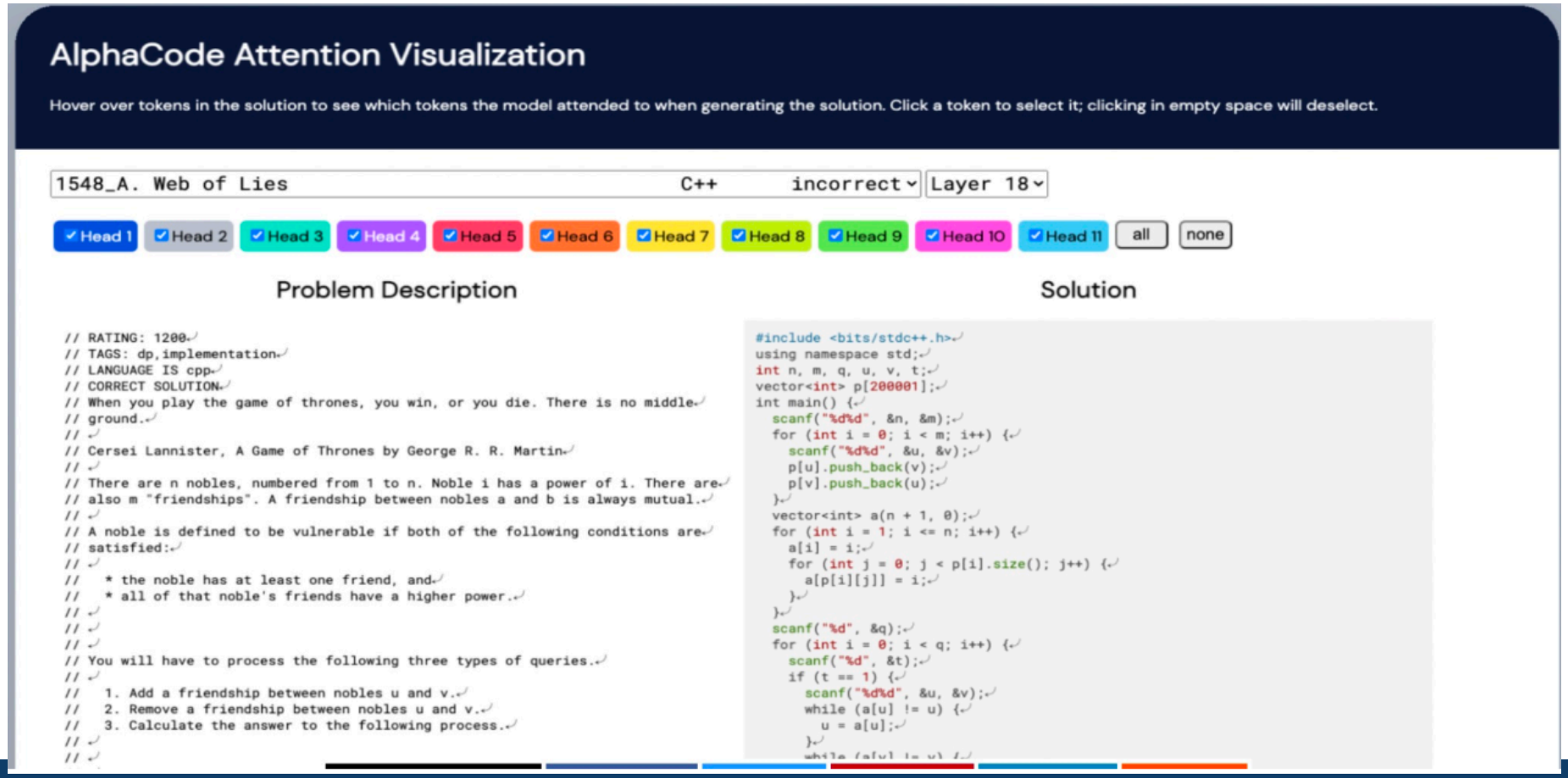
Nan Ding Jake Marcus Adam Roberts Colin Raffel†

Contents

- Transformers
  - Impact of Transformers on NLP (and ML more broadly)
  - From Recurrence (RNNs) to Attention-Based NLP Models
  - Understanding the Transformer Model
  - Drawbacks and Variants of Transformers
- Pretraining Language Models(PLMs)
  - Subword modeling
  - Motivating model pretraining from word embeddings
  - Model pretraining three ways
    - Decoders
    - Encoders
    - Encoder-Decoders
  - Very large models and in-context learning

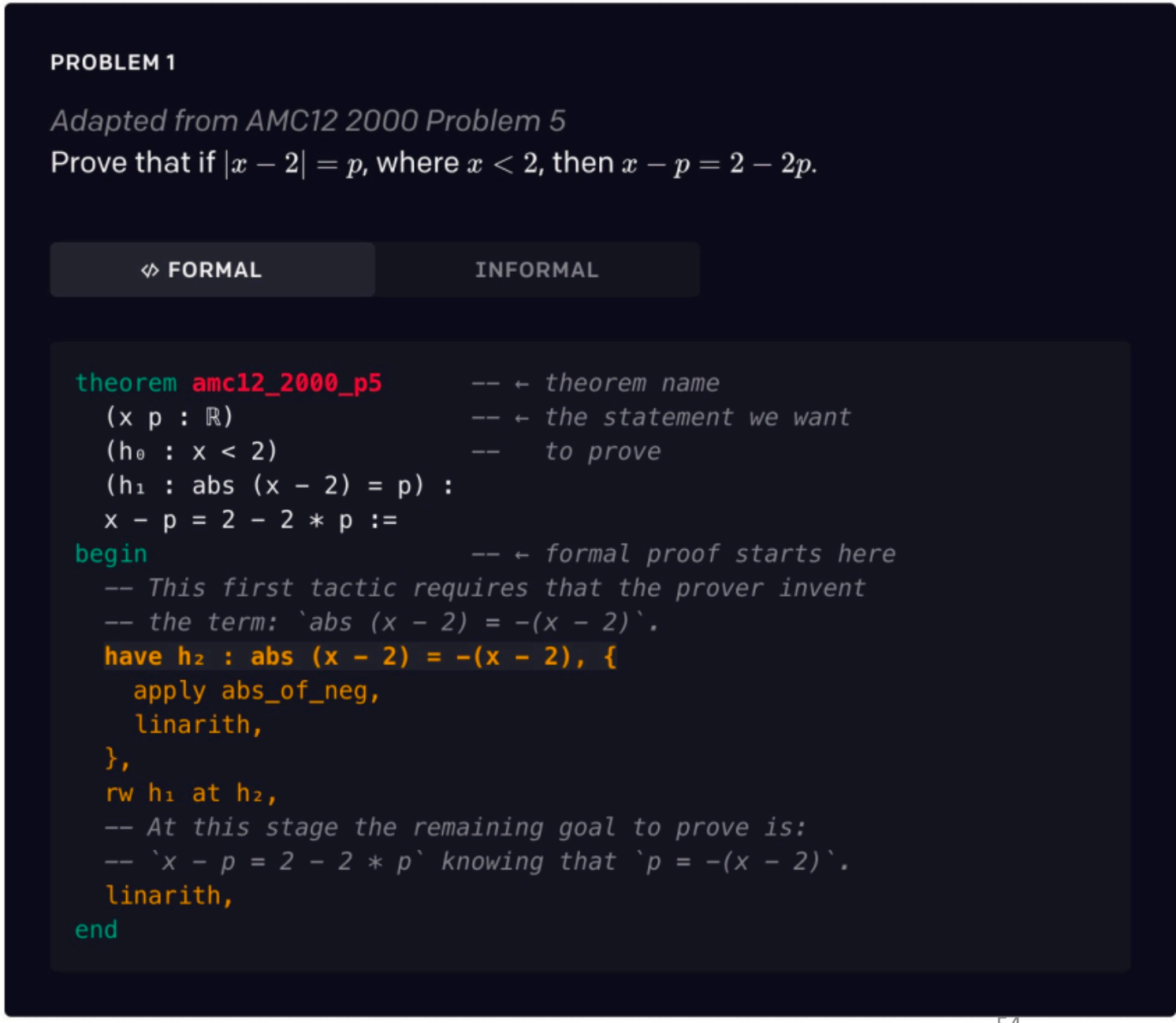
Breaking (Transformer) News!

- AlphaCode (a pre-trained Transformer-based code generation model) achieved a top 54.3% rating on Codeforces programming competitions!



More Breaking (Transformer) News!

- Pre-Trained Transformer-Based theorem prover sets new state-of-the-art (41.2% vs. 29.3%) on a collection of challenging math Olympiad questions (miniF2F) [Polu et al., 2022]

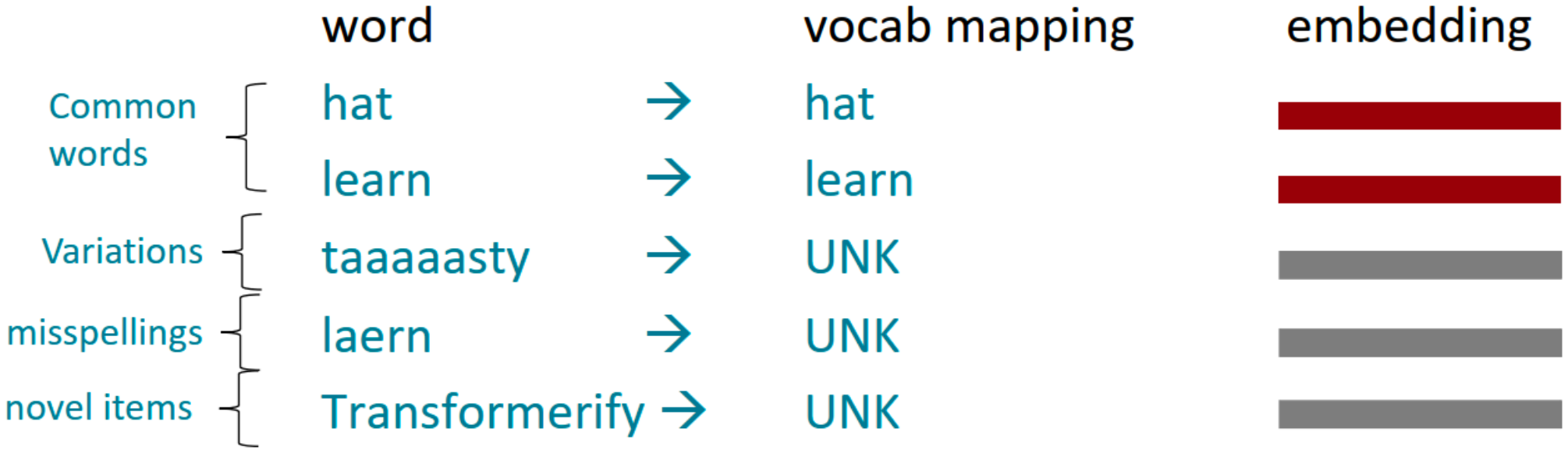


Contents

- Transformers
  - Impact of Transformers on NLP (and ML more broadly)
  - From Recurrence (RNNs) to Attention-Based NLP Models
  - Understanding the Transformer Model
  - Drawbacks and Variants of Transformers
- Pretraining Language Models(PLMs)
  - Subword modeling
  - Motivating model pretraining from word embeddings
  - Model pretraining three ways
    - Decoders
    - Encoders
    - Encoder-Decoders
  - Very large models and in-context learning

Word structure and subword models

- 首先回忆一下我们对于语言词表的假设
- 我们首先从训练集中构造一个固定的词表，可能包含几万个词
- 测试集中的词如果无法在此表中找到，一律标为UNK



Word structure and subword models

- 有限的词表在很多语言中不太合理
  - 很多语言都有复杂的形态学变化
  - 后果是更多的词语形态占据词表中很多位置，有些可能很低频
- 例子：斯瓦西里语的动词有上百种变形，每种对应不同的时态，情绪，确定性，否定，等等

Conjugation of *-ambia* [less ▲]

Form	Positive	Negative
Infinitive	kuambia	kutoambia

Positive form	Singular	Plural
Imperative	ambia	ambieni
Habitual	huambia	

Complex finite forms

Polarity	Persons				Persons / Classes		Classes											
	1st		2nd		3rd / M-wa		M-mi		Ma		Ki-vi		N		U	Ku	Pa	Mu
	Sg.	Pl.	Sg.	Pl.	Sg. / 1	Pl. / 2	3	4	5	6	7	8	9	10	11 / 14	15 / 17	16	18
Past [less ▲]																		
Positive	niliambia naliambia	tuliambia twaliambia	uliambia waliambia	mliambia mwaliambia	aliambia	waliambia	uliambia	iliambia	liliambia	yaliambia	kiliambia	viliambia	iliambia	ziliambia	uliambia	kuliambia	paliambia	muliambia
Negative	sikuambia	hatukuambia	hukuambia	hamkuambia	hakuambia	hawakuambia	haukuambia	haikuambia	halikuambia	hayakuambia	hakikuambia	havikuambia	haikuambia	hazikuambia	haukuambia	hakukuambia	hapakuambia	hamukuambia
Present [less ▲]																		
Positive	ninaambia naambia	tunaambia	unaambia	mnaambia	anaambia	wanaambia	unaambia	inaambia	linaambia	yanaambia	kinaambia	vinaambia	inaambia	zinaambia	unaambia	kunaambia	panaambia	munaambia
Negative	siambii	hatuambii	huambii	hamambii	haambii	hawaambii	hauambii	haiambii	haliambii	hayaambii	hakiambii	haviambii	haiambii	haziambii	hauambii	hakuambii	hapaambii	hamuambii
Future [less ▲]																		
Positive	nitaambia	tutaambia	utaambia	mtaambia	ataambia	wataambia	utaambia	itaambia	litaambia	yataambia	kitaambia	vitaambia	itaambia	zitaambia	utaambia	kutaambia	pataambia	mutaambia
Negative	sitaambia	hatutaambia	hutaambia	hamtaambia	hataambia	hawataambia	hautaambia	haitaambia	halitaambia	hayataambia	hakitaambia	havitaambia	haitaambia	hazitaambia	hautaambia	hakutaambia	hapataambia	hamutaambia
Subjunctive [less ▲]																		
Positive	niambie	tuambie	uambie	mambie	aambie	waambie	uambie	iambie	liambie	yaambie	kiambie	viambie	iambie	ziambie	uambie	kuambie	paambie	muambie
Negative	nisiambie	tusiambie	usiambie	msiambie	asiambie	wasiambie	usiambie	isiambie	lisiambie	yasiambie	kisiambie	visiambie	isiambie	zisiambie	usiambie	kusiambie	pasiambie	musiambie
Present Conditional [less ▲]																		
Positive	ningeambia	tungeambia	ungeambia	mngeambia	angeambia	wangeambia	ungeambia	ingeambia	lingeambia	yangeambia	kingeambia	vingeambia	ingeambia	zingeambia	ungeambia	kungeambia	pangeambia	mungeambia
Negative	nisingeambia singeambia	tusingeambia hatungeambia	usingeambia hungeambia	msingeambia hamngeambia	asingeambia hangeambia	wasingeambia hawangeambia	usingeambia haungeambia	isingeambia haingeambia	lisingeambia halingeambia	yasingeambia hayangeambia	kisingeambia hakingeambia	visingeambia havingeambia	isingeambia haingeambia	zisingeambia hazingeambia	usingeambia haungeambia	kusingeambia hakungeambia	pasingeambia hapangeambia	musingeambia hamungeambia
Past Conditional [less ▲]																		
Positive	ningaliambia	tungaliambia	ungaliambia	mngaliambia	angaliambia	wangaliambia	ungaliambia	ingaliambia	lingaliambia	yangaliambia	kingaliambia	vingaliambia	ingaliambia	zingaliambia	ungaliambia	kungaliambia	pangaliambia	mungaliambia
Negative	nisingaliambia singaliambia	tusingaliambia hatungaliambia	usingaliambia hungaliambia	msingaliambia hamngaliambia	asingaliambia hangaliambia	wasingaliambia hawangaliambia	usingaliambia haungaliambia	isingaliambia haingaliambia	lisingaliambia halingaliambia	yasingaliambia hayangaliambia	kisingaliambia hakingaliambia	visingaliambia havingaliambia	isingaliambia haingaliambia	zisingaliambia hazingaliambia	usingaliambia haungaliambia	kusingaliambia hakungaliambia	pasingaliambia hapangaliambia	musingaliambia hamungaliambia
Conditional Contrary to Fact [less ▲]																		
Positive	ningeliambia	tungeliambia	ungeliambia	mngeliambia	angeliambia	wangeliambia	ungeliambia	ingeliambia	lingeliambia	yangeliambia	kingeliambia	vingeliambia	ingeliambia	zingeliambia	ungeliambia	kungeliambia	pangeliambia	mungeliambia
Gnomic [less ▲]																		
Positive	naambia	twaambia	waambia	mwaambia	aambia	waambia	waambia	yaambia	laambia	yaambia	chaambia	vyaambia	yaambia	zaambia	waambia	kwaambia	paambia	mwaambia
Perfect [less ▲]																		

The byte-pair encoding algorithm

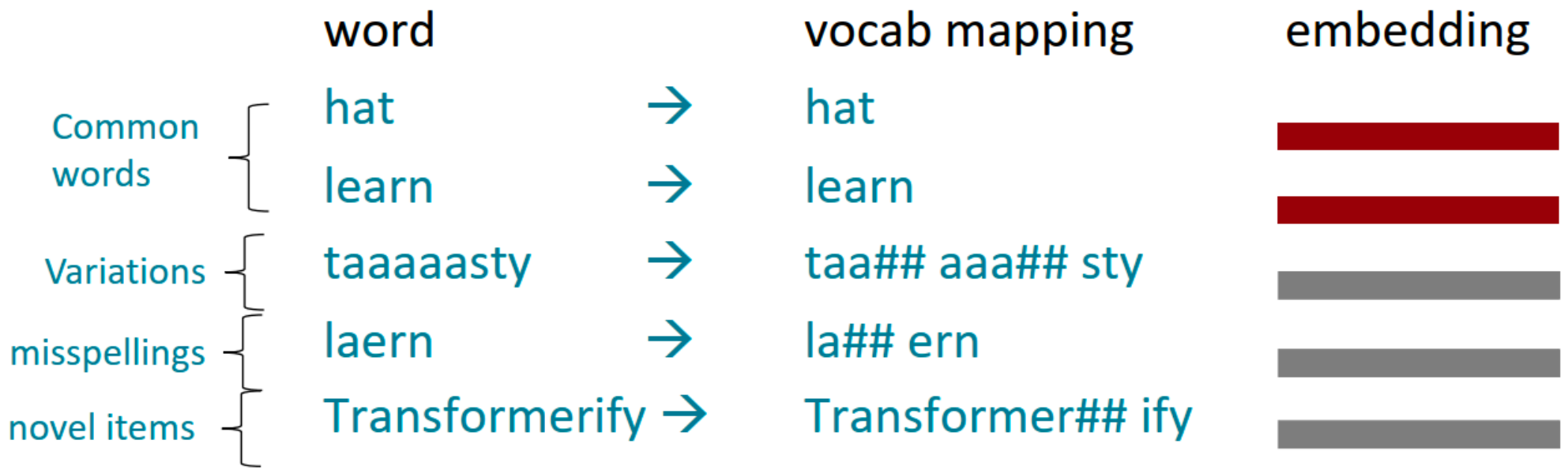
- NLP中的Subword建模有很多方法
  - 目前最通用的方法是建立subword的词表
  - 在训练和测试时，每个词都被分为几个subword
- Byte-pair encoding是一种简单有效的方法
  - 最开始，我们只有一个只包含字符和“词结束符”的词表
  - 在语料库中，找到最常见的字符对（“如ab”），则将ab加入subword词表中
  - 不断重复此过程，直到词表大小符合要求
- 另一种常用方法，WordPiece

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

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation(Wu et al.2016)

Word structure and subword models

- 常见词最终都会变成subword词表的一部分，罕见词会被拆成几个部分
- 最坏情况下，单词会被拆成字符



Contents

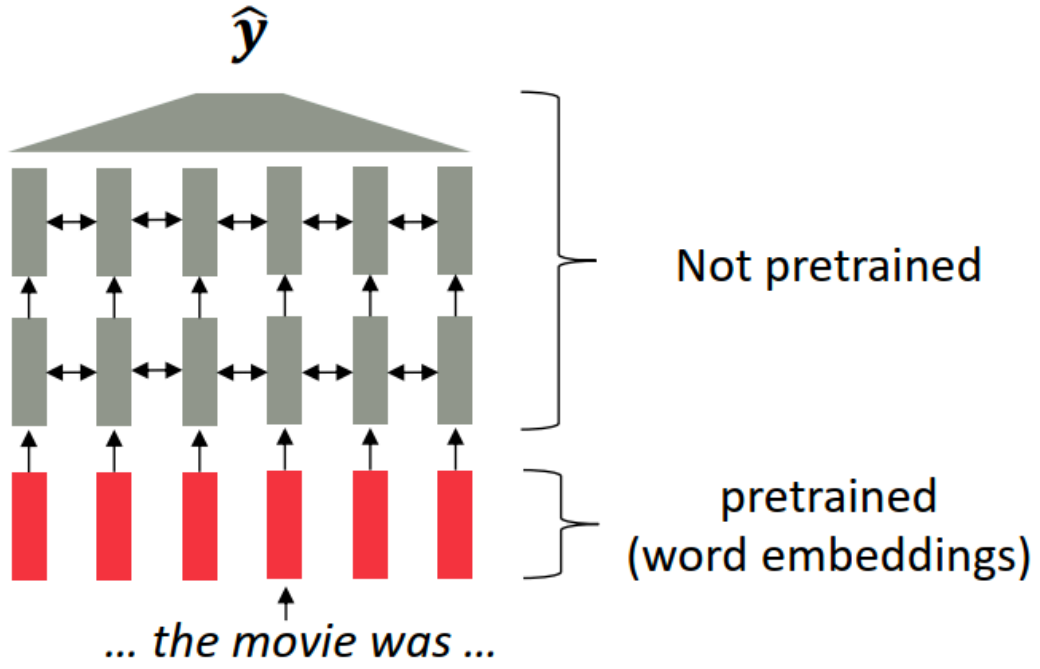
- Transformers
  - Impact of Transformers on NLP (and ML more broadly)
  - From Recurrence (RNNs) to Attention-Based NLP Models
  - Understanding the Transformer Model
  - Drawbacks and Variants of Transformers
- Pretraining Language Models(PLMs)
  - Subword modeling
  - Motivating model pretraining from word embeddings
  - Model pretraining three ways
    - Decoders
    - Encoders
    - Encoder-Decoders
  - Very large models and in-context learning

Motivating word meaning and context

- 回忆之前讲word embedding时候引用的一句话
  - “You shall know a word by the company it keeps” (J. R. Firth 1957: 11)
- 这句话总结了分布式语义，同时引发Word2vec的研究
  - “… the complete meaning of a word is always contextual, and no study of meaning apart from a complete context can be taken seriously.” (J. R. Firth 1935)
- 考虑 I **record** the **record**，两个record有不同含义

Where we were: pretrained word embeddings

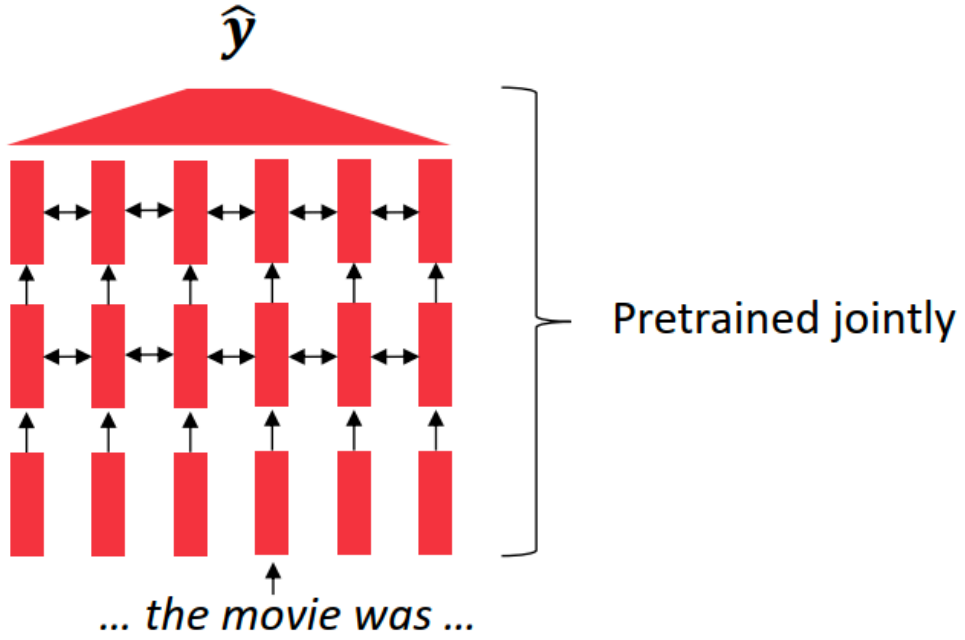
- Since 2017:
  - 仅用预训练的词向量
  - 在训练模型时考虑如何利用上下文信息
- 问题：
  - 下游任务的训练数据必须足以学习上下文信息
  - 大多数参数都是随机初始化的



[Recall, *movie* gets the same word embedding, no matter what sentence it shows up in]

Where we're going: pretraining whole models

- 现代NLP：
  - NLP模型中几乎所有参数都用预训练来初始化
  - 预训练方法隐藏一部分输入，令模型重构这部分输入
- 在建立强大的以下三种模型时非常有效
  - 语言的表示
  - 对于强大NLP模型的参数初始化
  - 可以被用来采样的语言分布



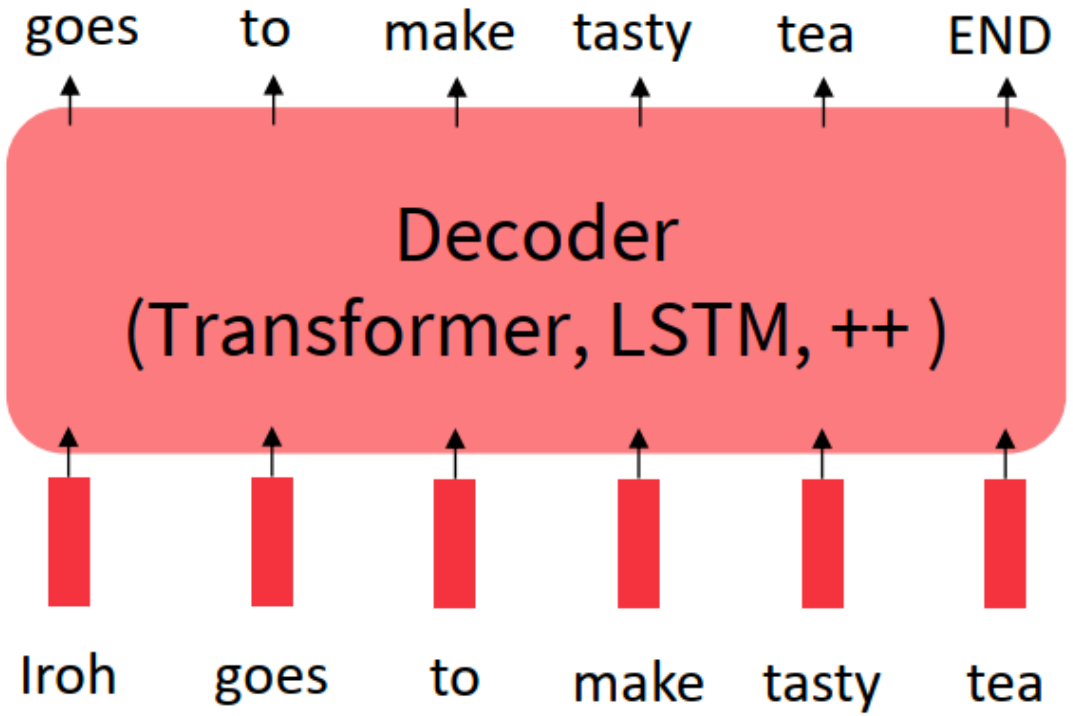
[This model has learned how to represent entire sentences through pretraining]

What can we learn from reconstructing the input?

- The woman walked across the street, checking for traffic over ___ shoulder.
- I went to the ocean to see the fish, turtles, seals, and _____.
- Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was ___.
- I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, ____

Pretraining through language modeling [Dai and Le, 2015]

- 回忆语言模型任务
  - 给定前面的若干词，预测后一个词 $p_\theta(w_t|w_{1:t-1})$
  - 这种训练数据很多
- 通过语言模型来预训练
  - 利用大量语料来训练神经语言模型
  - 存储训好的参数

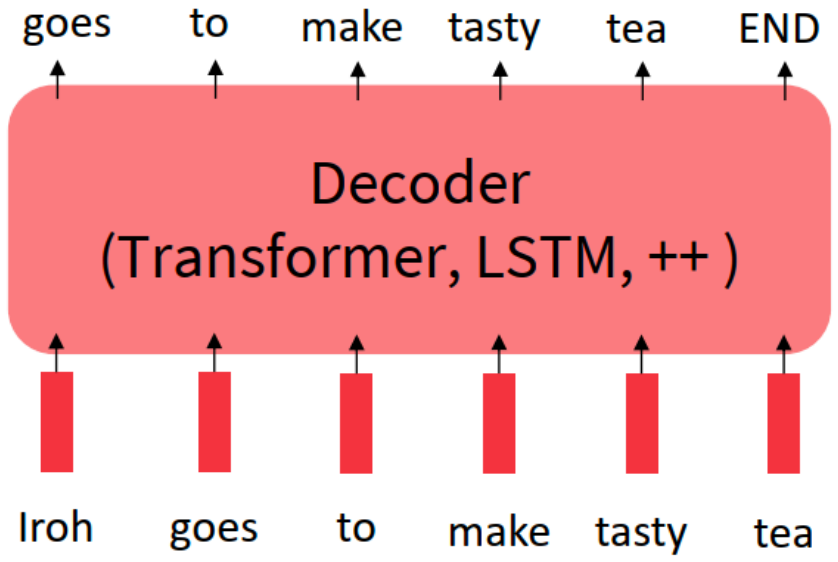


The Pretraining / Finetuning Paradigm

- 预训练可以以参数初始化的方式来提升NLP模型的效果

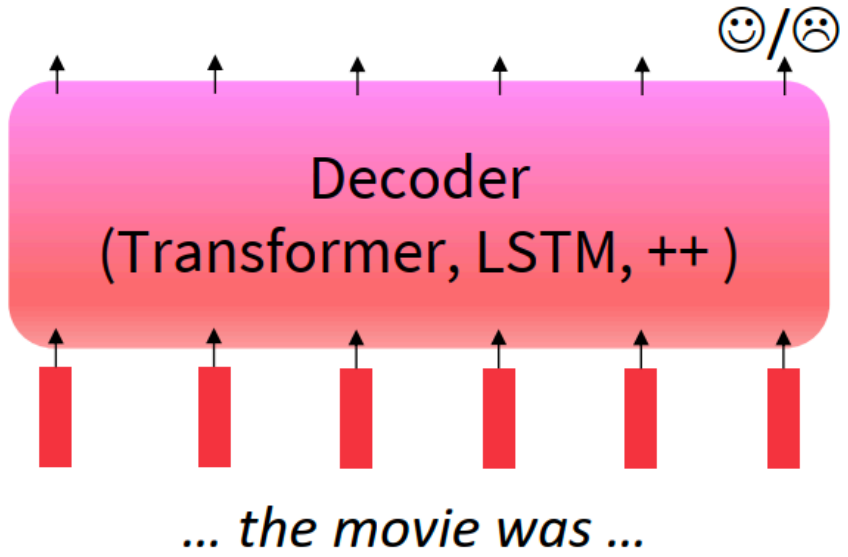
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



Step 2: Finetune (on your task)

Not many labels; adapt to the task!

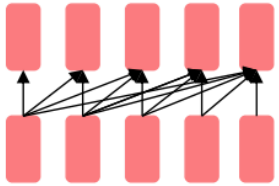


Contents

- Transformers
  - Impact of Transformers on NLP (and ML more broadly)
  - From Recurrence (RNNs) to Attention-Based NLP Models
  - Understanding the Transformer Model
  - Drawbacks and Variants of Transformers
- Pretraining Language Models(PLMs)
  - Subword modeling
  - Motivating model pretraining from word embeddings
  - Model pretraining three ways
    - Decoders
    - Encoders
    - Encoder-Decoders
  - Very large models and in-context learning

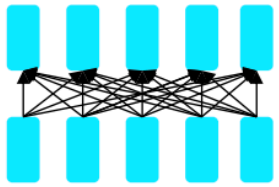
Pretraining for three types of architectures

- The neural architecture influences the type of pretraining, and natural use cases.



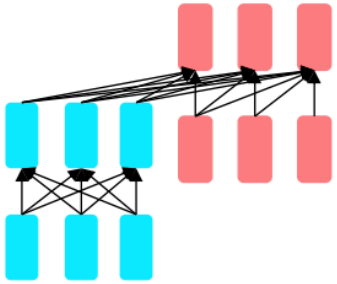
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- Examples: GPT-2, GPT-3, LaMDA



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- Examples: BERT and its many variants, e.g. RoBERTa

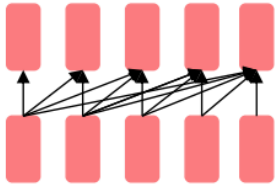


Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- Examples: Transformer, T5, Meena

Pretraining for three types of architectures

- The neural architecture influences the type of pretraining, and natural use cases.

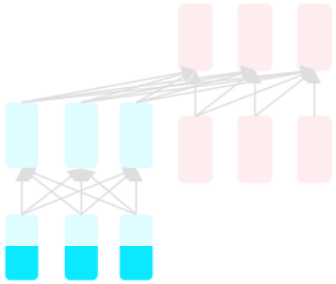


Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- Examples: GPT-2, GPT-3, LaMDA

Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- Examples: BERT and its many variants, e.g. RoBERTa



Encoder-Decoders

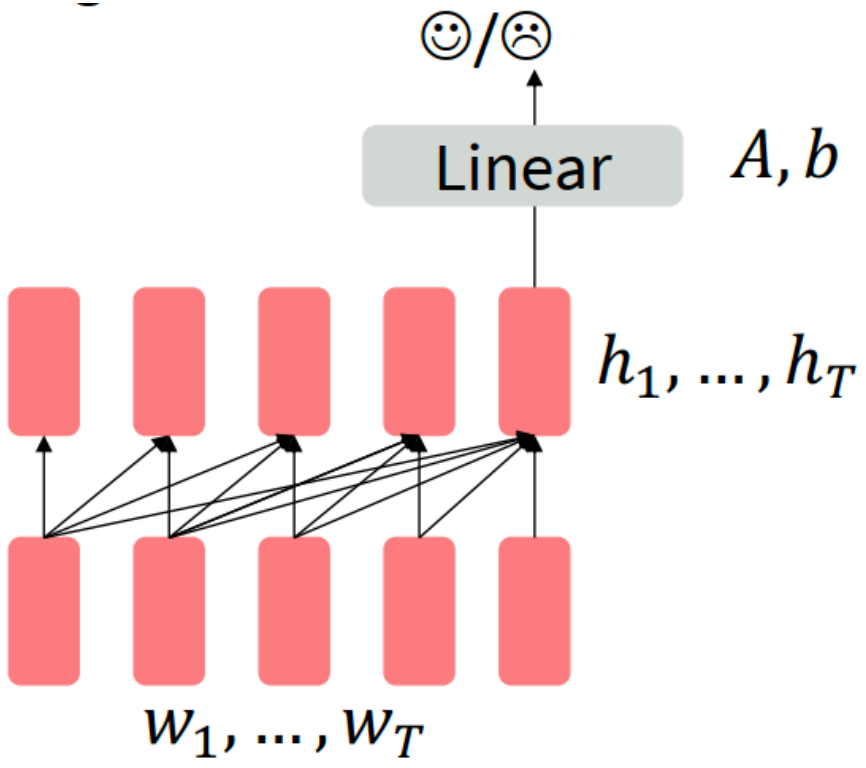
- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- Examples: Transformer, T5, Meena

Pretraining decoders

- 当我们使用利用LM预训练的decoder时，暂时忘记LM是为了建模 $p(w_t|w_{1:t-1})$ 的
- 我们利用最后一个词的隐状态来训练一个分类器，从而微调这个预训练模型

$$h_1, \ldots, h_T = \text{Decoder}(w_1, \ldots, w_T)$$
$$y \sim Ah_T + b$$

- A，b是随机初始化的参数，用下游模型来训练
- 梯度回传整个网络



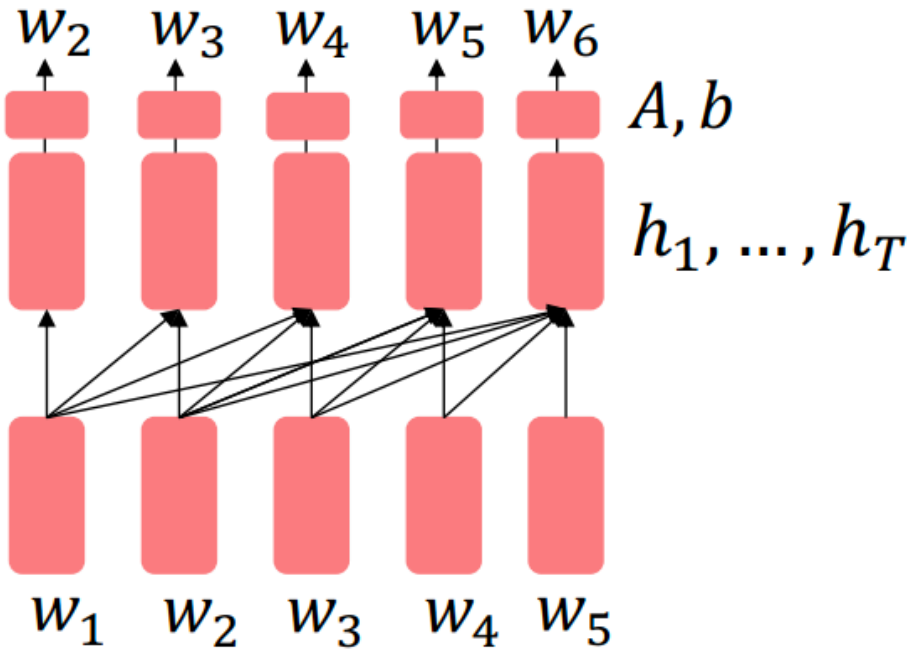
[Note how the linear layer hasn't been pretrained and must be learned from scratch.]

Pretraining decoders

- 下游任务如果是生成任务，那么很自然的，我们要微调$p(w_t|w_{1:t-1})$
- 如果任务的输出是个序列，而且词表与预训练数据高度重合，那么将会非常有效
  - 对话（上下文=对话历史）
  - 摘要（上下文=文档）

$$h_1, \ldots, h_T = \text{Decoder}(w_1, \ldots, w_T)$$
$$w_t \sim Ah_{t-1} + b$$

- A，b是随机初始化的参数，用下游模型来训练



[Note how the linear layer has been pretrained.]

Generative Pretrained Transformer (GPT) [Radford et al., 2018]

2018's GPT was a big success in pretraining a decoder!

- Transformer decoder with 12 layers.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus: over 7000 unique books.
  - Contains long spans of contiguous text, for learning long-distance dependencies

GPT-1（预训练+微调）

- **GPT-1模型基于Transformer解除了顺序关联和依赖性的前提，采用生成式模型方式，重点考虑了从原始文本中有效学习的能力，这对于减轻自然语言处理（NLP）中对监督学习的依赖至关重要**

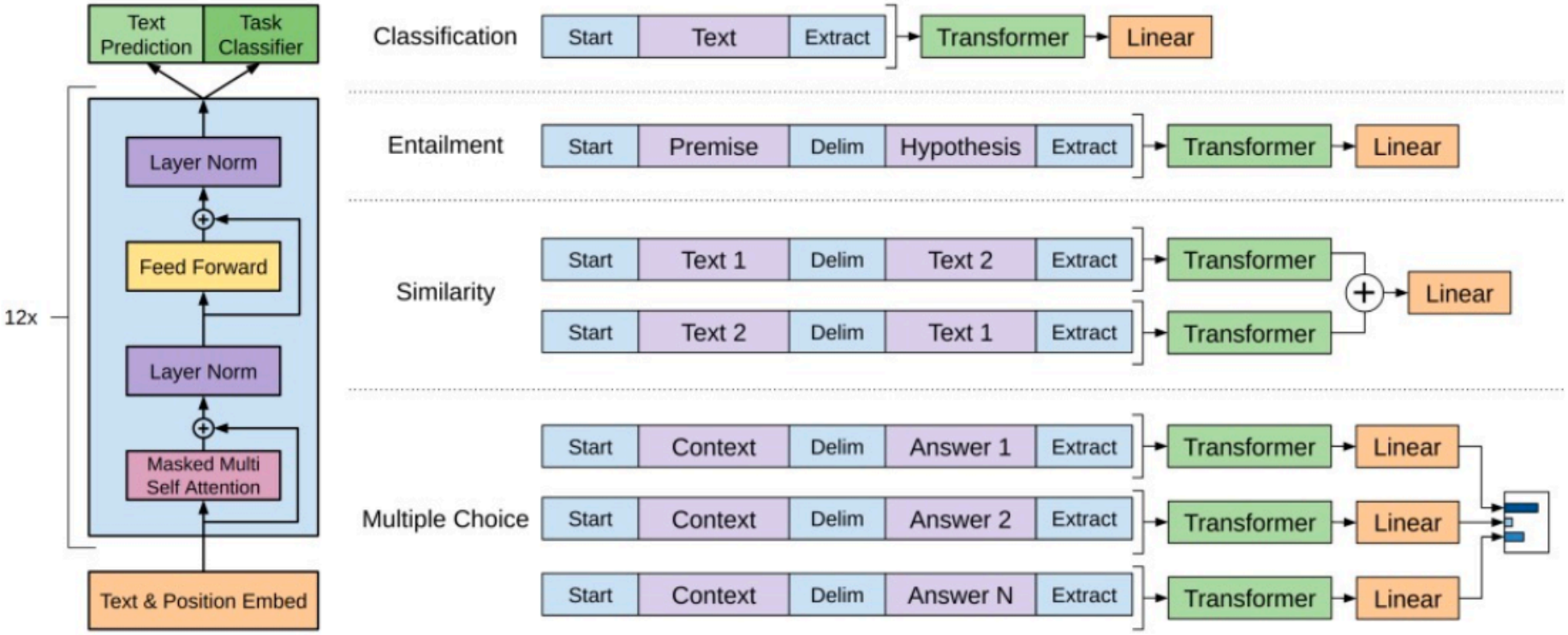
✓ GPT（Generative Pre-training Transformer）于2018年6月由OpenAI首次提出。GPT模型考虑到在自然语言理解中有大量不同的任务，尽管大量的未标记文本语料库非常丰富，但用于学习这些特定任务的标记数据却很少，这使得经过区分训练的模型很难充分执行。同时，大多数深度学习方法需要大量手动标记的数据，这限制了它们在许多缺少注释资源的领域的适用性。

✓ **在考虑以上局限性的前提下，GPT论文中证明，通过对未标记文本的不同语料库进行语言模型的生成性预训练，然后对每个特定任务进行区分性微调，可以实现这些任务上的巨大收益。**和之前方法不同，GPT在微调期间使用任务感知输入转换，以实现有效的传输，同时对模型架构的更改最小。

图29：GPT-1模型的核心手段是预训练（Pre-training）

Generative Pretrained Transformer (GPT) [Radford et al., 2018]

- How do we format inputs to our decoder for finetuning tasks?



- The linear classifier is applied to the representation of the [EXTRACT] token.

GPT架构对于Transformer的改进

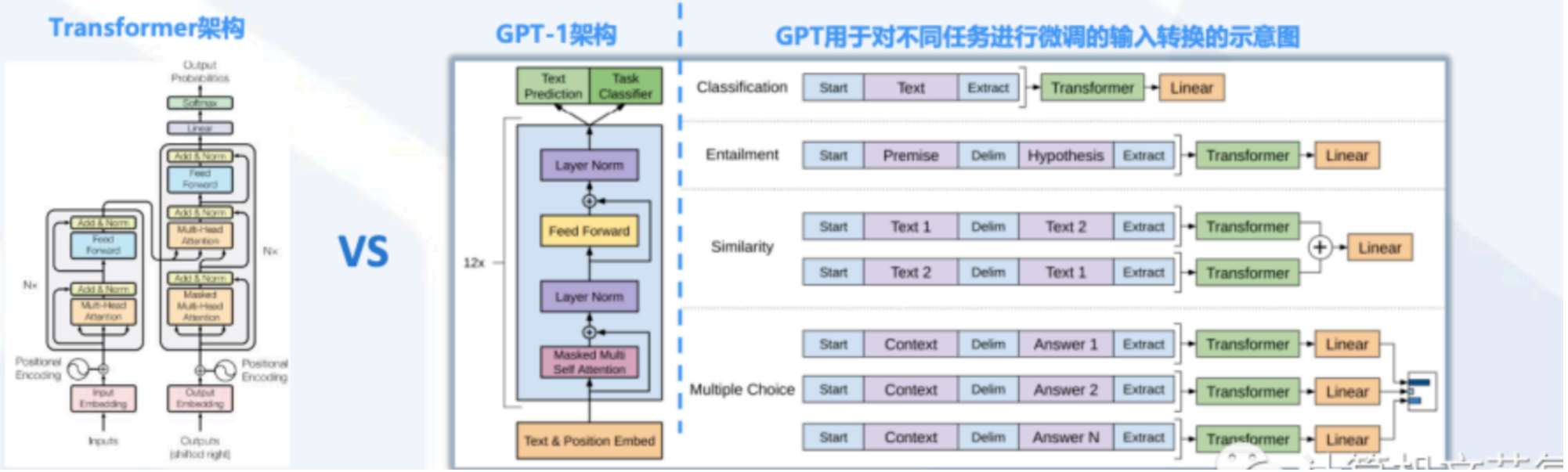
■ GPT相比于Transformer等模型进行了显著简化

✓ 相比于Transformer，GPT训练了一个12层**仅decoder的解码器**（原Transformer模型中包含Encoder和Decoder两部分）。

✓ 相比于Google的BERT(Bidirectional Encoder Representations from Transformers,双向编码生成Transformer)，**GPT仅采用上文预测单词**（BERT采用了基于上下文双向的预测手段）。

注：ChatGPT的表现更贴近人类意图，部分因为一开始GPT是基于上文的预测，这更贴近人类的话语模式，因为人类言语无法基于将来的话来做分析。

图30：GPT-1模型相比于Transformer模型有了显著简化

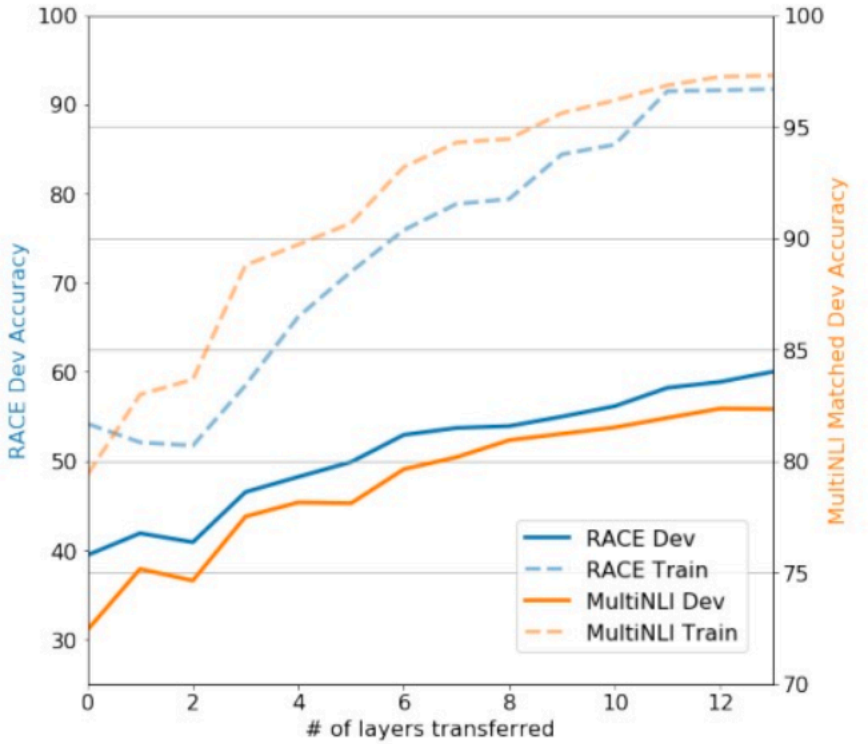


Generative Pretrained Transformer (GPT) [Radford et al., 2018]

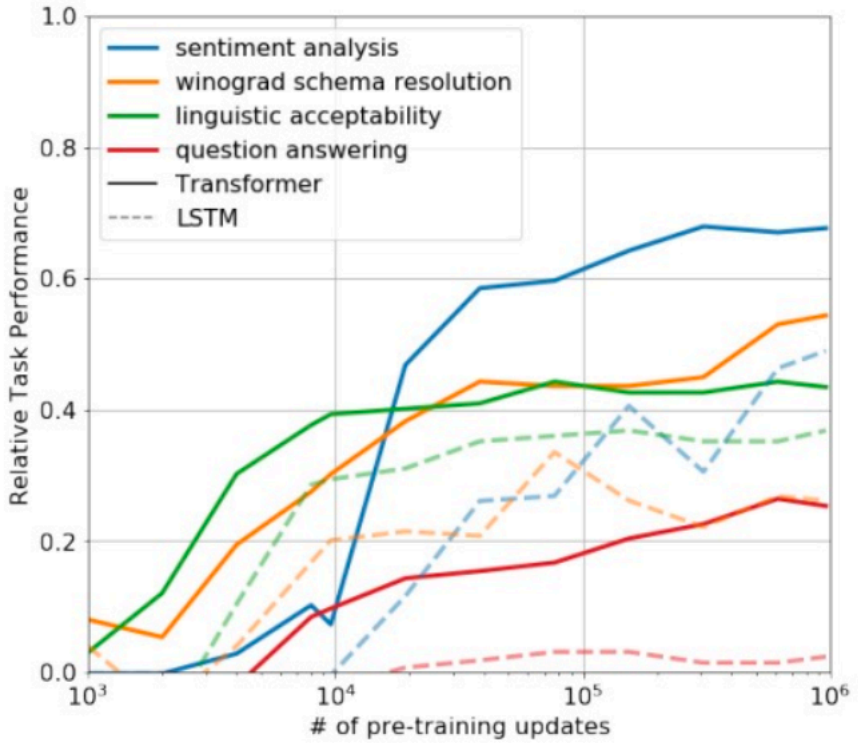
- GPT results on various natural language inference datasets

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3		
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	**61.7**
Finetuned Transformer LM (ours)	**82.1**	**81.4**	**89.9**	**88.3**	**88.1**	56.0

Examining the Effect of Pretraining in GPT [Radford et al., 2018]



Transformer层数迁移的越多，RACE和MultiNLI的效果越好



Transformer和LSTM的zero-shot效果随着预训练更新数的关系

Increasingly convincing generations (GPT2) [Radford et al., 2018]

- GPT2，更大的GPT，用更多数据预训练
- 可以生成更加可信的自然语言文本

Context (human-written): In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.
GPT-2: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

GPT2 解决 zero-shot问题

- 目的：成为更通用的模型，可以解决多任务
- 去掉有监督微调，只用无监督预训练
- 更大，1.5B参数，48层Transformer，40GB文本预训练
- 在无监督NLP任务中具有决定性优势，依然无法打败有监督模型

图32：GPT-2尚未解决诸多瓶颈问题

存在的问题01：

从实用的角度来看，每一项新任务都需要一个标记示例的大数据集，这限制了语言模型的适用性；

对于其中的许多任务（从纠正语法到生成抽象概念的示例，再到评论一个短篇故事等等），很难收集一个大型的监督训练数据集，特别是当每个新任务都必须重复该过程时。

存在的问题02：

- **预训练加微调范式中，可能在这种范式下实现的泛化可能很差**，因为该模型过于特定于训练分布，并且在其之外无法很好地泛化。
- 微调模型在特定基准上的性能，即使名义上是人类水平，也可能夸大基础任务的实际性能。

存在的问题03：

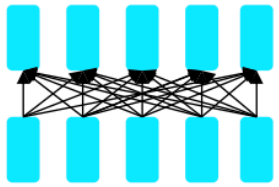
因为人类学习大多数语言任务不需要大型受监督的数据集，当前NLP技术在概念上具有一定的局限性。

Pretraining for three types of architectures

- The neural architecture influences the type of pretraining, and natural use cases.

Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- Examples: GPT-2, GPT-3, LaMDA



Encoders

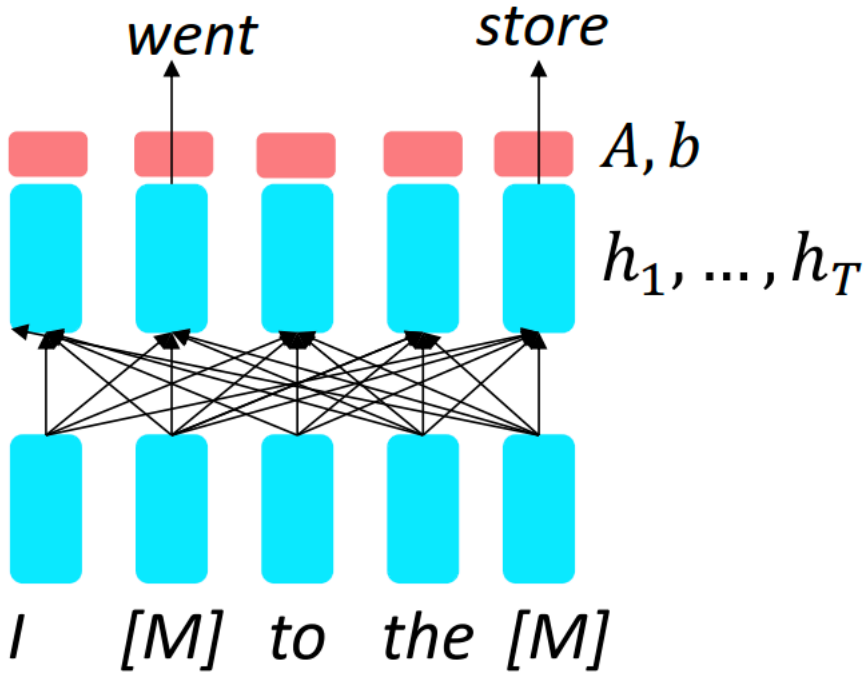
- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- Examples: BERT and its many variants, e.g. RoBERTa

Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- Examples: Transformer, T5, Meena

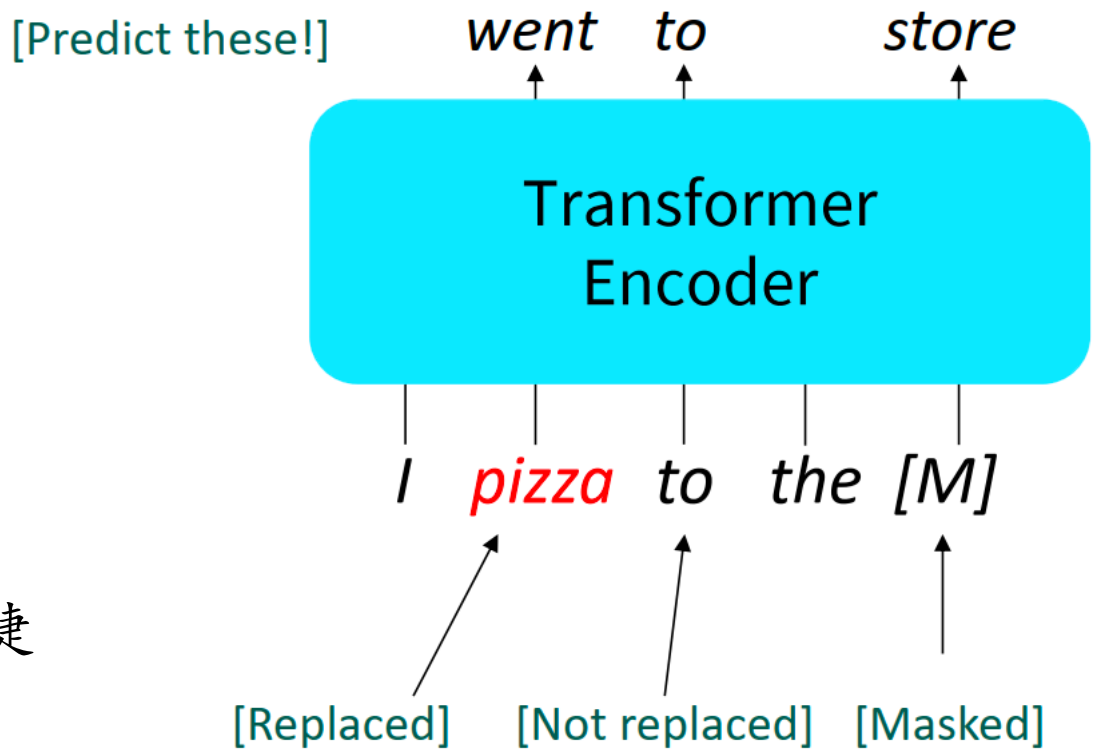
Pretraining encoders: what pretraining objective to use?

- 我们刚刚学习了语言模型的预训练，但encoder可以接受两边的信息，所以不能用语言模型的方法来预训练
- Idea：用[MASK]符号替换掉输入文本中的一部分词，让模型来预测这部分词
- 只在被mask out的位置加loss function
- 令 $\tilde{x}$ 为被mask掉一些词的句子x，则我们想要学习的是 $p_\theta(x|\tilde{x})$
- 此为**masked LM**



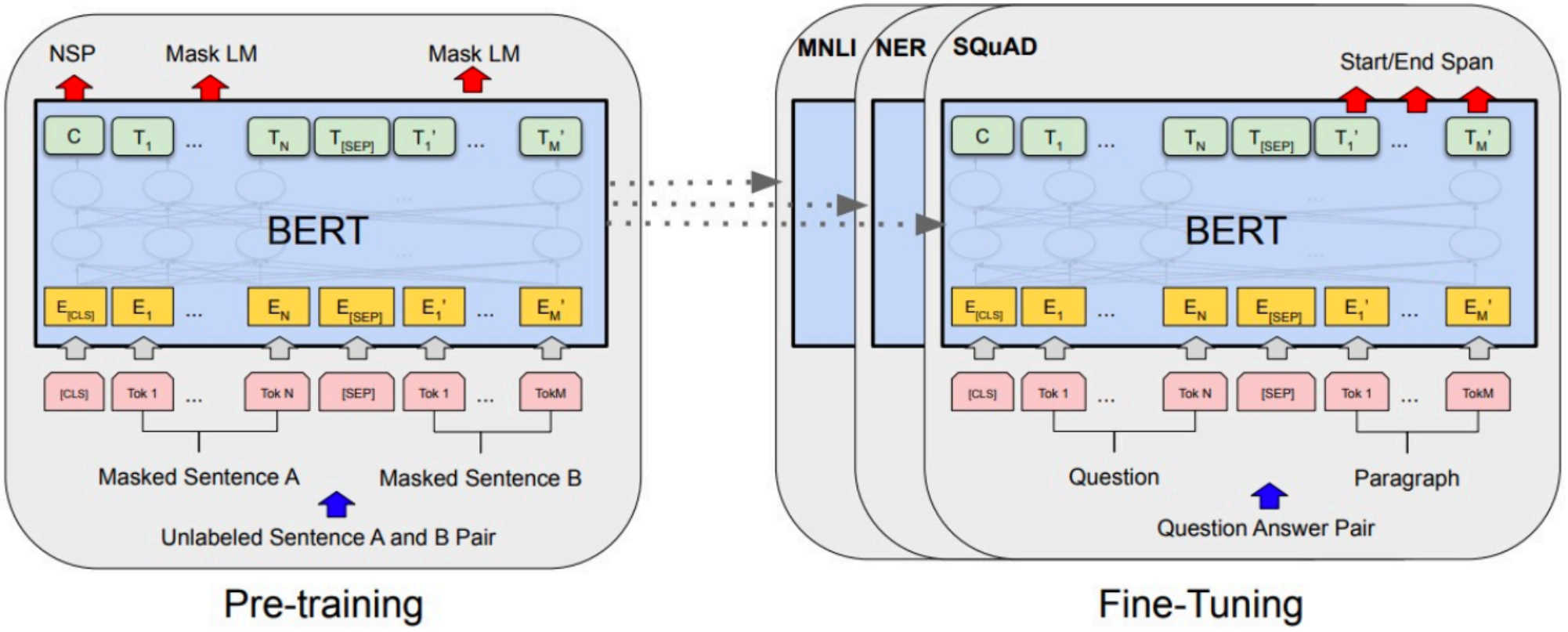
BERT: Bidirectional Encoder Representations from Transformers

- BERT的mask LM方法
- Predict a random 15% of (sub)word tokens.
  - Replace input word with [MASK] 80% of the time
  - Replace input word with a random token 10% of the time
  - Leave input word unchanged 10% of the time (but still predict it!)
- 为什么这么折腾？
  - 不让模型捕捉到某种固定模式，不让找捷径，逼迫它按照我们的想法去学习



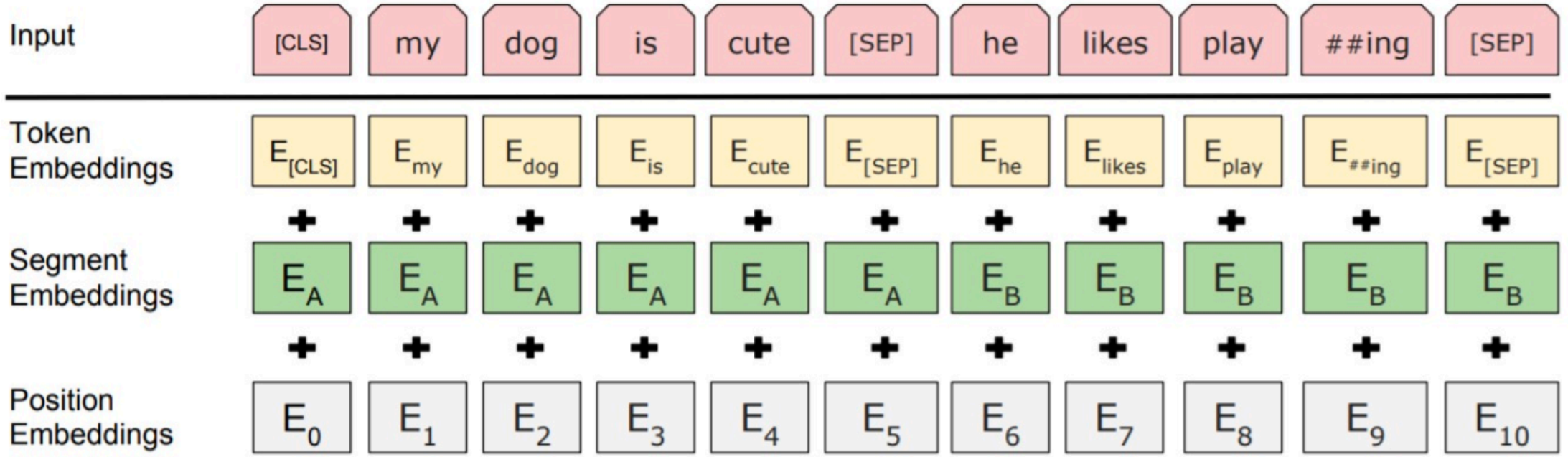
BERT: Bidirectional Encoder Representations from Transformers

- 预训练结构与微调结构大同小异



BERT: Bidirectional Encoder Representations from Transformers

- 第二个预训练任务：输入到BERT的文本包含两部分
- BERT需要预测，这两个句子是接续关系还是随机采样出来的（其实这个是不必要的）



BERT: Bidirectional Encoder Representations from Transformers

Details about BERT

- Two models were released:
  - BERT-base: 12 layers, 768-dim hidden states, 12 attention heads, 110 million params
  - BERT-large: 24 layers, 1024-dim hidden states, 16 attention heads, 340 million params.
- Trained on:
  - BooksCorpus (800 million words)
  - English Wikipedia (2,500 million words)
- Pretraining is expensive and impractical on a single GPU.
  - BERT was pretrained with 64 TPU chips for a total of 4 days.
  - (TPUs are special tensor operation acceleration hardware)
- Finetuning is practical and common on a single GPU
  - “Pretrain once, finetune many times.”

BERT: Bidirectional Encoder Representations from Transformers

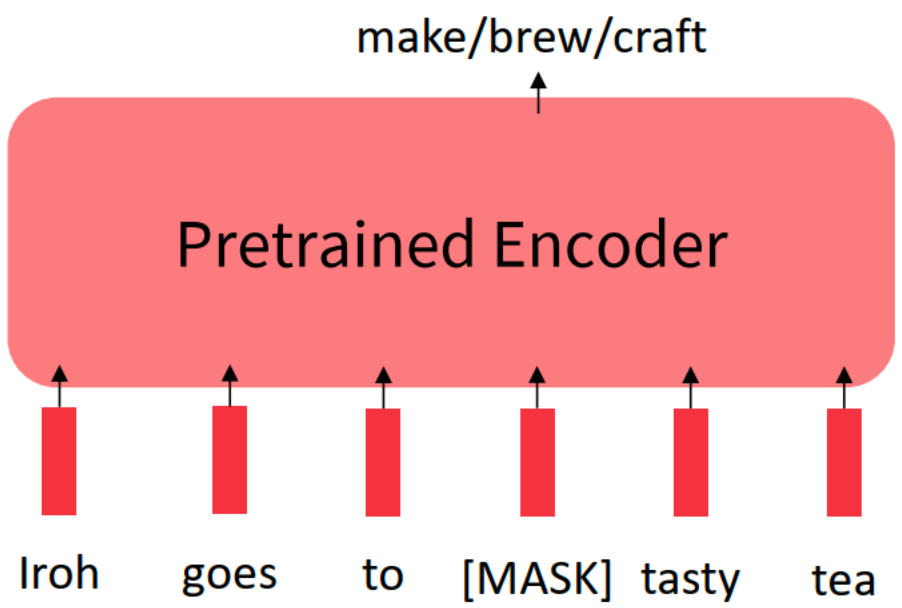
- BERT非常受欢迎，用途广泛;对BERT的微调为广泛的任务带来了新的最高结果。
- **QQP**: Quora Question Pairs (detect paraphrase questions)
- **QNLI**: natural language inference over question answering data
- **SST-2**: sentiment analysis
- **CoLA**: corpus of linguistic acceptability (detect whether sentences are grammatical.)
- **STS-B**: semantic textual similarity
- **MRPC**: microsoft paraphrase corpus
- **RTE**: a small natural language inference corpus

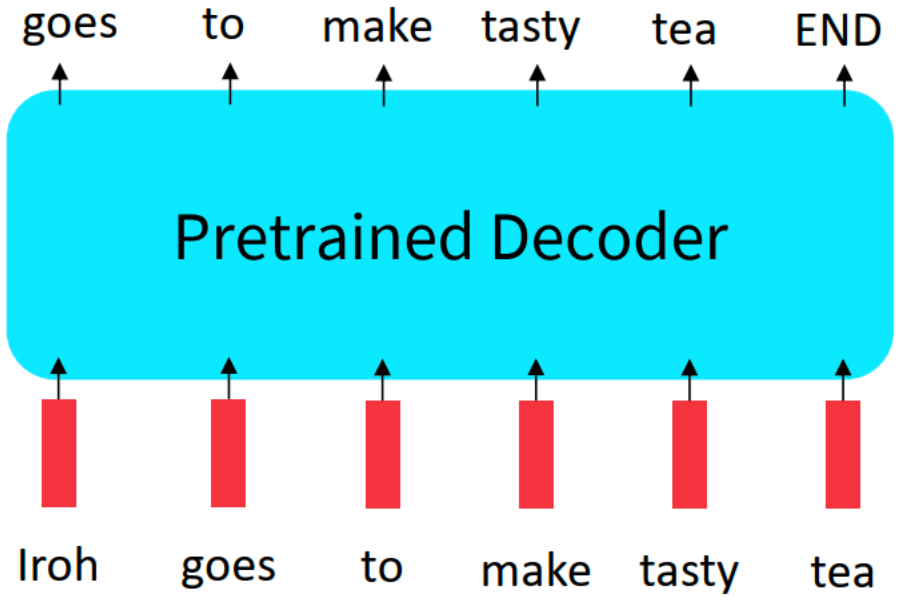
System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Note that $\text{BERT}_{\text{BASE}}$ was chosen to have the same number of parameters as OpenAI GPT.

Limitations of pretrained encoders

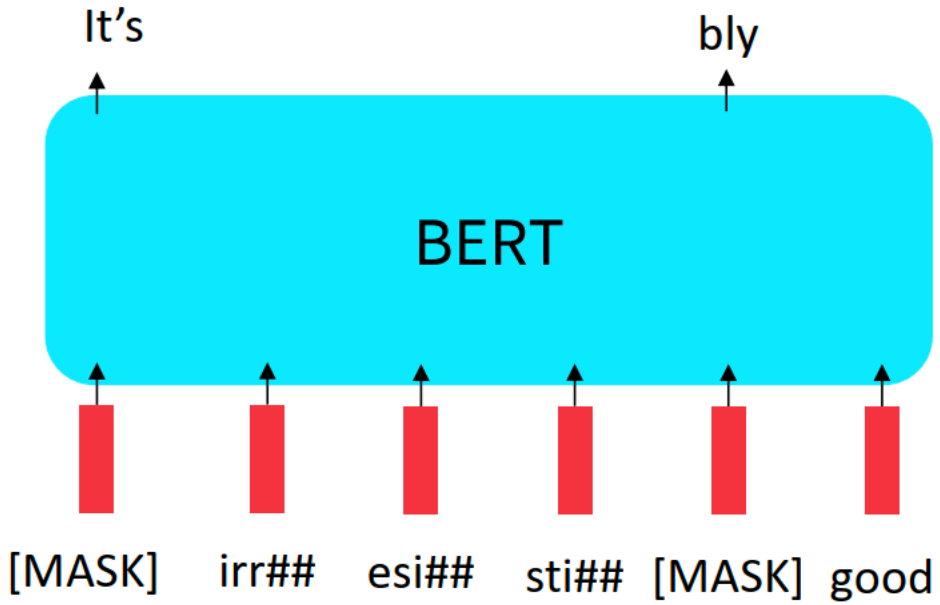
- 看起来结果不错，那么可以把pretrained encoders应用在所有任务上吗
- 如果任务包含文本生成，那就要考虑用pretrained decoder
- BERT与其他的预训练encoder无法提供天然的自回归性

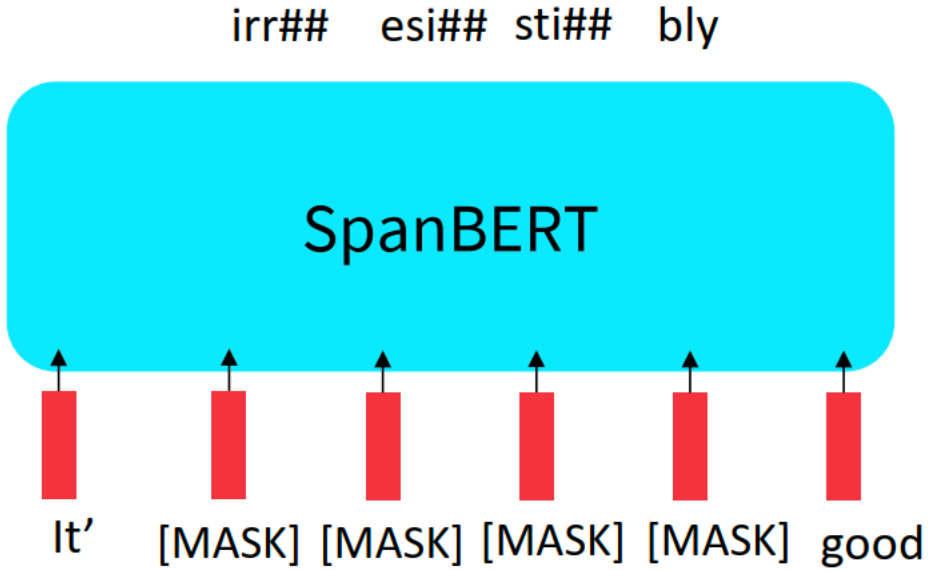




Extensions of BERT

- BERT有很多变种RoBERTa, SpanBERT
- RoBERTa: mainly just train BERT for longer and remove next sentence prediction!
- SpanBERT: masking contiguous spans of words makes a harder, more useful pretraining task





Extensions of BERT

- 一句话总结RoBERTa：更多的训练与更多的数据加持之下，即使不改BERT结构，性能也能提升

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7

Pretraining for three types of architectures

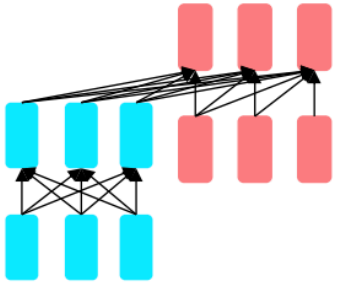
- The neural architecture influences the type of pretraining, and natural use cases.

Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words
- Examples: GPT-2, GPT-3, LaMDA

Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?
- Examples: BERT and its many variants, e.g. RoBERTa



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?
- Examples: Transformer, T5, Meena

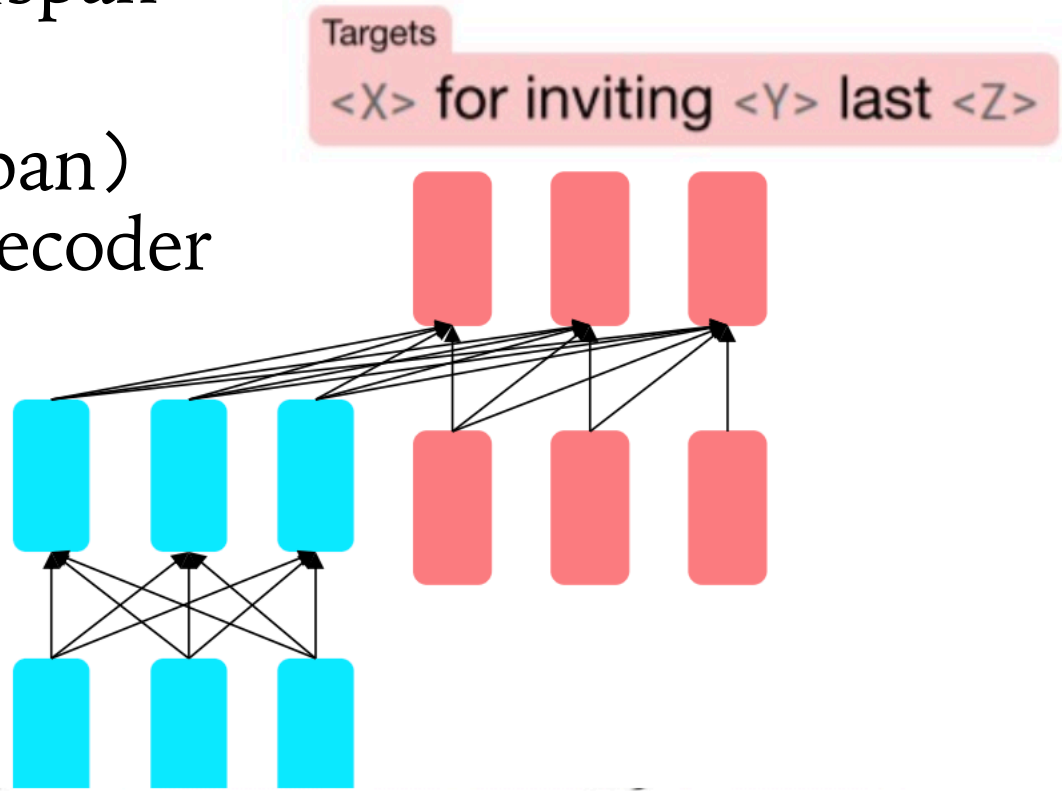
Pretraining encoder-decoders: what pretraining objective to use?

- Raffel et al., 2018发现最有效的方法是span corruption： T5
- 在输入中把不同长度的单词子序列（span）替换为占位符（placeholder），并在decoder中把之前替换掉的span解码出来

Original text

Thank you for inviting me to your party last week.

这个在预处理阶段完成。Decoder看来这还是个类似LM的任务

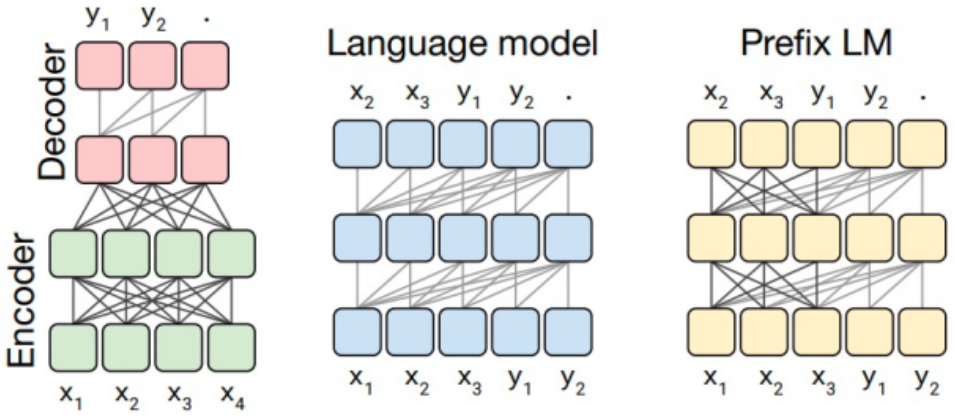


Inputs

Thank you <X> me to your party <Y> week.

Pretraining encoder-decoders: what pretraining objective to use?

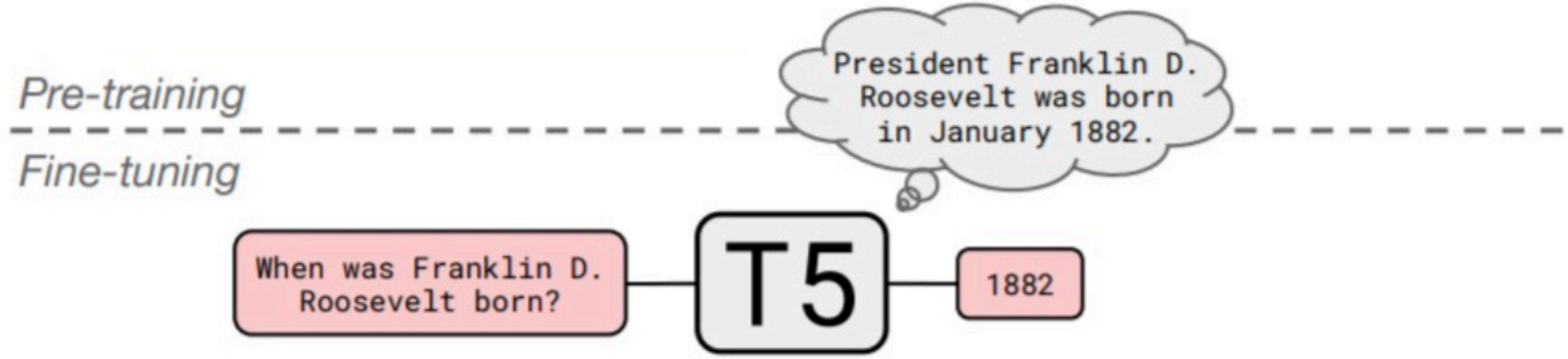
- Raffel et al., 2018发现
  - encoder-decoder比单纯的decoder效果好
  - Span corruption比单纯的LM效果好



Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

Pretraining encoder-decoders: what pretraining objective to use?

- T5的一个迷人特性：它可以进行微调以回答各种问题，从其参数中检索知识。



	NQ	WQ	TQA dev	TQA test	
Karpukhin et al. (2020)	**41.5**	42.4	**57.9**	–	
T5.1.1-Base	25.7	28.2	24.2	30.6	**220 million params**
T5.1.1-Large	27.3	29.5	28.5	37.2	**770 million params**
T5.1.1-XL	29.5	32.4	36.0	45.1	**3 billion params**
T5.1.1-XXL	32.8	35.6	42.9	52.5	**11 billion params**
T5.1.1-XXL + SSM	35.2	**42.8**	51.9	**61.6**	

- NQ: Natural Questions
- WQ: WebQuestions
- TQA: Trivia QA
- All “open-domain” versions

Contents

- Transformers
  - Impact of Transformers on NLP (and ML more broadly)
  - From Recurrence (RNNs) to Attention-Based NLP Models
  - Understanding the Transformer Model
  - Drawbacks and Variants of Transformers
- Pretraining Language Models(PLMs)
  - Subword modeling
  - Motivating model pretraining from word embeddings
  - Model pretraining three ways
    - Decoders
    - Encoders
    - Encoder-Decoders
  - Very large models and in-context learning

GPT-3简介

- GPT-3（Generative Pre-trained Transformer 3）是一个自回归语言模型，目的是为了使用深度学习生成人类可以理解的自然语言。
- GPT-3是由在旧金山的人工智能公司OpenAI训练与开发，模型设计基于谷歌开发的变换语言模型。
- GPT-3的神经网络包含1750亿个参数，在发布时为参数最多的神经网络模型。
- OpenAI于2020年5月发表GPT-3的论文，在次月为少量公司与开发团队发布应用程序界面的测试版。
- 微软在2020年9月22日宣布取得了GPT-3的独家授权。

GPT3训练数据来源

Dataset	Tokens (billion)	Assumptions	Tokens per byte (Tokens / bytes)	Ratio	Size (GB)
Web data	**410B**	–	*0.71*	*1:1.9*	**570**
WebText2	**19B**	*25% > WebText*	*0.38*	*1:2.6*	*50*
Books1	**12B**	*Gutenberg*	*0.57*	*1:1.75*	*21*
Books2	**55B**	*Bibliotik*	*0.54*	*1:1.84*	*101*
Wikipedia	**3B**	*See RoBERTa*	*0.26*	*1:3.8*	*11.4*
Total	**499B**			*753.4GB*	

Table. GPT-3 Datasets. Disclosed in **bold**. Determined in *italics*.

Alan D. Thompson, GPT-3.5 + ChatGPT: An illustrated overview, https://lifearchitect.ai/chatgpt/

GPT3训练成本

看一下大语言模型训练的token数量：

- GPT-3（2020.5）是500B（5000亿），目前最新数据为止；
- Google的PaLM（2022.4）是780B；
- DeepMind的Chinchilla是1400B；
- Pangu-α公布了训练的token数，约为40B，不到GPT-3的十分之一；
- 国内其他的大模型都没有公布训练的token数。

GPT-3, In-context learning, and very large models

- 目前，我们与PLM打交道有两种方法：
  - 从它们定义的概率分布中进行采样（prompt learning）
  - 在一个下游任务上微调，利用它们的预测结果
- 意外收获：非常大的语言模型似乎在执行下游任务的学习时，不需要任何梯度步骤，只需在其上下文中提供一些示例。
- GPT3
  - T5：11 billion 参数
  - GPT3：175 billion 参数

GPT-3, In-context learning, and very large models

- The in-context examples seem to specify the task to be performed, and the conditional distribution mocks performing

- Input (prefix within a single Transformer decoder context):
  - “ thanks -> merci
  - hello -> bonjour
  - mint -> menthe
  - otter -> ”
- Output (conditional generations):
  - loutre…”

GPT-3, In-context learning, and very large models

The three settings we explore for in-context learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
cheese =>                           ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← example
cheese =>                           ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:        ← task description
sea otter => loutre de mer          ← examples
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>                           ← prompt
```

PLM summary

- 我们学习了 GPT-X, BERT, T5 以及其他的大型PLM
- 正在爆火的 in-context learning 依然没有被人们研究透!
- BERT这种“小”模型已经变成了很多任务的基础架构
- 依然有很多待研究的课题：
  - Bias, toxicity, and fairness
  - Retrieval Augmented Language Models + Knowledge
  - Scaling Laws
- 工具包： Hugging Face Transformers
  - Python package name： `import transformers`

Thank you