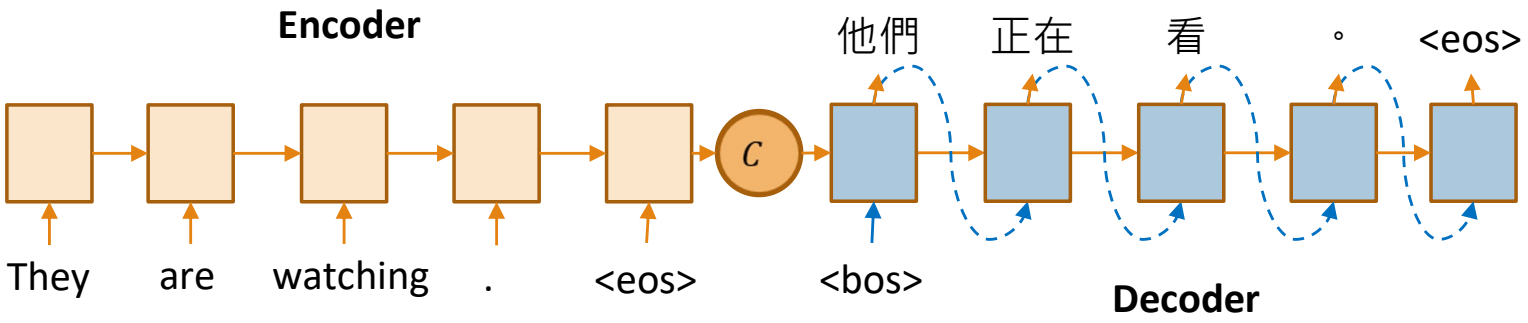
Transformer
and its followers

Yong-Sheng Chen
Dept. Computer Science, NYCU

Introduction - Deep Learning in NLP

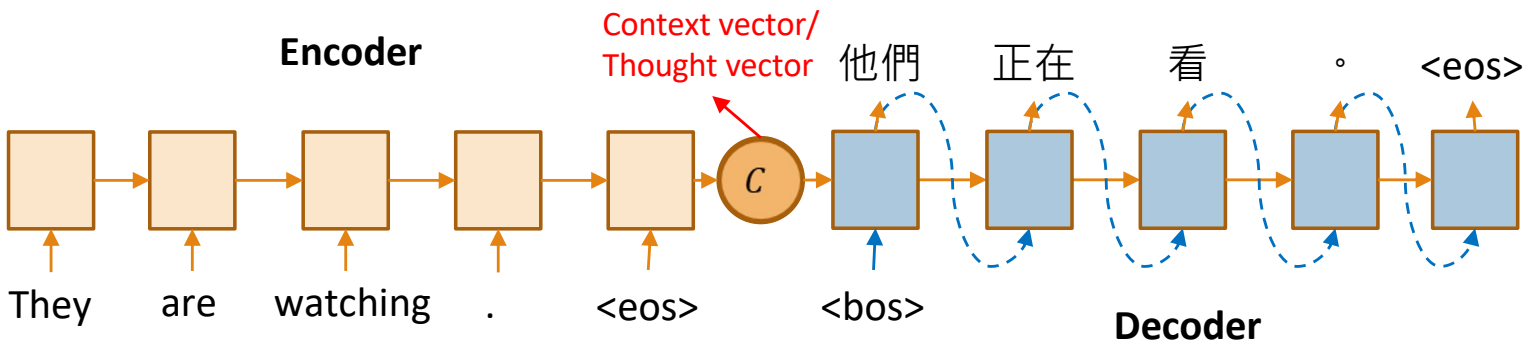
- Tasks that deep learning can apply: text categorization, question answering, machine translation, automatic summarization.
- Translation, as an important example:
  - The classic solution is the encoder-decoder framework, or sequence to sequence (seq2seq) model.



Source:https://iter01.com/563809.html

Introduction - Seq2seq

- The encoder compresses the input sentence into a fixed-length context vector.
- If the sentence is very long, a fixed-length context vector will not work well.

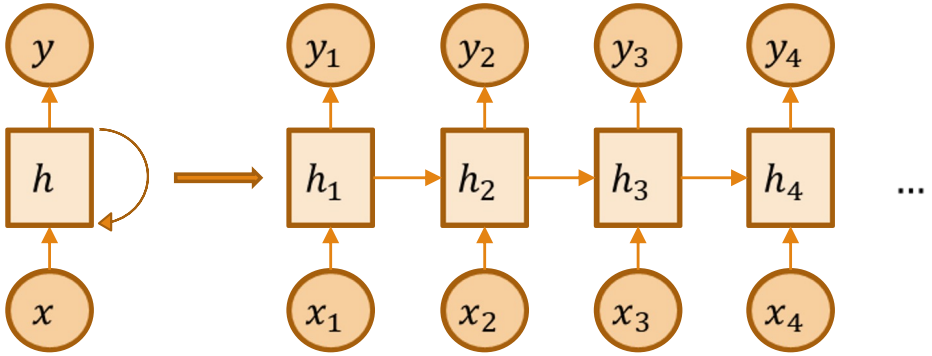


Source:https://iter01.com/563809.html

Introduction - Problem in RNNs

- RNN, LSTM, and GRN have been SOTA in sequence modeling and translation problem.
- Recurrent model generates hidden states as a function of the previous hidden state and input.

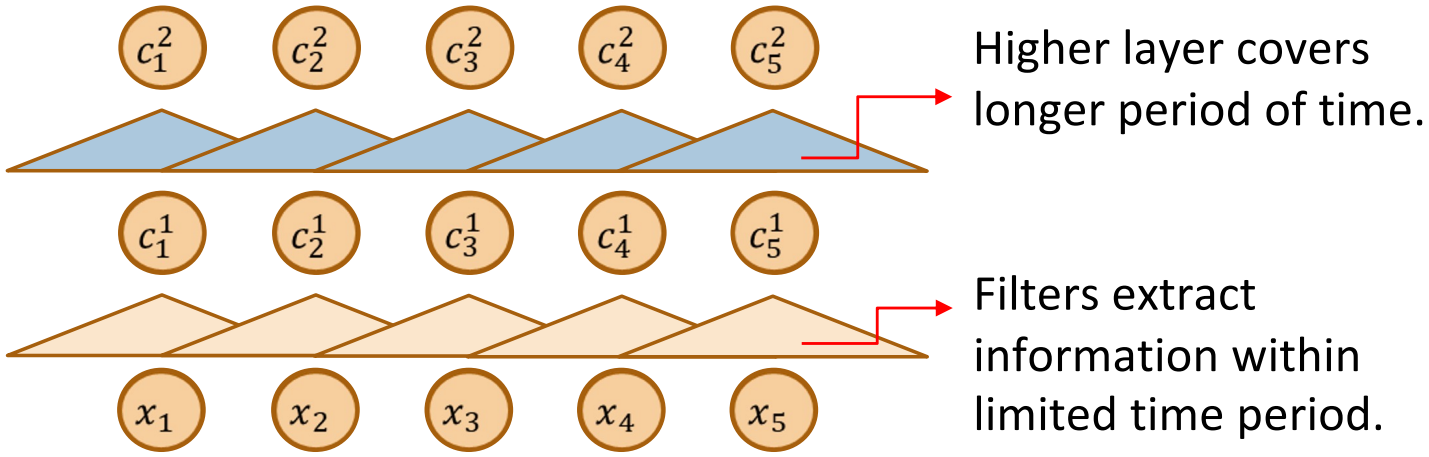
- Difficulty: it is **hard to parallelize the computation.**



Source:https://www.youtube.com/watch?v=ugWDIIOHtPA

Introduction - CNN models

- Reducing sequential computation by using CNN.
  - Extended Neural GPU[16], ByteNet[8], ConvS2S[9].
- There must be many layers to incorporate long-time information.
  1. Large computational cost.
  2. Shallow layer only considers short time information.



Source:https://www.youtube.com/watch?v=ugWDIIOHtPA

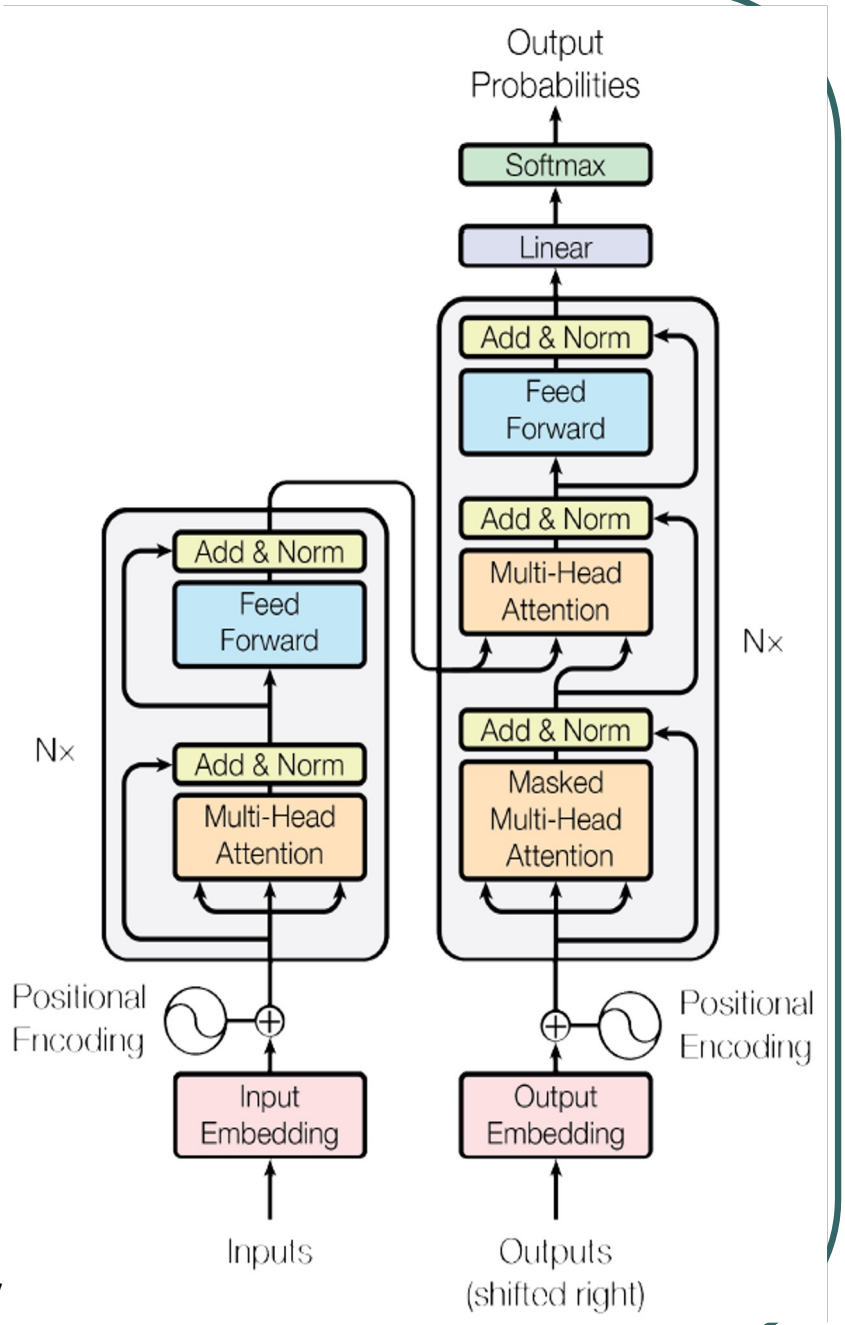
[16] Łukasz Kaiser and Samy Bengio. Can active memory replace attention? In Advances in Neural Information Processing Systems, (NIPS), 2016.
[18] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. arXiv preprint arXiv:1610.10099v2, 2017.
[9] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. arXiv preprint arXiv:1705.03122v2, 2017.

Transformer

- In the work “Attention Is All You Need” in *NIPS* 2017, the network architectures such as RNN and CNN are completely abandoned.
- It only adopt new attention mechanism: self-attention/intra-attention.
- A new end-to-end architecture with attention.

Transformer for NLP

1. Input
2. Embedding Layer
3. Position Encoding
4. Attention
   1. Self-Attention
   2. Multi-Head Attention

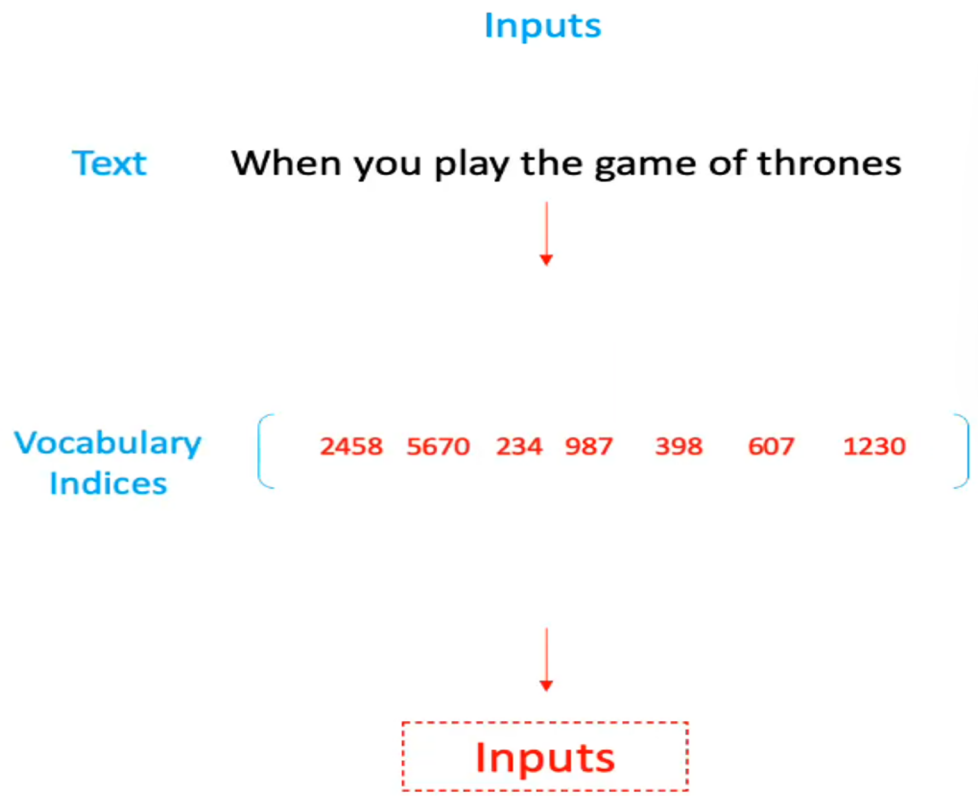


https://www.youtube.com/watch?v

Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

Input

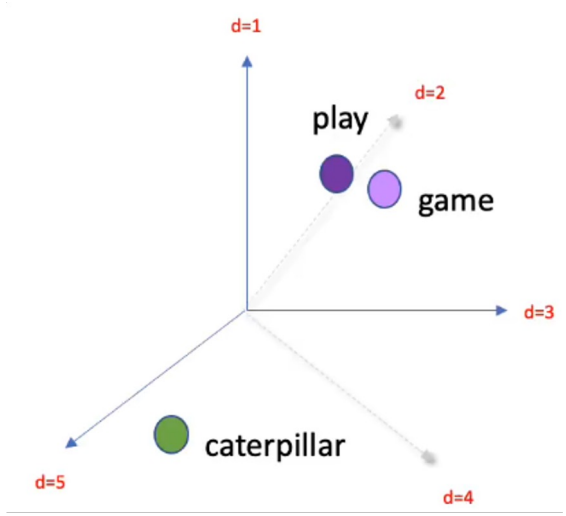
- Takes all the words present in our training data
- Assign an index for each word
- These indices are the input to the transformer

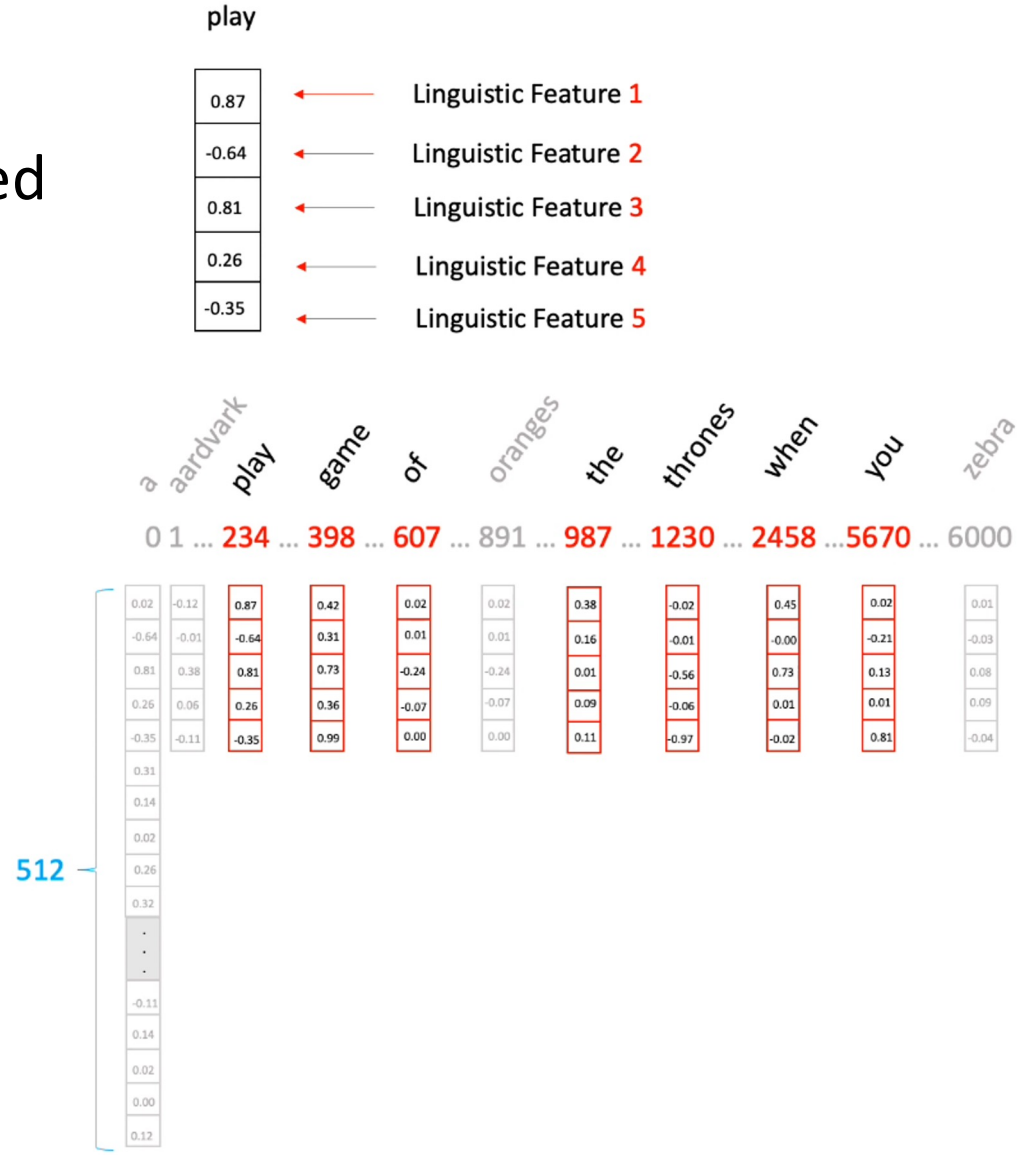


Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

Embedding Layer

- Vector representation:
  - Each word index is attached with a vector
  - Initialization with random numbers
  - Adjusted during training





Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

Position Encoding

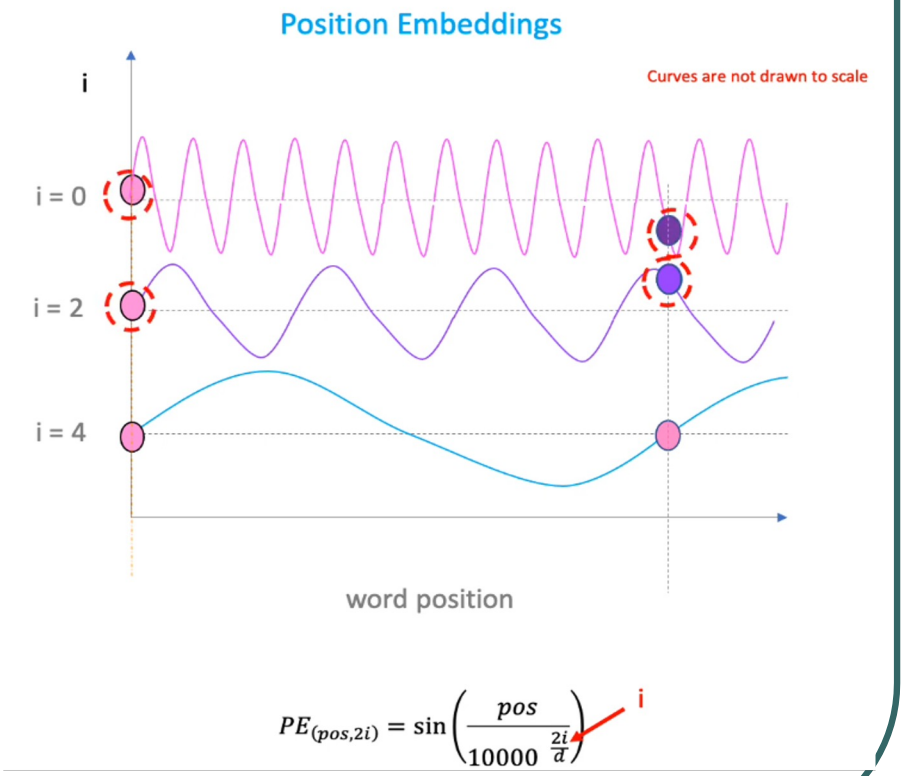
- Transformer calculate self-attention from all word embeddings and thus loses word ordering information.
- Position encoding:
  - For even indices, use Sin function
  - For odd indices, use Cos function

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

Vaswani et al 2017

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$e_0 + p_0$	$e_1 + p_1$	$e_2 + p_2$	$e_3 + p_3$
0.47	0.87	0.02	0.38
0.31	-0.64	0.01	0.16
0.73	0.81	-0.24	0.01
0.36	0.26	-0.07	0.09
0.99	-0.35	0.00	0.00



Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

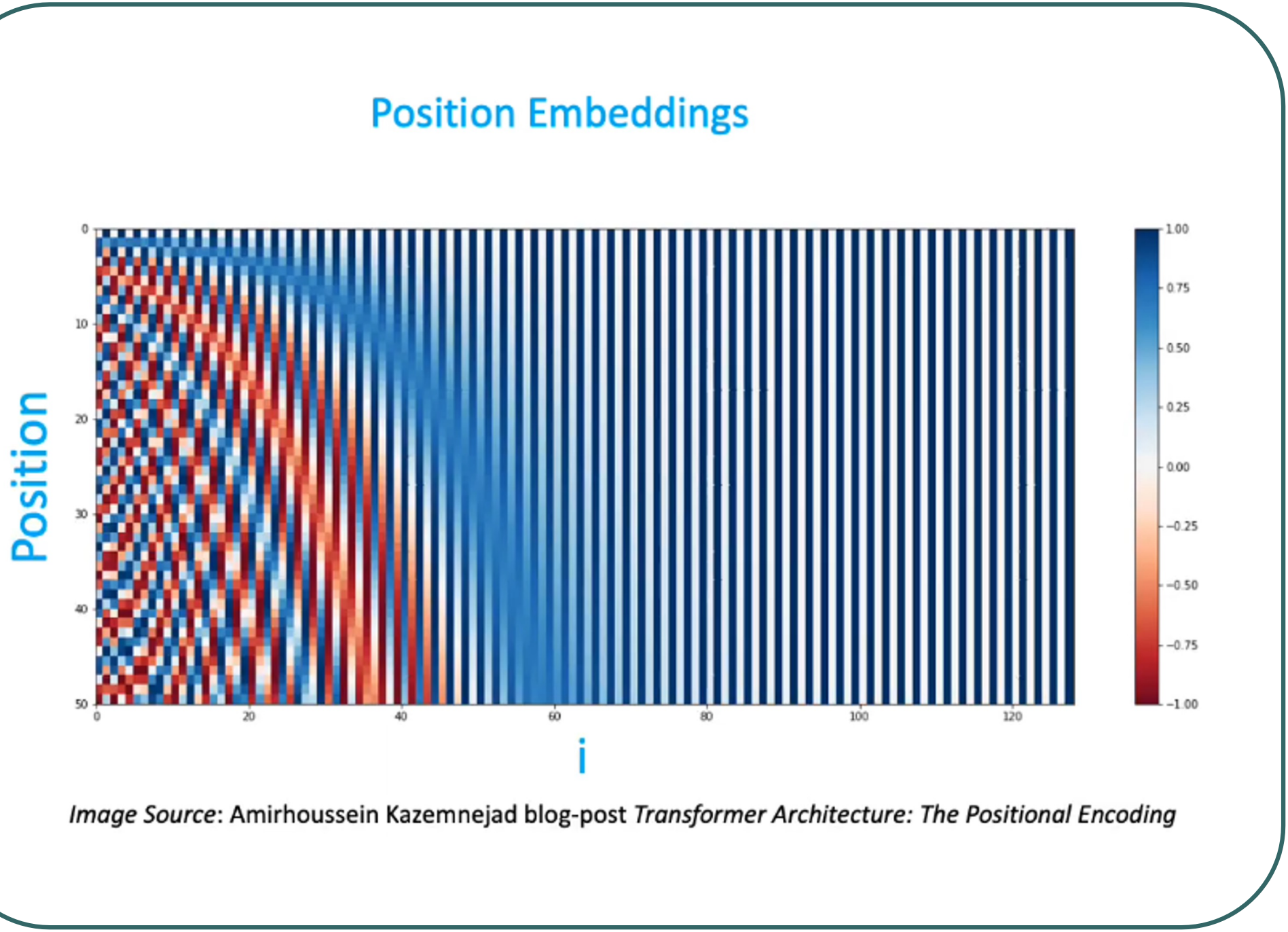


Image Source: Amirhoussein Kazemnejad blog-post *Transformer Architecture: The Positional Encoding*

Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

Self-Attention: query, key, value

- x: input token
- e: positional vector
- q: query

$$q_i = W_q a_i$$

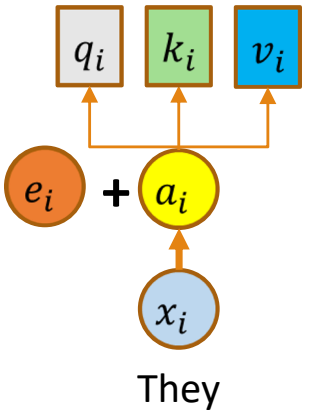
- k: key

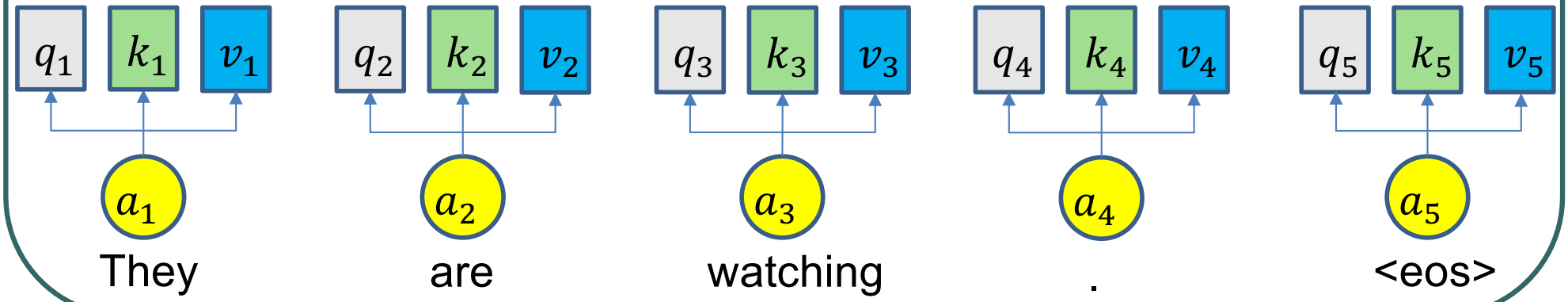
$$k_i = W_k a_i$$

- v: value

$$v_i = W_v a_i$$

trainable with random initialization



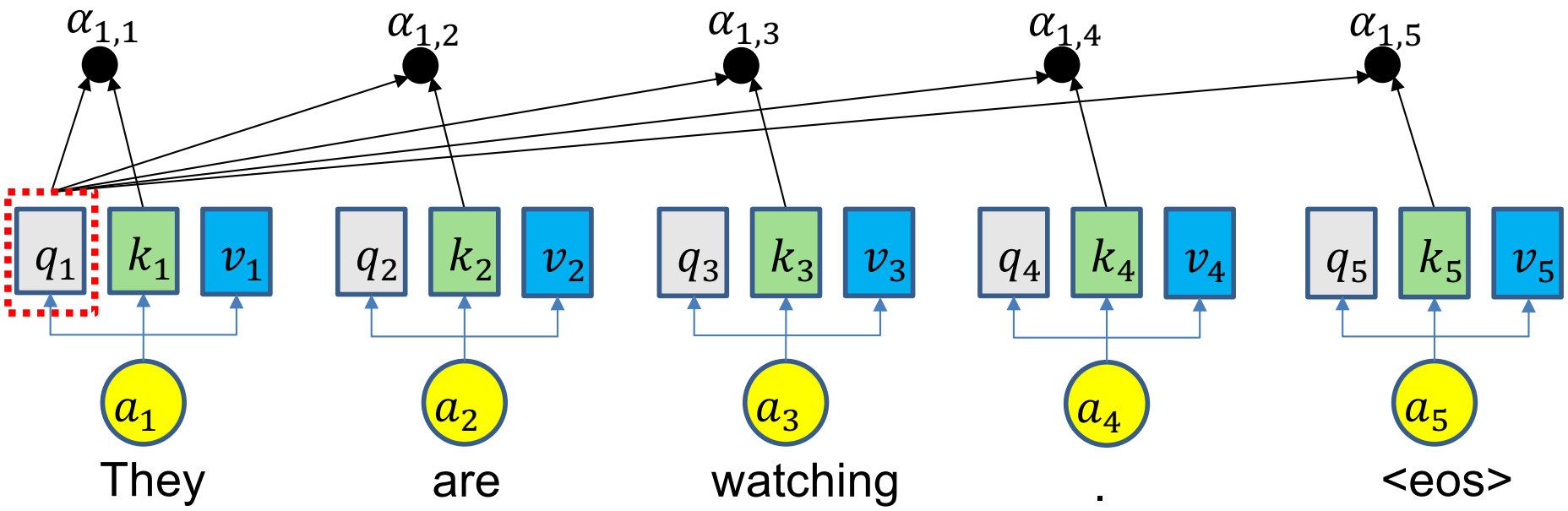


Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

Self-Attention

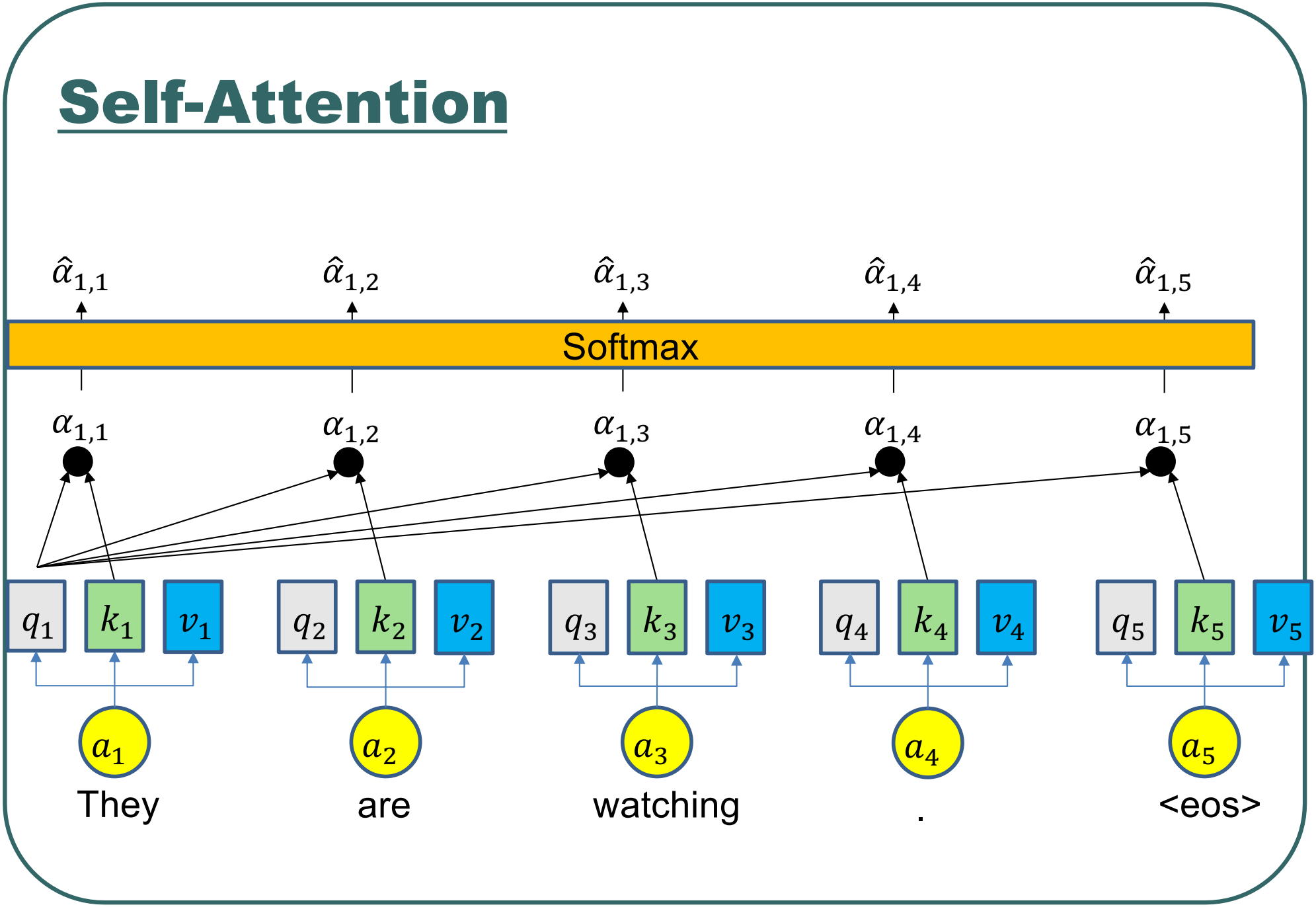
Calculate attention between each query and all keys

Scaled Dot-Product Attention: $\alpha_{1,i} = q_i \cdot k_i / \sqrt{d}$



Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

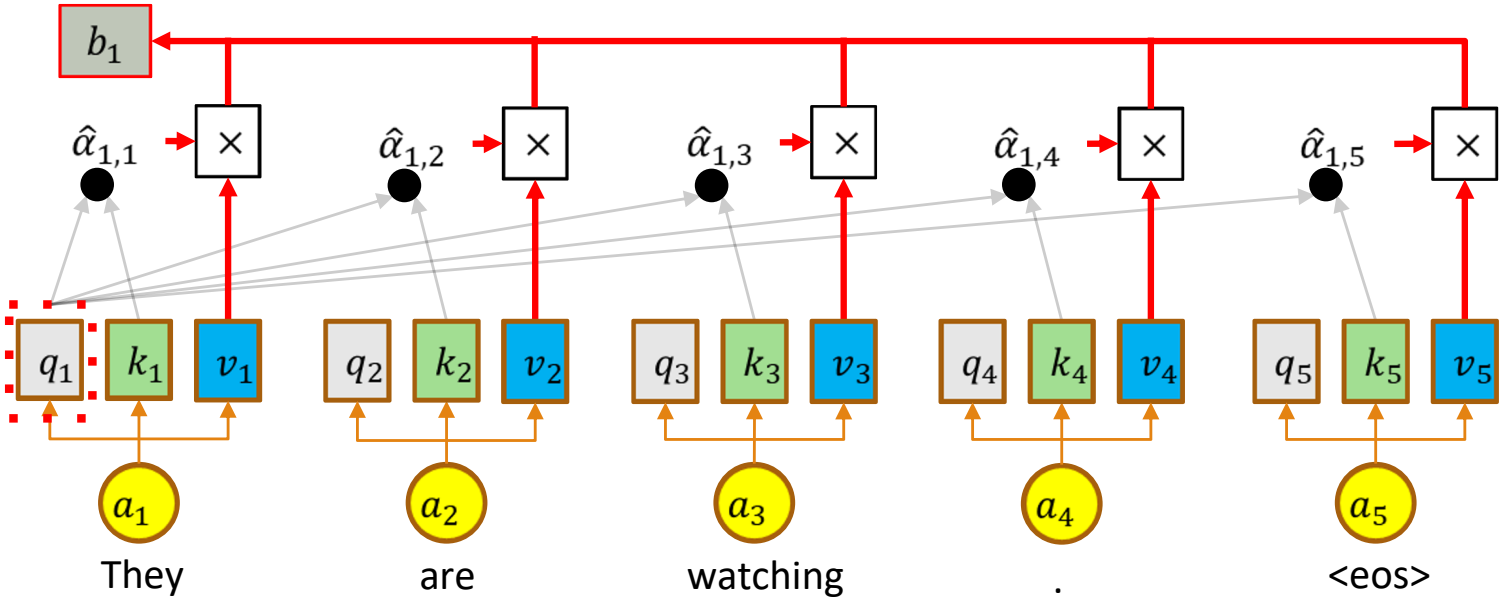
Self-Attention



Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

Self-Attention

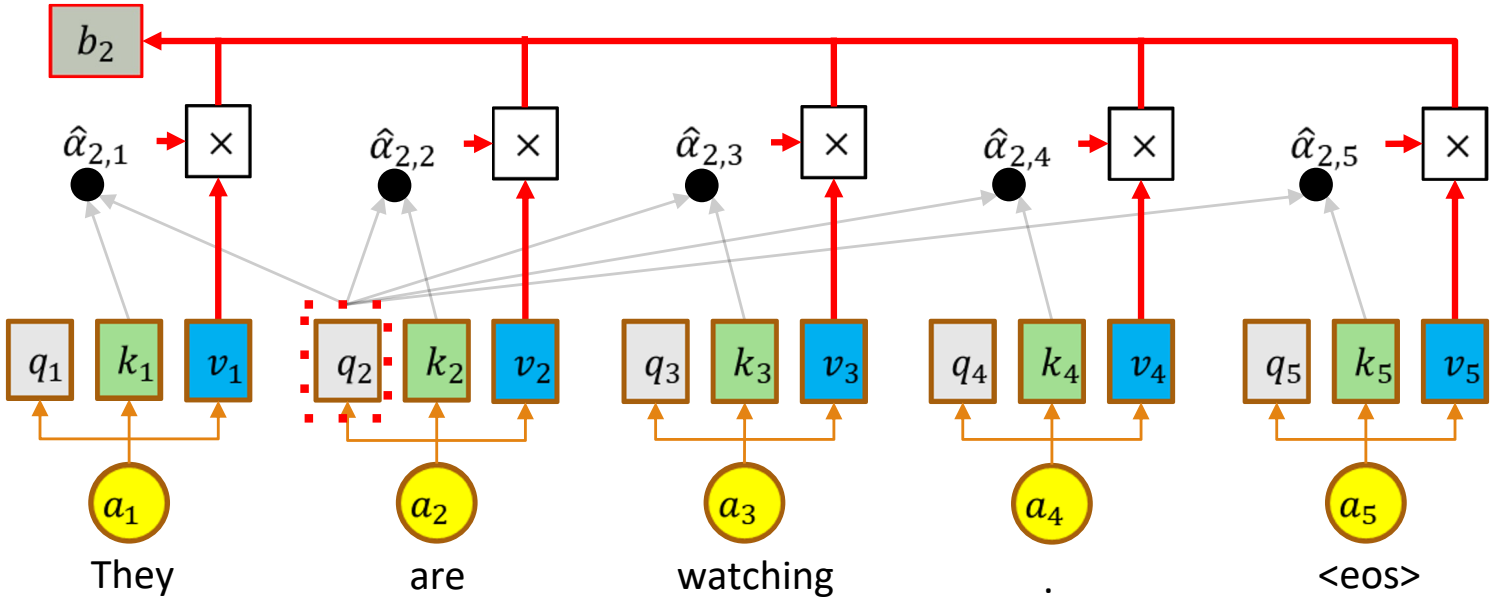
- Considering the whole sequence
- Calculate attention between each query and all keys
- $b_1 = \sum_i \hat{\alpha}_{1,i} v_i$



Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

Self-Attention

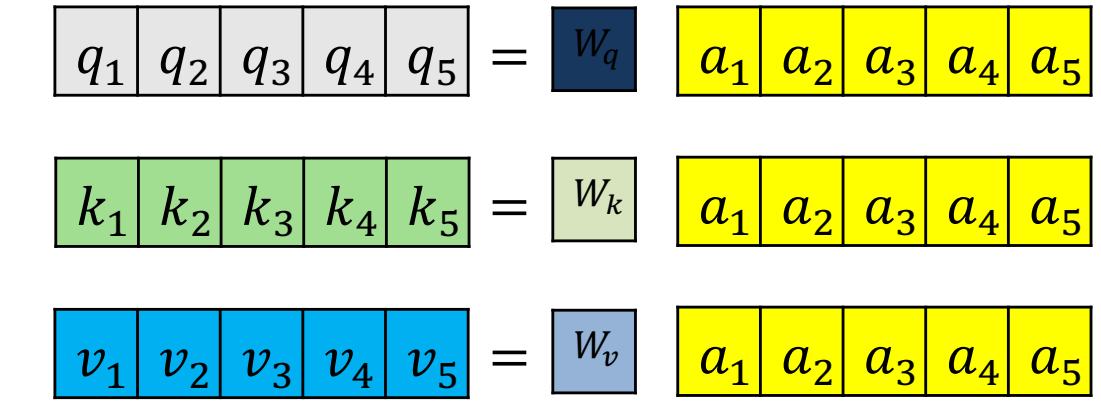
- Considering the whole sequence
- Calculate attention between each query and all keys
- $b_2 = \sum_i \hat{\alpha}_{2,i} v_i$

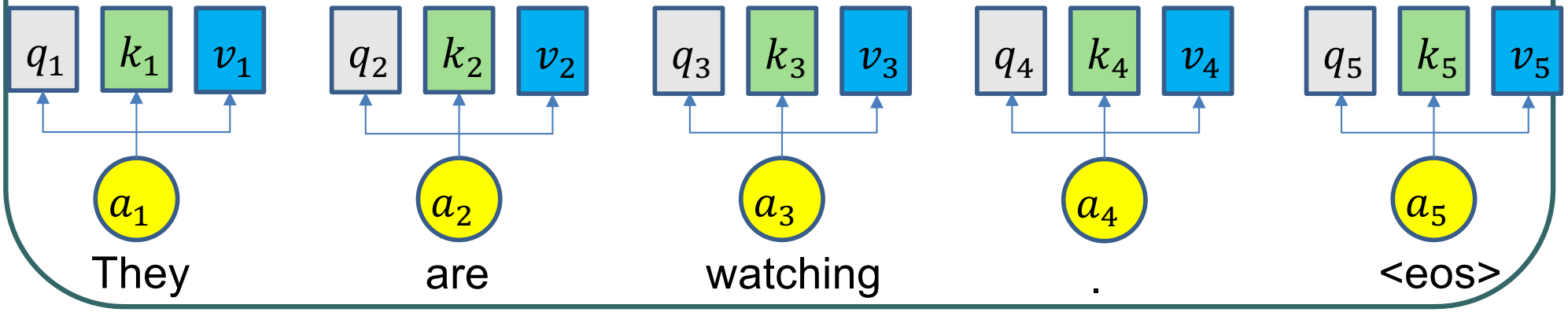


Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

How to parallelize

- $q_i = W_q a_i$
- $k_i = W_k a_i$
- $v_i = W_v a_i$

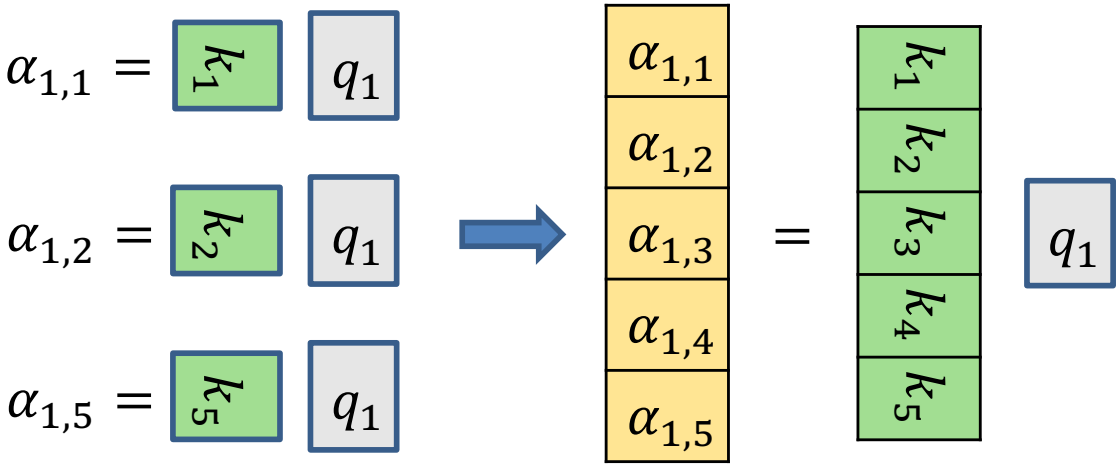


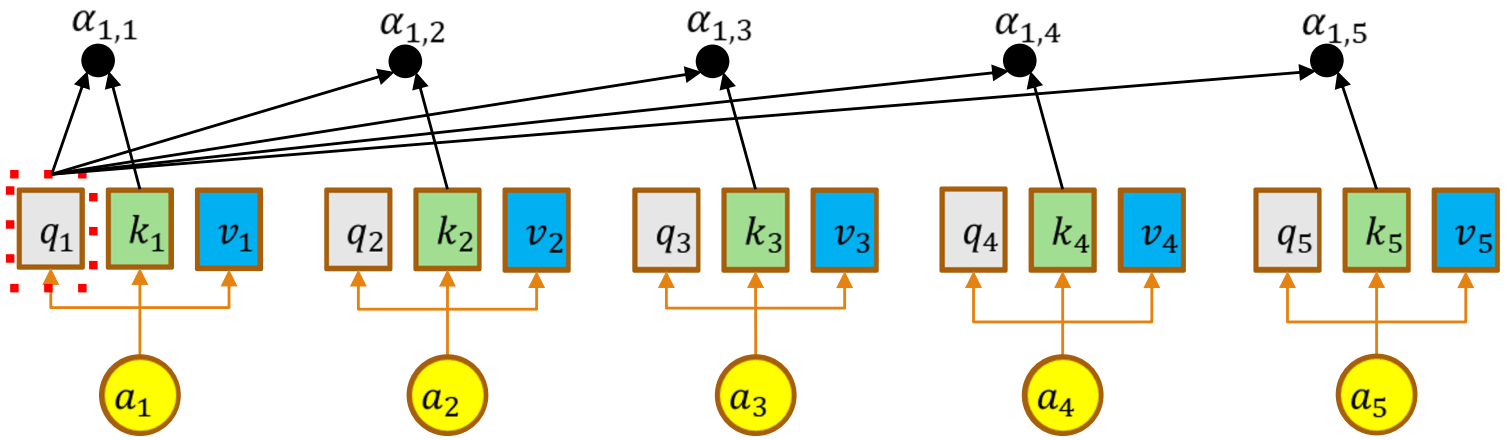


Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

How to parallelize

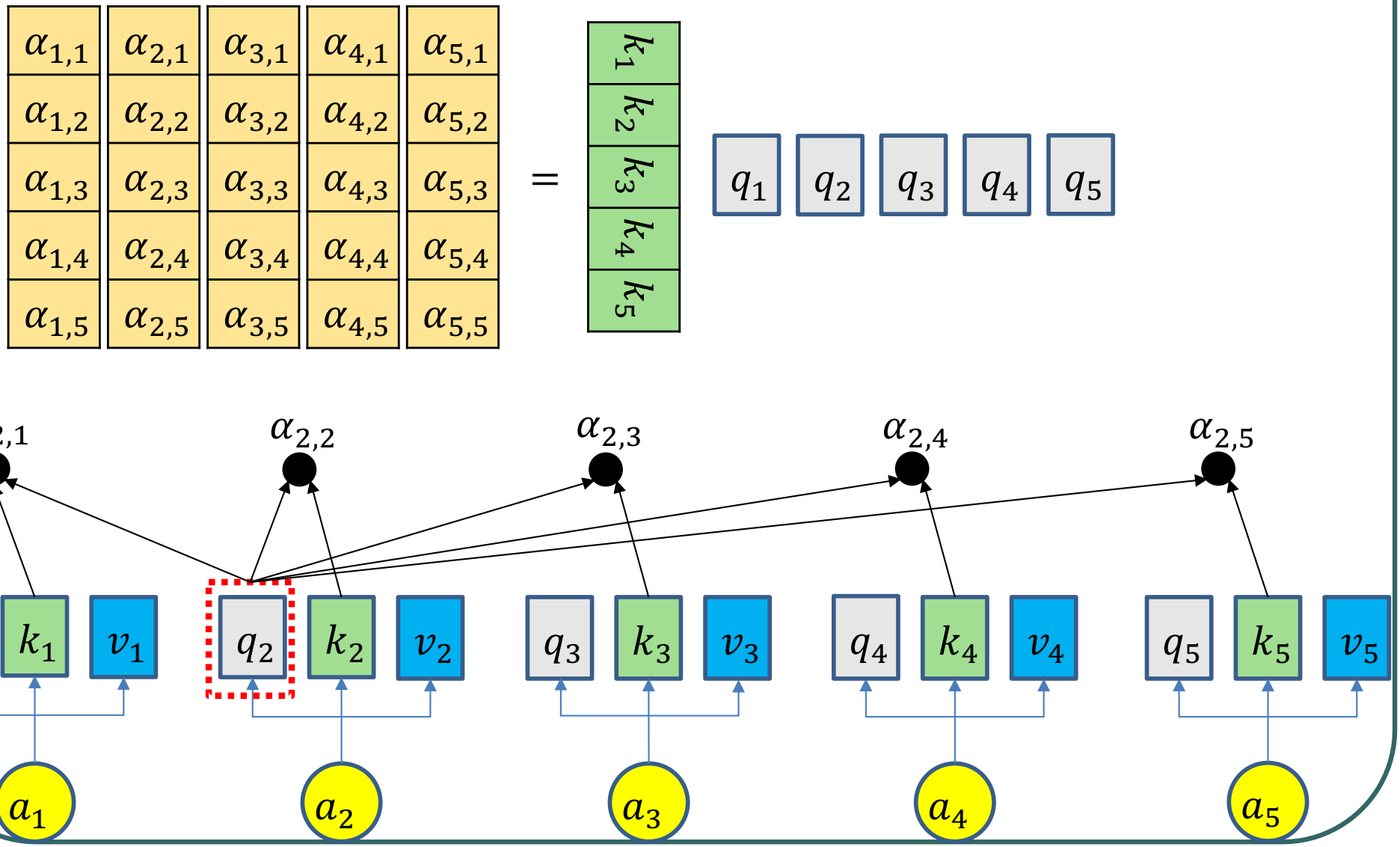
- $\alpha_{1,1} = k_1 \cdot q_1 / \sqrt{d}$
- $\alpha_{1,2} = k_2 \cdot q_1 / \sqrt{d}$
- $\alpha_{1,5} = k_5 \cdot q_1 / \sqrt{d}$





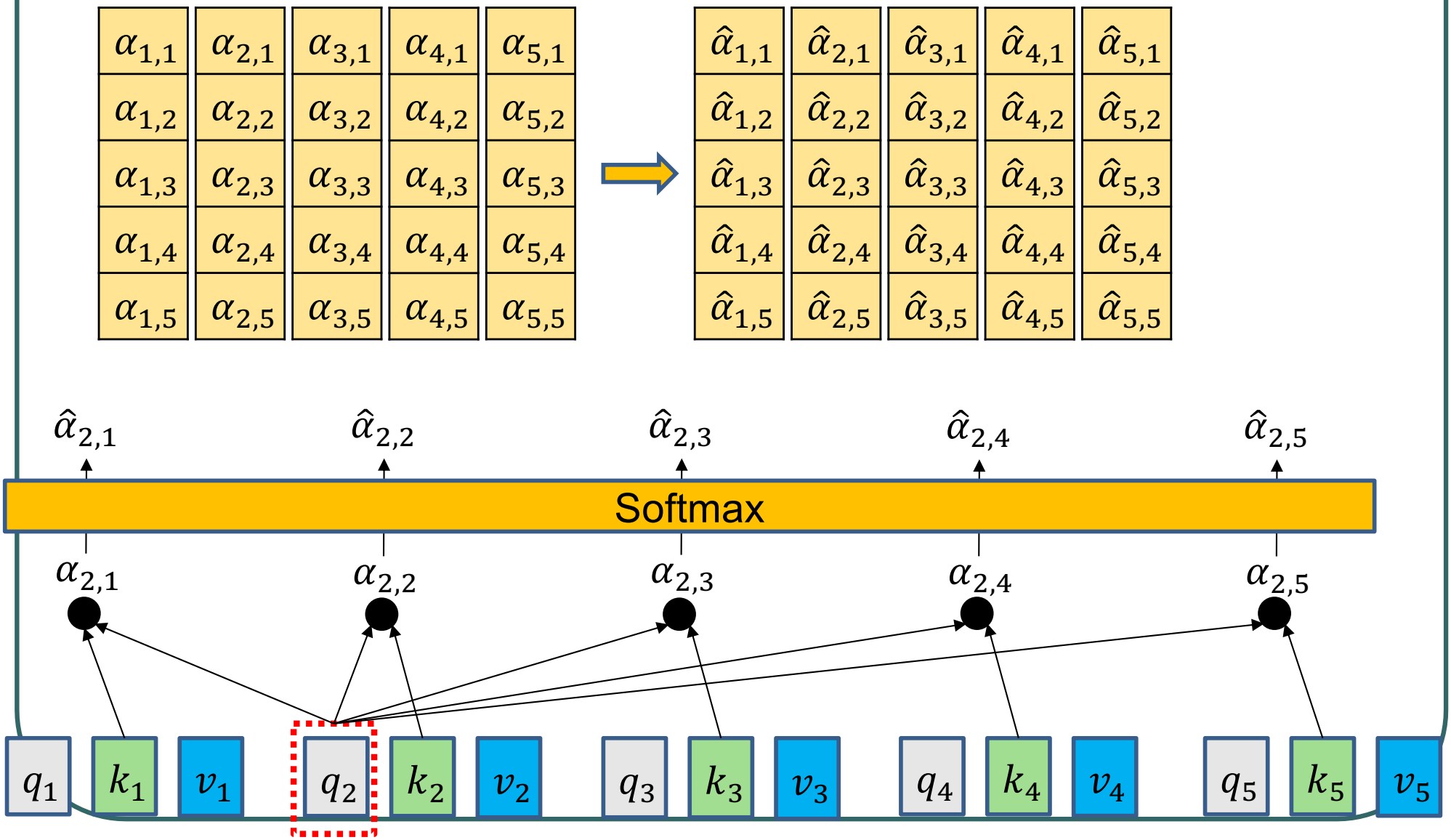
Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

How to parallelize



Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

How to parallelize



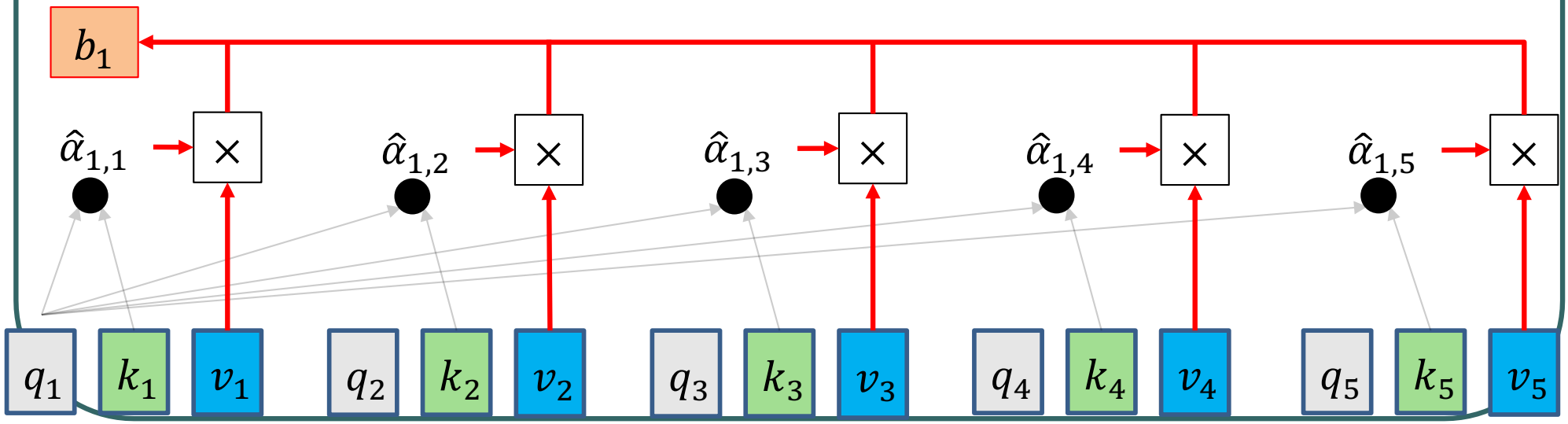
Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

How to parallelize

- $b_1 = \sum_i \hat{\alpha}_{1,i} v_i$

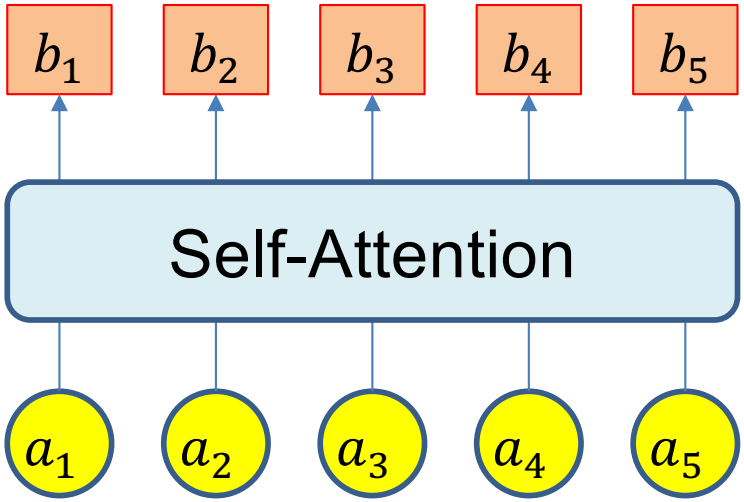
b_1 b_2 b_3 b_4 b_5 = v_1 v_2 v_3 v_4 v_5

$\hat{\alpha}_{1,1}$	$\hat{\alpha}_{2,1}$	$\hat{\alpha}_{3,1}$	$\hat{\alpha}_{4,1}$	$\hat{\alpha}_{5,1}$
$\hat{\alpha}_{1,2}$	$\hat{\alpha}_{2,2}$	$\hat{\alpha}_{3,2}$	$\hat{\alpha}_{4,2}$	$\hat{\alpha}_{5,2}$
$\hat{\alpha}_{1,3}$	$\hat{\alpha}_{2,3}$	$\hat{\alpha}_{3,3}$	$\hat{\alpha}_{4,3}$	$\hat{\alpha}_{5,3}$
$\hat{\alpha}_{1,4}$	$\hat{\alpha}_{2,4}$	$\hat{\alpha}_{3,4}$	$\hat{\alpha}_{4,4}$	$\hat{\alpha}_{5,4}$
$\hat{\alpha}_{1,5}$	$\hat{\alpha}_{2,5}$	$\hat{\alpha}_{3,5}$	$\hat{\alpha}_{4,5}$	$\hat{\alpha}_{5,5}$

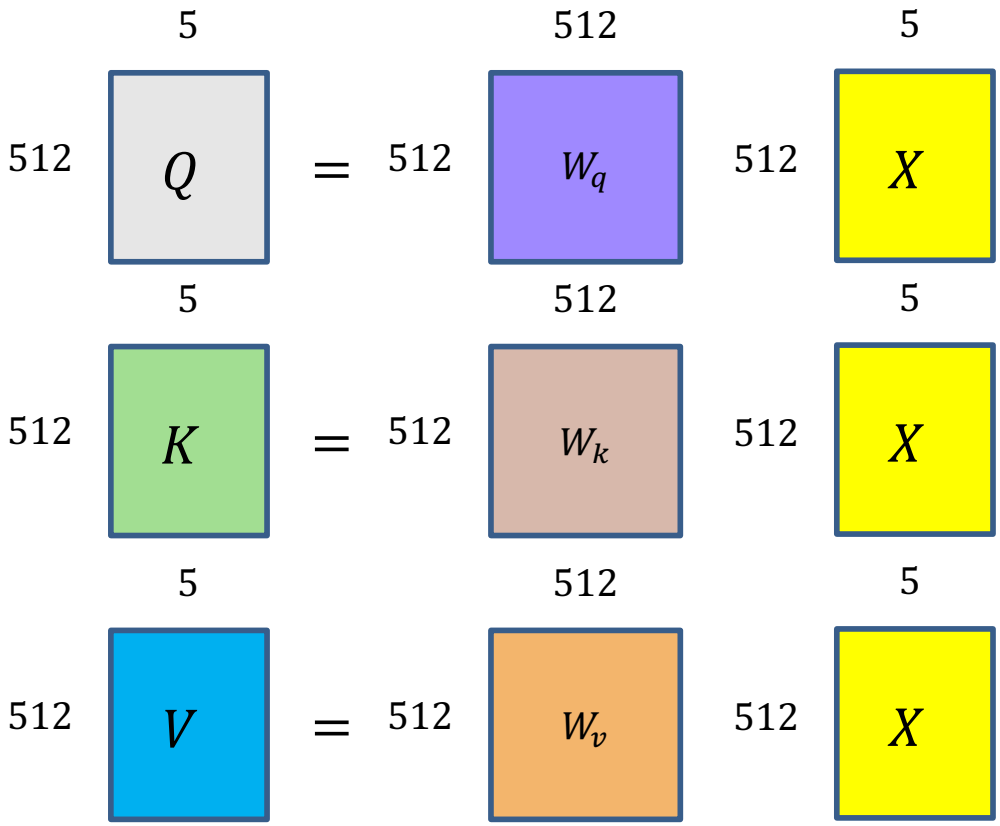


Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

How to parallelize

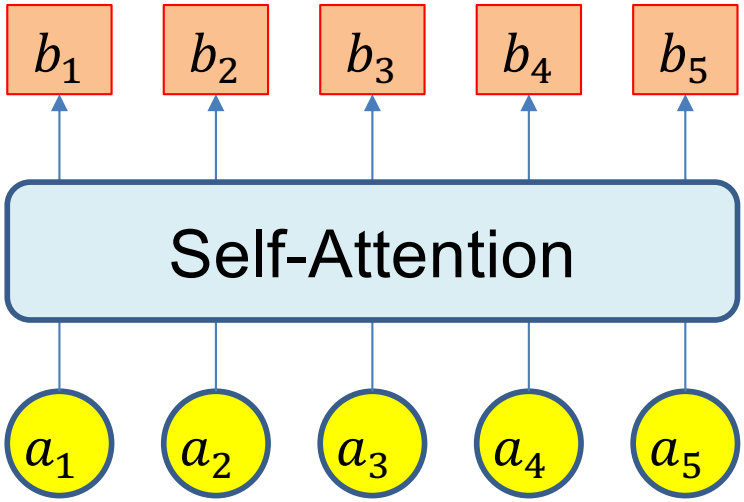


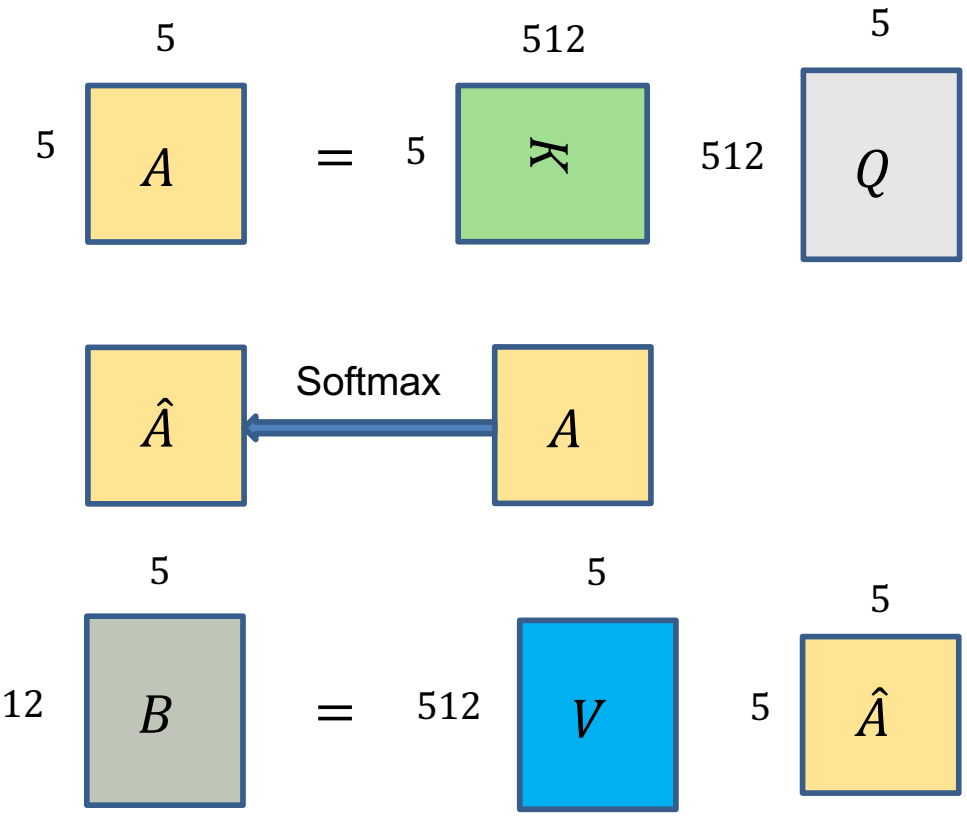
- $X \in R^{d \times N}$
  - $d = 512$: dimension of embedding
  - $N = 5$: length of input sequence



Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

How to parallelize



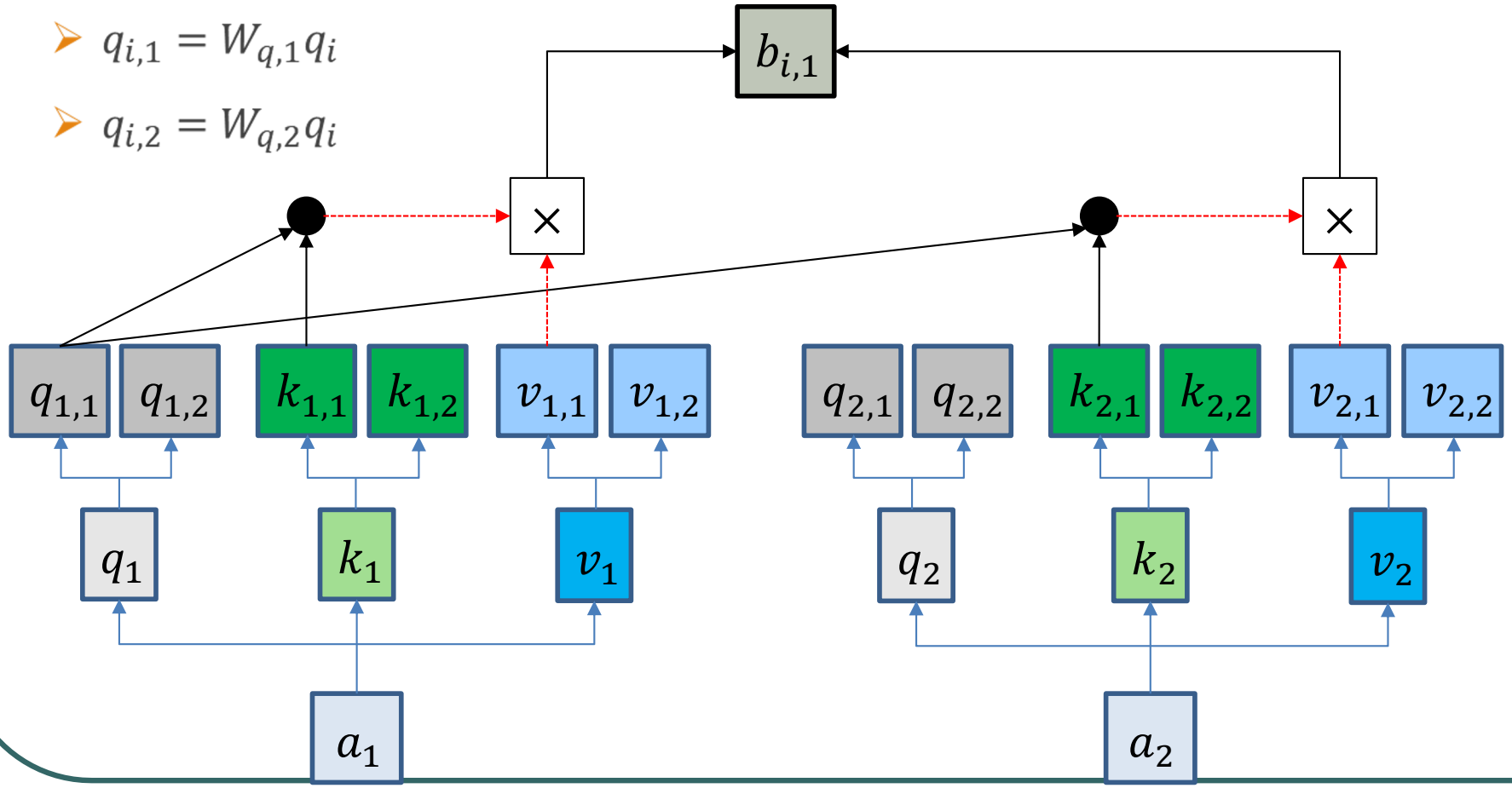


- $X \in R^{d \times N}$
  - $d = 512$: dimension of embedding
  - $N = 5$: length of input sequence

Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

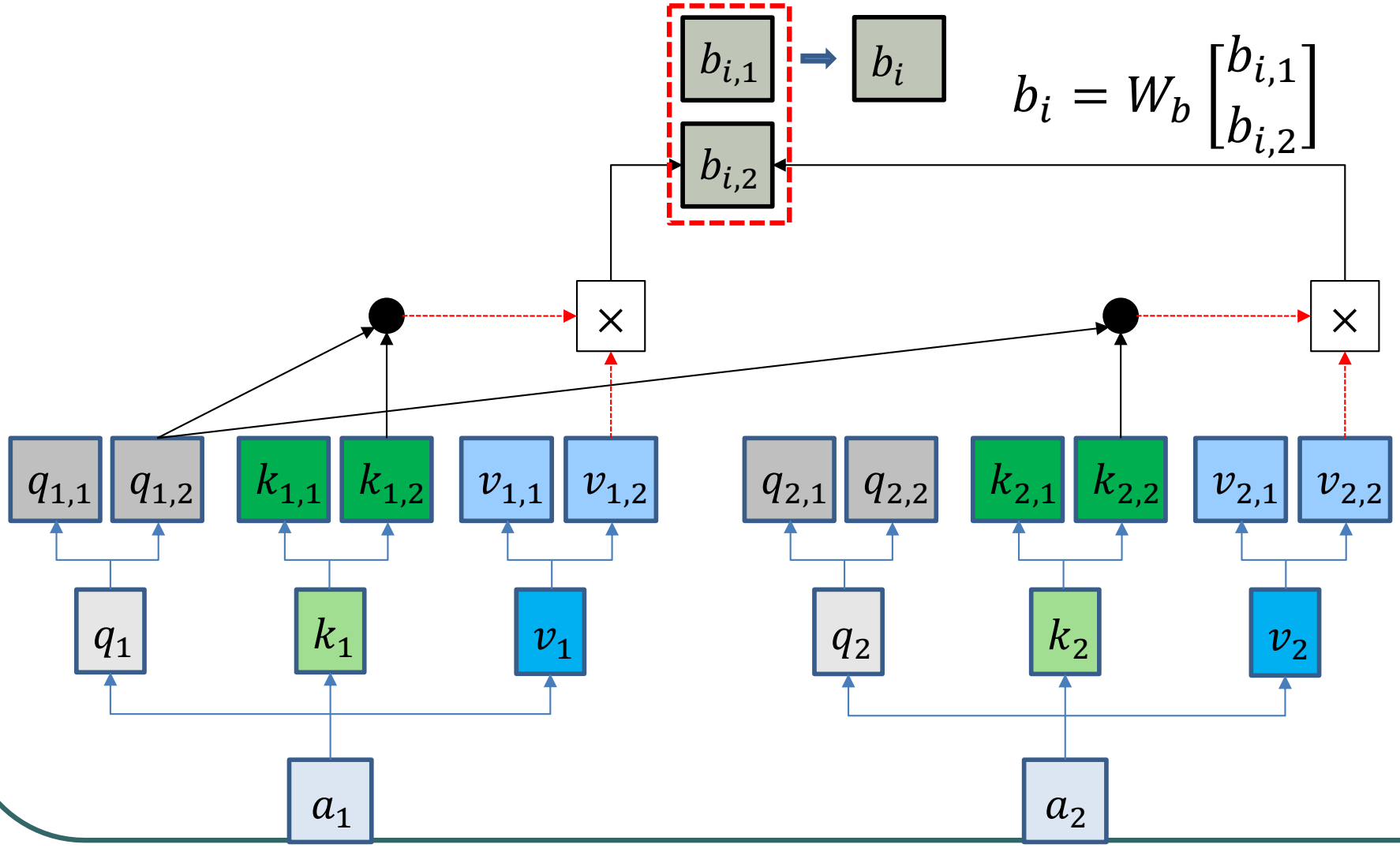
Multi-Head Attention (2 heads as example)

- $q_{i,1} = W_{q,1} q_i$
- $q_{i,2} = W_{q,2} q_i$



Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

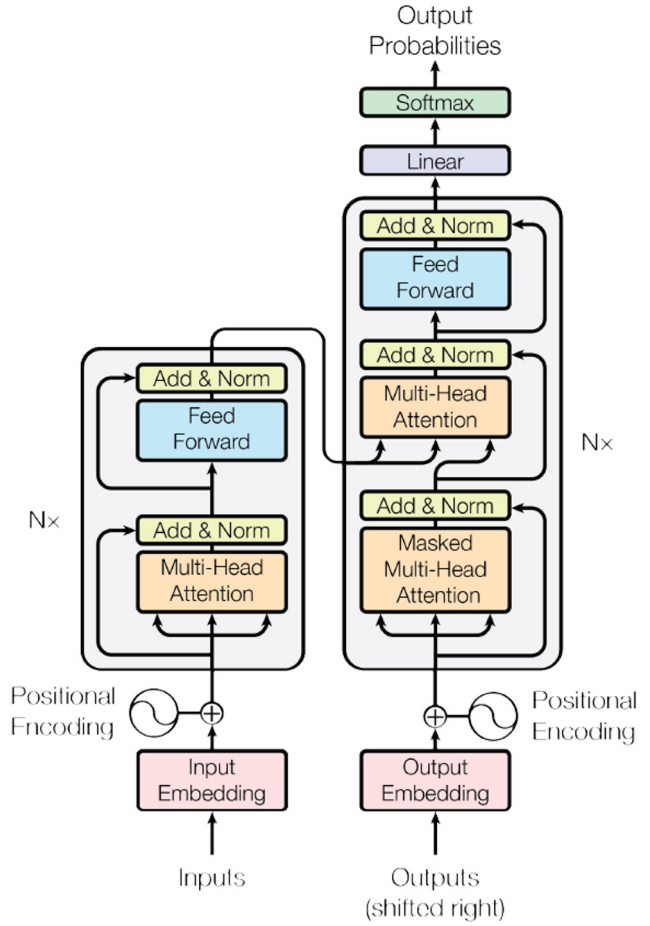
Multi-Head Attention (2 heads as example)



Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

Multi-Head Attention

- Attention helps the model to focus on important words in a given input sentence.
- Simple attention selectively focuses on words concerning some external queries, the more important the words, the more focus it has given.
- Self-Attention takes the relationship among words within the same sentence into account.

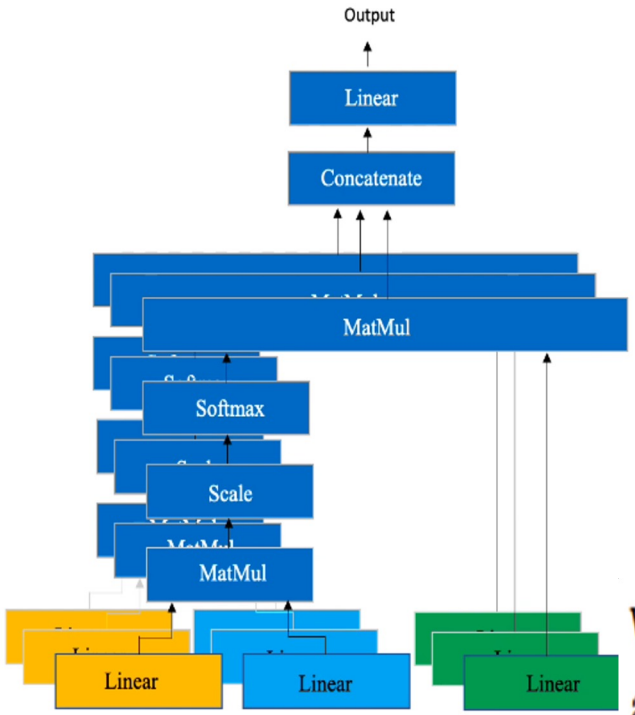


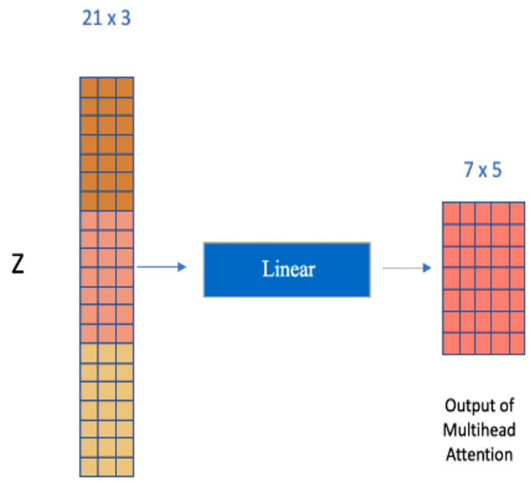
Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

Multi-Head Attention

Transformers learn multiple attention filters each focusing on a different phenomenon.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$





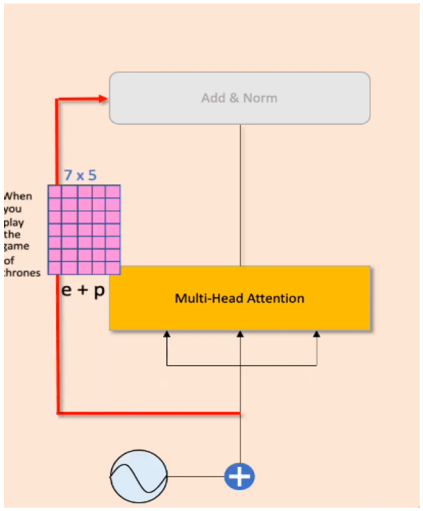
In the "Attention Is All You Need" paper
The author employ h = 8 parallel attention layers, or heads.

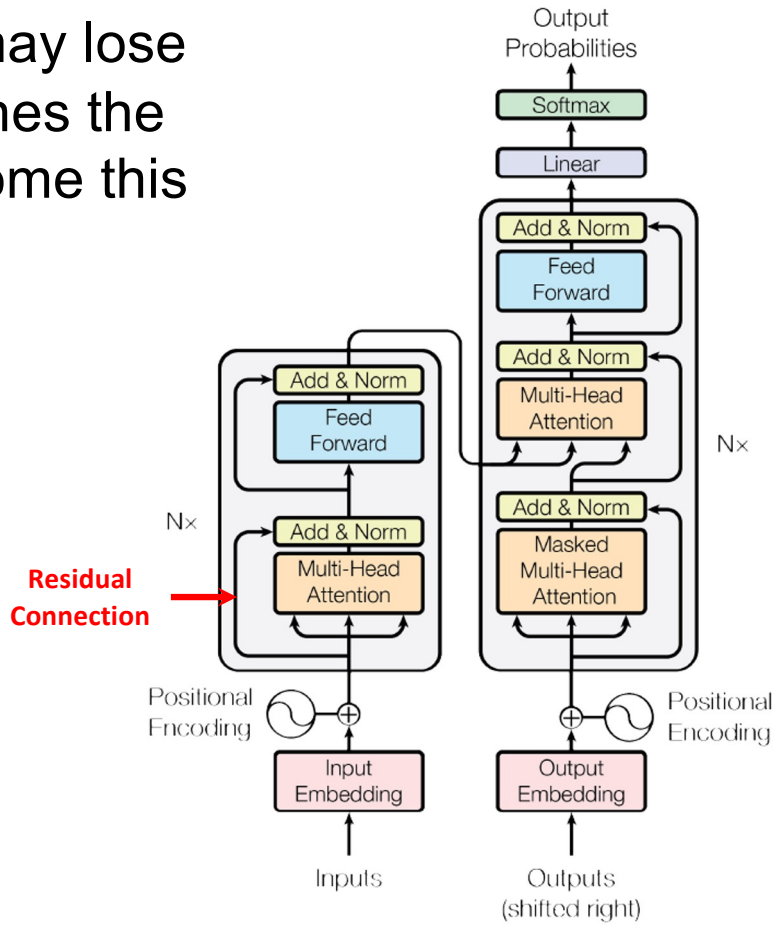
Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

Residual Connection

- Helps with knowledge preservation and vanishing gradient problem
- During the forward propagation, the input may lose some useful information by the time it reaches the last layer. Residual connections can overcome this problem.
- Residual link passes the output of the word embedding layer

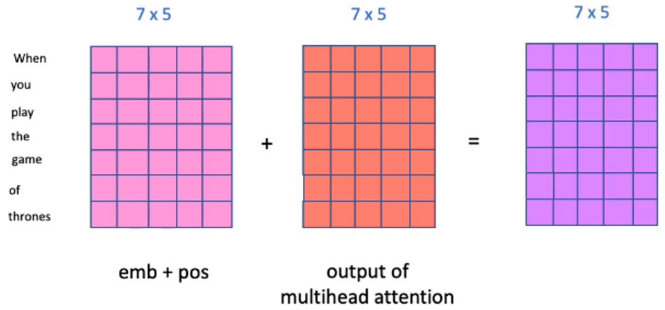




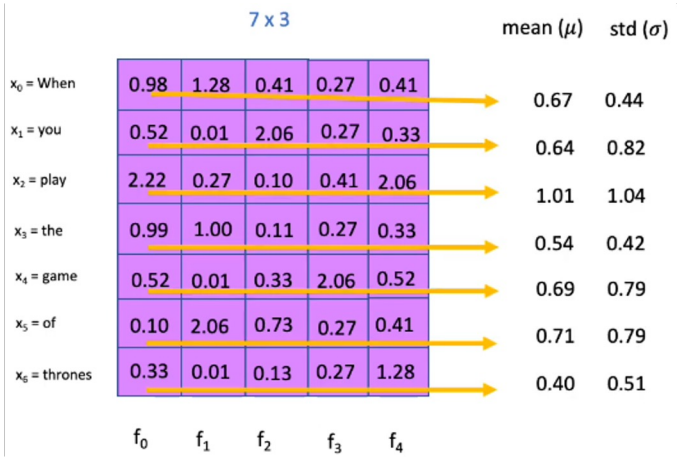
Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

Add & Norm layer

- This layer adds the two inputs:
  - Embedding+position encoding
  - Output of multihead attention

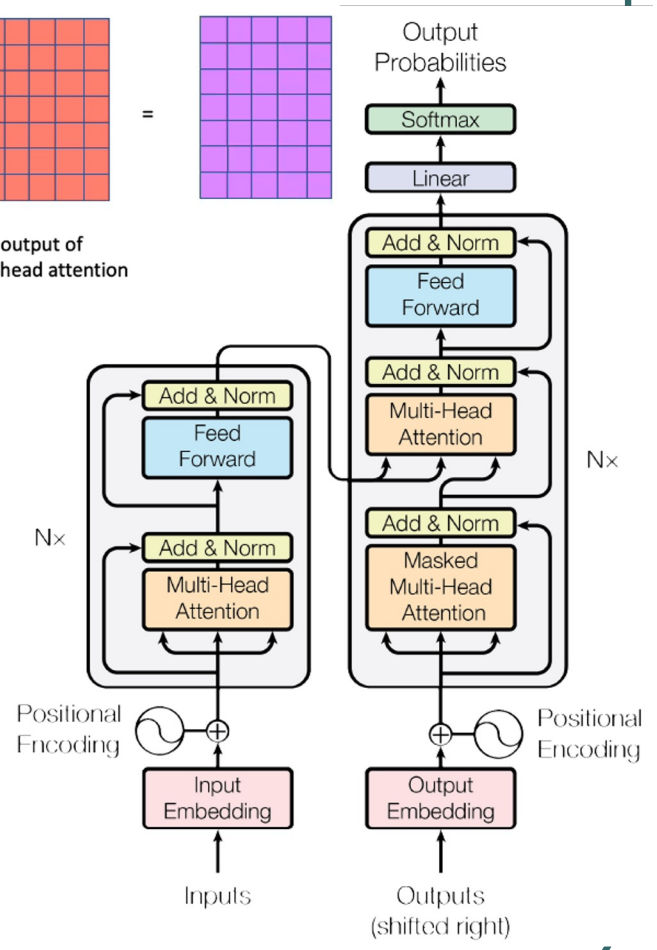


- Layer normalization: standardizing the neuron activations along the axes of the features.



$$x_i = \frac{x_i^d - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

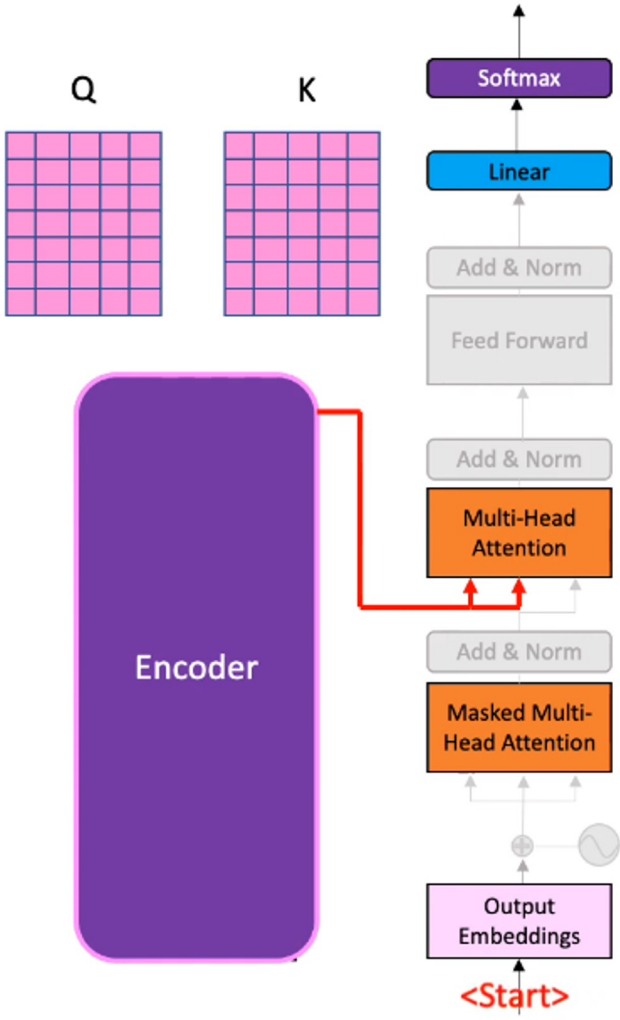
Epsilon is a small number to avoid division



Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

Decoder Components

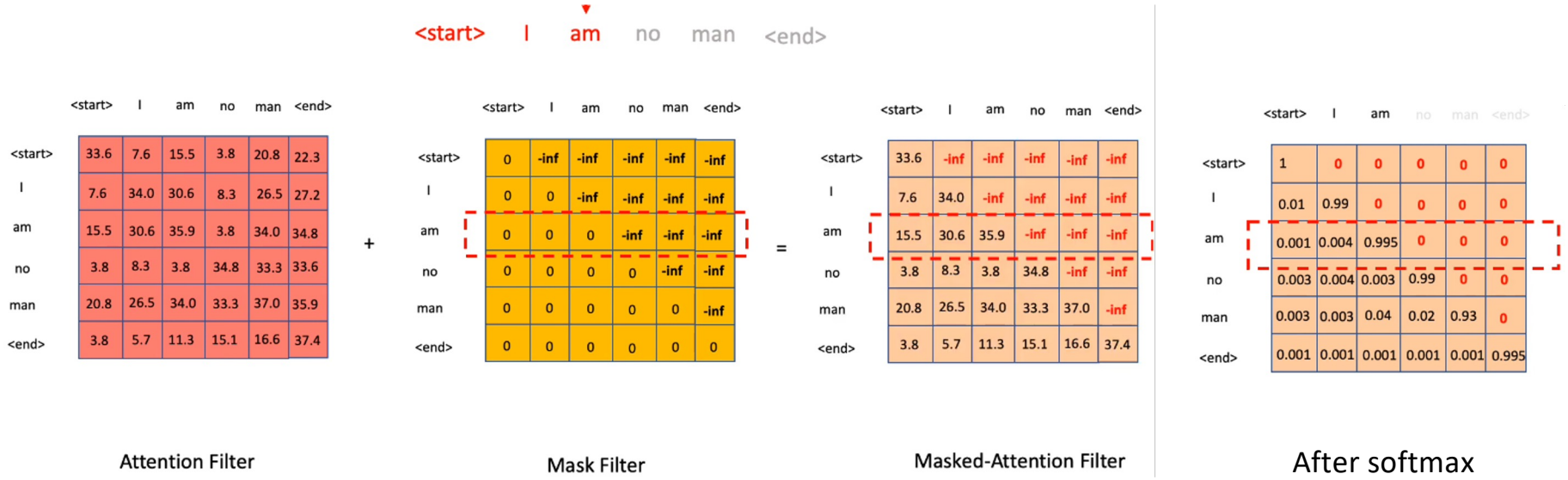
- While the encoder takes just one input, the decoder takes two:
  1. Output of the Encoder (split into 2 copies for Query and Key)
  2. Output text that has been generated (the first one is a special token indicating the start of sentence)
- Masked Multi-Head Attention is used during the Training Phase to mask the prediction of target.
- Once the input text are passed into the transformer, the decoder generates its first text, then the actual target get unmasked.
- The loss is computed using the Cross Entropy-Loss



Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

Masking

- Just before the attention matrix is passed into the softmax layer, the masking operation is performed.
- Masking is a filter matrix with all the future words having the score of "– infinity".
- The model is paying attention to the input as well as all the target tokens up to the Nth token.



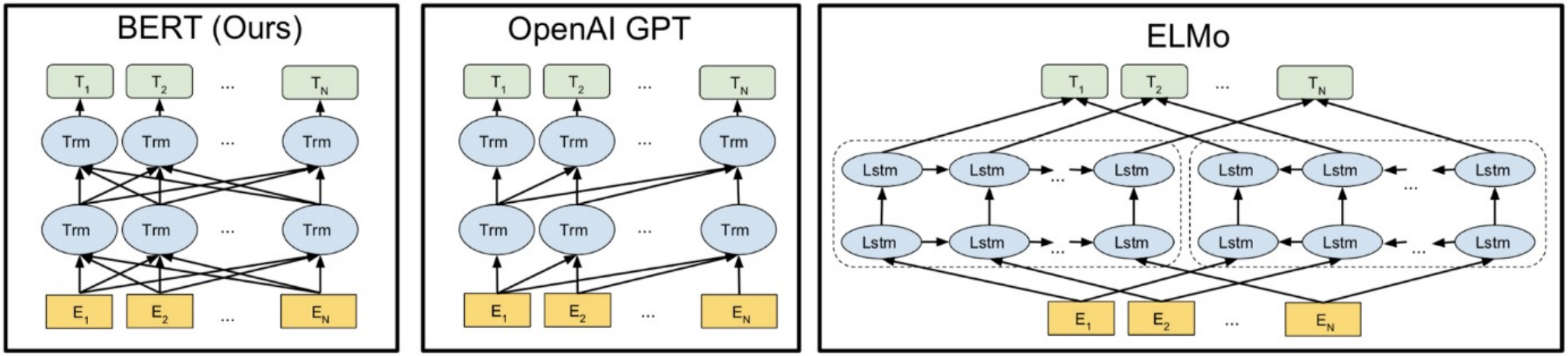
Vaswani et al.," Attention Is All You Need", *in NIPS*, 2017.

BERT

Jacob et al.," BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *in NAAL*, 2019.

Limitations of current techniques

- Language models in pre-training are unidirectional, and the power of the pre-trained representation is limited.
  - OpenAI GPT used left-to-right architecture
  - ELMo concatenates *forward* and *backward* language models



Jacob et al.," BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *in NAAL*, 2019.

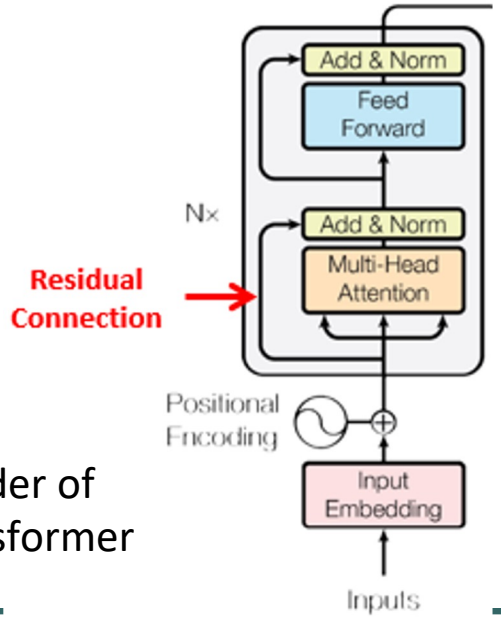
BERT: Biderectional Encoder Representations from Transformers

- Main ideas
  - Propose a new pre-training objective so that a deep bidirectional transformer can be trained
    - The "masked language model"(MLM): the objective is to predict the original word of a masked word based only on its context
    - "Next sentence prediction"
- Merits of BERT
  - BERT model can be fine-tuned for specific tasks to achieve SoTA performance
  - BERT advances the SoTA for eleven NLP tasks

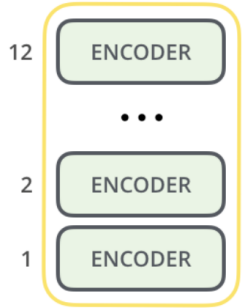
Jacob et al.," BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *in NAAL*, 2019.

Model Architecture

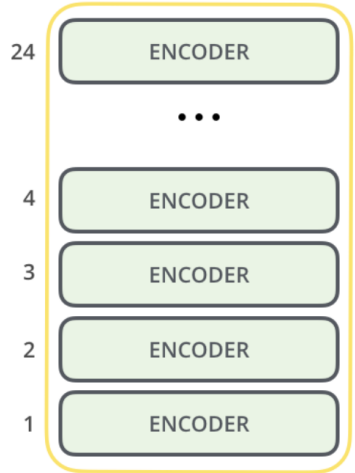
- BERT's model architecture is a multi-layer bidirectional Transformer encoder.
- In BERT experiments, the number of blocks N was chosen to be 12 and 24.
- Blocks do not share weights with each other.



Encoder of Transformer



$BERT_{BASE}$



$BERT_{LARGE}$

Jacob et al.," BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *in NAAL*, 2019.

Task#1 Masked LM

- 15% of the words are randomly masked
  - The task is to predict the masked words based on its left and right context.
- Not all tokens were masked in the same way (example sentence “My dog is hairy”)
  - 80% were replaced by the <MASK> token: ”My dog is <MASK>”
  - 10% were replaced by a random token: “My dog is apple”
  - 10% were left intact: “My dog is hairy”

Jacob et al.,” BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, *in NAAL*, 2019.

Task#2 Next Sentence Prediction

- Motivation
  - Many downstream tasks are based on understanding the relationship between two text sentences
    - Question Answering (QA)
  - Language modeling does not directly capture that relationship
- The task is binarized next sentence prediction.
  - Input = [CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]
  - Label = isNext
  - Input = [CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]
  - Label = NotNext

Jacob et al.," BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *in NAAL*, 2019.

Pre-training procedure

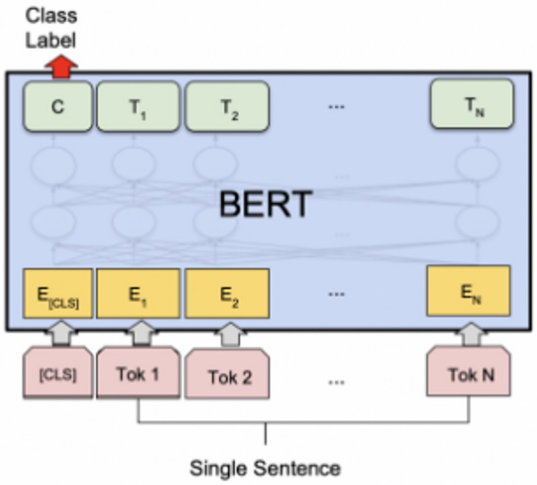
- Training data: BooksCorpus (800M words) + English Wikipedia (2500M words)
- To generate each training input sequences: sample two spans of text (A and B) from the corpus
  - The combined length is <= 500 tokens
  - 50% B is the actual next sentence that follows A and 50% of the time it is a random sentence from the corps.
- The training loss is the sum of the mean masked LM likelihood and the mean next sentence prediction likelihood.

Jacob et al.," BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *in NAAL*, 2019.

Fine-tuning procedure

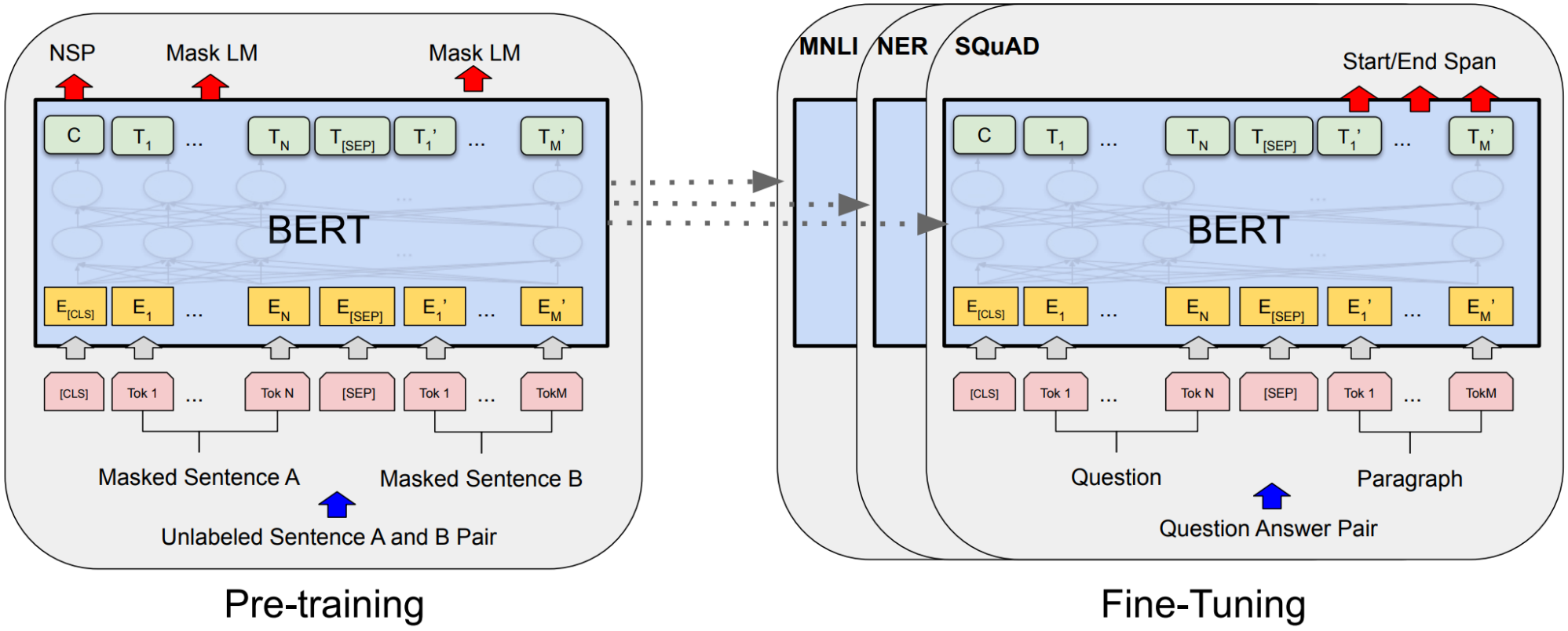
- For sequence-level classification task
  - Obtain the representation of the input sequence by using the final hidden state (hidden state at the position of the special token [CLS]) C $\in R^H$
  - Just add a classification layer and use softmax to calculate label probabilities. Parameters $W \in R^{K*H}$

P = softmax(CW^T)



Jacob et al.," BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *in NAAL*, 2019.

Pre-training and fine-tuning for BERT



Jacob et al.," BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *in NAAL*, 2019.

Vision Transformer (ViT)

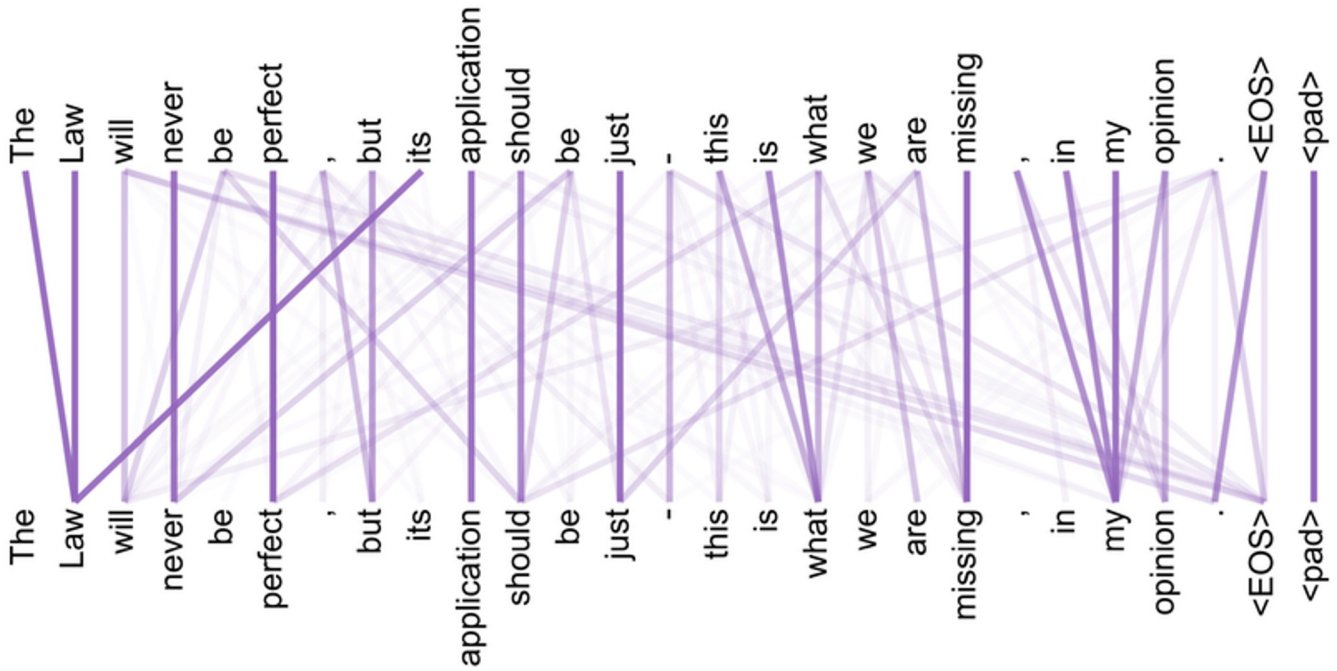
Alexey et al.,” An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, *in ICLR*, 2021.

Why don’t we use a full image for transformer?

Because complexity

We need to store n^2 parameters

$(3 * 224 * 224)^2$ = 22 billion parameters



Source: https://tugot17.github.io/Vision-Transformer-Presentation/#/13

Overview

- Divide an input image into 196 (14x14) small image patchs of size (16x16)
- Treat each patch as word embedding in NLP
- Use it as an input for traditional transformer encoder (like in BERT)
- Use 12 transformer layers (Norm, Multi-head attention, etc.)
- Take the last output, use it as input for Dense Layer with 1000 classes
- Voilà - you have a classification model

Alexey et al.,” An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, *in ICLR*, 2021.

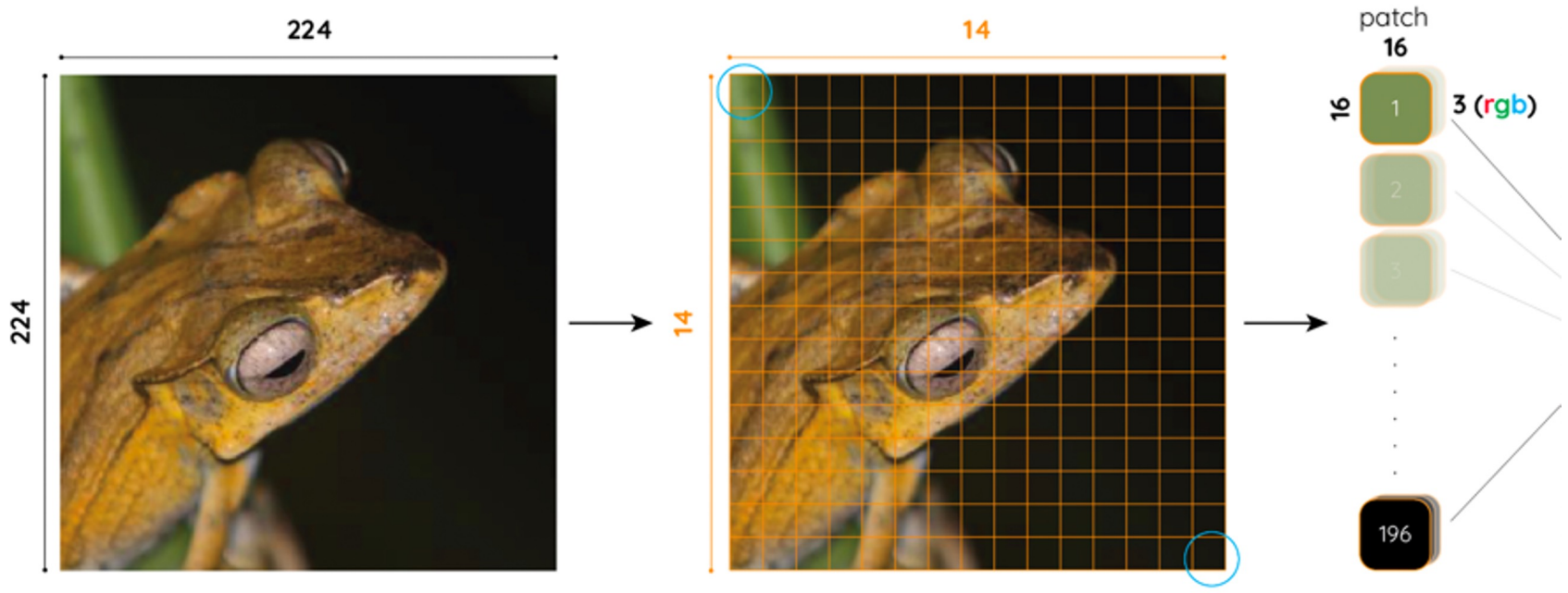
Overview

Source: https://tugot17.github.io/Vision-Transformer-Presentation/#/13

Alexey et al.,” An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, *in ICLR*, 2021.

Patch Embedding

Image of size (224,224,3)
Divided into 196 (14×14) patches of size 16×16x3



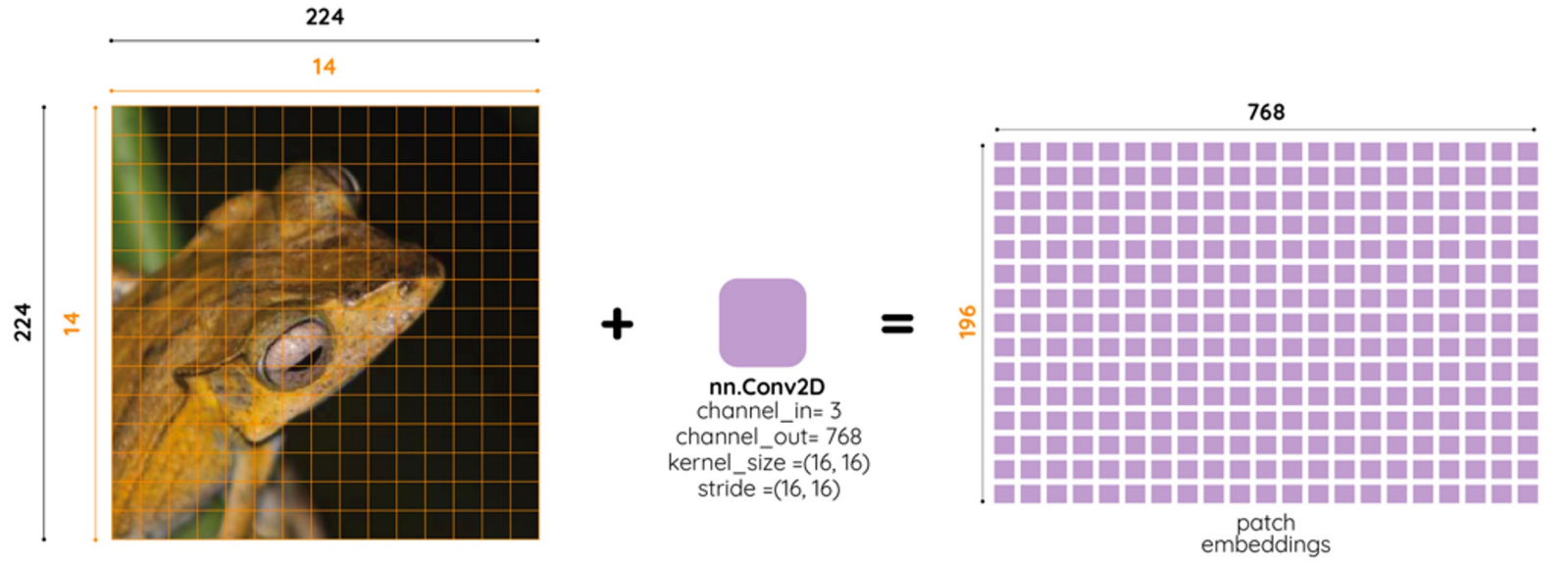
Source: https://tugot17.github.io/Vision-Transformer-Presentation/#/13

Alexey et al.,” An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, *in ICLR*, 2021.

Patch Embedding

Each patch is converted into a vector of size (3*3*16=768)

After that, we have 196 vectors of size 768, a matrix of size (196,768)

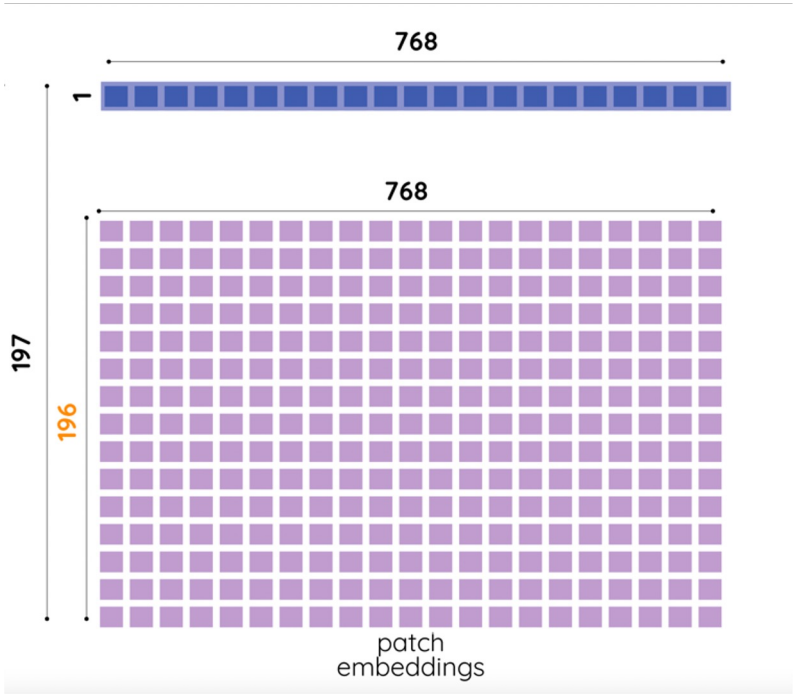


Source: https://tugot17.github.io/Vision-Transformer-Presentation/#/13

Alexey et al.,” An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, *in ICLR*, 2021.

[CLS] Token

Similarly to the situation in BERT we need to add a [CLS] token [CLS] token is a vector of size (1,768).
The final patch matrix has size (197,768), 196 from patches and 1 [CLS] token.

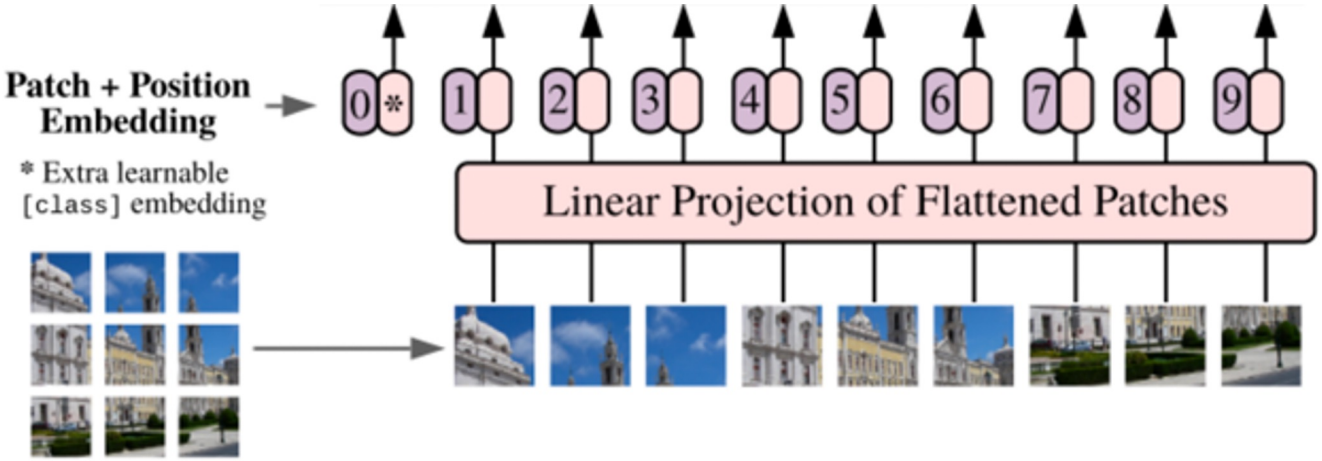


Source: https://tugot17.github.io/Vision-Transformer-Presentation/#/13

Alexey et al.,” An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, *in ICLR*, 2021.

Position Embedding

They used standard learnable 1D position embeddings and the resulting sequence of embedding vectors served as input to the encoder



Source: https://tugot17.github.io/Vision-Transformer-Presentation/#/13

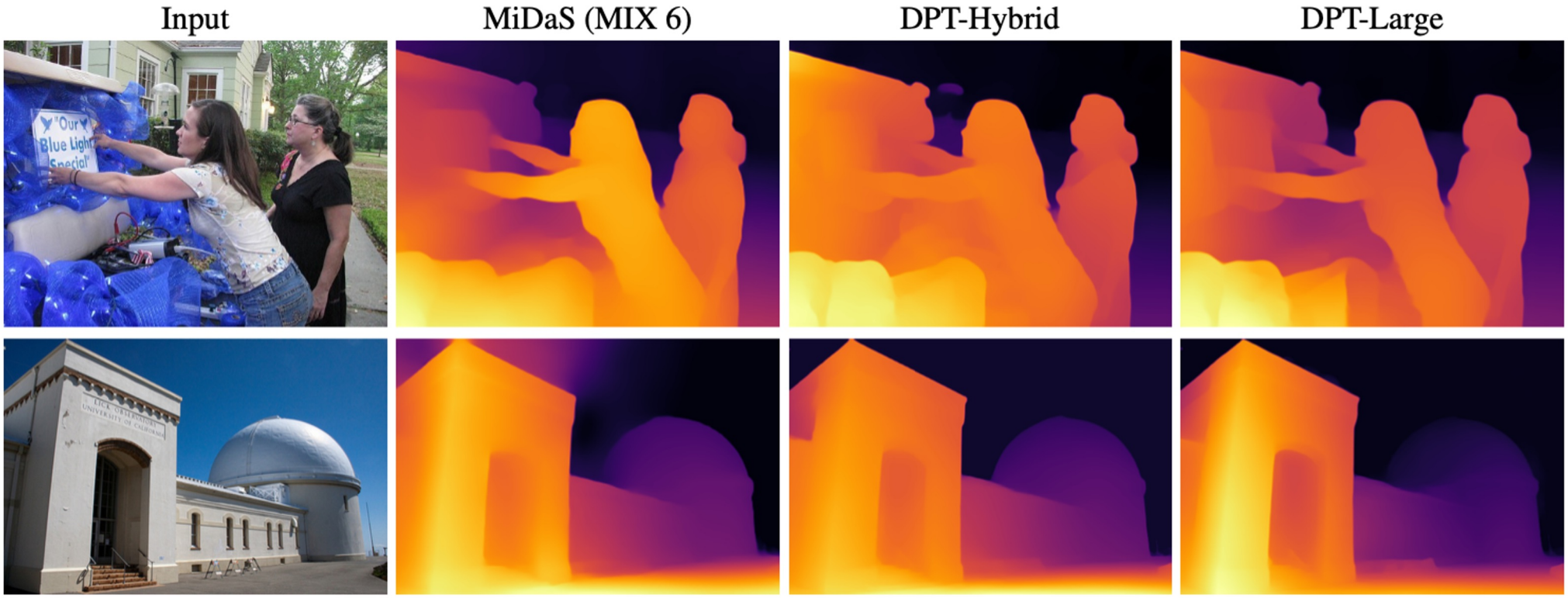
Alexey et al.,” An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, *in ICLR*, 2021.

Vision Transformers for Dense Prediction

Ranftl et al. “Vision Transformers for Dense Prediction” *in ICCV*, 2021.

Dense Prediction with ViT

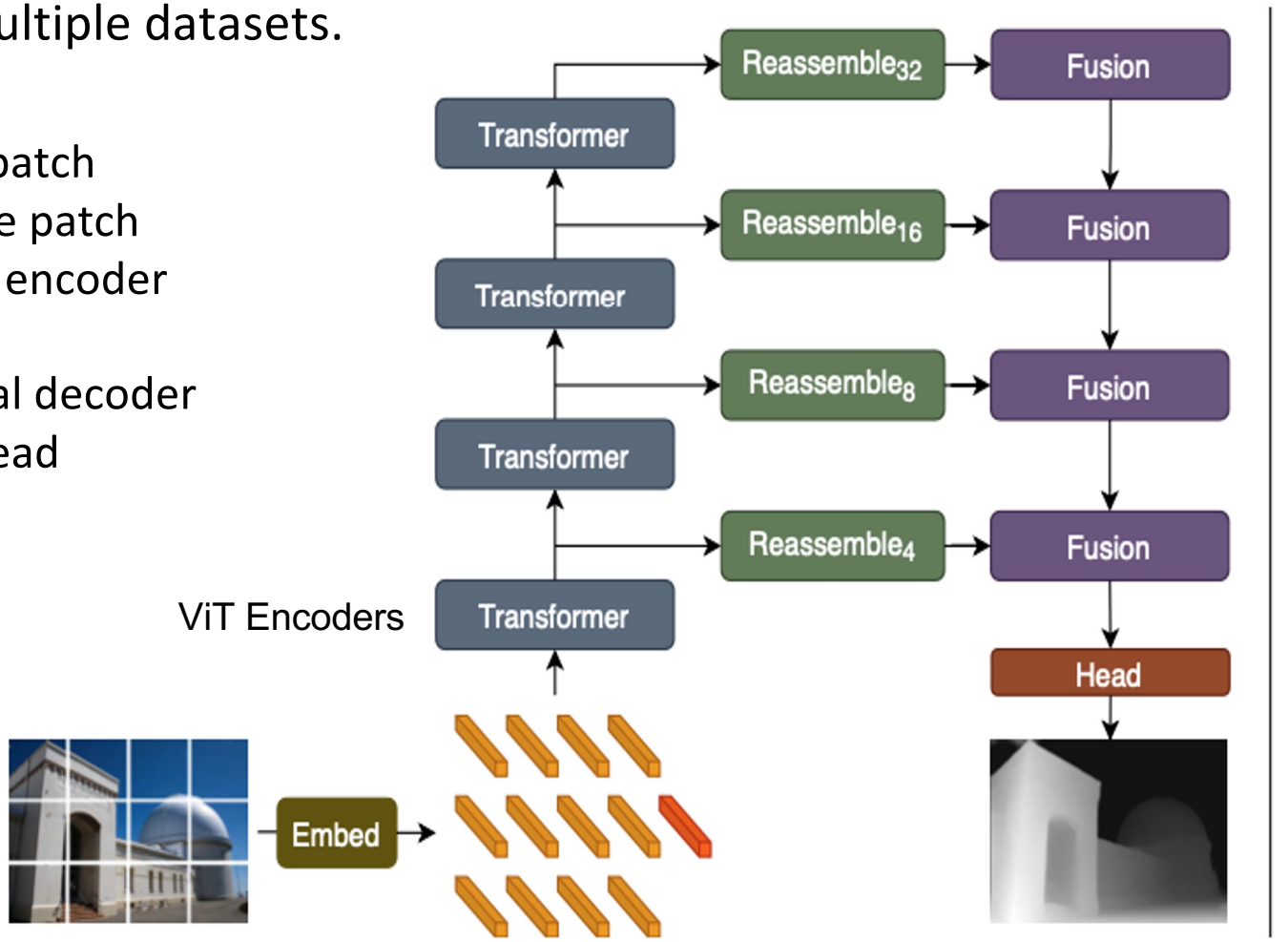
Ranftl et al. [1] leverages vision transformers in place of convolution neural networks as a backbone for dense prediction task, especially monocular depth estimation and semantic segmentation.



Ranftl et al. “Vision Transformers for Dense Prediction” *in ICCV*, 2021.

Dense Prediction Transformer

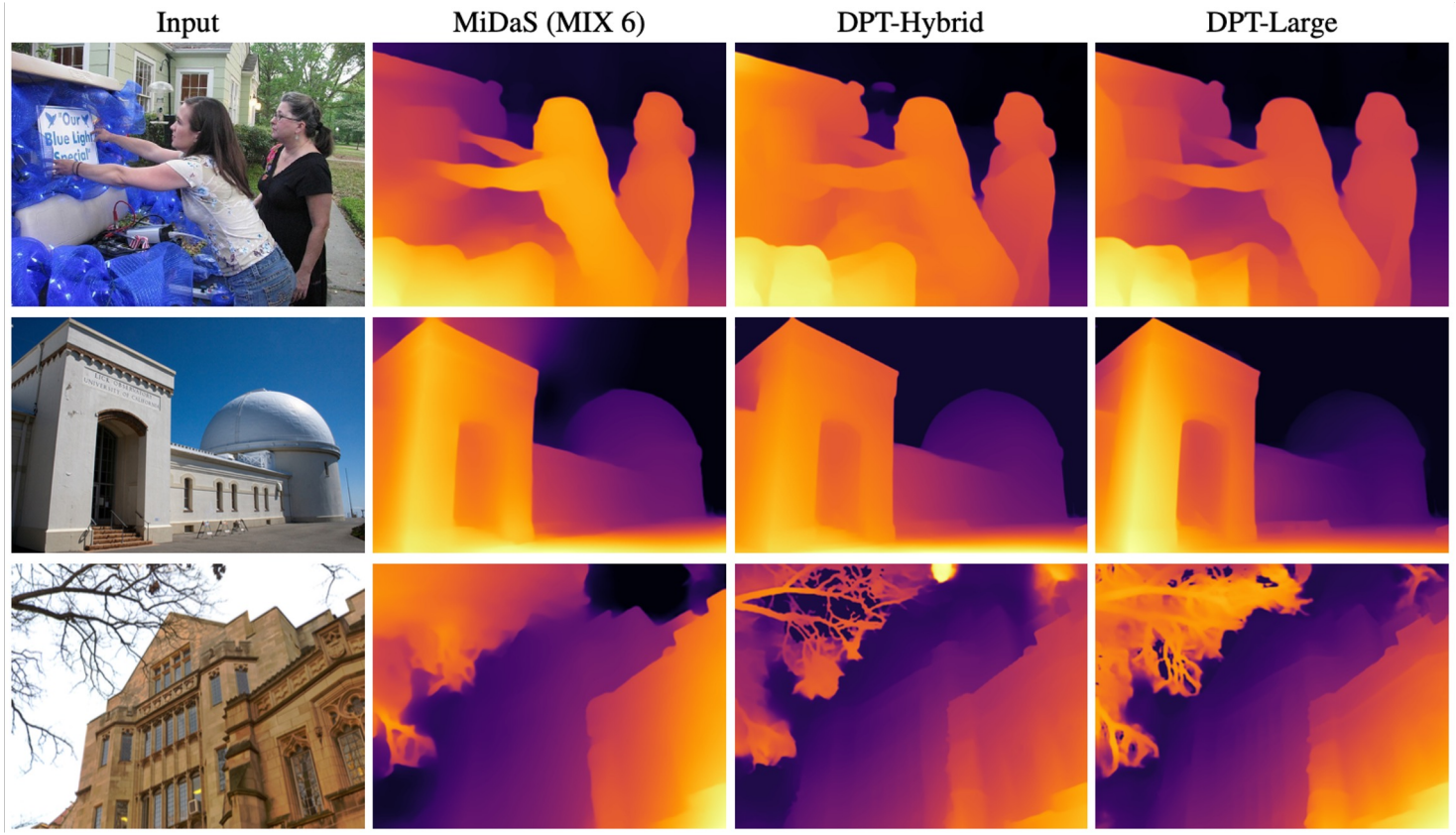
- Their pervious work: MiDaS.
  - It is convolution autoencoder for monocular depth estimation and train with multiple datasets.
- DPT:
  - Crop image patch
  - Embed image patch
  - Transformer encoder
  - Reassemble
  - Convolutional decoder
  - Prediction head



Ranftl et al. "Vision Transformers for Dense Prediction" *in ICCV*, 2021.

Monocular Depth Estimation

- Experimental protocol
  - Same loss and datasets with MiDaS[1]
    - They also conduct larger called MIX-6 and compare with MiDaS[1]
  - Evaluate with zero-shot cross-dataset transfer.



[1] Ranftl et al. "Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer." TPAMI 2020.
Ranftl et al. “Vision Transformers for Dense Prediction” ICCV21

Semantic Segmentation

- Experimental protocol
  - Follow the ResNeSt [1]
  - Train on ADE20K
  - Finetune DPT-Hyvird on Pascal Context dataset

	Backbone		pixAcc [%]	mIoU [%]
OCNet	ResNet101	[50]	–	45.45
ACNet	ResNet101	[14]	81.96	45.90
DeeplabV3	ResNeSt-101	[7, 51]	82.07	46.91
DeeplabV3	ResNeSt-200	[7, 51]	82.45	48.36
DPT-Hybrid	ViT-Hybrid		**83.11**	**49.02**
DPT-Large	ViT-Large		82.70	47.63

Table 4. Semantic segmentation results on the ADE20K validation set.

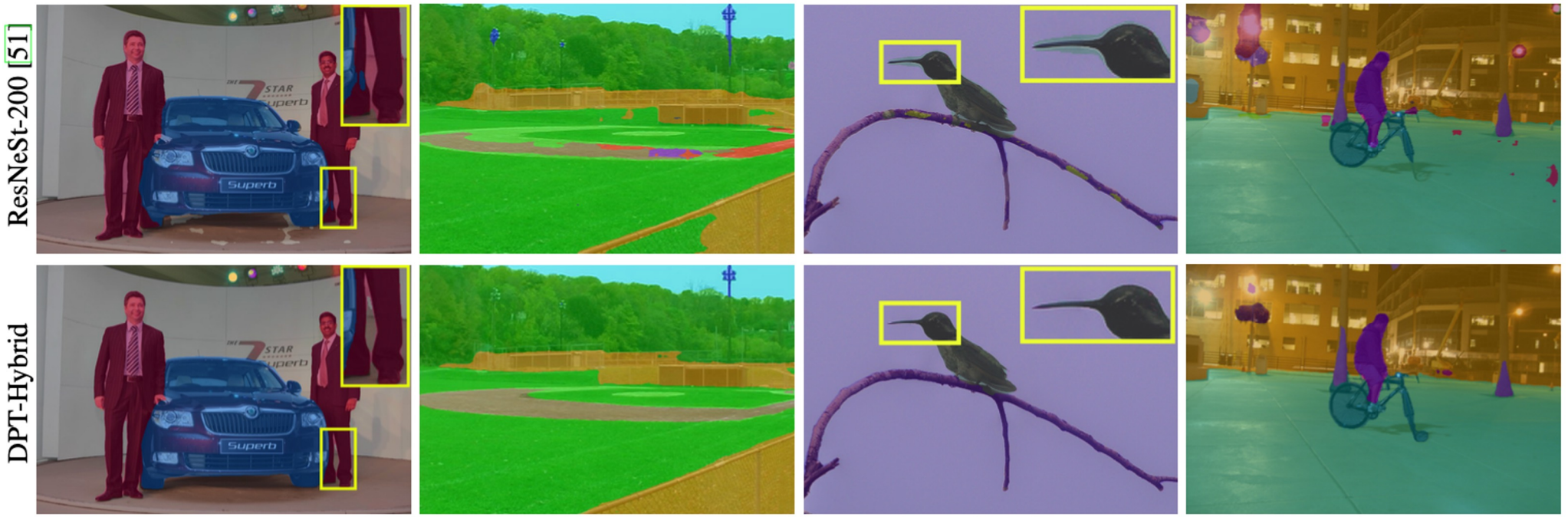
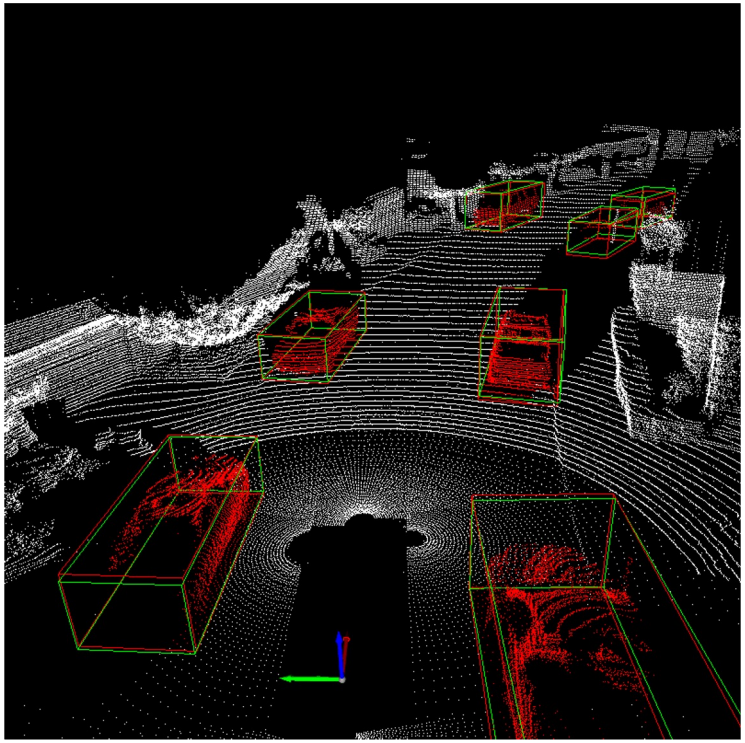


Figure 3. Sample results for semantic segmentation on ADE20K (first and second column) and Pascal Context (third and fourth column). Predictions are frequently better aligned to object edges and less cluttered.

[1]Zhang, Hang, et al. "Resnest: Split-attention networks." arXiv preprint arXiv:2004.08955 (2020).
Ranftl et al. “Vision Transformers for Dense Prediction” ICCV21

Point Transformers

Hengshuang et al. “Point Transformers” *in ICCV*, 2021.



Point cloud in 3D deep learning

Source: https://morioh.com/p/3b4c863218dc

Background – 3D data / point cloud

- 3D data arises in many application areas such as autonomous driving, augmented reality, and robotics.

- Unlike images, which are arranged on regular pixel grids, 3D point clouds are sets embedded in continuous space.

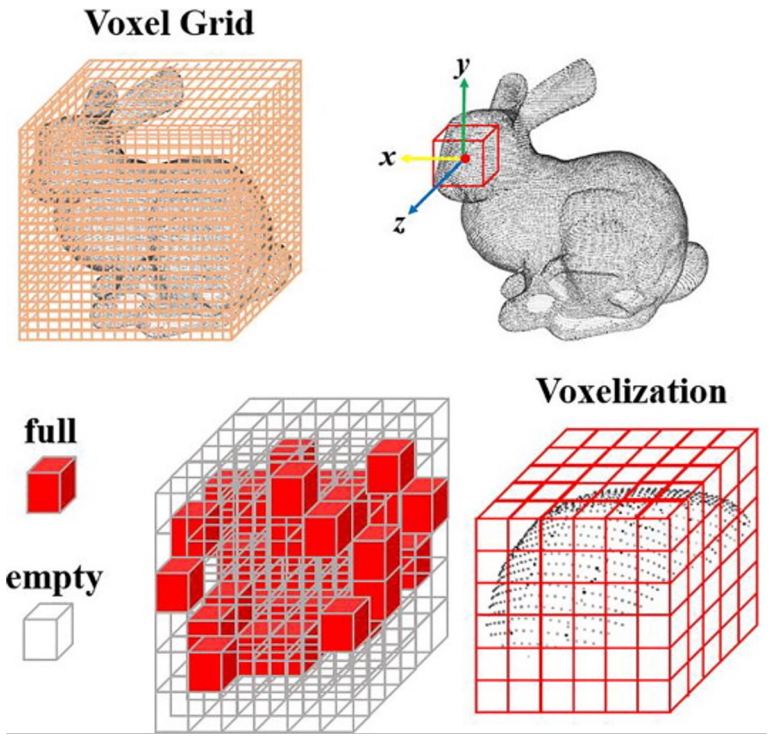
This makes 3D point clouds structurally different from images and precludes immediate application of deep network designs that have become standard in computer vision, such as networks based on the discrete convolution operator.

Hengshuang et al. "Point Transformers" *in ICCV*, 2021.

Challenge

- Some researchers voxelize the 3D space (see figure) to enable the application of 3D discrete convolutions.

This induces massive computational and memory costs and underutilizes the sparsity of point sets in 3D.



Source : https://www.sciencedirect.com/science/article/pii/S2215098618311649

Hengshuang et al. “Point Transformers” *in ICCV*, 2021.

Motivation - Success of Transformers

- The transformer family of models is particularly appropriate for point cloud processing because the self-attention operator, which is at the core of transformer networks, is in essence a set operator:

  it is invariant to permutation and cardinality of the input elements.

  The application of self-attention to 3D point clouds is therefore quite natural, since point clouds are essentially sets embedded in 3D space.

Hengshuang et al. “Point Transformers” *in ICCV*, 2021.

Output head

For semantic segmentation: the final decoder stage produces a feature vector for each point in the input point set. We apply an MLP to map this feature to the final logits.

For classification: we perform global average pooling over the pointwise features to get a global feature vector for the whole point set.

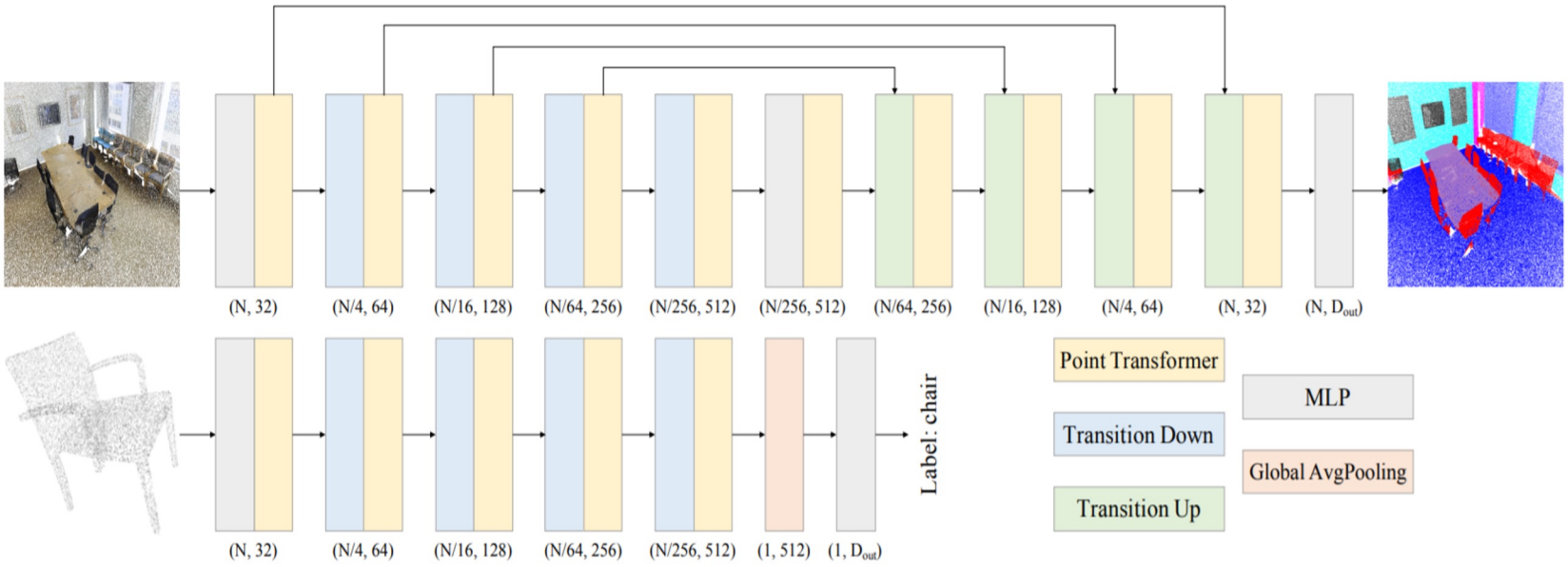


Figure 3. Point transformer networks for semantic segmentation (top) and classification (bottom).

Hengshuang et al. "Point Transformers" *in ICCV*, 2021.

Semantic Segmentation

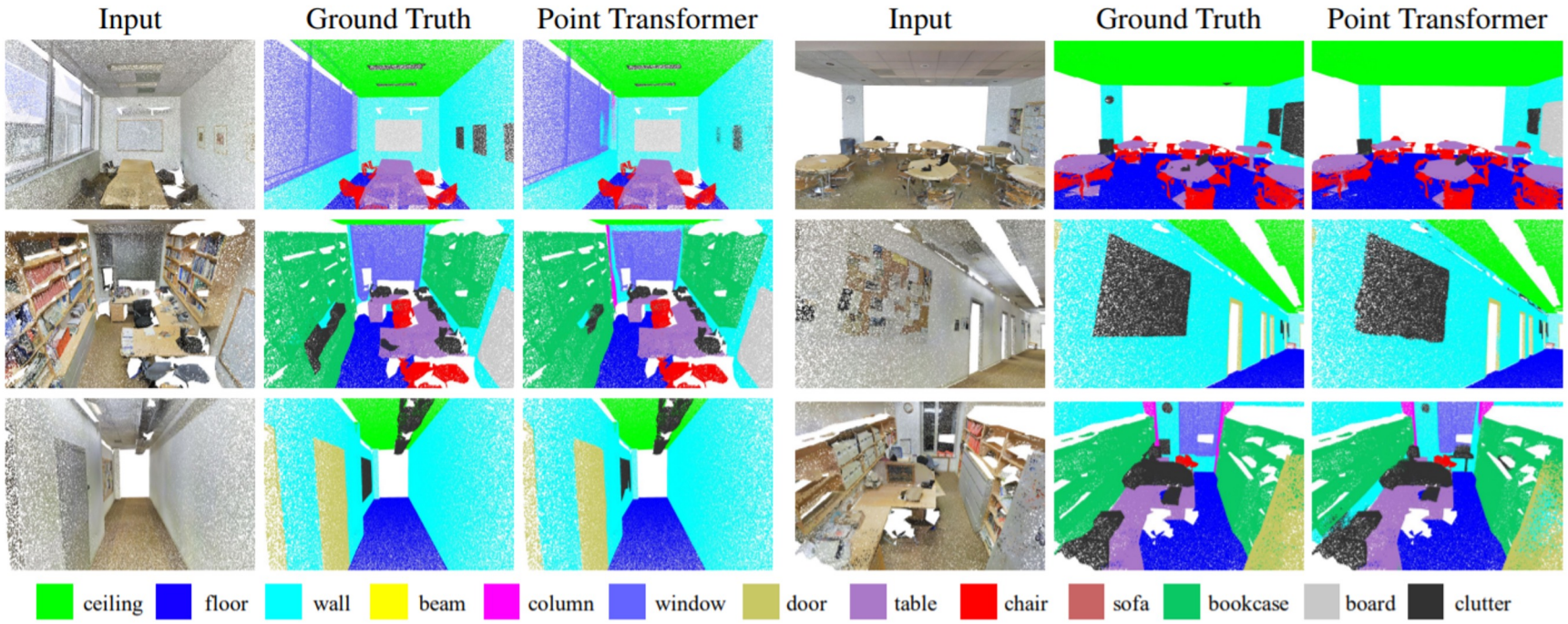


Figure 5. Visualization of semantic segmentation results on the S3DIS dataset.

Hengshuang et al. “Point Transformers*” in ICCV*, 2021.