Transformers

Natural Language Processing

(based on revision of John Hewitt Lectures)

Announcement

- TA announcements (if any)...

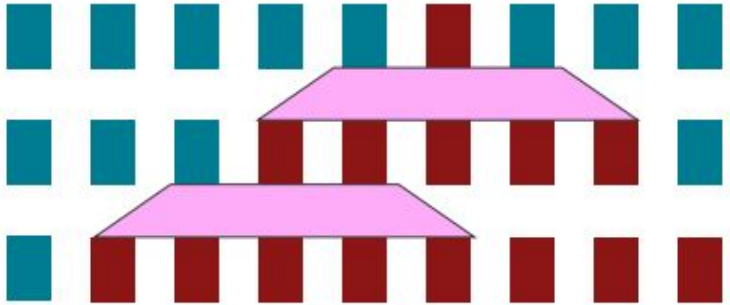
Suggested Readings

1. Attention Is All You Need (original paper)
2. The Illustrated Transformer (good visuals; good for understanding but can be confusing when coding)
3. Transformer (Google AI blog post)
4. Layer Normalization
5. Image Transformer (using transformers on images)
6. Music Transformer: Generating music with long-term structure (using transformers on music)

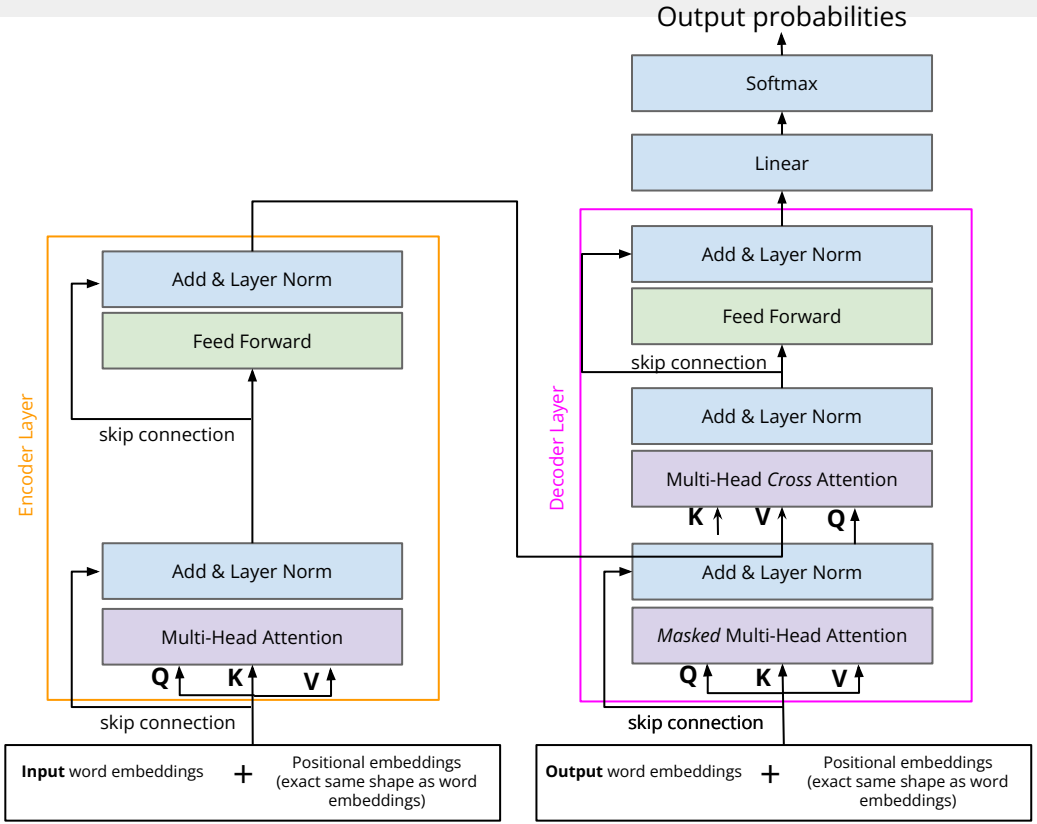
Transformers

Limitations of RNNs

- Ineffective to learn **long-distance relationships**
- Does not allow **parallelizability**
  - Inhibits training on very large datasets
- **Stacked 1D Convolution** may help but the window size can never be large enough!



Transformers [Vaswani et al., NeuroIPS 2017]



- Originally developed for **seq2seq** tasks, e.g., MT
- **Idea**: Use **attention ONLY** (without RNN) will
  - enable **parallelizability** hence more efficient
  - learn **long-range dependencies** better
  - provides **interpretability**
- **Obstacles**:
  - **Loss of sequential** information
  - The **decoding side** could be a problem if we do everything at the same time, because the input will see the answer (unlike the sequential RNN where each set of input is used to predict the output)
  - Need to be careful that the whole architecture is **always parallelizable at almost every step**

Attention Is All You Need, Vaswani et al. 2017, https://arxiv.org/pdf/1706.03762.pdf

Transformers - Positional Embeddings

- Inputting the word embeddings directly to an attention-only module would be **silly**, since attention **does not capture sequential information** (i.e., we lose that because we did not use RNN)
- To address this limitation, we simply create a **positional embeddings** with exact shape as the word embeddings, and then add them together.
- Let $\mathbf{x}_1, \cdots, \mathbf{x}_T$ be the input word embeddings (one vec per word. We can stack them into a big matrix

$$\mathbf{X}; \quad \mathbf{X} \in \mathbb{R}^{T \times d}$$

where T is the number of words and d is the embedding size

- We can create a positional embedding

$$\mathbf{P}; \quad \mathbf{P} \in \mathbb{R}^{T \times d}$$

- Now we can simply add them up to create the new input:

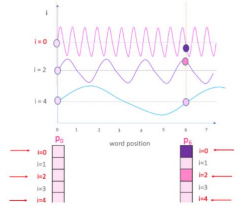
$$\mathbf{X} = \mathbf{X} + \mathbf{P}$$

Input word embeddings + Positional embeddings (exact same shape as word embeddings)

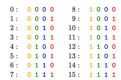
How to **get this positional embedding**?

- **Use [0, 1]**: We can encode each position in the range of 0 and 1. For example, given three words, pos1 has value of 0, pos2 has value of 0.5, and pos3 has value of 1. Anyhow, this method yield **inconsistent meaning for different length sentences**.
- Use **numberings**: We can encode each position simply by numbering 1, 2, 3 and so on. The problem with this approach is that the model cannot generalize in case that **some testing sentences are longer than training sentences**.
- Use **sine/cosine wave**: In transformer, the authors have use sine and cosine functions of different frequencies, where `pos` refers to the position, `i` refers to the value of the vector indexed at `i`. Note that the value ranged from [-1, 1].

$$\mathbf{P}_{(\text{pos},2i)} = \sin\left(\frac{\text{pos}}{10000^{\frac{2i}{d}}}\right) \quad \mathbf{P}_{(\text{pos},2i+1)} = \cos\left(\frac{\text{pos}}{10000^{\frac{2i}{d}}}\right)$$



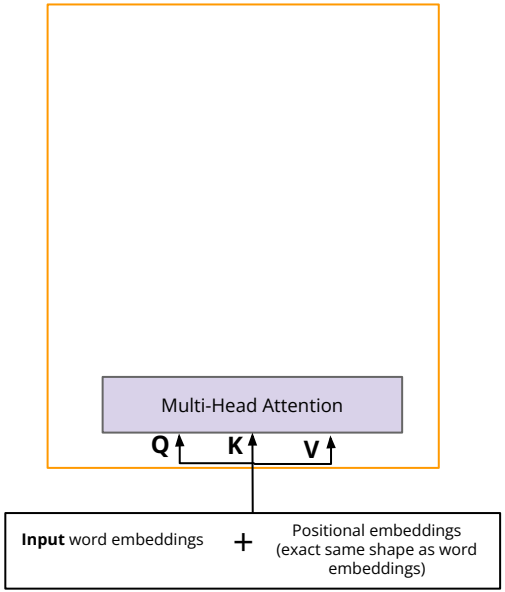
Intuition behind: You can see sine and cosine as a way to alternate bits just like in binary format like this:



Pros: Can extend to infinite length
Cons: Not learnable

- Simply let the positional embeddings be learnable parameters by simply passing through some linear layers! Most people use this!
  - Pros: Learnable
  - Cons: Can't extrapolate to indices outside 1,..., T

Transformers - Multi-Head Attention



$\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are simply three same copies of X but passing through some transformation, i.e.,

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q; \quad \mathbf{K} \in \mathbb{R}^{T\times d}; \mathbf{X} \in \mathbb{R}^{T\times d}; \mathbf{W}_q \in \mathbb{R}^{d\times d}$$
$$\mathbf{K} = \mathbf{X}\mathbf{W}_k; \quad \mathbf{K} \in \mathbb{R}^{T\times d}; \mathbf{X} \in \mathbb{R}^{T\times d}; \mathbf{W}_k \in \mathbb{R}^{d\times d}$$
$$\mathbf{V} = \mathbf{X}\mathbf{W}_v; \quad \mathbf{V} \in \mathbb{R}^{T\times d}; \mathbf{X} \in \mathbb{R}^{T\times d}; \mathbf{W}_v \in \mathbb{R}^{d\times d}$$
$$T \text{ is the number of words and } d \text{ is the embedding size}$$

Anyhow, authors of the Transformers paper claimed that it is beneficial to create different versions of Q, K, V so that the model can capture various forms of attention useful for decoding. Thus, let's say we want 3 versions (a.k.a. heads = h) (in the paper, they use 8 heads), we do like this:

$$\mathbf{Q}_1 = \mathbf{X}\mathbf{W}_{q1}; \quad \mathbf{Q}_2 = \mathbf{X}\mathbf{W}_{q2}; \quad \mathbf{Q}_3 = \mathbf{X}\mathbf{W}_{q3} \in \mathbb{R}^{T\times d_k}$$
$$\mathbf{K}_1 = \mathbf{X}\mathbf{W}_{k1}; \quad \mathbf{K}_2 = \mathbf{X}\mathbf{W}_{k2}; \quad \mathbf{K}_3 = \mathbf{X}\mathbf{W}_{k3} \in \mathbb{R}^{T\times d_k}$$
$$\mathbf{V}_1 = \mathbf{X}\mathbf{W}_{v1}; \quad \mathbf{V}_2 = \mathbf{X}\mathbf{W}_{v2}; \quad \mathbf{V}_3 = \mathbf{X}\mathbf{W}_{v3} \in \mathbb{R}^{T\times d_k}$$
$$\mathbf{W}_{qi} \in \mathbb{R}^{d\times d_k}; \mathbf{W}_{ki} \in \mathbb{R}^{d\times d_k}; \mathbf{W}_{vi} \in \mathbb{R}^{d\times d_k}; d_k = d/\text{heads}$$

Once we got h number of Q, K, V, we calculate the attention using this simple formula:

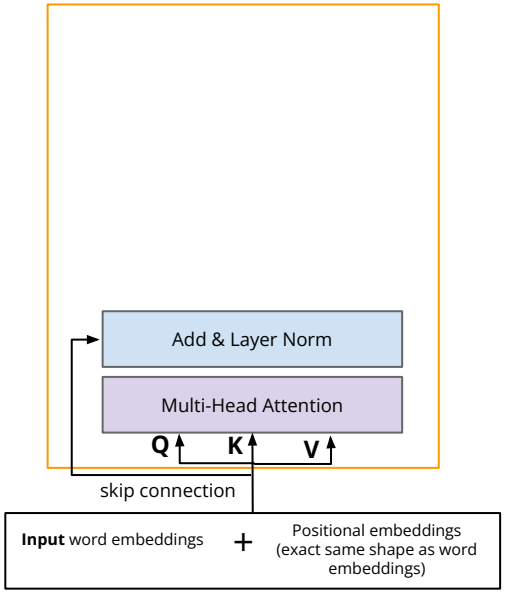
$$\text{Att}_i = \frac{\text{softmax}(\mathbf{Q}_i\mathbf{K}_i^\top)}{\sqrt{d_k}}\mathbf{V}_i$$

We then concatenate all these attentions, and multiply just another W matrix to learn any linear transformations required.

$$\text{MultiHead} = \text{concat}(\text{Att}_1, \text{Att}_2, \cdots, \text{Att}_h)\mathbf{W}_o; \quad \mathbf{W}_o \in \mathbb{R}^{hd_k\times d}$$

Transformers - Skip connection and layer norm

- After we perform the multi-head attention, we add the output with the original input (i.e., X) through **skip connection**
  - Skip connection helps with training especially in a very deep network
- We then perform **layer normalization**
  - Helps model train faster by cutting down on uninformative variation **within each layer**



- Let $\mathbf{x} \in \mathbb{R}^d$ be an individual (word) vector in the model
- Let μ be the mean

$$\mu = \frac{1}{d}\sum_{j=1}^{d} x_j; \quad \mu \in \mathbb{R}$$

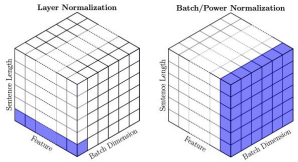
- Let σ be the std

$$\sigma = \sqrt{\frac{1}{d}\sum_{j=1}^{d}(x_j - \mu)^2}; \quad \sigma \in \mathbb{R}$$

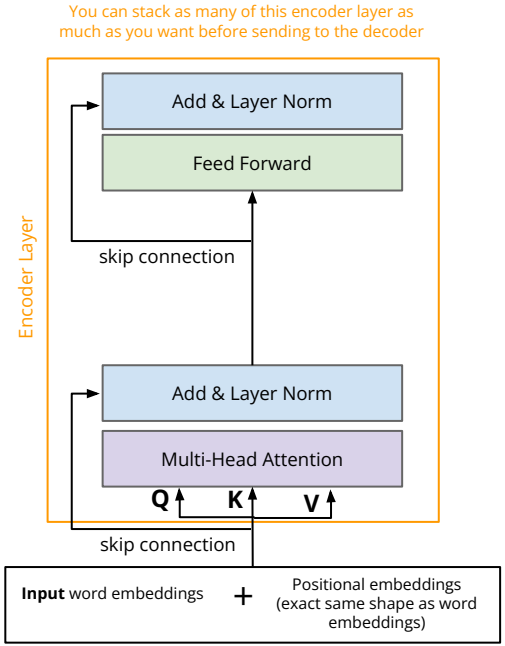
- Let $\gamma \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$ be learned "gain" and "bias" parameters. (optional)
- Then layer normalization computes

$$\text{output} = \frac{\mathbf{x} - \mu}{\sqrt{\sigma + \epsilon}} * \gamma + \beta$$

- Difference between **batch norm vs. layer norm:** batchnorm normalizes **across whole batch in each dimension,** while layernorm normalizes **each word independently of other words**. In NLP, layernorm tend to be used instead of batchnorm that is mostly used in CV task.
  - In NLP tasks, the sentence length often varies -- thus, if using batchnorm, it would be uncertain what would be the appropriate normalization constant (the total number of elements to divide by during normalization) to use. According to the paper that provided the image on our left, "statistics of NLP data across the batch dimension exhibit large fluctuations throughout training. This results in instability, if BN is naively implemented." [Source: stackexchange]



Transformers - Encoders

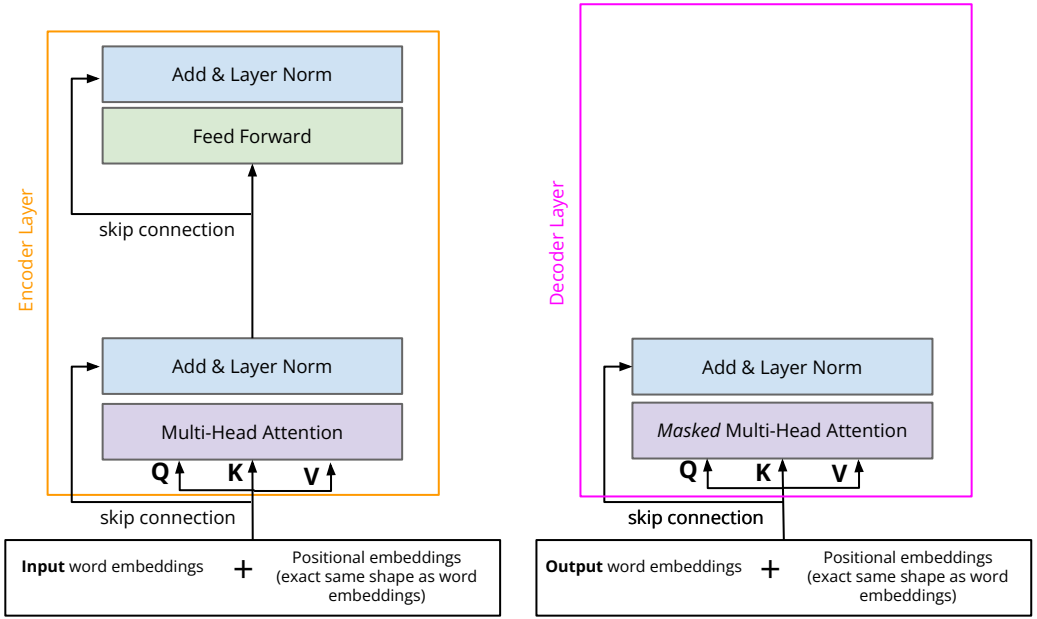


- To add some **non-linearities**, we add a feedforward layer
  - In the paper, they used two linear layers with a ReLU in between

$$\mathrm{FFN}(x) = \max(0, \mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

- Then we again perform **skip connections, layer norms**
- We can **repeatedly stack this whole encoder layer** as many times we want!
  - The paper used **6** of these encoder layers (and also **6** decoder layers which we will cover soon)
- We are done on the encoder side!

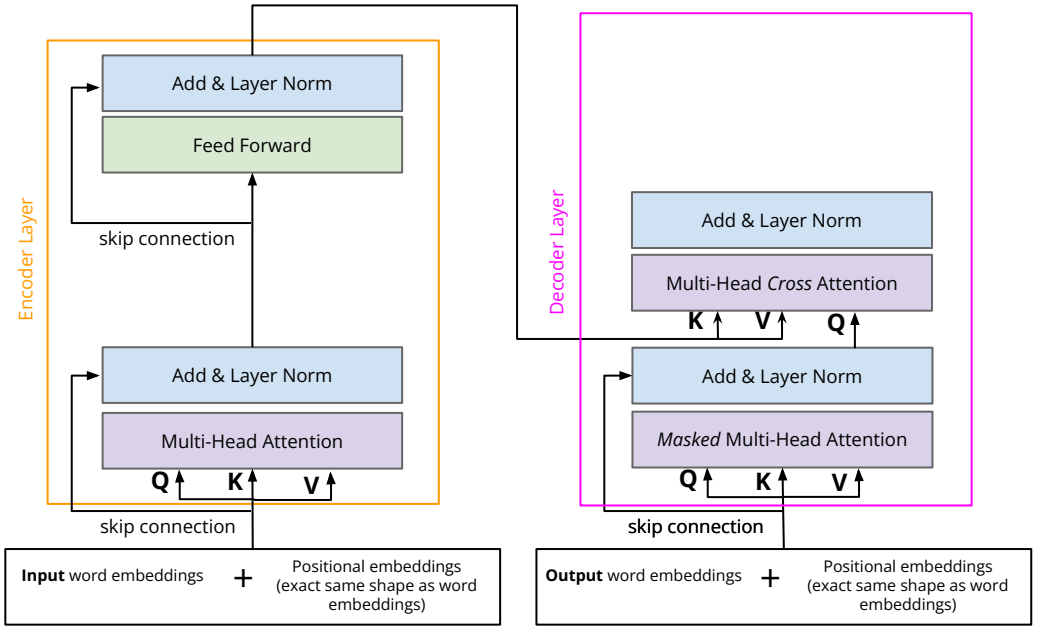
Transformers - Masked Multi-Head Attention



- To use self-attention in **decoders**, we need to ensure **we cannot peek at the future**
- To enable **parallelization**, we **mask the attention to future words by setting attention scores to -inf**

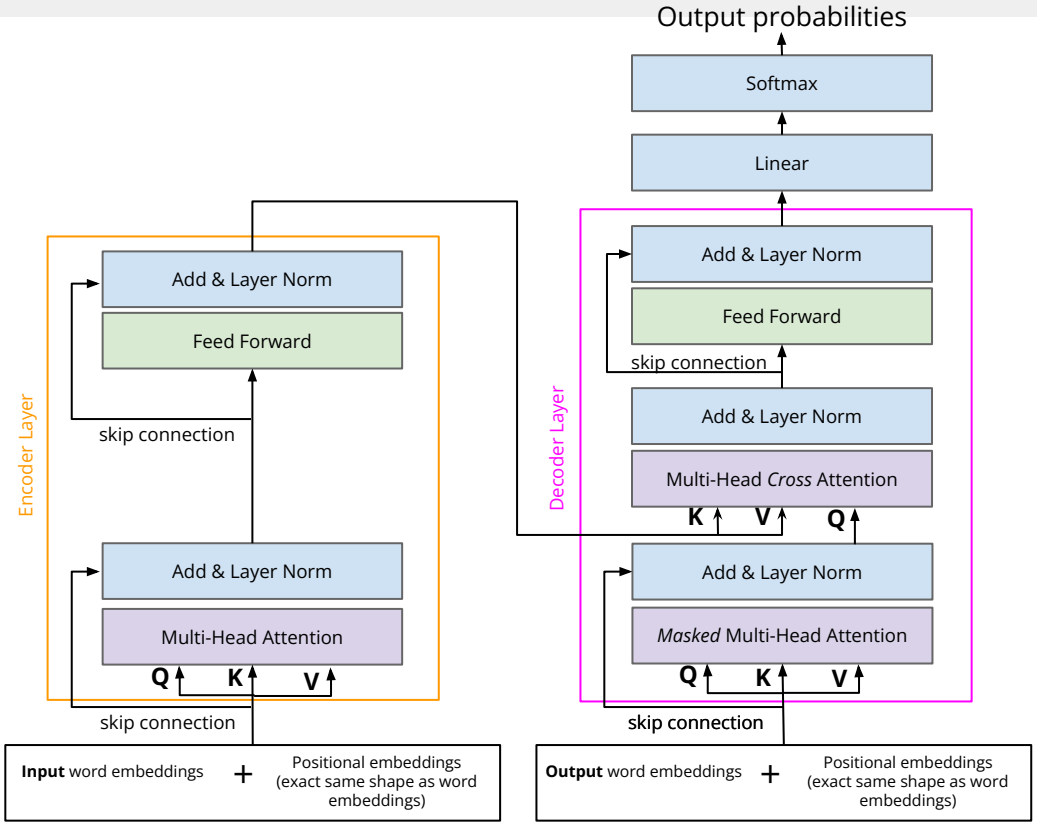
	<START>	deep	learning	is	fun	<END>
<START>	-∞	-∞	-∞	-∞	-∞	-∞
deep		-∞	-∞	-∞	-∞	-∞
learning			-∞	-∞	-∞	-∞
is				-∞	-∞	-∞
fun					-∞	-∞
<END>						-∞

Transformers - Cross attention



- **Cross attention** is just slightly different from normal self-attention, i.e., the queries are from the decoder, while the key and values are from the encoder side
  - Don't be confused here - it's exactly the same as when we first learn attention in L7, where the decoder states are dotted with the encoder states

Transformers - Decoders



- Similar to encoder, you can stack as much decoder layers you want
- That's it!

Some more details:

- On both encoder and decoder layers, before inputting to the next layer, they performed a **dropout** of 0.1
- Also, they perform a **dropout** of 0.1 to the sums of the embeddings and the positional embeddings in both the encoder and decoder stacks
- **Adam** optimizer is used (detailed learning rates and momentum in the paper)
- The **embedding size** is 512
- The **dimension of the feedforward layer** is 2048
- Number of **epochs trained**: 100K

Transformers Results

Transformers with MT

Table 2: 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]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Training took 3.5 days on 8 P100 GPUs (used beam search where k = 4). Training cost is a function of training time, number of GPUs used and capacity of each GPU used.

Transformers with text summarization

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

DMCA is some form of **m**emory **c**ompressed **a**ttention, where attention is reduced with strided convolution to reduce operations. **MoE** (mixtures of experts) is proposed by another paper [Shazeer et al. 2017.] which is a layer consists of number of experts, each a simple FF network, and a trainable gating network which selects a combination of the experts (focuses on reducing computational cost). L stands for local attention layer.

Generating wikipedia by summarizing long sequences, Liu et al., 2018 (WikiSum dataset), https://arxiv.org/pdf/1801.10198.pdf

Transformers with SuperGLUE

	Rank	Name	Model	Score
	1	Liam Fedus	SS-MoE	91.0
	2	Microsoft Alexander v-team	Turing NLR v5	90.9
	3	ERNIE Team - Baidu	ERNIE 3.0	90.6
+	4	Zirui Wang	T5 + UDG, Single Model (Google Brain)	90.4
+	5	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4	90.3
	6	SuperGLUE Human Baselines	SuperGLUE Human Baselines	89.8
+	7	T5 Team - Google	T5	89.3
+	8	Huawei Noah's Ark Lab	NEZHA-Plus	86.7
+	9	Alibaba PAI&ICBU	PAI Albert	86.1
+	10	Infosys : DAWN : AI Research	RoBERTa-iCETS	86.0

All of these are based on transformers!

Improvements focus on increasing the number of parameters, reducing training cost, using different positional embeddings, different training scheme (e.g., multi-tasks), different regularization technique. Certainly take a look at these top performers!

The **General Language Understanding Evaluation** (GLUE) benchmark is a collection of nine sentence - or sentence-pair language understanding tasks. A public leaderboard for tracking performance. The tasks are selected to favor models that share information across tasks using parameter sharing or other transfer learning techniques. The ultimate goal is to develop a generalized and robust NL systems.

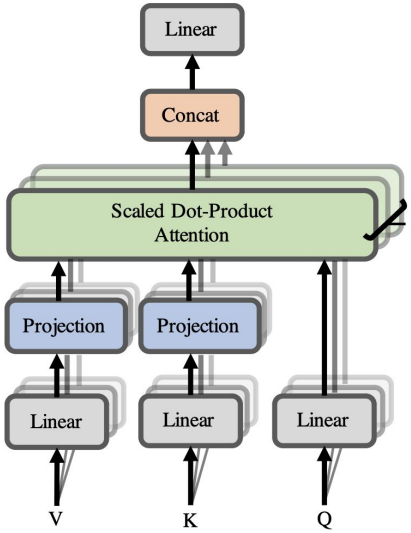
https://super.gluebenchmark.com/leaderboard

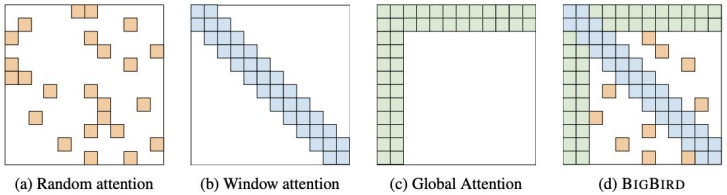
Transformers Limitations

Transformers limitations - costs

- **Quadratic costs** in self-attention
  - Computing all pairs of interactions means our computation grows quadratically with the sequence length! (Don't confuse this with the benefits of parallelizability)
    - In practice, sequence length is set to 512
    - What if our sequence length is more than 10,000? E.g., summarization?
  - Compared to recurrent models, it only grew linearly!

- **Fixes**:
  - Project values and keys to lower dimension (Linformer [Wang et al., 2020])
  - Replace all-pairs interactions with less aggressive interactions like local windows, random interactions, etc.



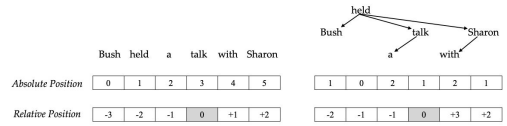


Linformer: Self-Attention with Linear Complexity, Wang et al., 2020, https://arxiv.org/pdf/2006.04768.pdf

Big Bird: Transformers for Longer Sequences, Zaheer et al. 2021 https://arxiv.org/pdf/2007.14062.pdf

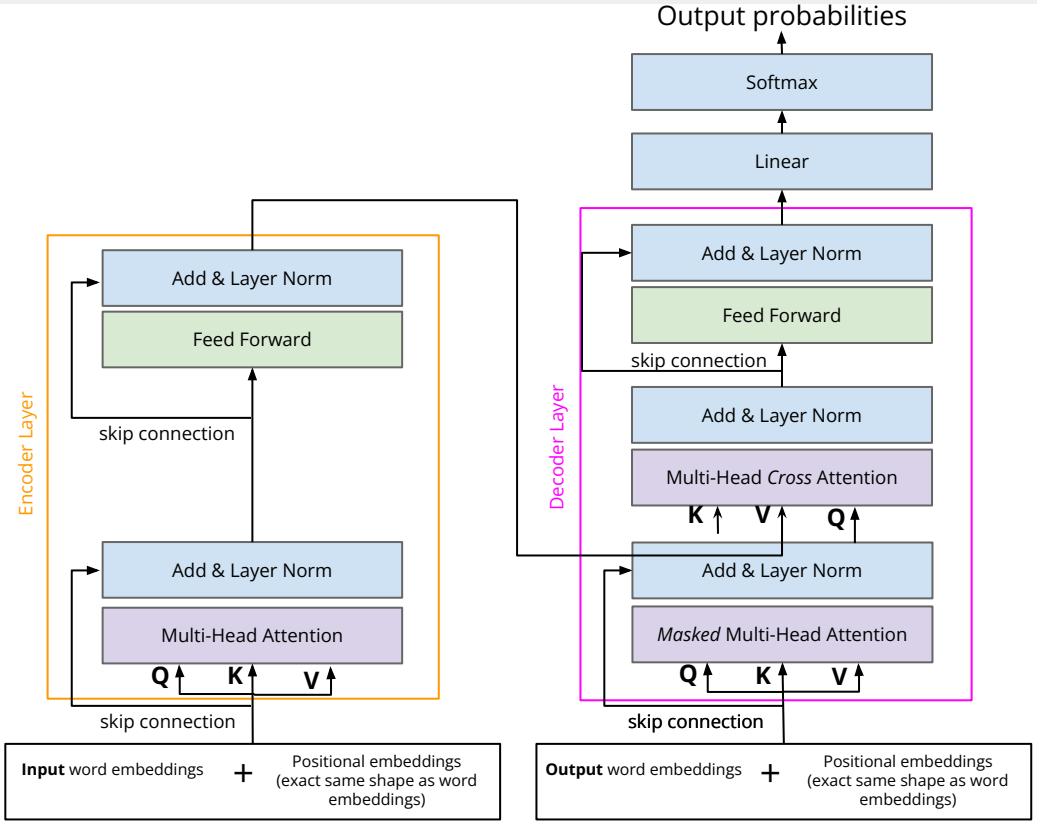
Transformers limitations - positions

- Ways to represent position? Some developments:
  - **Relative linear position attention** [Shaw et al., 2018]
    - Each word has NOT only one positional embeddings, but equal to the sequence length, in order to describe the relationship between words (very similar to graph)
      - How to learn these relationships -> self attention again!
    - Improves 1.3 BLEU and 0.3 BLEU on WMT 2014 English to German and English to French translation tasks over absolute position representations
  - **Dependency syntax-based position** [Wang et al., 2019]
    - Use depth to represent absolute structural position (use the original paper approach)
    - Use distance of each word pair in the tree to represent relative position (use Shaw et al. 2018 method)
    - To combine, pass through some nonlinear functions
    - Improves 0.61 BLEU on average



Self-Attention with Relative Position Representations, Shaw et al., 2018, https://arxiv.org/abs/1803.02155
Self-Attention with Structural Position Representations, Wang, et al., 2019, https://arxiv.org/pdf/1909.00383.pdf

Summary :-)


Output probabilities
Softmax
Linear
Add & Layer Norm
Feed Forward
skip connection
Add & Layer Norm
Multi-Head Cross Attention
K
V
Q
Add & Layer Norm
Masked Multi-Head Attention
Q
K
V
skip connection
Decoder Layer
Add & Layer Norm
Feed Forward
skip connection
Add & Layer Norm
Multi-Head Attention
Q
K
V
skip connection
Encoder Layer
Input word embeddings
+
Positional embeddings (exact same shape as word embeddings)
Output word embeddings
+
Positional embeddings (exact same shape as word embeddings)