CS480/680: Introduction to Machine Learning

Lecture 11: Transformer

Hongyang Zhang

UNIVERSITY OF WATERLOO

March 5, 2024

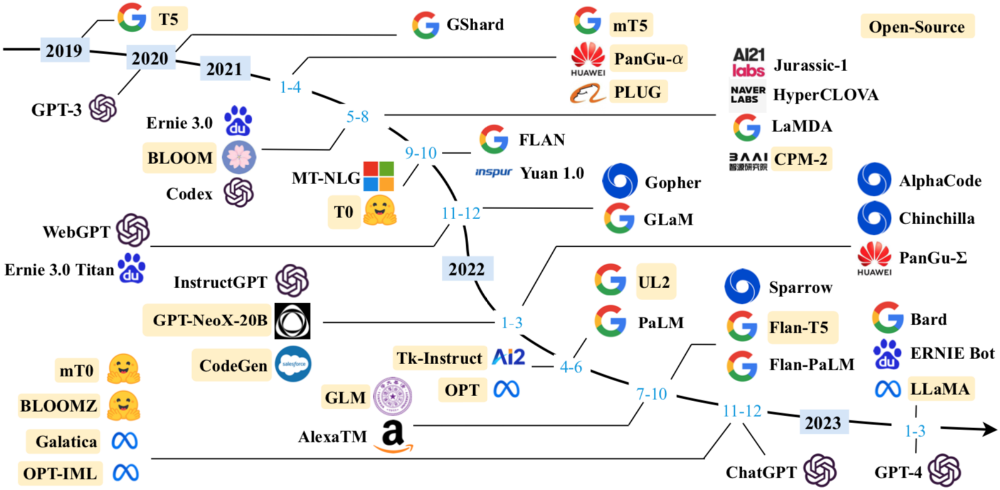

OpenAI
ChatGPT

Capability

- Writing homeworks
- Writting codes
- Analyzing texts
- Taking exams
- Reasoning
- Interacting with the real world
- ....

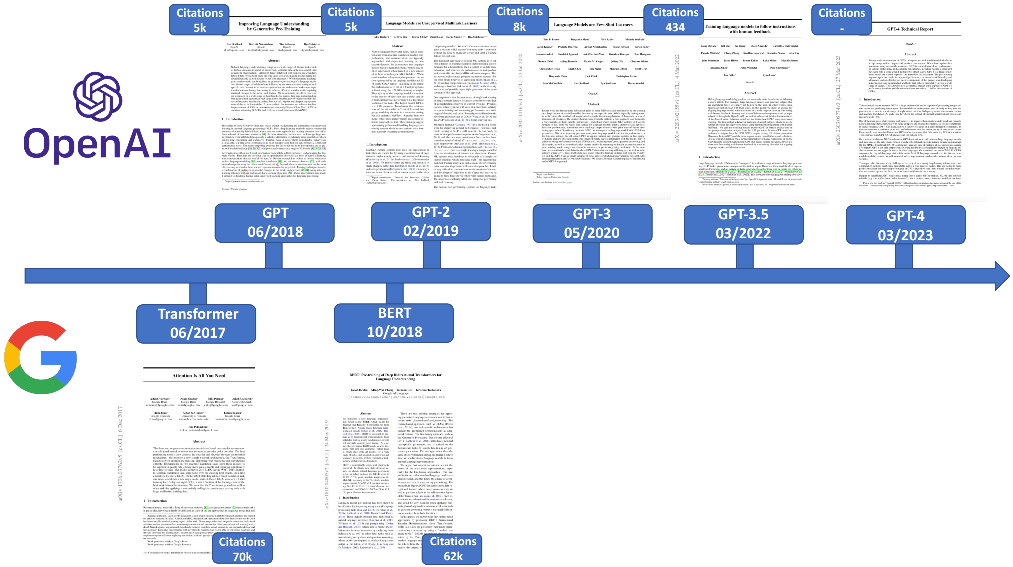
Exam	GPT-4	GPT-3.5
Uniform Bar Exam	298/400 (~90th)	213/400 (~10th)
LSAT	163/180 (~88th)	149/180 (~40th)
SAT Reading & Writing	710/800 (~93rd)	670/800 (~87th)
SAT Math	700/800 (~89th)	590/800 (~70th)
GRE Verbal	169/170 (~99th)	154/170 (~63rd)
GRE Writing	4/6 (~54th)	4/6 (~54th)
AP Biology	5/5 (85th-100th)	4/5 (62nd-85th)
AP Calculus BC	4/5 (43rd-59th)	1/5 (0th-7th)
AP Chemistry	4/5 (71st-88th)	2/5 (22nd-46th)
AP English Language and Composition	2/5 (14th-44th)	2/5 (14th-44th)
AP English Literature and Composition	2/5 (8th-22nd)	2/5 (8th-22nd)
AP Macroeconomics	5/5 (84th-100th)	2/5 (33rd-48th)
Introductory Sommelier	92%	80%
Advanced Sommelier	77%	46%
Leetcode (easy)	31/41	12/41
Leetcode (hard)	3/45	0/45

A Big Picture of Foundation Models



W. Zhao et al. "A Survey of Large Language Models". arXiv:2303.18223.

OpenAI vs. Google in Papers



Papers to Read

- (Transformer) Attention Is All You Need
- (GPT) Improving Language Understanding by Generative Pre-training
- (BERT) BERT: Pre-training of Deep Bidirectional Transformer for Language Understanding
- (GPT-2) Language Models are Unsupervised Multitask Learners
- (GPT-3) Language Models are Few-Shot Learners
- (GPT-3.5) Training Language Models to follow Instructions with Human Feedbacks
- (GPT-4) GPT-4 Technical Report

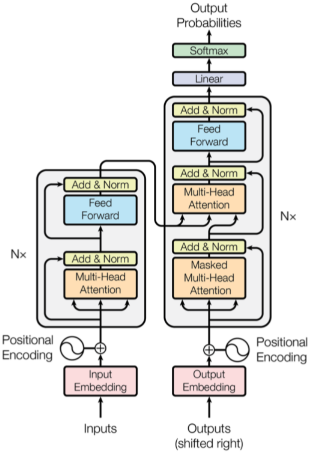
Papers to Read

- (Transformer) Attention Is All You Need
- (GPT) Improving Language Understanding by Generative Pre-training
- (BERT) BERT: Pre-training of Deep Bidirectional Transformer for Language Understanding
- (GPT-2) Language Models are Unsupervised Multitask Learners
- (GPT-3) Language Models are Few-Shot Learners
- (GPT-3.5) Training Language Models to follow Instructions with Human Feedbacks
- (GPT-4) GPT-4 Technical Report

Transformer is for sequence-to-sequence tasks

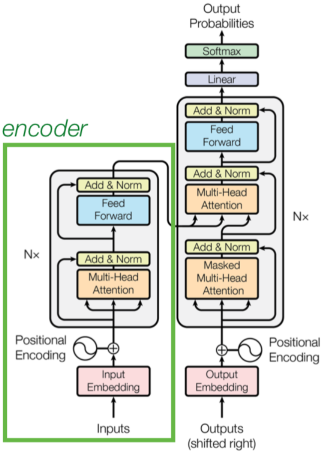
- Initially, transformer is designed for machine translation tasks
  - **Goal:** given English sentence X with words (a.k.a. tokens) $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$, produce Chinese translation Y with words/tokens $\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_m$
- Now transformer is used for other sequence-to-sequence tasks, e.g., QA tasks
- **Mathematical goal:** output prob $\mathrm{argmax}_Y \, p(\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_m | \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$

Transformer Architecture



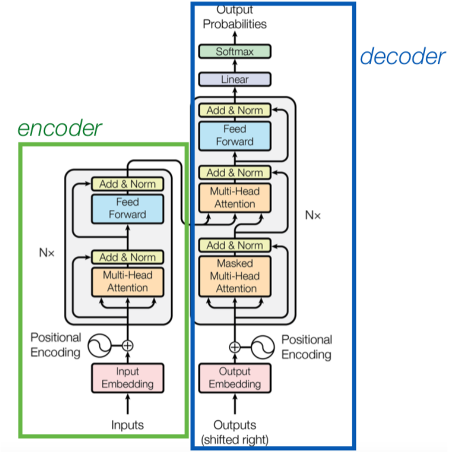
A. Vaswani et al. "Attention is All you Need". In: Advances in Neural Information Processing Systems 30. 2017, pp. 5998–6008.

Transformer Architecture



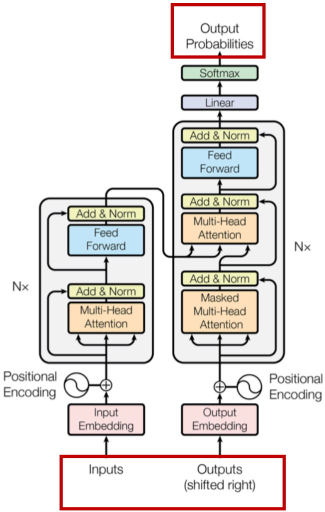
A. Vaswani et al. "Attention is All you Need". In: Advances in Neural Information Processing Systems 30. 2017, pp. 5998–6008.

Transformer Architecture



A. Vaswani et al. "Attention is All you Need". In: Advances in Neural Information Processing Systems 30. 2017, pp. 5998–6008.

Transformer Architecture

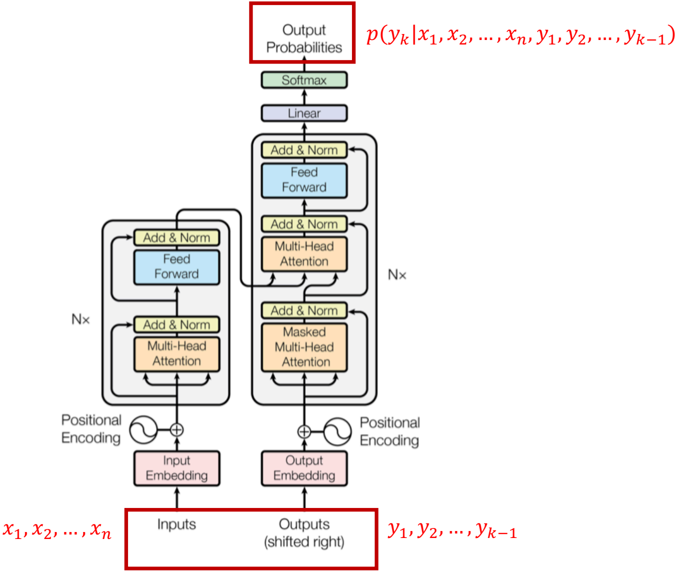


Input and Output

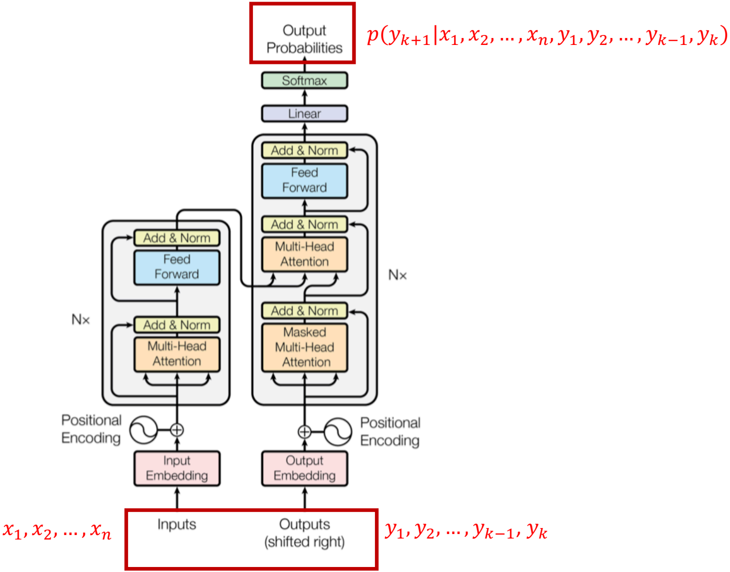
- Input sequence $X = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$, a.k.a. the prompt
- Output sequence $Y = (\mathbf{y}_1, \ldots, \mathbf{y}_m)$
- Mathematical goal: compute $\mathrm{argmax}_Y\, p(\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_m | \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$
  - One token after another (greedy): $\mathrm{argmax}_{\mathbf{y}_k}\, p(\mathbf{y}_k | \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n, \mathbf{y}_1, ..., \mathbf{y}_{k-1})$
  - This is also known as auto-regressive
- Examples:

Step 0 X: Where is University of Waterloo?
Step 1 Y: [START]; $\Pr$(It | X [START]) highest
Step 2 Y: [START] It; $\Pr$(is | X [START] It) highest
Step 3 Y: [START] It is; $\Pr$(at | X [START] It is) highest
Step 4 Y: [START] It is at; $\Pr$(Waterloo | X [START] It is at) highest
Step 5 Y: [START] It is at Waterloo; $\Pr$([END] | X [START] It is at Waterloo) highest
Step 6 Y: [START] It is at Waterloo [END]

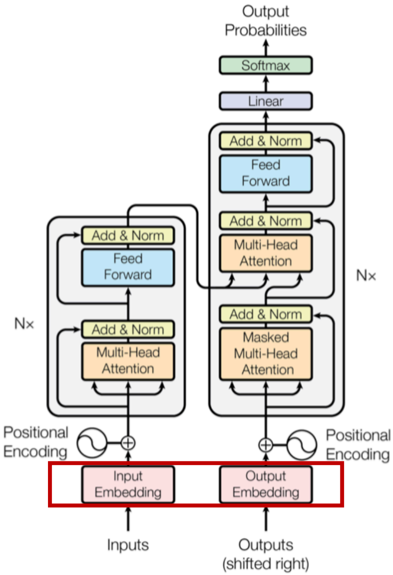
Transformer Architecture (k-th step)



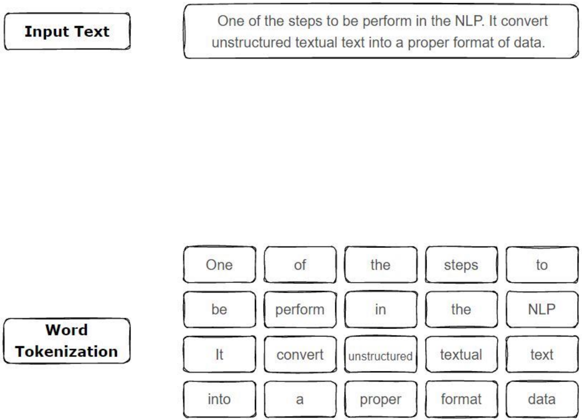
Transformer Architecture ($k+1$-th step)



Transformer Architecture

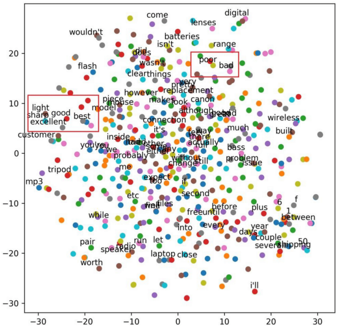


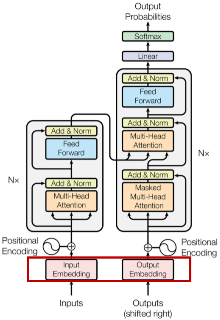
Tokenizer



- *tiktoken* is a fast tokenizer for use with OpenAI's models
- `https://github.com/openai/tiktoken`

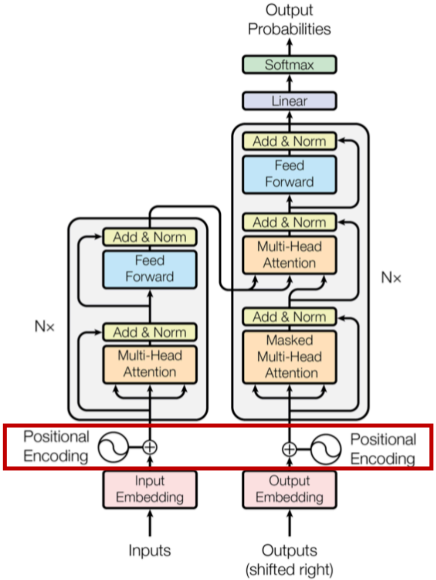
Token Embedding





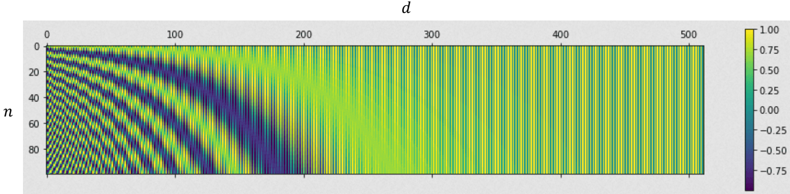
- A mapping from tokens to vectors
  - Convert the input tokens to vectors of dimension $d = 512$
- Token embedding mapping is trained independently from the LLM.
- Good properties: words of similar meaning are close in the embedding space

Transformer Architecture



Positional Encoding

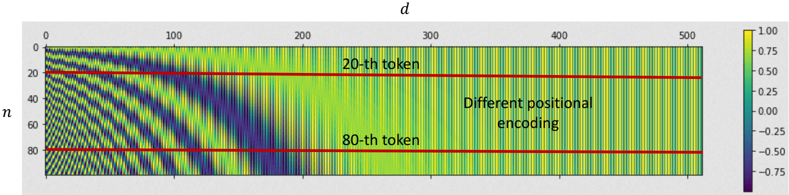
- Word order matters, different meanings:
  - THE CHICKEN crossed the road
  - the road THE CHICKEN crossed
  - THE CHICKEN the road crossed
  - crossed the road THE CHICKEN
- Positional encoding: $W^p \in \mathbb{R}^{n \times d}$
  - $W^p_{t,2i} = \sin(t/10000^{2i/d}),\ W^p_{t,2i+1} = \cos(t/10000^{2i/d}),\ i = 0, \ldots, \frac{d}{2} - 1$



- NO parameter to be learned
- Simply add W^p to the $n \times d$ token embedding

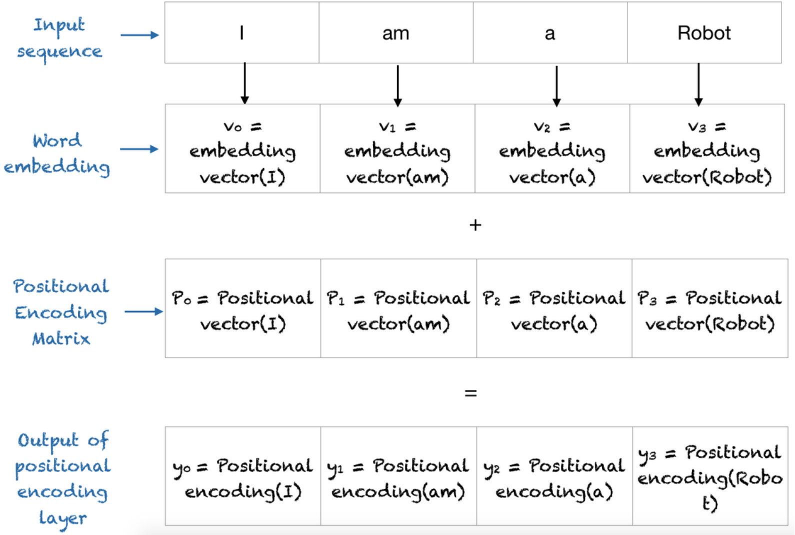
Positional Encoding

- Word order matters, different meanings:
  - THE CHICKEN crossed the road
  - the road THE CHICKEN crossed
  - THE CHICKEN the road crossed
  - crossed the road THE CHICKEN
- Positional encoding matrix: $W^p \in \mathbb{R}^{n \times d}$
  - $W^p_{t,2i} = \sin(t/10000^{2i/d}),\ W^p_{t,2i+1} = \cos(t/10000^{2i/d}),\ i = 0, \ldots, \frac{d}{2} - 1$

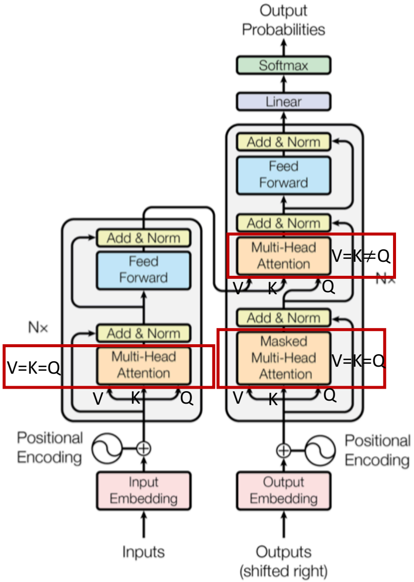


- NO parameter to be learned
- Simply add W^p to the $n \times d$ token embedding

Positional Encoding

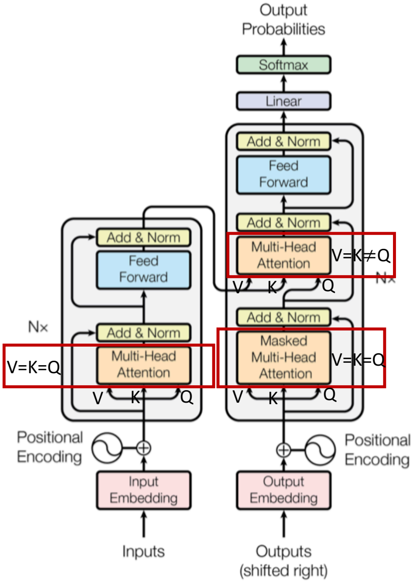


Transformer Architecture

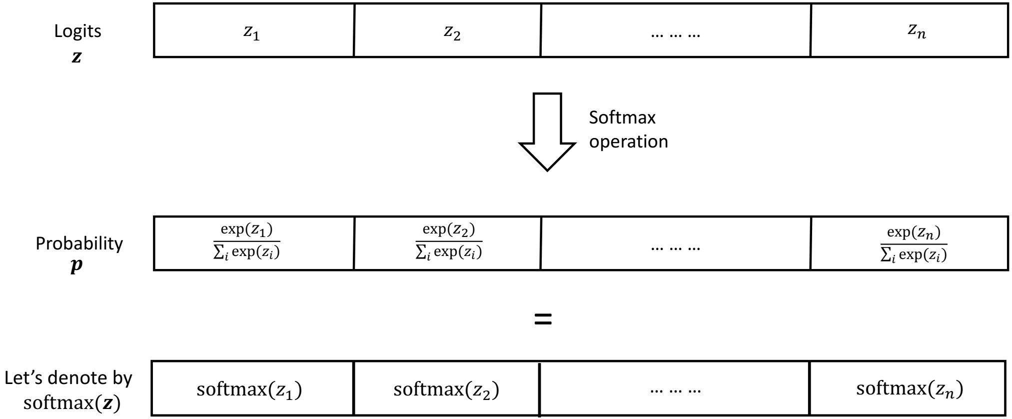


Attention Layer Inputs and Outputs

- Inputs: Value $V \in \mathbb{R}^{n \times d}$, Key $K \in \mathbb{R}^{n \times d}$, Query $Q \in \mathbb{R}^{m \times d}$
- Output: an $m \times d$ matrix



Softmax Recap



Attention Layer

Let $\mathbf{v}_i^\top$, $\mathbf{k}_i^\top$ and $\mathbf{q}_i^\top$ stand for the row vectors of value, key and query. Let

$$V = \begin{bmatrix} \mathbf{v}_1^\top \\ \vdots \\ \mathbf{v}_n^\top \end{bmatrix} \in \mathbb{R}^{n\times d}, \quad K = \begin{bmatrix} \mathbf{k}_1^\top \\ \vdots \\ \mathbf{k}_n^\top \end{bmatrix} \in \mathbb{R}^{n\times d}, \quad Q = \begin{bmatrix} \mathbf{q}_1^\top \\ \vdots \\ \mathbf{q}_m^\top \end{bmatrix} \in \mathbb{R}^{m\times d}.$$

Then (Softmax operation is row-wise, i.e., $\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}$):

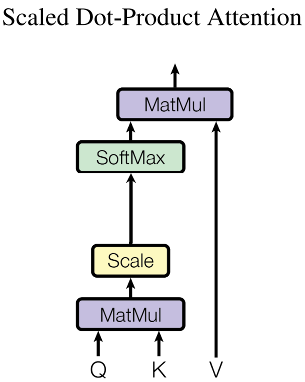
$$\begin{aligned} \text{Attention}(V, K, Q) &= \texttt{softmax}(\frac{QK^\top}{\sqrt{d}})V \\ &= \begin{bmatrix} \texttt{softmax}(\frac{\langle \mathbf{q}_1, \mathbf{k}_1 \rangle}{\sqrt{d}})\mathbf{v}_1^\top + \ldots + \texttt{softmax}(\frac{\langle \mathbf{q}_1, \mathbf{k}_n \rangle}{\sqrt{d}})\mathbf{v}_n^\top \\ \vdots \\ \texttt{softmax}(\frac{\langle \mathbf{q}_m, \mathbf{k}_1 \rangle}{\sqrt{d}})\mathbf{v}_1^\top + \ldots + \texttt{softmax}(\frac{\langle \mathbf{q}_m, \mathbf{k}_n \rangle}{\sqrt{d}})\mathbf{v}_n^\top \end{bmatrix} \in \mathbb{R}^{m\times d} \end{aligned}$$

- Inner product $\langle \mathbf{q}_i, \mathbf{k}_j \rangle$ measures similarity: more similar, more contributions

Matrix Form of Attention

$\text{Attention}(V, K, Q) = \texttt{softmax}(\frac{QK^\top}{\sqrt{d}})V$

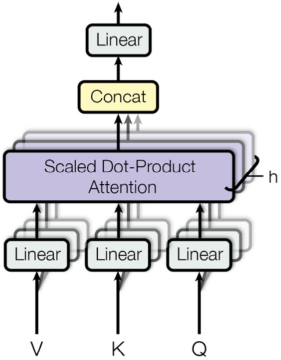
- Each output is a convex combination of value rows
- Self-attention: $Q = K = V$
- There is NO learnable parameter so far!



Learnable Attention Layer and Multi-head Attention

$$\text{Attention}(VW^v, KW^k, QW^q)$$
$$= \texttt{softmax}\left(\frac{QW^q(KW^k)^\top}{\sqrt{d}}\right)VW^v$$

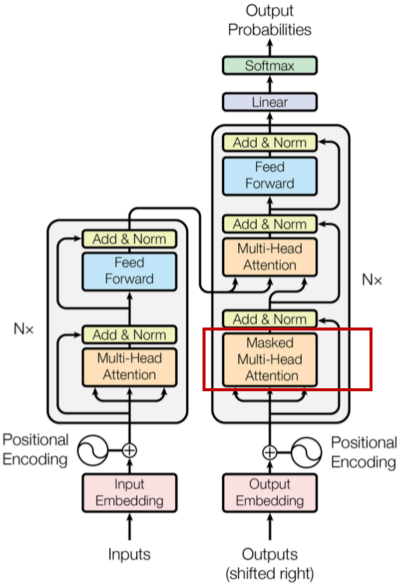
Multi-Head Attention



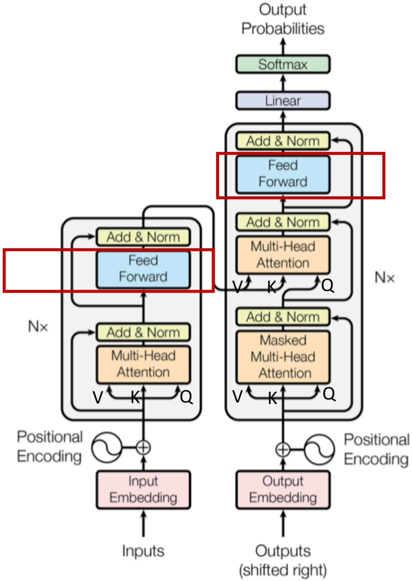
- Replace Q, K and V with QW^q, KW^k and VW^v
- $\{W^q, W^k, W^v\} \in \mathbb{R}^{512\times 64}$ are learnable linear layers
- Can add $h = 8$ linear layers W_i^q's, W_i^k's and W_i^v's in parallel and concatenate their output later (output dim $= 64 \times 8 = 512$)

Masked Multi-head Attention

- We should not look at future words. Masking them:
  - E.g., we have already outputted "University of Waterloo", and we want to predict the next word
  - University of Waterloo $\underbrace{\text{locates}}_{\text{[Mask]}}\ \underbrace{\text{at}}_{\text{[Mask]}}\ \underbrace{\text{Waterloo}}_{\text{[Mask]}}$
- Input the masked sequence into the attention layer



Transformer Architecture



Feed-Forward Layer

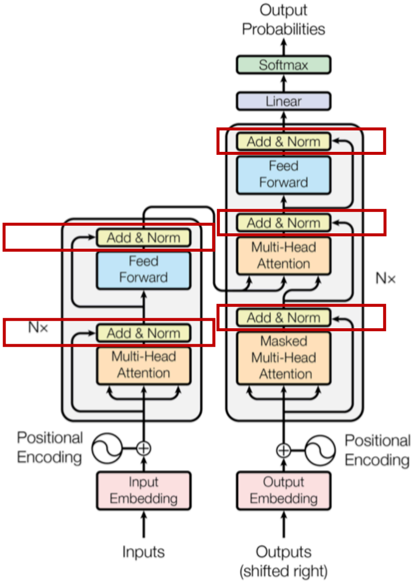
- Feed-Forward Network

$$\texttt{MLP}(\mathbf{x}) = \max(0, \mathbf{x}^\top W_1 + \mathbf{b}_1^\top) \cdot W_2 + \mathbf{b}_2^\top$$

  - Two-layer MLP
  - ReLU activation
  - $W_1 \in \mathbb{R}^{d \times 4d}$, $W_2 \in \mathbb{R}^{4d \times d}$

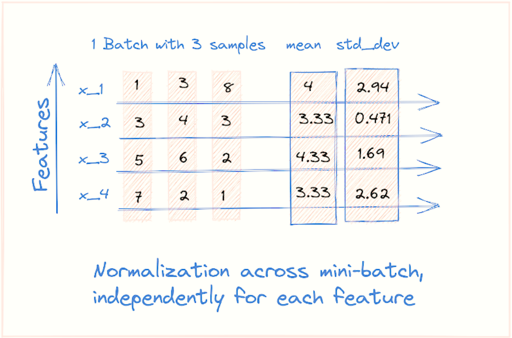
- Residual connections and layer normalization

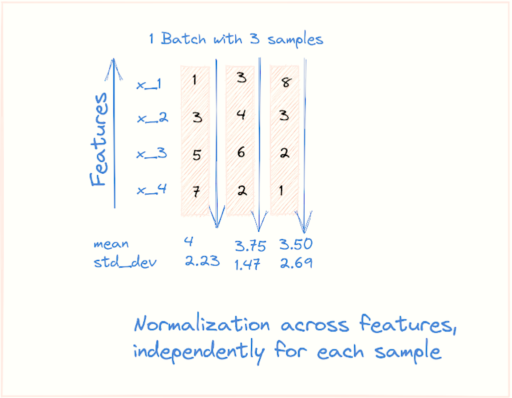
Transformer Architecture



Layer Normalization

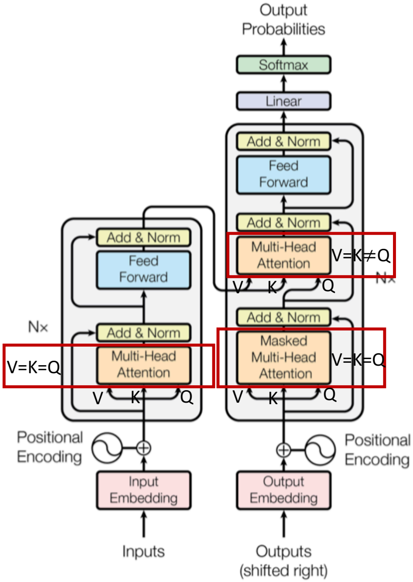
- Batch size (# sequences) is often small, e.g., 1, in the NLP tasks
- Therefore, batch normalization might not be a good choice
- Often use layer normalization instead (normalization across the features)





Overview of Transformer

- Only three tunable hyper-parameters:
  - Number of layers: $N = 6$
  - Output dimension of all modules is $d = 512$
  - Number of heads: $h = 8$
- The module that connects encoder and decoder is a multi-head attention, where value and key are from encoder, and query is from decoder
- In the other two attention modules, value=key=query

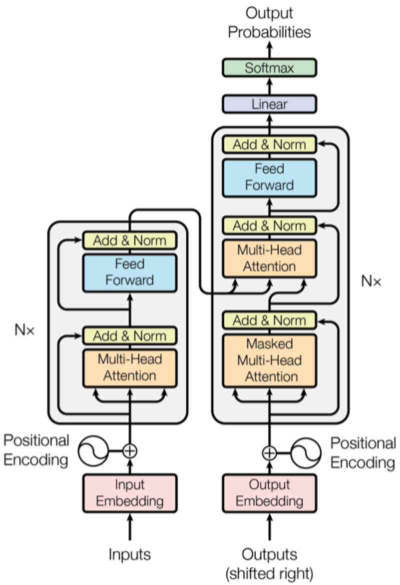


Transformer Loss

- Pretraining Task: predict next words
- Train by minimizing the log-loss between true next word and predicted next word:

$$\min_{W} \hat{\mathbb{E}} \left[- \left\langle Y, \log \hat{Y} \right\rangle \right]$$

  - $Y = [\mathbf{y}_1, \ldots, \mathbf{y}_l]$ is output sequence, one-hot
  - $\hat{Y} = [\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_l]$ is the predicted probabilities



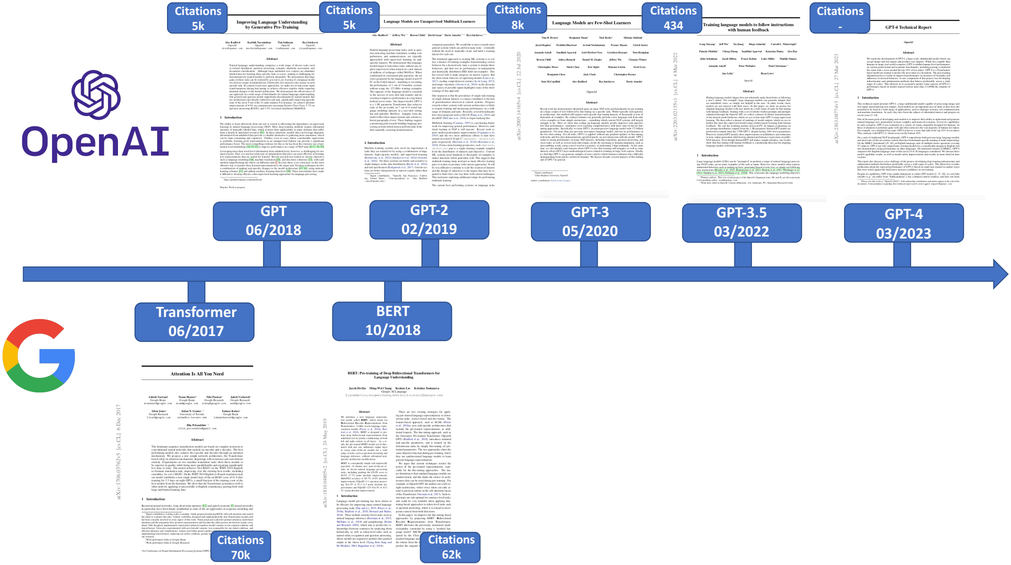
Does It Work?

The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

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]		39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]		40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]		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	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

We will see more experiments in the next lecture “Large Language Models”.

Contents in the Next Lecture




Questions
?
?
Answers
?