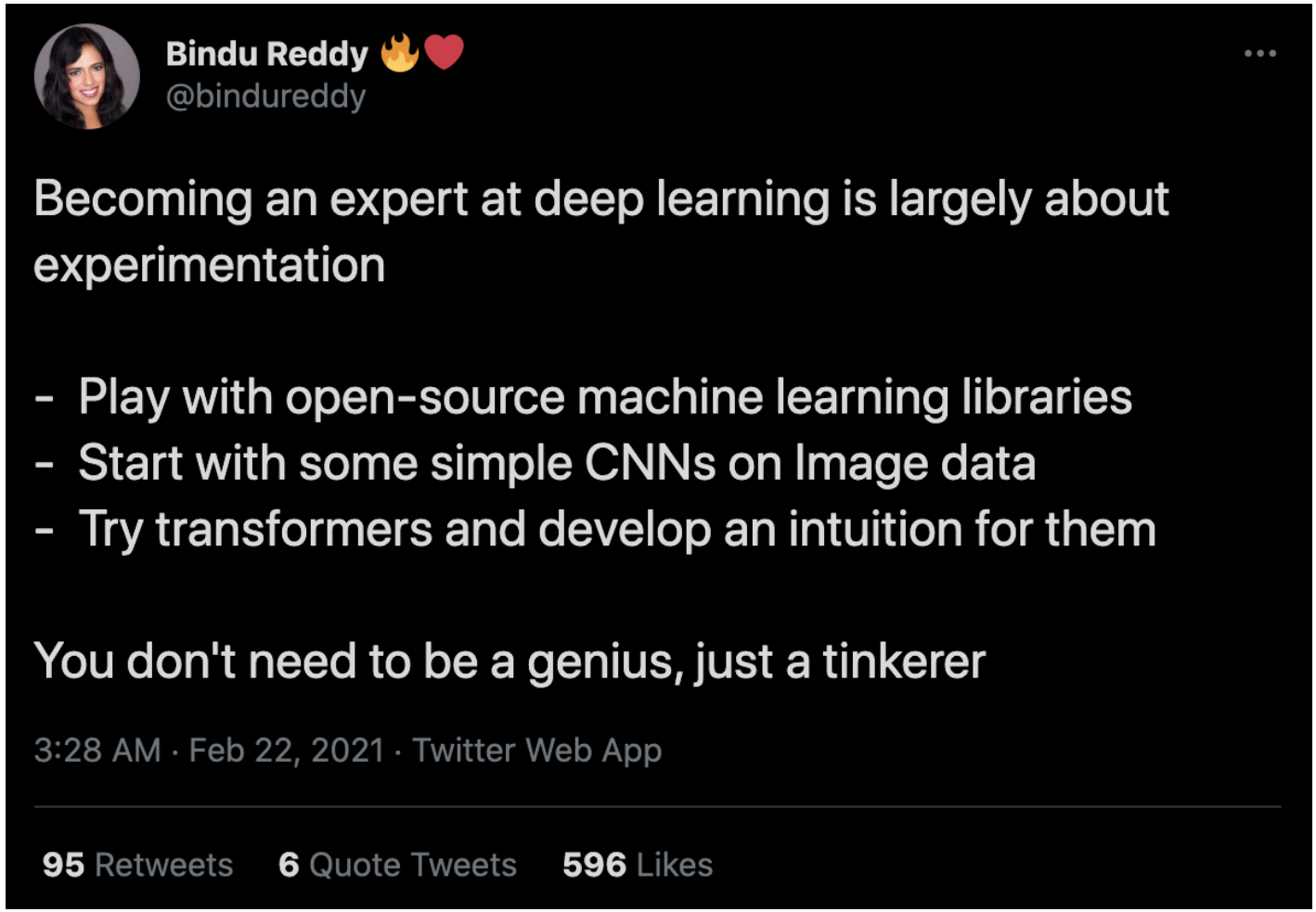
Attention and Transformers

Yang You

HPC-AI Lab of NUS Computer Science

ai.comp.nus.edu.sg

Transformer is transforming everything in deep learning



- All the attention of deep learning is now on Attention!

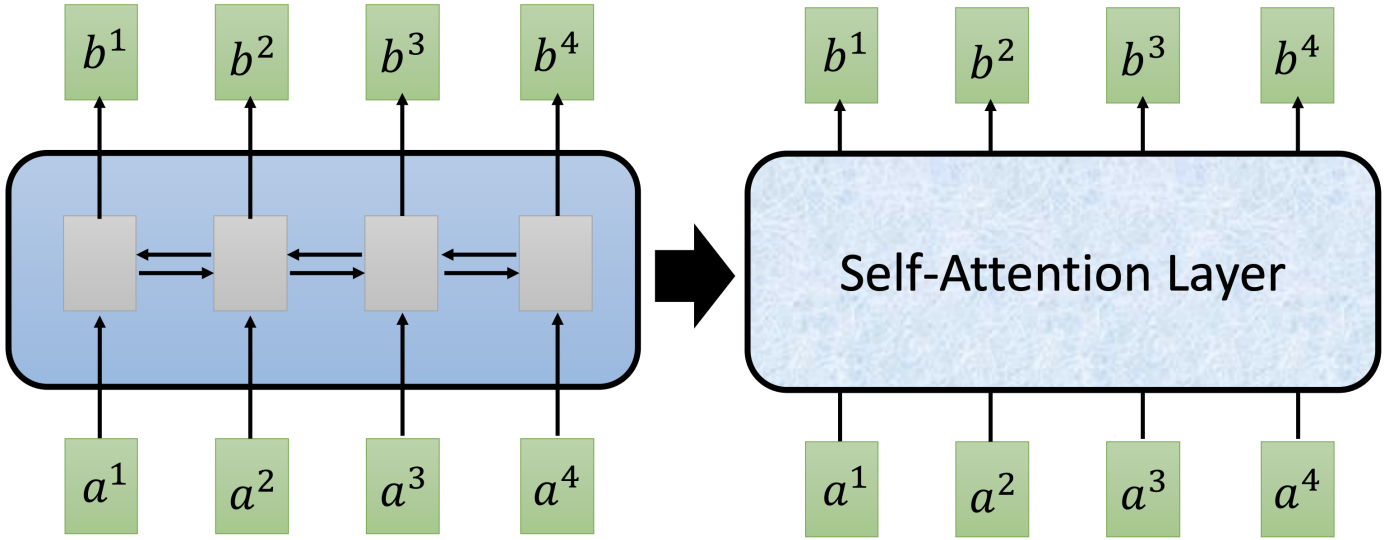
Why do we want to study Transformers?

- Transformers have become the default model-of-choice in NLP
  - The dominating approach is to pre-train on a huge dataset and then fine-tune on a smaller task-specific dataset
  - Belongs to self-attention-based architectures (Vaswani et al., 2017)
  - It is computationally efficient and scalable
    - With the models and datasets growing, there is still no sign of saturating performance
    - We now can train models of unprecedented size (>100B parameters)
    - e.g. BERT, GPT-3, PaLM, Switch-Transformers

Why do we want to study Transformers?

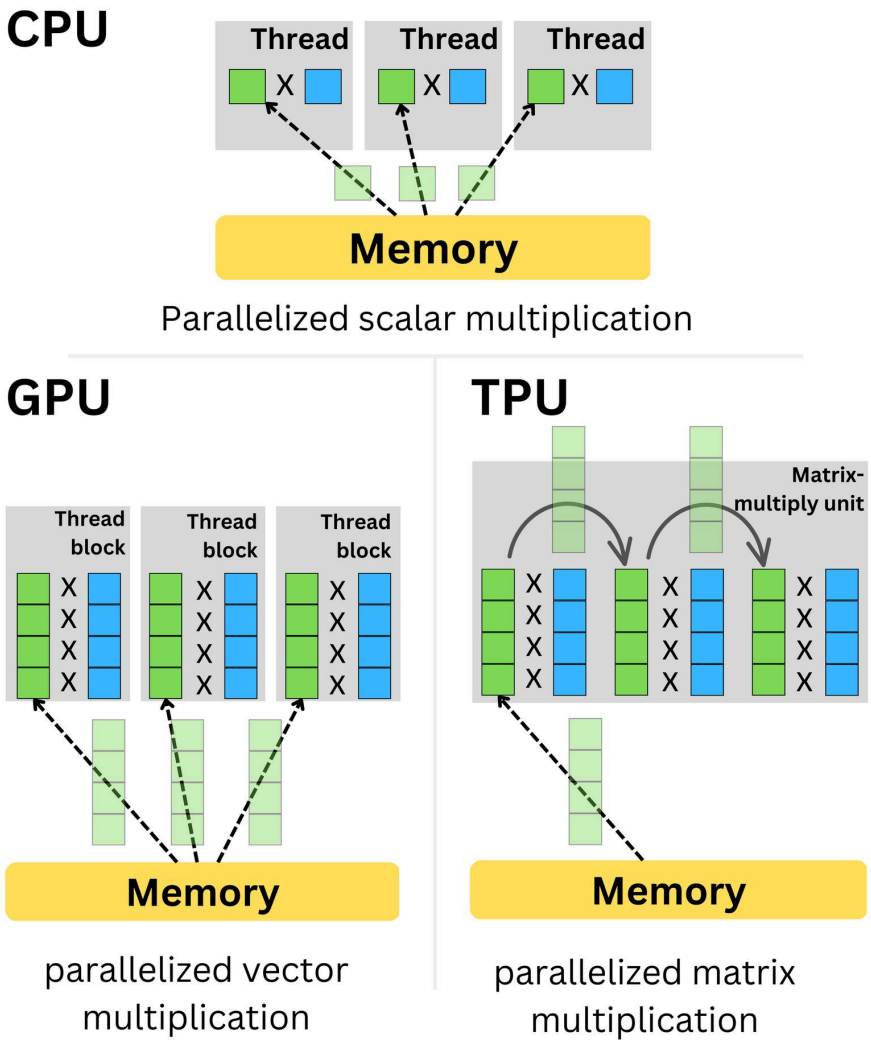
- Transformers are friendly to modern hardware like TPUs and GPUs
- Why?

Self-Attention



- We can replace anything processed by RNN with Attention
  - In RNN, b^1, b^2, b^3, b^4 will be computed sequentially
  - In Attention, b^1, b^2, b^3, b^4 can be computed in parallel

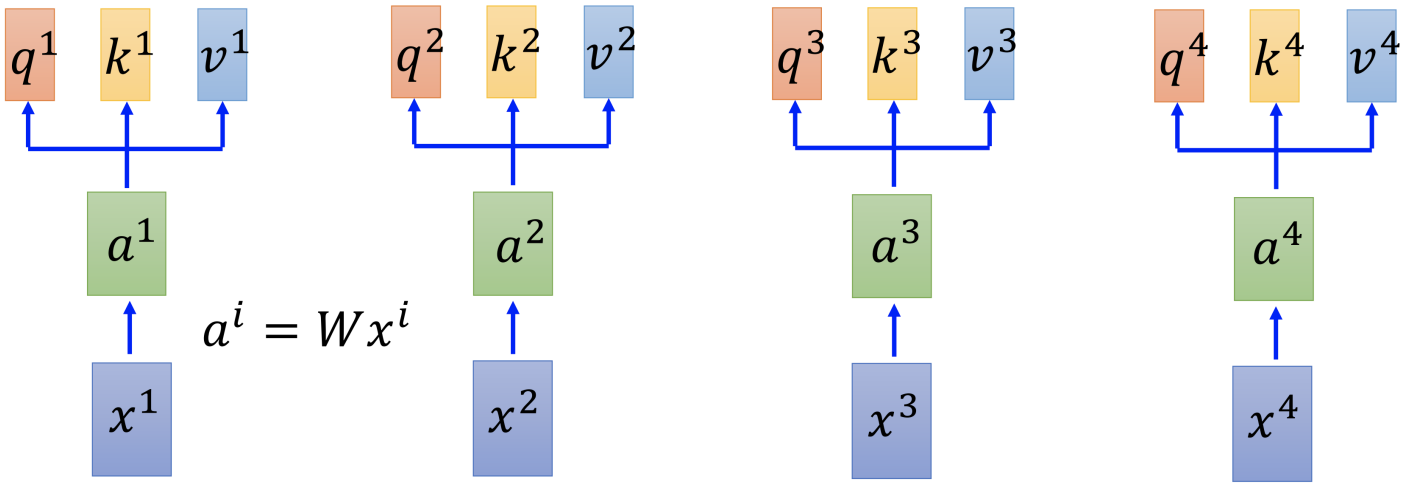
CPU vs GPU vs TPU



Why do we want to study Transformers?

- Transformers are friendly to modern hardware like TPUs and GPUs
  - We need a high parallelism to make full use of the computing units in TPUs and GPUs (e.g. thousands of threads)

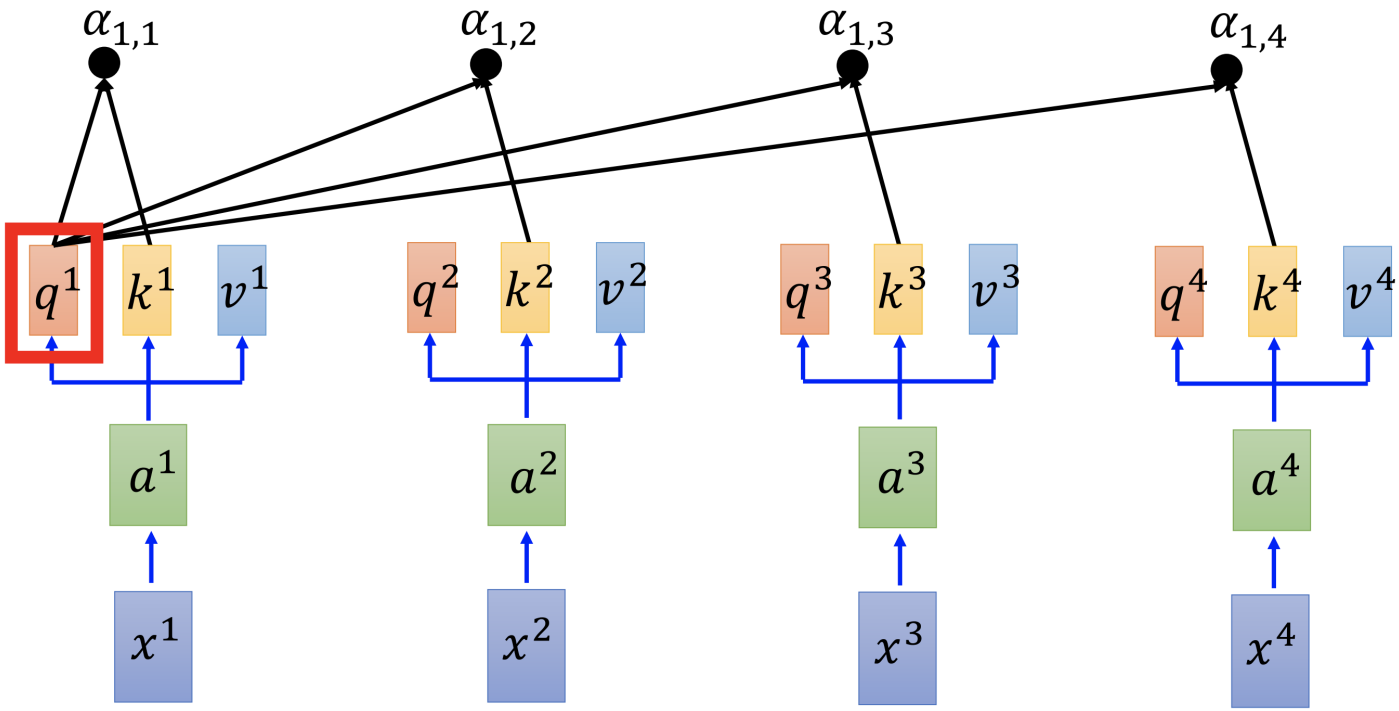
Self-Attention Architecture



- q: query (to match others)
  - $q^i = W^q a^i$
- k: key (to be matched)
  - $k^i = W^k a^i$
- v: value (to be extracted)
  - $v^i = W^v a^i$

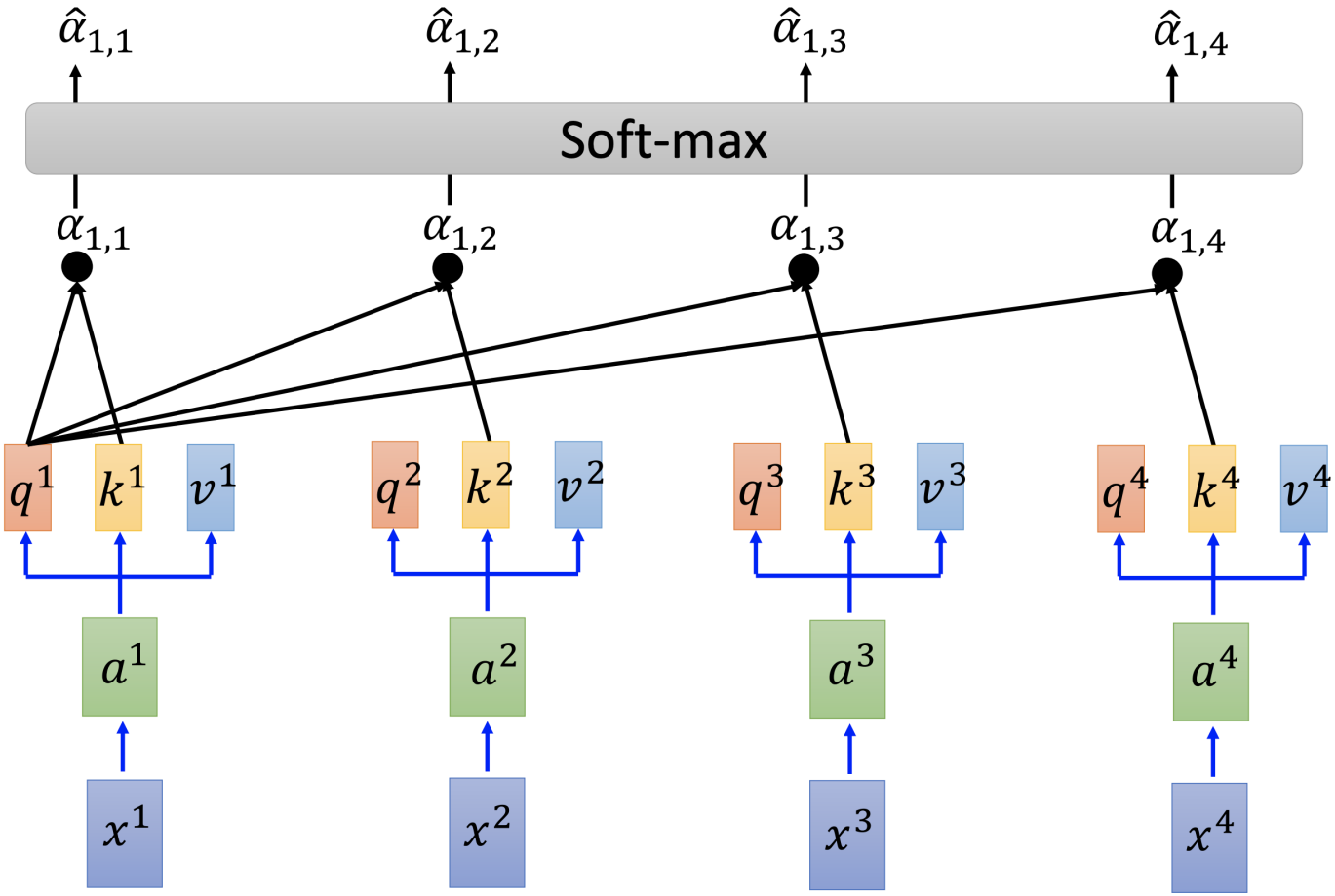
Computation of Self Attention

Scaled Dot-Product Attention: $\alpha_{1,i} = \underbrace{q^1 \cdot k^i}_{\text{dot product}} / \sqrt{d}$



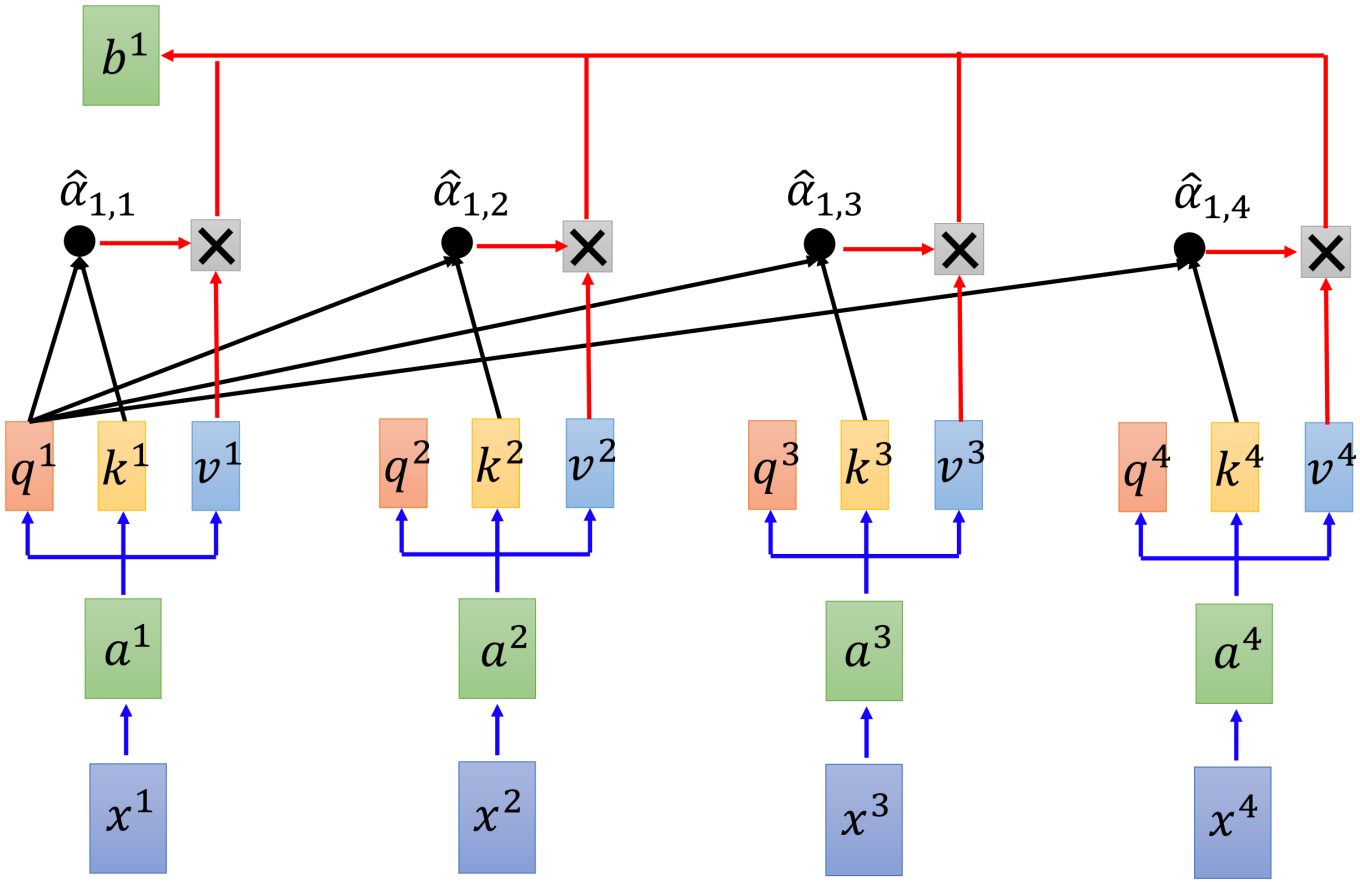
- d: the length of vector q or vector k
- We use each query q to match each key k
- $\alpha_{i,j}$: the strength of the relationship between x^i and x^j

Just doing a softmax



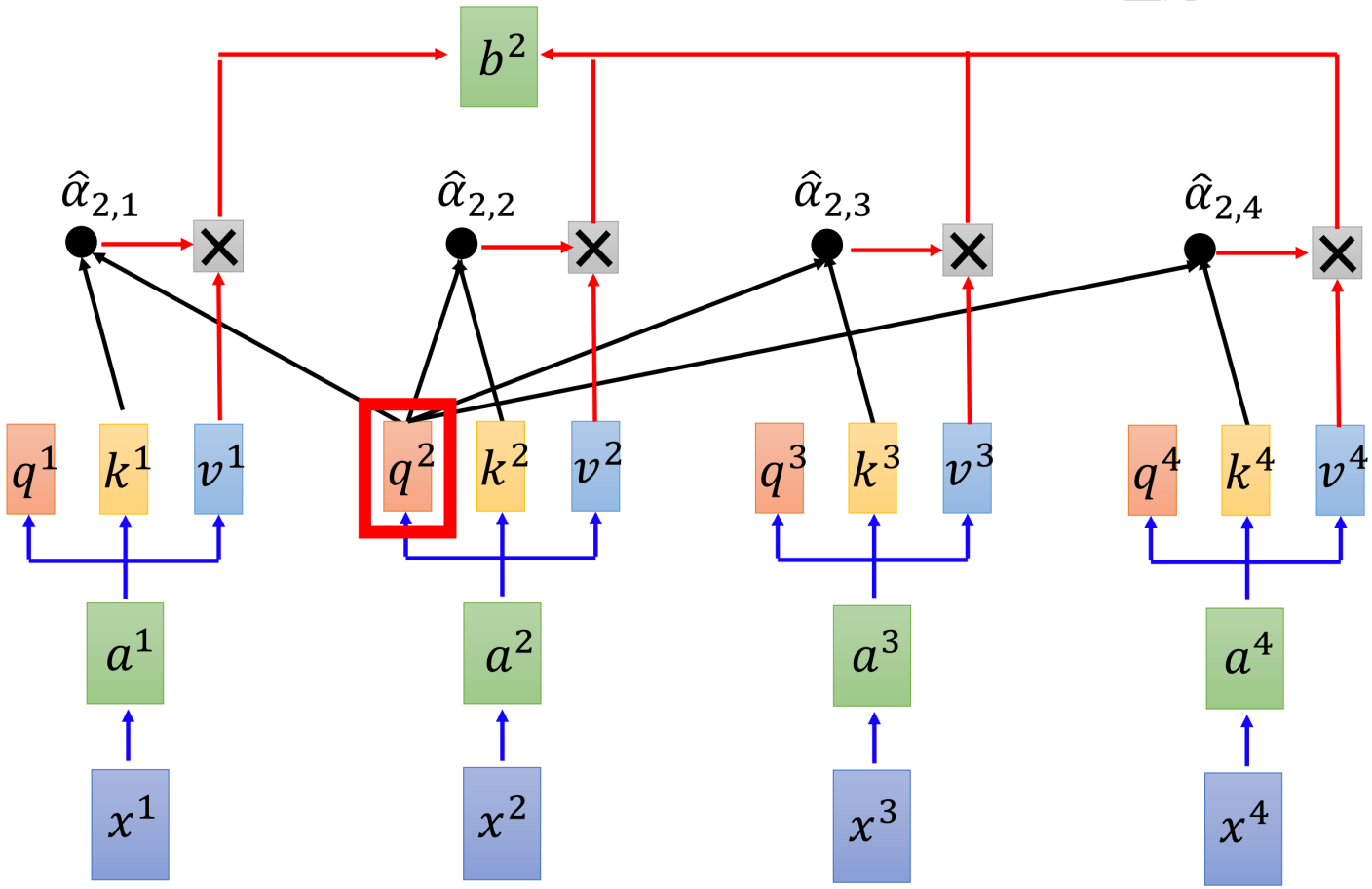
- $\hat{\alpha}_{1,i} = \frac{exp(\alpha_{1,i})}{\sum_j exp(\alpha_{1,j})}$: the strength of the relationship between x^1 and x^j
- $\hat{\alpha}_{i,j} \times v^j$: the attention of x^i on x^j
  - stronger relationship, stronger attention

Computing the first output



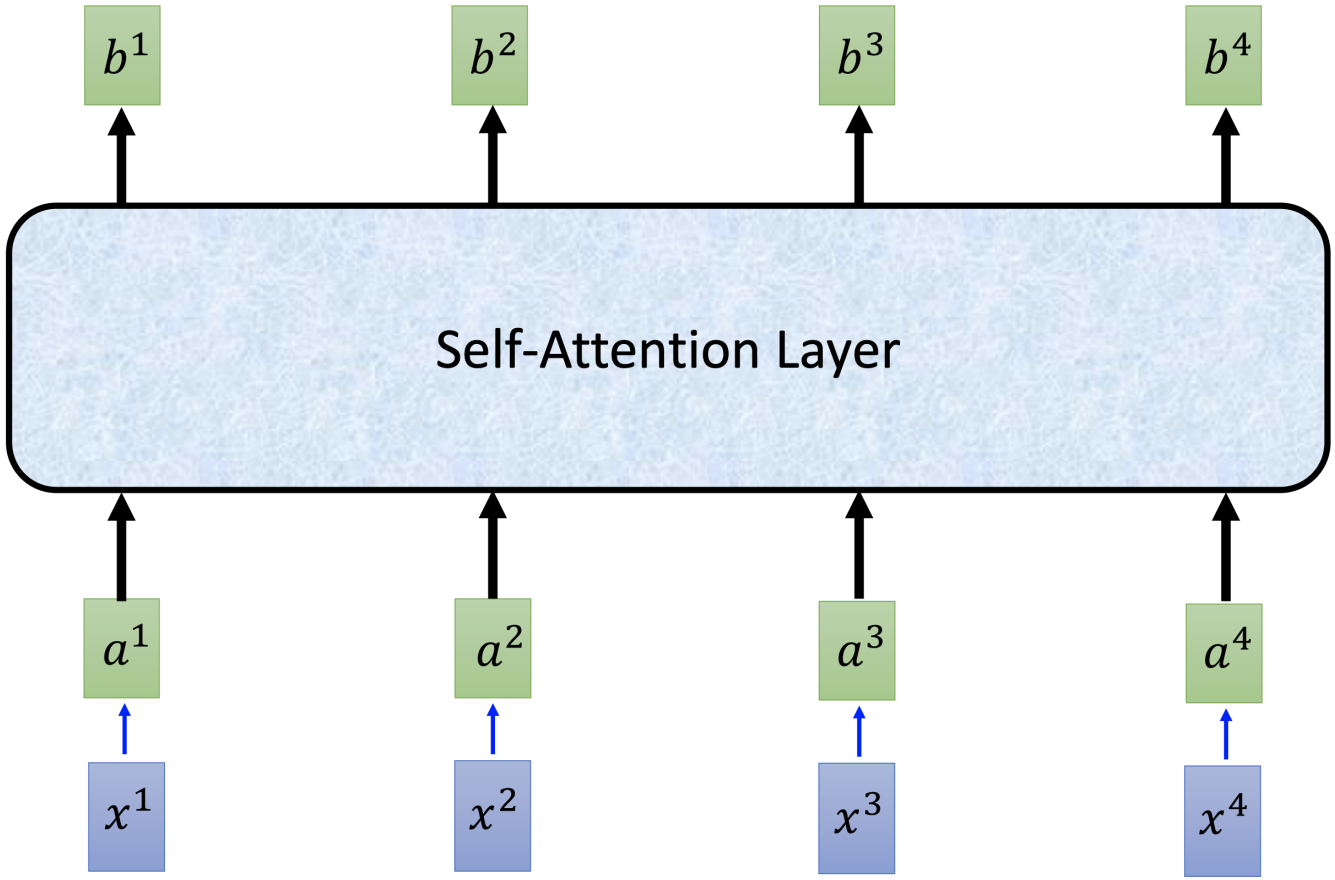
- $b^1 = \sum_i \hat{\alpha}_{1,i} \times v^i$: all the attentions of x^1 on other inputs and itself

Computing the second output



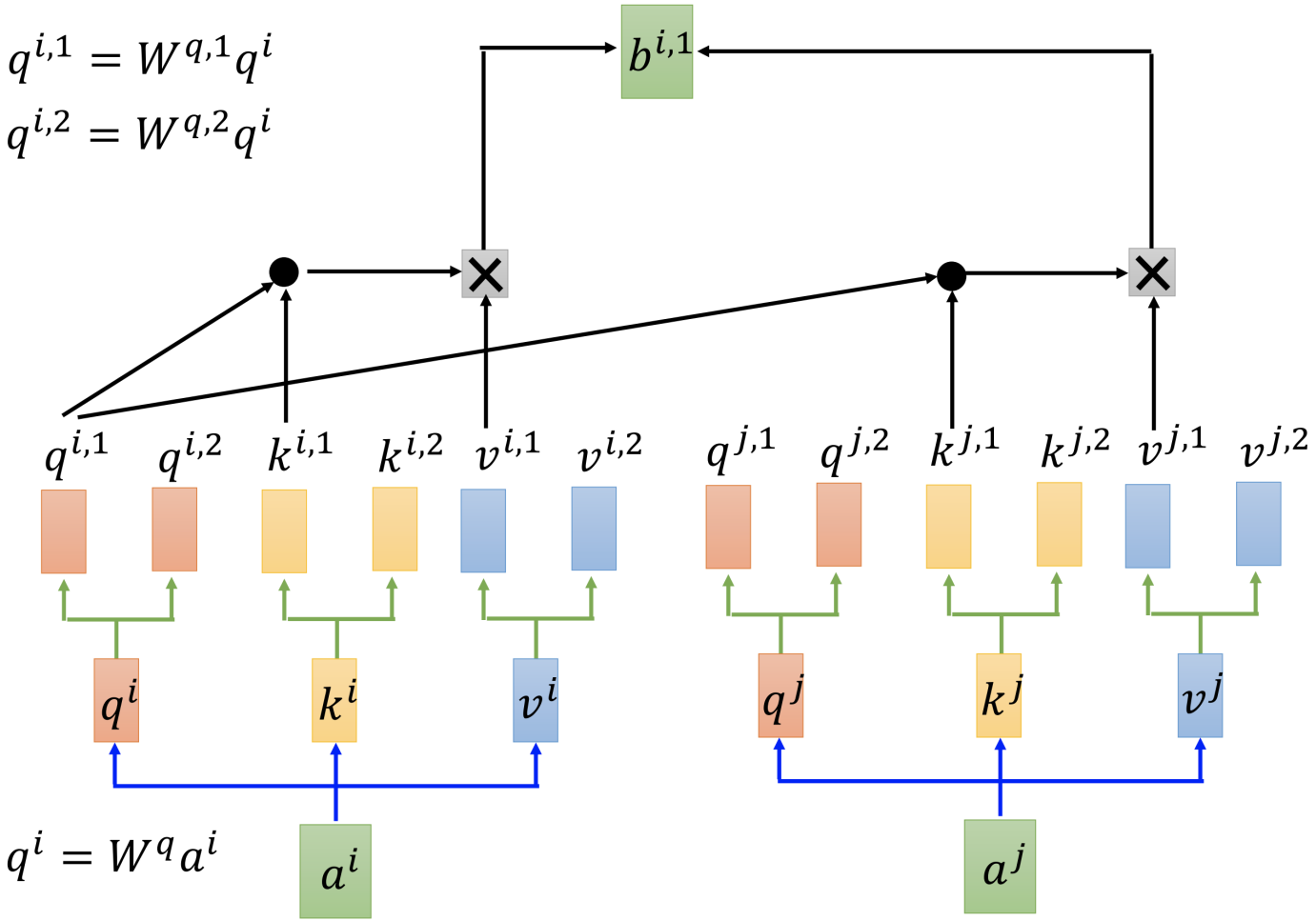
- $b^2 = \sum_i \hat{\alpha}_{2,i} \times v^i$: all the attentions of x^2 on other inputs and itself

Overview of Self-Attention



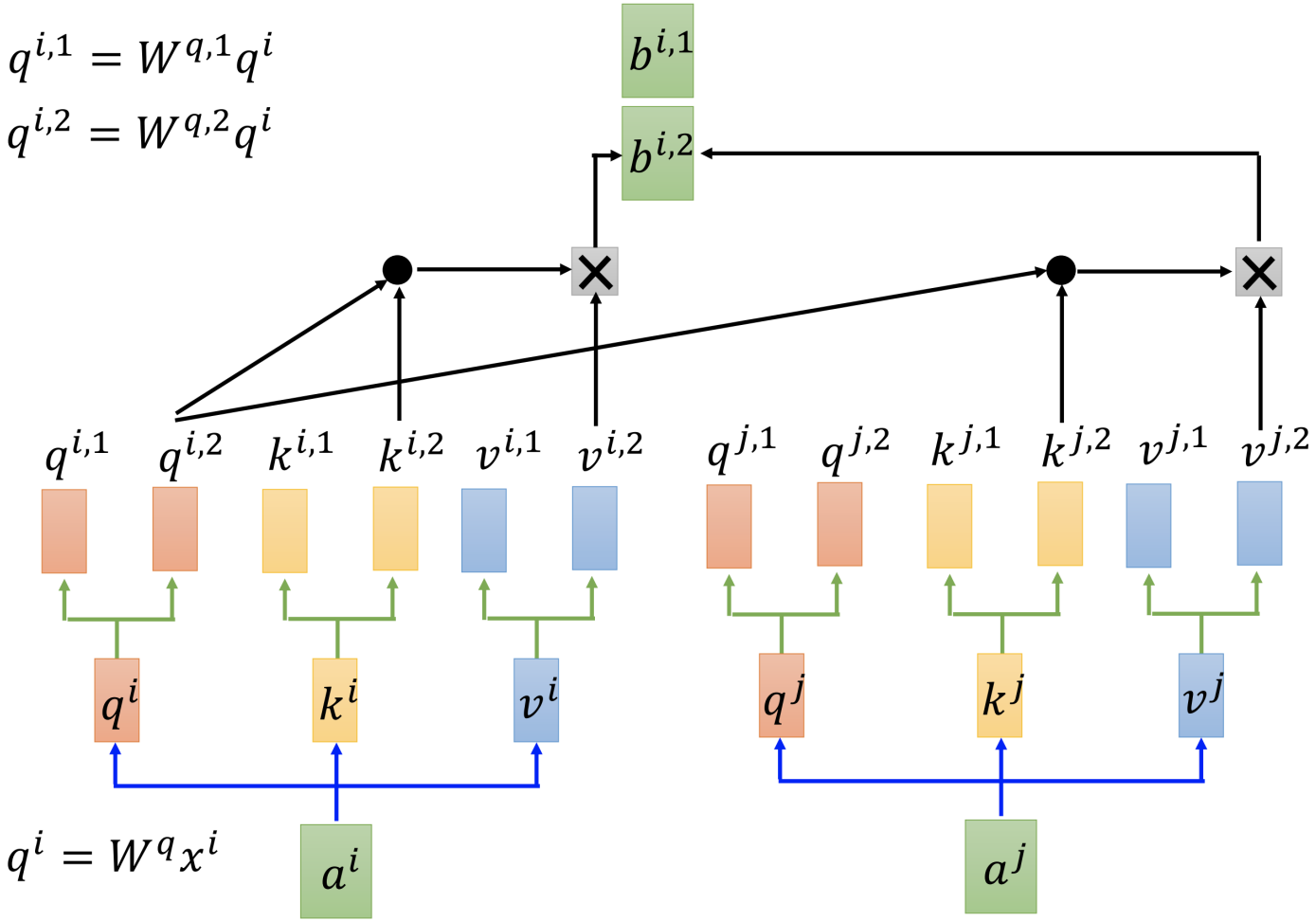
- b^1, b^2, b^3, b^4 can be computed in parallel (no data dependency)
- Idea: building a relationship between any 2 inputs (including itself)

Multi-Head Self-Attention



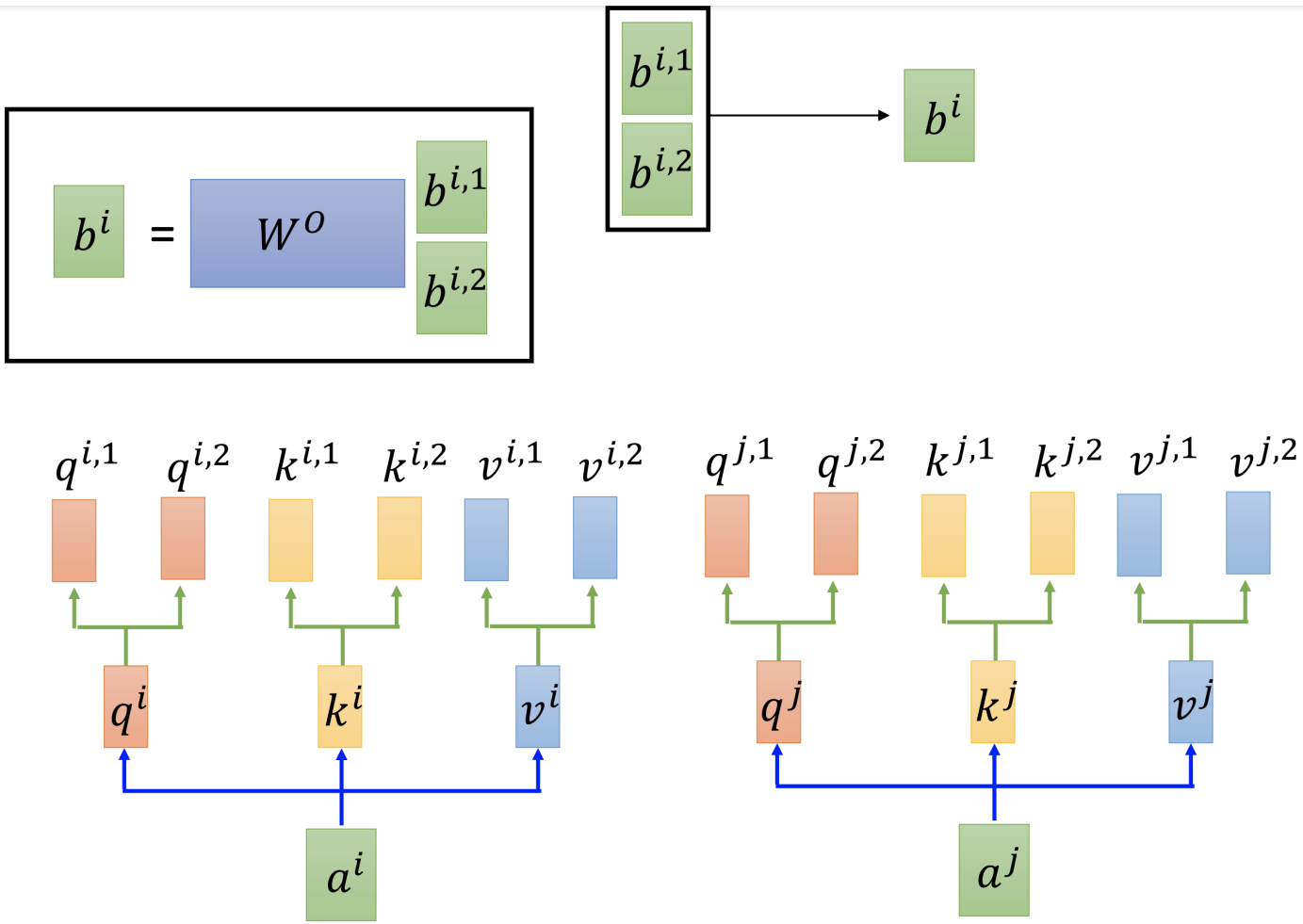
- Example of 2 heads
- They can only communicate within the same head group

Multi-Head Self-Attention



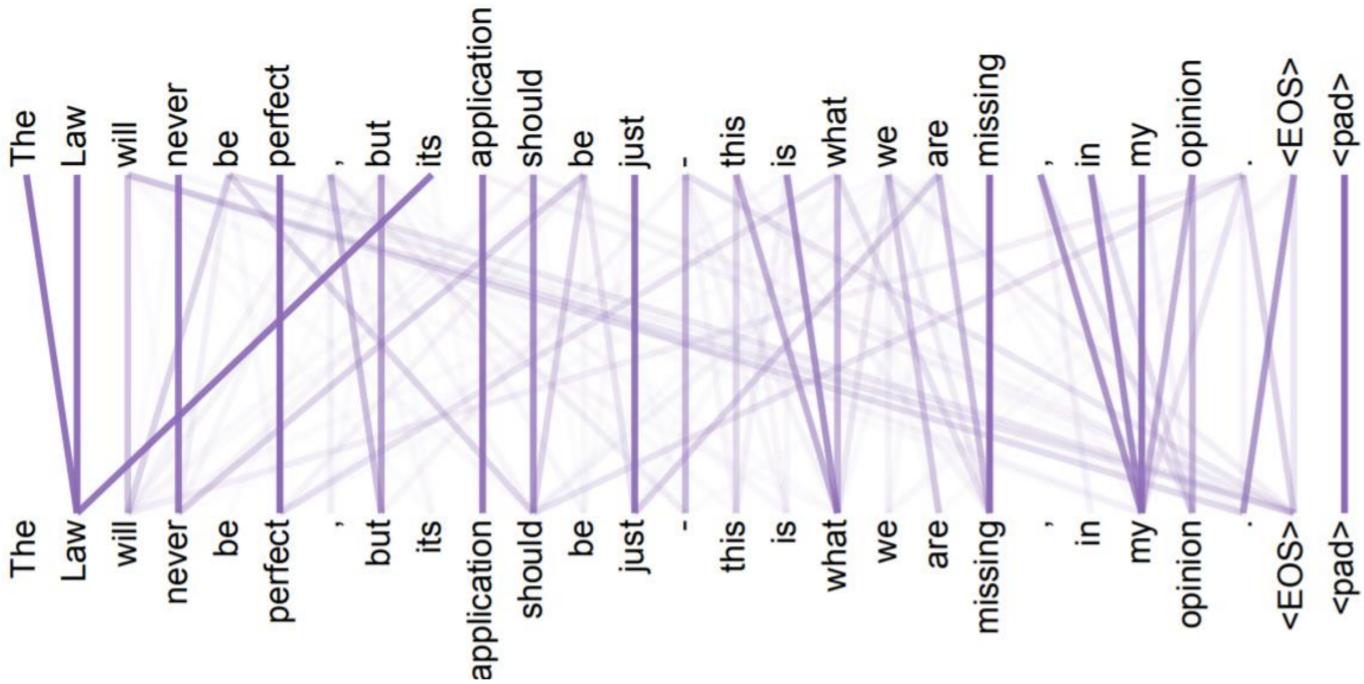
- Example of 2 heads
- They can only communicate within the same head group

Multi-Head Self-Attention

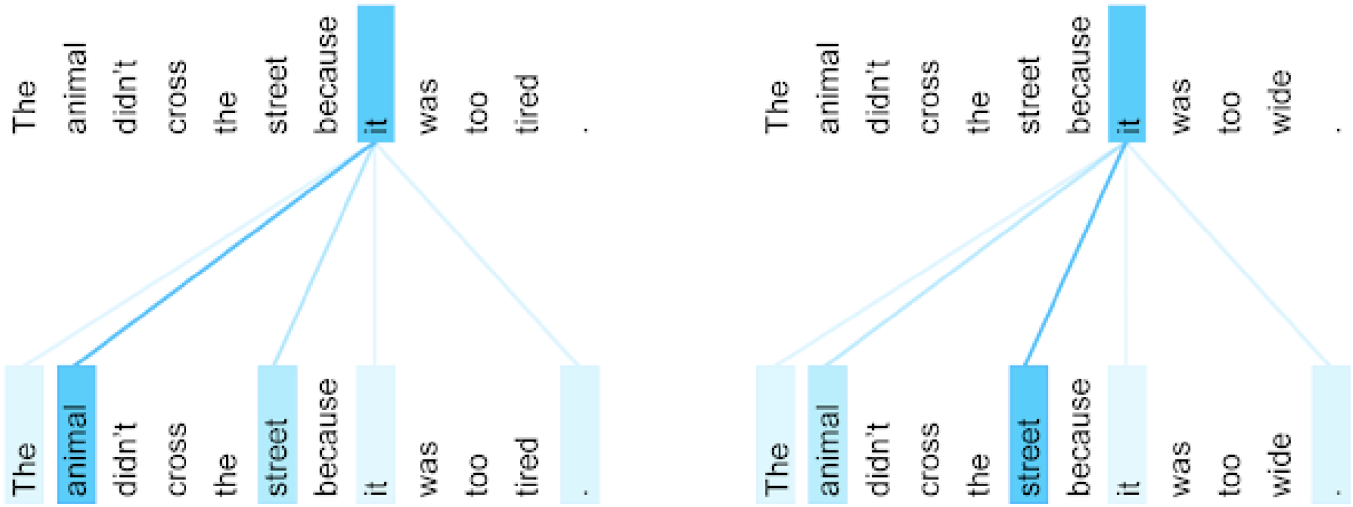


- We can merge the outputs the multiple heads into one output

Self-Attention Visualization

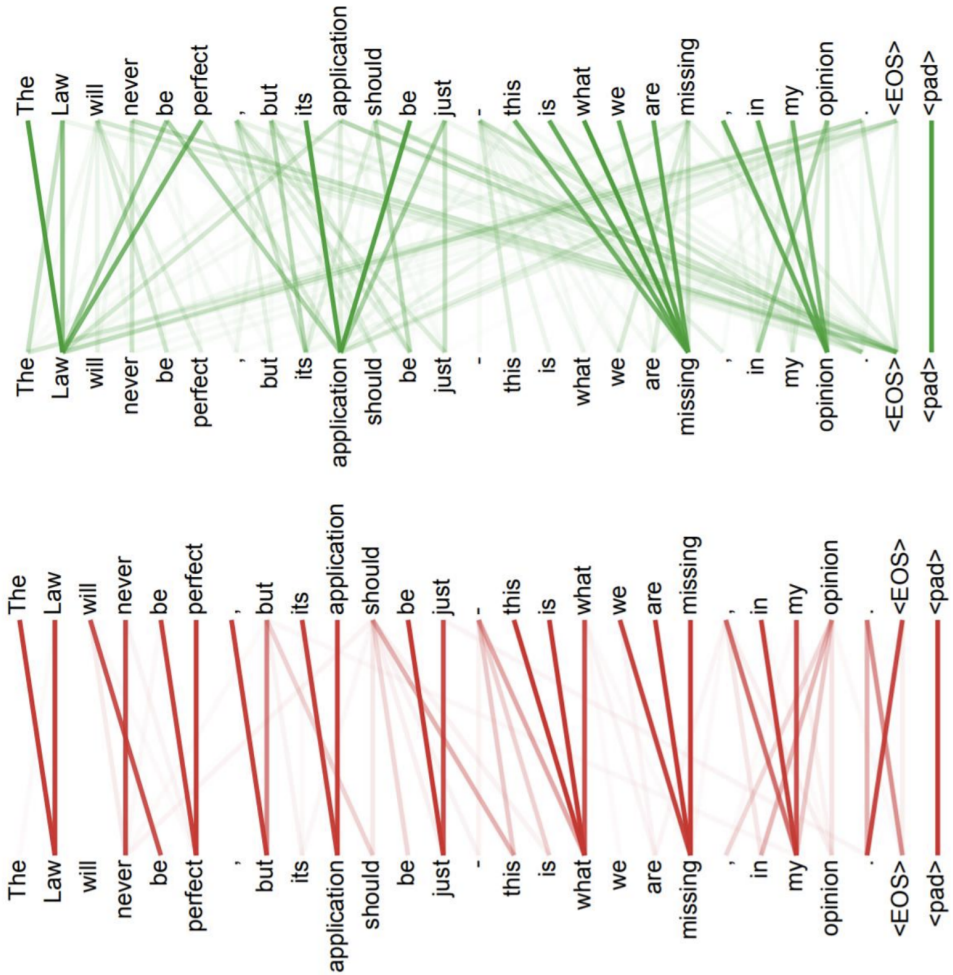


Self-Attention Visualization



- The encoder self-attention distribution for the word "it" from the 5th to the 6th layer of a Transformer trained on English to French translation (one of eight attention heads).

Multi-Head Self-Attention Visualization



- Some heads focus on local information
- Some heads focus on global information

What's next?

hardmaru @hardmaru · Jan 21

Transformers are the new LSTMs

23 61 648

hardmaru @hardmaru · Jan 21

This tweet didn't live up to its expectations:

We'll see in 46 years...

> **hardmaru** @hardmaru · May 15, 2017
>
> LSTMs are like the AK47 of neural nets. No matter how hard we try to replace it with something new, it will still be used 50 years from now.

7 17 252

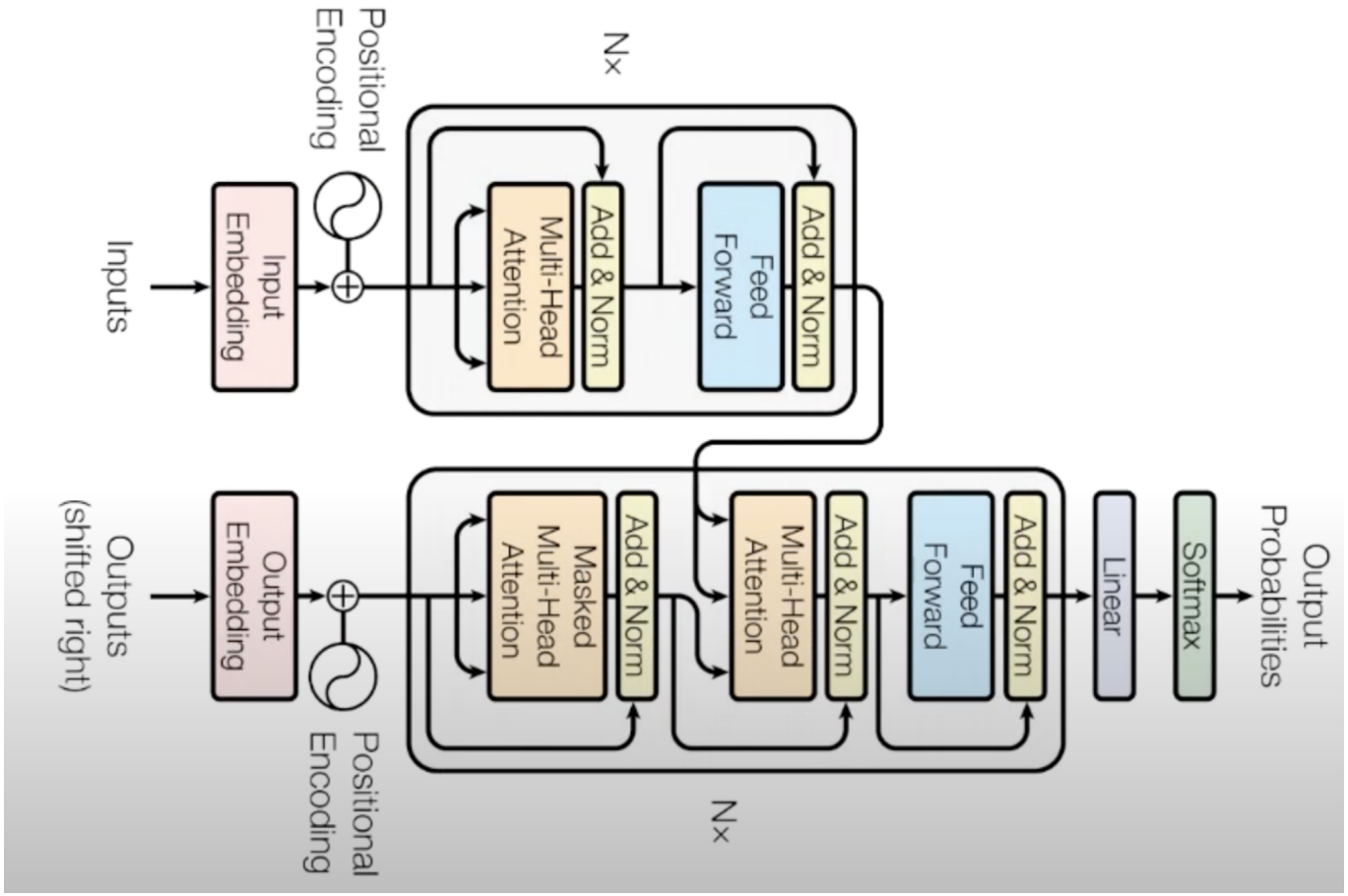
What's next?

10 Novel Applications using Transformers [DL]

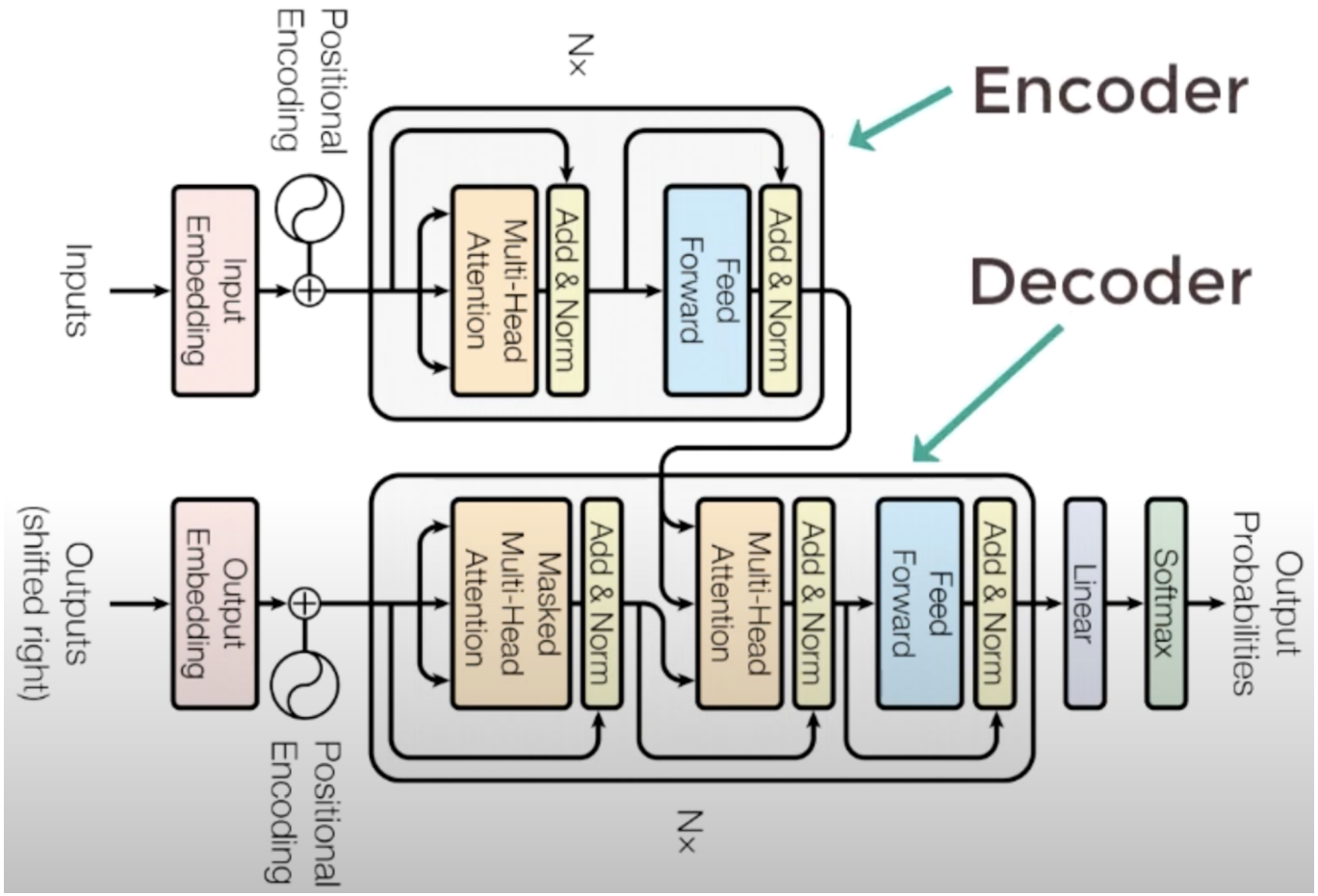
Transformers have had a lot of success in training neural language models. In the past few weeks, we've seen several trending papers with code applying Transformers to new types of task:

🏞 **Transformer for Image Synthesis** - 🔗 Esser et al. (2020)
🔵 **Transformer for Multi-Object Tracking** - 🔗 Sun et al. (2020)
🎶 **Transformer for Music Generation** - 🔗 Hsiao et al. (2021)
💃 **Transformer for Dance Generation with Music** - 🔗 Huang et al. (2021)
🔮 **Transformer for 3D Object Detection** - 🔗 Bhattacharyya et al. (2021)
🗿 **Transformer for Point-Cloud Processing** - 🔗 Guo et al. (2020)
⏰ **Transformer for Time-Series Forecasting** - 🔗 Lim et al. (2020)
👁 **Transformer for Vision-Language Modeling** - 🔗 Zhang et al. (2021)
🛣 **Transformer for Lane Shape Prediction** - 🔗 Liu et al. (2020)
🌠 **Transformer for End-to-End Object Detection** - 🔗 Zhu et al. (2021)

Transformer Flow

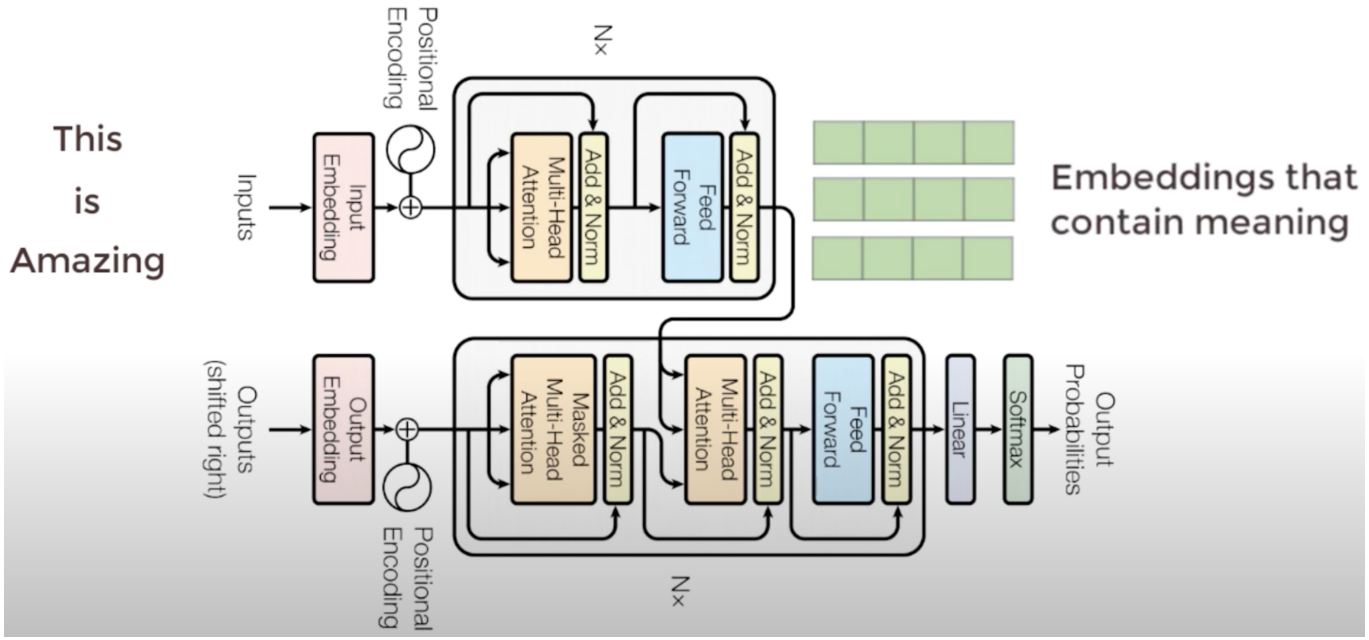


Transformer Flow



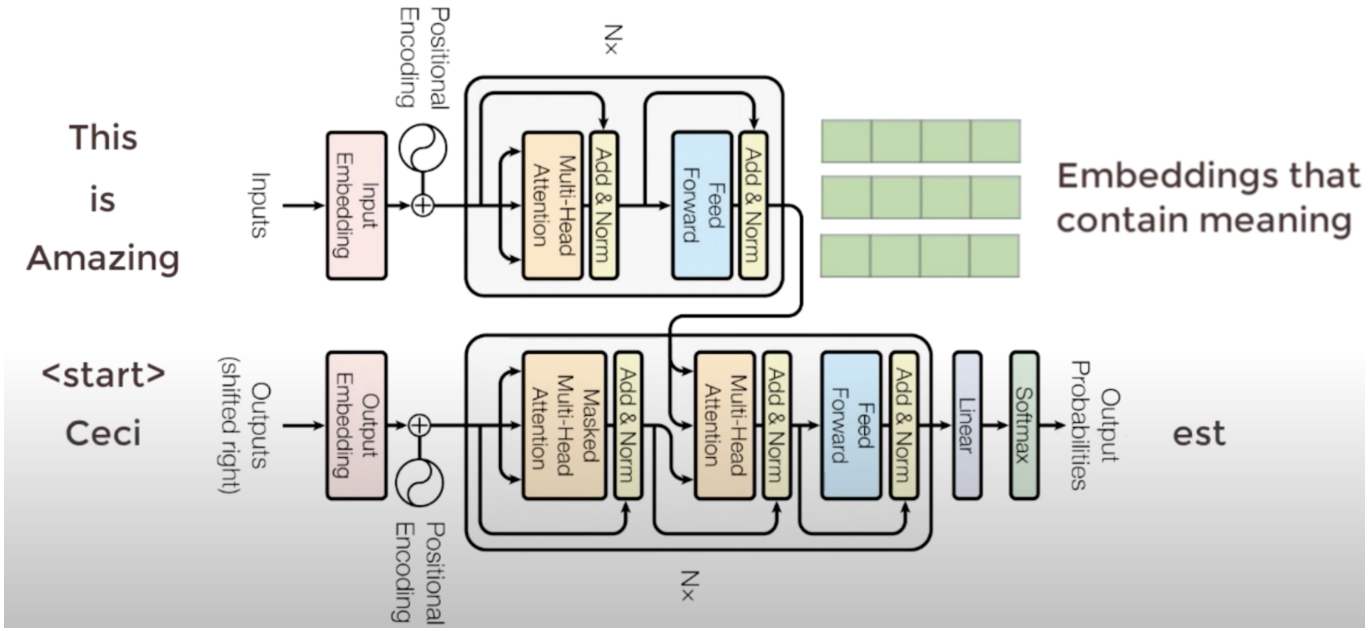
- e.g. English to French translation

Transformer Flow



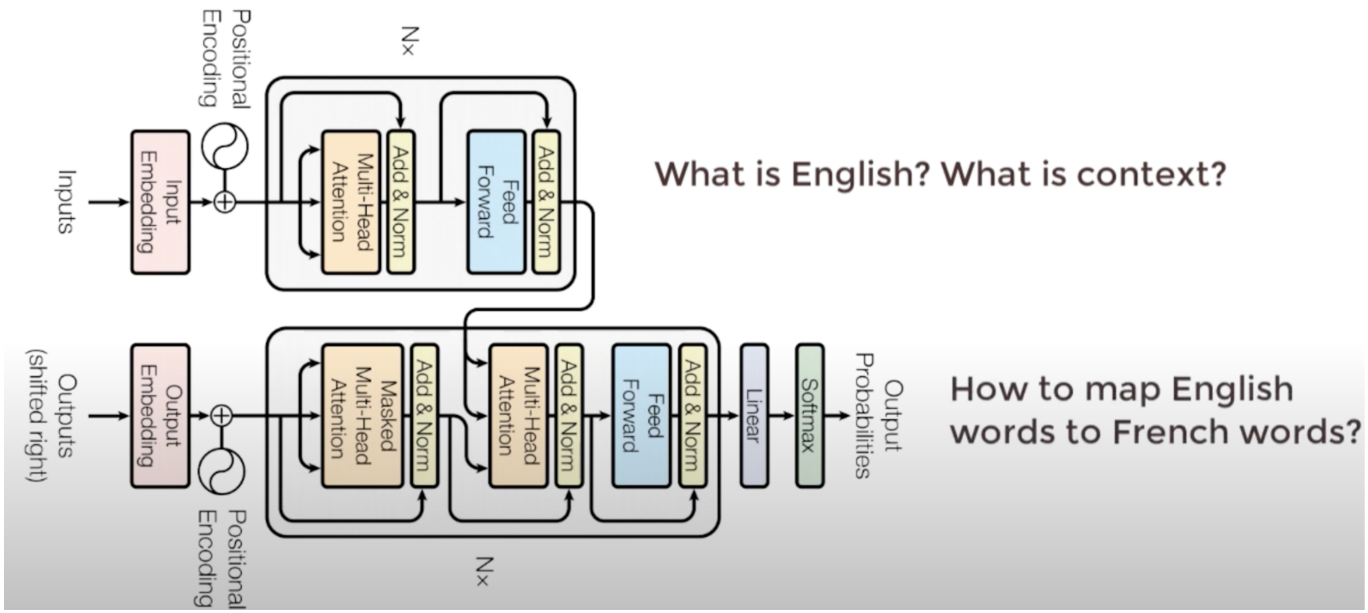
- The encoder takes English words simultaneously and generates embeddings for every word simultaneously.
- Embeddings are vectors that contain the meaning of the words.
- Similar words have closer numbers in the embeddings.

Transformer Flow



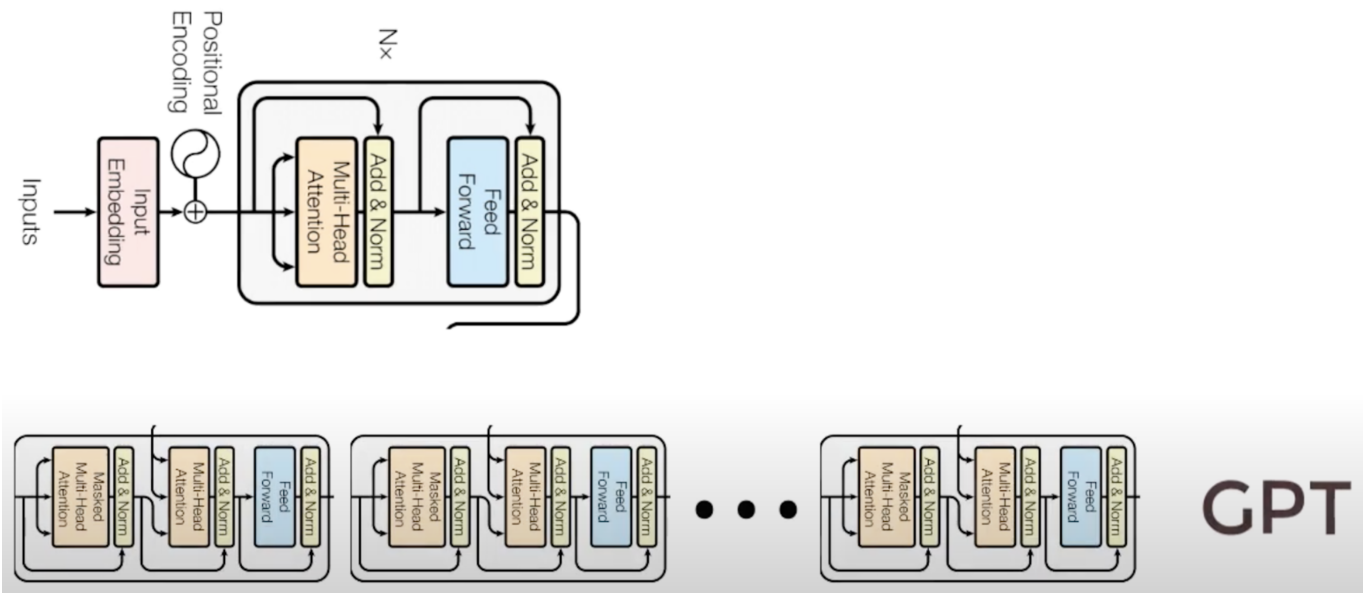
- The decoder takes these embeddings from the encoder and previously generated words of the translated French sentence.
- The decoder uses them to generate the next French word.
- We keep generating the French translation one word a time until the end of the sentence is reached.

Why this is so much better than LSTM?



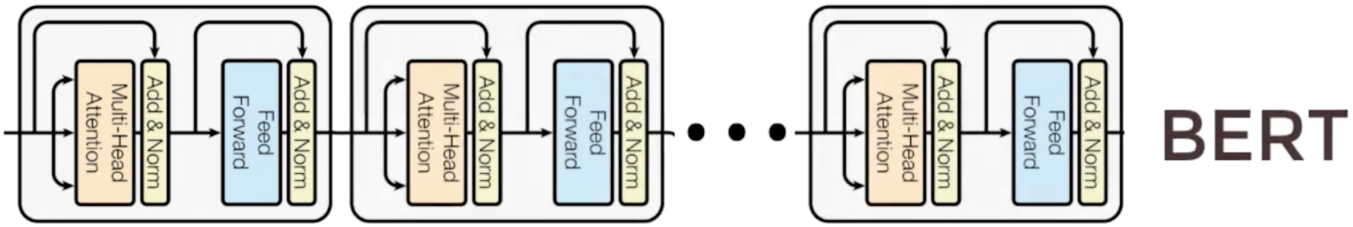
- We can physically see a separation in tasks.
  - The encoder learns what is English.
    - What is grammar and what is context?
  - The decoder learns how do English words relate to French words.
- They separately have some underlying understandings of languages.

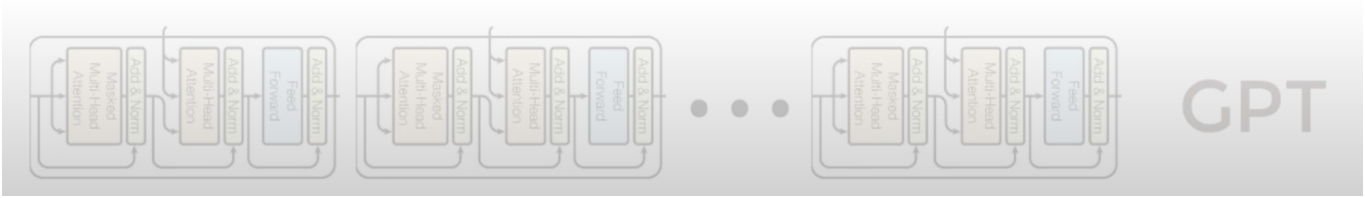
Transformer Flow



- We stack the decoders and we get GPT transformer architecture.

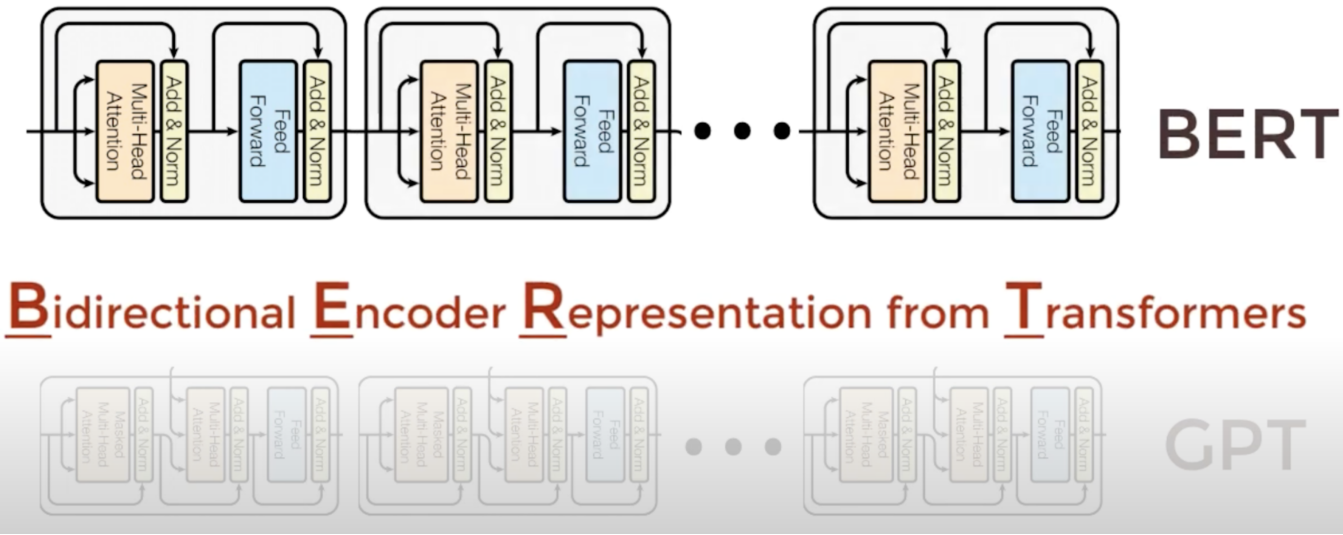
Transformer Flow





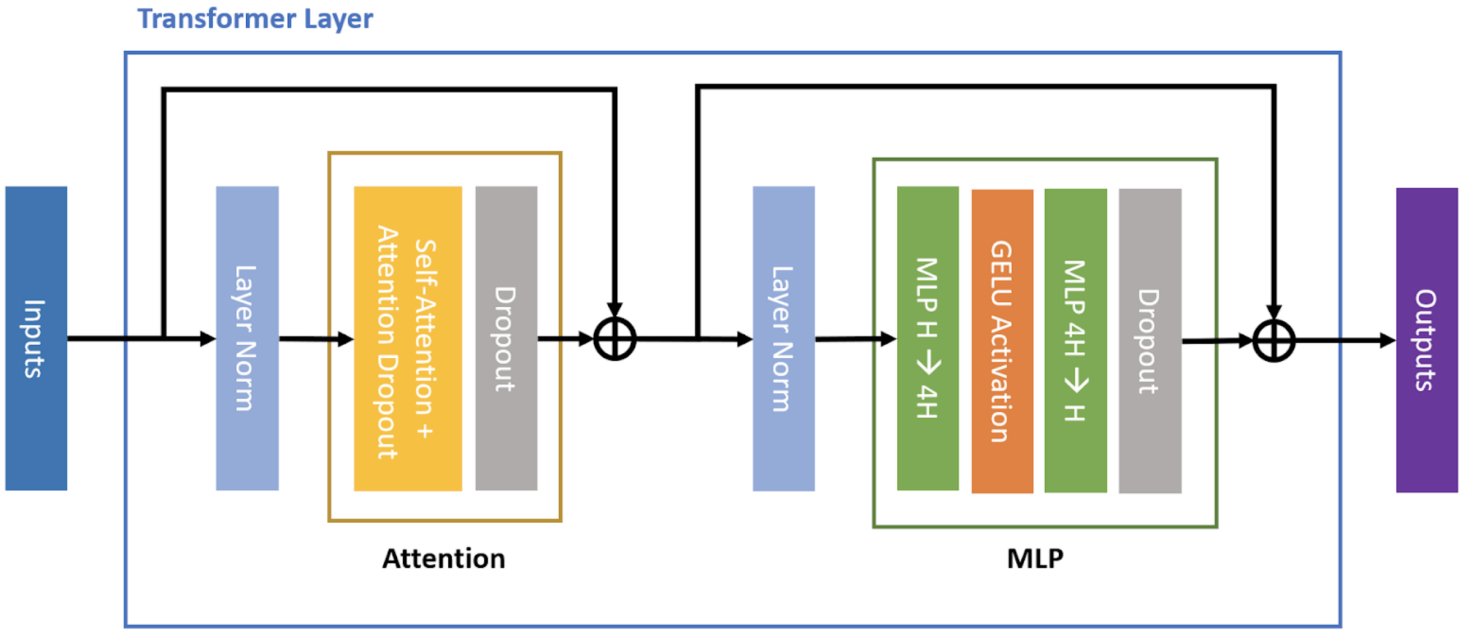
- We stack the encoders and we get BERT

Transformer Flow



- We stack the encoders and we get BERT

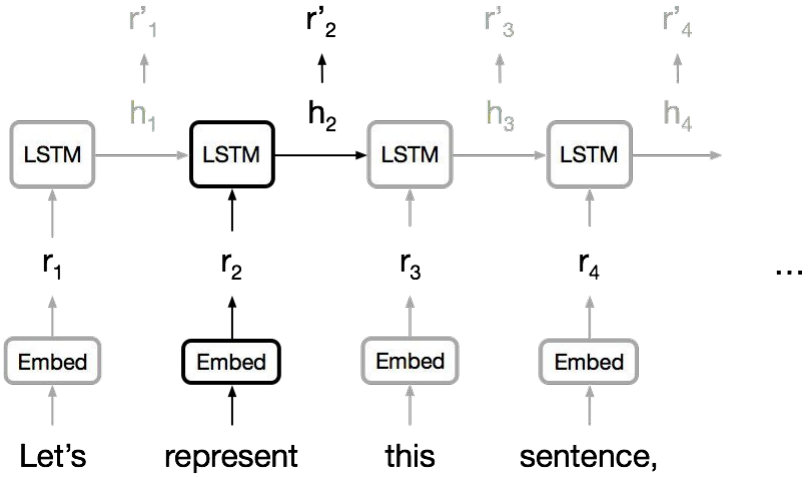
Transformer Flow



More details:
Attention & Transformers

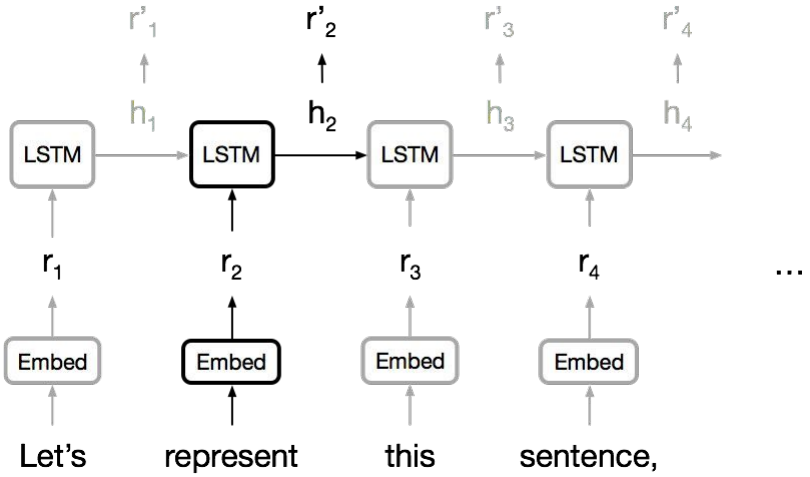
Limitations of RNNs & LSTMs

- **Poor Sequence Parallelism**
  - LSTM relies on a sequential computation process, where calculations for each time step depend on the previous time step.
  - This makes it difficult to achieve sequence parallelism, as computations must wait for the results of the previous time step.
  - In contrast, Transformer, based on self-attention mechanism, can compute representations for all positions simultaneously, enabling easier parallelization and faster training.



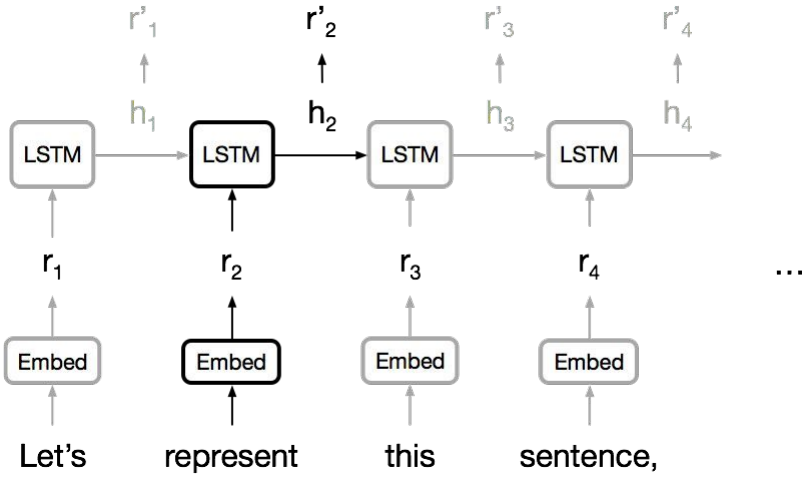
Limitations of RNNs & LSTMs

- **Difficulty in Handling Long-Distance Dependencies**
  - LSTM struggles with capturing long-range dependencies as sequences grow longer.
  - Gradient vanishing or exploding can occur during backpropagation through time, hindering the model's ability to capture relationships between distant positions.
  - In contrast, Transformer's self-attention allows direct modeling of relationships between any two positions in the sequence, making it more effective at handling long-distance dependencies.



Limitations of RNNs & LSTMs

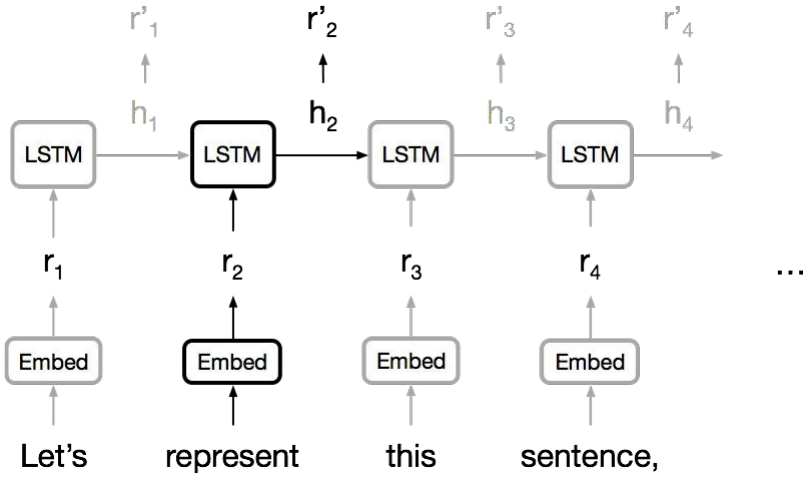
- **Inefficient Parameter Usage**
  - LSTM requires maintaining a large number of parameters, such as hidden states and memory cells, particularly when dealing with long sequences.
  - This demands more parameters and computational resources, especially for large-scale data.
  - In contrast, Transformer only needs to maintain input embeddings and positional encodings, and models relationships between sequences using self-attention, resulting in more efficient parameter usage.



LSTM vs RNN

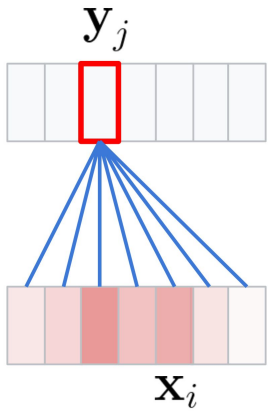
Limitations of RNNs & LSTMs

- **Limited Parallelism**: LSTM relies on sequential computation, making it difficult to parallelize effectively. In contrast, Transformer's self-attention mechanism enables parallel computation across all positions in the sequence.
- **Difficulty with Long Dependencies**: LSTM struggles to capture relationships between distant positions in long sequences due to gradient issues. Transformer handles long-range dependencies more effectively through self-attention.
- **Inefficient Parameter Usage**: LSTM requires maintaining many parameters, especially for long sequences, leading to inefficiency. Transformer's parameter usage is more efficient as it only maintains input embeddings and positional encodings.



Self-Attention

- Each of the tokens (=vectors) attends to all tokens
  - Extra tricks: learned key, query, and value projections, inverse-sqrt scaling in the softmax, and multi-headed attention (omit for simplicity)
- It's a set operation (permutation-invariant)
  - ...and hence need "position embeddings" to "remember" the spatial structure
- It's a global operation
  - Aggregates information from all tokens



$$\boldsymbol{\alpha}_j = softmax(\frac{K\mathbf{x}_1 \cdot Q\mathbf{x}_j}{\sqrt{d_K}}, \ldots, \frac{K\mathbf{x}_n \cdot Q\mathbf{x}_j}{\sqrt{d_K}})$$

$$\mathbf{y}_j = \sum_{i=1}^{n} \alpha_{ji} V\mathbf{x}_i$$

Simplified! Multi-headed attention not shown

Self-Attention

Self-Attention with Queries, Keys, Values

Let's add learnable parameters ($k \times k$ weight matrices),
and turn each vector $\mathbf{x}^{(i)}$ into three versions:

- **Query** vector $\mathbf{q}^{(i)} = \mathbf{W}_q \mathbf{x}^{(i)}$
- **Key** vector: $\mathbf{k}^{(i)} = \mathbf{W}_k \mathbf{x}^{(i)}$
- **Value** vector: $\mathbf{v}^{(i)} = \mathbf{W}_v \mathbf{x}^{(i)}$

The **attention weight of the *j*-th position** to compute the **new output for the *i*-th position** depends on the **query of i** and the **key of j (scaled):**

$$w_j^{(i)} = \frac{\exp(\mathbf{q}^{(i)}\mathbf{k}^{(j)})/\sqrt{k}}{\sum_j (\exp(\mathbf{q}^{(i)}\mathbf{k}^{(j)})/\sqrt{k})}$$

The **new output vector for the i-th position** depends on the **attention weights** and **value** vectors of all **input positions j**:

$$\mathbf{y}^{(i)} = \sum_{j=1..T} w_j^{(i)} \mathbf{v}^{(j)}$$

Self-Attention

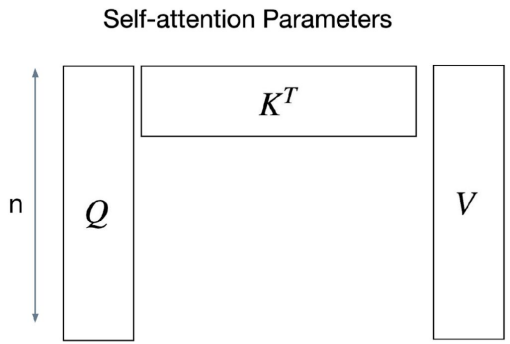
Transformer Self-Attention Layer

Input: X (matrix of n embedding vectors, each dim m)

Parameters (learned): W_Q, W_K, W_V

Compute:

$$Q = XW_Q$$
$$K = XW_K$$
$$V = XW_V$$

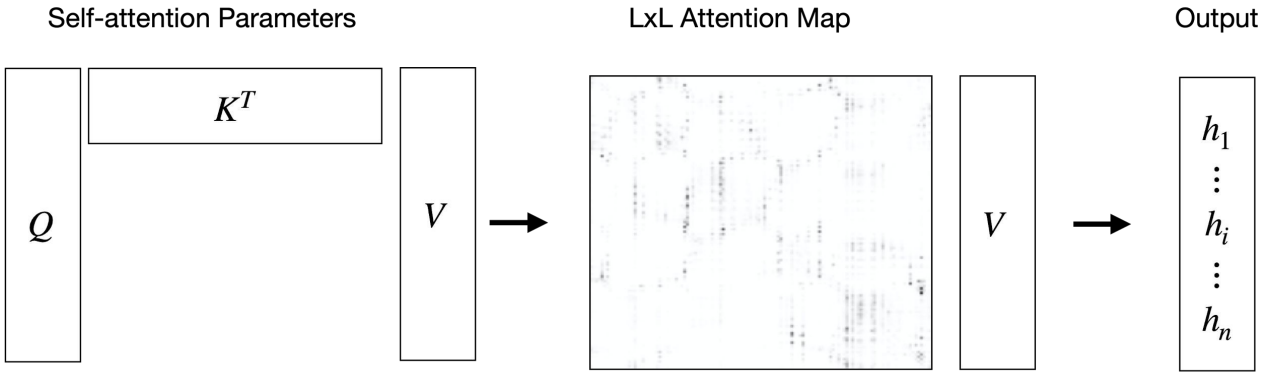


$$Q, K, V \in \mathbb{R}^{n \times m}$$

$$QK^T \in \mathbb{R}^{n \times n}$$

Self-Attention

Transformer Self-Attention Layer



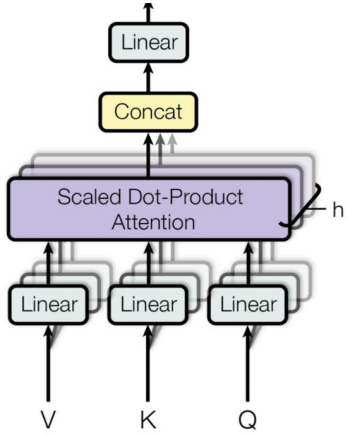
Output matrix H = $\text{softmax}(\frac{1}{\sqrt{d}} QK^T) \cdot V$

Self-attention explicitly models interactions between all pairs of input embeddings

Self-Attention

Multi-Head Attention

- Learn *h* different linear projections of Q,K,V
- Compute attention separately on each of these *h* versions
- Concatenate and project the resultant vectors to a lower dimensionality.
- Each attention head can use low dimensionality



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Position Embedding

Add positional embeddings to input embeddings

-Same dimension

-Can be learned or fixed

Fixed encoding: sin / cos of different frequencies:

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

Transformer

Attention Is All You Need

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

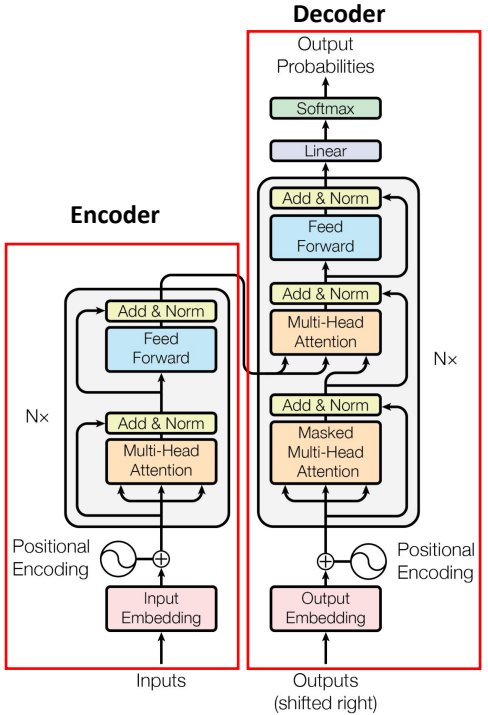
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Attention Is All You Need

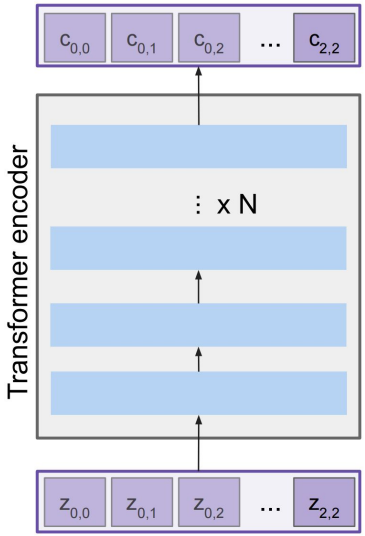
- Typically applied to 1-D sequence data
- Encoder
  - A stack of alternating self-attention and MLP blocks
  - Residuals and LayerNorm
- Decoder
  - A slightly more complicated architecture useful when the output space is different from the input space (e.g. translation)



Attention Is All You Need., Advances in Neural Information Processing Systems 30 (NIPS 2017)

Transformer

The Transformer Encoder Block

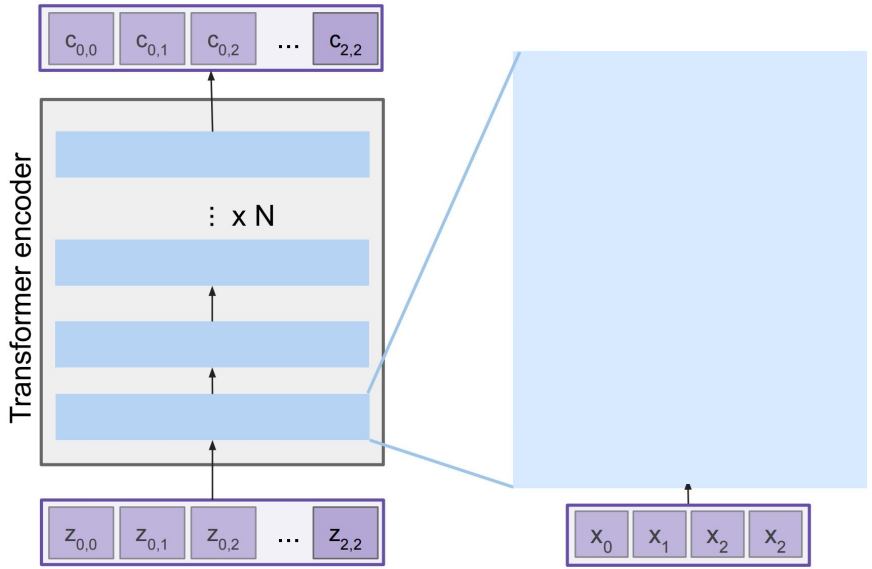


- Made up of N encoder blocks
- In “Attention is All You Need”, N=6

Attention Is All You Need., Advances in Neural Information Processing Systems 30 (NIPS 2017)

Transformer

The Transformer Encoder Block

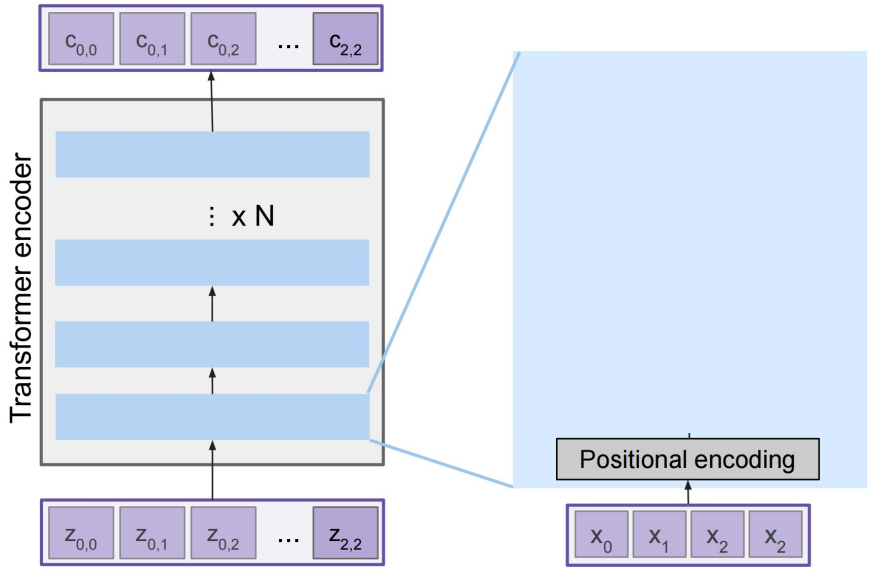


Let's dive into one encoder block as an example

Attention Is All You Need., Advances in Neural Information Processing Systems 30 (NIPS 2017)

Transformer

The Transformer Encoder Block

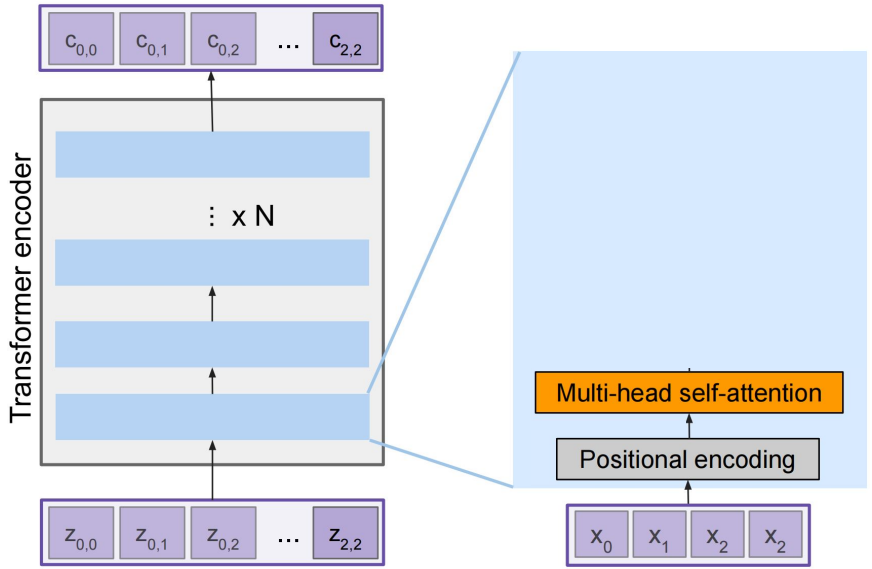


Add positional encoding

Attention Is All You Need., Advances in Neural Information Processing Systems 30 (NIPS 2017)

Transformer

The Transformer Encoder Block



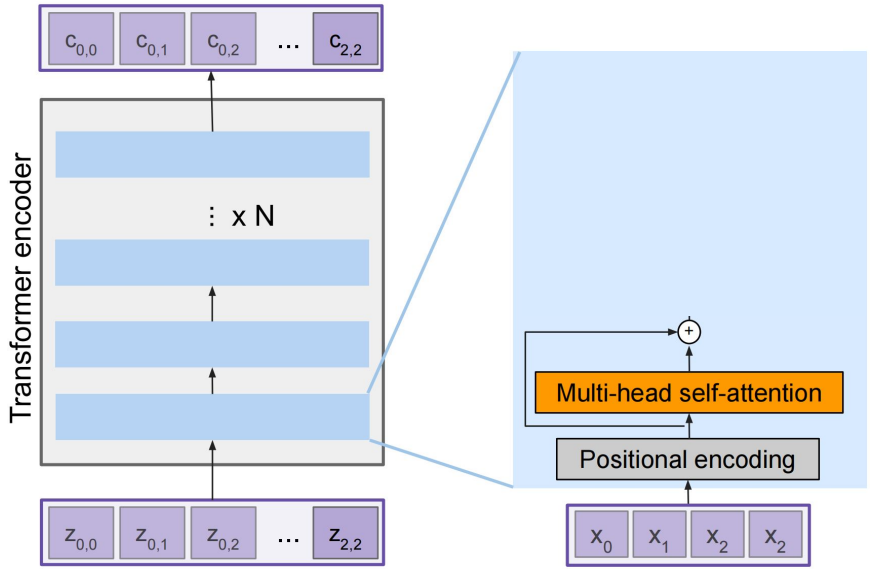
Attention attends over all the vectors

Add positional encoding

Attention Is All You Need., Advances in Neural Information Processing Systems 30 (NIPS 2017)

Transformer

The Transformer Encoder Block



Residual connection

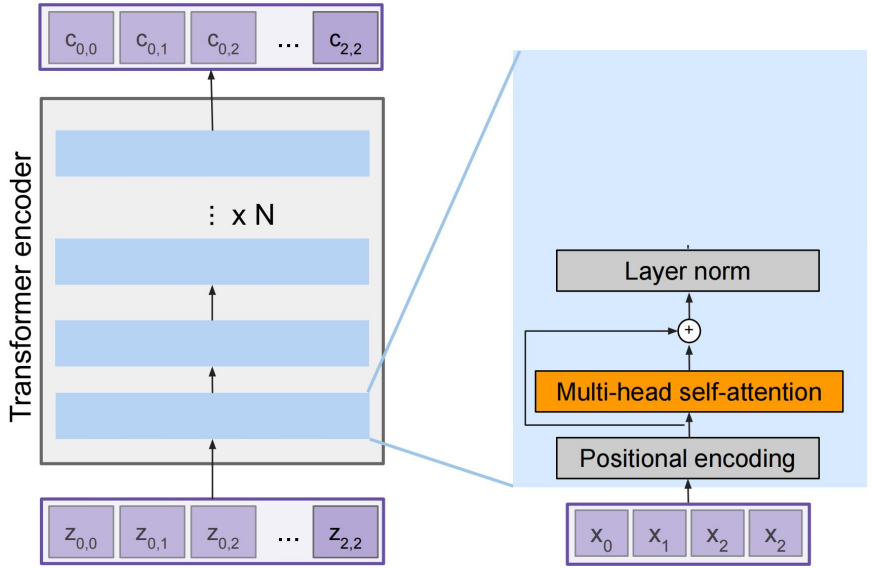
Attention attends over all the vectors

Add positional encoding

Attention Is All You Need., Advances in Neural Information Processing Systems 30 (NIPS 2017)

Transformer

The Transformer Encoder Block



LayerNorm over each vector individually

Residual connection

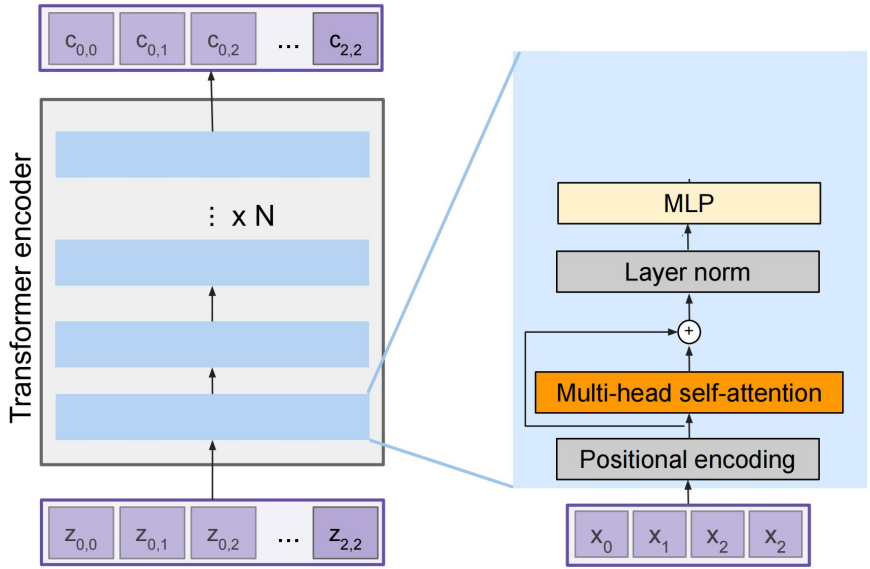
Attention attends over all the vectors

Add positional encoding

Attention Is All You Need., Advances in Neural Information Processing Systems 30 (NIPS 2017)

Transformer

The Transformer Encoder Block



MLP over each vector individually

LayerNorm over each vector individually

Residual connection

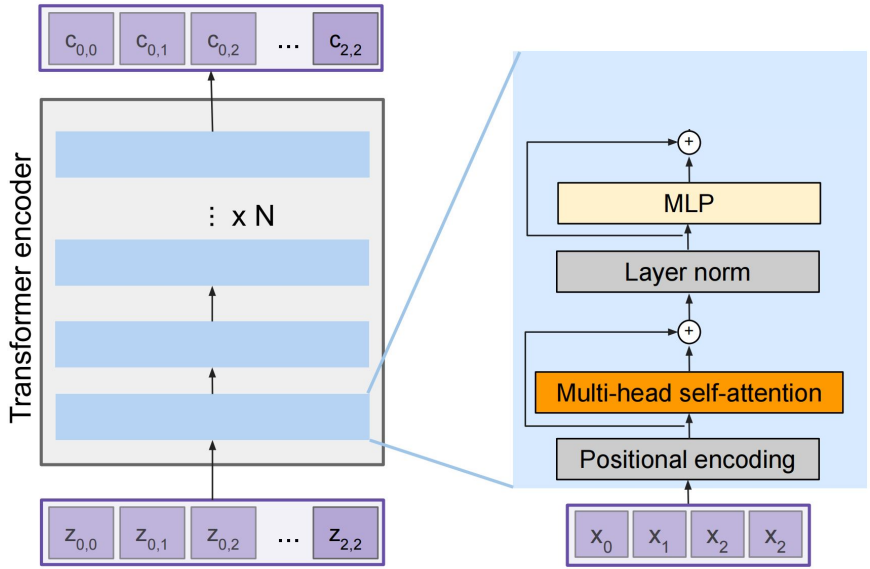
Attention attends over all the vectors

Add positional encoding

Attention Is All You Need., Advances in Neural Information Processing Systems 30 (NIPS 2017)

Transformer

The Transformer Encoder Block



Residual connection

MLP over each vector individually

LayerNorm over each vector individually

Residual connection

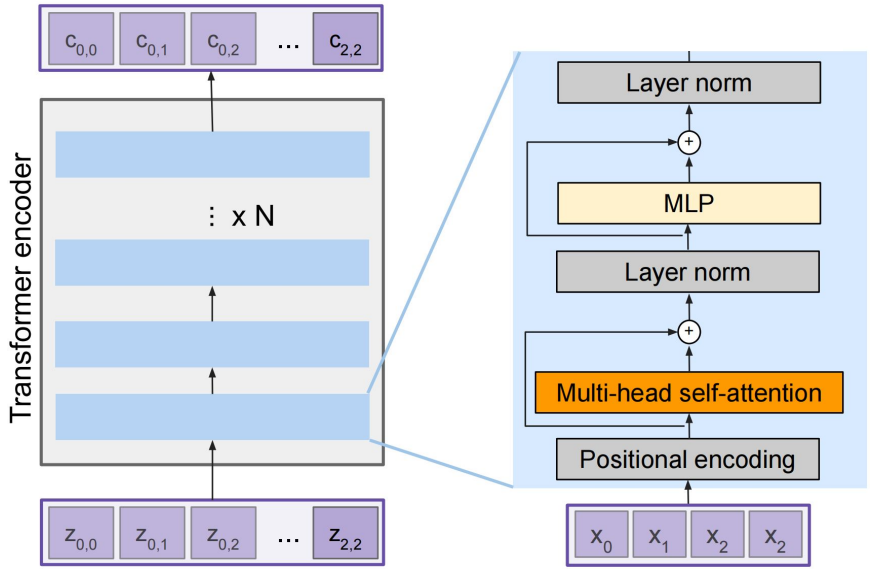
Attention attends over all the vectors

Add positional encoding

Attention Is All You Need., Advances in Neural Information Processing Systems 30 (NIPS 2017)

Transformer

The Transformer Encoder Block



LayerNorm over each vector individually

Residual connection

MLP over each vector individually

LayerNorm over each vector individually

Residual connection

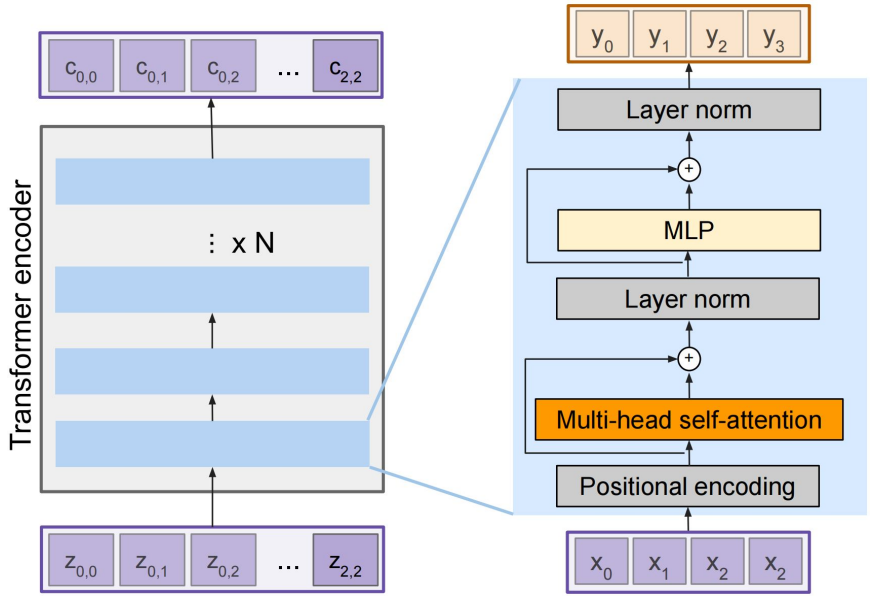
Attention attends over all the vectors

Add positional encoding

Attention Is All You Need., Advances in Neural Information Processing Systems 30 (NIPS 2017)

Transformer

The Transformer Encoder Block



Transformer Encoder Block:

Inputs: Set of vectors x
Outputs: Set of vectors y

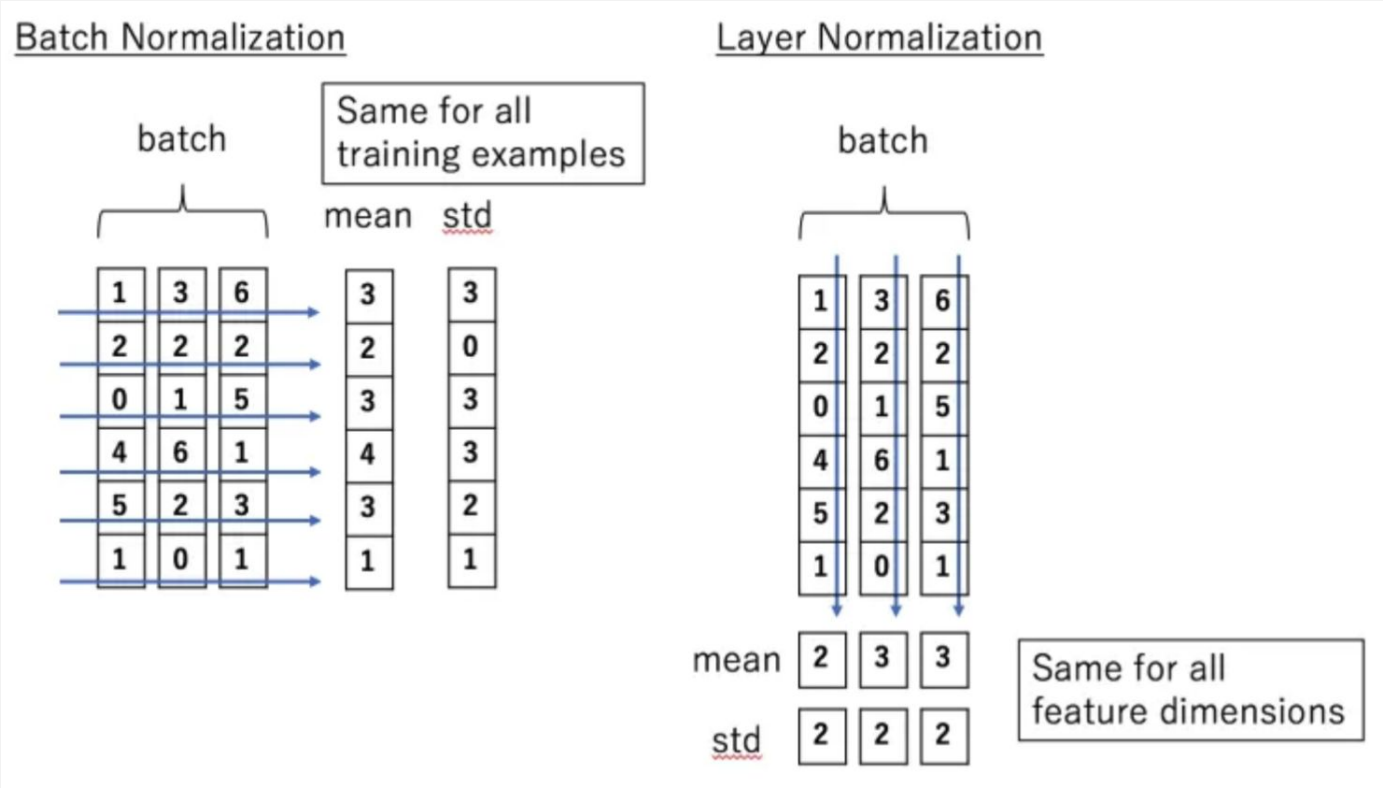
Self-attention is the only interaction between vectors.

Layer norm and MLP operate independently per vector.

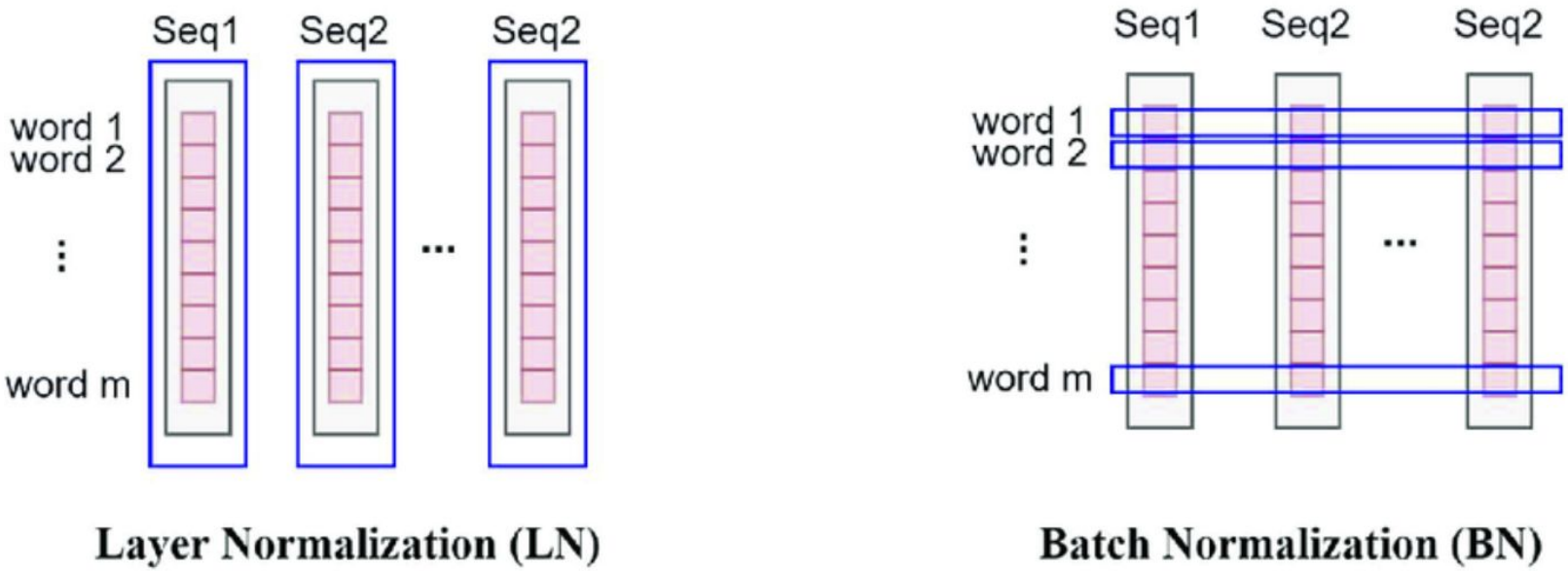
Highly scalable, highly parallelizable, but high memory usage.

Attention Is All You Need., Advances in Neural Information Processing Systems 30 (NIPS 2017)

Layer Norm vs Batch Norm



Layer Norm vs Batch Norm



Layer Normalization (LN) and Batch Normalization (BN).

Similarity between Layer Norm and Batch Norm

Element-wise operation: Both Layer Normalization and Batch Normalization work on each element individually, without considering other elements.

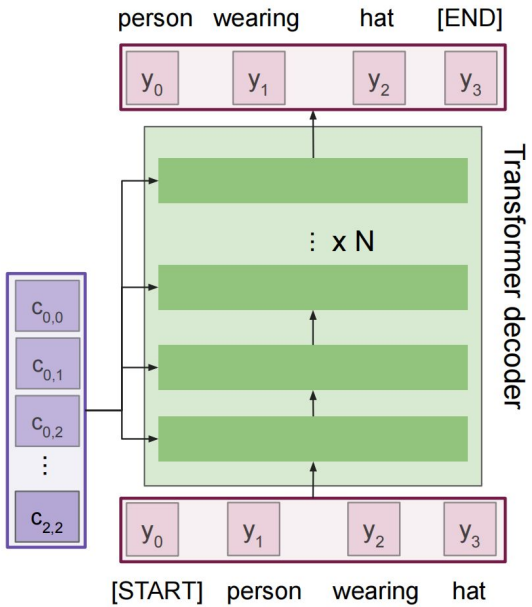
Normalization across features: They both normalize across features. Layer Norm normalizes all features for each sample, while Batch Norm normalizes all samples for each feature within a batch.

Reducing internal covariate shift: Both techniques aim to reduce internal covariate shift during training, helping the network converge faster and generalize better.

Preserving input-output shape: After normalization, the input and output shapes remain the same. For an input tensor of shape (batch_size, features), both Layer Norm and Batch Norm produce an output tensor with the same shape (batch_size, features).

Transformer

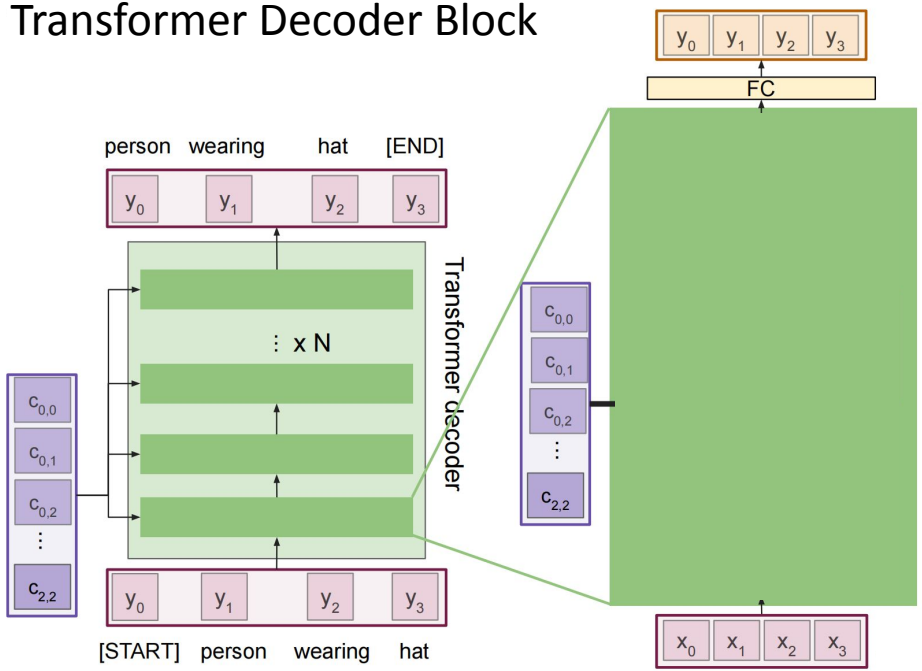
The Transformer Decoder Block



- Made up of N encoder blocks
- In “Attention is All You Need”, N=6

Transformer

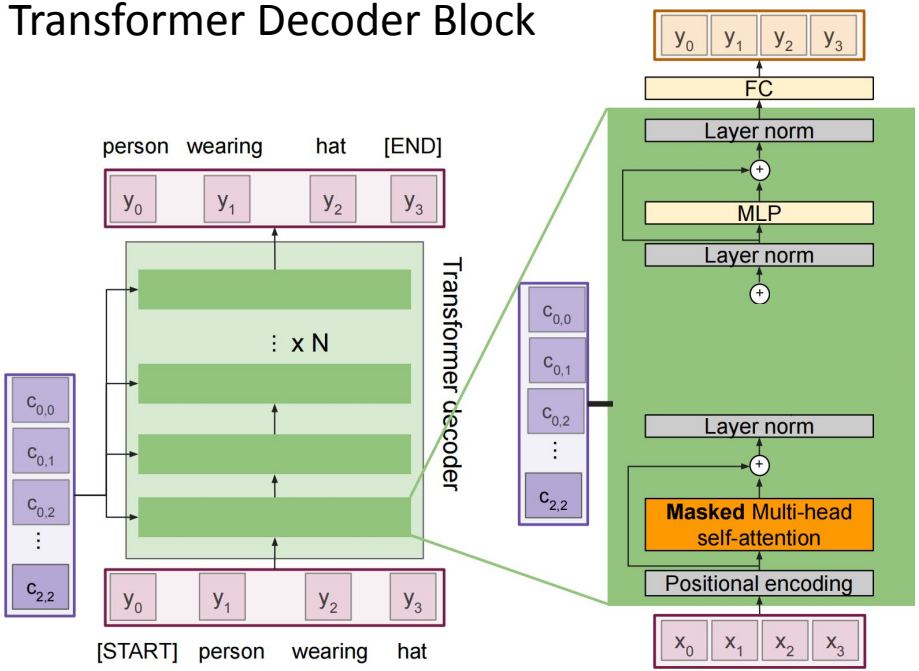
The Transformer Decoder Block



Let's dive into one decoder block as an example

Transformer

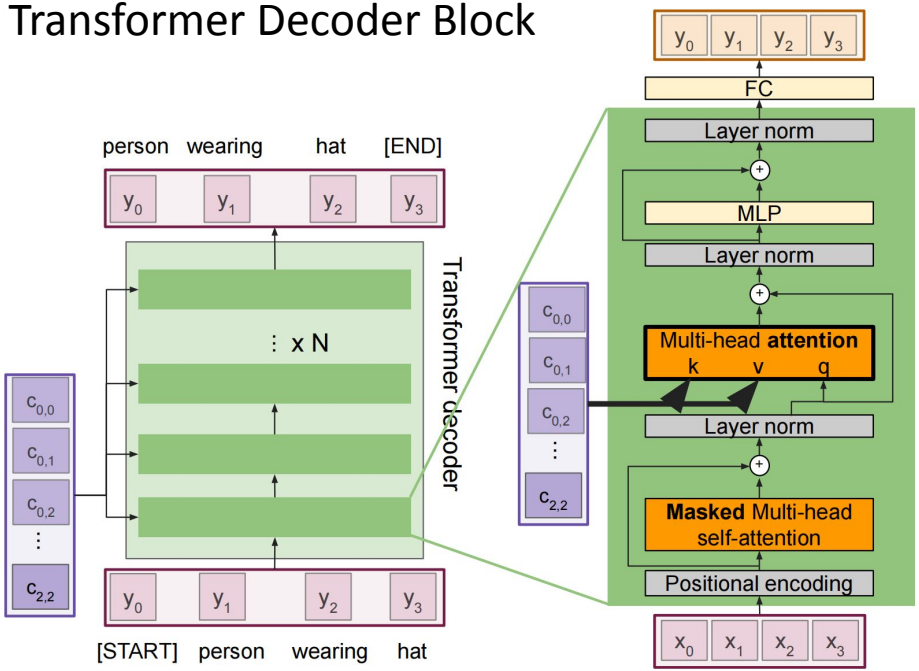
The Transformer Decoder Block



Most of the network is the same the transformer encoder.

Transformer

The Transformer Decoder Block

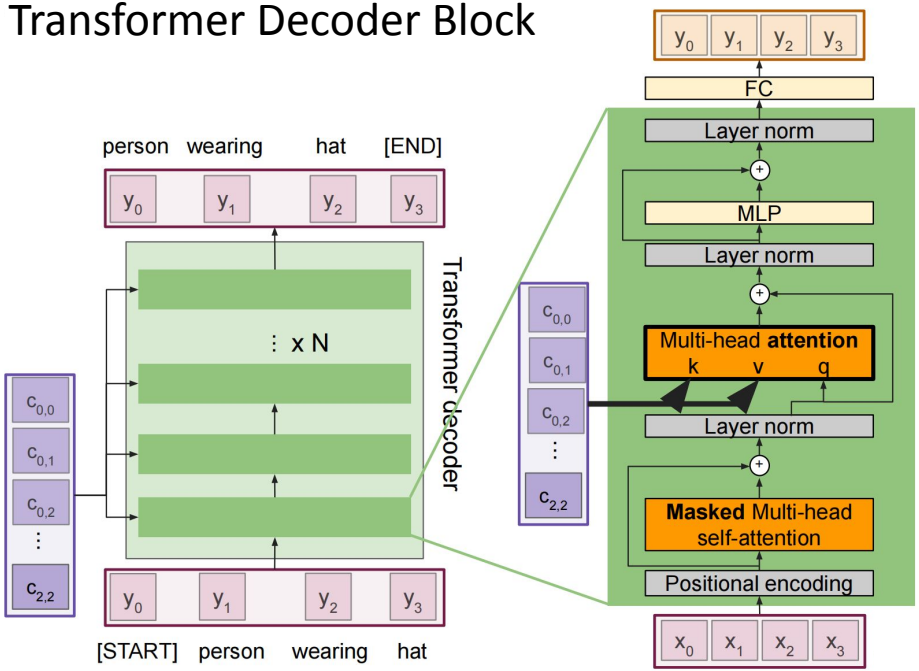


Multi-head attention block attends over the transformer encoder outputs.

The outputs from the encoder act as “K” and “V”.

Transformer

The Transformer Decoder Block



Transformer Decoder Block:

Inputs: Set of vectors x and
Set of context vectors c.
Outputs: Set of vectors y.

Masked Self-attention only interacts with past inputs.

Multi-head attention block is NOT self-attention. It attends over encoder outputs.

Highly scalable, highly parallelizable, but high memory usage.

Attention & Transformer Summary

Self-attention also known as Scaled Dot-Product Attention

Self-attention is one of the most common attention mechanisms used in Transformers. In self-attention, the model can establish relationships between different positions within an input sequence. This allows the model to better understand how each input position relates to other positions.

- Query (Q): Represents the current position or token for which we want to compute attention scores.
- Key (K): Represents other positions or tokens in the input sequence.
- Value (V): Contains information associated with each position.
- We compute the dot product between the Query and Key vectors.
- To prevent large dot products from causing gradients to explode, we scale the dot product by dividing it by the square root of the dimension of the Key vectors.

Attention & Transformer Summary

Self-attention → Enhance → Multi-Head Self-Attention

- In Multi-Head Attention, we use several attention heads in parallel.
- Each head learns different representations for Query, Key, and Value.
- These heads allow the model to capture diverse patterns and relationships within the input sequence.
- After computing attention scores for each head, we concatenate the outputs from all heads.The concatenated output is then linearly transformed to obtain the final output.

Attention & Transformer Summary

Self-attention

also known as

reduce computational cost

Global attention

- Global Attention allows the model to consider relationships between all input positions when computing attention scores, not just the positions directly related to the current position.
- In other words, it provides a broader context for understanding the input sequence.

Local Attention

- Local Attention differs from global attention. It focuses only on the positions near the current position within the input sequence.
- It restricts the attention to a local neighborhood, reducing computational costs.

Attention & Transformer Summary

Multi-Head Self-attention

- Used in Transformer Encoder and Transformer Decoder

Multi-Head Cross-attention

- Cross Attention involves considering not only the internal relationships within the current sequence but also the interactions between another sequence (usually another input sequence or context sequence) and the current sequence.
- Used in Transformer Decoder

Applications

- Vision: ViT, Swin Transformer, DETR
- Language: BERT, GPTs
- Vision-Language: CLIP

Summary

- **Transformers** are a type of layer that uses **self-attention** and layer norm.
- It is **highly scalable** and **high parallelizable**.
- **Faster training, larger models, better performance** on both vision and language tasks.
- Replacing **RNNs**, **LSTMs**, and even **CNNs**.

References

- Attention and Transformers: http://cs231n.stanford.edu/slides/2022/lecture_11_ruohan.pdf
- Transformers. https://fullstackdeeplearning.com/spring2021/lecture-4/
- Vision Transformers. https://web.eecs.umich.edu/~justincj/slides/eecs498/WI2022/598_WI2022_lecture18.pdf
- Transformers For Vision. https://cs.nyu.edu/~fergus/teaching/vision/5_transformers.pdf
- Attention is all you need. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, Illia Polosukhin. Advances in neural information processing systems 2017.
- Self-Attention for Vision. https://icml.cc/media/icml-2021/Slides/10842.pdf
- End-to-End Object Detection with Transformers. https://www.cs.utexas.edu/~yukez/cs391r_fall2021/slides/pre_09-14_Jay.pdf

THANK YOU