

# Lecture 5. Transformers

(“Attention Is All You Need” Vaswani et al., 2017)

Spring 2023

# Outline

## NLP

- Contextualized Word Embeddings

- Subword Tokenization

- Byte-Pair Encoding

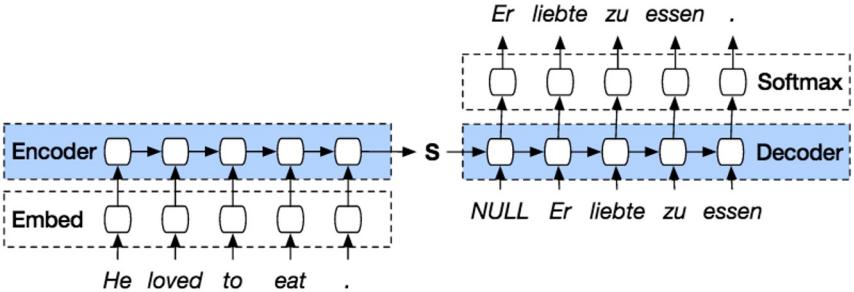
- GPT, BERT, RoBERTa

## ML

- Transformer

- Self Attention

# Recall: RNN Encoder-Decoder; Attention; Self Attention



# Recall: RNN Encoder-Decoder; Attention; Self Attention

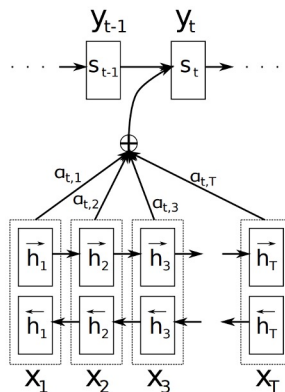
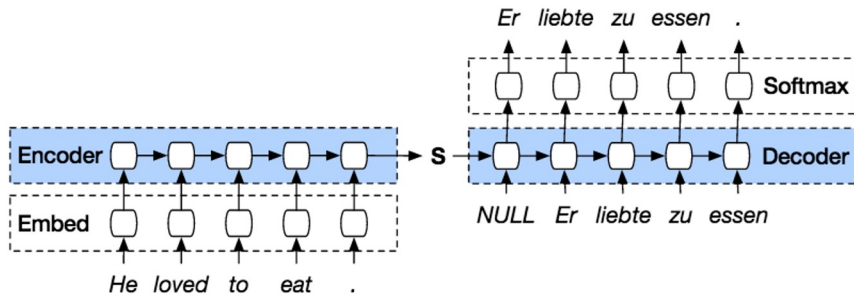
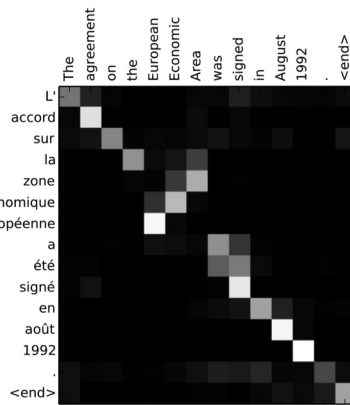


Figure 1: The graphical illustration of the proposed model trying to generate the  $t$ -th target word  $y_t$  given a source sentence  $(x_1, x_2, \dots, x_T)$ .





# Attention Is All You Need (Vaswani et al., 2017)

---

## Attention Is All You Need

---

## Transformer

**Ashish Vaswani\***  
Google Brain  
avaswani@google.com

**Noam Shazeer\***  
Google Brain  
noam@google.com

**Niki Parmar\***  
Google Research  
nikip@google.com

**Jakob Uszkoreit\***  
Google Research  
usz@google.com

**Llion Jones\***  
Google Research  
llion@google.com

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

**Lukasz Kaiser\***  
Google Brain  
lukaszkaizer@google.com

**Illia Polosukhin\* ‡**  
illia.polosukhin@gmail.com

### Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

# Why Transformers

Incredibly powerful  
when trained on  
massive amounts of  
natural language text.



The image is a screenshot of a web page from The New York Times. At the top left is a hamburger menu icon. The page title is "The New York Times" in a serif font. To the right of the title is an "Account" link with a dropdown arrow. The main headline is in a bold, italicized serif font: "Meet GPT-3. It Has Learned to Code (and Blog and Argue)." Below the headline is a paragraph of text: "The latest natural-language system generates tweets, pens poetry, summarizes emails, answers trivia questions, translates languages and even writes its own computer programs." On the right side of the page is a vertical illustration of a diverse group of stylized, cartoonish human faces with various features like glasses, beards, and different skin tones.

☰

The New York Times

Account ▾

***Meet GPT-3. It Has Learned to Code (and Blog and Argue).***

The latest natural-language system generates tweets, pens poetry, summarizes emails, answers trivia questions, translates languages and even writes its own computer programs.

# GPT-3

- Built by OpenAI (funded by Microsoft, Google, FB)
- Trained a Transformer to do standard language modeling, i.e. **next word prediction**
- GPT-3 is a tremendously powerful text generator

It was openly released immediately...fearing misuse (e.g. fake news construction).

- GPT-2 was released in November of 2019. (Available via [huggingface.com](https://huggingface.com))
- GPT-3 (175billion) available via an API offered by Microsoft (not free). Now **Codex** is free.



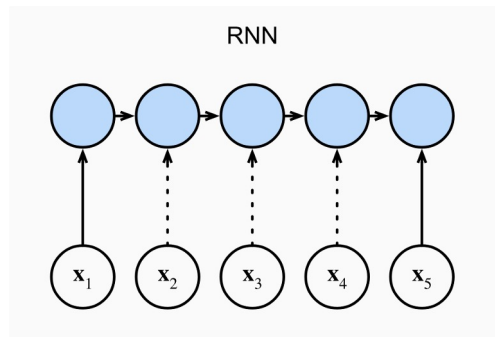
# Transformers



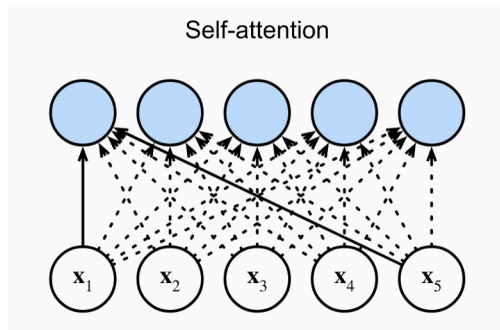
- Introduced in **Attention Is All You Need** (Vaswani et al. *NeurIPS* 2017)
- A purely attention-based architecture (highly parallelizable), i.e. **no recurrence**
- Deep model for NLP (12 layers)
- Originally envisioned for seq2seq tasks (encoder is 6 layers, decoder is 6 layers). The encoder and decoder are the same “architecture” applied differently

# Transformer - Key Idea

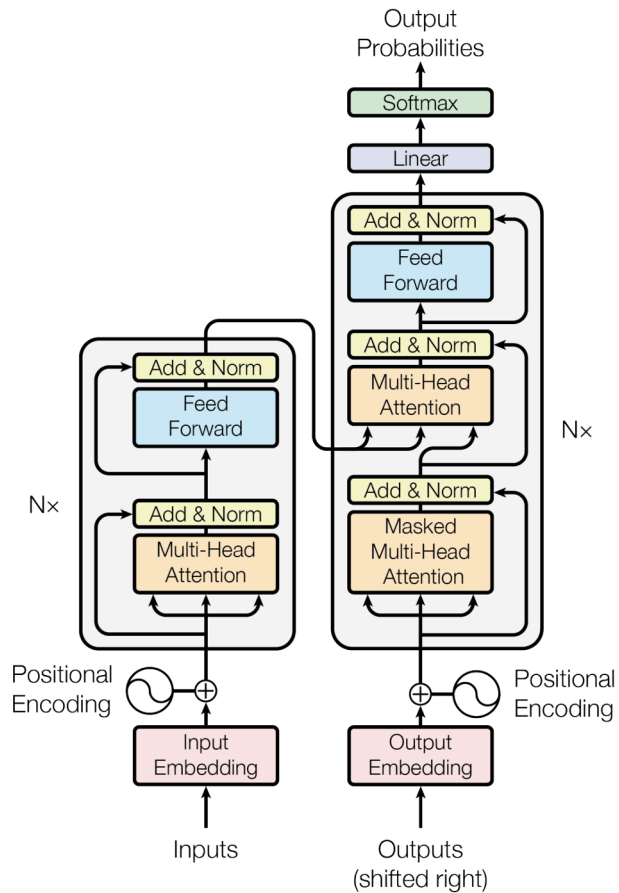
In RNN, the first token is linked to next token, and so on, using RNN unit.



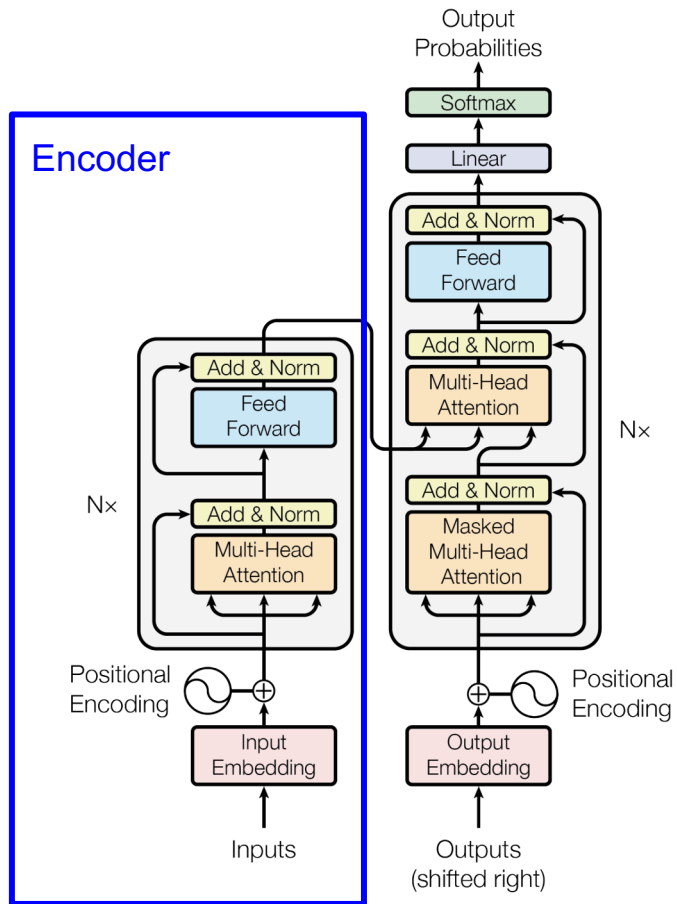
In Transformer, every token is linked to *all other tokens* using *self attention*.



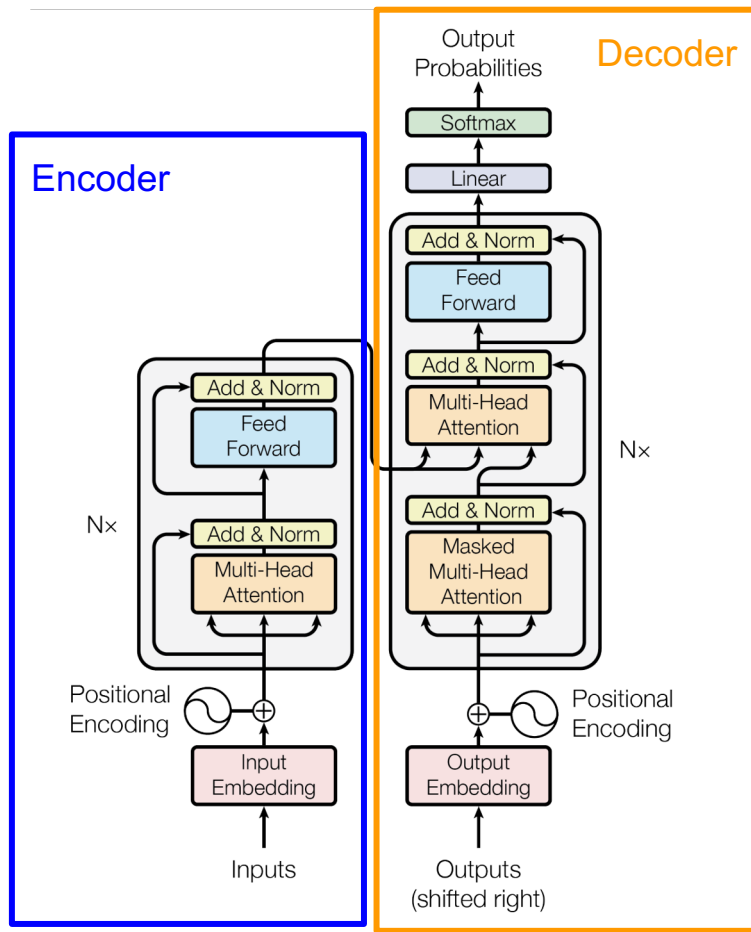
# Transformer - Architecture Overview



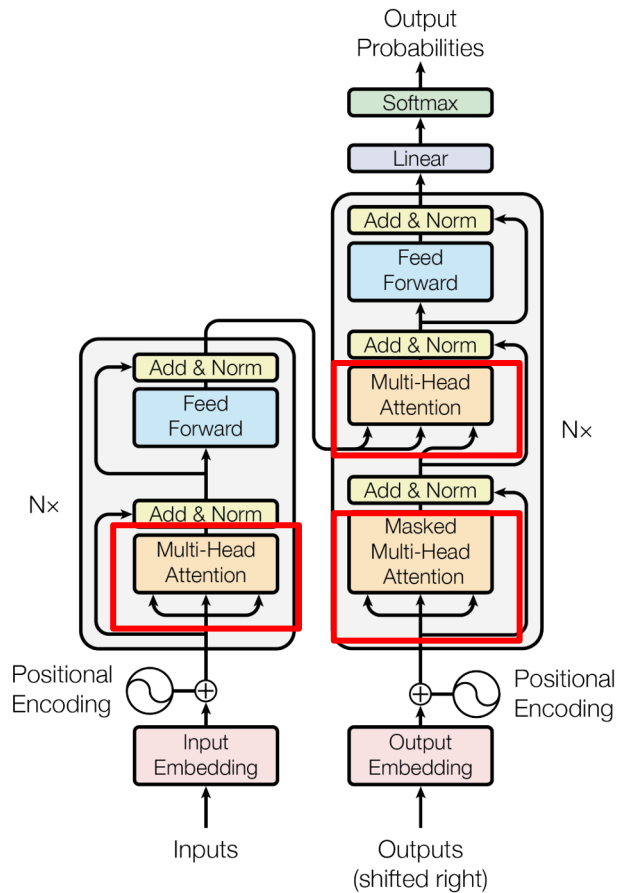
# Transformer - Architecture Overview



# Transformer - Architecture Overview



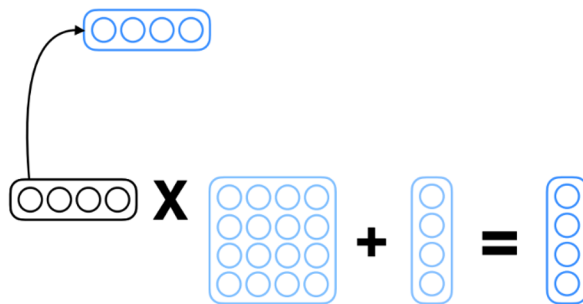
# Transformer - Self Attention



# Transformer - Self Attention



# Transformer - Self Attention



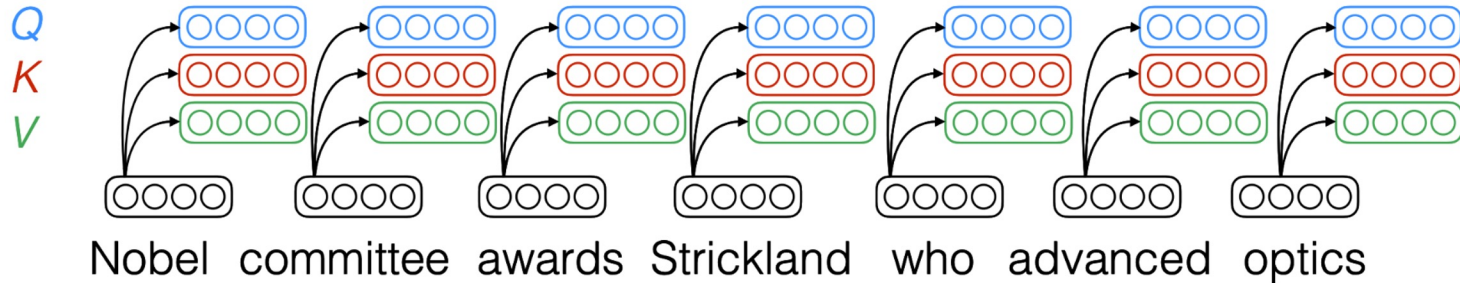
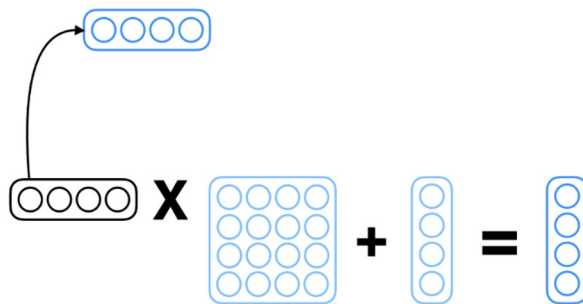
---

⊙⊙⊙⊙   ⊙⊙⊙⊙   ⊙⊙⊙⊙   ⊙⊙⊙⊙   ⊙⊙⊙⊙   ⊙⊙⊙⊙   ⊙⊙⊙⊙  
Nobel committee awards Strickland who advanced optics



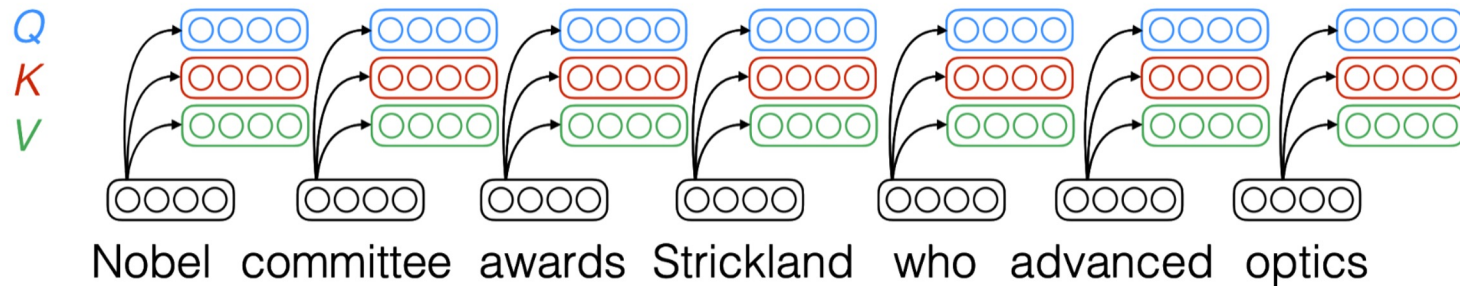
# Transformer - Self Attention

Q: Query  
K: Key  
V: Value



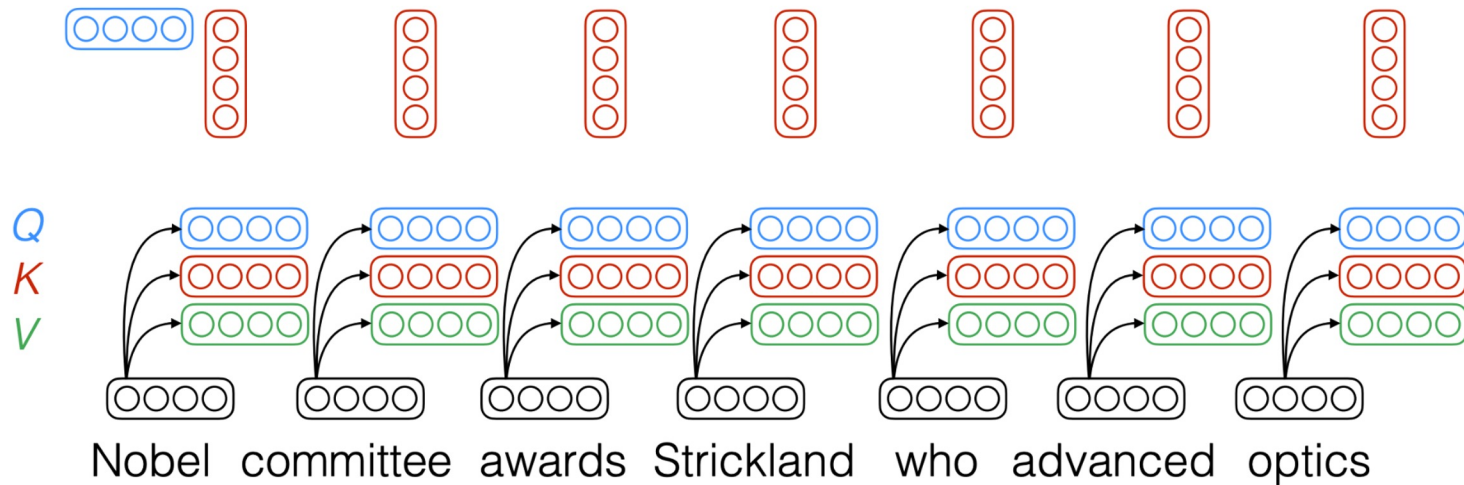
# Transformer - Self Attention

Q: Query  
K: Key  
V: Value



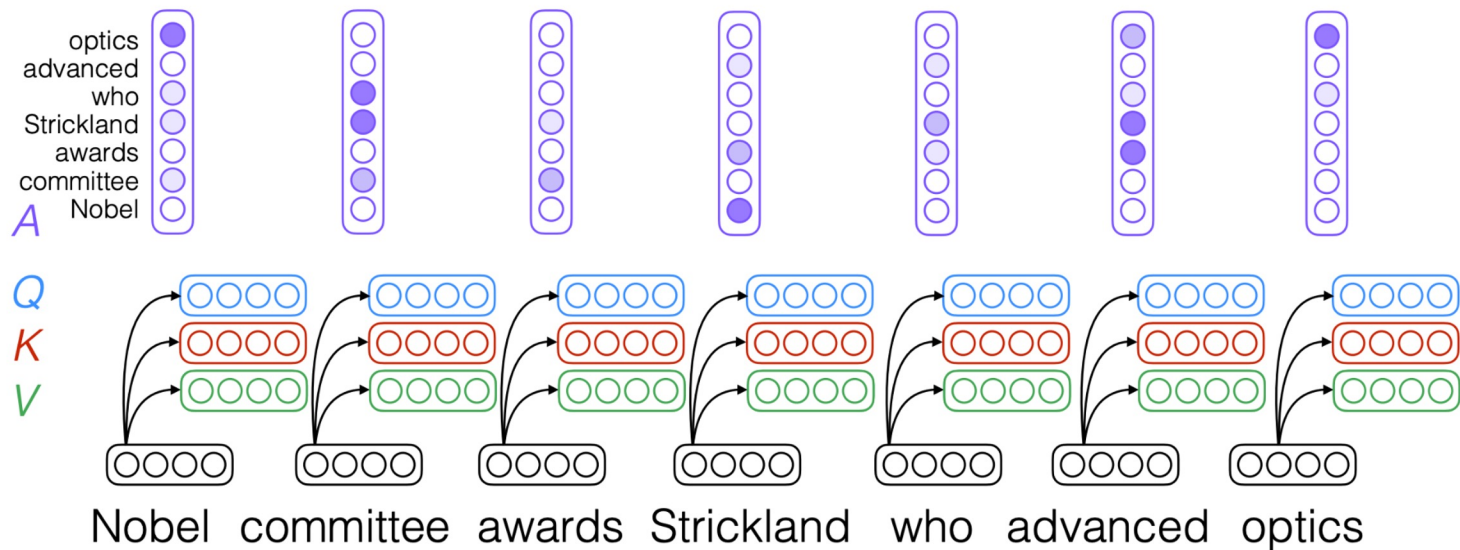
# Transformer - Self Attention

Calculate Attention scores: between Q: Query and K: Key

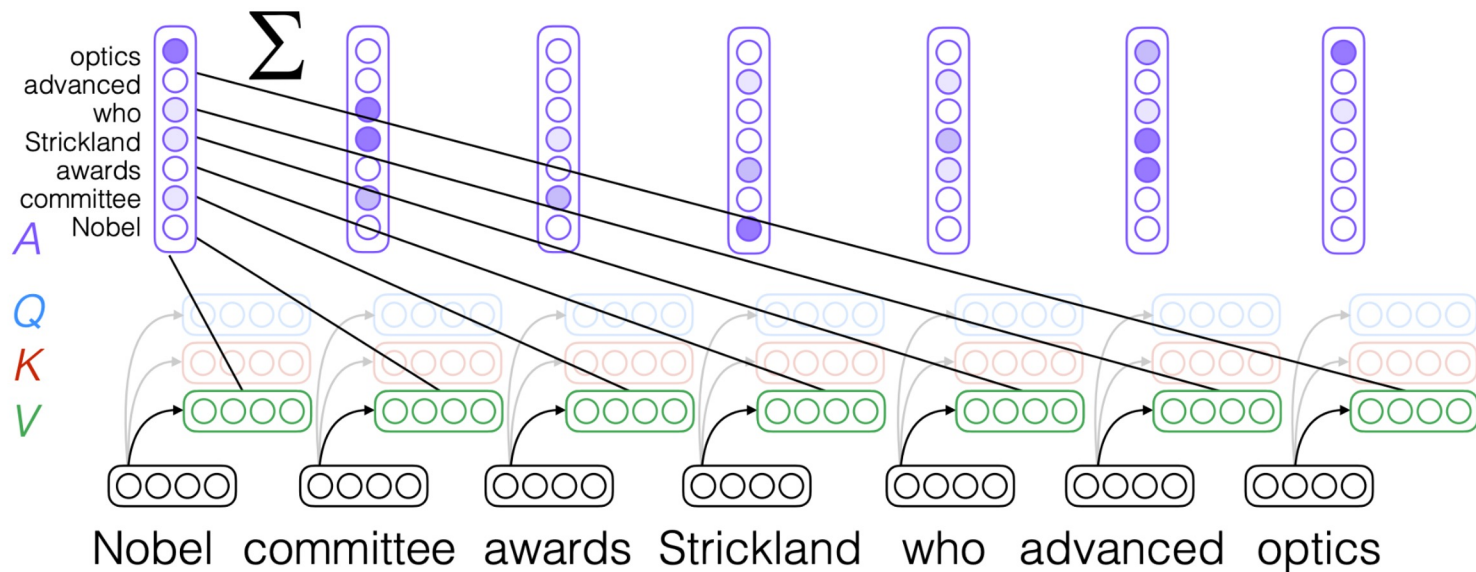


# Transformer - Self Attention

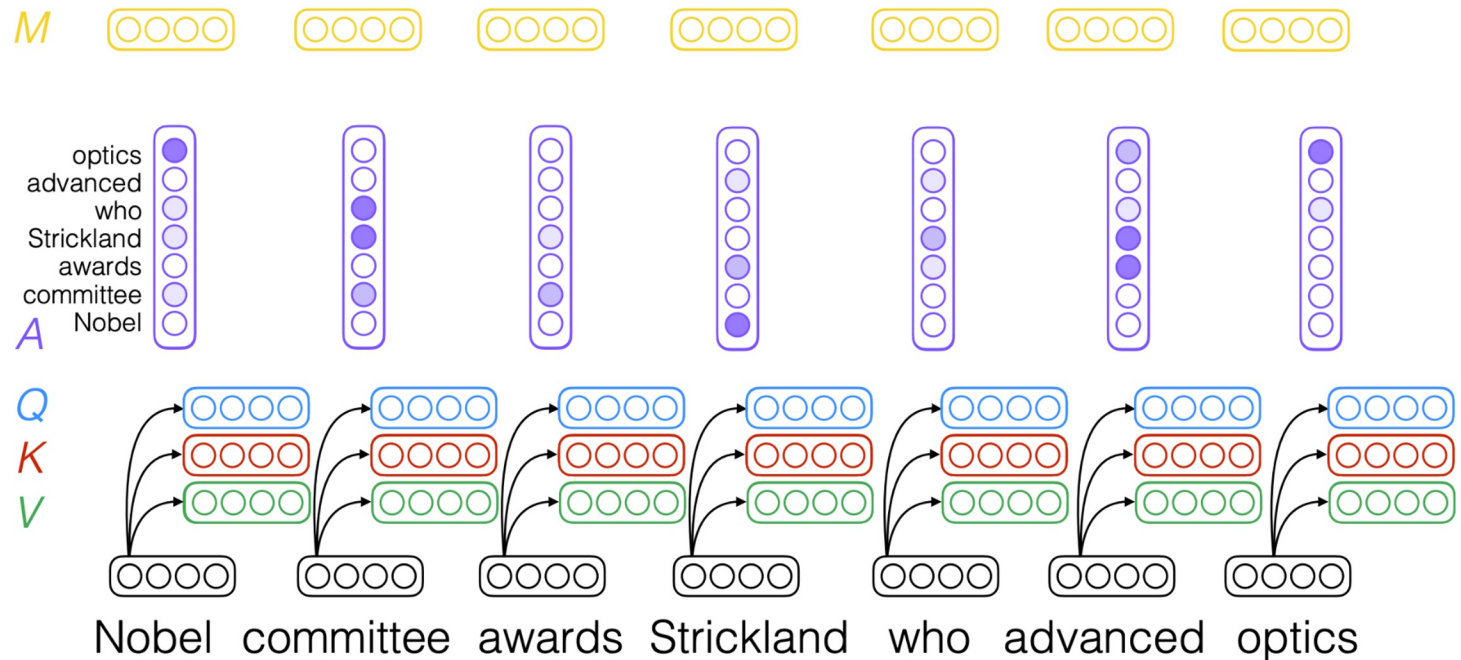
Each word is attended to all other words.



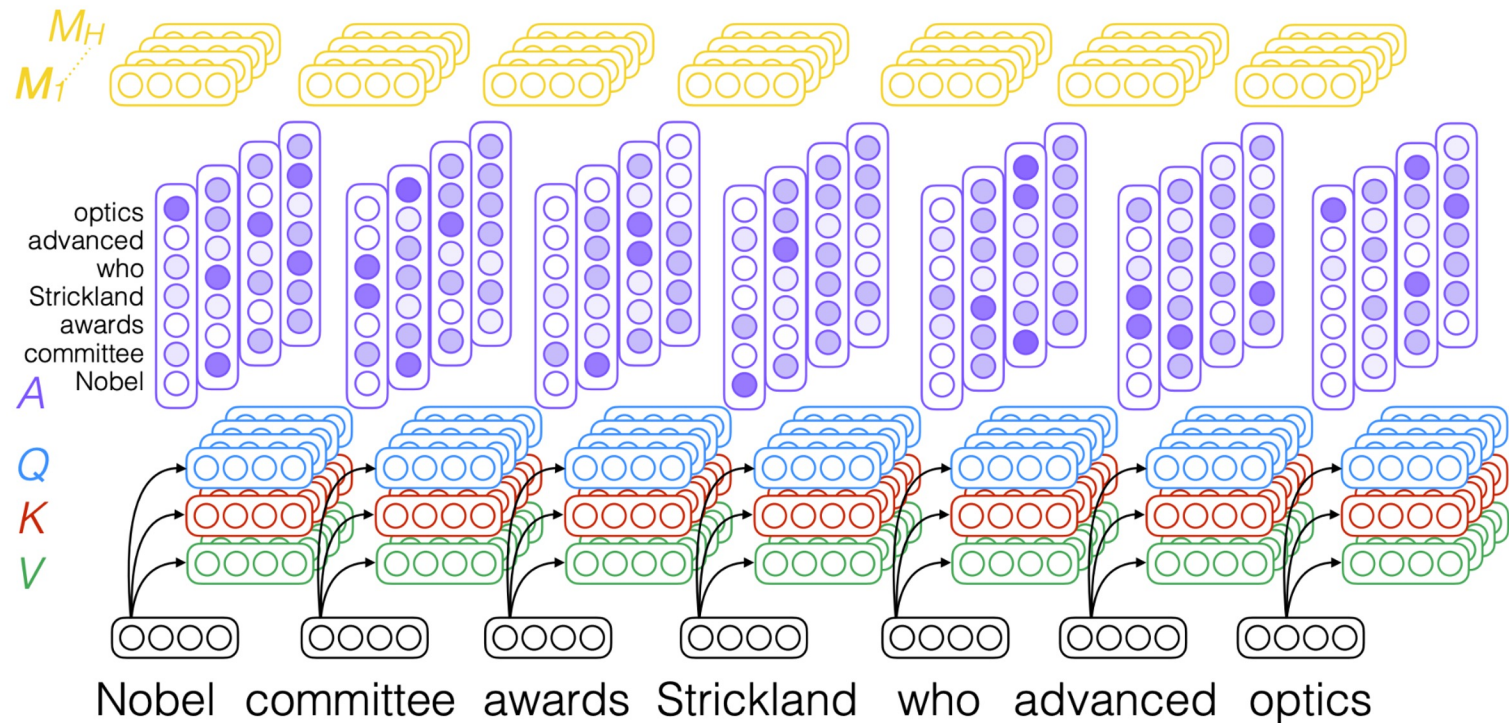
# Transformer - Self Attention



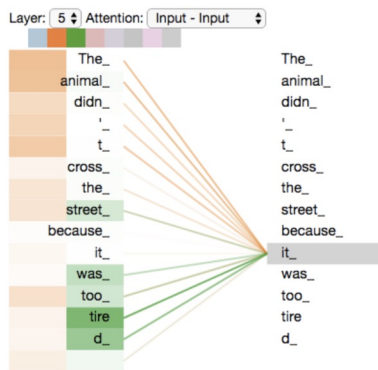
# Transformer - Self Attention



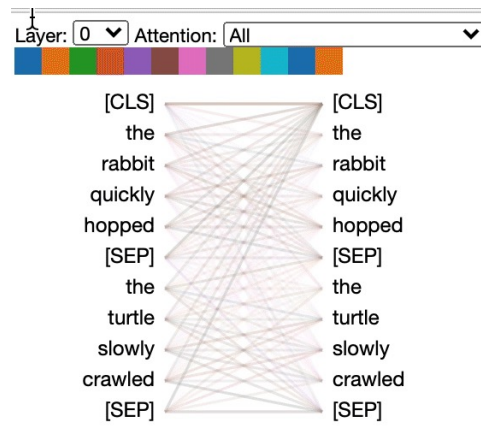
# Transformer - Multi-Head Self Attention



# Transformer - Multi-Head Self Attention



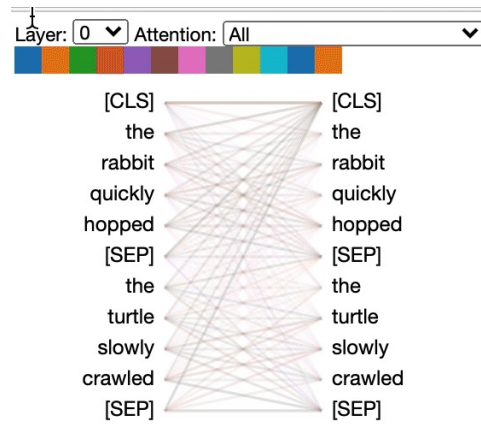
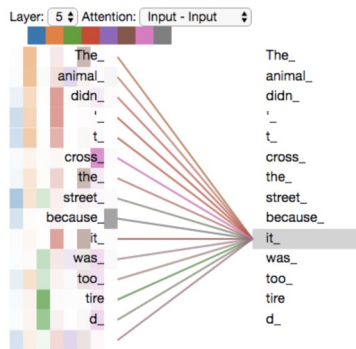
As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".



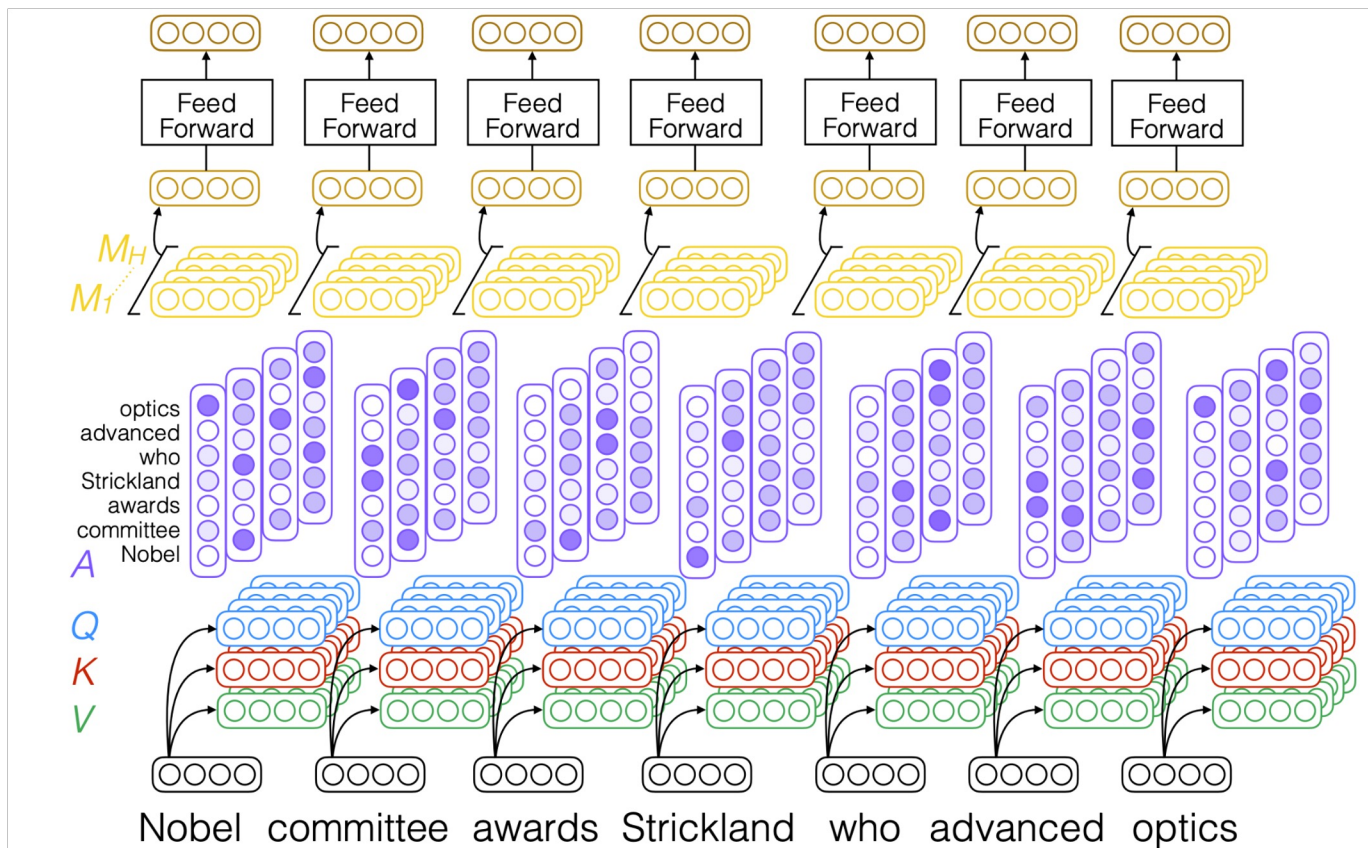


# Transformer - Multi-Head Self Attention

If we add all the attention heads to the picture, however, things can be harder to interpret:

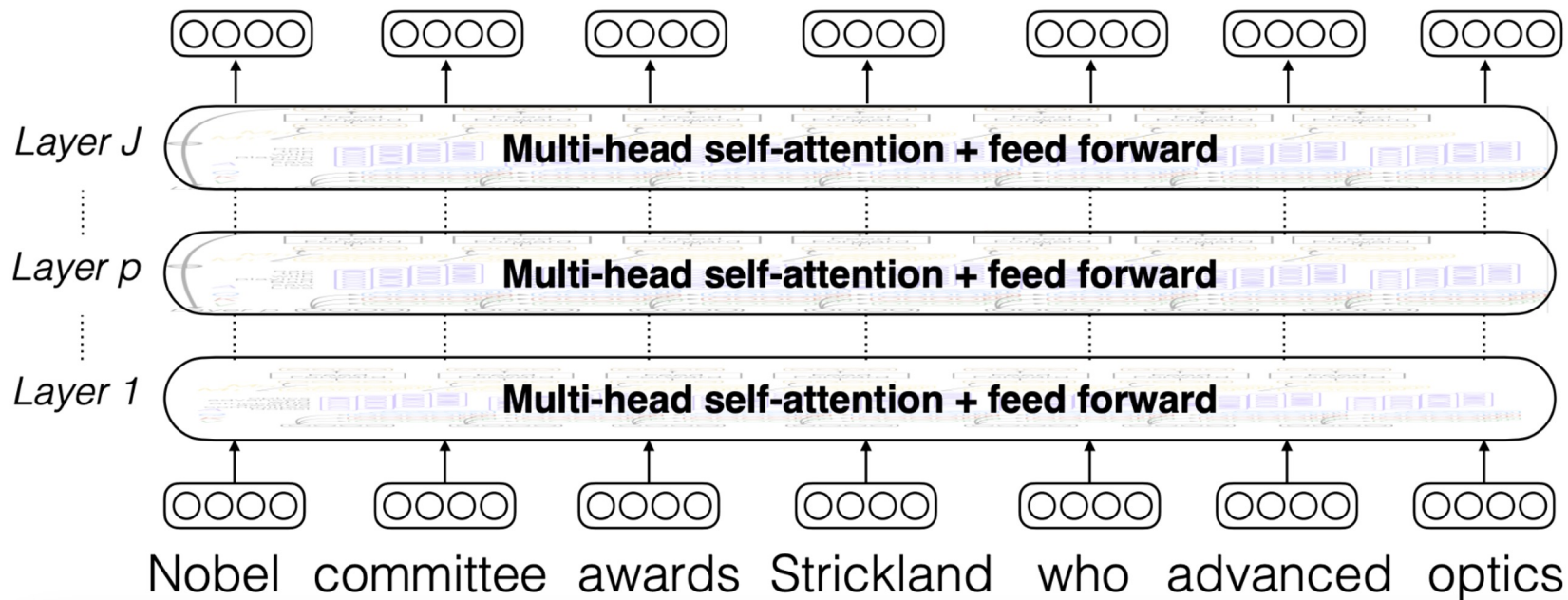


# Transformer Layer



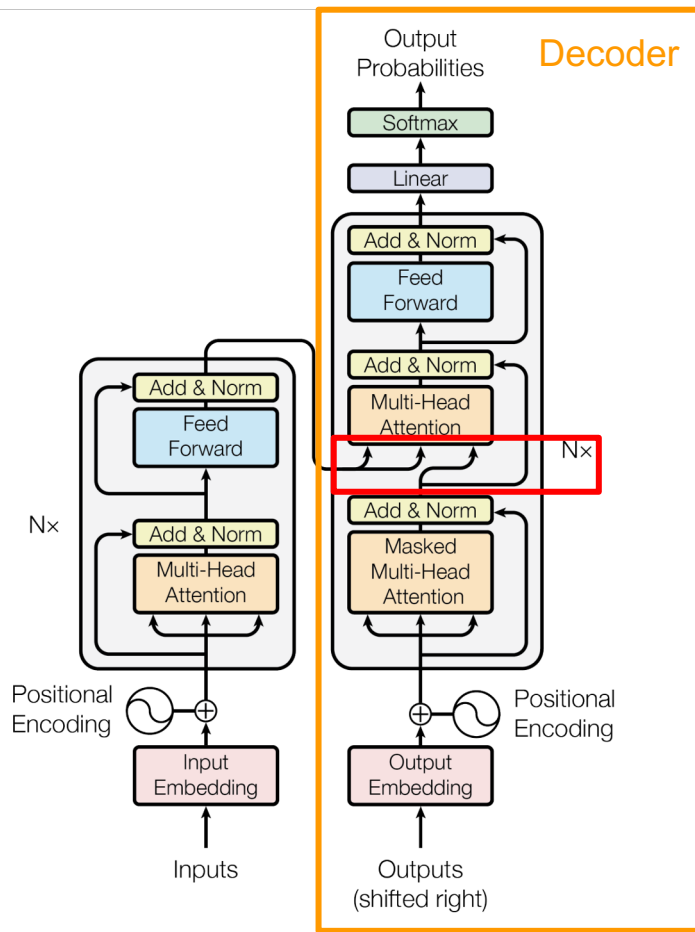
Figures from Emma Strubell

# Transformer

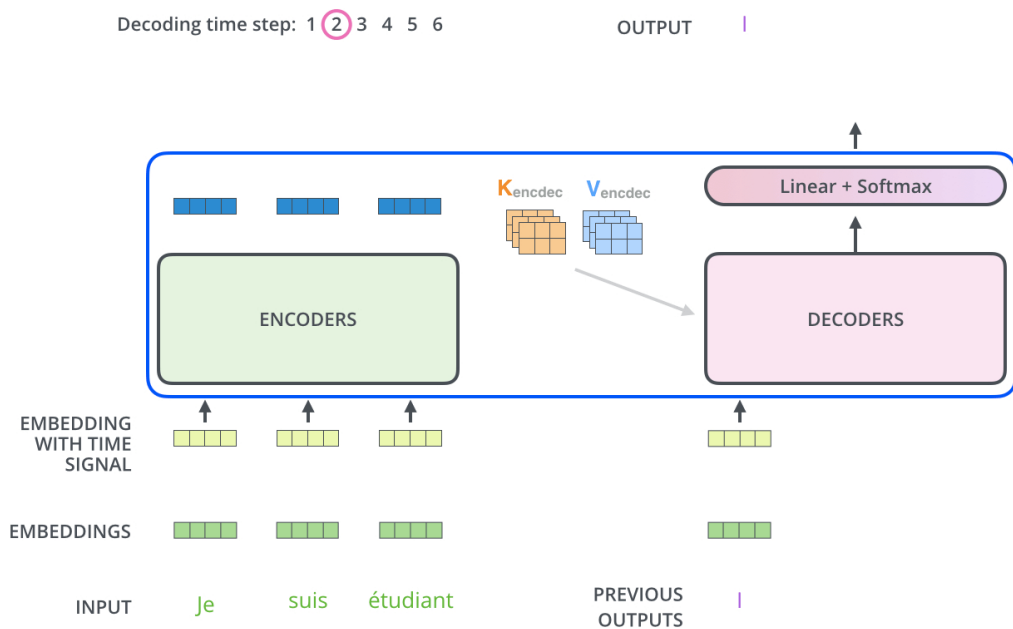


# Transformer – decoder side

# Transformer - Architecture Overview

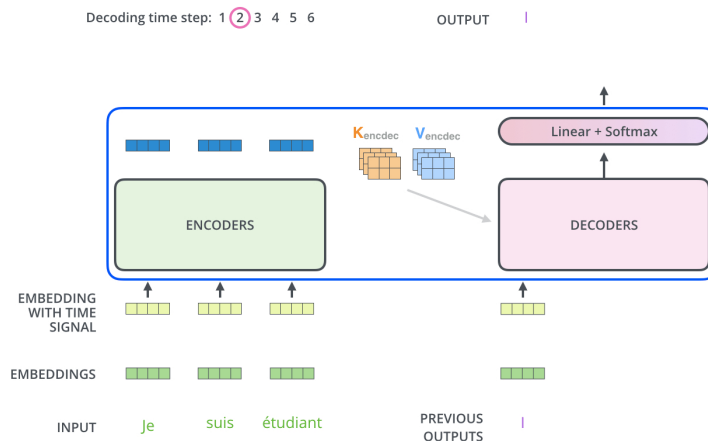


# Transformer – decoder side



# Transformer – decoder side

- The self attention layers in the decoder operate in a slightly different way than the one in the encoder:
  - In the decoder, the self-attention layer is only allowed to **attend to earlier positions in the output sequence**.
  - The “Encoder-Decoder Attention” layer works just like multiheaded self-attention, except it creates its Queries matrix from the layer below it, **and takes the Keys and Values matrix from the output of the encoder stack**.



# Transformer – decoder side

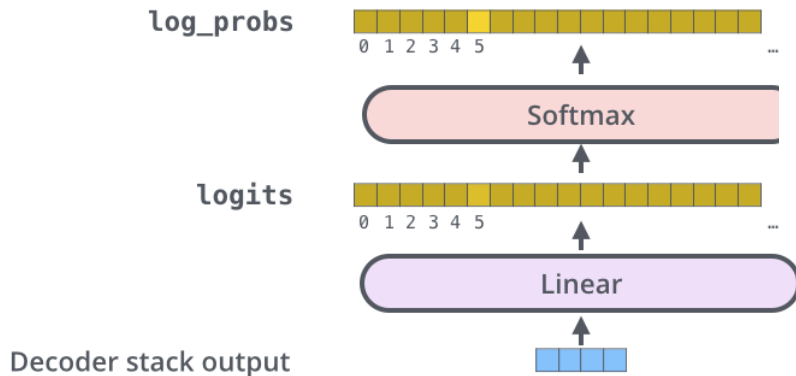
- The Final Linear and Softmax Layer

Which word in our vocabulary  
is associated with this index?

am

Get the index of the cell  
with the highest value  
(**argmax**)

5





# Transformer – decoder side

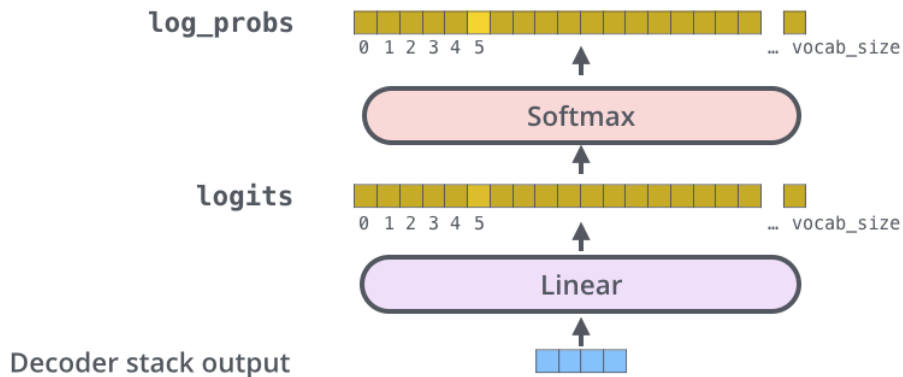
- The Final Linear and Softmax Layer


Which word in our vocabulary  
is associated with this index?

am

Get the index of the cell  
with the highest value  
(argmax)

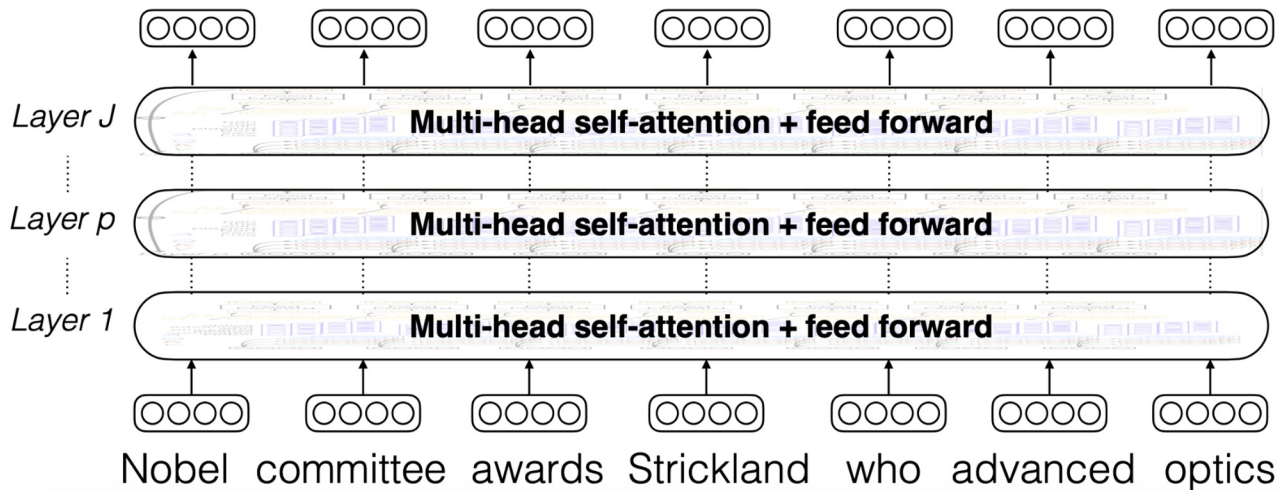
5



-  Illustrated-transformer by Jay
  - <https://jalammar.github.io/illustrated-transformer/>

# Contextualized Word Embeddings

- Problems with word2vec/GloVe: single representation for each word.
- Contextualized Word Embeddings: representations depend on context.



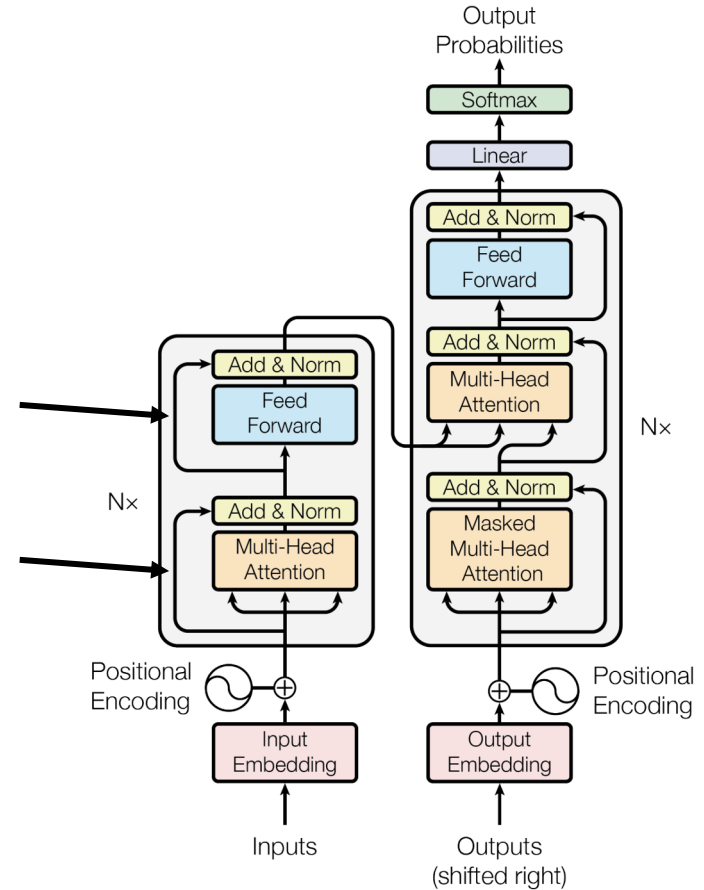
# Other Components in Transformers

- Residual Connection
- Layer Normalization
- Masked Self Attention in Decoder
- Encoder-Decoder Cross Attention
- Sinusoidal Positional Encoding

# Residual Connection

Residual Connection ([He et al., 2016](#))

- Add input to its output



# Layer Normalization

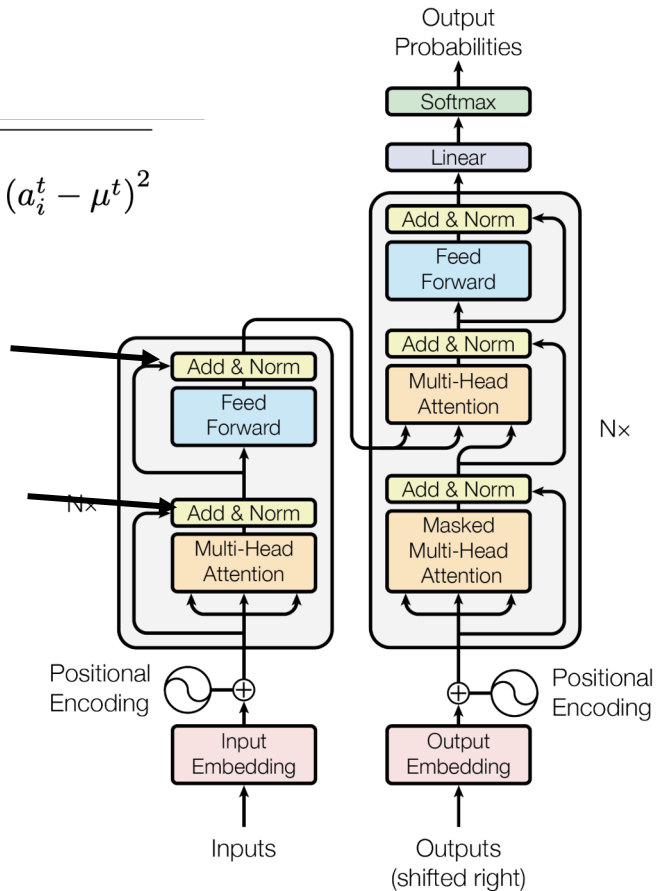
## Layer Normalization ([Ba et al., 2016](#))

$$\mathbf{h}^t = f \left[ \frac{\mathbf{g}}{\sigma^t} \odot (\mathbf{a}^t - \mu^t) + \mathbf{b} \right] \quad \mu^t = \frac{1}{H} \sum_{i=1}^H a_i^t \quad \sigma^t = \sqrt{\frac{1}{H} \sum_{i=1}^H (a_i^t - \mu^t)^2}$$

```
class LayerNorm(nn.Module):
    "Construct a layernorm module (See citation for details)."
```

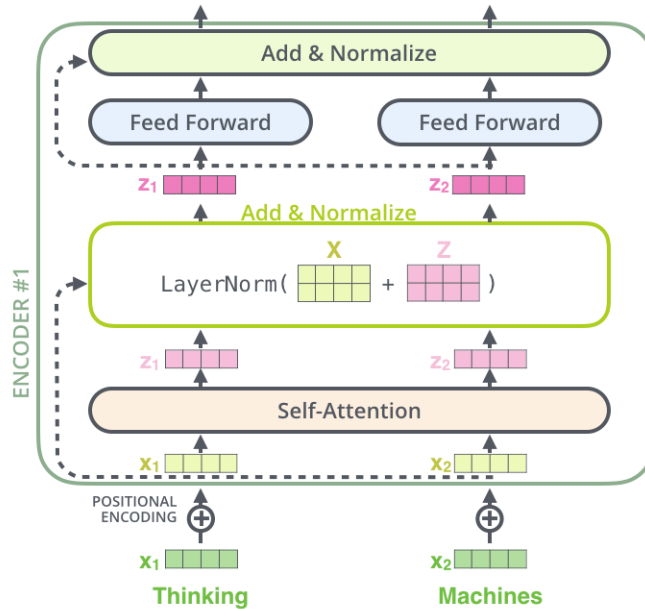
```
    def __init__(self, features, eps=1e-6):
        super(LayerNorm, self).__init__()
        self.a_2 = nn.Parameter(torch.ones(features))
        self.b_2 = nn.Parameter(torch.zeros(features))
        self.eps = eps
```

```
    def forward(self, x):
        mean = x.mean(-1, keepdim=True)
        std = x.std(-1, keepdim=True)
        return self.a_2 * (x - mean) / (std + self.eps) + self.b_2
```



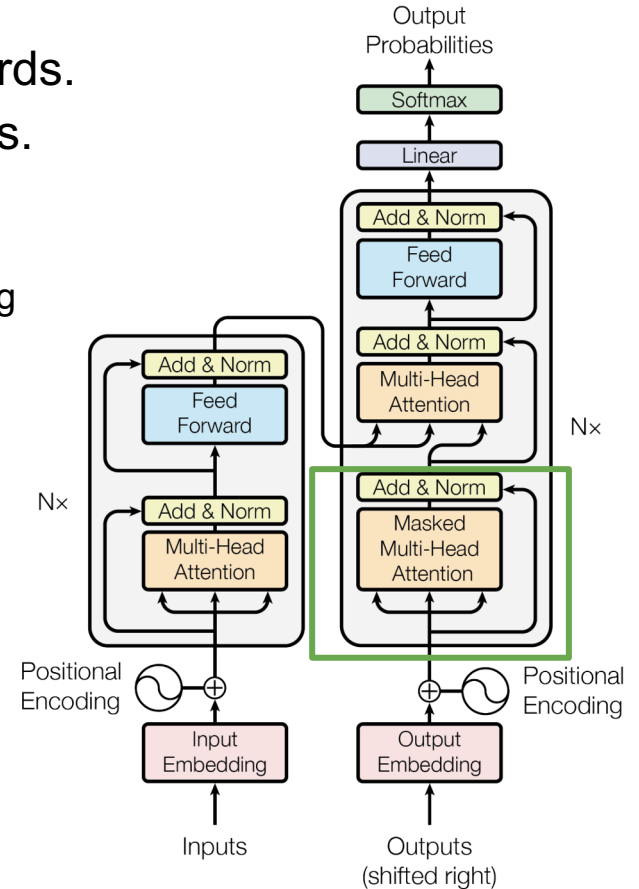
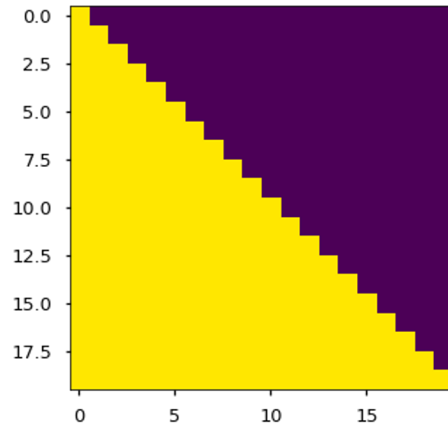
# Residual Connection & Layer Normalization

- To put them together



# Masked Self Attention in Decoder

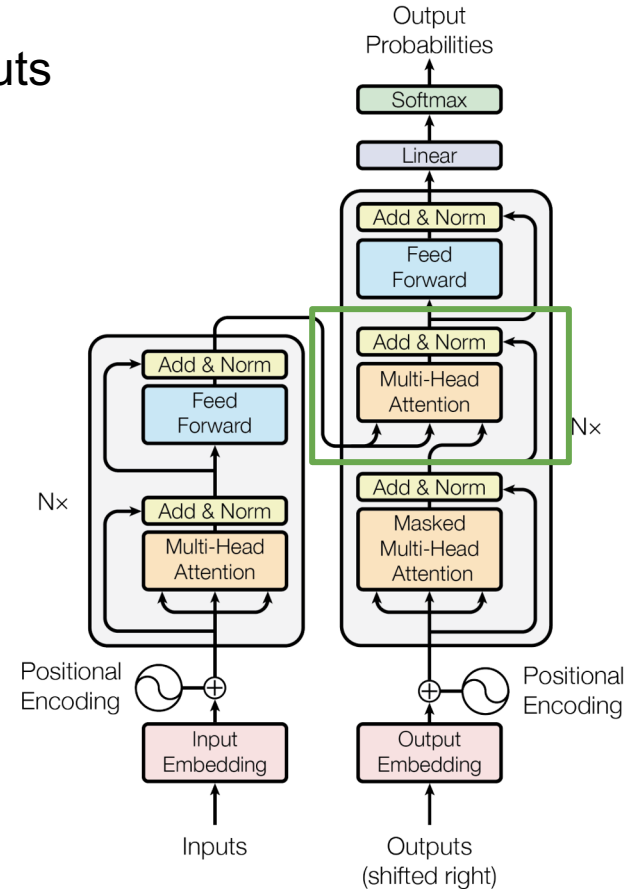
- In encoder, we can always look at all input words.
- In decoder, we can only look at previous words.
  - That means, we need to "**mask**" future words when performing attention.
  - Words are blocked for attending to future words during training.





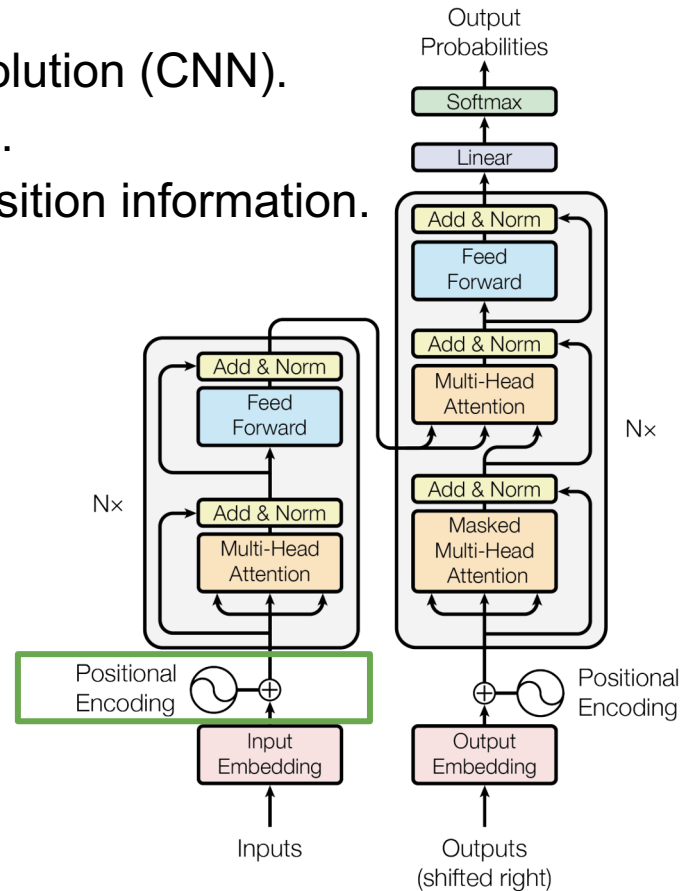
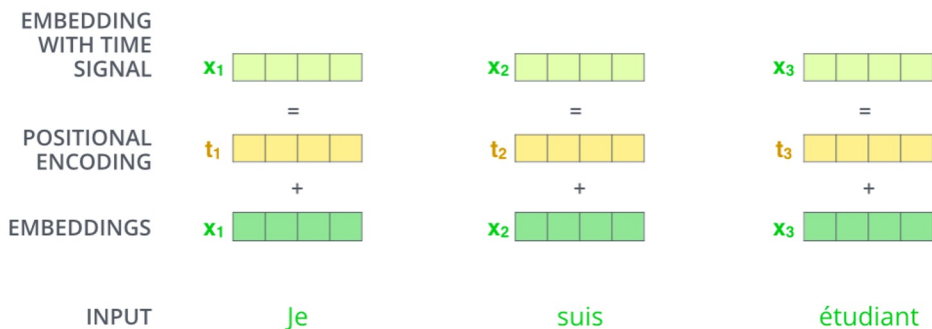
# Encoder-Decoder Cross Attention

- Attention between encoder self-attention outputs and decoder masked self attention outputs.



# Positional Encoding

- There is no recurrence (RNN) and no convolution (CNN).
- Self-attention do not have order information.
- So we need positional encoding to bring position information.



# Sinusoidal Positional Encoding

- There are many different choices of positional encodings ([Gehring et al., 2017](#))
  - some are **learned** (BERT, Devlin et al. 2018) and some are fixed
  - some for **relative** position and some for **absolute** position.
- In Transformer, we use **Sinusoidal position encodings**: fixed embeddings and can generalize to any sequence length.

# Sinusoidal Positional Encoding

- There are many different choices of positional encodings ([Gehring et al., 2017](#))
  - some are learned and some are fixed
  - some for relative position and some for absolute position.
- In Transformer, we use **Sinusoidal position encodings**: fixed embeddings and can generalize to any sequence length.
- Intuition: (4-dimensional position encodings for 16 positions)

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

# Sinusoidal Positional Encoding

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

4-dimensional position  
encodings for 16 positions

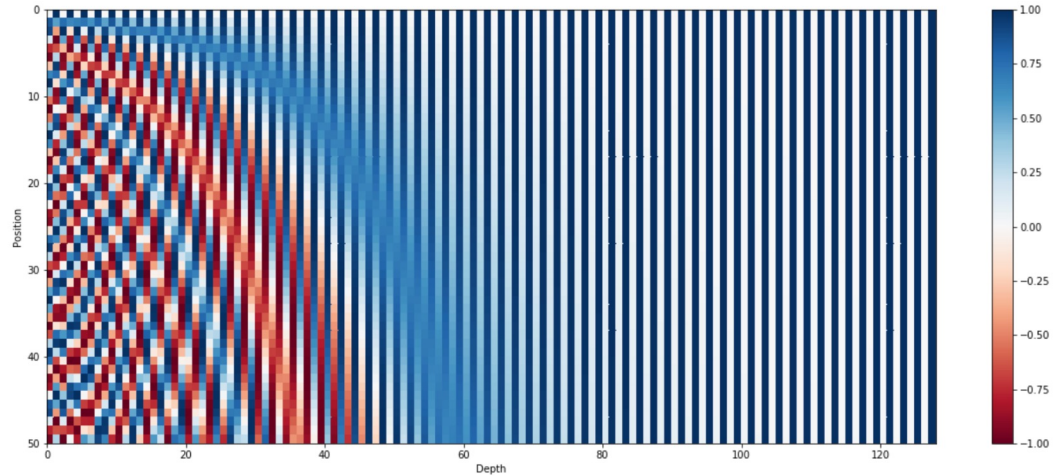
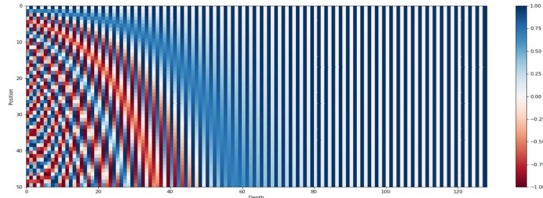


Figure 2 - The 128-dimensional positional encoding for a sentence with the maximum length of 50. Each row represents the embedding vector  $\vec{p}_t$

# Sinusoidal Positional Encoding



$$\vec{p}_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k + 1 \end{cases}$$

$$\text{with } \omega_k = \frac{1}{10000^{2k/d}}$$

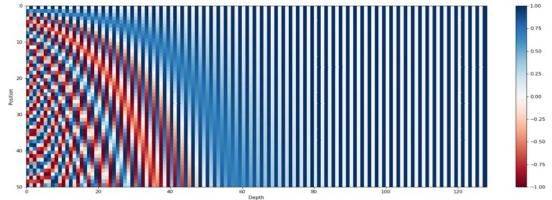
t: position index  
i: dimension index  
d: total dimension

$$\vec{p}_t = \begin{bmatrix} \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \\ \sin(\omega_2 \cdot t) \\ \cos(\omega_2 \cdot t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} \cdot t) \\ \cos(\omega_{d/2} \cdot t) \end{bmatrix}_{d \times 1}$$

# Sinusoidal Positional Encoding - Properties

- It should output a unique encoding for each time-step (word's position in a sentence)
- Distance between any two time-steps should be consistent across sentences with different lengths.
- Our model should generalize to longer sentences without any efforts. Its values should be bounded.
- It must be deterministic.

# Positional Encoding



- “Attention is all you need”: Our model should generalize to longer sentences without **any efforts**. Its values should be bounded. Deterministic
- There are many different choices of positional encodings ([Gehring et al., 2017](#))
  - some are **learned** ?



# Other Tricks in Transformers

- Subword Tokenization
- Optimizer
- Label Smoothing

# OOV Issue

How to handle unknown words, i.e., a word at test time that we've never seen in our training data. It is also called OOV (out-of-vocabulary) words.

If we use word-level tokenization, we cannot have an index for unknown words, and we cannot have a word embedding for unknown words.

# OOV Issue

**Old solution:** replace low-frequency words in training data with a special token <UNK>; then after training, we have a word embedding for <UNK>; if we have an unknown word during testing, we replace it with <UNK>.

New solution: **Subword tokenization.**

# Subword Tokenization

Use Subword Units for tokenization.

Based on a simple algorithm called byte pair encoding (Gage, 1994).

First used successfully for NLP for machine translation by (Sennrich et al., 2016).

# Byte Pair Encoding

Perform pre-tokenization

word	frequency
hug	10
pug	5
pun	12
bun	4
hugs	5

Form a base vocabulary: **b, g, h, n, p, s, u**

# Byte Pair Encoding

Count all the character pair.

<b>word</b>	<b>frequency</b>	<b>character pair</b>	<b>frequency</b>
h+u+g	10	<i>ug</i>	
p+u+g	5	<i>pu</i>	
p+u+n	12	<i>un</i>	
b+u+n	4	<i>hu</i>	
h+u+g+s	5	<i>gs</i>	

# Byte Pair Encoding

Count all the character pair.

<b>word</b>	<b>frequency</b>	<b>character pair</b>	<b>frequency</b>
h+u+g	10	<i>ug</i>	20
p+u+g	5	<i>pu</i>	17
p+u+n	12	<i>un</i>	16
b+u+n	4	<i>hu</i>	15
h+u+g+s	5	<i>gs</i>	5

# Byte Pair Encoding

Merge the most frequent character pair, i.e., **ug**, and add it to the vocabulary.

Then rotokenize and repeat this process.

The next to merge is ?

---

word	frequency
<i>h+ug</i>	10
<i>p+ug</i>	5
<i>p+u+n</i>	12
<i>b+u+n</i>	4
<i>h+ug+s</i>	5



# Byte Pair Encoding

Merge the most frequent character pair, i.e., **ug**, and add it to the vocabulary.  
Then rotokenize and repeat this process.

The next to merge is **un**, so we add un to the vocabulary.

---

word	frequency	character pair	frequency
<i>h+ug</i>	10	<i>un</i>	16
<i>p+ug</i>	5	<i>h+ug</i>	15
<i>p+u+n</i>	12	<i>pu</i>	12
<i>b+u+n</i>	4	<i>p+ug</i>	5
<i>h+ug+s</i>	5	<i>ug+s</i>	5

# Byte Pair Encoding

Repeat until some fixed number of merges, or until we reach a target vocab size.

<b>word</b>	<b>frequency</b>
<i>hug</i>	10
<i>p+ug</i>	5
<i>p+un</i>	12
<i>b+un</i>	4
<i>hug + s</i>	5

new vocab: **b, g, h, n, p, s, u, ug, un, hug**

# Byte Pair Encoding

To avoid OOV, all possible characters / symbols need to be included in the base vocab.

- This can be huge if including all **unicode** characters (size 144,697)!
- GPT-2 uses **bytes** as the base vocabulary (size 256) and then applies BPE on top of this sequence (with some rules to prevent certain types of merges).
- GPT-2 has a vocabulary size of 50,257, which corresponds to the 256 bytes base tokens, a special end-of-text token and the symbols learned with 50,000 merges.

Commonly have vocabulary sizes of 32K to 64K

# Encoding and Decoding

## Encoding

- Sort the vocab from longest subwords to shortest subwords.
- Go through the vocab, and replace the longest subword first.
- This is very computationally expensive.
- In practice, we can cache how popular words are encoded.

## Decoding

- In fact, how to determine word boundaries.

# Encoding and Decoding

## Encoding

- Sort the vocab from longest subwords to shortest subwords.
- Go through the vocab, and replace the longest subword first.
- This is very computationally expensive.
- In practice, we can cache how popular words are encoded.

## Decoding

- In fact, we will add `</w>` to each word to indicate word boundaries.
- For example, if the encoded sequence is [“the</w>”, “high”, “est</w>”, “moun”, “tain</w>”], we immediately know the decoded sequence “the highest mountain</w>”.

# Other subword Tokenization

## Byte Pair Encoding

- merge by frequency
- used by GPT-2.

## WordPiece (Schuster et al., 2012)

- merge by likelihood given by language models, not by frequency
- used by BERT.

## SentencePiece (Kudo et al., 2018)

- All tokenization algorithms described so far have the same problem: It is assumed that the input text uses spaces to separate words. However, not all languages use spaces to separate words.
- SentencePiece can train subword models directly from raw sentences

# Can we don't do tokenization at all?

Old solution: use <UNK>.

New solution: Subword tokenization

**New New solution:** tokenizer-free and directly operate on bytes!

- [CANINE: Pre-training an Efficient Tokenization-Free Encoder for Language Representation.](#) (Clark et al, 2022)
- [ByT5: Towards a token-free future with pre-trained byte-to-byte models.](#)

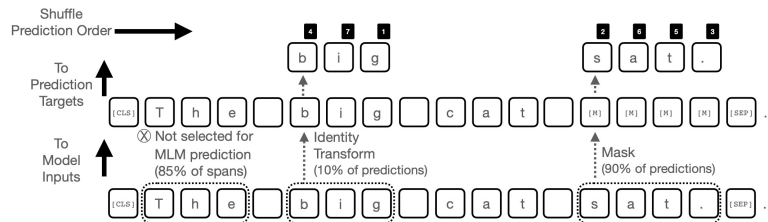
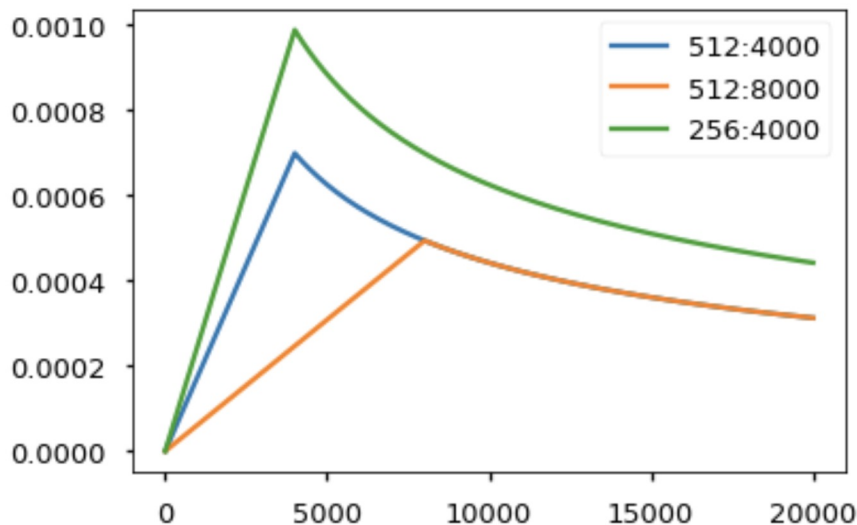


Figure 2: CANINE-C pre-training data preparation (§3.2.1). Character-wise predictions are made by an autoregressive transformer layer that predicts then reveals one character at a time, in a shuffled order.

# Optimizer

## Adam Optimizer with Linear Warmup Learning Rate

We used the Adam optimizer (cite) with  $\beta_1 = 0.9$ ,  $\beta_2 = 0.98$  and  $\epsilon = 10^{-9}$ . We varied the learning rate over the course of training, according to the formula:  $lr_{rate} = d_{model}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$  This corresponds to increasing the learning rate linearly for the first  $warmup\_steps$  training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. We used  $warmup\_steps = 4000$ .





- Adam

$$m_0 = 0, v_0 = 0$$

$$m_{t+1} \leftarrow \beta_1 m_t + (1 - \beta_1) \nabla_{\theta} \mathcal{L}(\theta)$$

Momentum

$$v_{t+1} \leftarrow \beta_2 v_t + (1 - \beta_2) \nabla_{\theta} \mathcal{L}(\theta)^2$$

RMS Prop

$$\theta_j \leftarrow \theta_j - \frac{\epsilon}{\sqrt{v_{t+1} + 1e^{-5}}} m_{t+1}$$

RMS Prop + Momentum

Algo of Adam.

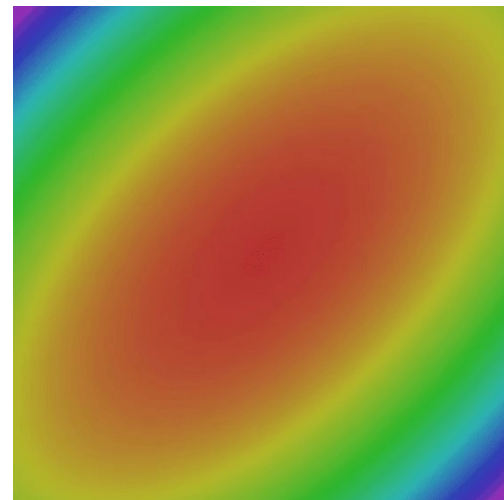
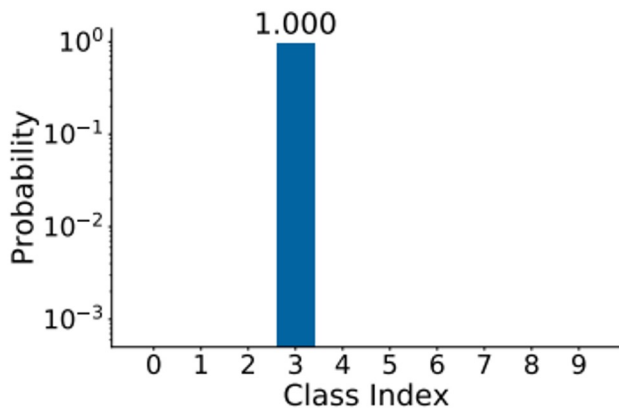


Fig. 6 Comparison of all optimization algorithms. (Visualization)

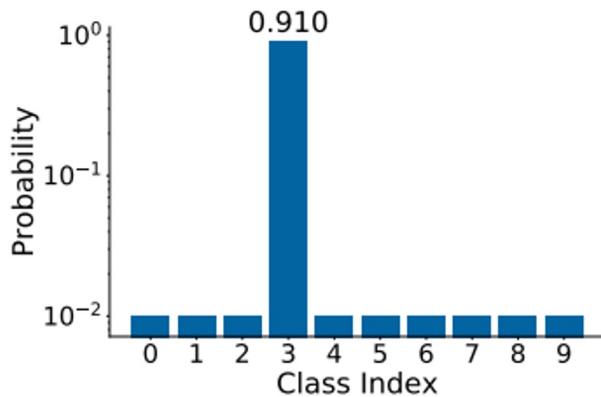
# Label Smoothing

During training, we employed label smoothing of value  $\epsilon_{ls} = 0.1$  (cite). This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score.

*We implement label smoothing using the KL div loss. Instead of using a one-hot target distribution, we create a distribution that has **confidence** of the correct word and the rest of the **smoothing** mass distributed throughout the vocabulary.*



(a) Hard Label

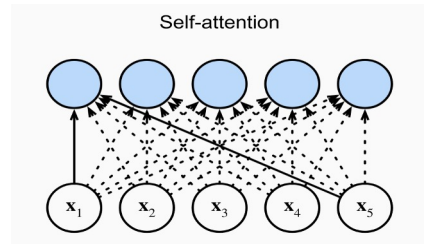


(b) LS

# Computational Complexity: Quadratic in Length!

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types.  $n$  is the sequence length,  $d$  is the representation dimension,  $k$  is the kernel size of convolutions and  $r$  the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$



# Why Transformers

- Everything is attention (and FFNNs), you can train much larger networks if you have the right hardware (GPUs, TPUs) due to the improved parallelism. So much cheaper than RNNs.
- Downside: Only a few companies have the expertise and machine power (TPUs rather than GPUs) and money to build large language models.
  - GPT-2: 1.5B parameters
  - GPT-3: 175B parameters

# Results

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 [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	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>41.0</b>	$2.3 \cdot 10^{19}$	

# Model Variations

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

	$N$	$d_{\text{model}}$	$d_{\text{ff}}$	$h$	$d_k$	$d_v$	$P_{\text{drop}}$	$\epsilon_{ls}$	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096						4.75	26.2	90	
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)				positional embedding instead of sinusoids						4.92	25.7	
big	6	1024	4096	16			0.3		300K	<b>4.33</b>	<b>26.4</b>	213

# Transformer-based Language Models

It is becoming the underlying architecture of most popular language models. Two most famous are:

- GPT: OpenAI Transformer-based Language Models
- BERT: Google Transformer-based Language Models
- RoBERTa: Facebook Transformer-based Language Models

# Transformer beyond NLP

Transformer can also be used in many other applications: [Vision](#), [Speech](#), [Reinforcement Learning](#), ...

People even consider we no longer need CNN/RNN or any other neural nets, and Transformer is unifying many ML problems.

- Check Facebook's [data2vec](#)
- Check this [Twitter](#) post by Andrej Karpathy

*"But as of approx. last two years, even the neural net architectures across all areas are starting to look identical - a Transformer (definable in ~200 lines of PyTorch [github.com/karpathy/minGP\\_...](#)), with very minor differences. Either as a strong baseline or (often) state of the art."*



# Reading

[Annotated Transformers](#) by Alexander M. Rush

[The Illustrated Transformer](#) by Jay Alammar