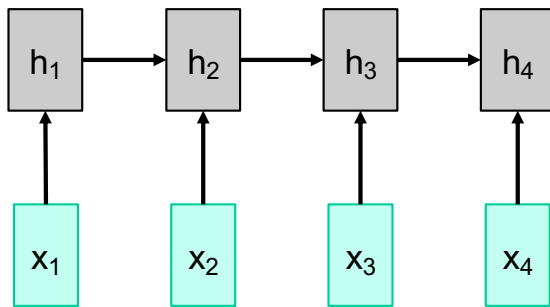
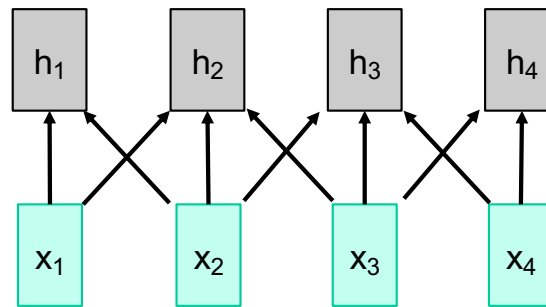


Review: Sequence modeling methods

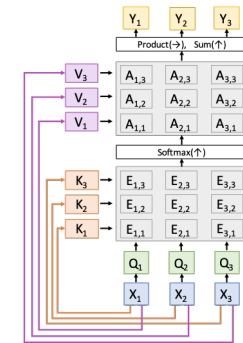
RNNs



1D convolutional networks



Transformers



Works on **ordered sequences**

- Pros: Not limited by fixed context size (in principle): After one RNN layer, h_T "sees" the whole sequence
- Con: Hidden states have limited expressive capacity
- Con: Not parallelizable: need to compute hidden states sequentially

Works on **1D grids**

- Pro: Each output can be computed in parallel (at training time)
- Con: Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence

- Works on **sets of vectors**

Transformers: The nuts and bolts

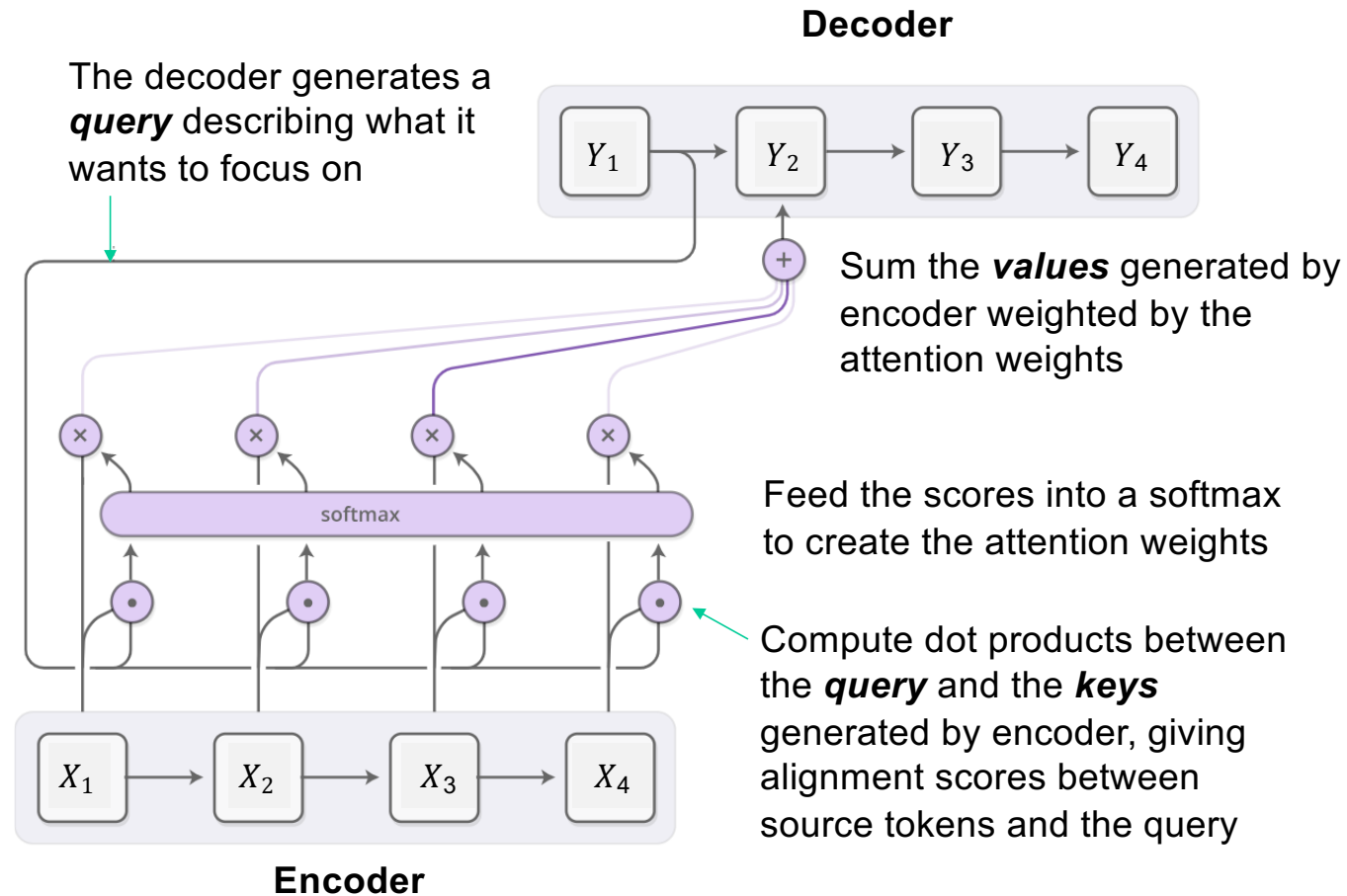


<https://xkcd.com/1838/>

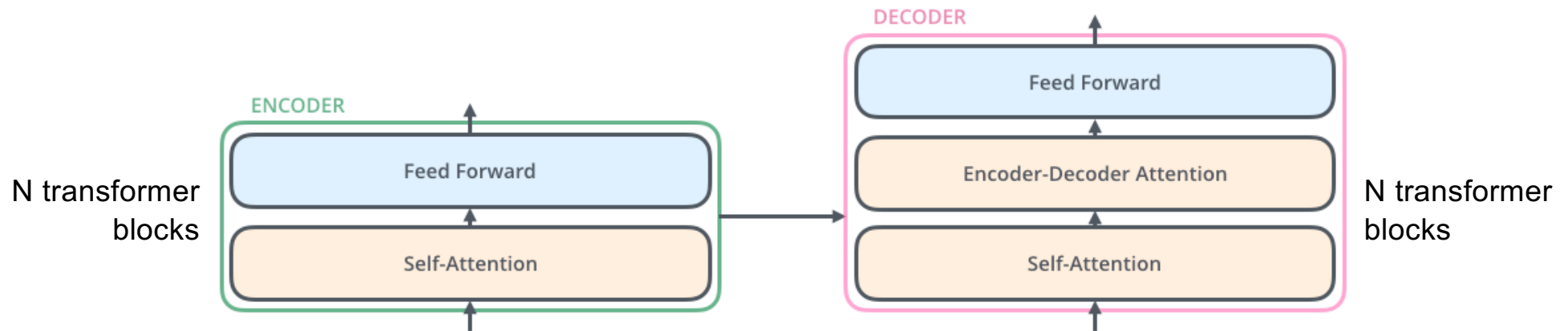
Outline

- Attention models
- The transformer block in detail
- Early architectures for language modeling

Review: Key-Value-Query attention model



Review: Basic transformer model



- **Encoder self-attention:** queries, keys, and values come from previous layer of encoder
- **Decoder self-attention:** values corresponding to future decoder outputs are masked out
- **Encoder-decoder attention:** queries come from previous decoder layer, keys and values come from output of encoder

Key-Value-Query attention in transformers

Key-Value-Query attention in transformers

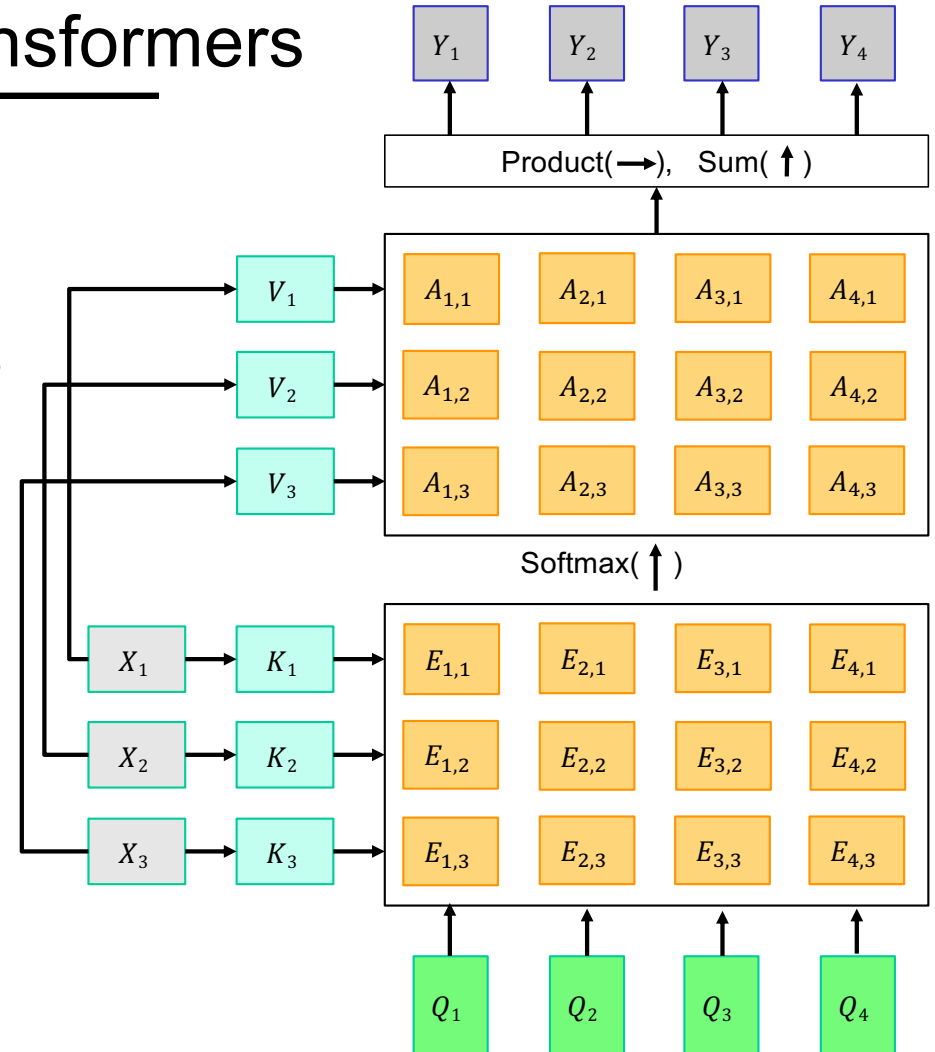
- Key vectors: $K = XW_K$
- Value Vectors: $V = XW_V$
- Query vectors
- Similarities: *scaled dot-product attention*

$$E_{i,j} = \frac{(Q_i \cdot K_j)}{\sqrt{D}} \text{ or } E = QK^T / \sqrt{D}$$

(D is the dimensionality of the keys)

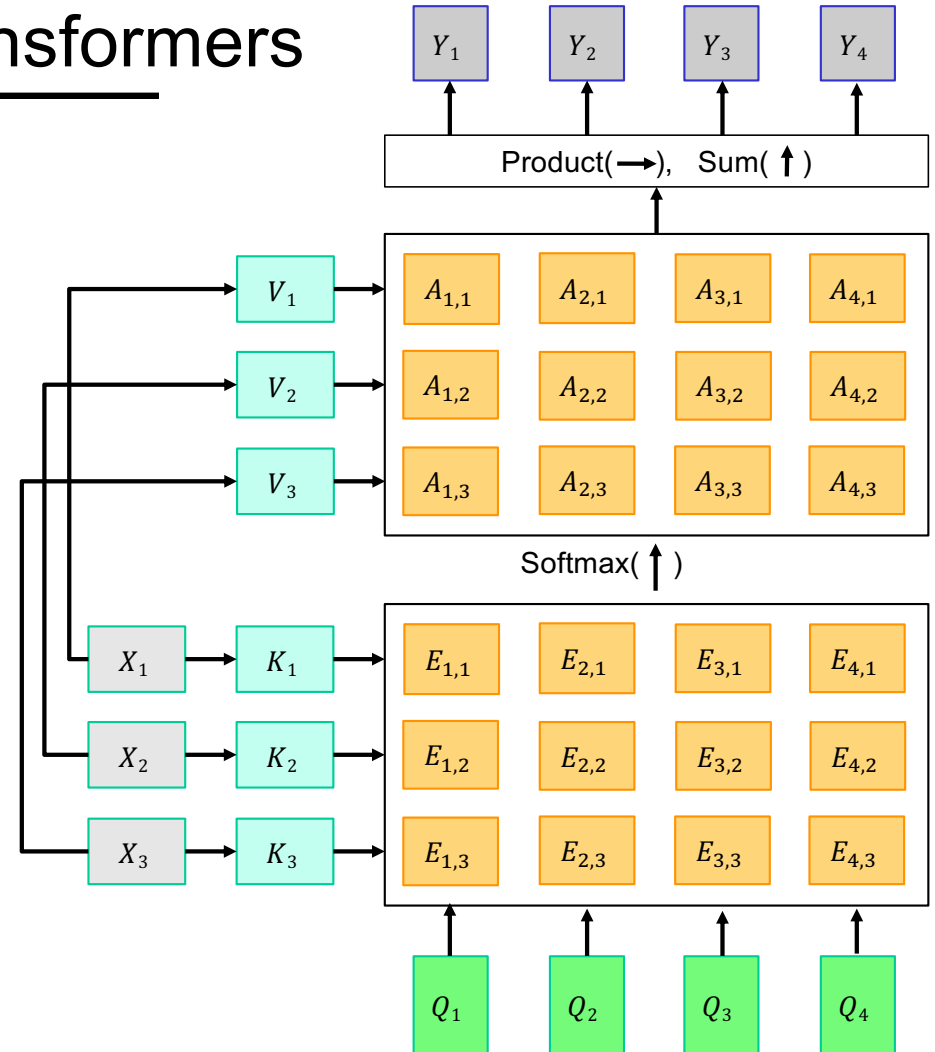
- Attn. weights: $A = \text{softmax}(E, \text{dim} = 1)$
- Output vectors:

$$Y_i = \sum_j A_{i,j} V_j \text{ or } Y = AV$$



Key-Value-Query attention in transformers

- How does permuting the order of the *queries* change the output?
- How does changing the order of the *keys/values* change the output?



Self-attention layer

Self-attention layer

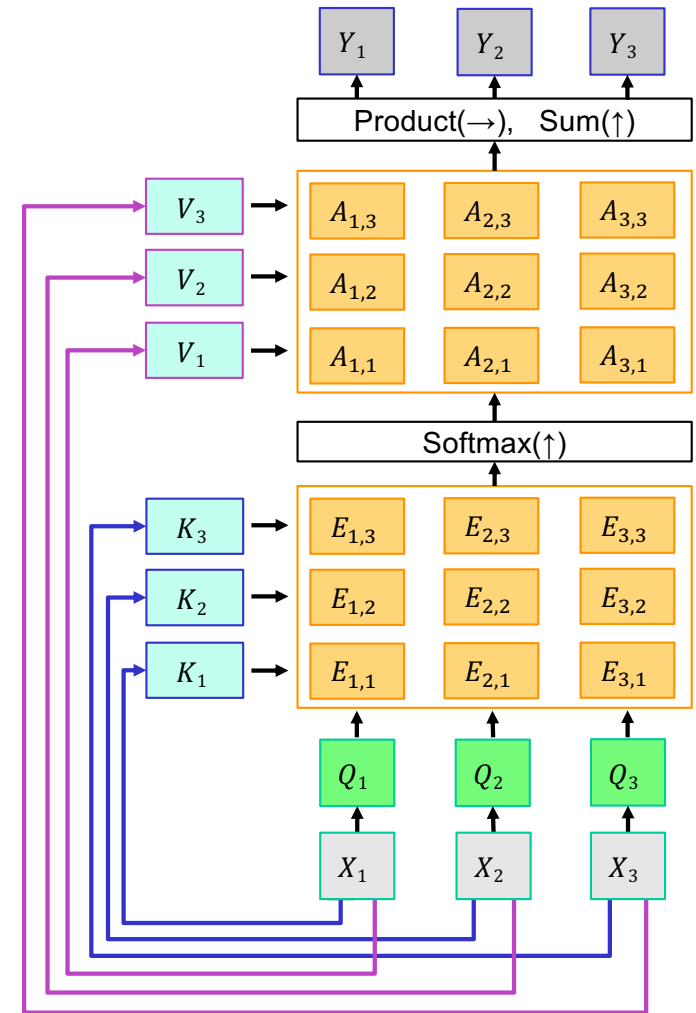
- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$
- Similarities: *scaled dot-product attention*

$$E_{i,j} = \frac{(Q_i \cdot K_j)}{\sqrt{D}} \text{ or } E = QK^T / \sqrt{D}$$

(D is the dimensionality of the keys)

- Attn. weights: $A = \text{softmax}(E, \text{dim} = 1)$
- Output vectors:

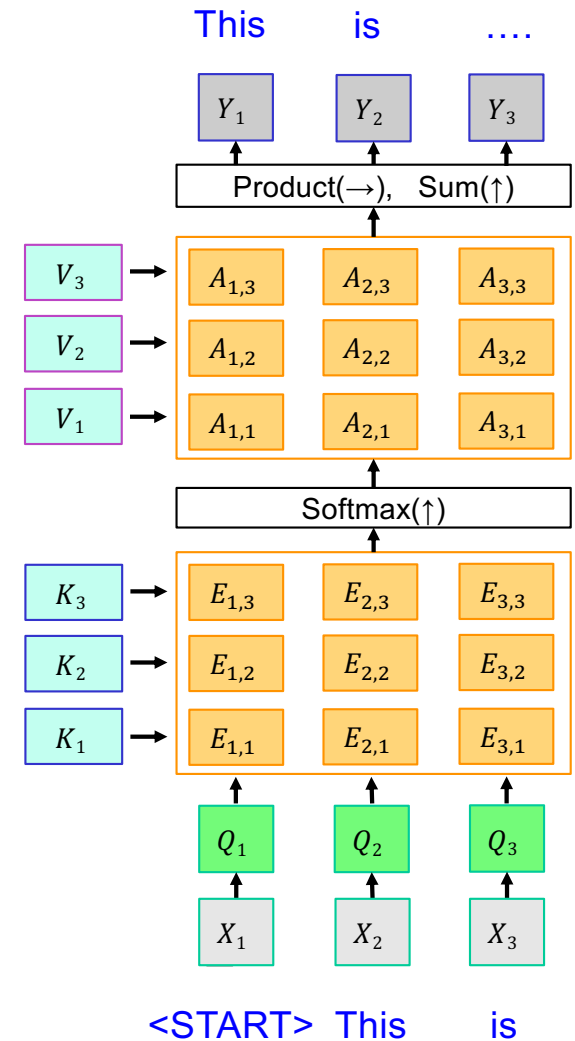
$$Y_i = \sum_j A_{i,j} V_j \text{ or } Y = AV$$



Masked self-attention layer

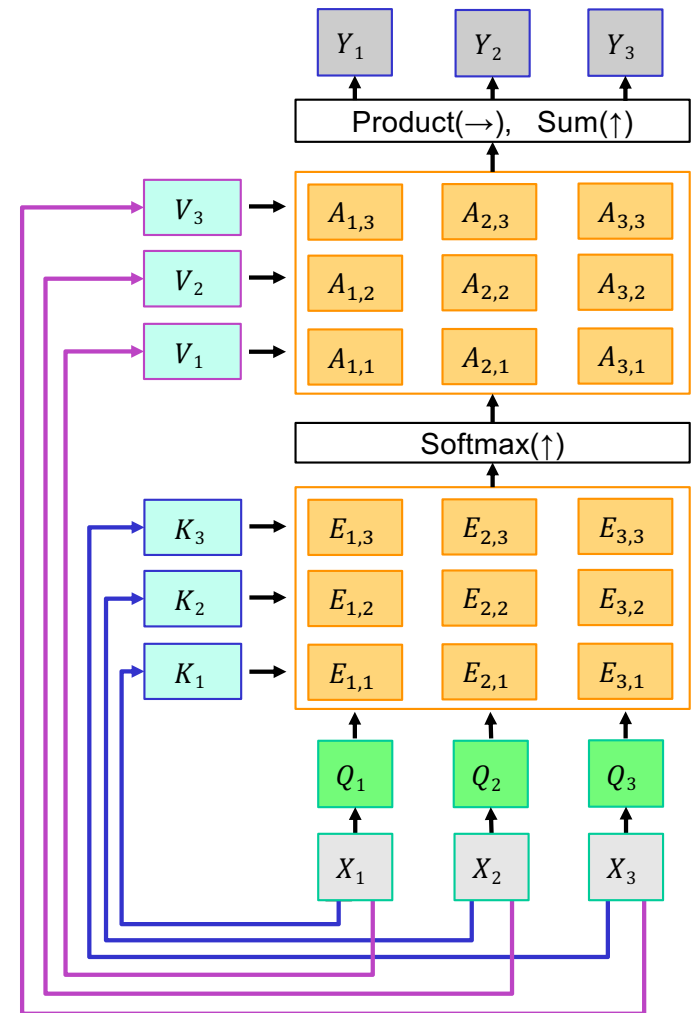
Masked self-attention layer

- The decoder should not “look ahead” in the output sequence



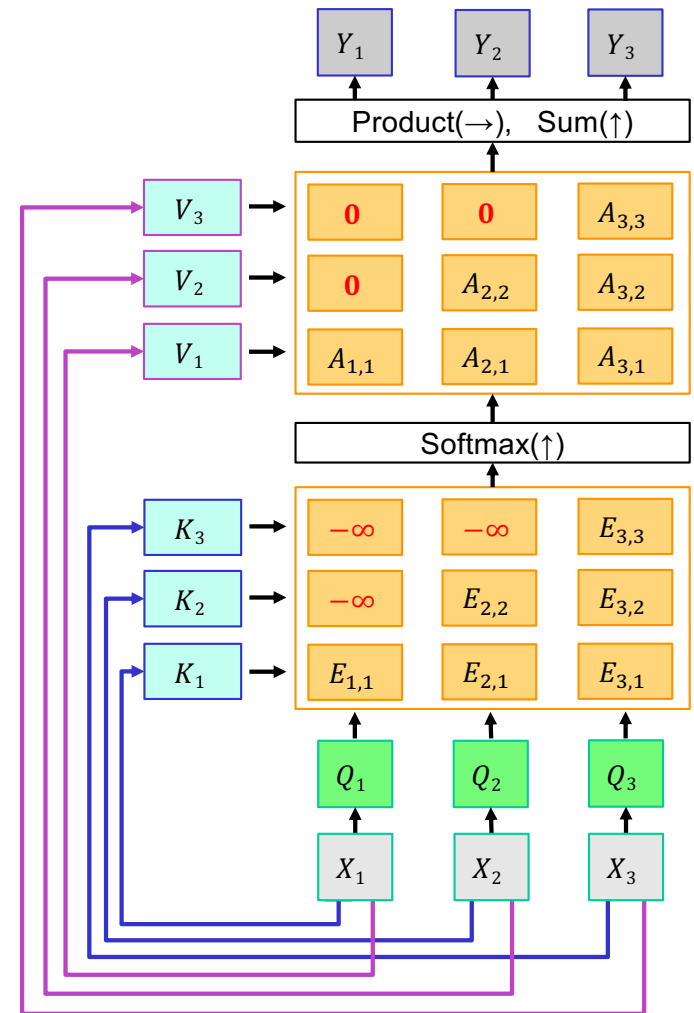
Masked self-attention layer

- The decoder should not “look ahead” in the output sequence

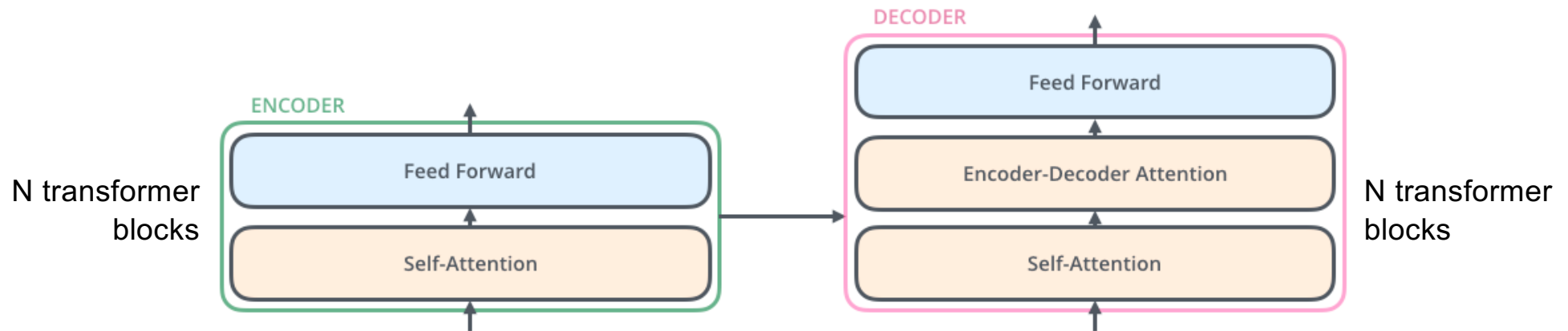


Masked self-attention layer

- The decoder should not “look ahead” in the output sequence



Attention mechanisms: Summary



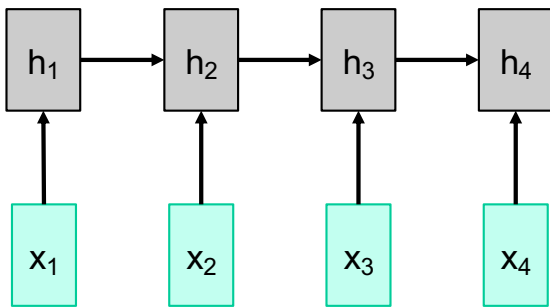
- **Encoder self-attention:** queries, keys, and values come from previous layer of encoder
- **Decoder self-attention:** values corresponding to future decoder outputs are masked out
- **Encoder-decoder attention:** queries come from previous decoder layer, keys and values come from output of encoder

Attention mechanisms: Illustration

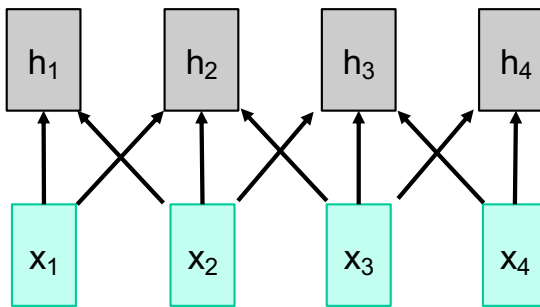
<https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>

Transformers: Pros and cons

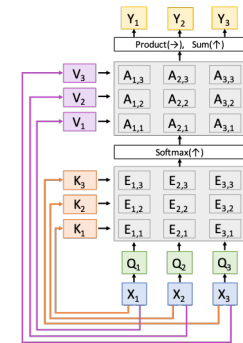
RNNs



1D convolutional networks



Transformers



Works on **ordered sequences**

- Pros: Not limited by fixed context size (in principle): After one RNN layer, h_T "sees" the whole sequence
- Con: Not parallelizable: need to compute hidden states sequentially
- Con: Hidden states have limited expressive capacity

Works on **multidimensional grids**

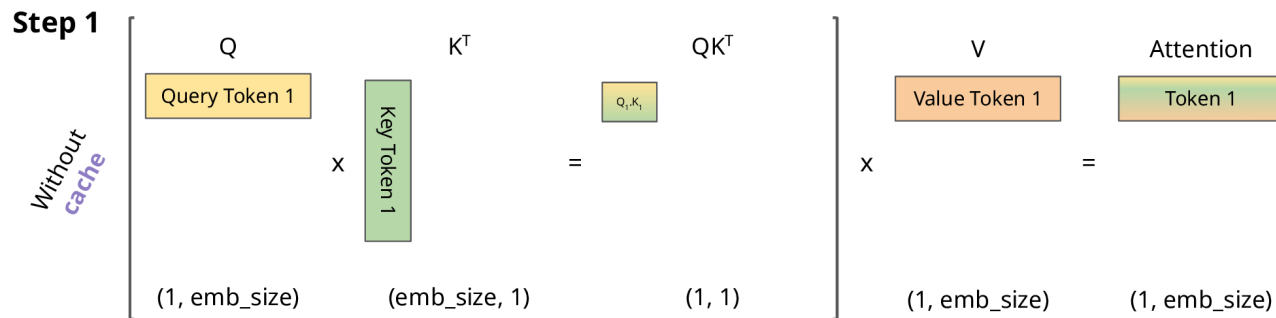
- Pro: Each output can be computed in parallel (at training time)
- Con: Need to stack many conv layers for outputs to "see" the whole sequence

Works on **sets of vectors**

- Pro: Good at long sequences: after one self-attention layer, each output "sees" all inputs!
- Pro: Each output can be computed in parallel (at training time)
- Con: Memory-intensive: cost of attention operator is *quadratic* in input size

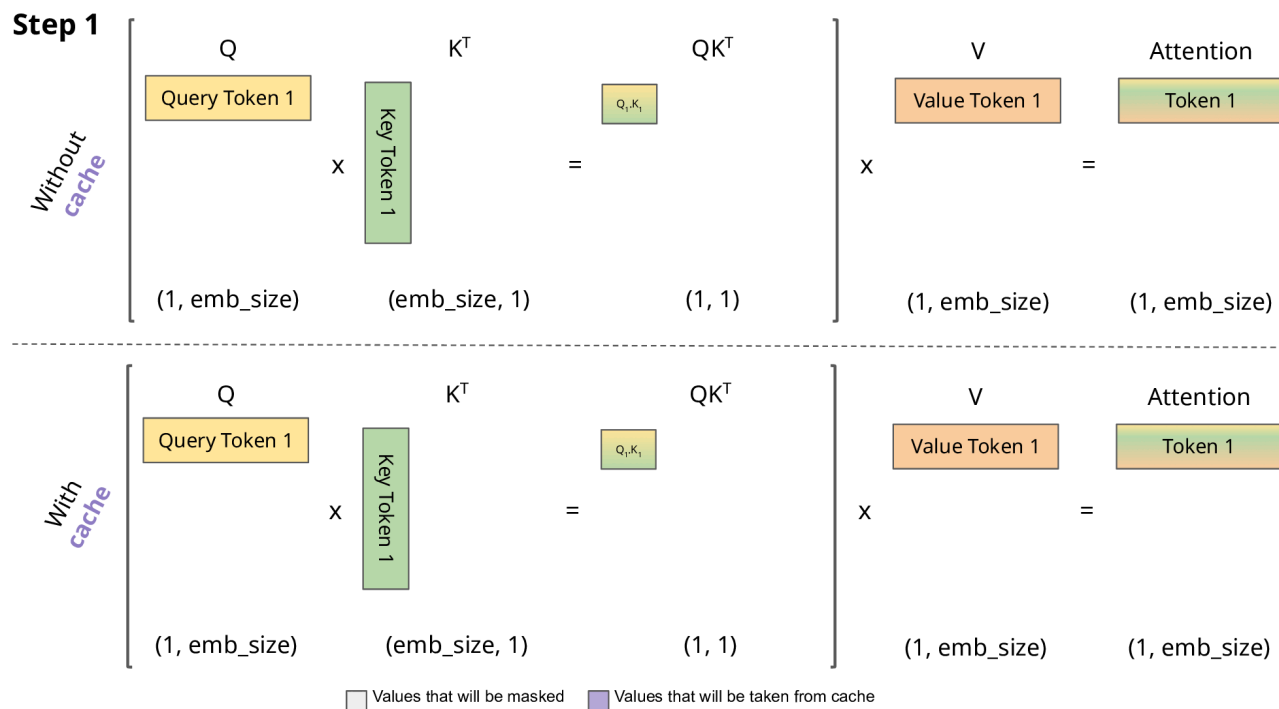
Making attention more efficient: KV-cache

- Idea: cache previous attention computations during autoregressive generation (decoding)



Making attention more efficient: KV-cache

- Idea: cache previous attention computations during autoregressive generation (decoding)



[Figure source](#)

Making attention more efficient: FlashAttention

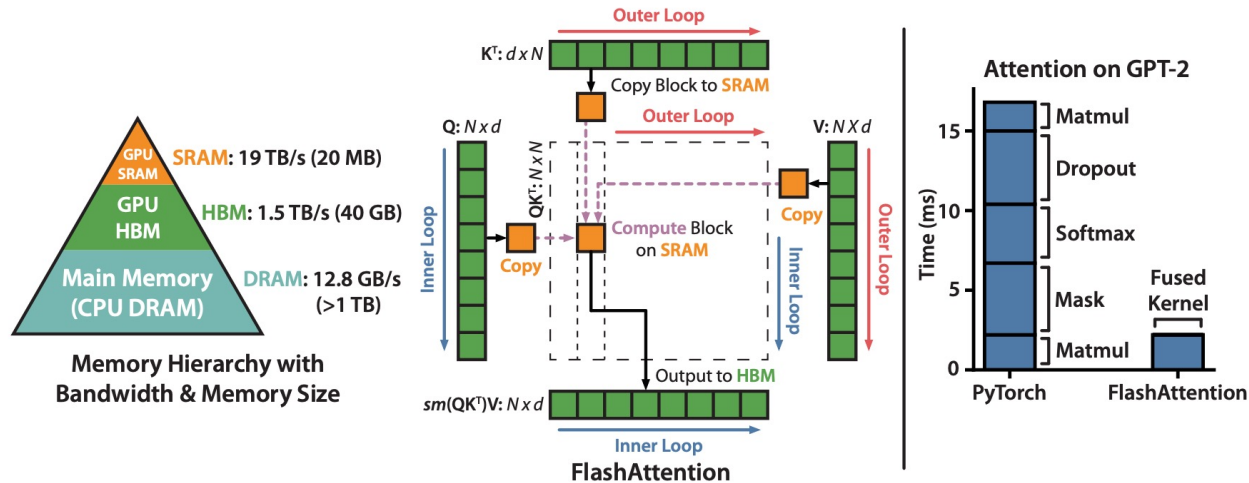


Figure 1: **Left:** FLASHATTENTION uses tiling to prevent materialization of the large $N \times N$ attention matrix (dotted box) on (relatively) slow GPU HBM. In the outer loop (red arrows), FLASHATTENTION loops through blocks of the K and V matrices and loads them to fast on-chip SRAM. In each block, FLASHATTENTION loops over blocks of Q matrix (blue arrows), loading them to SRAM, and writing the output of the attention computation back to HBM. **Right:** Speedup over the PyTorch implementation of attention on GPT-2. FLASHATTENTION does not read and write the large $N \times N$ attention matrix to HBM, resulting in an $7.6\times$ speedup on the attention computation.

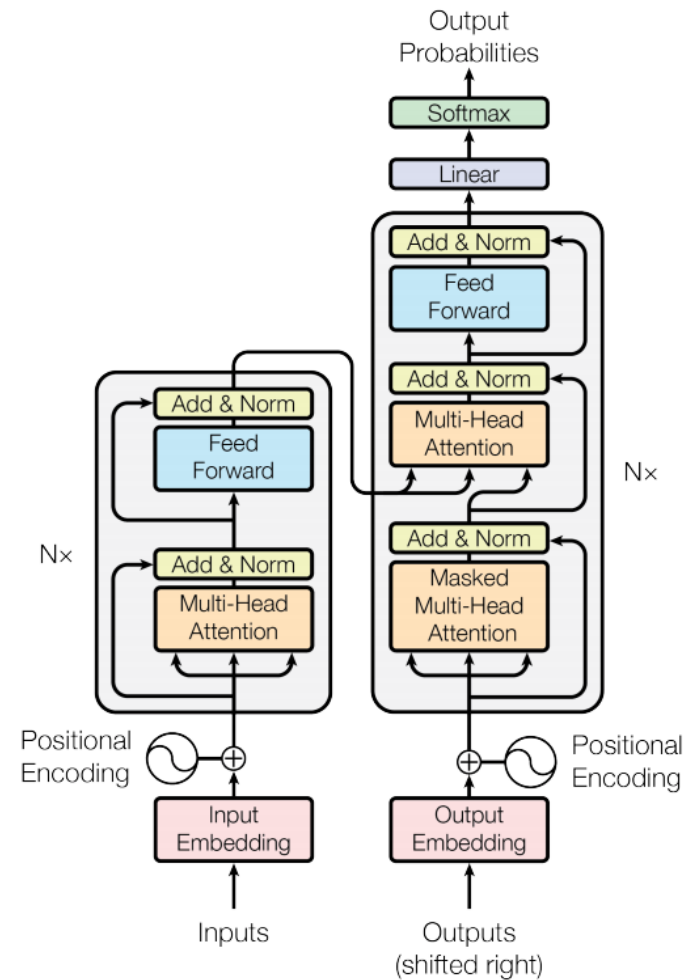
T. Dao et al. [FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness](#). NeurIPS 2022

Latest: [FlashAttention-4](#)

Supported in PyTorch: <https://pytorch.org/blog/pytorch2-2/>

Outline

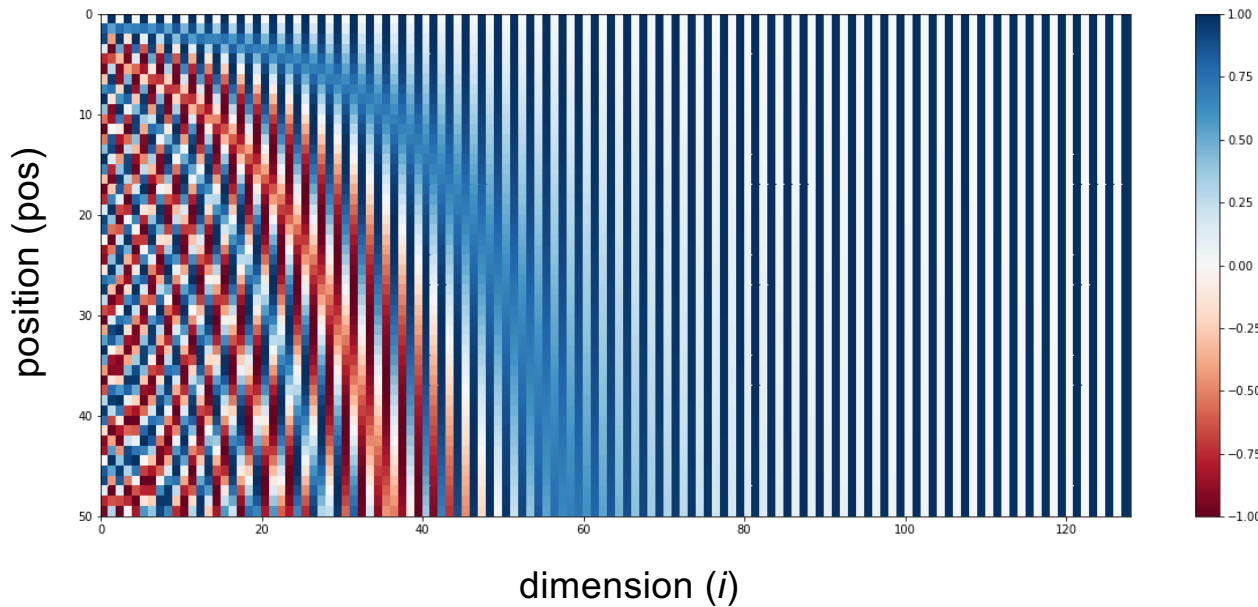
- Attention models
- The transformer block in detail



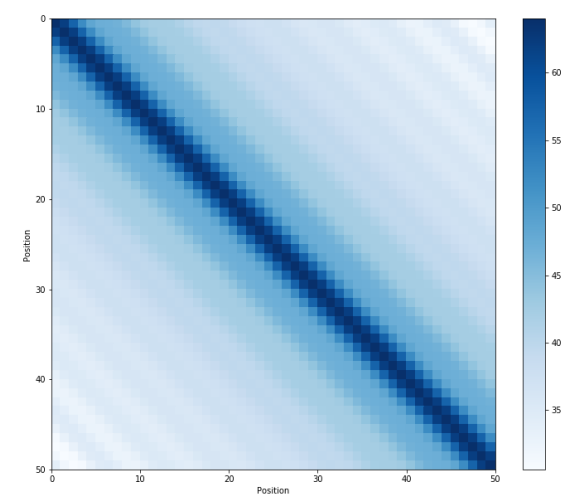
Positional embedding

- To give transformer information about ordering of tokens, add function of position (based on sines and cosines) to every input

Original embedding: $PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$
 $PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$



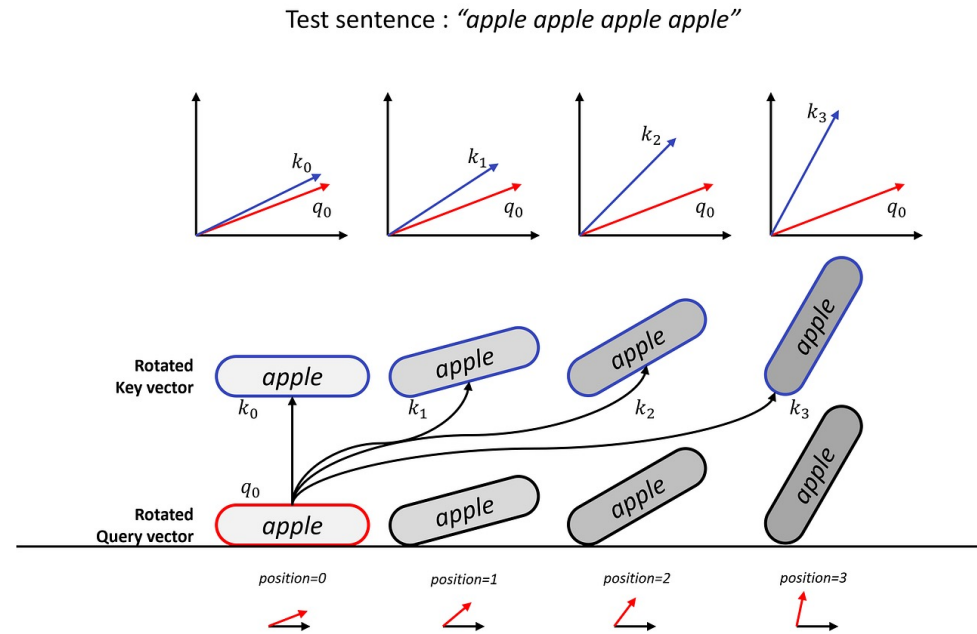
Dot product of every pair of positional embeddings



[Figure source](#)

RoPE: Rotary Positional Embedding

- Encode positional information by *rotating* key and query vectors by an angle that depends on the position



[Figure source](#)

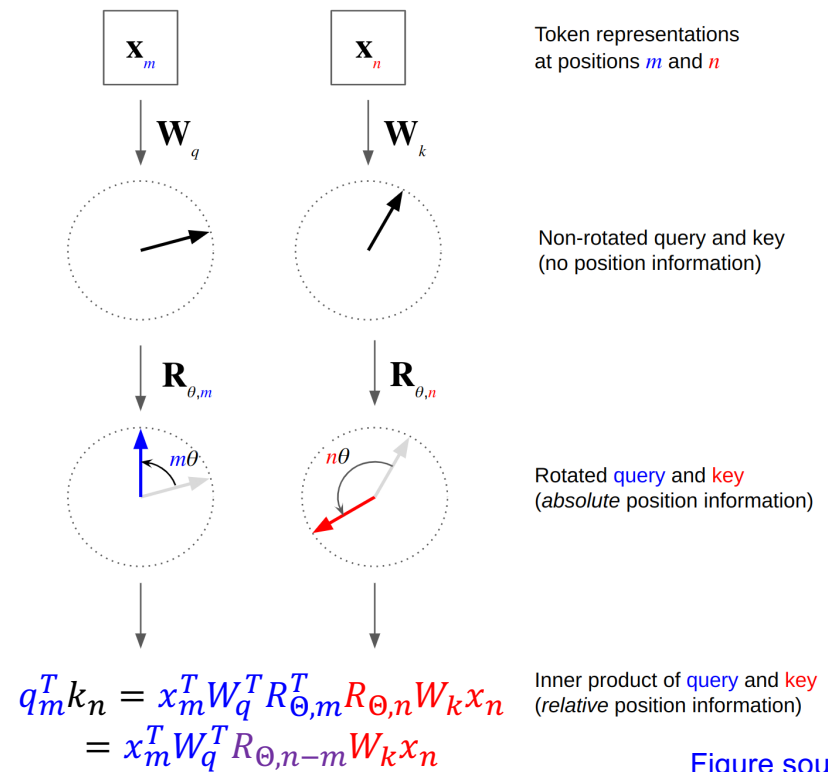
RoPE: Rotary Positional Embedding

- Encode positional information by *rotating* key and query vectors by an angle that depends on the position
- Compose special rotation matrix by stacking two-dimensional rotation matrices

$$\Theta = \{\theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \dots, d/2]\}$$
$$\mathbf{R}_{\Theta, m}^d = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

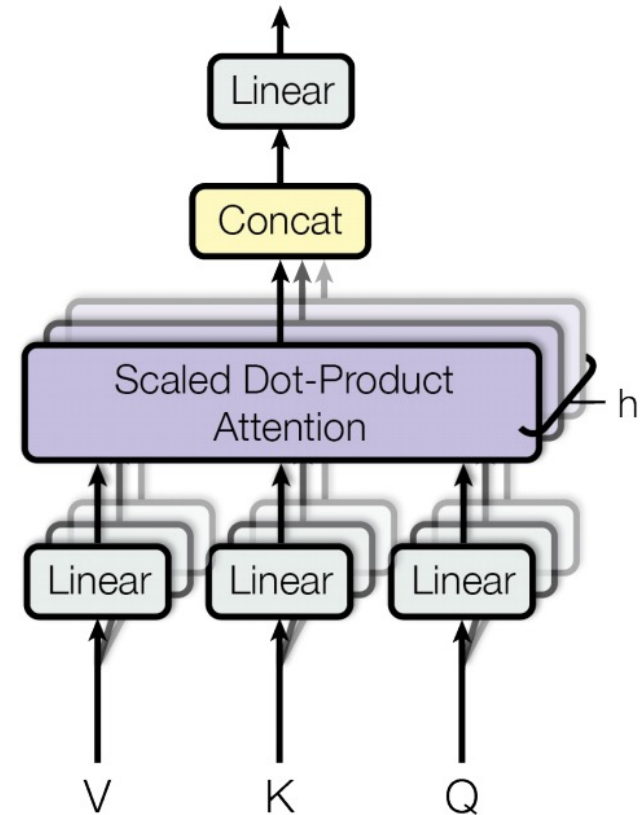
RoPE: Rotary Positional Embedding

- Encode positional information by *rotating* key and query vectors by an angle that depends on the position
- Compose special rotation matrix by stacking two-dimensional rotation matrices
- Dot product between keys and queries depends on their *relative*, not *absolute* position
- Apply this operation every time before computing attention



Multi-head attention

- Run h attention models in parallel on top of different linearly projected versions of Q, K, V ; concatenate and linearly project the results
- Intuition: enables model to attend to different kinds of information at different positions (see [visualization tool](#))



Multi-query and grouped-query attention

- Multi-query attention (MQA): different queries share the same keys and values – just need one KV-cache
- Grouped-query attention (GQA): queries in the same group share the same keys and values

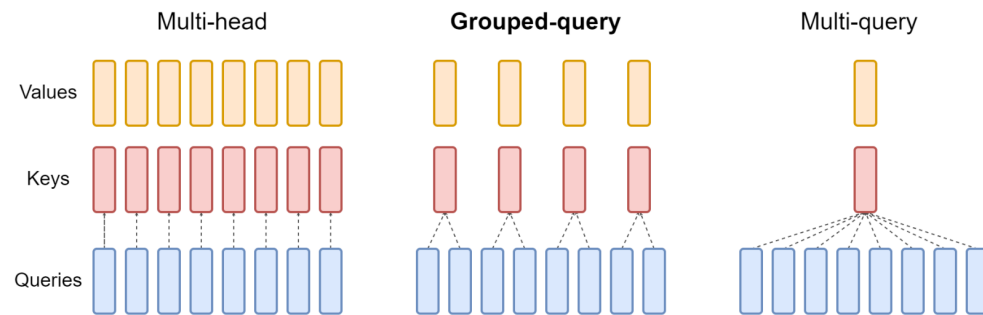


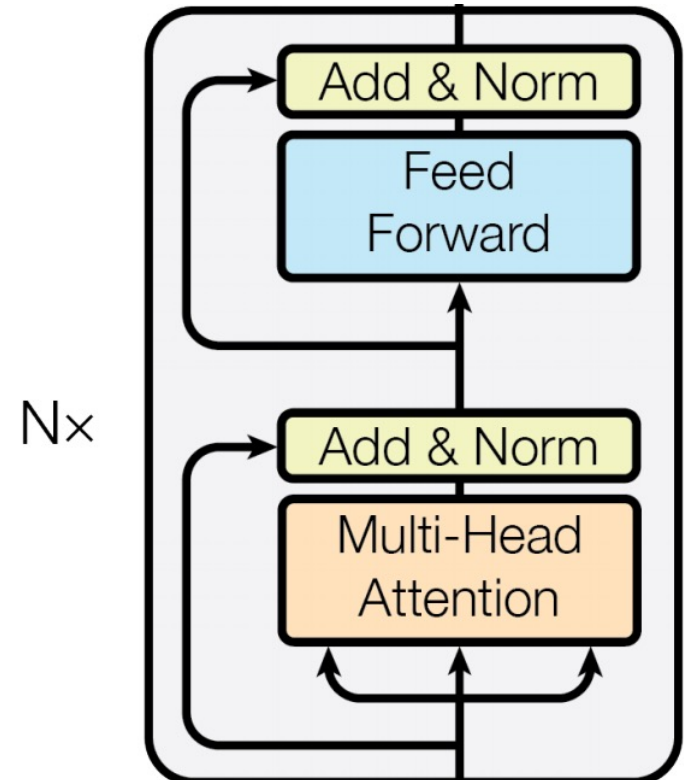
Figure 2: Overview of grouped-query method. Multi-head attention has H query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

N. Shazeer. [Fast Transformer Decoding: One Write-Head is All You Need](#). arXiv 2019

J. Ainslie et al. [GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints](#). EMNLP 2023

Feedforward network (FFN or MLP)

- Original transformer:
two linear layers with ReLUs in between, applied independently to each vector



Original architecture: $N = 12$ blocks,
embedding dimension = 512,
6 attention heads

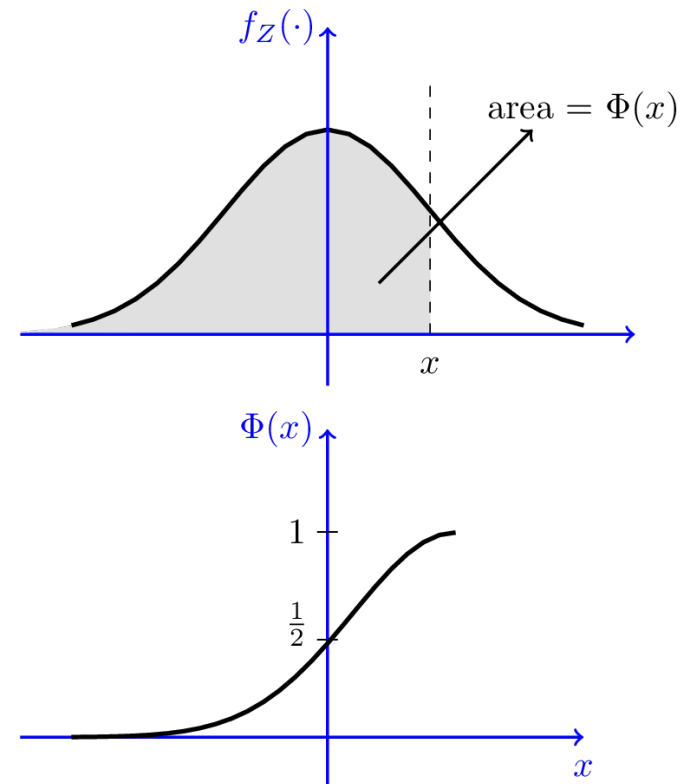
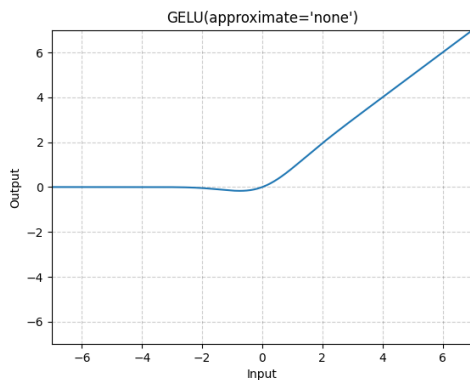
Nonlinearities beyond ReLU

- **GELU**: Gaussian Error Linear Unit

$\text{GELU}(x) = x * \Phi(x)$ where $\Phi(x)$ is the Gaussian CDF

Approximation:

$$\text{GELU}(x) \approx 0.5x * \left(1 + \text{Tanh} \left(\sqrt{\frac{2}{\pi}} * (x + 0.044715x^3) \right) \right)$$



[Figure source](#)

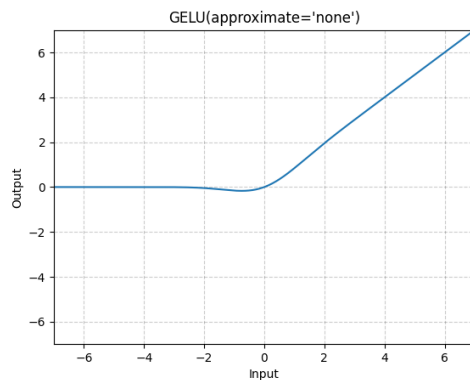
Nonlinearities beyond ReLU

- **GELU**: Gaussian Error Linear Unit

$GELU(x) = x * \Phi(x)$ where $\Phi(x)$ is the Gaussian CDF

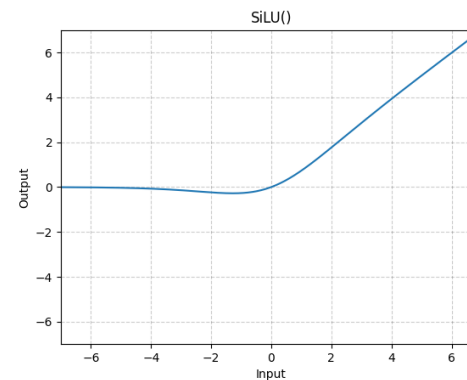
Approximation:

$$GELU(x) \approx 0.5x * \left(1 + \text{Tanh} \left(\sqrt{\frac{2}{\pi}} * (x + 0.044715x^3) \right) \right)$$



- **SiLU**: Sigmoid Linear Unit (also known as Swish)

$SiLU(x) = x * \sigma(x)$ where $\sigma(x)$ is the sigmoid



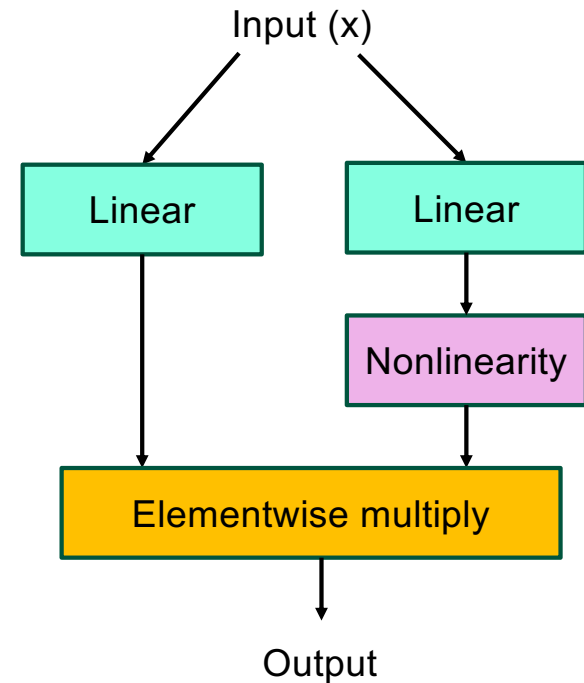
More complex FFN

- GLU (Gated Linear Unit):

$$\begin{aligned} \text{GLU}(x, W, V, b, c) \\ = (Wx + b) \odot \sigma(Vx + c) \end{aligned}$$

- SwiGLU:

$$\begin{aligned} \text{SwiGLU}(x, W, V, b, c) \\ = (Wx + b) \odot \text{Swish}(Vx + c) \end{aligned}$$



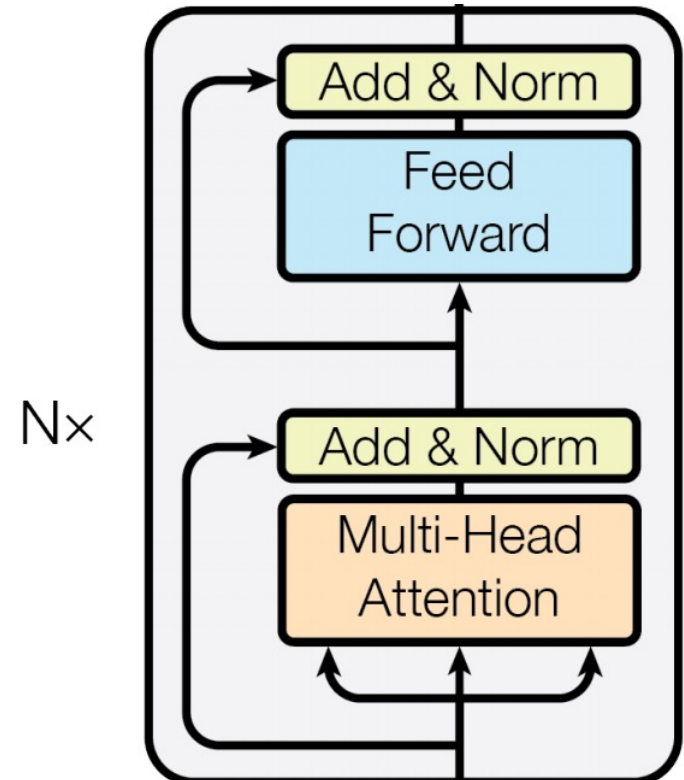
Y. Dauphin et al. [Language modeling with gated convolutional networks](#). ICML 2017

N. Shazeer. [GLU variants improve transformer](#). arXiv 2020

https://en.wikipedia.org/wiki/Gating_mechanism

More complex FFN

- Original transformer:
two linear layers with ReLUs in between, applied independently to each vector
- More advanced FFN: SwiGLU followed by linear transformation – three weight matrices in total



Layer normalization

- **Add & Norm:** residual connection followed by *layer normalization*
 - Compute mean and std. dev. across the dimensions of a single vector; use them to normalize that vector; then apply learned per-dimension scaling and shifting

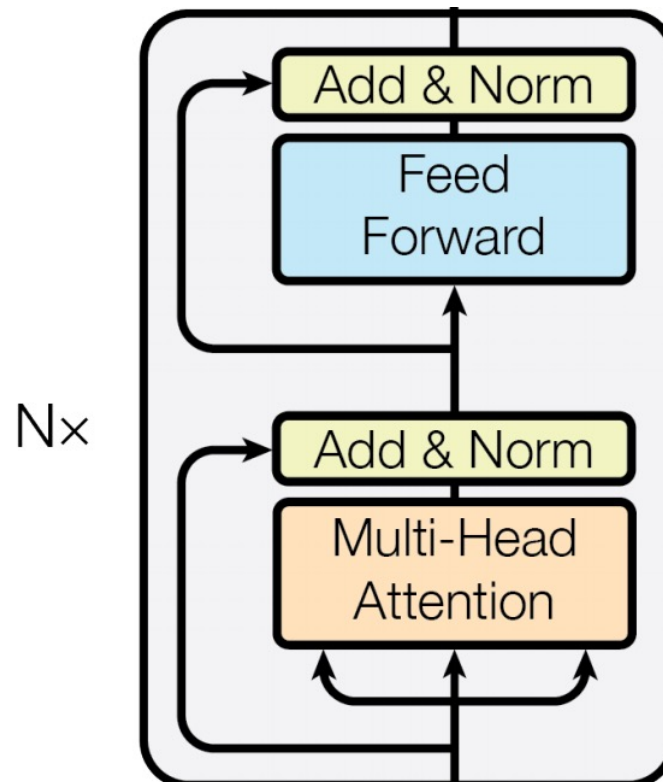
Given h_1, \dots, h_N (D dimensions)
scale: γ (D dimensions)
shift: β (D dimensions)

$$\mu_i = (\sum_j h_{ij}) / D \quad (\text{scalar})$$

$$\sigma_i = \sqrt{\sum_j (h_{ij} - \mu_i)^2 / D} \quad (\text{scalar})$$

$$z_i = (h_i - \mu_i) / \sigma_i$$

$$y_i = \gamma \odot z_i + \beta$$



Transformer normalization variants

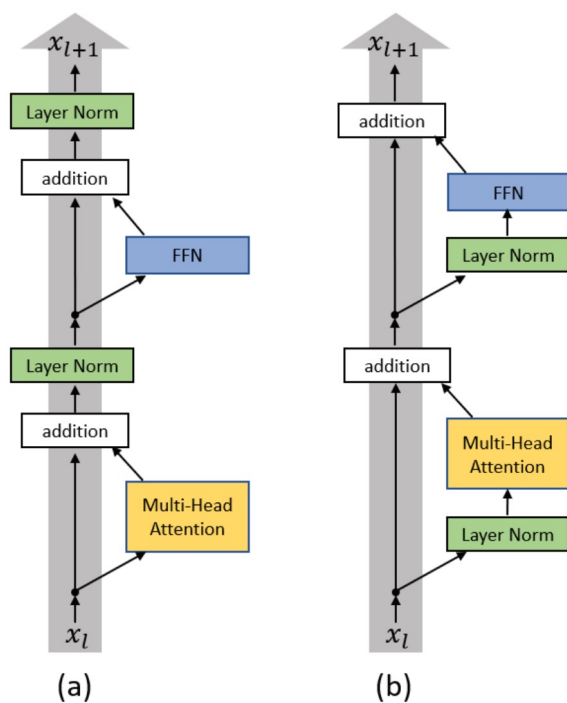


Figure 1. (a) Post-LN Transformer layer; (b) Pre-LN Transformer layer.

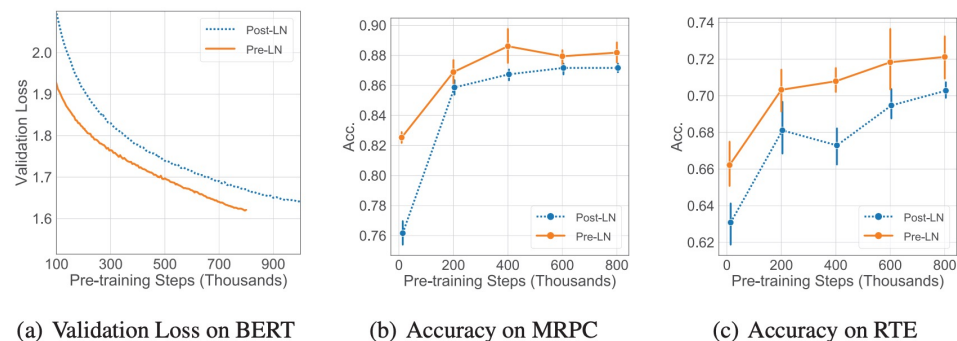


Figure 5. Performances of the models on unsupervised pre-training (BERT) and downstream tasks

Transformers without normalization

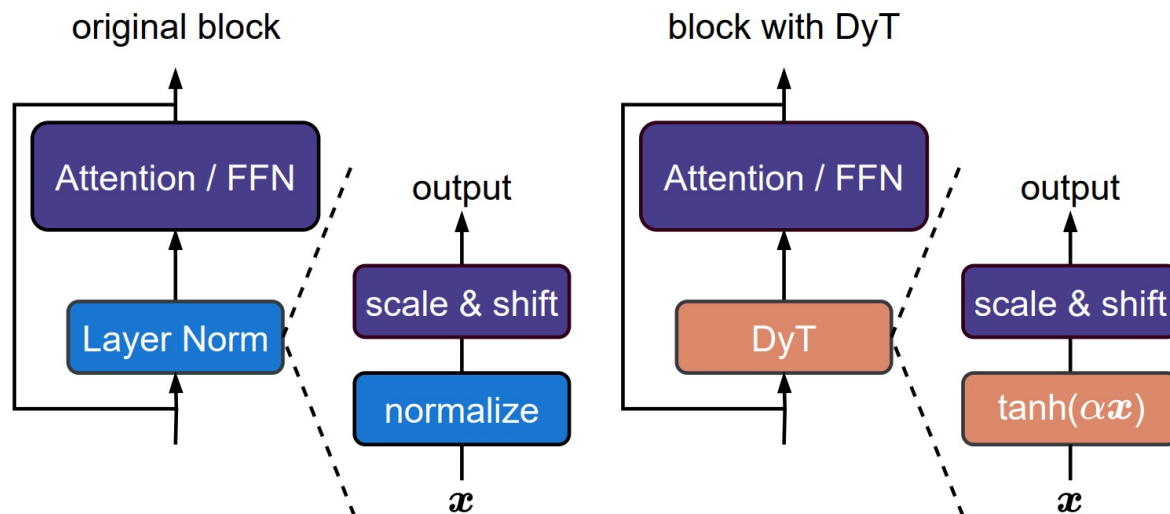
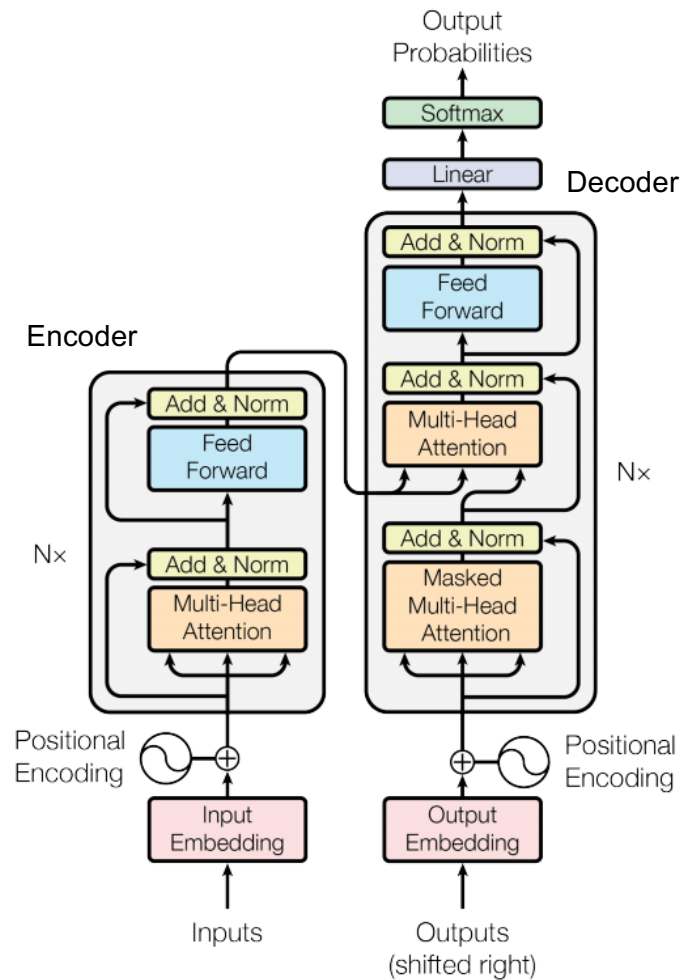


Figure 1. *Left*: original Transformer block. *Right*: block with our proposed Dynamic Tanh (DyT) layer. DyT is a straightforward replacement for commonly used Layer Norm [5] (in some cases RMSNorm [89]) layers. Transformers with DyT match or exceed the performance of their normalized counterparts.

Transformer architecture: Zooming back out



A. Vaswani et al., [Attention is all you need](#), NeurIPS 2017

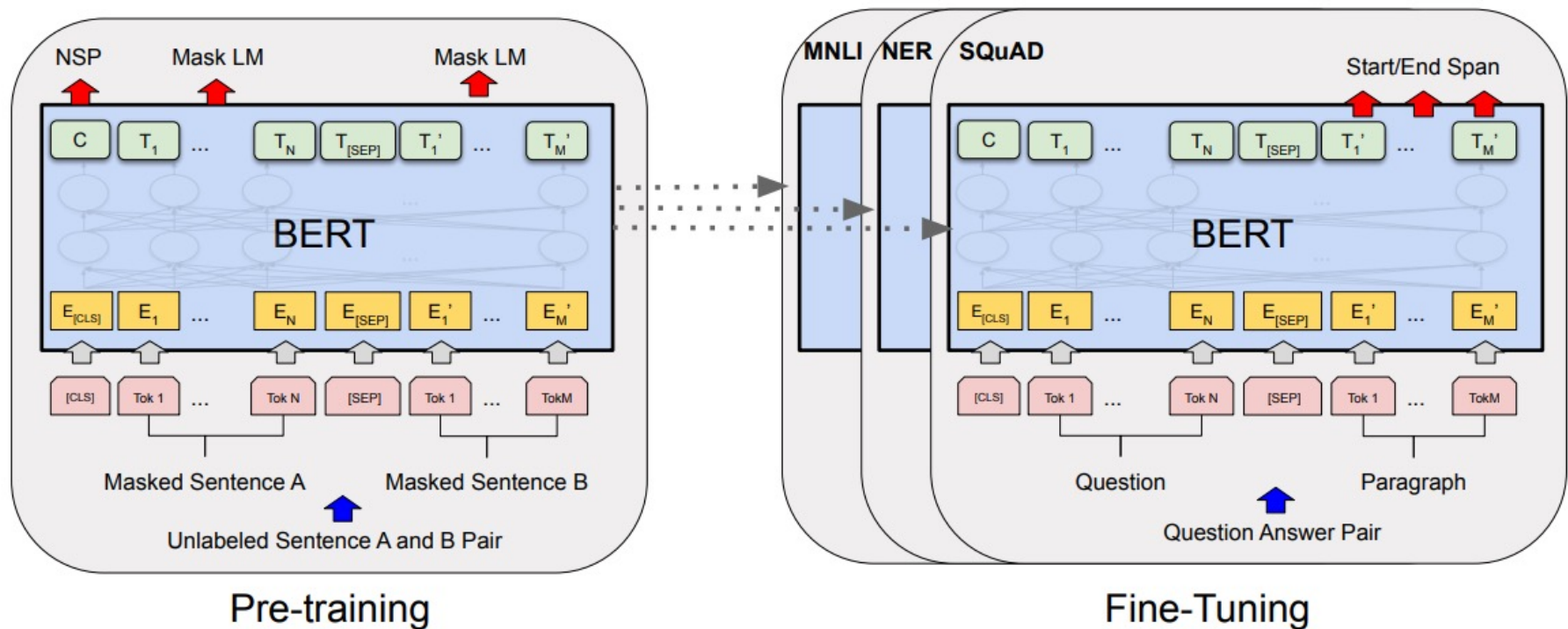
Outline

- Attention models
- The transformer block in detail
- Early architectures for language modeling

Self-supervised language modeling with transformers

1. Download A LOT of text from the internet
2. Train a giant transformer using a suitable pretext task
3. Fine-tune the transformer on desired NLP task (optional)

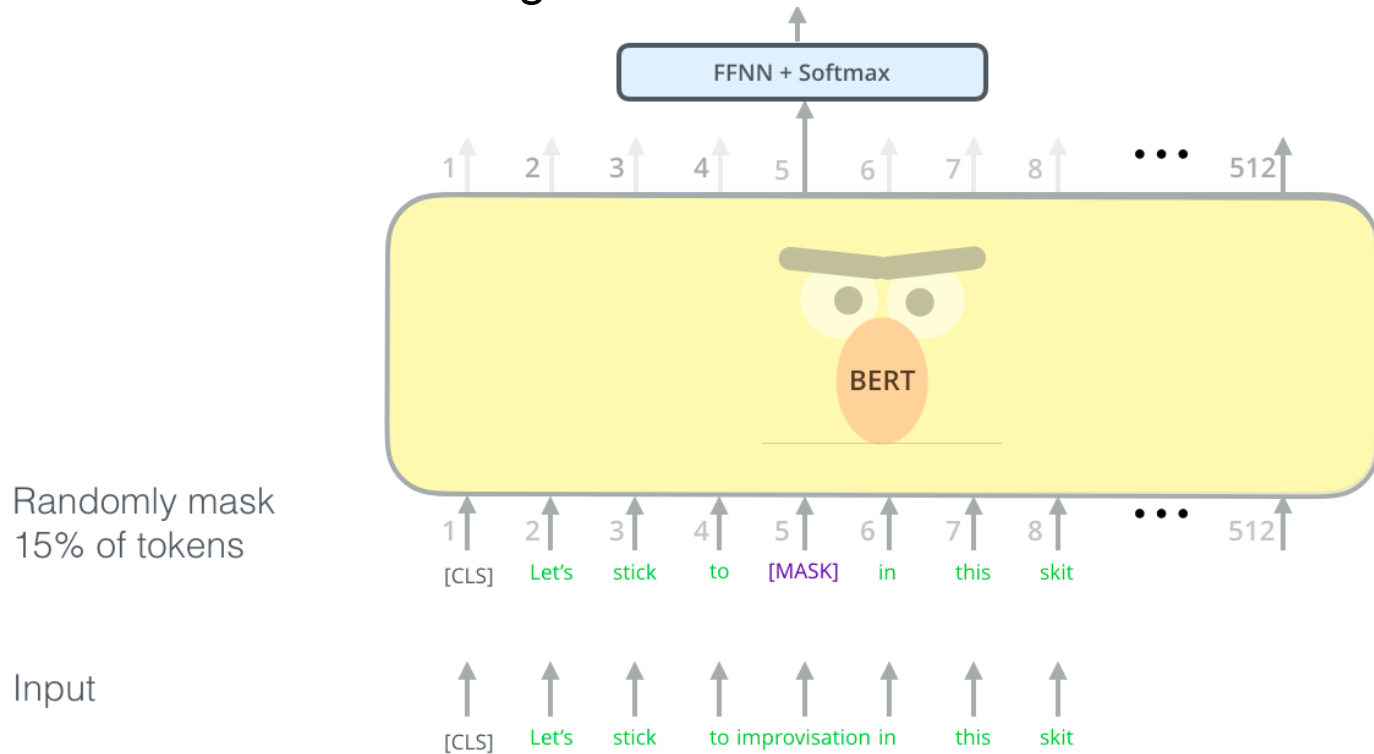
Bidirectional encoder representations from transformers (BERT)



J. Devlin et al. [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](#). EMNLP 2018

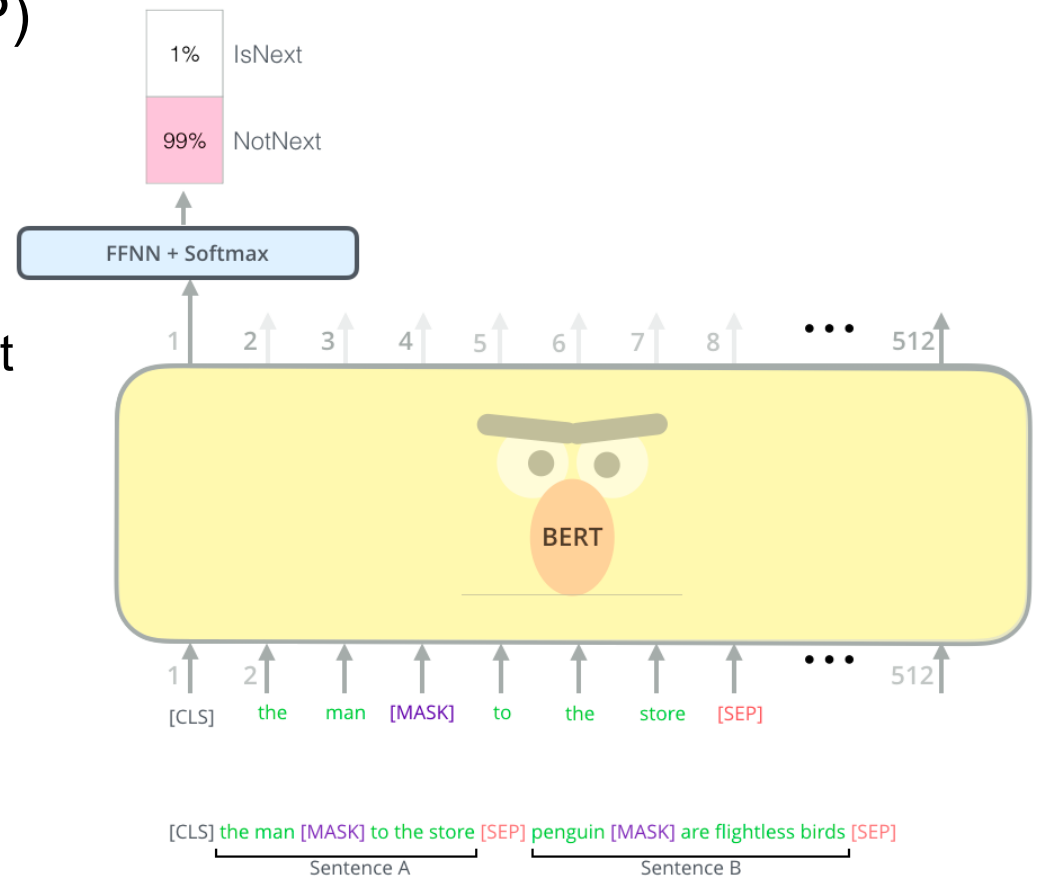
BERT: Pretext tasks

- Masked language model (MLM)
 - Randomly mask 15% of tokens in input sentences, goal is to reconstruct them using bidirectional context



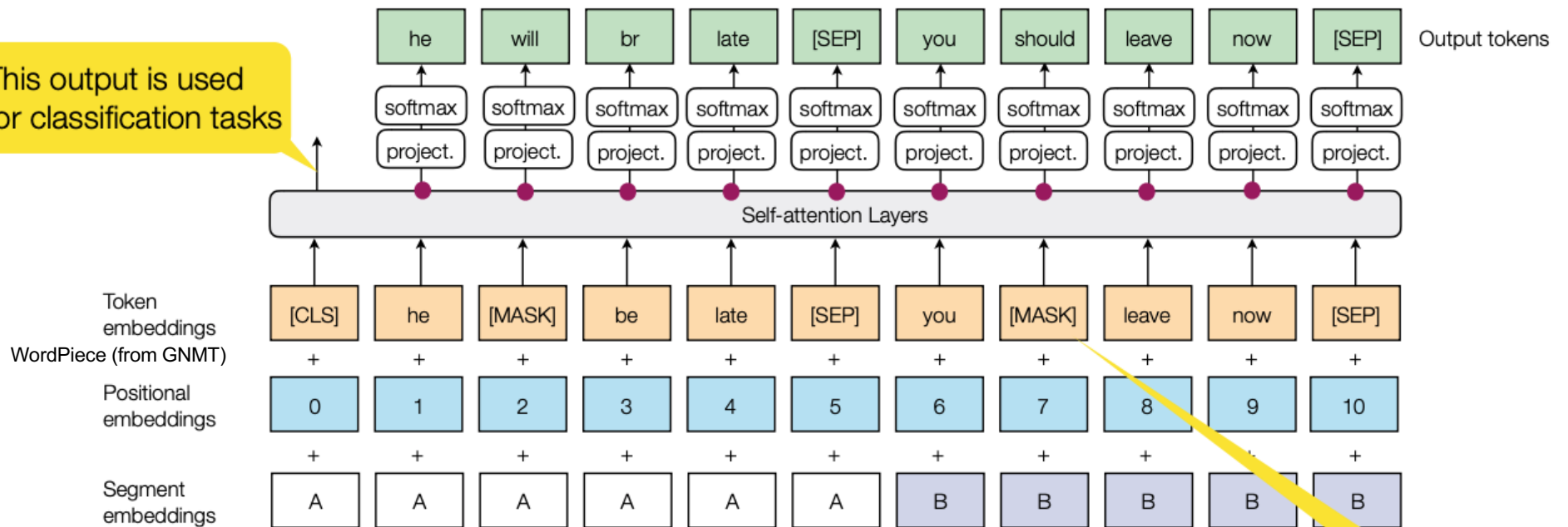
BERT: Pretext tasks

- Next sentence prediction (NSP)
 - Useful for Question Answering and Natural Language Inference tasks
 - In the training data, 50% of the time B is the actual sentence that follows A (labeled as IsNext), and 50% of the time it is a random sentence (labeled as NotNext).



BERT: More detailed view

This output is used for classification tasks



Trained on Wikipedia (2.5B words) + BookCorpus (800M words)

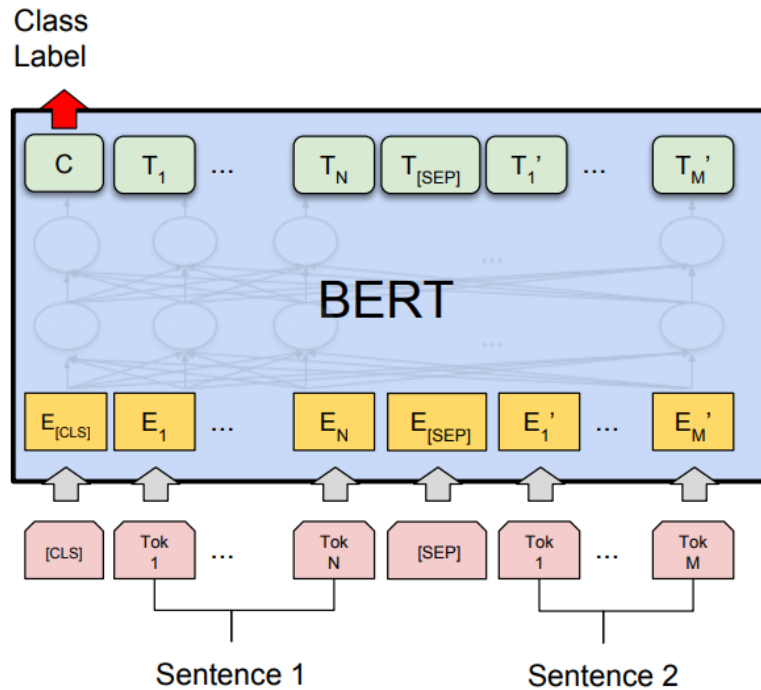
15% of tokens get masked

BERT: Evaluation

- General Language Understanding Evaluation (GLUE) benchmark (gluebenchmark.com)

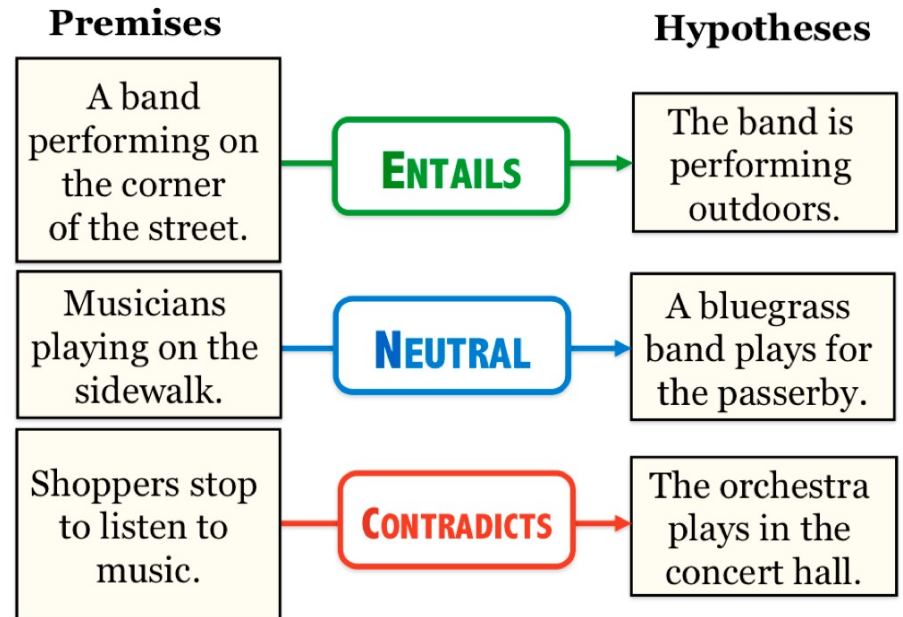
System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT _{BASE}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

BERT: Downstream tasks



(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG

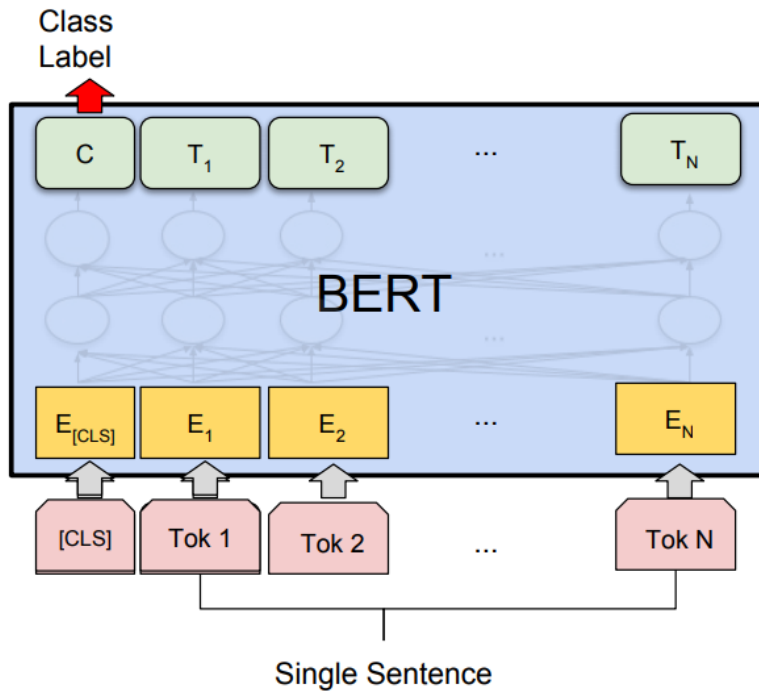
Textual entailment



Source: J. Hockenmaier

Entailment, textual equivalence and similarity

BERT: Downstream tasks



(b) Single Sentence Classification Tasks:
SST-2, CoLA

CoLa

Sentence: The wagon rumbled down the road.

Label: Acceptable

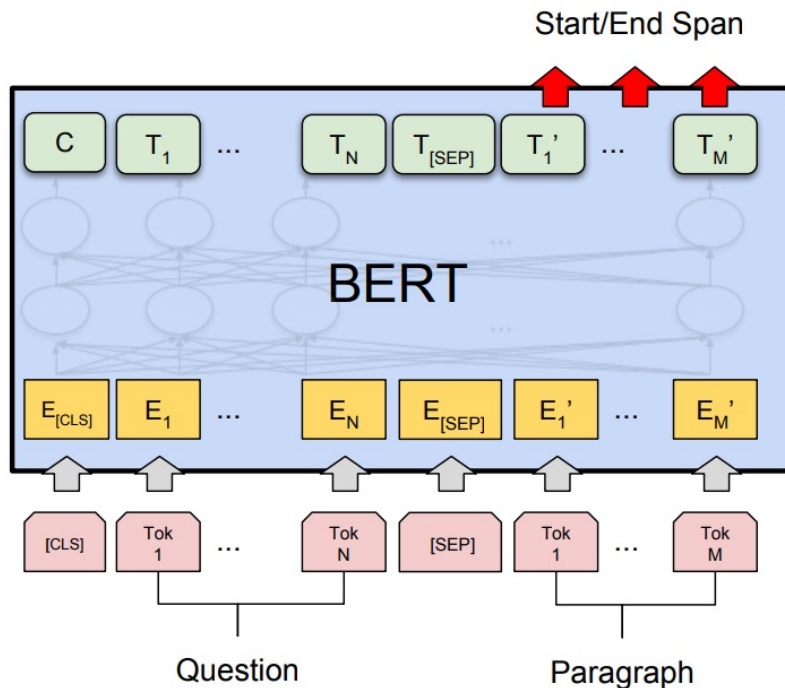
Sentence: The car honked down the road.

Label: Unacceptable

Sentiment classification, linguistic acceptability

[Image source](#)

BERT: Downstream tasks



(c) Question Answering Tasks:
SQuAD v1.1

Find span in paragraph that contains the answer

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called "showers".

What causes precipitation to fall?

gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?

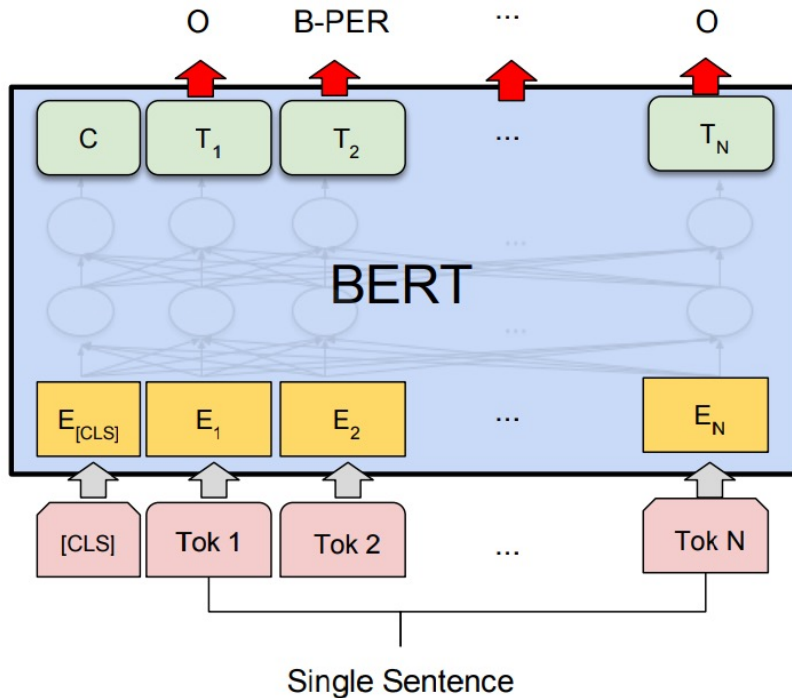
graupel

Where do water droplets collide with ice crystals to form precipitation?

within a cloud

Source: [SQuAD v1.1 paper](#)

BERT: Downstream tasks



When **Sebastian Thrun** PERSON started at **Google** ORG in **2007** DATE, few people outside of the company took him seriously. "I can tell you very senior CEOs of major **American** NORP car companies would shake my hand and turn away because I wasn't worth talking to," said **Thrun** PERSON, now the co-founder and CEO of online higher education startup Udacity, in an interview with **Recode** ORG **earlier this week** DATE.

A little **less than a decade later** DATE, dozens of self-driving startups have cropped up while automakers around the world clamor, wallet in hand, to secure their place in the fast-moving world of fully automated transportation.

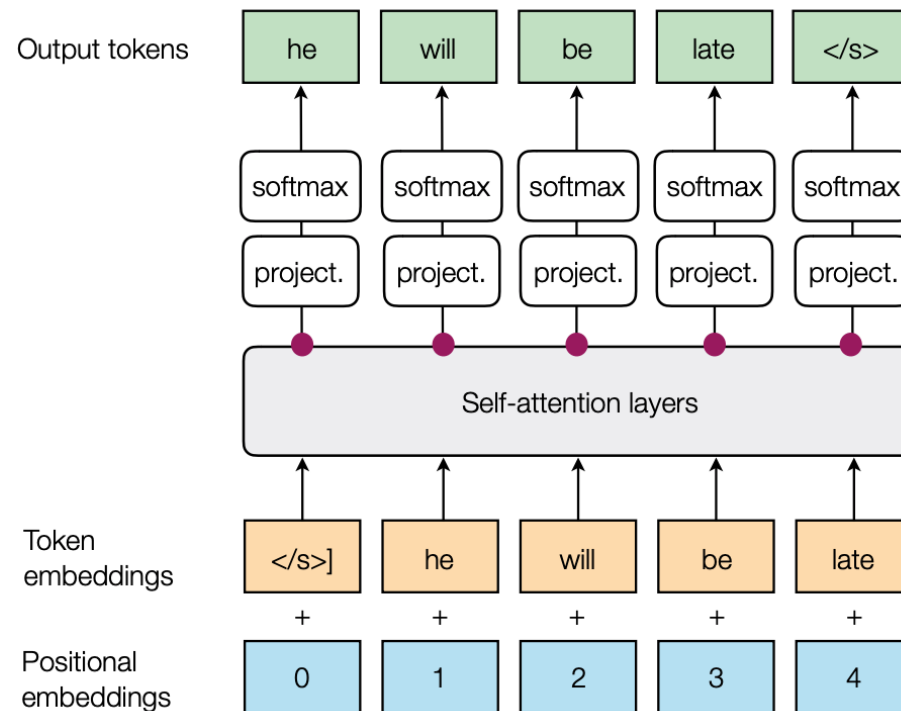
[Image source](#)

(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

Named entity recognition

OpenAI GPT (Generative Pre-Training)

- Pre-training task: next token prediction (causal language modeling)



A. Radford et al. [Improving language understanding by generative pre-training](#). 2018

[Image source](#)