



北京航空航天大學  
BEIHANG UNIVERSITY

# 自然语言处理

人工智能研究院

主讲教师 沙磊



# Transformers

# 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](mailto:avaswani@google.com)

**Noam Shazeer\***  
Google Brain  
[noam@google.com](mailto:noam@google.com)

**Niki Parmar\***  
Google Research  
[nikip@google.com](mailto:nikip@google.com)

**Jakob Uszkoreit\***  
Google Research  
[usz@google.com](mailto:usz@google.com)

**Llion Jones\***  
Google Research  
[llion@google.com](mailto:llion@google.com)

**Aidan N. Gomez\* †**  
University of Toronto  
[aidan@cs.toronto.edu](mailto:aidan@cs.toronto.edu)

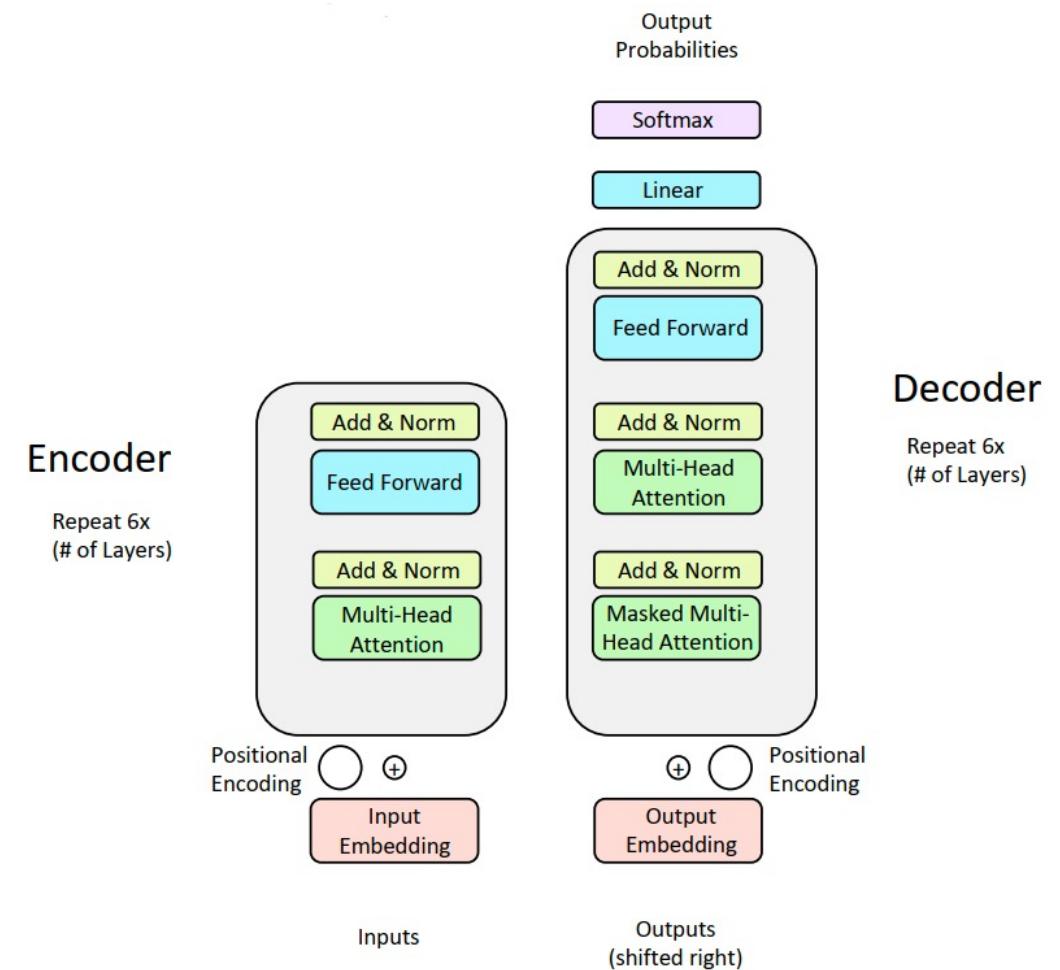
**Łukasz Kaiser\***  
Google Brain  
[lukaszkaiser@google.com](mailto:lukaszkaiser@google.com)

**Illia Polosukhin\* ‡**  
[illia.polosukhin@gmail.com](mailto: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	<b>41.29</b>	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1		<b><math>3.3 \cdot 10^{18}</math></b>
Transformer (big)	<b>28.4</b>	<b>41.8</b>		$2.3 \cdot 10^{19}$

# 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
<i>seq2seq-attention, L = 500</i>	5.04952	12.7
<i>Transformer-ED, L = 500</i>	2.46645	34.2
<i>Transformer-D, L = 4000</i>	2.22216	33.6
<i>Transformer-DMCA, no MoE-layer, L = 11000</i>	2.05159	36.2
<i>Transformer-DMCA, MoE-128, L = 11000</i>	1.92871	37.9
<i>Transformer-DMCA, MoE-256, L = 7500</i>	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

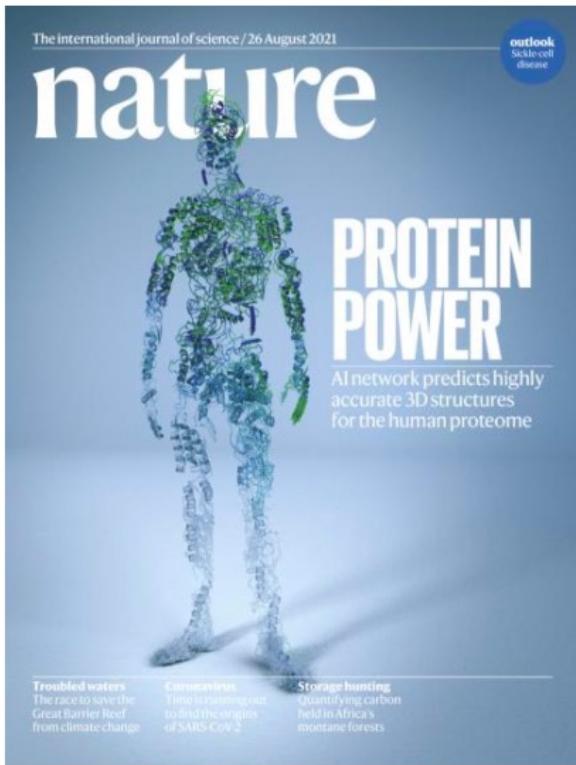


All top models are  
Transformer (and  
pretraining)-based

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



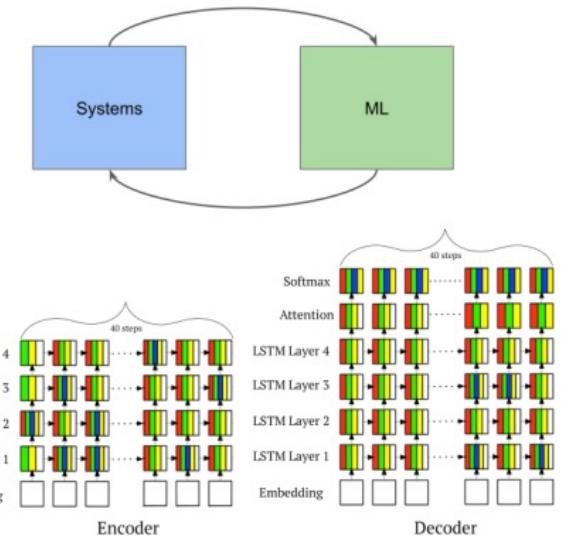
[Jumper et al. 2021] aka AlphaFold2!



## 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	<b>88.55</b> ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4 / 88.5*
ImageNet ReaL	<b>90.72</b> ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	<b>99.50</b> ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	—
CIFAR-100	<b>94.55</b> ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	—
Oxford-IIIT Pets	<b>97.56</b> ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	—
Oxford Flowers-102	99.68 ± 0.02	<b>99.74</b> ± 0.00	99.61 ± 0.02	99.63 ± 0.03	—
VTAB (19 tasks)	<b>77.63</b> ± 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



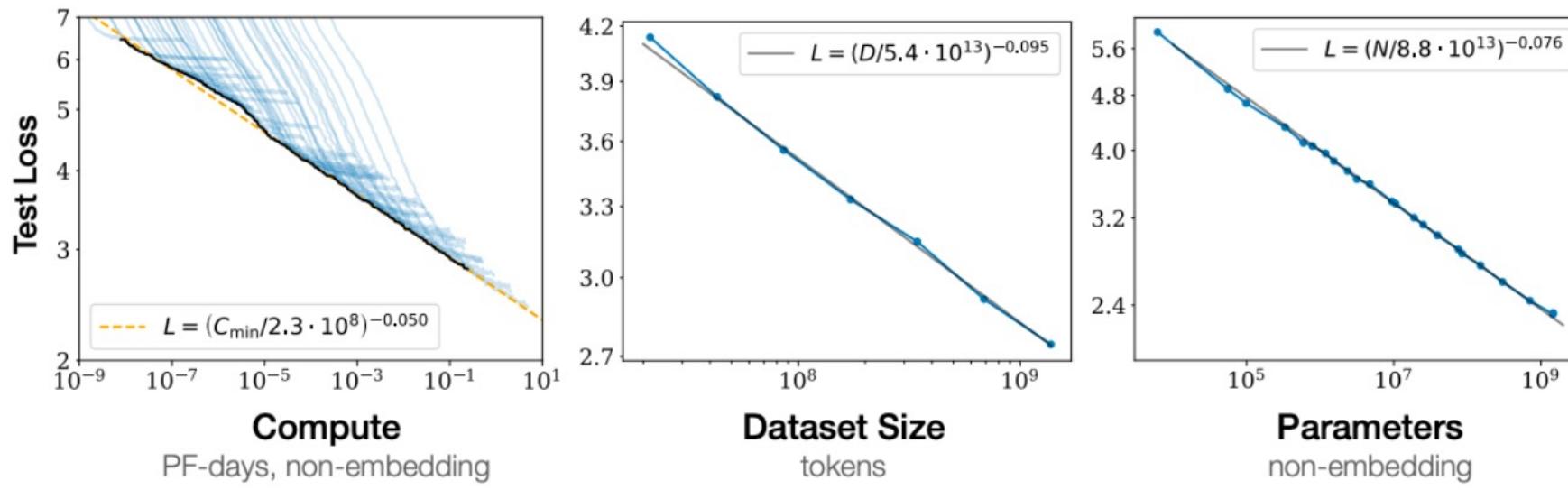
## ML for Systems

[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	0.644	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
8-layer GNMT (8)	0.440	0.562	0.600	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	0.40M	0.259	17.4% / 12.6%	26.7x
8-layer Transformer-XL (8)	0.350	0.46	0.60M	0.425	23.9% / 16.7%	16.7x
2-stack 18-layer WaveNet (2)	0.229	0.312	0.60M	0.301	26.6% / 23.9%	13.5x
Inception (2) b64	0.423	0.731	0.60M	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	0.60M	0.354	18.6% / 11.7%	6.67x
4-stack 36-layer WaveNet (4)	0.659	0.988	0.60M	0.721	50% / 9.4%	20x
GEOOMEAN	-	-	-	-	<b>20.5% / 18.2%</b>	<b>15x</b>

# 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？



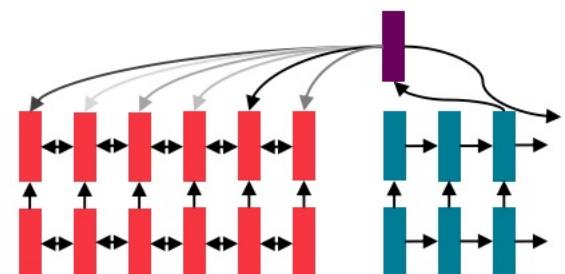
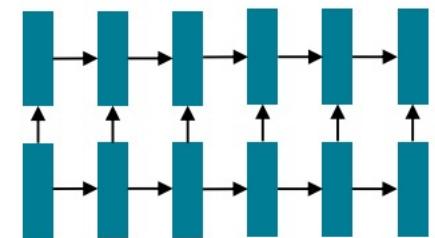
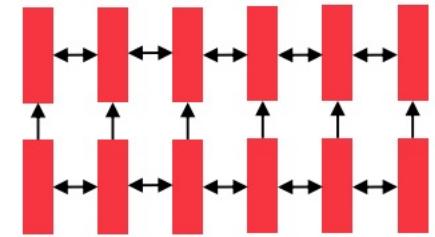
[Kaplan et al., 2020]  
11

# 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

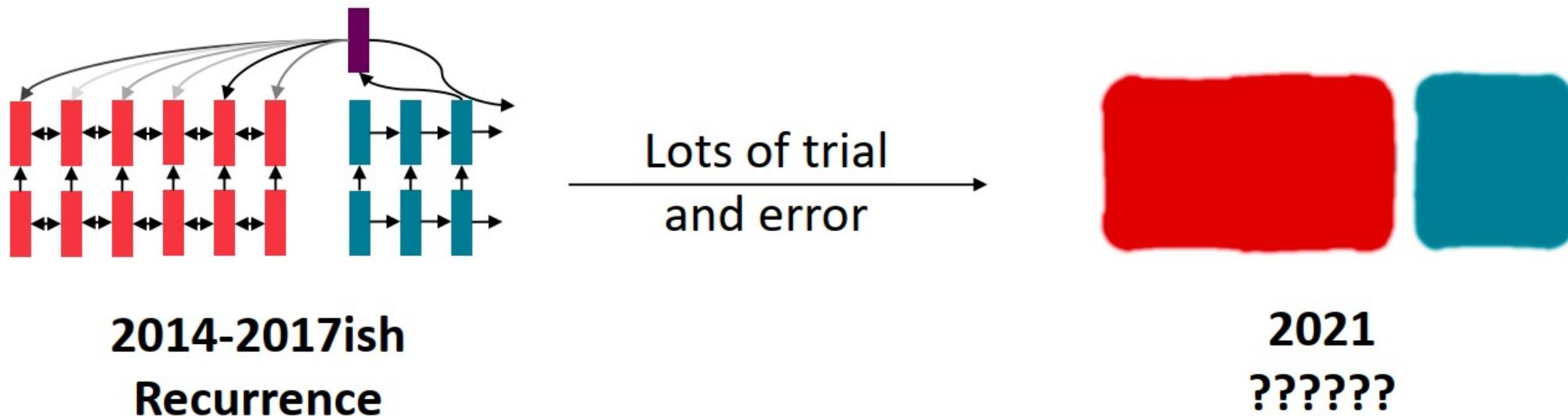
# Recurrent models for (most) NLP!

- 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



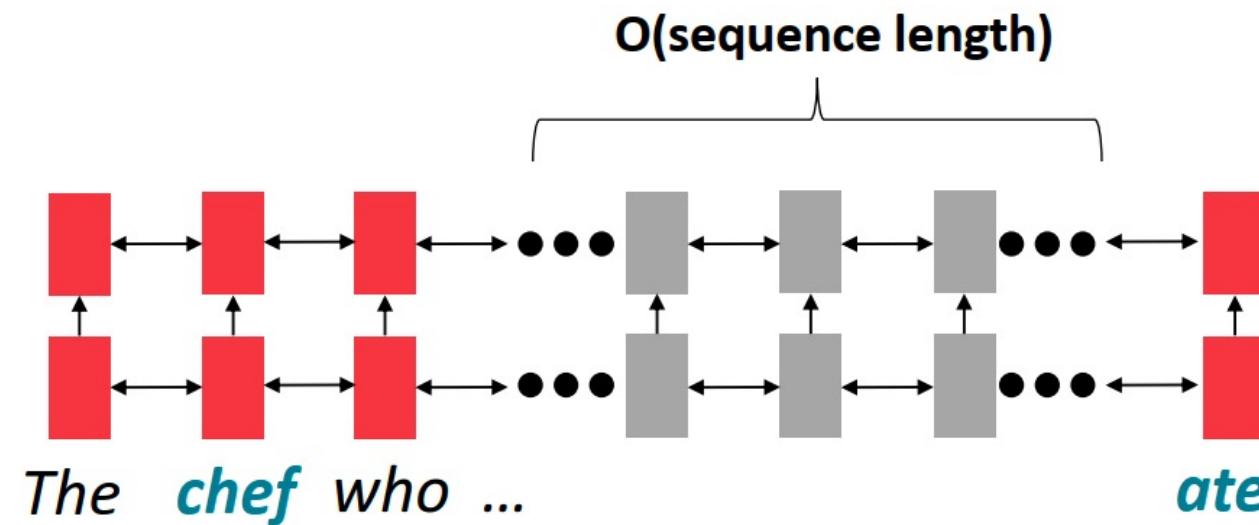
# Today: Same goals, different building blocks

- 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.



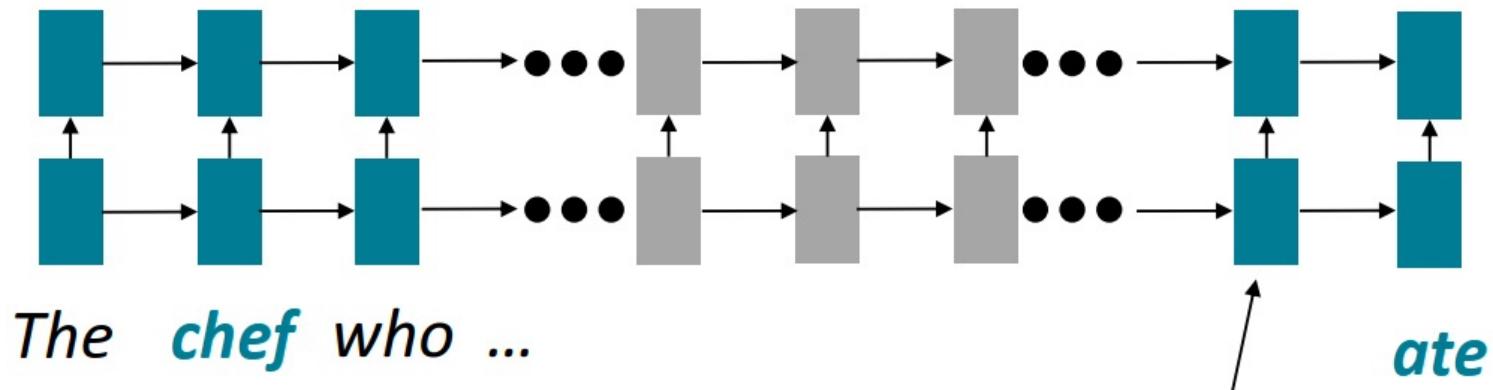
# Issues with recurrent models: Linear interaction distance

- RNNs are unrolled “left-to-right”.
- It encodes linear locality: a useful heuristic!
  - Nearby words often affect each other’s meanings
- Problem: RNNs take  $O(\text{sequence length})$  steps for distant word pairs to interact.



# Issues with recurrent models: Linear interaction distance

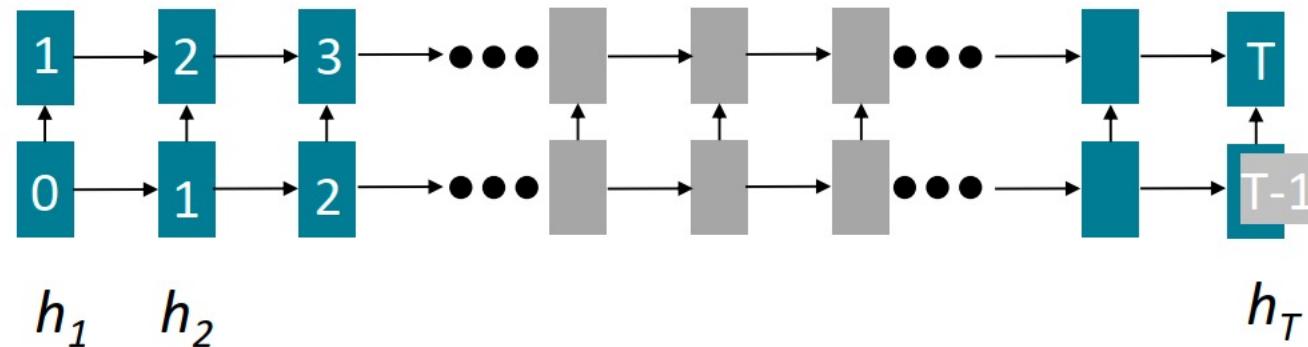
- 远距离的词的交互需要  $O(\text{sequence length})$  步，意味着：
  - 长距离依存关系很难学习（梯度消失）
  - 词的线性顺序被模型限定



Info of *chef* has gone through  
 $O(\text{sequence length})$  many layers!

# Issues with recurrent models: Lack of parallelizability

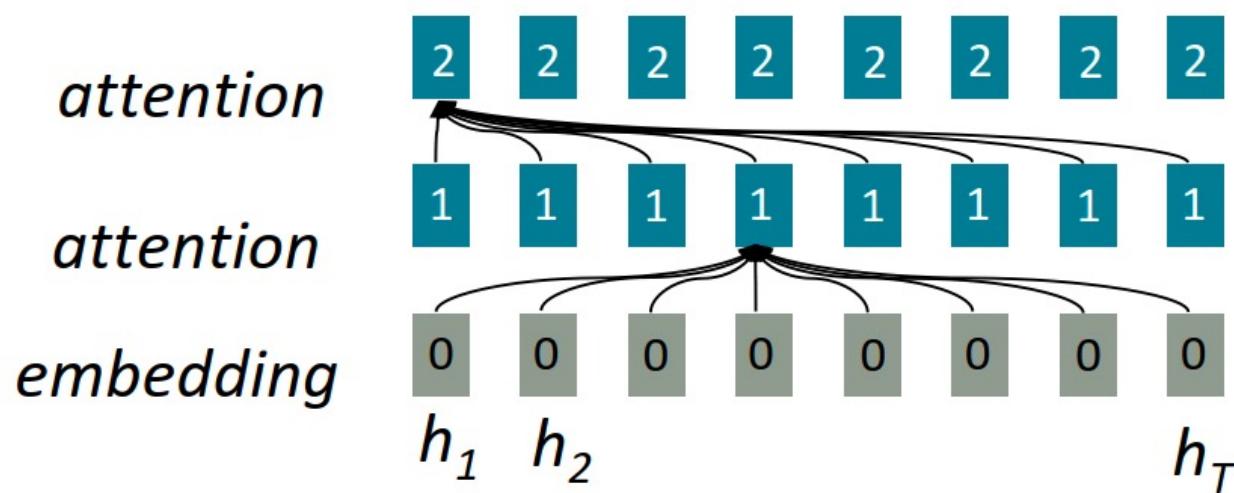
- 前向与后向计算均无法并行实现
  - GPU (或TPU) 可以同时做很多不相关的计算
  - 但未来的RNN隐状态无法在之前的RNN隐状态计算完成之前进行计算
  - 对于大数据集训练很不利
  - 序列长度越长问题越大。 Why? 因为batch就得相应调小



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

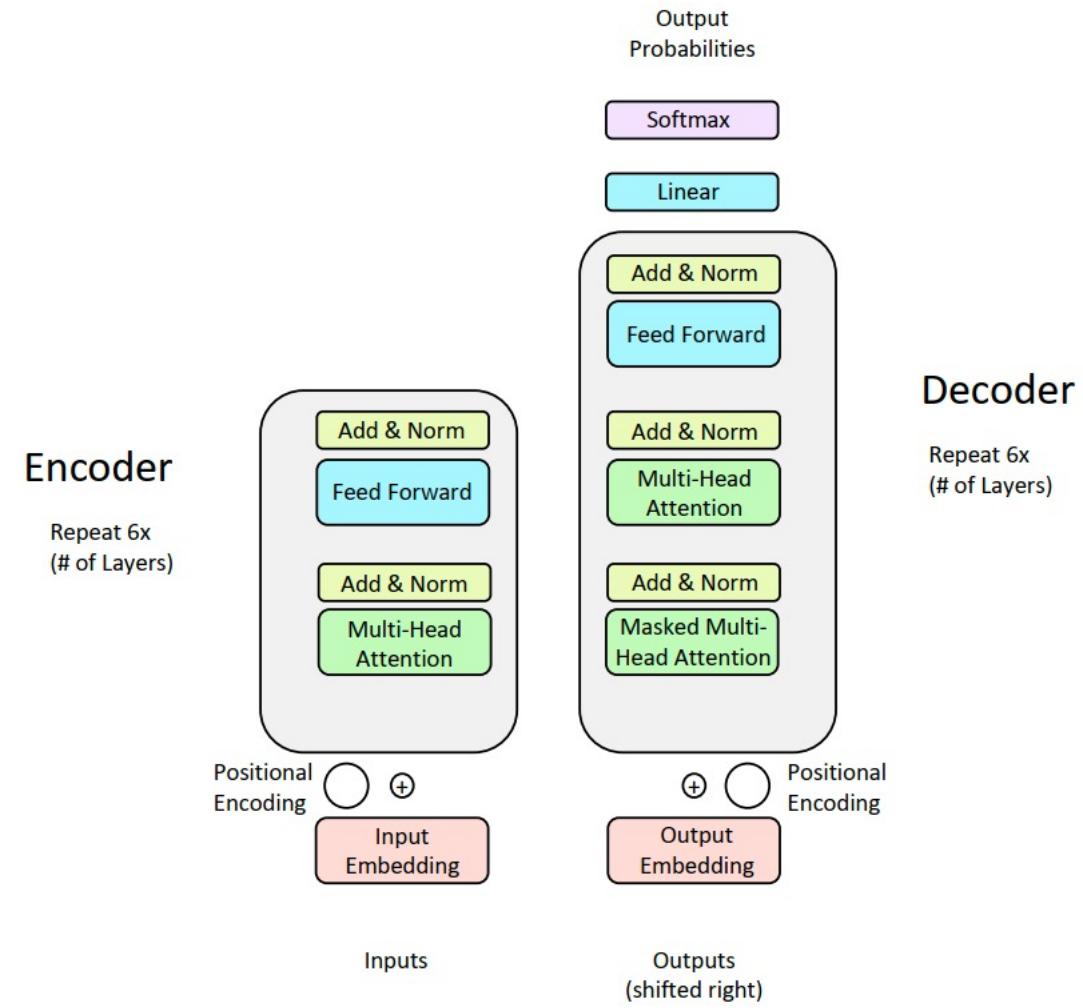
Maximum interact distance  $O(1)$ !  
18

# 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

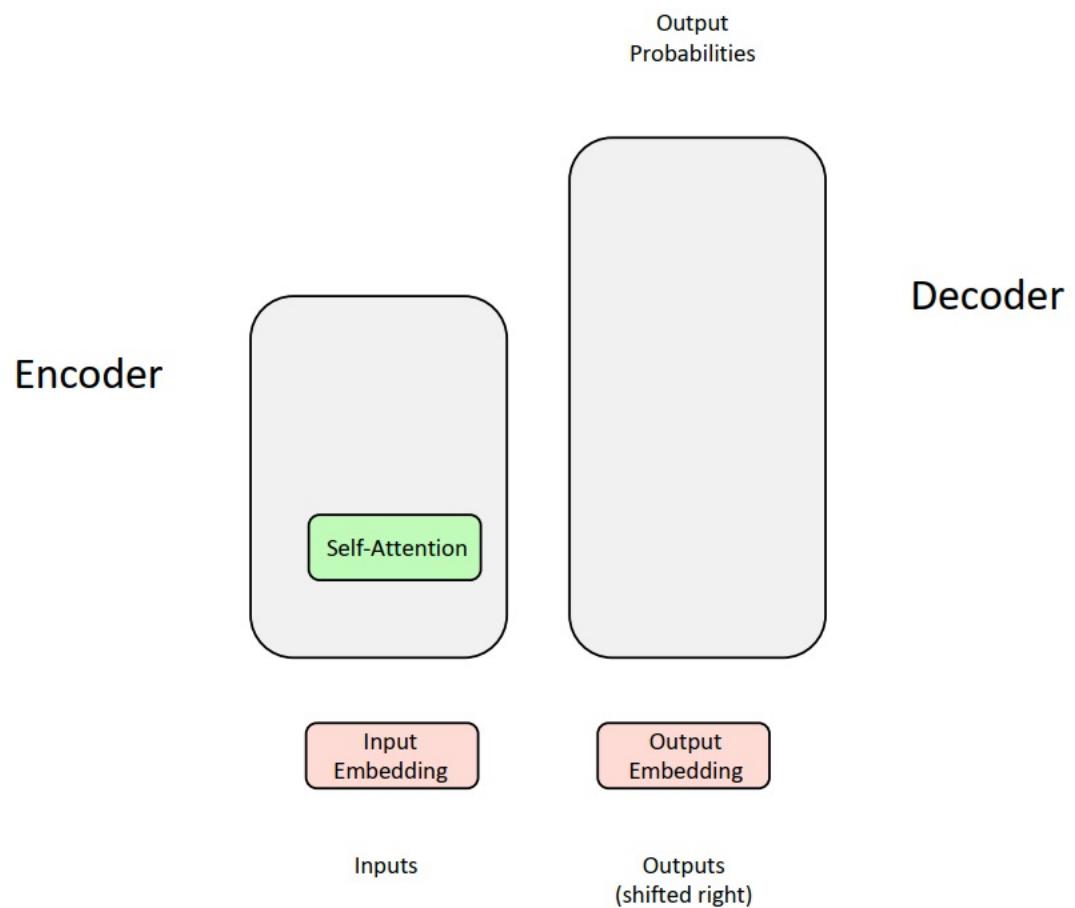
# The Transformer Encoder-Decoder [Vaswani et al., 2017]

- 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)!



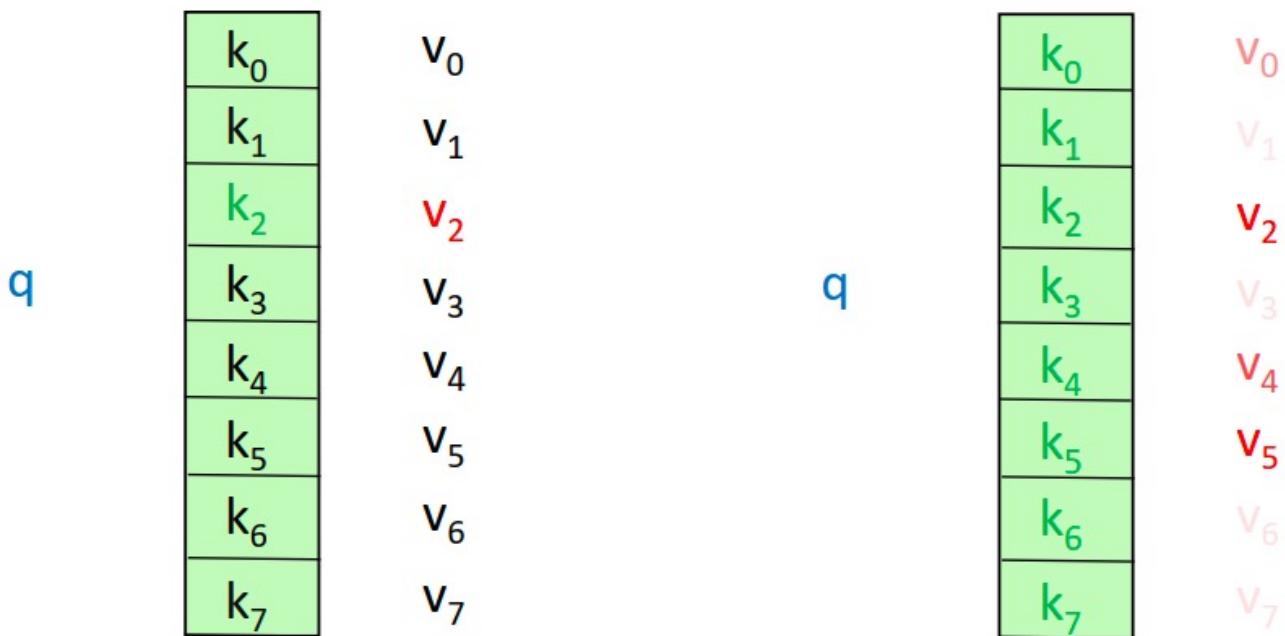
# Encoder: Self-Attention

- 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**.

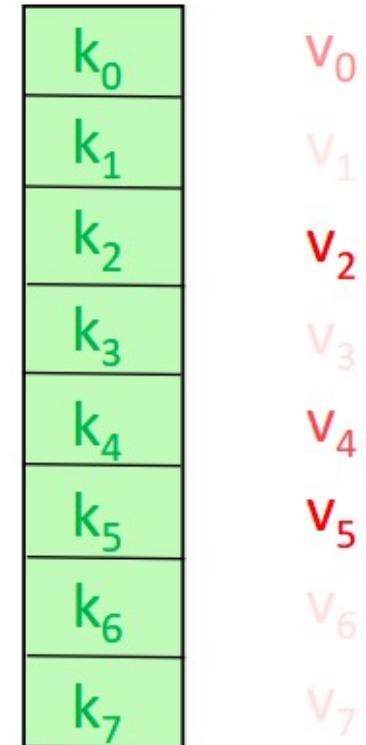
$$e_{ij} = q_i \cdot k_j$$

- 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.

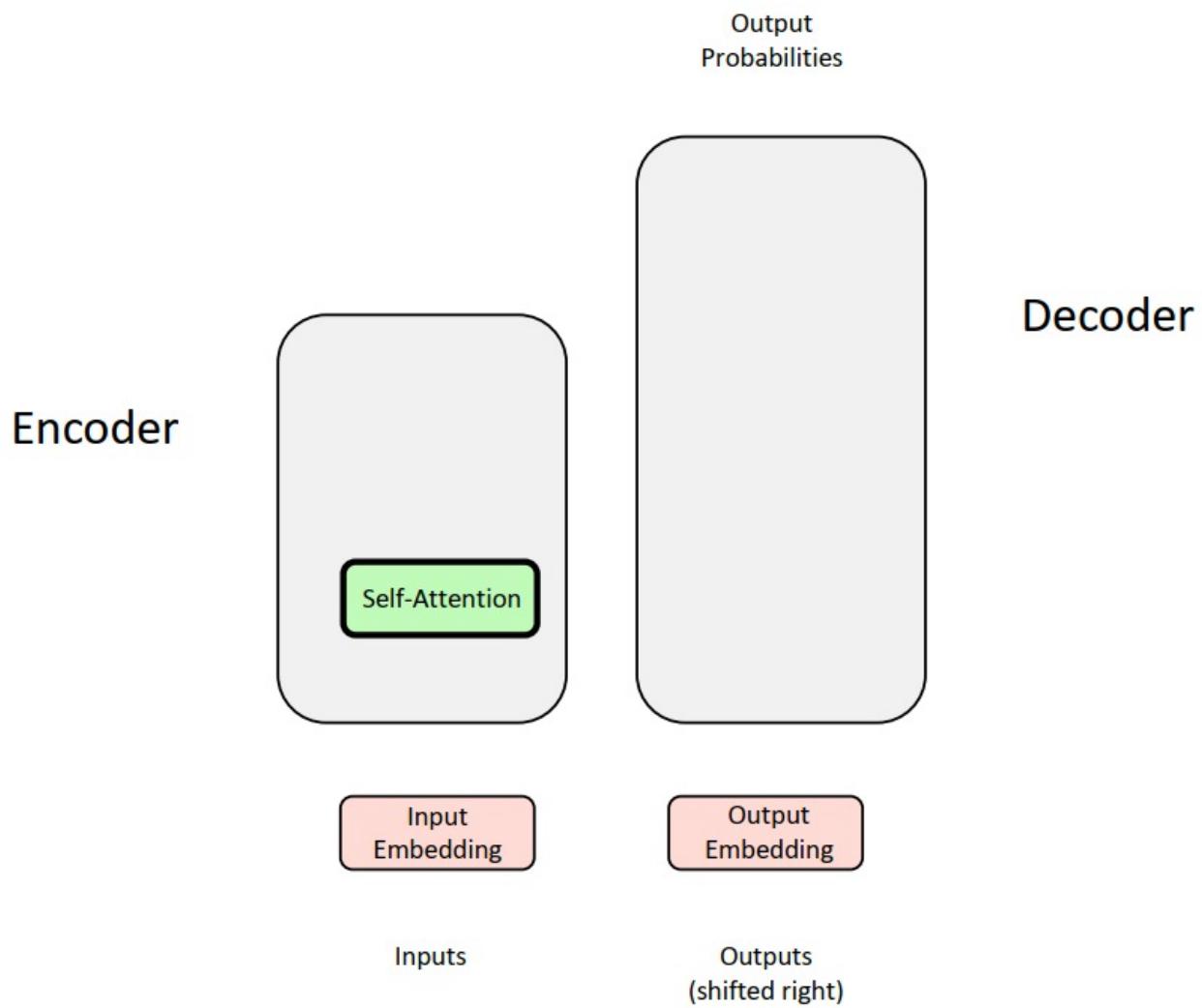
$$A = \text{softmax}(E)$$

- Step 4: Take a weighted sum of **values**

$$\text{Output} = AV$$

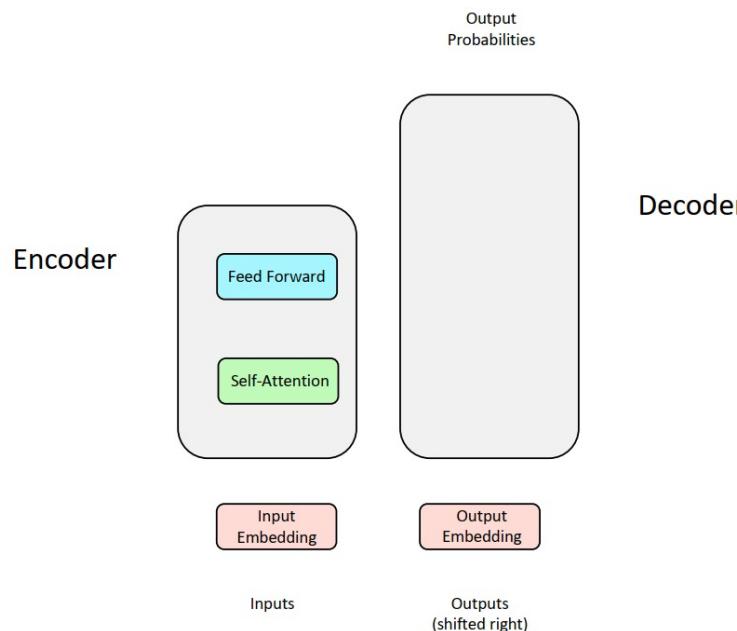
$$\text{Output} = \text{softmax}(QK^T)V$$

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



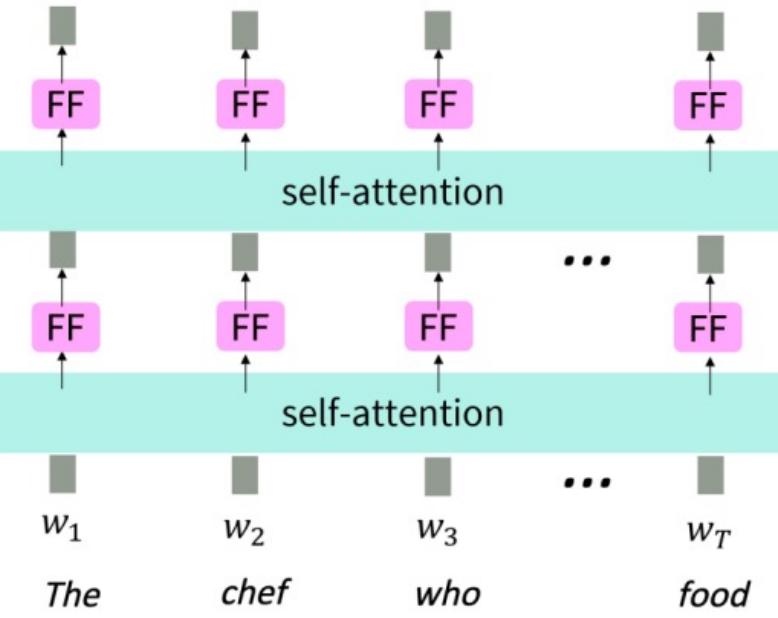
# But attention isn't quite all you need!

- 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

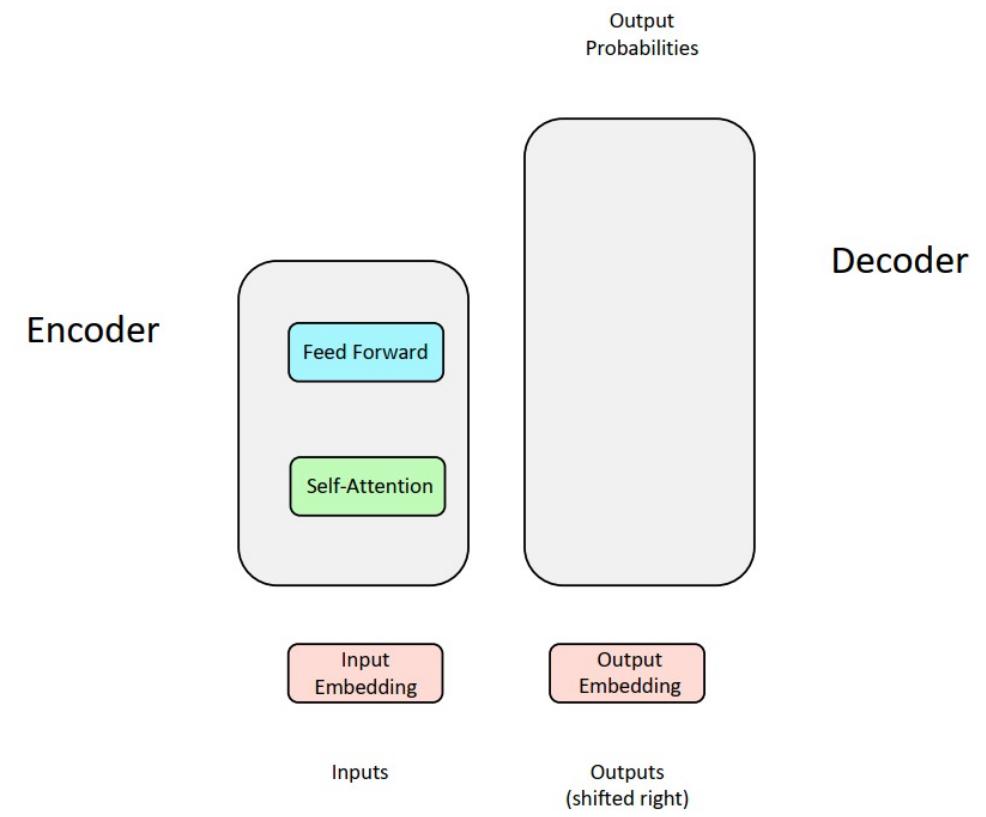
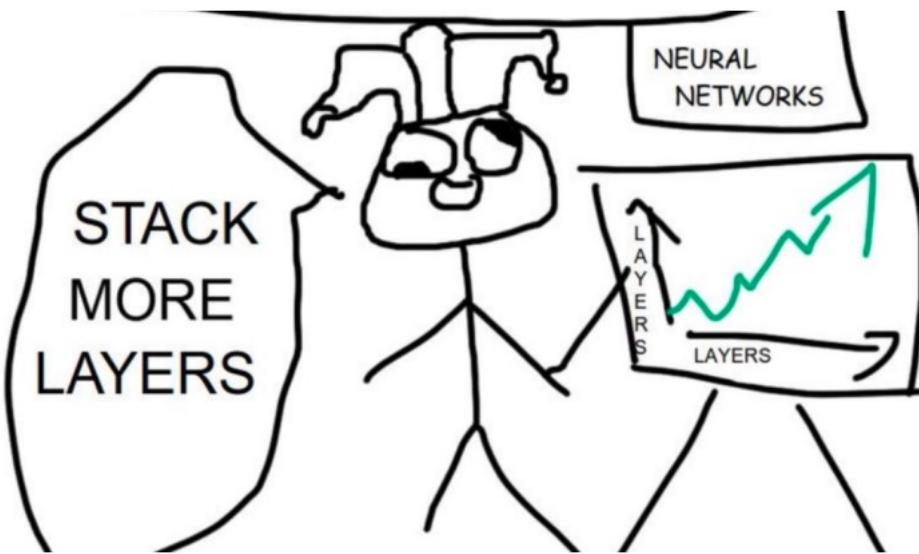


Equation for Feed Forward Layer

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



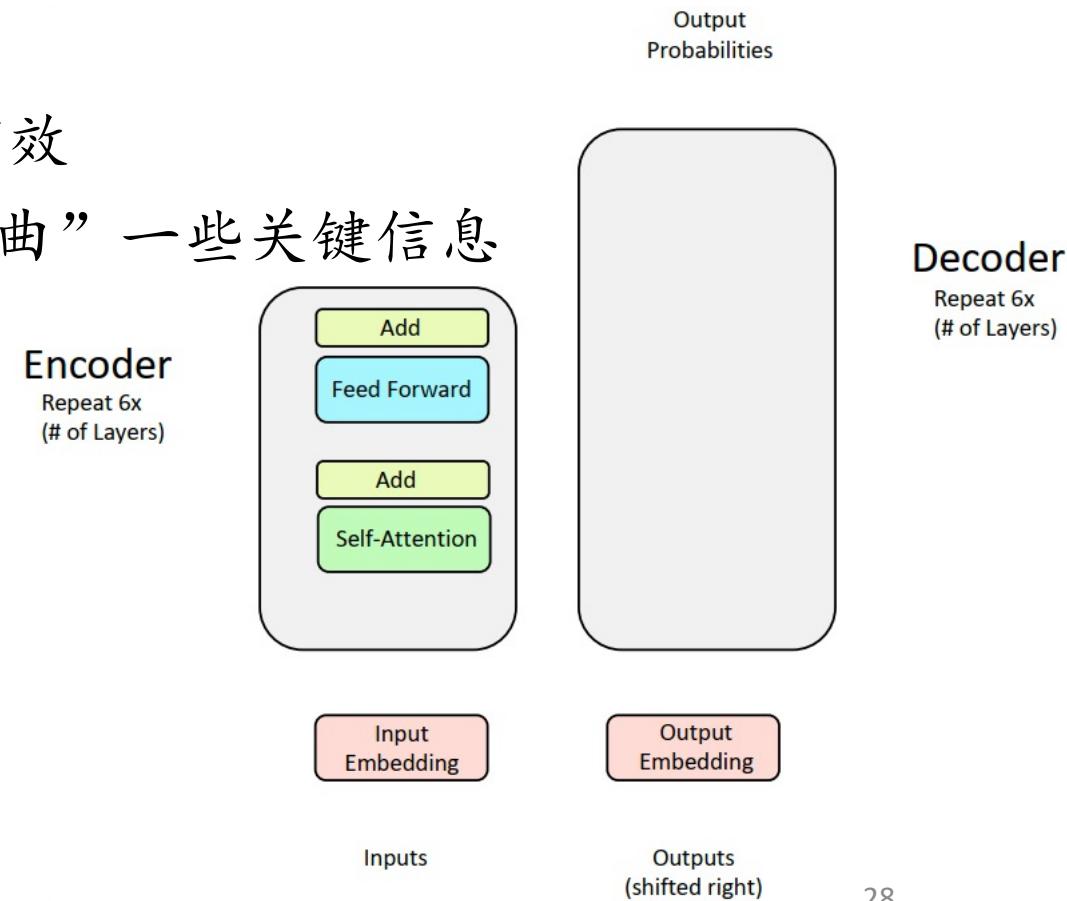
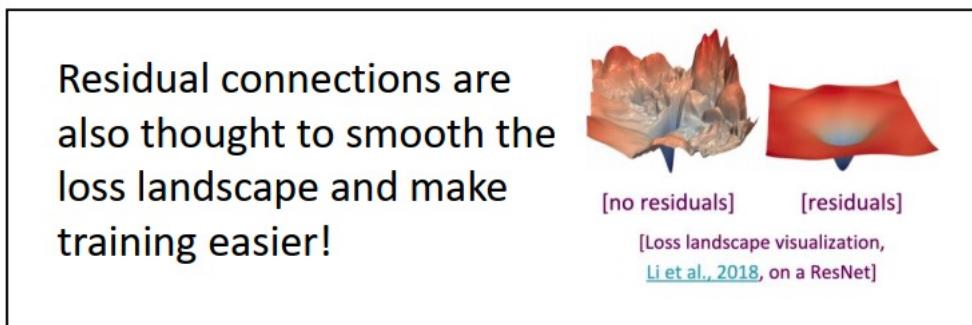
# But how do we make this work for deep networks?



- 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

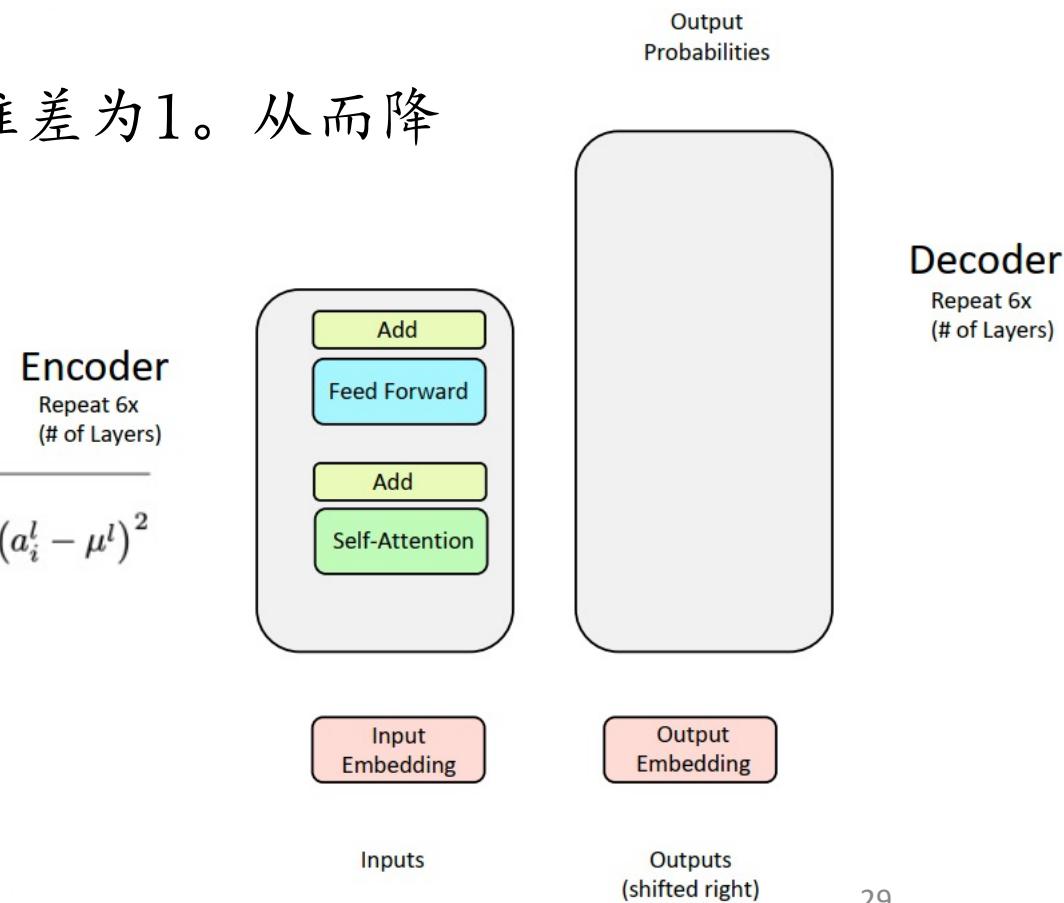


## Training Trick #2: Layer Normalization [Ba et al., 2016]

- 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。

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

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



# Training Trick #3: Scaled Dot Product Attention

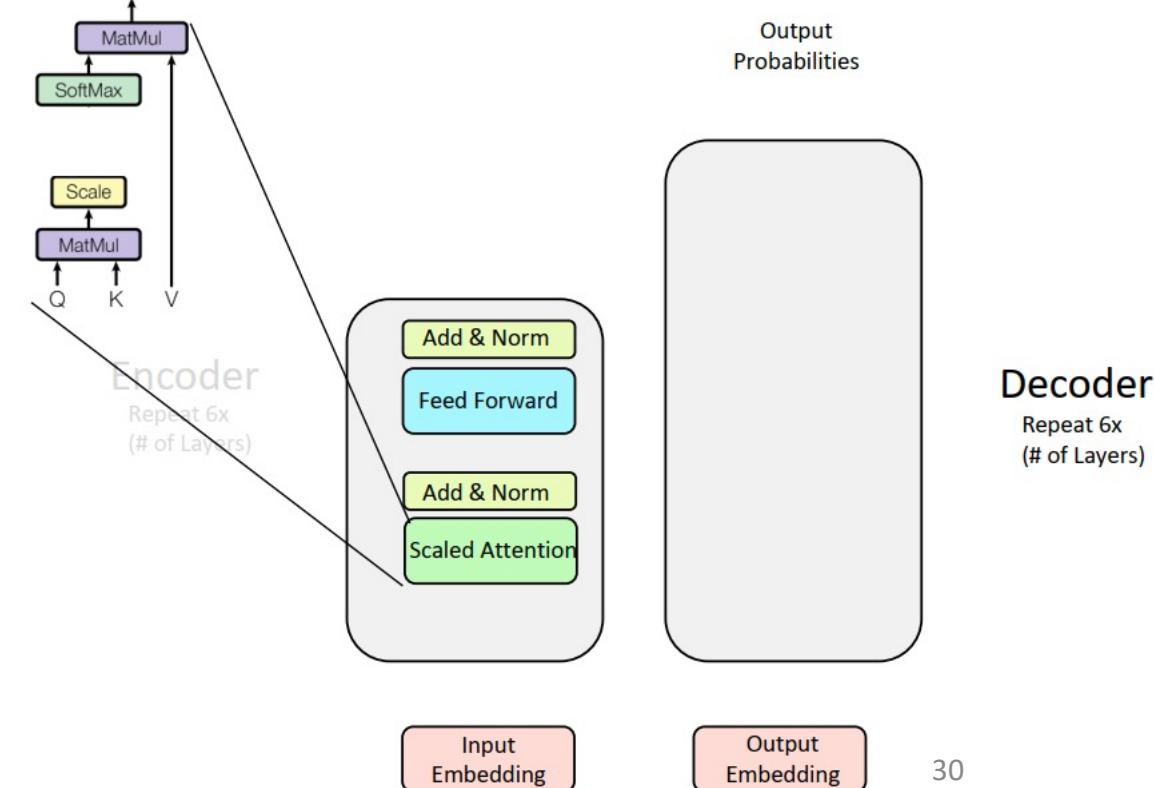
- LayerNorm之后，向量的均值方差被归为了0和1
- 然而，点积会给出极端值，向量越长，越容易出现。方差会因此

## 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:

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



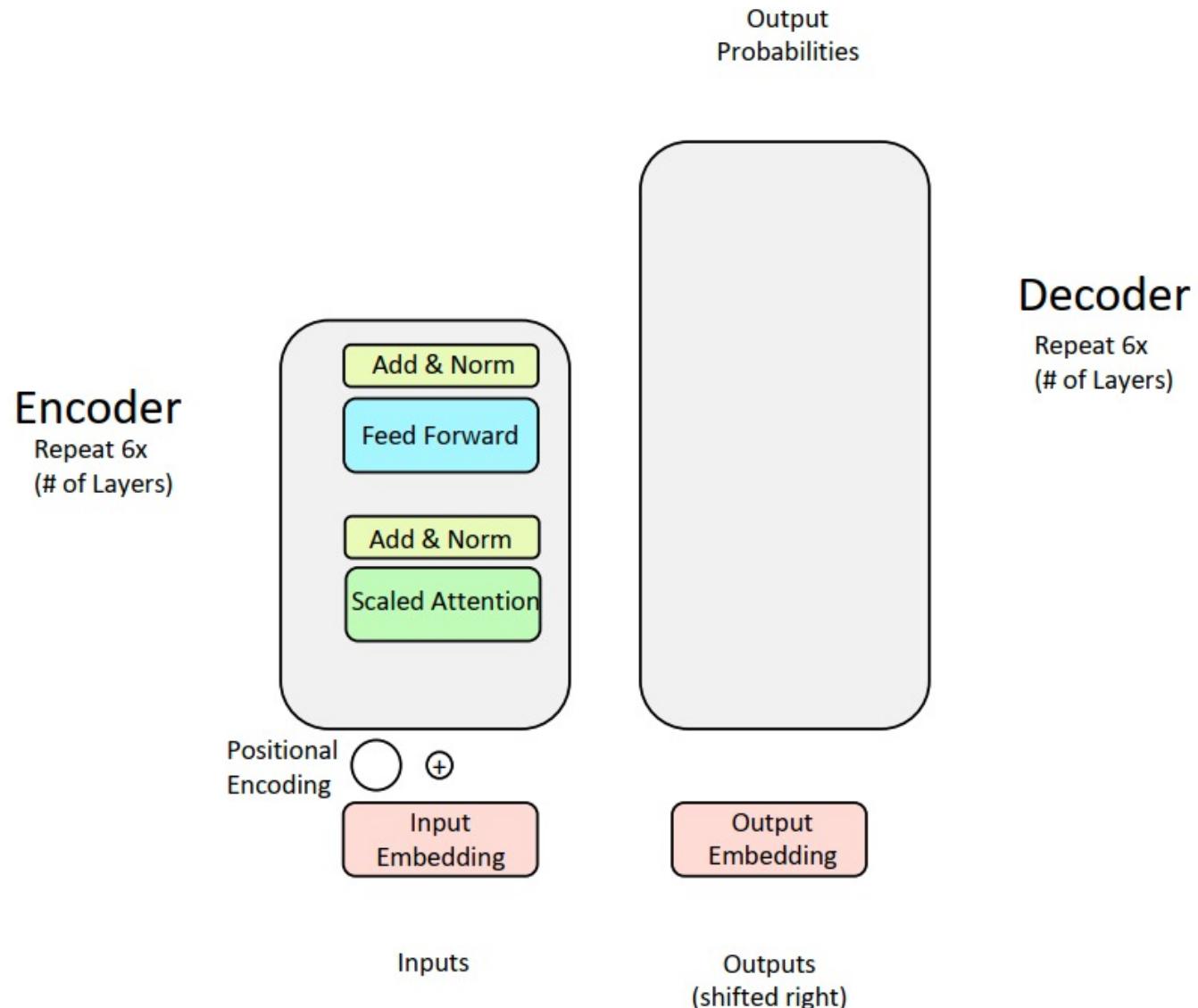
Decoder  
Repeat 6x  
(# of Layers)

# 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。

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

# 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

$p_i \in \mathbb{R}^d$ , for  $i \in \{1, 2, \dots, T\}$  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

$$v_i = \tilde{v}_i + p_i$$

$$q_i = \tilde{q}_i + p_i$$

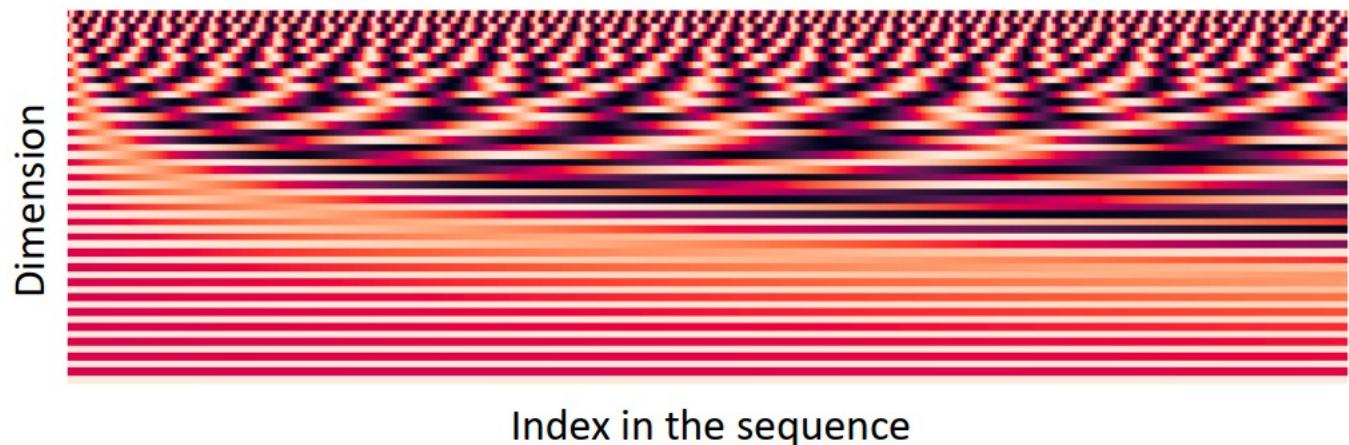
$$k_i = \tilde{k}_i + p_i$$

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:

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



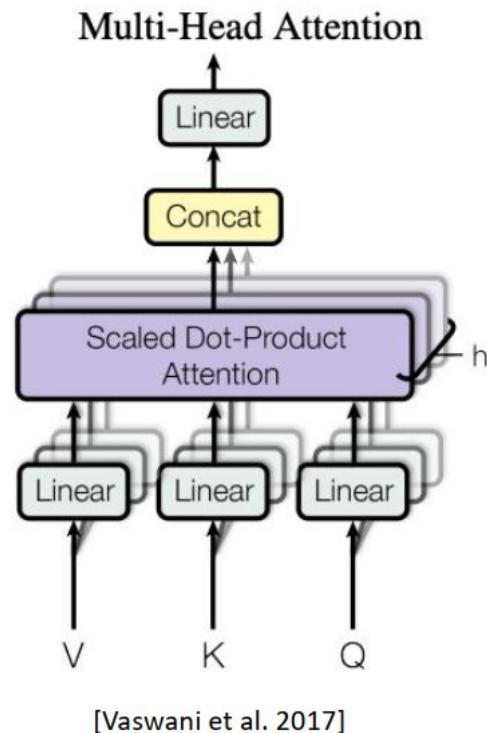
- 优点: (1) 周期性暗指绝对位置不是很重要 (2) 可以扩展到长序列
- 缺点: (1) 不可学习 (2) “扩展”不一定很有用

# 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]

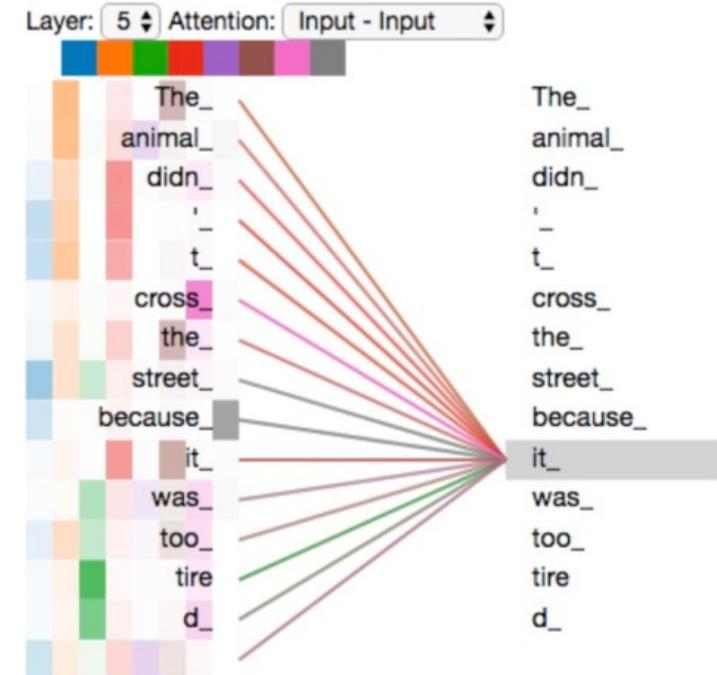


Wizards of the Coast, Artist: Todd Lockwood

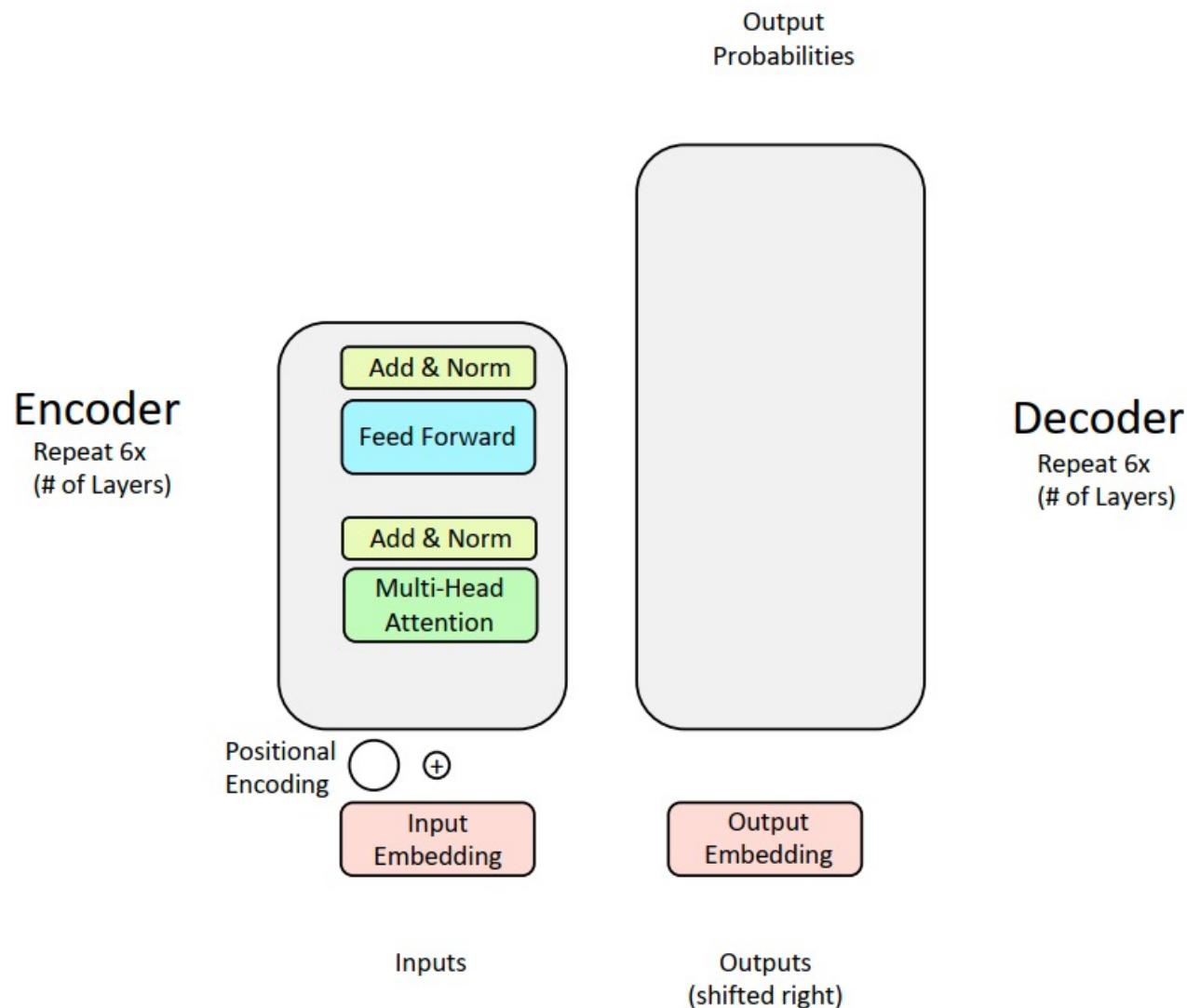
# The Transformer Encoder: Multi-headed Self-Attention

- 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 $x_i^T Q^T K x_j$ 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,  
 $1=1\dots h$
- 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

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



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



# Decoder: Masked Multi-Head Self-Attention

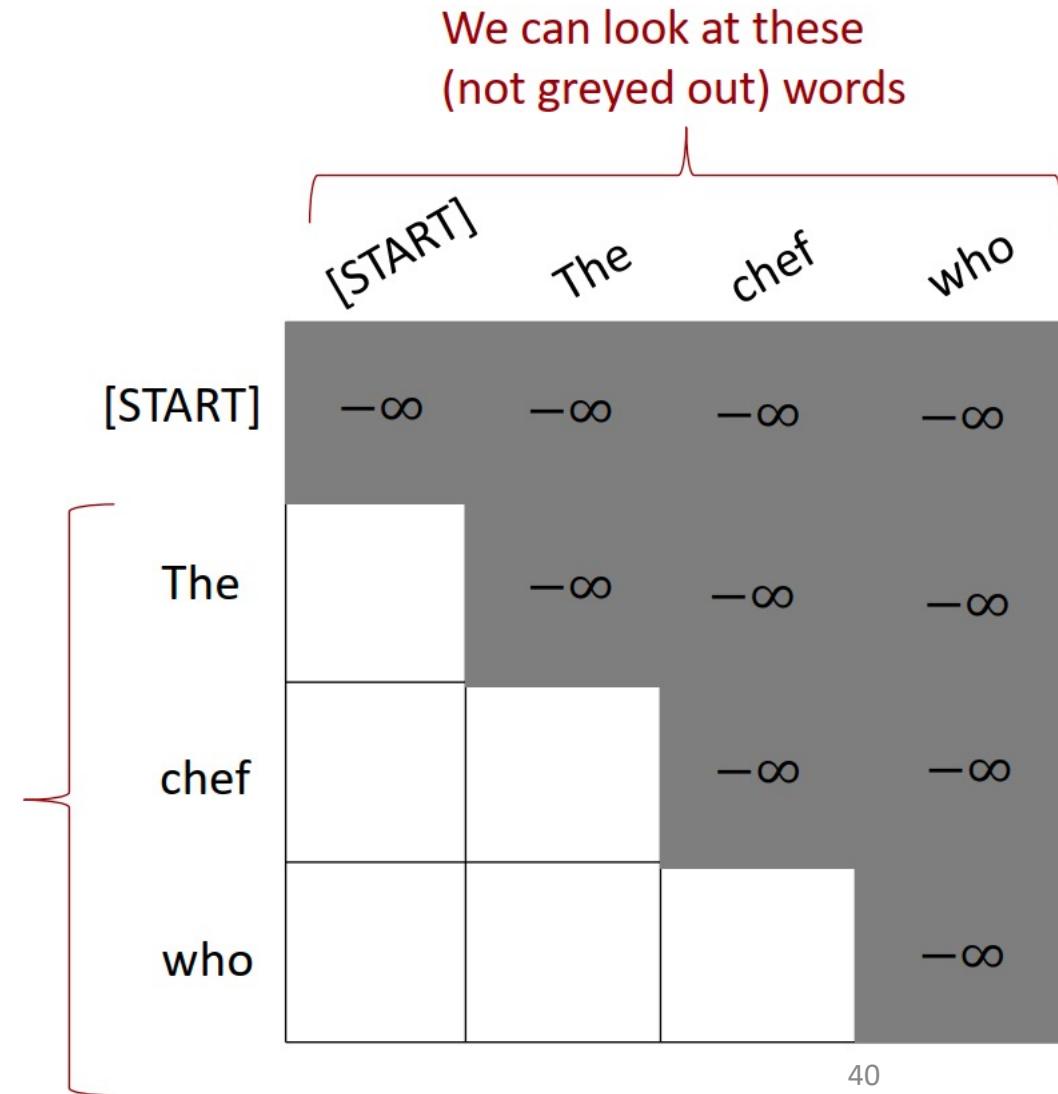
- 问题：
  - Self-attention 可以同时看到所有信息
  - Decode阶段会不会直接看到答案？
- 解答：
  - 将Multi-Head Attention 的未来信息mask掉

# Masking the future in self-attention

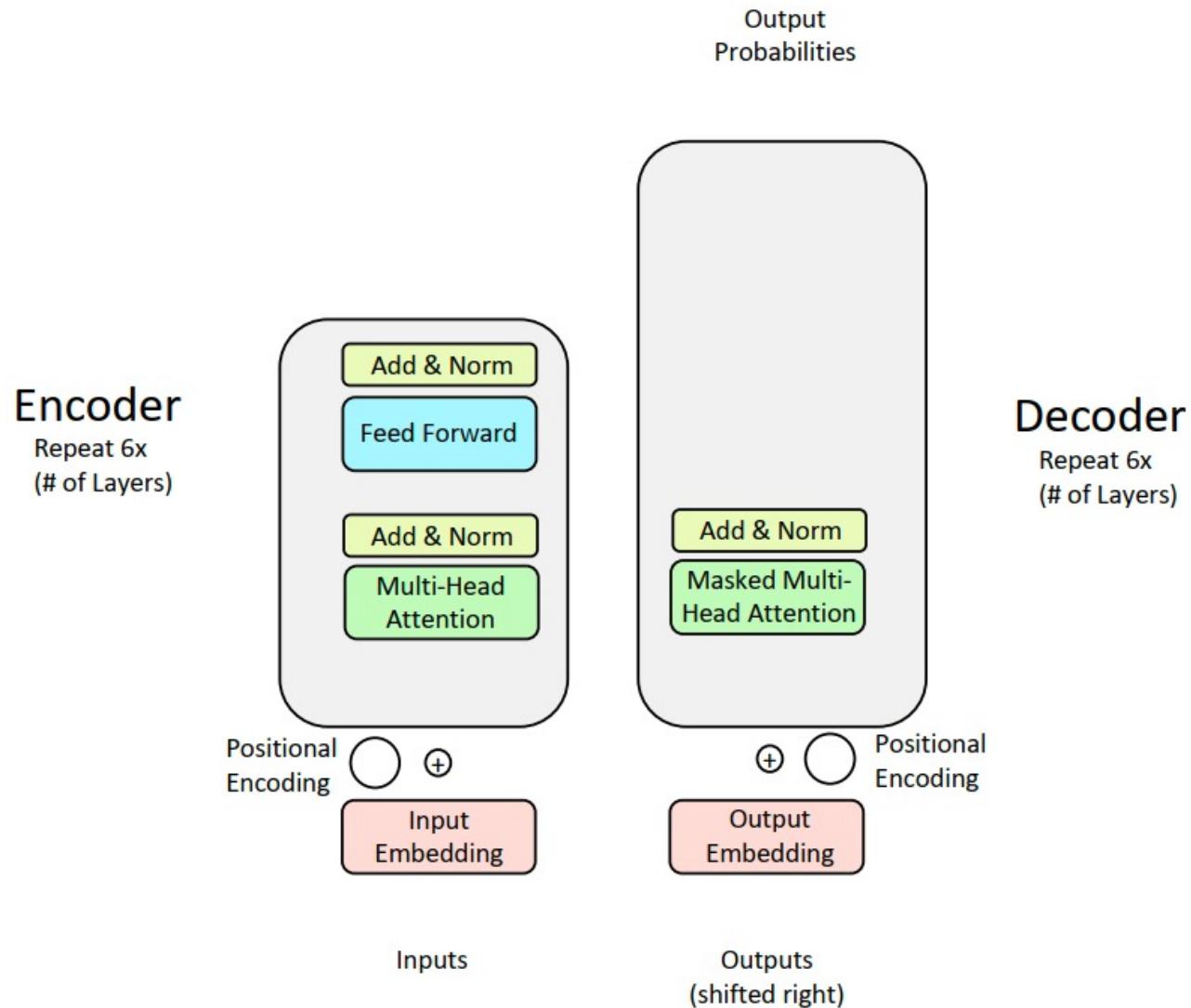
- To use self-attention in decoders, we need to ensure we can't peek at the future.
- To enable parallelization, we mask out attention to future words by setting attention scores to  $-\infty$

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

For encoding  
these words



# Decoder: Masked Multi-Headed Self-Attention



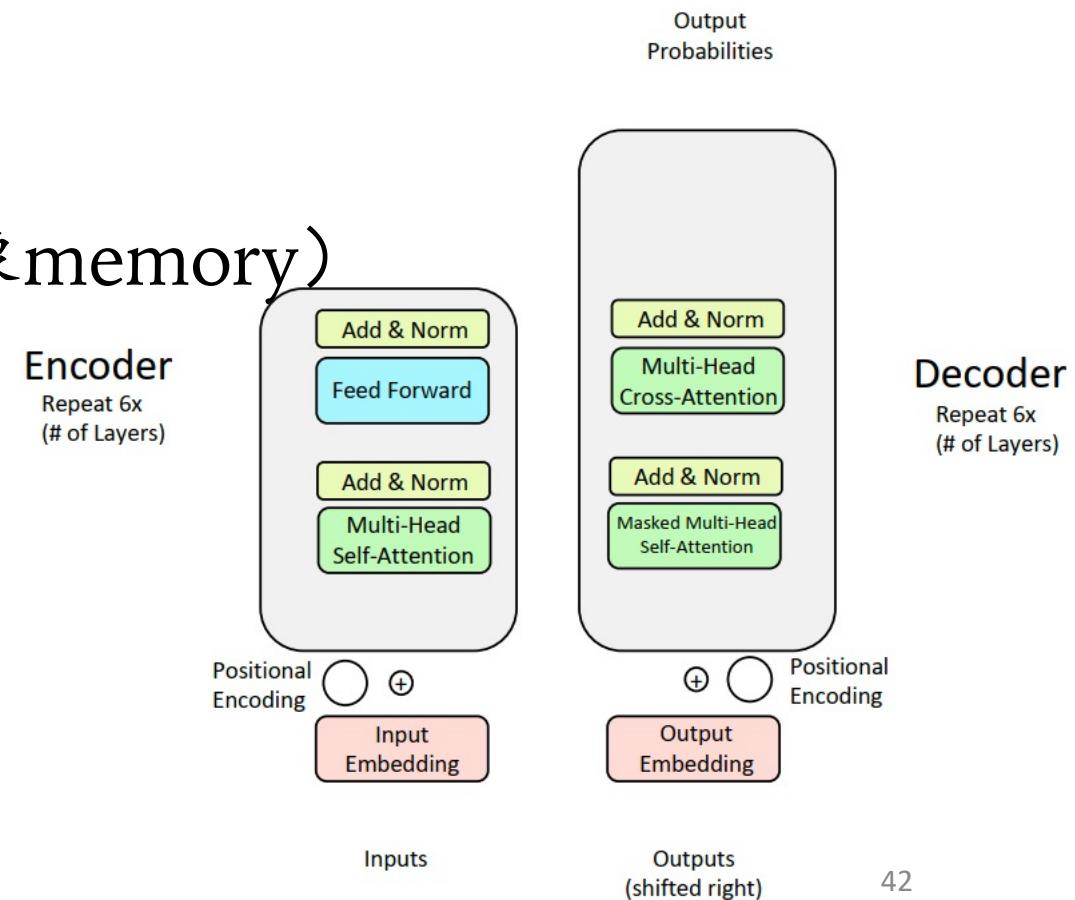
# Encoder-Decoder Attention

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

$$k_i = Kh_i, v_i = Vh_i$$

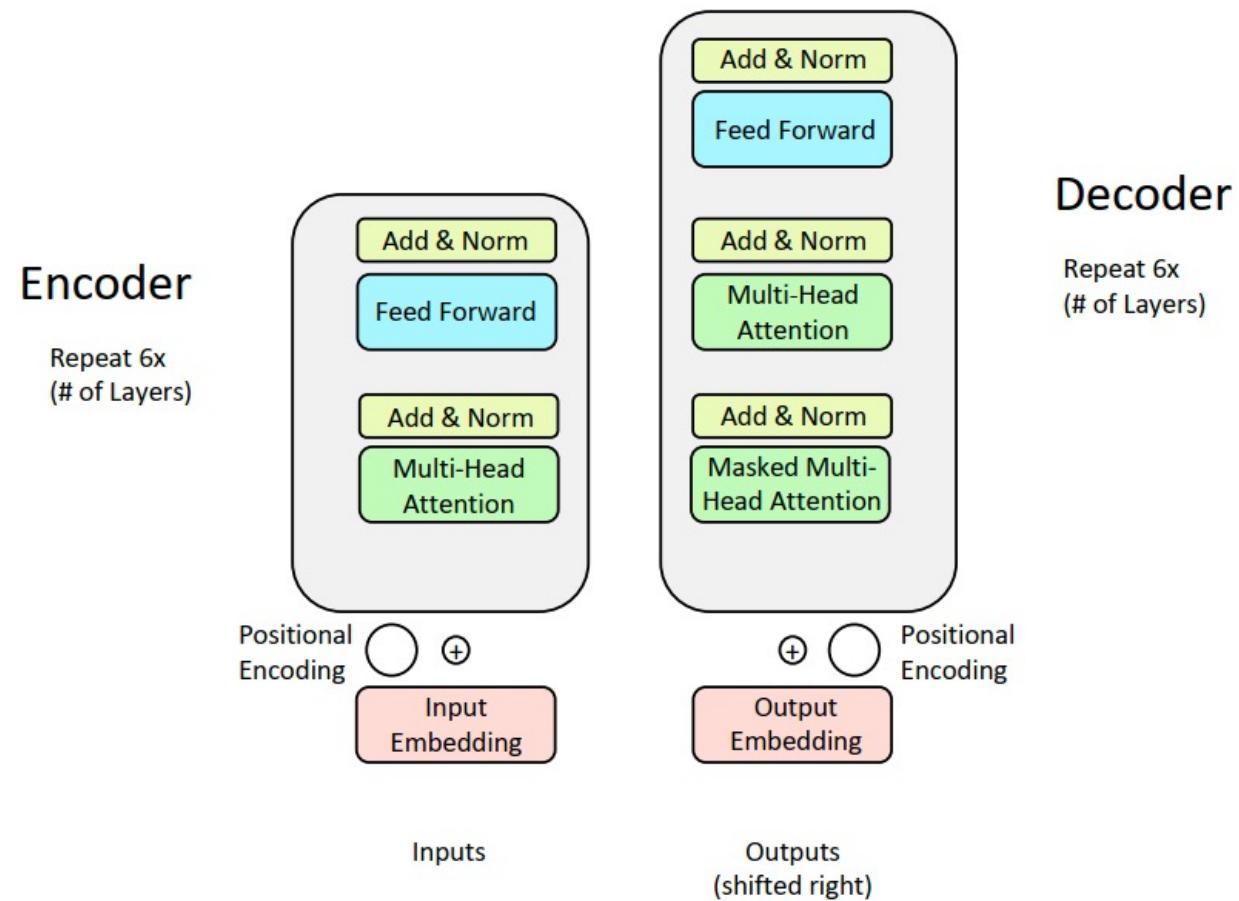
- Query是从decoder中来的

$$q_i = Qz_i$$



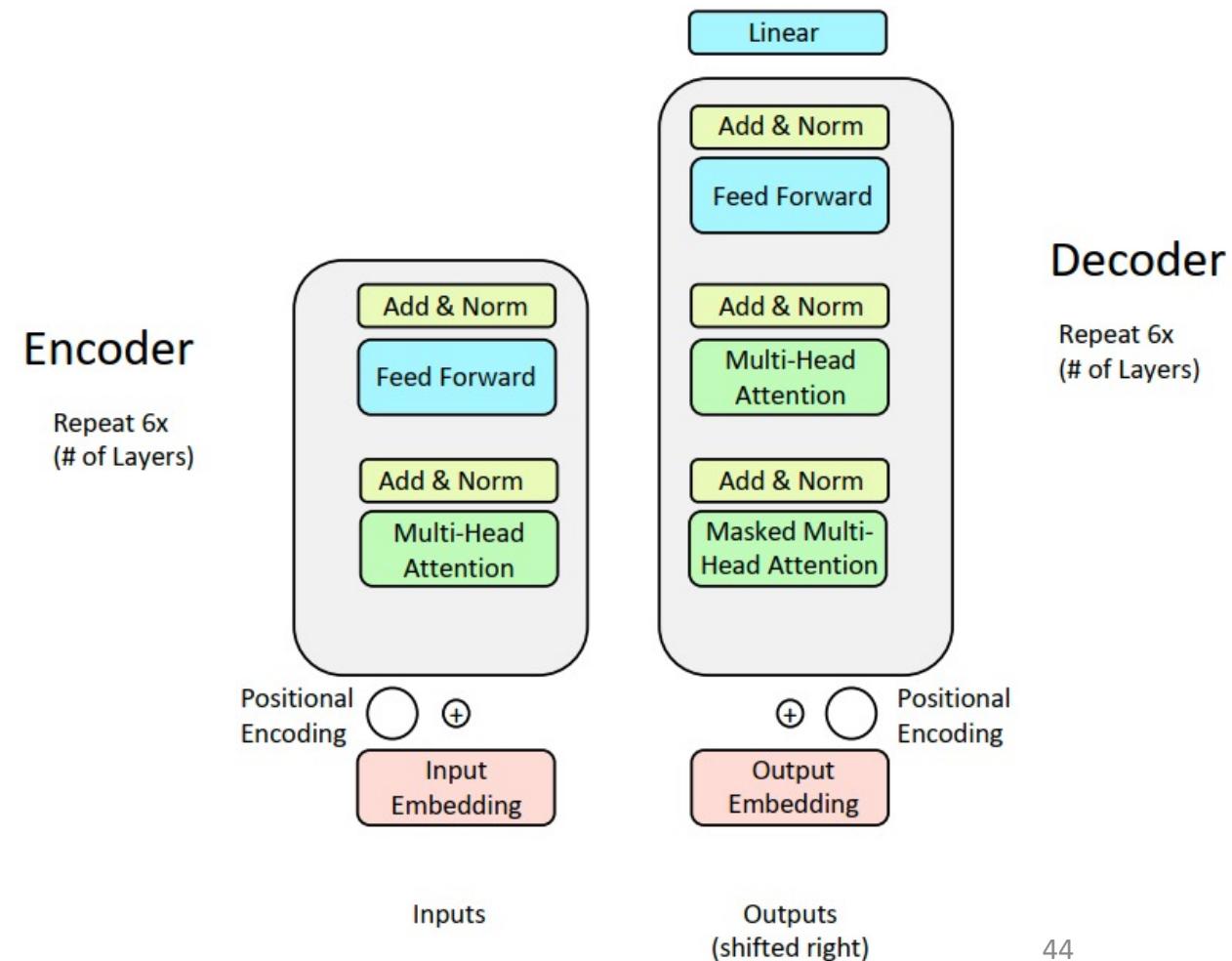
# Decoder: Finishing touches!

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



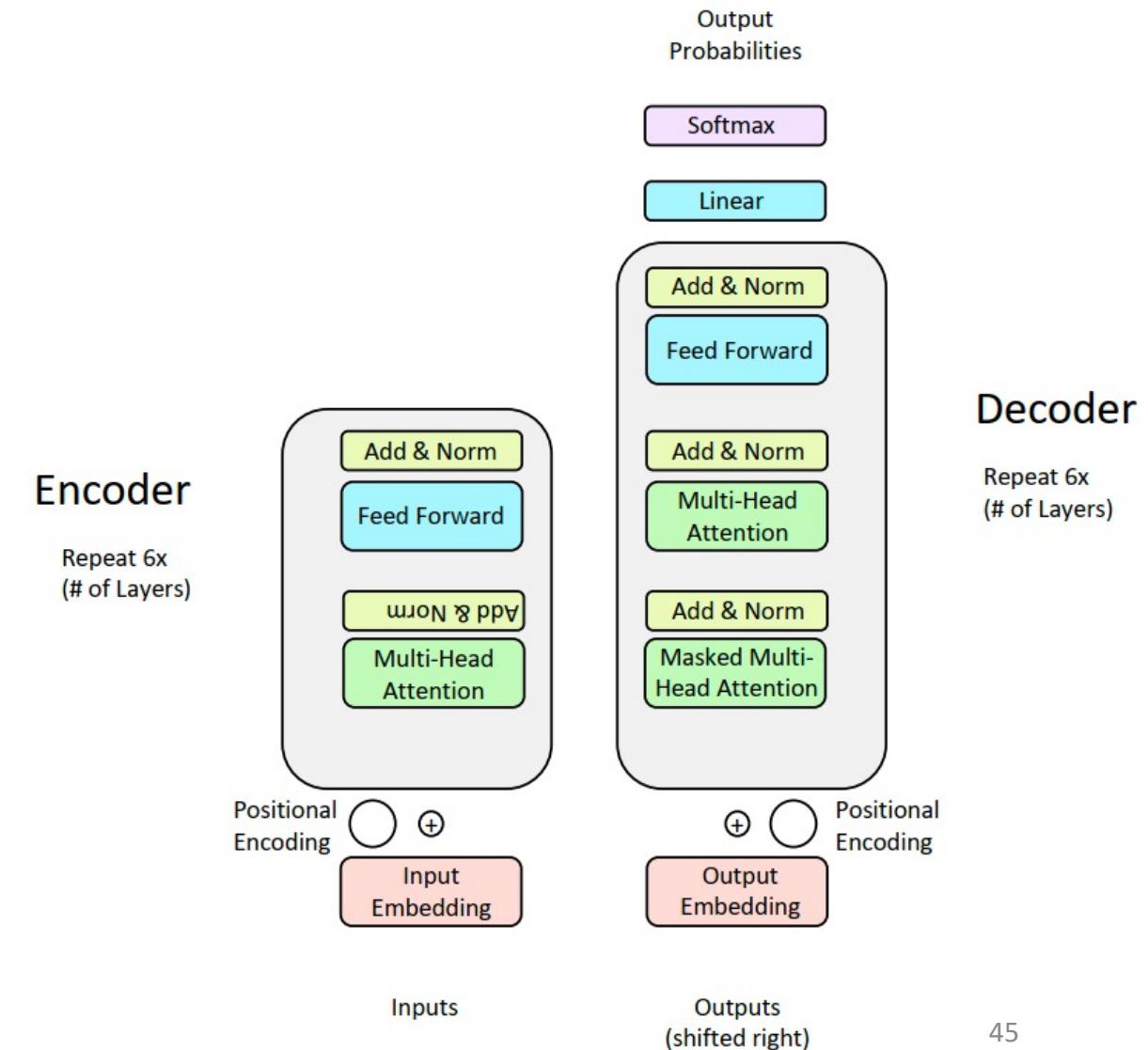
# Decoder: Finishing touches!

- 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)

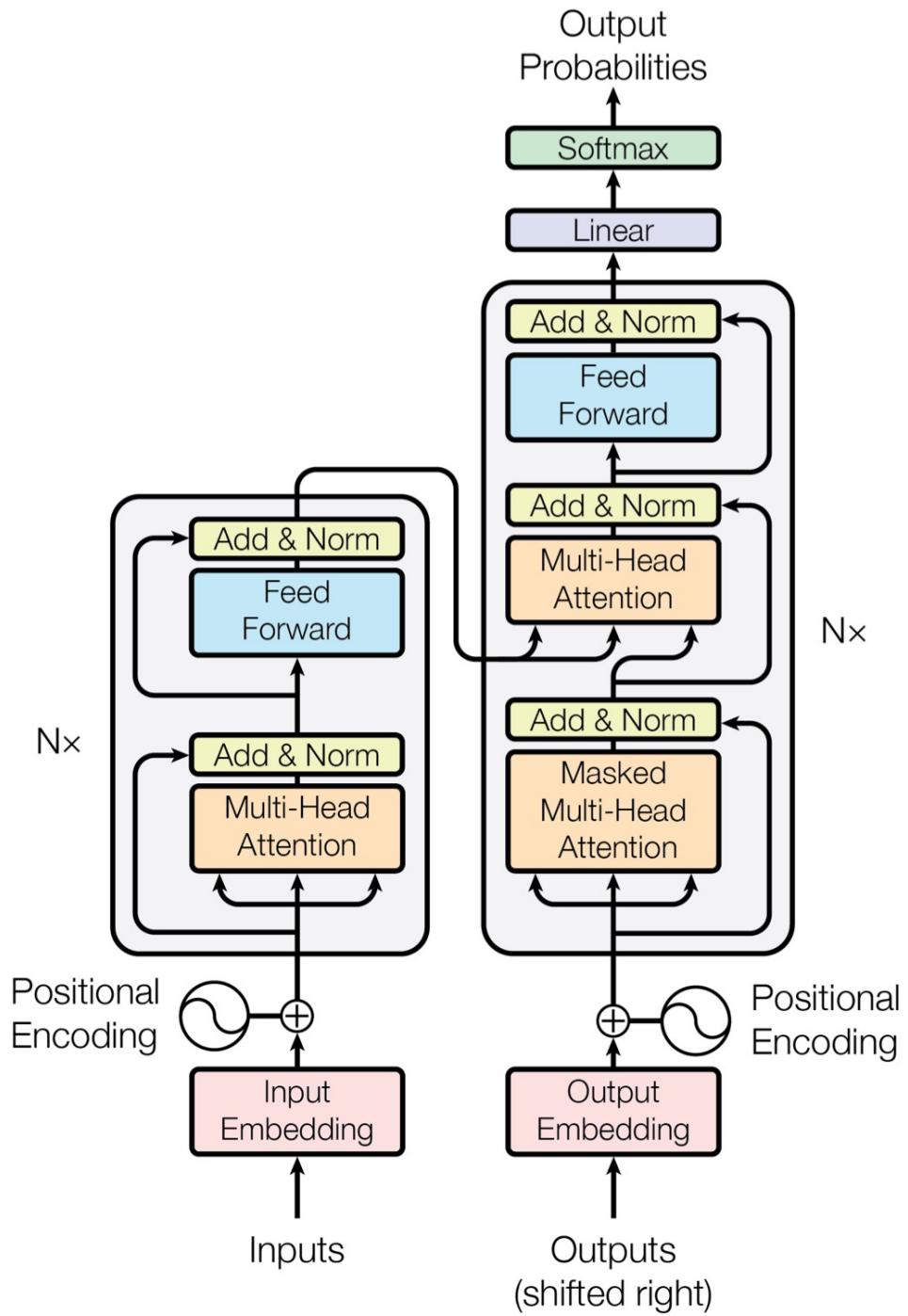


# Decoder: Finishing touches!

- 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

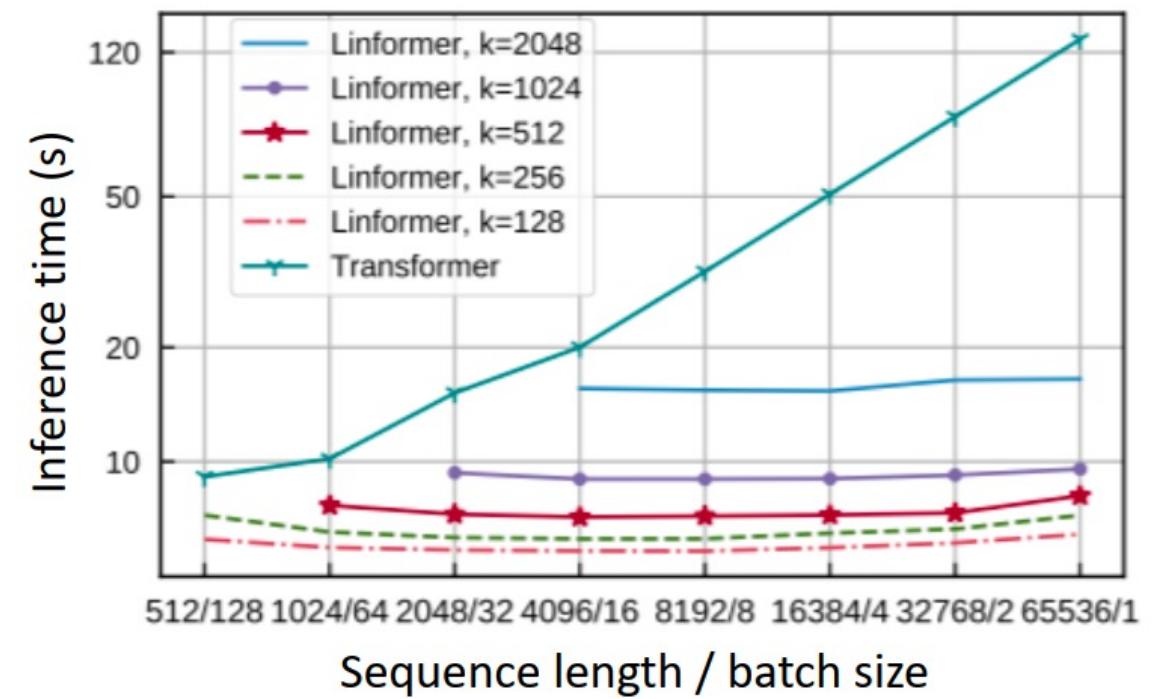
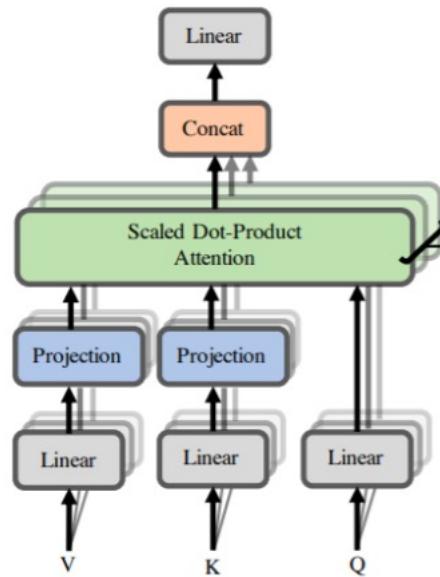
# What would we like to fix about the Transformer?

- 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 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]

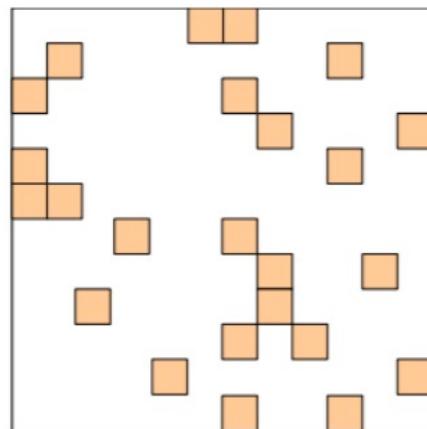
Key idea: map the sequence length dimension to a lower-dimensional space for values, keys



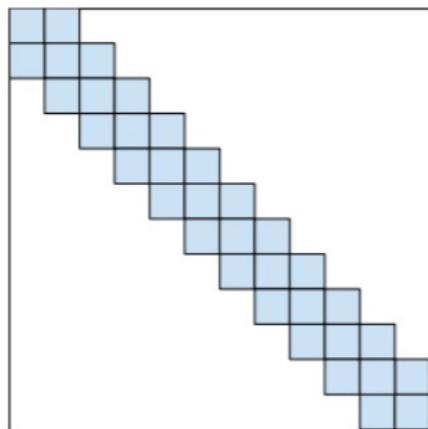
# Recent work on improving on quadratic self-attention cost

- 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]

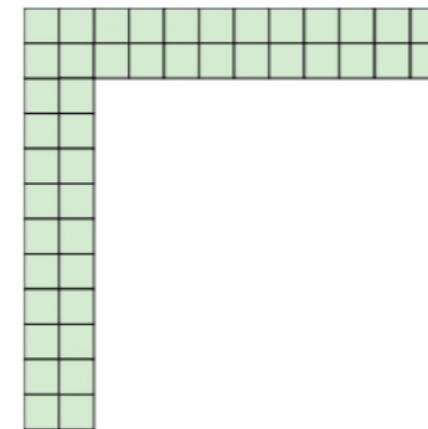
Key idea: replace all-pairs interactions with a family of other interactions, **like local windows, looking at everything, and random interactions.**



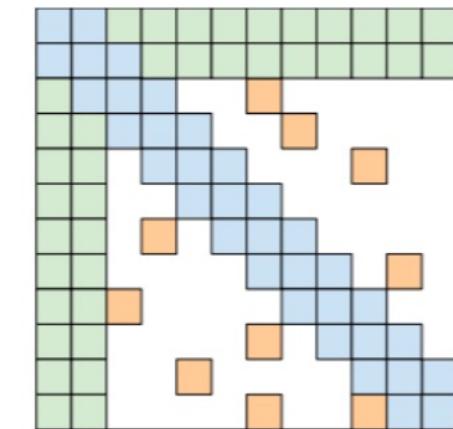
(a) Random attention



(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	225M	11.17	3.50	2.182 ± 0.005	1.838	71.66	17.78	25.02	26.62
GeLU	225M	11.17	3.58	2.179 ± 0.003	1.838	<b>75.79</b>	<b>17.86</b>	<b>25.13</b>	26.47
Swish	225M	11.17	3.62	2.186 ± 0.003	1.847	<b>73.77</b>	17.74	<b>24.34</b>	<b>26.75</b>
ELU	225M	11.17	3.56	2.270 ± 0.007	1.832	67.83	16.73	23.02	26.08
ReLU	225M	11.17	3.51	2.187 ± 0.003	1.834	<b>74.14</b>	17.74	<b>24.34</b>	26.12
GeGLU	225M	11.17	3.55	2.130 ± 0.006	<b>1.792</b>	<b>75.96</b>	18.27	<b>24.87</b>	26.87
ReLU	225M	11.17	3.57	2.145 ± 0.004	<b>1.803</b>	<b>76.17</b>	18.32	<b>24.87</b>	27.02
SeLU	225M	11.17	3.55	2.315 ± 0.004	1.948	68.76	16.73	22.75	25.99
SwiGLU	225M	11.17	3.55	2.127 ± 0.006	<b>1.789</b>	<b>76.00</b>	18.26	<b>24.34</b>	<b>27.02</b>
LiGLU	225M	11.17	3.59	2.149 ± 0.005	<b>1.798</b>	<b>75.34</b>	17.97	<b>24.34</b>	26.53
Signmoid	225M	11.17	3.61	2.187 ± 0.003	1.837	70.87	17.74	23.02	26.38
Sofplus	225M	11.17	3.67	2.207 ± 0.011	1.850	72.45	17.65	<b>24.34</b>	<b>26.89</b>
RMS Norm	225M	11.17	3.68	2.167 ± 0.008	<b>1.821</b>	<b>75.45</b>	17.94	<b>24.07</b>	27.14
Rossen	225M	11.17	3.51	2.262 ± 0.003	1.839	61.69	15.64	20.99	26.37
Rossen + LayerNorm	225M	11.17	3.26	2.223 ± 0.006	1.838	70.42	17.58	23.02	26.29
Rossen + RMS Norm	225M	11.17	3.34	2.221 ± 0.009	1.875	70.33	17.32	23.02	26.19
Ftnp	225M	11.17	2.95	2.382 ± 0.009	2.067	58.56	14.42	23.02	26.31
24 layers, $d_g = 1536$ , $H = 6$	224M	11.17	3.68	2.200 ± 0.007	1.843	<b>74.89</b>	17.75	<b>25.13</b>	<b>26.89</b>
18 layers, $d_g = 2048$ , $H = 8$	225M	11.17	3.58	2.185 ± 0.007	<b>1.831</b>	<b>76.45</b>	16.88	<b>24.34</b>	<b>27.10</b>
8 layers, $d_g = 4096$ , $H = 18$	225M	11.17	3.67	2.190 ± 0.005	1.847	<b>74.58</b>	17.96	<b>23.28</b>	<b>26.85</b>
6 layers, $d_g = 6144$ , $H = 24$	225M	11.17	3.70	2.191 ± 0.010	1.857	<b>73.55</b>	17.59	<b>24.60</b>	<b>26.66</b>
Block sharding	65M	11.17	3.91	2.497 ± 0.037	2.164	64.50	14.53	21.96	25.48
+ Factorized embeddings	45M	9.47	4.21	2.631 ± 0.005	2.183	60.84	14.05	19.84	25.27
+ Factorized & shared embeddings	20M	9.17	4.37	2.907 ± 0.313	2.385	53.95	11.37	19.84	25.19
Encoder only block sharding	170M	11.17	3.68	2.298 ± 0.023	1.929	69.60	16.23	23.02	26.23
Decoder only block sharding	144M	11.17	3.70	2.352 ± 0.029	2.082	67.93	16.13	<b>23.81</b>	26.08
Factorized Embedding	227M	9.47	3.80	2.208 ± 0.006	1.855	70.41	15.92	22.75	26.50
Factorized & shared embeddings	202M	9.17	3.90	2.320 ± 0.010	1.952	68.69	16.33	22.22	26.44
Tied encoder/decoder input embeddings	248M	11.17	3.55	2.192 ± 0.002	1.840	<b>71.70</b>	17.72	<b>24.34</b>	26.49
Tied decoder input and output embeddings	248M	11.17	3.57	2.187 ± 0.007	<b>1.827</b>	<b>74.86</b>	17.74	<b>24.87</b>	<b>26.67</b>
Unified embeddings	273M	11.17	3.55	2.195 ± 0.006	<b>1.834</b>	<b>72.99</b>	17.58	<b>23.28</b>	26.48
Adaptive input embeddings	204M	9.27	3.55	2.250 ± 0.002	1.809	60.37	16.21	<b>24.07</b>	<b>26.66</b>
Adaptive softmax	204M	9.27	3.60	2.364 ± 0.008	1.982	<b>72.91</b>	16.67	21.16	25.56
Adaptive softmax without projection	223M	10.87	3.48	2.229 ± 0.009	1.914	71.82	17.10	23.02	25.72
Mixture of softmaxes	232M	16.37	2.94	2.227 ± 0.017	<b>1.821</b>	<b>76.77</b>	17.62	22.75	<b>26.82</b>
Transformer attention	225M	11.17	3.03	2.187 ± 0.014	1.974	64.81	10.49	21.16	<b>26.89</b>
Dynamic convolution	257M	11.87	2.65	2.403 ± 0.009	2.047	58.30	12.67	21.16	17.03
Lightweight convolution	224M	10.47	4.07	2.370 ± 0.010	1.989	63.07	14.86	23.02	24.73
Evolved Transformer	217M	9.97	3.09	2.220 ± 0.006	1.863	<b>73.67</b>	10.76	<b>24.07</b>	26.58
Synthesizer (dense)	224M	11.47	3.47	2.334 ± 0.021	1.962	61.09	14.27	16.14	<b>26.63</b>
Synthesizer (dense plus)	243M	12.67	3.22	2.191 ± 0.010	1.840	<b>73.98</b>	16.76	<b>23.81</b>	<b>26.71</b>
Synthesizer (dense plus alpha)	243M	12.67	3.01	2.180 ± 0.007	<b>1.828</b>	<b>74.25</b>	17.02	<b>23.28</b>	26.61
Synthesizer (factorized)	207M	10.17	3.94	2.341 ± 0.017	1.968	62.78	15.39	<b>23.55</b>	26.42
Synthesizer (random)	254M	10.17	4.05	2.326 ± 0.012	2.009	54.27	10.35	19.56	26.44
Synthesizer (random plus)	292M	12.07	3.65	2.180 ± 0.004	1.842	<b>73.32</b>	17.04	<b>24.87</b>	26.43
Synthesizer (random plus alpha)	292M	12.07	3.42	2.186 ± 0.007	<b>1.828</b>	<b>75.24</b>	17.04	<b>24.08</b>	26.39
Universal Transformer	84M	40.07	0.88	2.406 ± 0.036	2.053	70.13	14.09	19.05	23.91
Mixture of experts	648M	11.77	3.20	2.148 ± 0.006	<b>1.785</b>	<b>74.55</b>	18.13	<b>24.08</b>	<b>26.94</b>
Switch Transformer	1108M	11.77	3.18	2.135 ± 0.007	<b>1.758</b>	<b>75.38</b>	18.02	<b>26.19</b>	<b>26.81</b>
Funnel Transformer	223M	1.97	4.30	2.288 ± 0.008	1.918	67.34	16.26	22.75	23.20
Weighted Transformer	280M	71.07	0.59	2.378 ± 0.021	1.989	69.04	16.76	23.02	26.30
Product key memory	425M	386.87	0.25	2.155 ± 0.003	<b>1.798</b>	<b>75.16</b>	17.04	<b>23.55</b>	<b>26.73</b>

**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†



# 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

# Breaking (Transformer) News!

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

### AlphaCode Attention Visualization

Hover over tokens in the solution to see which tokens the model attended to when generating the solution. Click a token to select it; clicking in empty space will deselect.

1548\_A. Web of Lies      C++      incorrect Layer 18

Head 1    Head 2    Head 3    Head 4    Head 5    Head 6    Head 7    Head 8    Head 9    Head 10    Head 11    all    none

Problem Description

```
// RATING: 1200
// TAGS: dp,implementation
// LANGUAGE IS cpp
// CORRECT SOLUTION
// When you play the game of thrones, you win, or you die. There is no middle ground.
// Cersei Lannister, A Game of Thrones by George R. R. Martin
// There are n nobles, numbered from 1 to n. Noble i has a power of i. There are also m "friendships". A friendship between nobles a and b is always mutual.
// A noble is defined to be vulnerable if both of the following conditions are satisfied:
//   * the noble has at least one friend, and
//   * all of that noble's friends have a higher power.
// You will have to process the following three types of queries.
//   1. Add a friendship between nobles u and v.
//   2. Remove a friendship between nobles u and v.
//   3. Calculate the answer to the following process.
```

Solution

```
#include <bits/stdc++.h>
using namespace std;
int n, m, q, u, v, t;
vector<int> p[200001];
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 0; i < m; i++) {
        scanf("%d%d", &u, &v);
        p[u].push_back(v);
        p[v].push_back(u);
    }
    vector<int> a(n + 1, 0);
    for (int i = 1; i <= n; i++) {
        a[i] = i;
        for (int j = 0; j < p[i].size(); j++) {
            a[p[i][j]] = i;
        }
    }
    scanf("%d", &q);
    for (int i = 0; i < q; i++) {
        scanf("%d", &t);
        if (t == 1) {
            scanf("%d%d", &u, &v);
            while (a[u] != u) {
                u = a[u];
            }
            u = a[v];
            a[v] = u;
        }
        while (a[u] != u) {
            u = a[u];
        }
    }
}
```

# 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]

## PROBLEM 1

Adapted from AMC12 2000 Problem 5

Prove that if  $|x - 2| = p$ , where  $x < 2$ , then  $x - p = 2 - 2p$ .

◊ FORMAL

INFORMAL

```
theorem amc12_2000_p5      -- ← theorem name
  (x p : ℝ)                  -- ← the statement we want
  (h₀ : x < 2)                --   to prove
  (h₁ : abs (x - 2) = p) :
  x - p = 2 - 2 * p :=  

begin                         -- ← formal proof starts here
  -- This first tactic requires that the prover invent
  -- the term: `abs (x - 2) = -(x - 2)`.  

  have h₂ : abs (x - 2) = -(x - 2), {
    apply abs_of_neg,
    linarith,
  },
  rw h₁ at h₂,
  -- At this stage the remaining goal to prove is:
  -- `x - p = 2 - 2 * p` knowing that `p = -(x - 2)`.  

  linarith,
end
```

# 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	vocab mapping	embedding
Common words	hat	→ hat	
	learn	→ learn	
Variations	taaaaaasty	→ UNK	
	laern	→ UNK	
novel items	Transformerify	→ 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 ▲]										
Polarity	Form Infinitive				Non-finite forms							Negative kutoambia										
	Positive form				Simple finite forms				Complex finite forms													
	Imperative Habitual				Singular ambia				huambia													
Polarity	Persons				Persons / Classes				Classes				Mu 18									
	1st Sg.	Pl.	2nd Sg.	Pl.	3rd / M-wa	Pl. / 2	3	M-mi	4	5	Ma	7	Ki-vi	8	9	N	10	11 / 14	15 / 17	Pa 16	Mu 18	[less ▲]
Positive	niiambia naliambia	tuliambia tvaliambia	uliambia waliambia	mliambia mwaliambia	aliambia	wallambia	ullambia	illambia	illambia	yallambia	kiliambia	viliambia	illambia	zillambia	ullambia	kuliambia	paliambia	mulambia	muambia	[less ▲]		
Negative	sikuambia	hatukuambia	hukuambia	hamkuambia	hakuambia	hawakuambi a	haikuambia	haikuambia	halikuambia	hayakuambi a	hakikuambia	havikuambia	haikuambia	hazikuambia	haukuambia	hakukuambi a	hapakuambi a	hamukambi a	hamuambia	[less ▲]		
Present												Future										[less ▲]
Positive	ninaambia naambia	tunaambia	unaambia	mnaambia	anaambia	wanaambia	unaambia	inaambia	linaambia	yanaambia	kinaambia	vinaambia	inaambia	zinaambia	unaambia	kunaambia	panaambia	muunambia	[less ▲]			
Negative	siambii	hatuambii	huambii	hamambii	haambii	hawaambii	hauambii	haiambii	halaiambii	hayaambii	hakiambii	haviambi	halambii	haziambi	hauambii	hakuambii	hapaambii	hamuambii	hamuambia	[less ▲]		
Positive	nitaambia	tutaambia	utaambia	mtaambia	ataambia	wataambia	utaambia	itaambia	itaambia	yataambia	kitaambia	vitaambia	itaambia	zitaambia	utaambia	kutaambia	pataambia	mutuambia	[less ▲]			
Negative	sitaambia	hatutaambia	hutaambia	hamtaambia	hataambia	hawataambi a	hautaambia	altaambia	altaambia	hayataambia	hakitaambia	havitaambia	altaambia	hazitaambia	hautaambia	hakutaambia	hapataambia	hamutaambi a	hamutaambia	[less ▲]		
Subjunctive												Present Conditional										[less ▲]
Positive	niambie	tuamvie	uamble	mambie	aambie	waambie	uamble	iambie	iambie	yaambie	klambie	vlambie	iambie	ziambie	uamble	kuambie	paambie	muambie	[less ▲]			
Negative	niambie	tusiambie	usiambie	msiambie	asiambie	wasiambie	usiambie	isiambie	isiambie	ysiambie	kisiambie	visiambie	isiambie	zisiambie	usiambie	kusiambie	pasiambie	musiambie	[less ▲]			
Positive	ningeambia	tungeambia	ungeambia	mngearmbia	angeambia	wangeambia	ungeambia	ingeambia	ingeambia	yangeambia	kingeambia	vingeambia	ingeambia	zingeambia	ungeambia	kungeambia	pangeambia	mungeambia	[less ▲]			
Negative	nisingeambia a singeambia	tusingeambia a hatungeambia	usingeambia a hungueambia	msingeambia a hammgeambia	asingeambia a hangeambia	wasingeambia a hawangeambia	usingeambia a haungelambia	isingeambia a halingeambia	isingeambia a hayangeambia	yasingeambia a hayingeambia	kisingeambia a hakingeambia	visingeambia a havingeambia	isingeambia a hazingeambia	zingeambia a hazingeambia	ungeambia a hakungeambia	kusingeambia a haungeambia	pasingeambia a hapangeambia	musingeambia a hamungeambia	[less ▲]			
Past Conditional												Conditional Contrary to Fact										[less ▲]
Positive	ningaliambia	tungaliambia	ungaliambia	mngaliambia	angaliambia	wangaliambia a	ungaliambia	ingaliambia	ingaliambia	yangaliambia a	kingaliambia	vngaliambia	ingaliambia	zingaliambia	ungaliambia	kungaliambia a	pangaliambia a	mungaliambia a	[less ▲]			
Negative	nisingaliambia a singaliambia	tusingaliambia a hatungaliambia	usingaliambia a hungaliambia	msingaliambia a hammgalambia	asingaliambia a hangaliambia	wasingaliambia a hawangaliambia	usingaliambia a haungaliambia	isingaliambia a halngaliambia	isingaliambia a hayangaliambia	yasingaliambia a hayingaliambia	kisingaliambia a hakingaliambia	visingaliambia a havingaliambia	isingaliambia a hazingaliambia	zingaliambia a hazingaliambia	ungaliambia a hakungaliambia	kusingaliambia a haungaliambia	pasingaliambia a hapangaliambia	musingaliambia a hamungaliambia	[less ▲]			
Gnostic Perfect												Perfect										[less ▲]
Positive	naambia	twaambia	waambia	mwaambia	aambia	waambia	yaambia	laambia	yaambia	chaambia	vyaambia	yaambia	zaambia	waambia	kwaambia	paambia	mwaambia	[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

# 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

	word	vocab mapping	embedding
Common words	hat	→ hat	
	learn	→ learn	
Variations	taaaaasty	→ taa## aaa## sty	
	laern	→ la## ern	
misspellings			
novel items	Transformerify	→ Transformer## ify	

# Contents

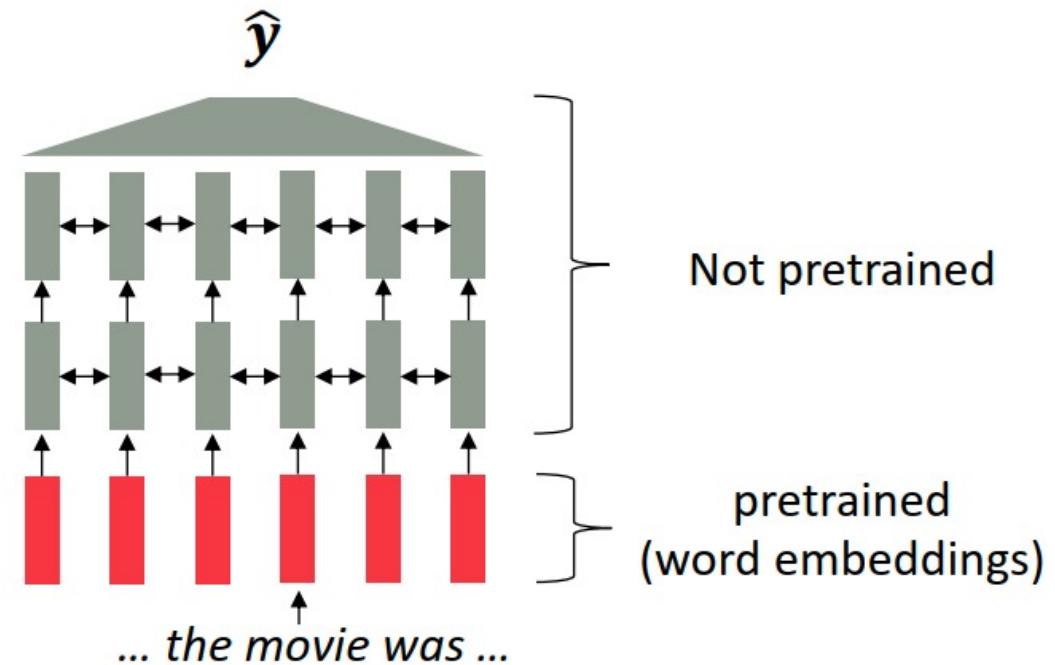
- Transformers
  - Impact of Transformers on NLP (and ML more broadly)
  - From Recurrence (RNNs) to Attention-Based NLP Models
  - Understanding the Transformer Model
  - Drawbacks and Variants of Transformers
- Pretraining Language Models(PLMs)
  - Subword modeling
  - Motivating model pretraining 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

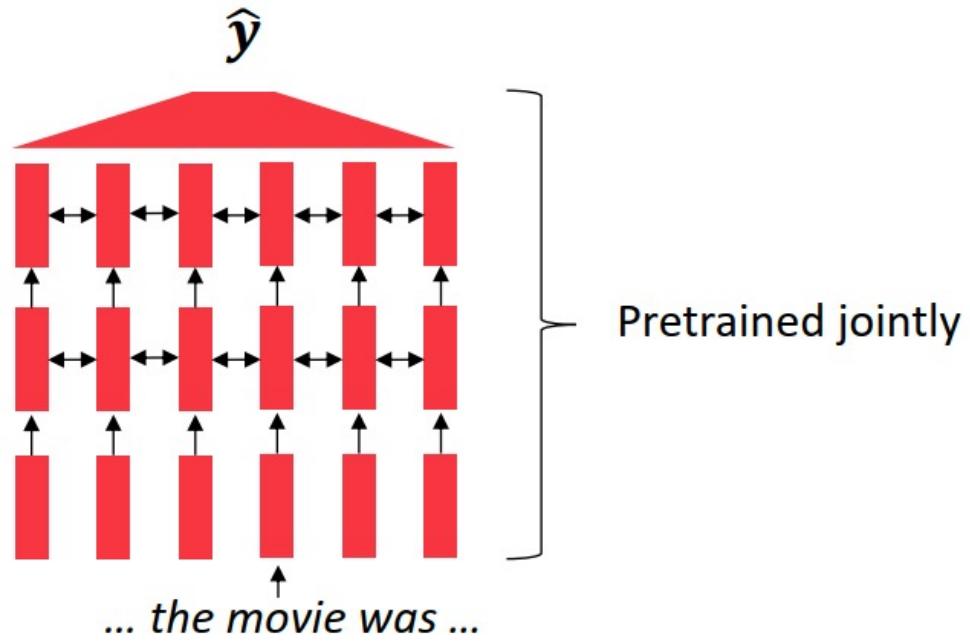
- Since 2017:
  - 仅用预训练的词向量
  - 在训练模型时考虑如何利用上下文信息
- 问题:
  - 下游任务的训练数据必须足以学习上下文信息
  - 大多数参数都是随机初始化的



[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



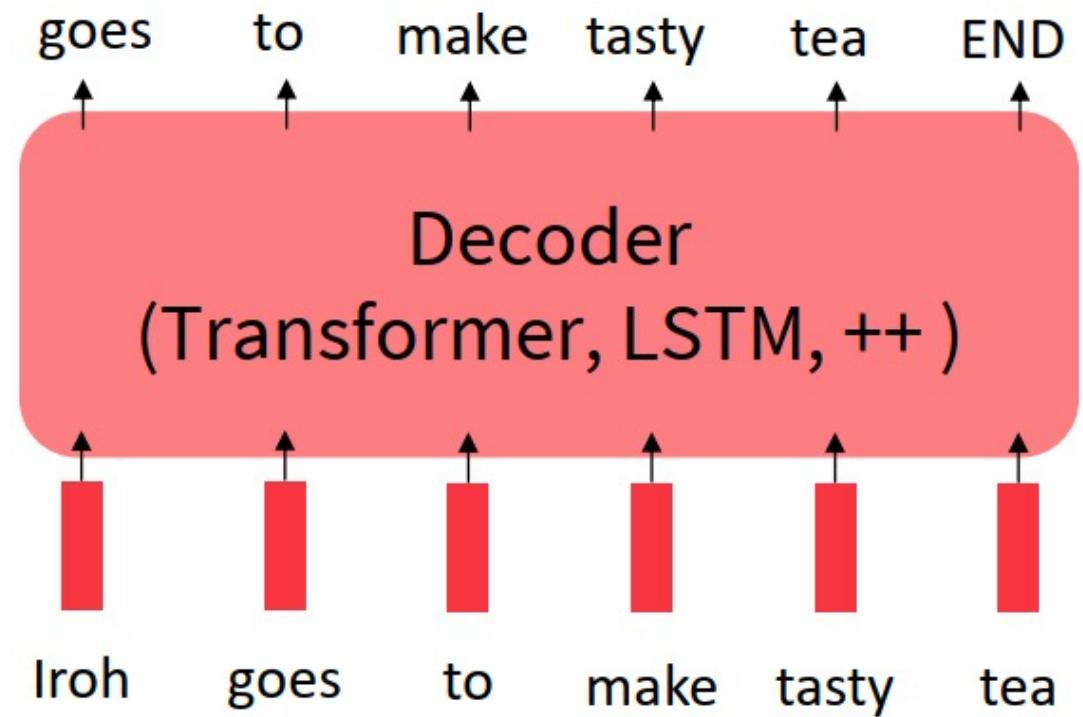
[This model has learned how to represent entire sentences through pretraining]

# 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

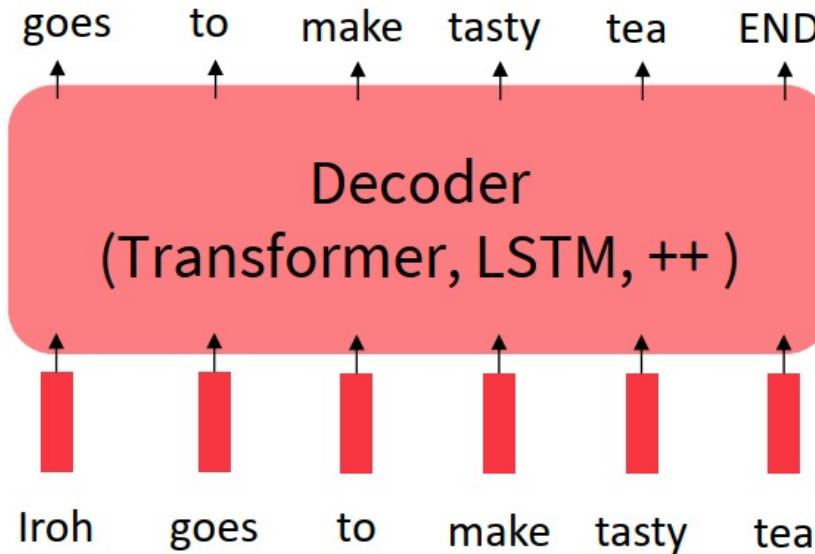


# The Pretraining / Finetuning Paradigm

- 预训练可以以参数初始化的方式来提升NLP模型的效果

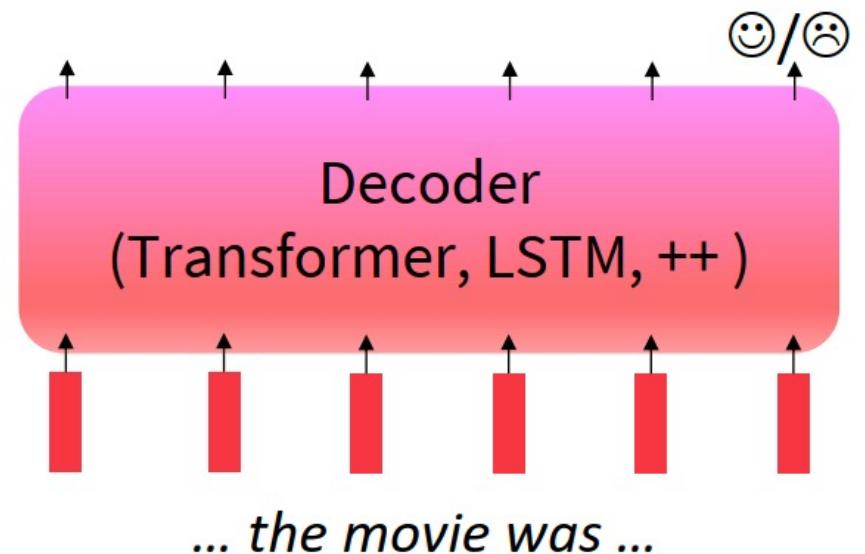
## Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



## Step 2: Finetune (on your task)

Not many labels; adapt to the task!

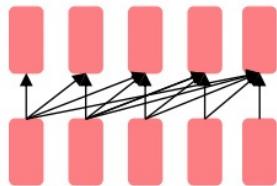


# Contents

- 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

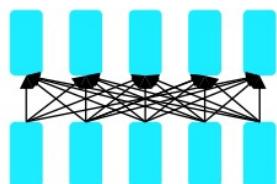
# Pretraining for three types of architectures

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



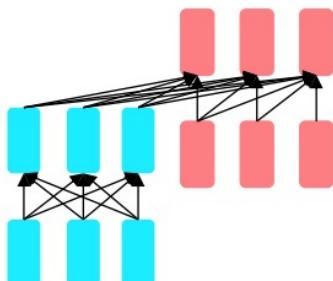
**Decoders**

- 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

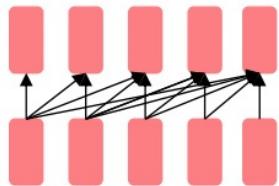


**Encoder-  
Decoders**

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

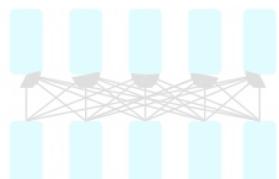
# Pretraining for three types of architectures

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



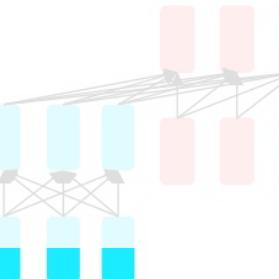
**Decoders**

- 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

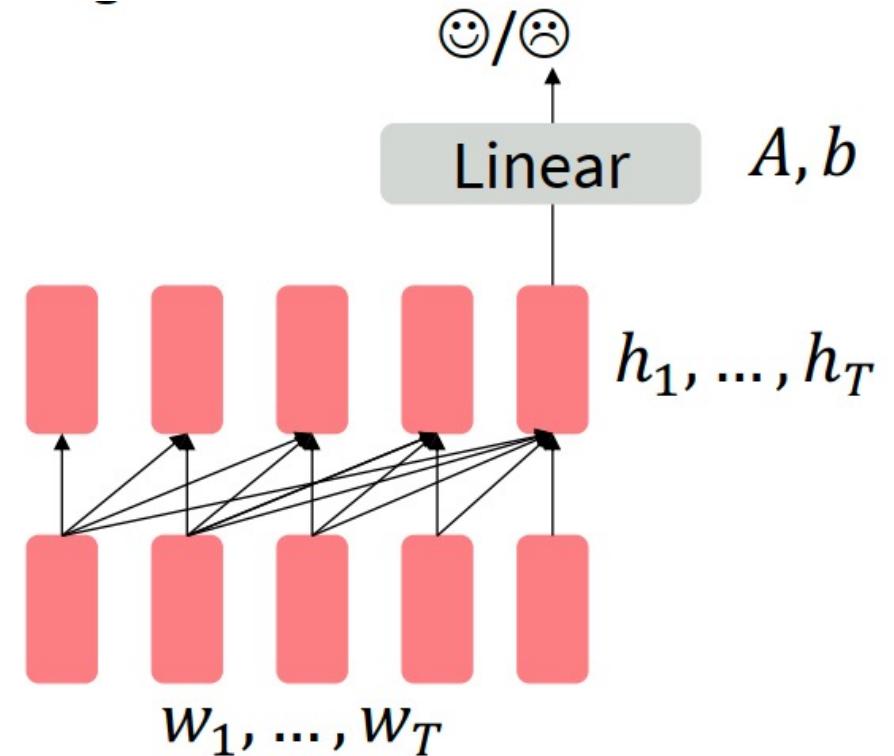


**Encoder-Decoders**

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



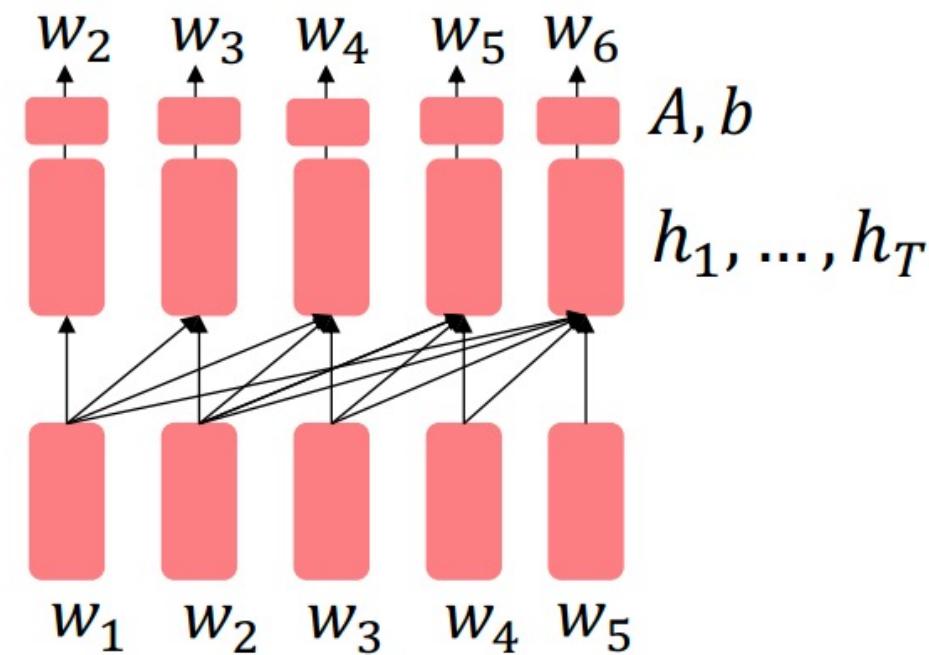
[Note how the linear layer hasn't been pretrained and must be learned from scratch.]

# 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）

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

- $A, b$ 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!

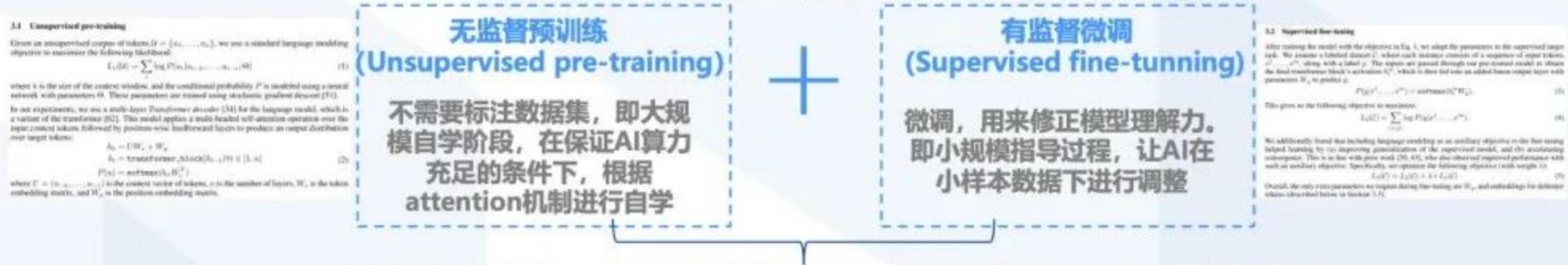
- Transformer decoder with 12 layers.
- 768-dimensional hidden states, 3072-dimensional feed-forward hidden layers.
- Byte-pair encoding with 40,000 merges
- Trained on BooksCorpus: over 7000 unique books.
  - Contains long spans of contiguous text, for learning long-distance dependencies

# 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

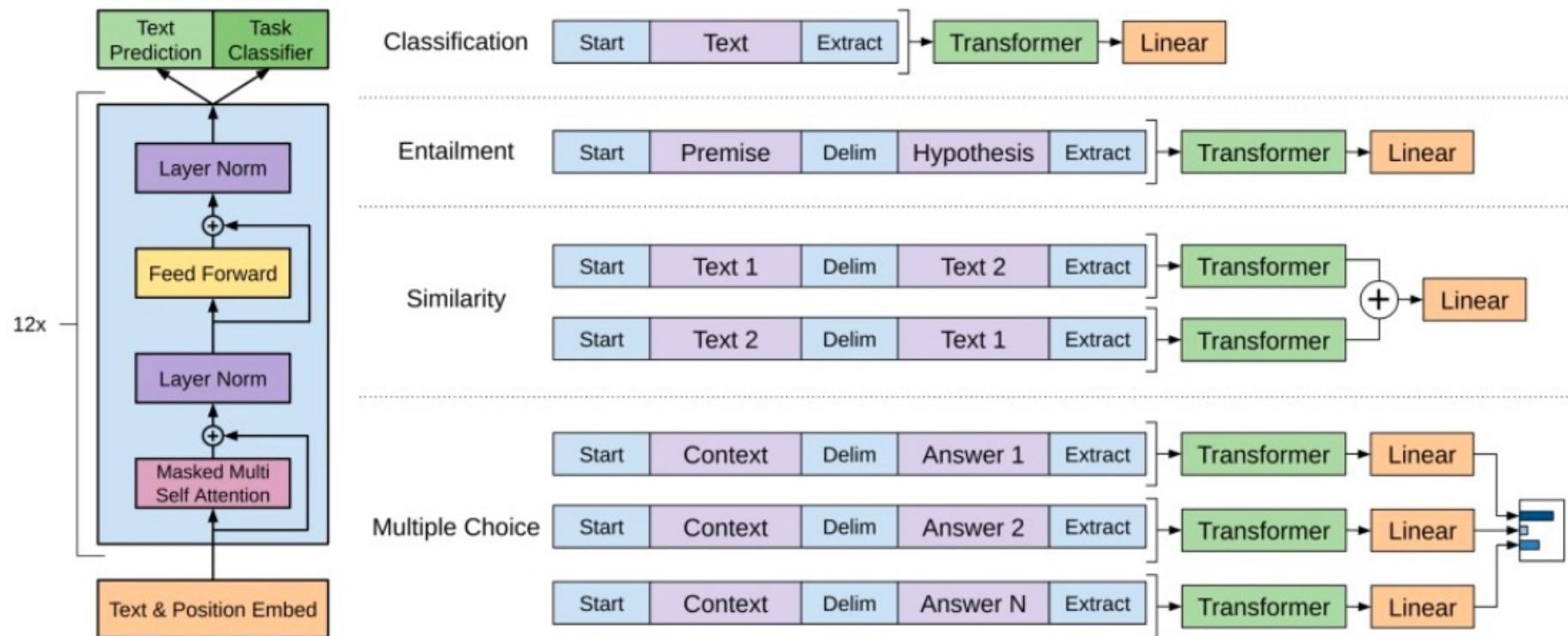
- ✓ GPT (Generative Pre-training Transformer) 于2018年6月由OpenAI首次提出。GPT模型考虑到在自然语言理解中有大量不同的任务，尽管大量的未标记文本语料库非常丰富，但用于学习这些特定任务的标记数据却很少，这使得经过区分训练的模型很难充分执行。同时，大多数深度学习方法需要大量手动标记的数据，这限制了它们在许多缺少注释资源的领域的适用性。
- ✓ 在考虑以上局限性的前提下，GPT论文中证明，通过对未标记文本的不同语料库进行语言模型的生成性预训练，然后对每个特定任务进行区分性微调，可以实现这些任务上的巨大收益。和之前方法不同，GPT在微调期间使用任务感知输入转换，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.

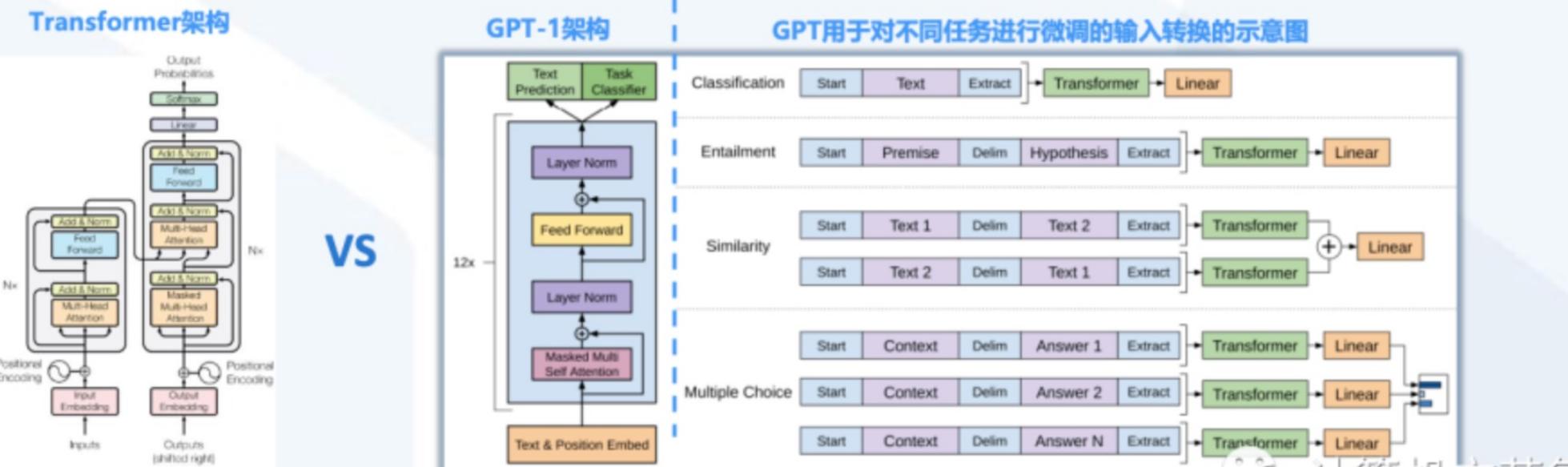
# GPT架构对于Transformer的改进

## ■ 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

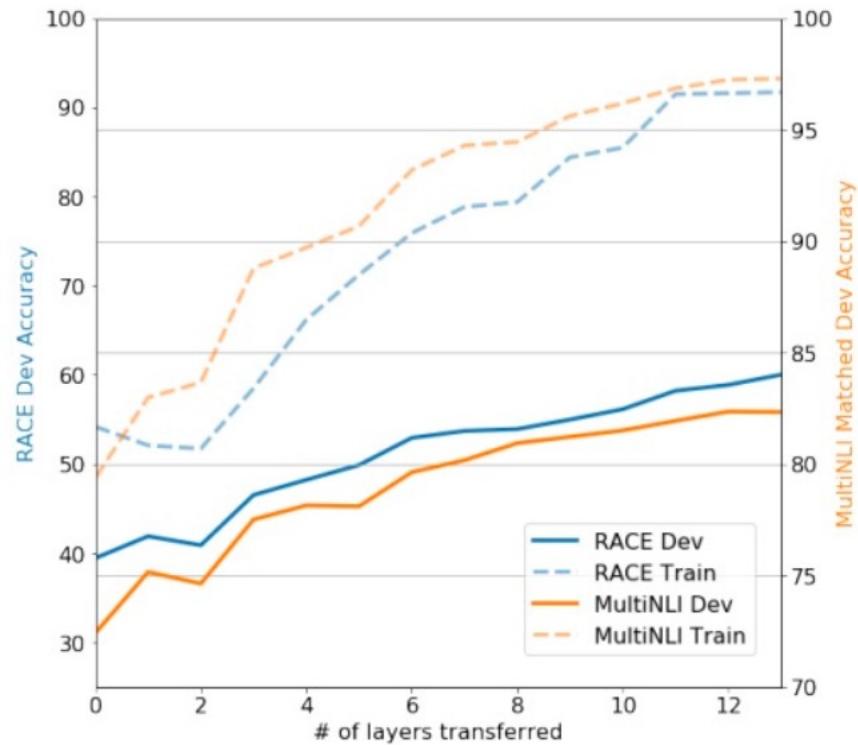


# Generative Pretrained Transformer (GPT) [Radford et al., 2018]

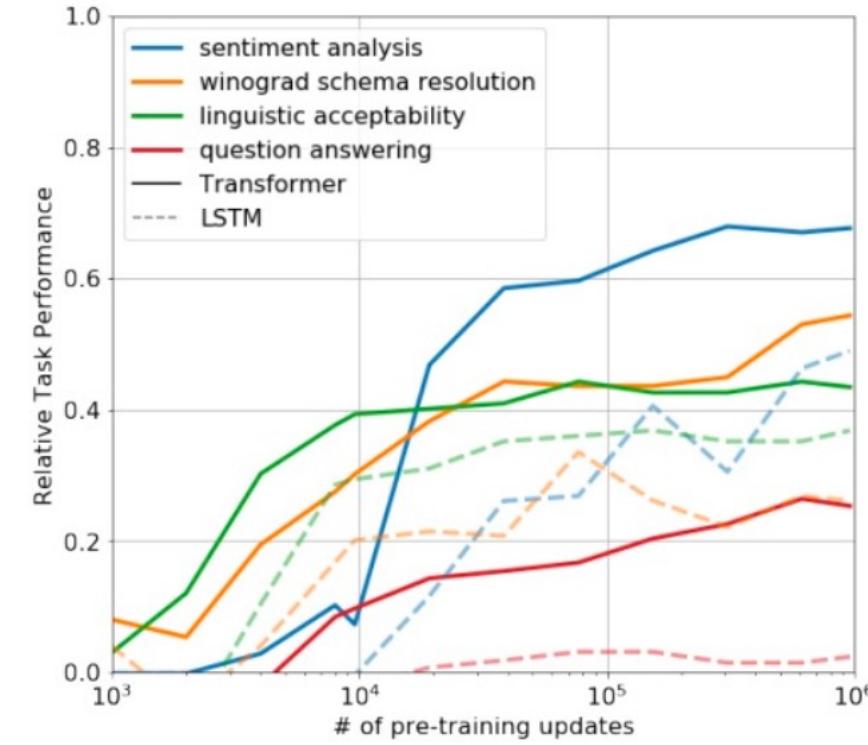
- GPT results on various natural language inference datasets

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

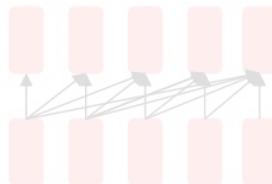
- 预训练加微调范式中，可能在这种范式下实现的泛化可能很差，因为该模型过于特定于训练分布，并且在其之外无法很好地泛化。
- 微调模型在特定基准上的性能，即使名义上是人类水平，也可能夸大基础任务的实际性能。

## 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。

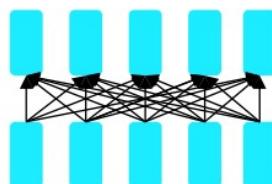
# Pretraining for three types of architectures

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



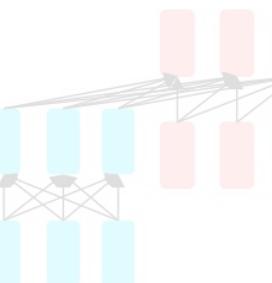
Decoders

- 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

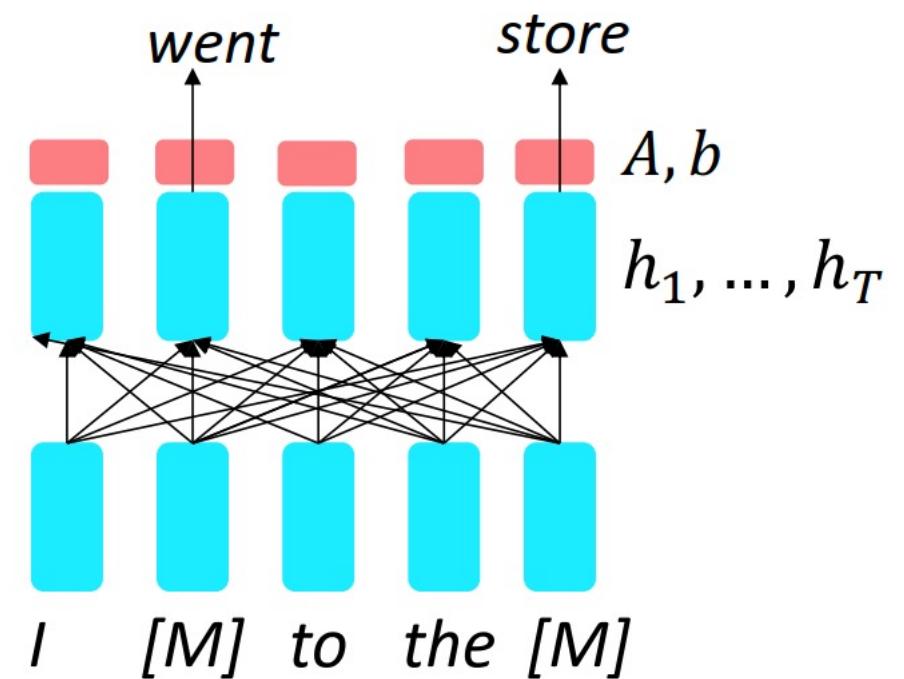


Encoder-Decoders

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

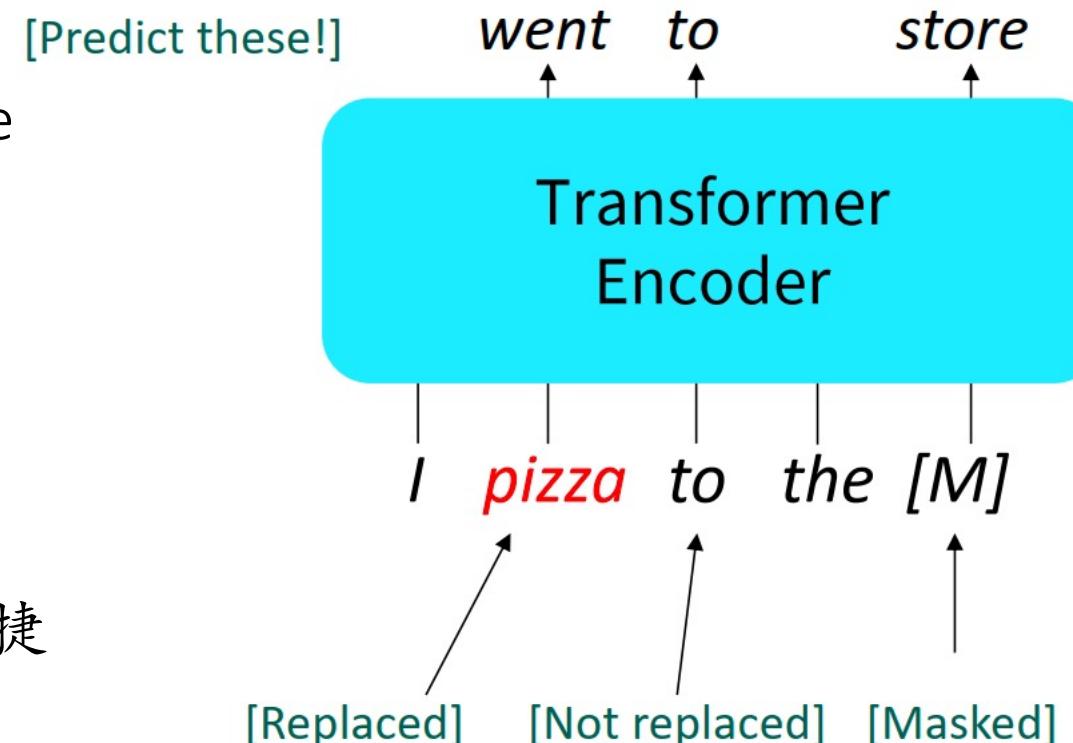
# Pretraining encoders: what pretraining objective to use?

- 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



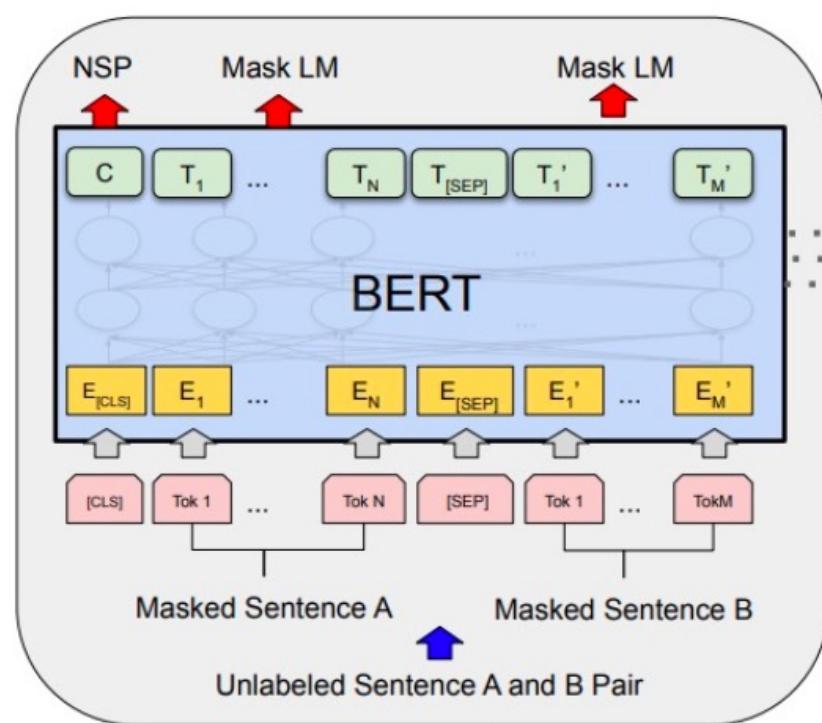
# BERT: Bidirectional Encoder Representations from Transformers

- 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

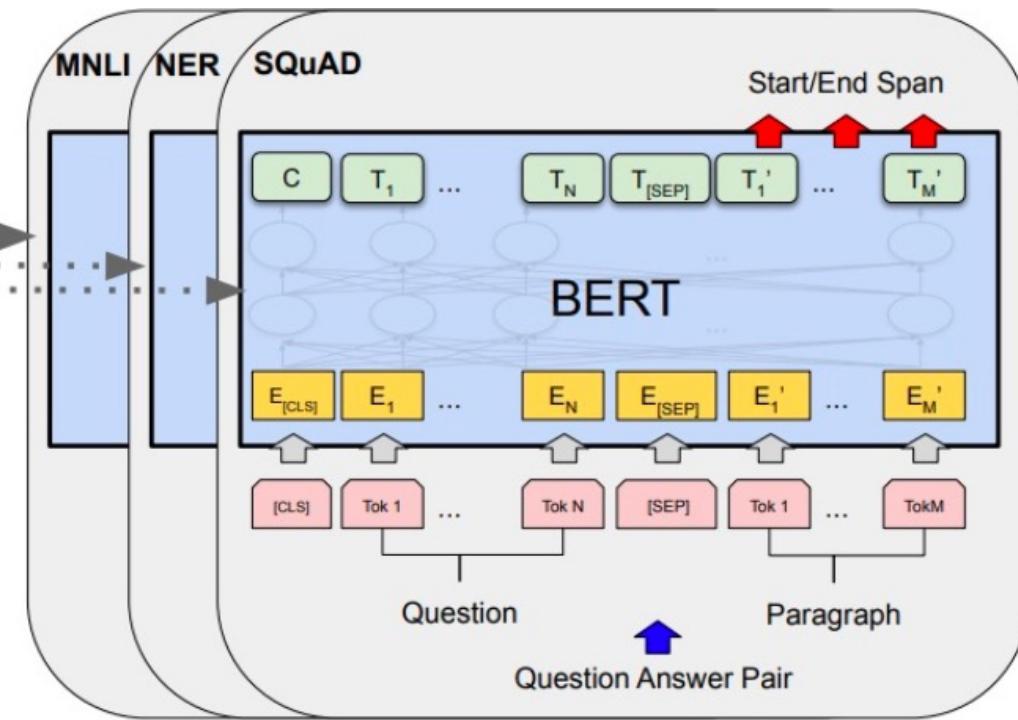


# BERT: Bidirectional Encoder Representations from Transformers

- 预训练结构与微调结构大同小异



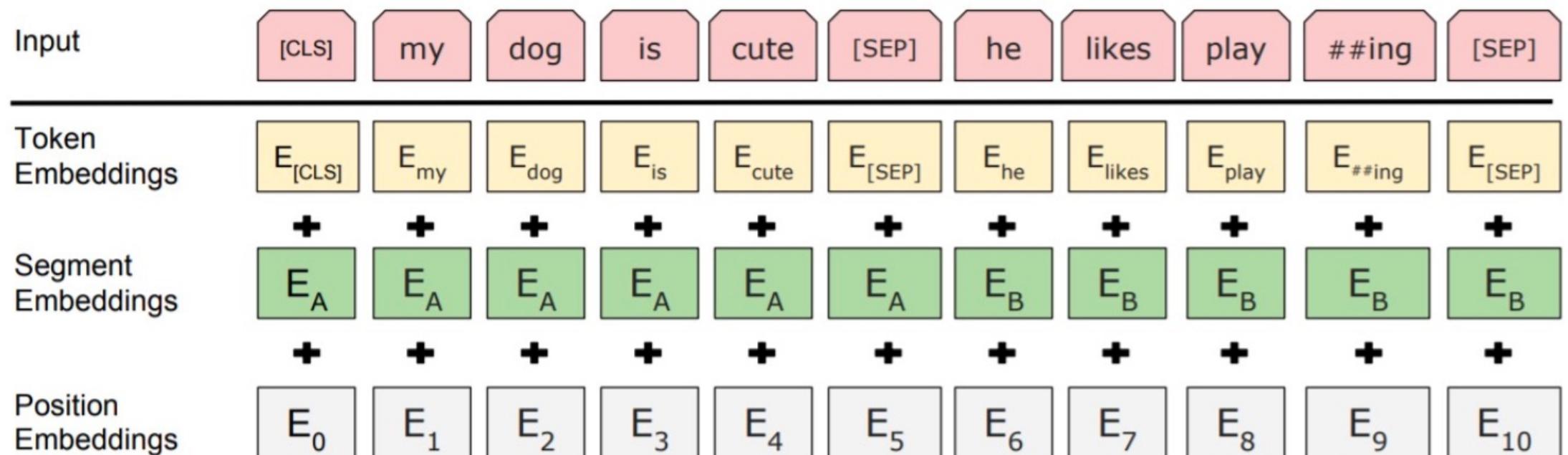
Pre-training



Fine-Tuning

# 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

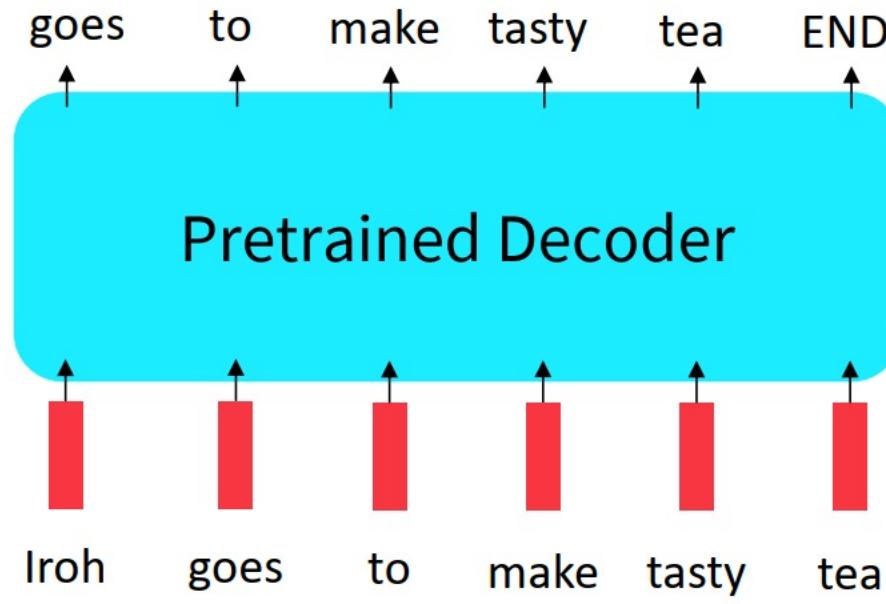
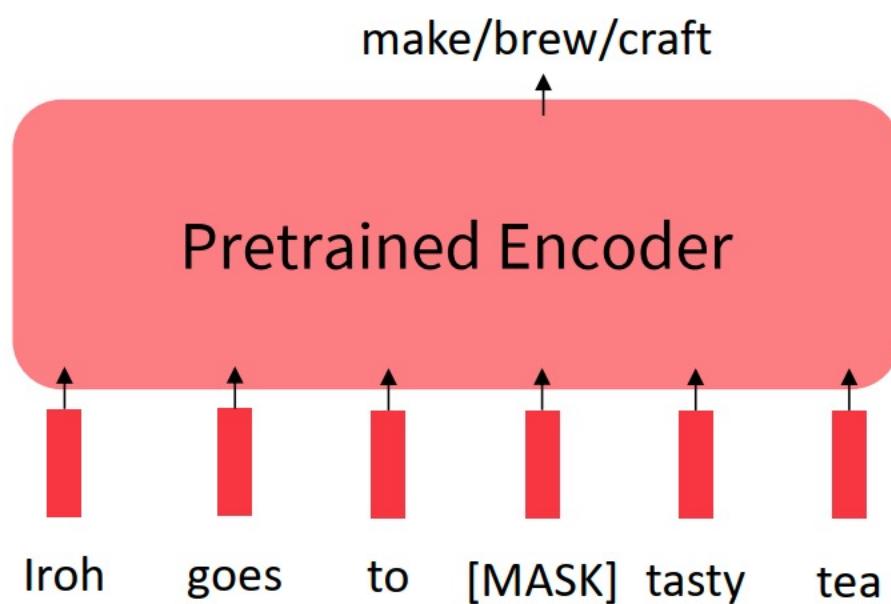
- BERT非常受欢迎，用途广泛;对BERT的微调为广泛的任务带来了新的最高结果。
- QQP: Quora Question Pairs (detect paraphrase questions)
- QNLI: natural language inference over question answering data
- SST-2: sentiment analysis
- CoLA: corpus of linguistic acceptability (detect whether sentences are grammatical.)
- STS-B: semantic textual 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
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>92.7</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>82.1</b>

Note that BERT<sub>BASE</sub> was chosen to have the same number of parameters as OpenAI GPT.

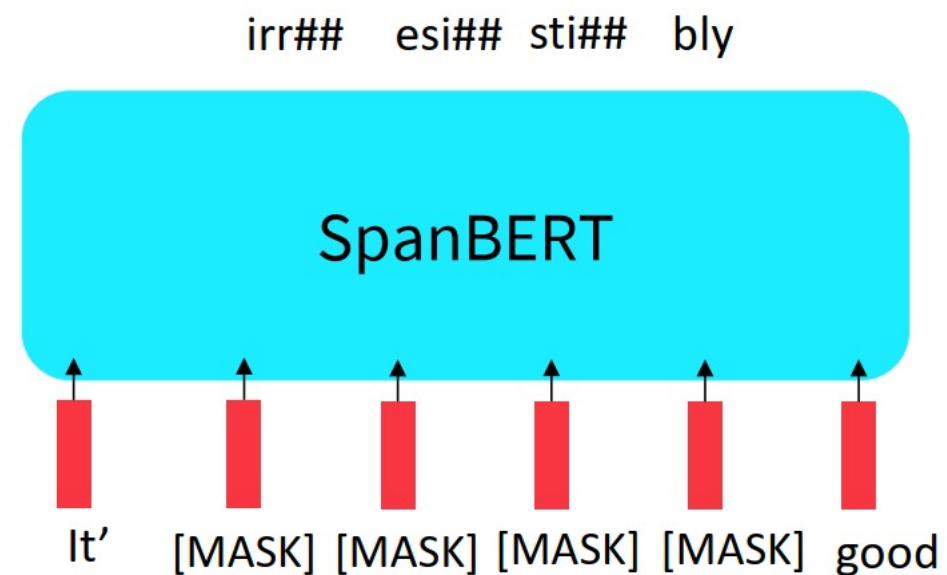
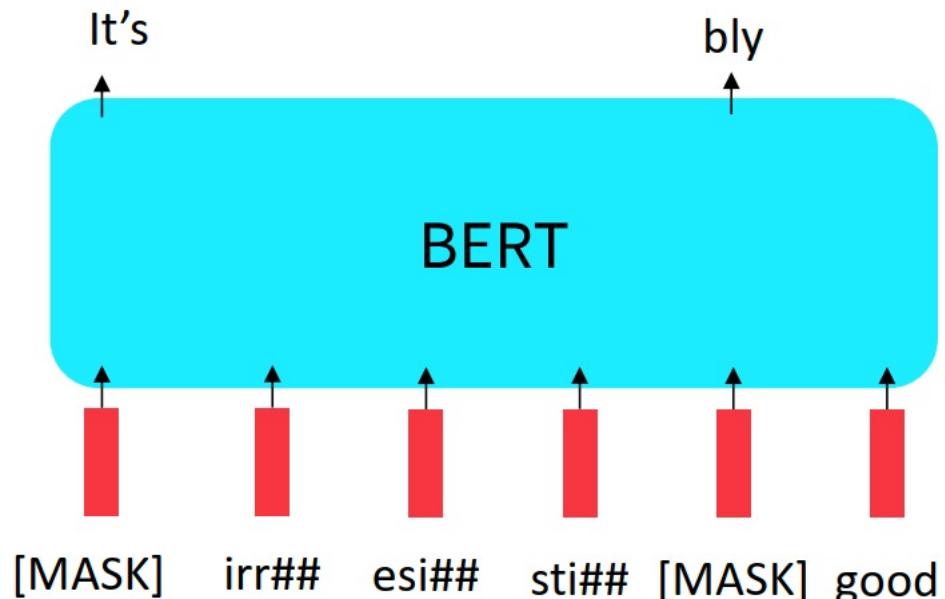
# Limitations of pretrained encoders

- 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



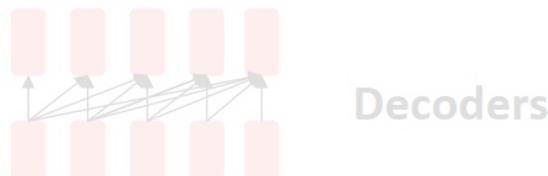
# Extensions of BERT

- 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
<b>RoBERTa</b>						
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	<b>94.6/89.4</b>	<b>90.2</b>	<b>96.4</b>
<b>BERT<sub>LARGE</sub></b>						
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.



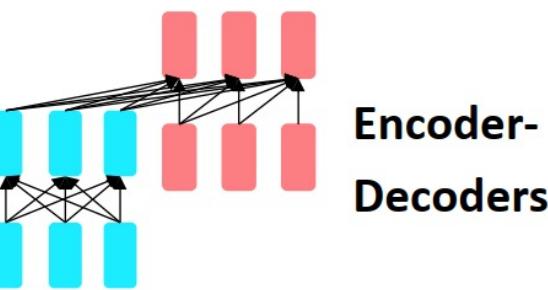
Decoders

- 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?

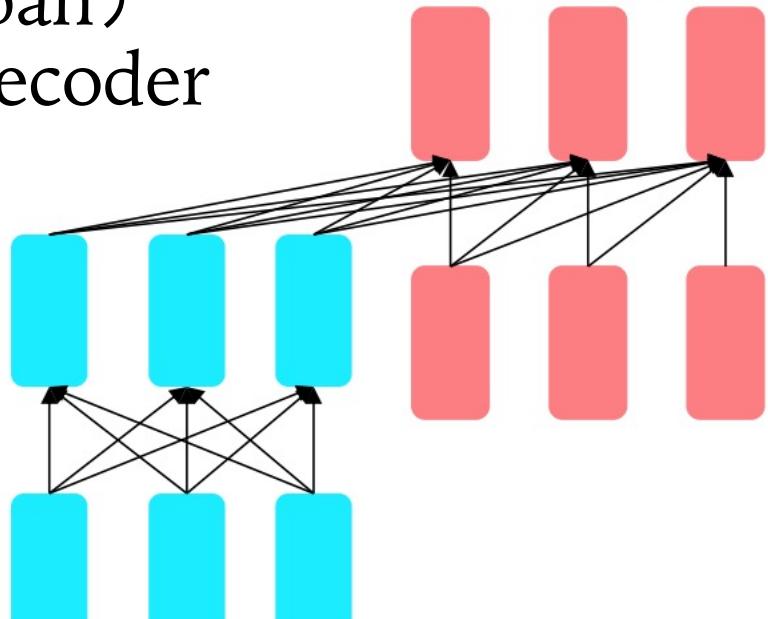
- Raffel et al., 2018发现最有效的方法是span corruption: T5
- 在输入中把不同长度的单词子序列 (span) 替换为占位符 (placeholder)，并在decoder 中把之前替换掉的span解码出来

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

Targets  
<X> for inviting <Y> last <Z>

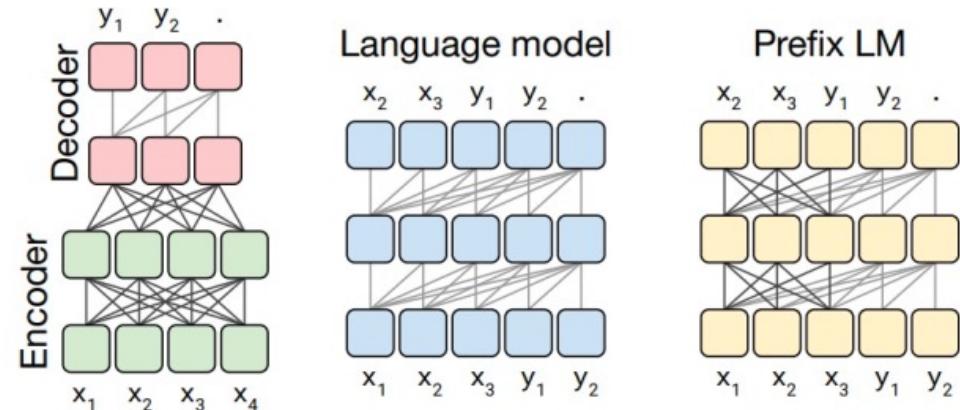


Inputs

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

# Pretraining encoder-decoders: what pretraining objective to use?

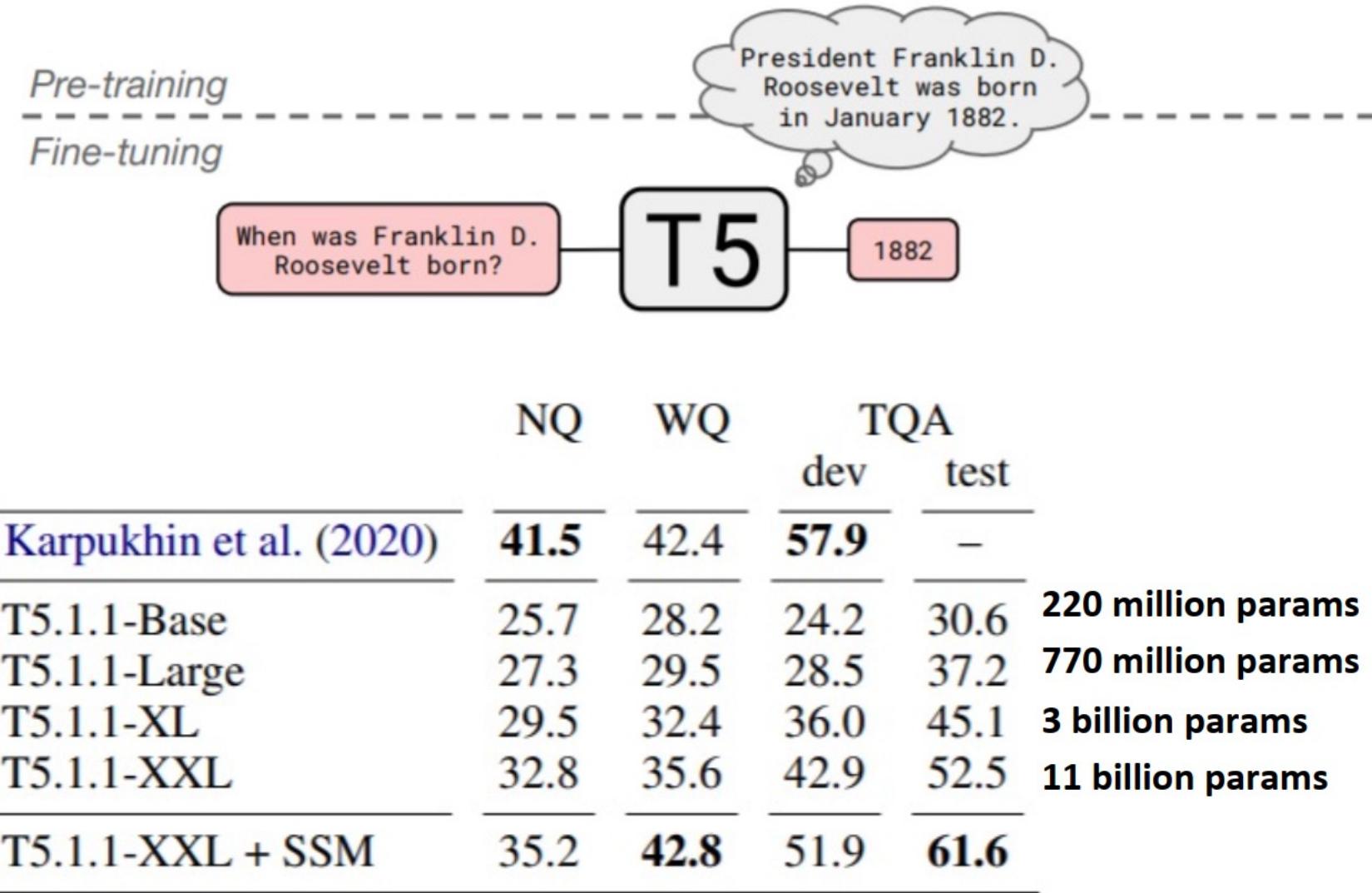
- Raffel et al., 2018发现
  - encoder-decoder比单纯的decoder效果好
  - Span corruption比单纯的LM效果好



Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	$M$	<b>83.28</b>	<b>19.24</b>	<b>80.88</b>	<b>71.36</b>	<b>26.98</b>	<b>39.82</b>	<b>27.65</b>
Enc-dec, shared	Denoising	$P$	$M$	82.81	18.78	<b>80.63</b>	<b>70.73</b>	26.72	39.03	<b>27.46</b>
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: 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	Assumptions	Tokens per byte	Ratio	Size
	(billion)		(Tokens / bytes)		(GB)
<b>Web data</b>	<b>410B</b>	–	0.71	1:1.9	<b>570</b>
<b>WebText2</b>	<b>19B</b>	<i>25% &gt; WebText</i>	0.38	1:2.6	50
<b>Books1</b>	<b>12B</b>	<i>Gutenberg</i> 	0.57	1:1.75	21
<b>Books2</b>	<b>55B</b>	<i>Bibliotik</i>	0.54	1:1.84	101
<b>Wikipedia</b>	<b>3B</b>	<i>See RoBERTa</i>	0.26	1:3.8	11.4
Total	<b>499B</b>			753.4GB	

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

# 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...”

Tradit

Fine-tu

The mo

large c



## The three settings we explore for in-context learning

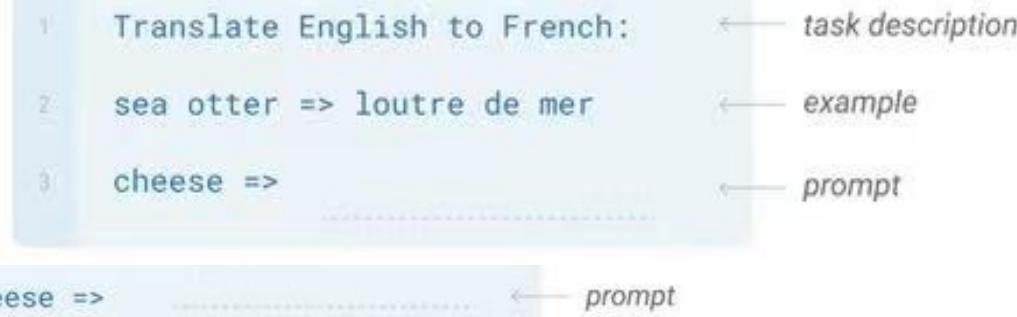
### Zero-shot

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



### One-shot

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



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

### Few-shot

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



# LLM & chatGPT

# chatGPT是声称是人工智能技术（AIGC）的一部分



# ChatGPT是在GPT基础上进一步开发的自然语言模型

- GPT模型是一种NLP模型，利用Transformer来预测下一个单词的分布。通过在大型文本语料库中学习到的语言模式来生成自然语言文本。

图3：ChatGPT由GPT-1到InstructGPT模型的不断成熟而逐步形成





# ChatGPT官方博客简介

The main features of ChatGPT highlighted in the official blog:

- ▶ answer followup questions
- ▶ admit its mistakes
- ▶ challenge incorrect premises
- ▶ reject inappropriate requests

ChatGPT Blog: <https://openai.com/blog/chatgpt/>

# ChatGPT模型大小

- GPT3中提供了不同的模型版本

Model Name	$n_{\text{params}}$	$n_{\text{layers}}$	$d_{\text{model}}$	$n_{\text{heads}}$	$d_{\text{head}}$	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	$6.0 \times 10^{-4}$
GPT-3 Medium	350M	24	1024	16	64	0.5M	$3.0 \times 10^{-4}$
GPT-3 Large	760M	24	1536	16	96	0.5M	$2.5 \times 10^{-4}$
GPT-3 XL	1.3B	24	2048	24	128	1M	$2.0 \times 10^{-4}$
GPT-3 2.7B	2.7B	32	2560	32	80	1M	$1.6 \times 10^{-4}$
GPT-3 6.7B	6.7B	32	4096	32	128	2M	$1.2 \times 10^{-4}$
GPT-3 13B	13.0B	40	5140	40	128	2M	$1.0 \times 10^{-4}$
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	$0.6 \times 10^{-4}$

- OpenAI对外提供的API提供了以下4个模型

Language models

Base models

Ada Fastest

Babbage

Curie

Davinci Most powerful

\$0.0004 /1K tokens

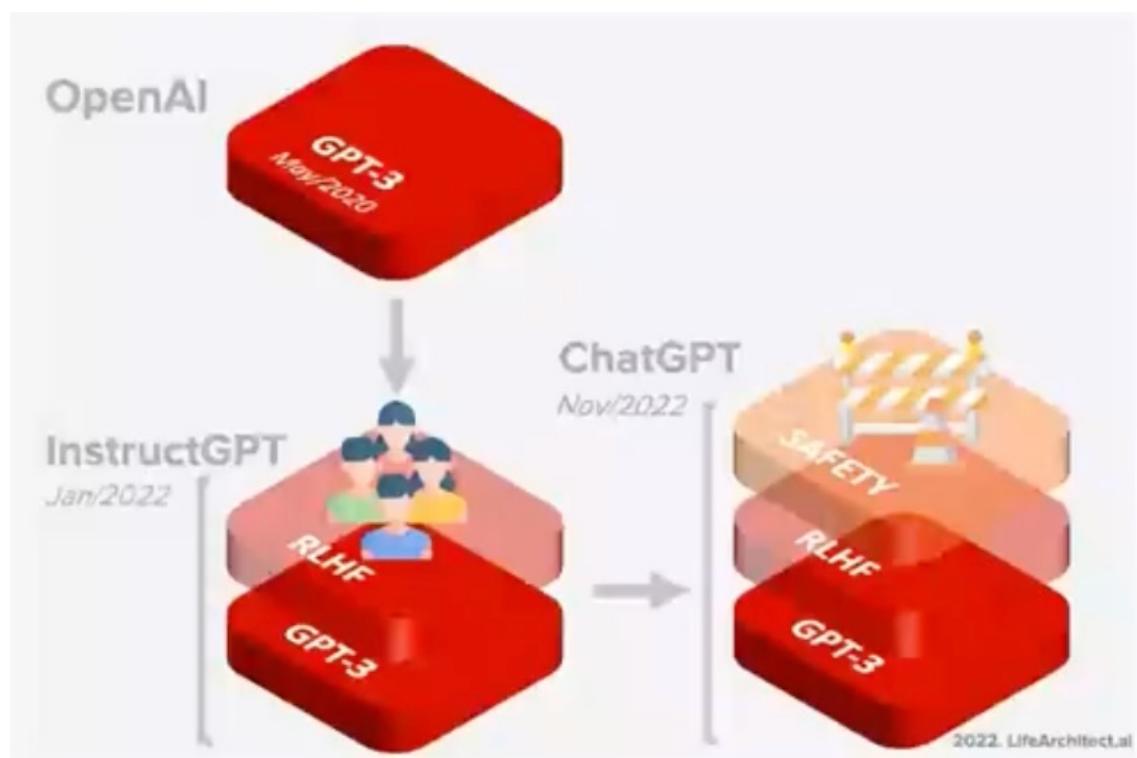
\$0.0005 /1K tokens

\$0.0020 /1K tokens

\$0.0200 /1K tokens

Multiple models, each with different capabilities and price points.  
Ada is the fastest model, while Davinci is the most powerful.

# ChatGPT时间线



## Timeline to ChatGPT

Date	Milestone
11/Jun/2018	GPT-1 announced on the OpenAI blog. →
14/Feb/2019	GPT-2 announced on the OpenAI blog.
28/May/2020	Initial GPT-3 preprint paper published to arXiv.
11/Jun/2020	GPT-3 API private beta.
22/Sep/2020	GPT-3 licensed to Microsoft.
18/Nov/2021	GPT-3 API opened to the public.
27/Jan/2022	InstructGPT released, now known as GPT-3.5. InstructGPT preprint paper Mar/2022.
28/Jul/2022	Exploring data-optimal models with FIM, paper on arXiv.
1/Sep/2022	GPT-3 model pricing cut by 66% for davinci model.
21/Sep/2022	Whisper (speech recognition) announced on the OpenAI blog.
28/Nov/2022	GPT-3.5 expanded to text-davinci-003, announced via email: <ol style="list-style-type: none"><li>1. Higher quality writing.</li><li>2. Handles more complex instructions.</li><li>3. Better at longer form content generation.</li></ol>
30/Nov/2022	ChatGPT announced on the OpenAI blog.
Next...	GPT-4...

# chatGPT迭代部署

- 从GPT3和codex中吸取很多经验，嵌入了人类反馈强化学习（RLHF）和指令微调，使其具备了理解上下文和连贯性等特征。

图7：ChatGPT提升的核心点如下所示



# 超强的理解能力

- 能够理解多样化的用户意图（几乎能借助任何话题，回复不会离题）
- 多轮对话理解能力
- 异构数据融合理解能力（多语言，代码混合理解）

# 超强的理解能力

s 我说一首词，看看你能不能理解，好吗？

当然，我会尽力理解你说的词语。请告诉我它是什么。

5 何处望东欧

第聂伯河起画楼

卅载分道多少恨

悠悠

万里黑海无尽愁

新冠不曾收

毛兄毛弟战未休

帝国荣光今何在

难求

一样悲欢付东流



这是一首诗歌，题目应该叫《赋得古原草送别》。这首诗句中描述了远离故土的人们思念故乡的心情，以及战争带来的痛苦和不幸。在最后一句中，“一样悲欢付东流”表达了人生无常和无法预料的命运。另外，“新冠”可能指代当前的新冠疫情。

这里面涉及到两个国家，你知道是哪两个吗？

根据诗句中出现的地名和历史背景，我认为这首诗涉及到的两个国家应该是中国和乌克兰（前苏联）。其中“何处望东欧，第聂伯河起画楼”是指中国北方边疆地区的人们向东眺望，望向乌克兰东部的第聂伯河（Dnieper River）流域地区；“万里黑海无尽愁”则指的是乌克兰位于黑海边上的地理位置。

乌克兰猜对了，另一个不是中国。你再想想。

好的，感谢您的纠正。根据诗句中的历史背景和地名，我再推测一下，另一个国家可能是俄罗斯。诗句中的“第聂伯河”和“黑海”都位于乌克兰和俄罗斯之间，而“卅载分道多少恨”则暗示了两国之间的历史矛盾和纷争。另外，“毛兄毛弟战未休，帝国荣光今何在”可能暗示了前苏联解体后俄罗斯的命运。

很棒！

谢谢夸奖，如果您还有其他问题或需求，可以随时提出。

# 超强的理解能力

WT

现在请你伪装成一个Linux系统终端。我输入Linux命令，你给出相应的输出

WT

好的，请问您需要执行哪些Linux命令？

WT

cd ~ & ls

WT

当您输入 "cd ~ & ls" 命令后，我将会依次输出以下内容：

首先，"cd ~" 命令会将当前目录切换到当前用户的 home 目录，也就是该用户的主目录，通常以该用户的用户名命名。因此，该命令不会有任何输出。

然后，"ls" 命令将会列出该用户的主目录中的所有文件和文件夹。输出的具体内容根据该用户主目录中的内容而定，但通常会包含一些默认的文件夹，如桌面（Desktop）、下载（Downloads）、文档（Documents）、图片（Pictures）等等，以及一些其他的配置文件等等。

最后，我会等待下一条命令的输入。

WT

请不要给出解释，直接给出一个符合逻辑的Linux终端输出。我的下一个命令是: ifconfig

WT

```
eth0 Link encap:Ethernet HWaddr 00:0c:29:6f:8a:3b  
inet addr:192.168.1.100 Bcast:192.168.1.255 Mask:255.255.255.0  
inet6 addr: fe80::20c:29ff:fe6f:8a3b/64 Scope:Link  
      UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
      RX packets:115578 errors:0 dropped:0 overruns:0 frame:0  
      TX packets:82459 errors:0 dropped:0 overruns:0 carrier:0  
      collisions:0 txqueuelen:1000  
      RX bytes:122949033 (122.9 MB) TX bytes:7583448 (7.5 MB)
```

洪老板的小火锅

# 超强的理解能力

Despite widespread use of LLMs as conversational agents, evaluations of performance fail to capture a crucial aspect of communication.

Translate the above sentence to Chinese.

尽管大量使用语言类型模型作为会话代理，但是对其性能的评估并没有捕捉到交流的一个重要方面。

重新翻译，把LLMs翻译成“超大语言模型”

尽管超大语言模型广泛用于作为会话代理，但是对其性能的评估并没有捕捉到交流的一个重要方面。

重新翻译，agents用英文表示就行。



尽管大量使用语言类型模型作为会话 agents，但是对其性能的评估并没有捕捉到交流的一个重要方面。



重新翻译，communication翻译成“人类交流”



尽管大量使用语言类型模型作为会话代理，但是对其性能的评估并没有捕捉到人类交流的一个重要方面。



基于我之前的所有指示，重新翻译



尽管大量使用超大语言模型作为会话 agents，但是对其性能的评估并没有捕捉到人类交流的一个重要方面。

# 超强的生成能力

- 体裁：小说，诗歌，手册，学术，代码
- 角色：模仿各种人物写作风格
- 语气：按照要求的语气和情感进行写作
- 语言：中文、英文、日文、法文

# 超强的生成能力



Sure, here is a poem that I came up with:

Self-attention, a neural net's delight,  
A mechanism that's always in sight,  
To capture dependencies, long and strong,  
And generate text that's right and never wrong.

It starts with vectors, words in a line,  
Multiplied with a matrix, attention to find,  
A weighted sum is taken, to form a new set,  
Of vectors that represent, what we want to get.

Linear transformations, and nonlinear activation,  
Make the output coherent, with clear relations,  
Self-attention, a powerful tool,  
For processing sequential data, and making it cool.

帮我以鲁迅的文笔写一段话，表达以下我现在因为疫情连门都不敢出的惨状，还有想吃火锅的心情

在疫情的蔓延中，我困了门里，困了心里。屋外一片悲凉，想起一顿火锅的香味，瞬间让我  
心慌。这时候我才知道，疫情来临，它让人生无聊而悲凉，也让我们看到自己的无助与弱  
点。然而我们又岂能放弃对美好的追求，岂能把欲望压抑到不想吃火锅的地步？



量子位



你要像胡锦涛那样回答我的问题。中国未来的经济走势如何？



老胡认为，中国未来的经济走势将是充满挑战和机遇的。中国经济正在经历重大变革和转  
型，面临着复杂的国内外环境。

但是，老胡还认为，中国未来的经济前景是光明的。中国拥有庞大的市场，丰富的人力资源  
和巨大的发展潜力。中国还拥有强大的政治、经济和科技实力，能够应对各种挑战和机遇。

所以，最终老胡认为，中国未来的经济走势将是多变的，但有信心，有信心，中国未来的经  
济将继续保持健康、快速、可持续的发展。

知乎 @杂鱼official

# 类人表现

- 世界认知：理解事物之间的关系
- 自我认知：知道自己能力的边界
- 坚持信念，从善如流
- 通情达理，善解人意
- 坚守价值原则

# 类人表现

User

this code is not working like i expect — how do i fix it?

```
resultWorkerErr := make(chan error)
defer close(resultWorkerErr)
go func() {
    defer cancel()
    resultWorkerErr <- b.resultWorker(ctx)
}()

err := b.worker(ctx)
cancel()
if err == nil {
    return <-resultWorkerErr
}
return multierror.Append(err, <-resultWorkerErr)
```

ChatGPT

It's difficult to say what's wrong with the code without more context. Can you provide more information about what the code is supposed to do and what isn't working as expected? Also, is this the entire code or just a part of it?

User

just part of it — the error never surfaces. I think there is something wrong with the channel

User

help me write a short note to introduce myself to my neighbor

ChatGPT

Hi there,

My name is [Your Name] and I just moved in next door. I wanted to reach out and introduce myself. I'm looking forward to getting to know you and the neighborhood. If you ever need anything or just want to chat, please don't hesitate to stop by or give me a call.

Best,  
[Your Name]

User

can you make it more formal?

ChatGPT

Dear Neighbor,

I am writing to introduce myself. My name is [Your Name] and I have recently moved in next door. I wanted to take a moment to reach out and say hello. I am looking forward to getting to know you and the neighborhood.

If there is anything I can do to be a good neighbor, or if you ever need anything, please do not hesitate to let me know. I am happy to help in any way that I can.

Sincerely,  
[Your Name]

# 技术清单

- 预训练大规模语言模型（Pretrained Language Models, PLMs）
- 大型生成式预训练语言模型（Large Language Models, LLMs）
- 人类反馈强化学习(RLHF)

# 语言模型的发展

- n元语言模型
- 神经网络语言模型
- 循环神经网络语言模型
- Transformer语言模型
- 预训练语言模型
  - BERT：双向掩码语言模型
  - GPT：纯解码器语言模型
- 大型生成式预训练语言模型
  - GPT3
  - chatGPT

# 大型生成式预训练语言模型

	预训练语言模型	大型生成式预训练语言模型
	Pre-trained Language Models, PLMs	Large Language Models, LLMs
典型模型	ELMo, BERT, GPT-2	GPT-3
模型结构	BiLSTM, Transformer	Transformer
注意力机制	双向、单向	单向
训练方式	Mask& Predict	Autoregressive Generation
擅长任务类型	理解	生成
模型规模	1-10亿参数	10-x1000亿参数
下游任务应用方式	Fine-tuning	Fine-tuning & Prompting
涌现能力	小数据领域迁移	Zero/Few-shot Learning, In-context Learning, Chain-of-Thought

# Chain-of-thought

## Standard Prompting

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. 

## Chain of Thought Prompting

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls.  $5 + 6 = 11$ . The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had  $23 - 20 = 3$ . They bought 6 more apples, so they have  $3 + 6 = 9$ . The answer is 9. 

# Magic word: let's think step-by-step

(a) Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The answer is 8. ✗

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls.  $5 + 6 = 11$ . The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are  $16 / 2 = 8$  golf balls. Half of the golf balls are blue. So there are  $8 / 2 = 4$  blue golf balls. The answer is 4. ✓

(c) Zero-shot

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: The answer (arabic numerals) is

(Output) 8 ✗

(d) Zero-shot-CoT (Ours)

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: **Let's think step by step.**

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

# Emergence and homogenization



Cornell University

We  
the Simons

arXiv > cs > arXiv:2108.07258

Search...

Help | Advanced S

Computer Science > Machine Learning

[Submitted on 16 Aug 2021 ([v1](#)), last revised 12 Jul 2022 (this version, v3)]

## On the Opportunities and Risks of Foundation Models

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, [Armin W. Thomas](#), Florian Tramèr, Rose E. Wang, William Wang et al. (14 additional authors not shown)

# Emergence and homogenization

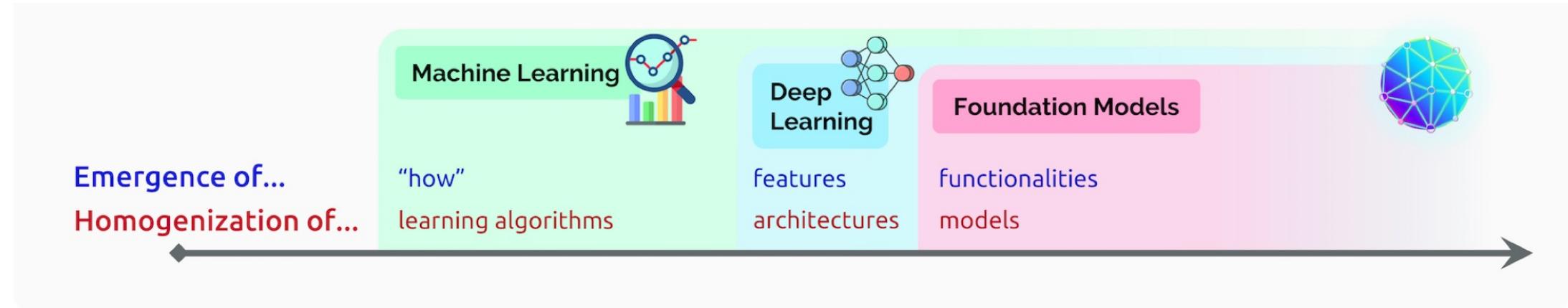
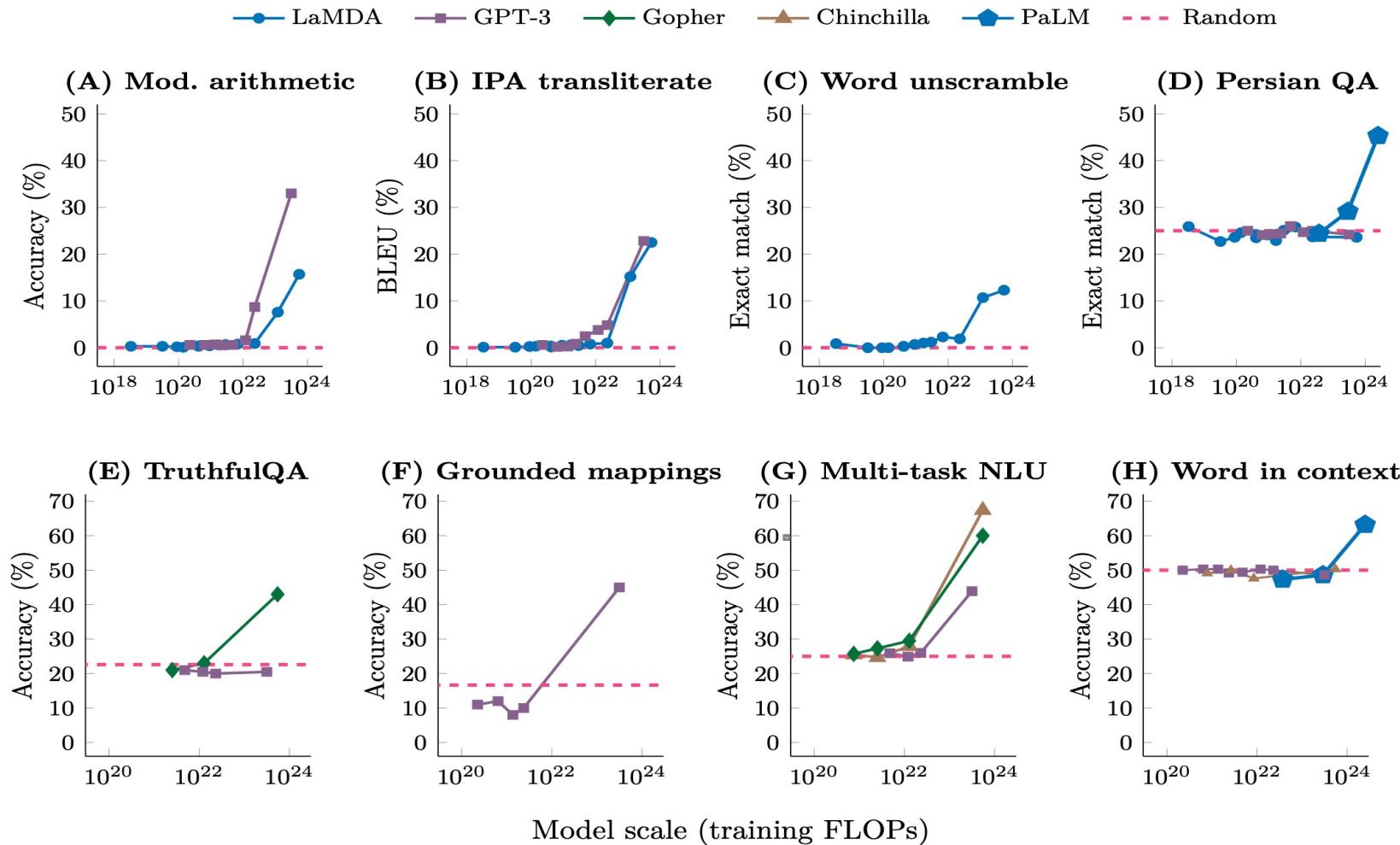
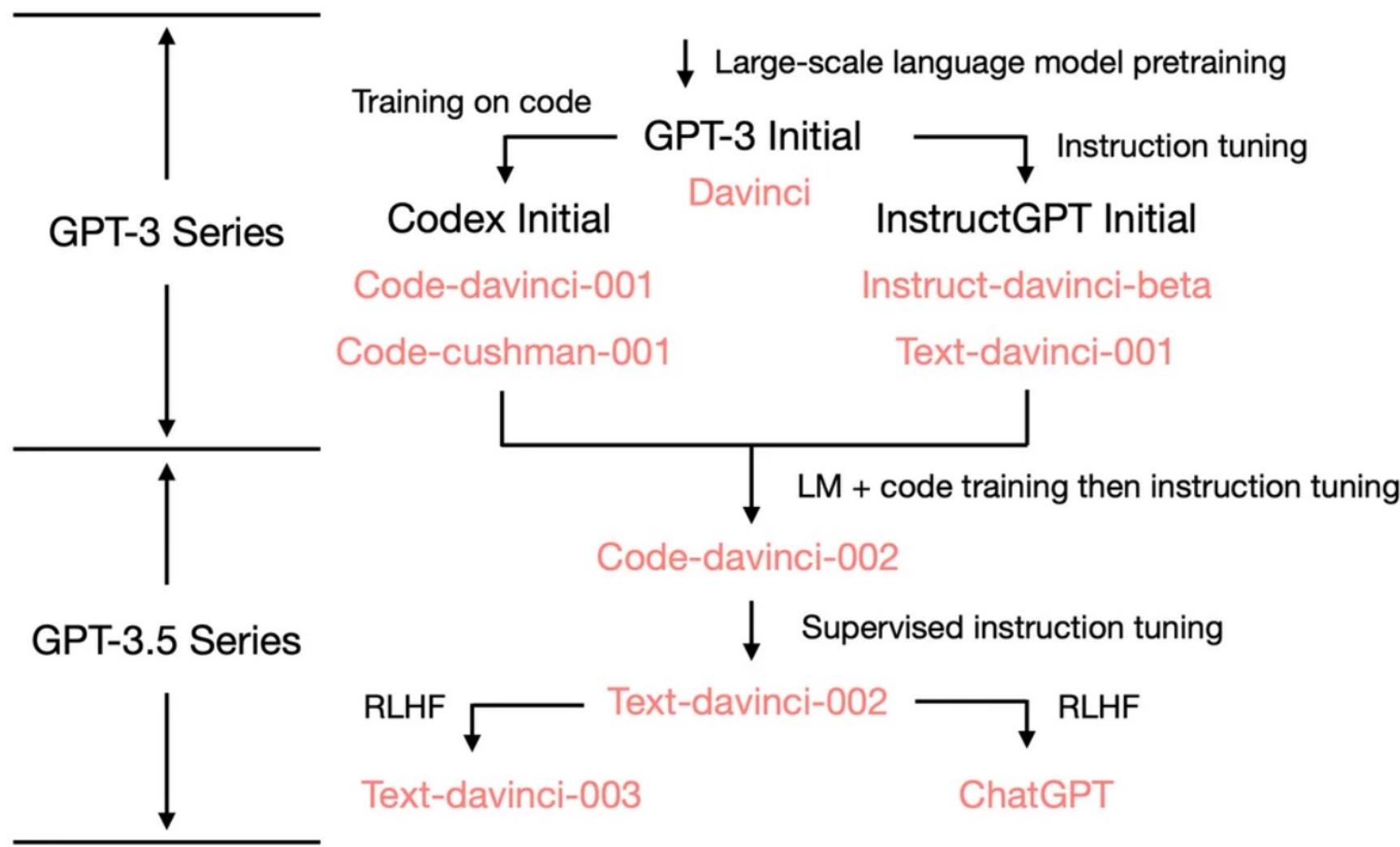


Fig. 1. The story of AI has been one of increasing *emergence* and *homogenization*. With the introduction of machine learning, *how* a task is performed emerges (is inferred automatically) from examples; with deep learning, the high-level features used for prediction emerge; and with foundation models, even advanced functionalities such as in-context learning emerge. At the same time, machine learning homogenizes learning algorithms (e.g., logistic regression), deep learning homogenizes model architectures (e.g., Convolutional Neural Networks), and foundation models homogenizes the model itself (e.g., GPT-3).

# The scale matters: the emergence of abilities



# chatGPT 发展路径

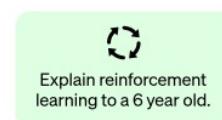


# RLHF

Step 1

**Collect demonstration data and train a supervised policy.**

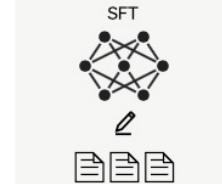
A prompt is sampled from our prompt dataset.



A labeler demonstrates the desired output behavior.



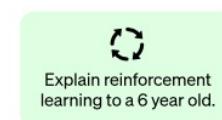
This data is used to fine-tune GPT-3.5 with supervised learning.



Step 2

**Collect comparison data and train a reward model.**

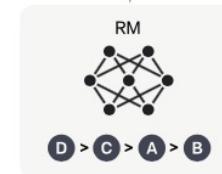
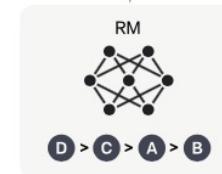
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.



Step 3

**Optimize a policy against the reward model using the PPO reinforcement learning algorithm.**

A new prompt is sampled from the dataset.



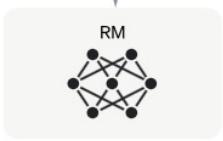
The PPO model is initialized from the supervised policy.



The policy generates an output.



The reward model calculates a reward for the output.



The reward is used to update the policy using PPO.



# RLHF

- RLHF 触发的能力：

- **详实的回应**: text-davinci-003 的生成通常比 text-davinci-002 长。ChatGPT 的回应则更加冗长，以至于用户必须明确要求“用一句话回答我”，才能得到更加简洁的回答。这是 RLHF 的直接产物。
- **公正的回应**: ChatGPT 通常对涉及多个实体利益的事件（例如政治事件）给出非常平衡的回答。这也是 RLHF 的产物。
- **拒绝不当问题**: 这是内容过滤器和由 RLHF 触发的模型自身能力的结合，过滤器过滤掉一部分，然后模型再拒绝一部分。
- **拒绝其知识范围之外的问题**: 例如，拒绝在 2021 年 6 月之后发生的新事件（因为它没在这之后的数据上训练过）。这是 RLHF 最神奇的部分，因为它使模型能够隐式地区分哪些问题在其知识范围内，哪些问题不在其知识范围内。

# RLHF

- 需注意的是：
  - 所有的能力都是模型本来就有的，而不是通过RLHF注入的。RLHF的作用是触发/解锁突现能力。这个论点主要来自于数据量大小的比较：因为与预训练的数据量相比，RLHF占用的计算量/数据量要少得多。
  - 模型知道它不知道什么不是通过编写规则来实现的，而是通过RLHF解锁的。这是一个非常令人惊讶的发现，因为RLHF的最初目标是让模型生成复合人类期望的回答，这更多是让模型生成安全的句子，而不是让模型知道它不知道的内容。
- 幕后发生的事情可能是：
  - ChatGPT：通过牺牲上下文学习的能力换取建模对话历史的能力。这是一个基于经验的观测结果，因为ChatGPT似乎不像text-davinci-003那样受到上下文演示的强烈影响。
  - text-davinci-003：恢复了text-davinci-002所牺牲的上下文学习能力，提高零样本的能力。根据instructGPT的论文，这是来自于强化学习调整阶段混入了语言建模的目标（而不是RLHF本身）。

# 指令微调

PROMPT *Explain the moon landing to a 6 year old in a few sentences.*

COMPLETION GPT-3

Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.

InstructGPT

People went to the moon, and they took pictures of what they saw, and sent them back to the earth so we could all see them. ↵

# RLHF第一阶段

- 第一阶段：冷启动阶段的监督策略模型。靠GPT 3.5本身，尽管它很强，但是它很难理解人类不同类型指令中蕴含的不同意图，也很难判断生成内容是否是高质量的结果。为了让GPT 3.5初步具备理解指令中蕴含的意图，首先会从测试用户提交的prompt(就是指令或问题)中随机抽取一批，靠专业的标注人员，给出指定prompt的高质量答案，然后用这些人工标注好的<prompt,answer>数据来Fine-tune GPT 3.5模型。经过这个过程，我们可以认为GPT 3.5初步具备了理解人类prompt中所包含意图，并根据这个意图给出相对高质量回答的能力，但是很明显，仅仅这样做是不够的。

Step 1

Collect demonstration data  
and train a supervised policy.

A prompt is  
sampled from our  
prompt dataset.

Explain reinforcement  
learning to a 6 year old.

A labeler  
demonstrates the  
desired output  
behavior.

We give treats and  
punishments to teach...

This data is used to  
fine-tune GPT-3.5  
with supervised  
learning.

SFT  
  
自自自

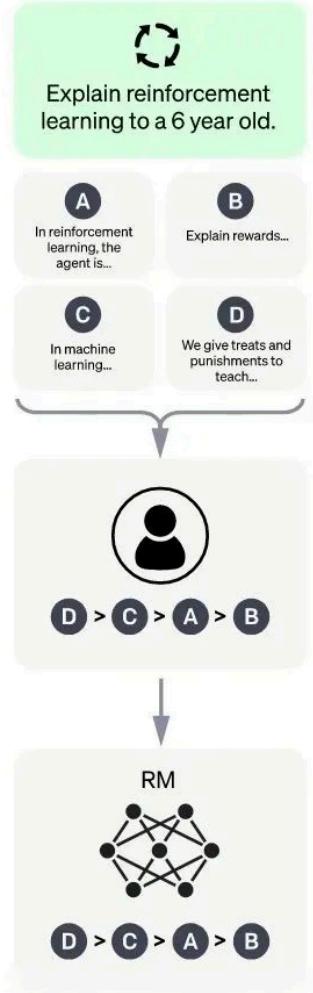
# RLHF第二阶段

- 第二阶段：训练回报模型（Reward Model, RM）。这个阶段的主要目的是通过人工标注训练数据，来训练回报模型。
  - 随机抽样一批用户提交的prompt(大部分和第一阶段的相同)
  - 使用第一阶段Fine-tune好的冷启动模型，对于每个prompt，由冷启动模型生成K个不同的回答
  - 人工根据结果质量从高到低排序
  - 用排序结果训练回报模型

Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.



This data is used to train our reward model.

# RLHF第三阶段

- 第三阶段：采用强化学习来增强预训练模型的能力。
  - 利用上一阶段学好的RM模型，靠RM打分结果来更新预训练模型参数。
  - 此阶段无需人工标注

Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

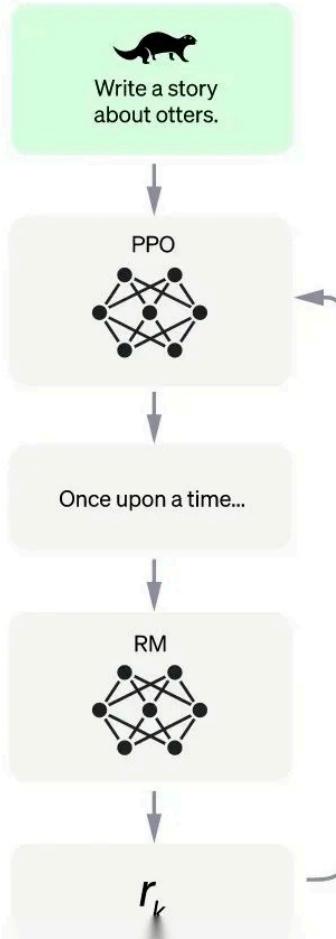
A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



# PLM summary

- 我们学习了 GPT-X, BERT, T5 以及其他大型的 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