COMP5212: Machine Learning

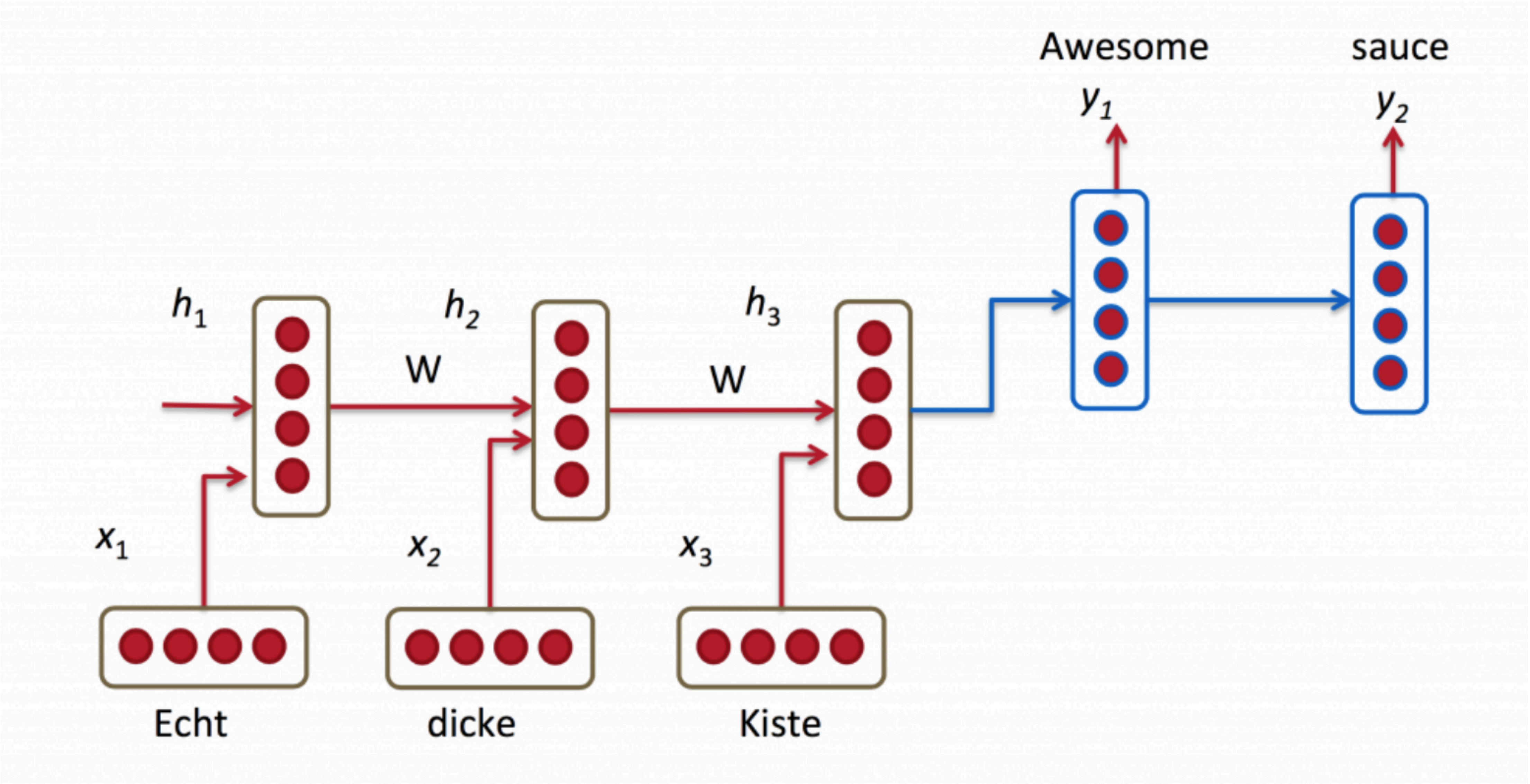
Lecture 15

Minhao Cheng

Recurrent Neural Network

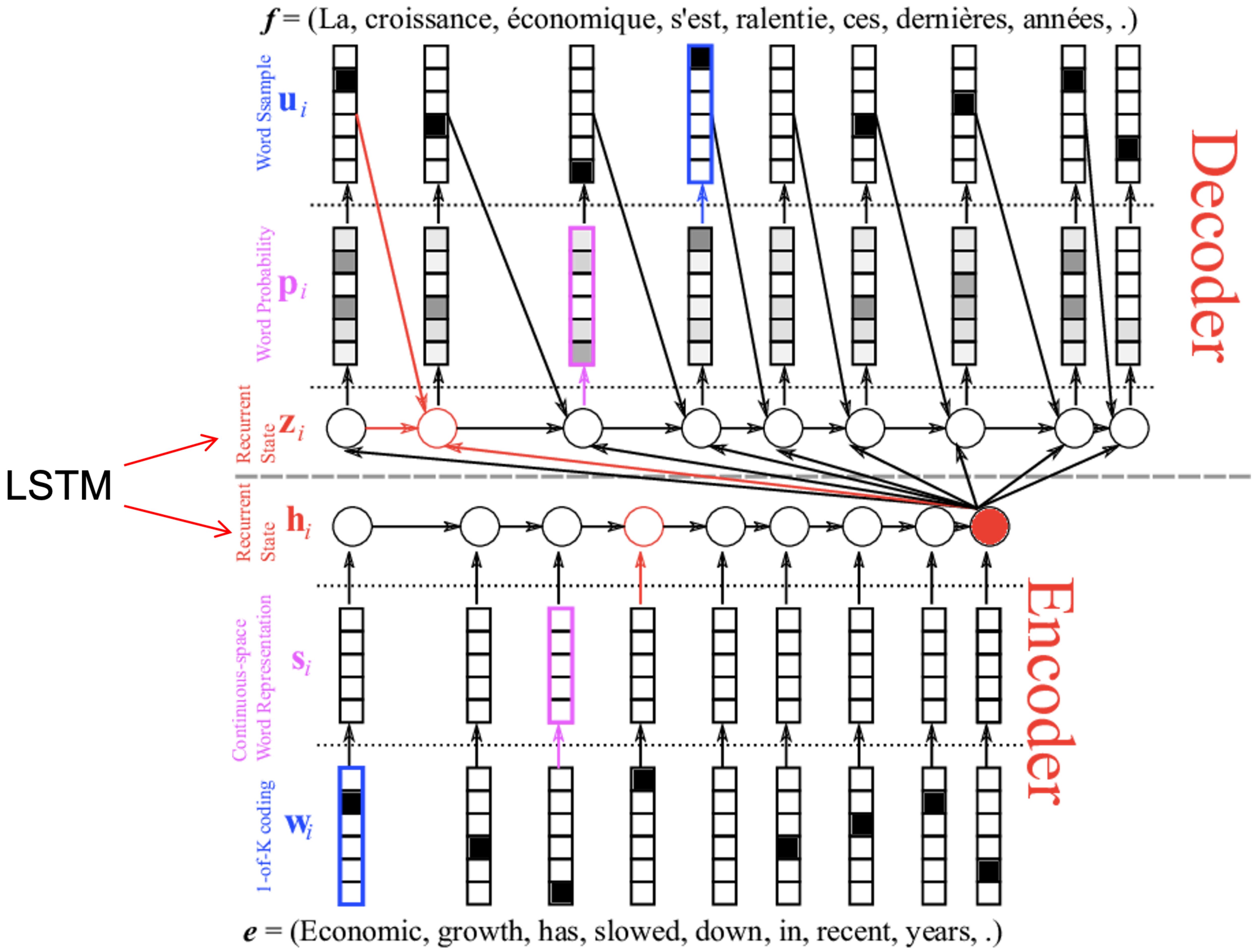
Neural Machine Translation (NMT)

- Out the translated sentence from an input sentence
- Training data: a set of input-output pairs (supervised setting)
- Encoder-decoder approach:
  - Encoder: Use (RNN/LSTM) to encode the input sentence input a latent vector
  - Decoder: Use (RNN/LSTM) to generate a sentence based on the latent vector



Recurrent Neural Network

Neural Machine Translation



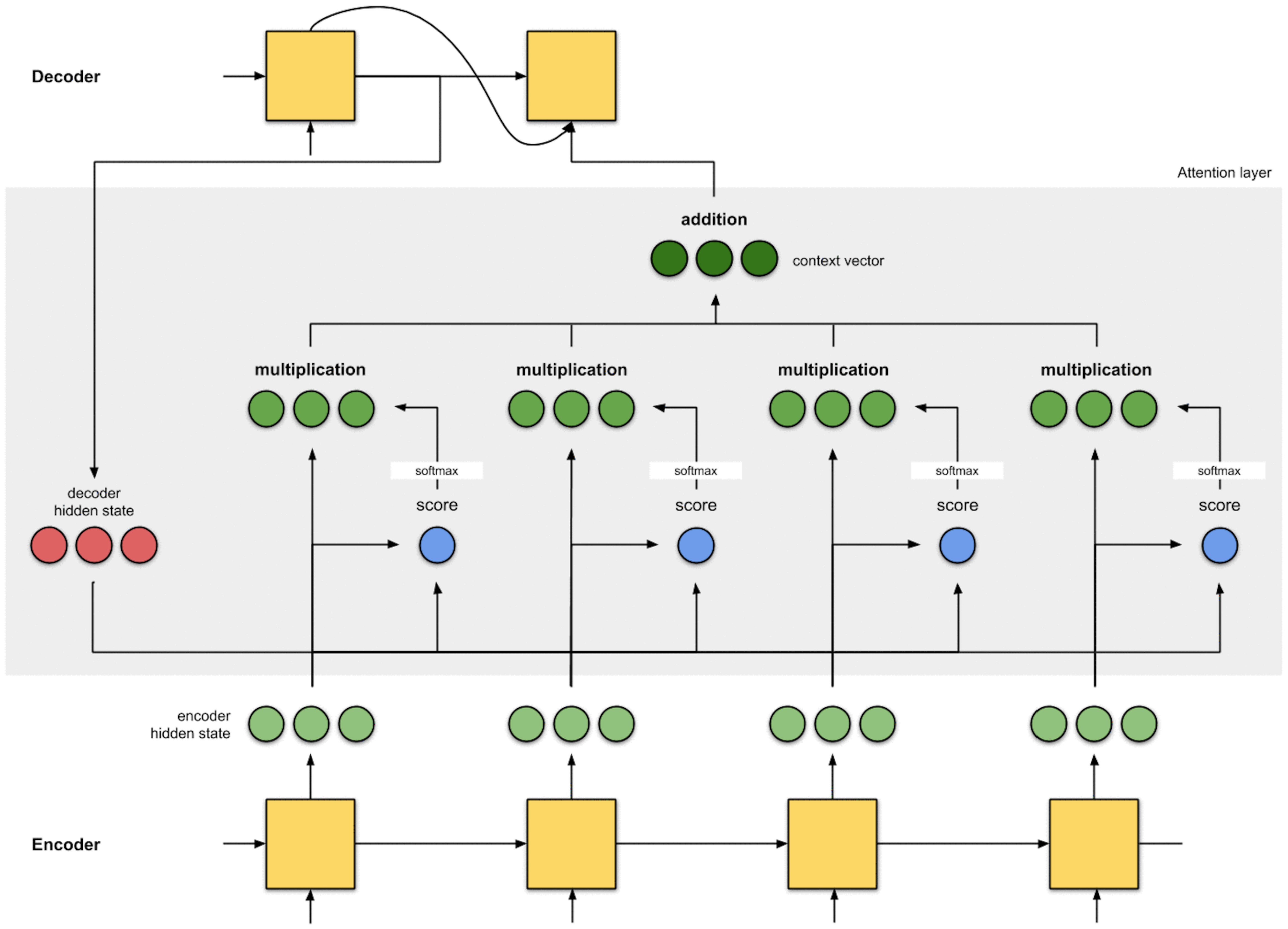
Recurrent Neural Network

Attention in NMT

- Usually, each output word is only related to a subset of input words (e.g., for machine translation)
- Let u be the current decoder latent state, $v_1, \ldots, v_n$ be the latent sate for each input word
- Compute the weight of each state by
  - $p = \text{Softmax}(u^T v_1, \ldots, u^T v_n)$
- Compute the context vector by $Vp = p_1 v_1 + \ldots + p_n v_n$

Recurrent Neural Network

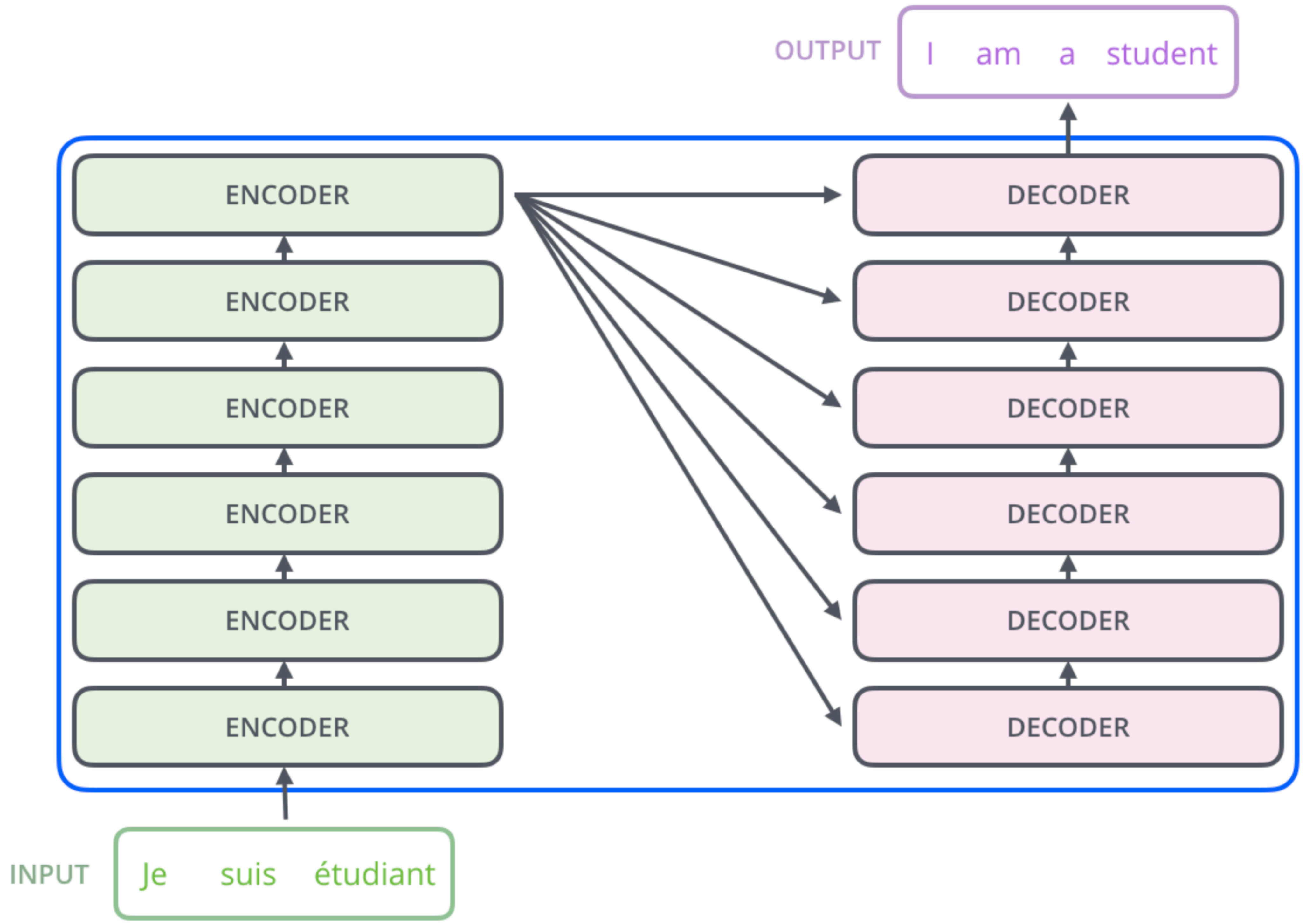
Attention in NMT



Transformer

Transformer

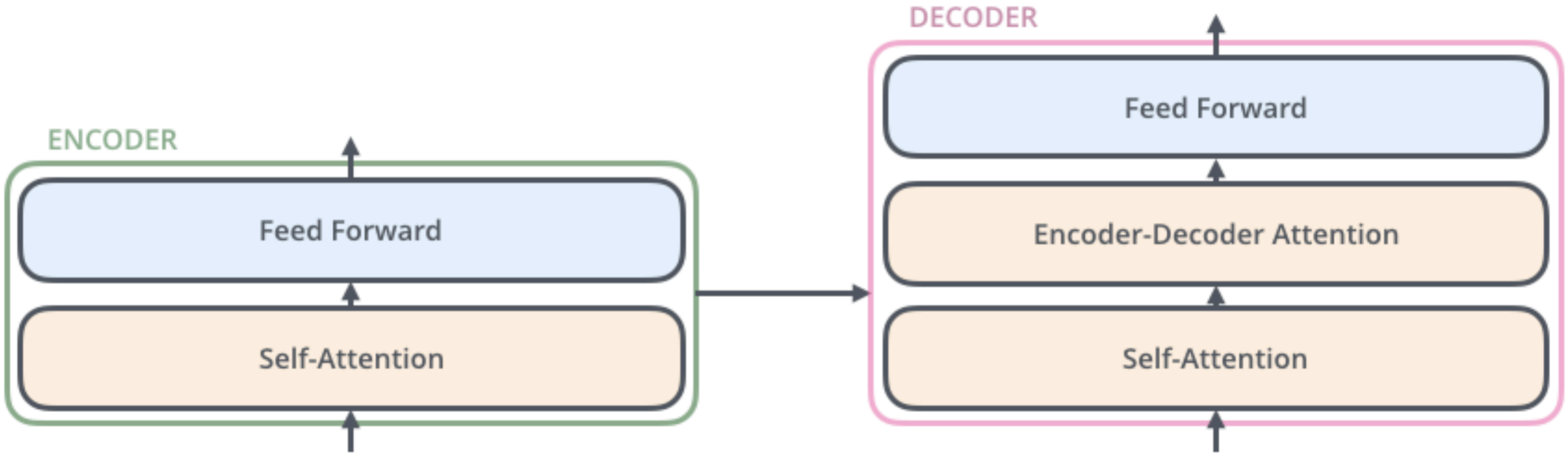
- An architecture that replies entirely on attention without using CNN/RNN
- Proposed in ``Attention Is All You Need'' (Vaswani et al., 2017)
- Initially used for neural machine translation



Transformer

Encoder and Decoder

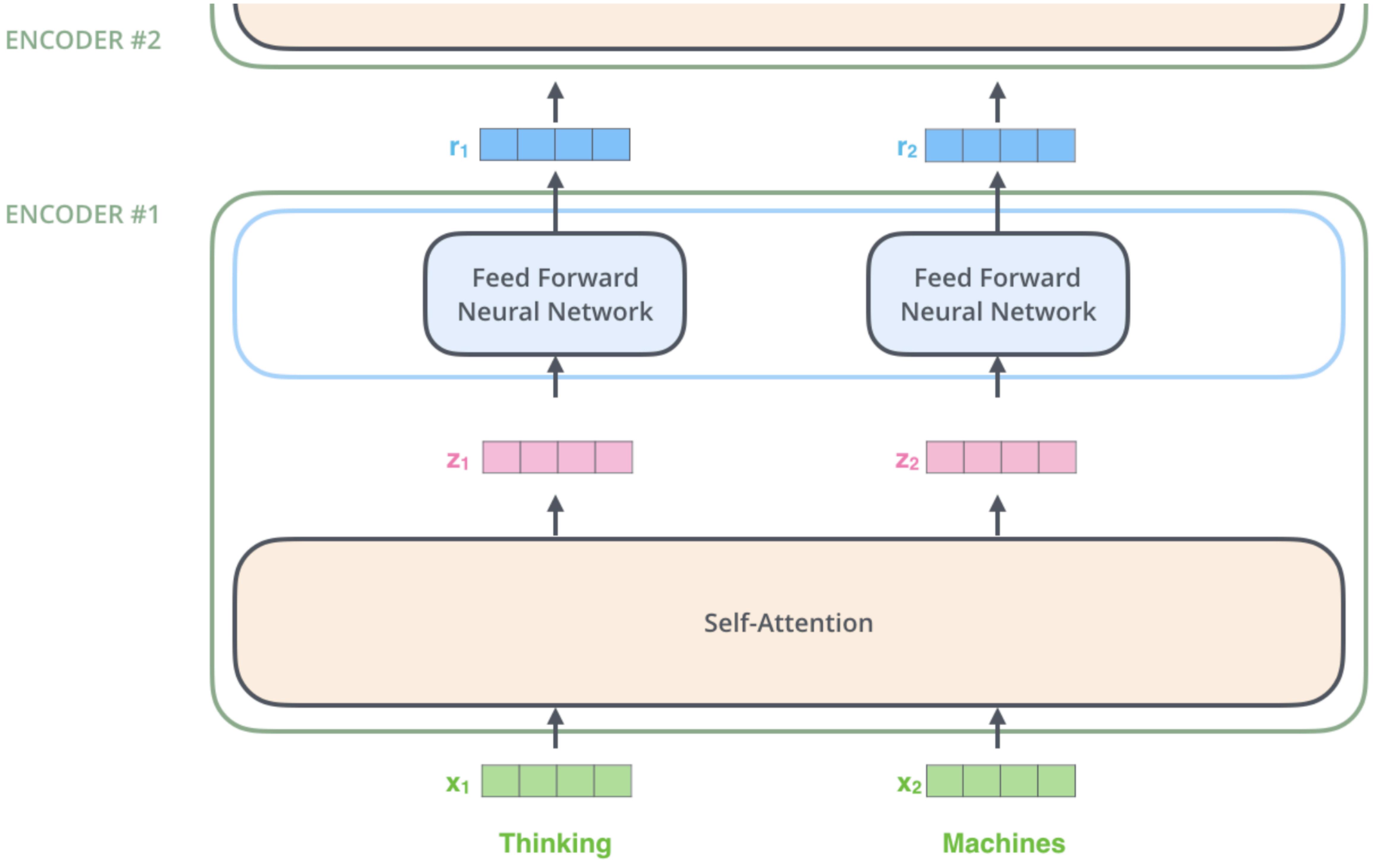
- Self attention layer: the main architecture used in Transformer
- Decoder: will have another attention layer to help it focuses on relevant parts of input sentences.



Transformer

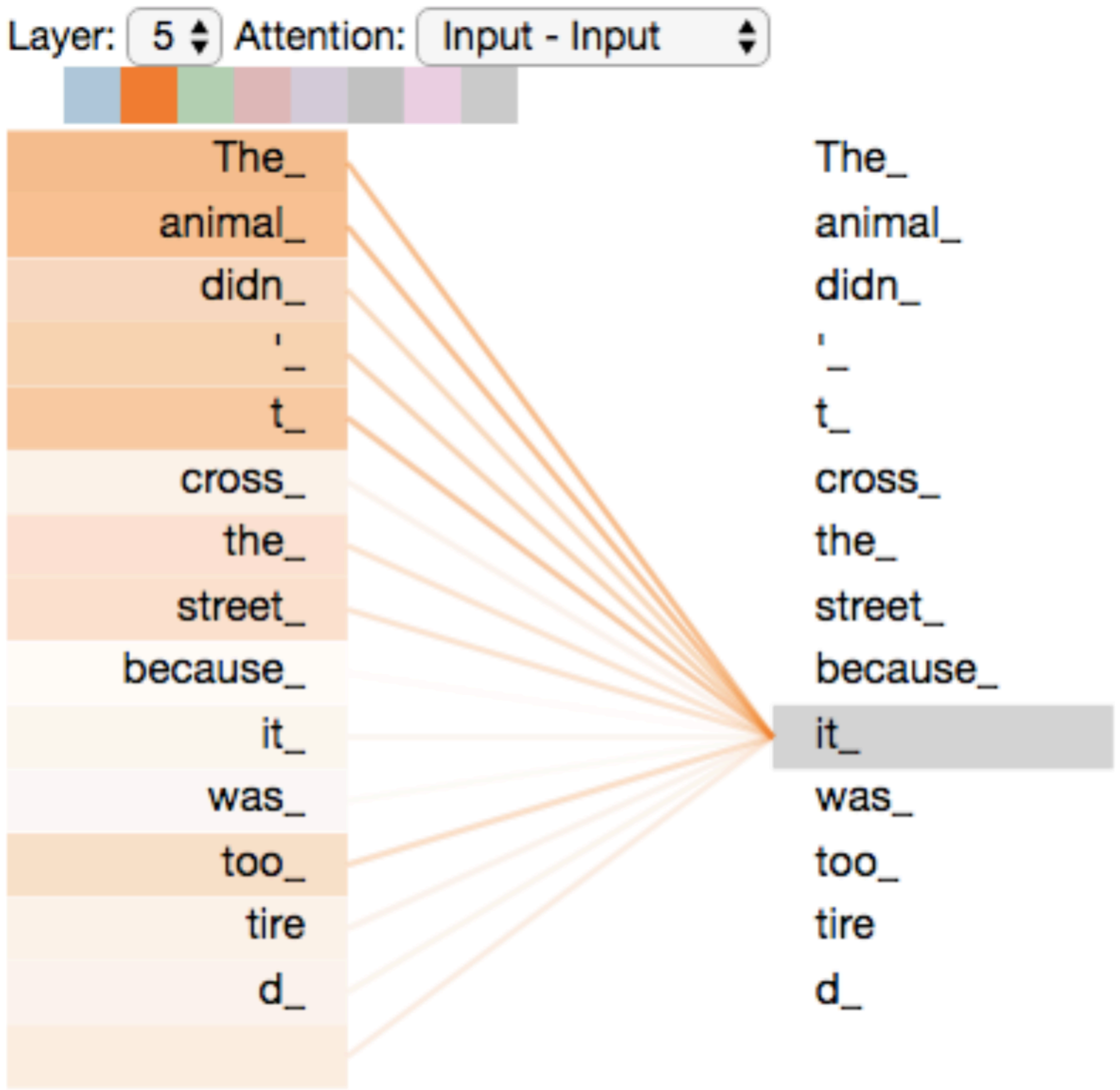
Encoder

- Each word has a corresponding ``latent vector'' (initially the word embedding for each word)
- Each layer of encoder:
  - Receive a list of vectors as input
  - Passing these vectors to a self-attention layer
  - Then passing them into a feed-foward layer
  - Output a list of vectors



Transformer
Self-attention layer

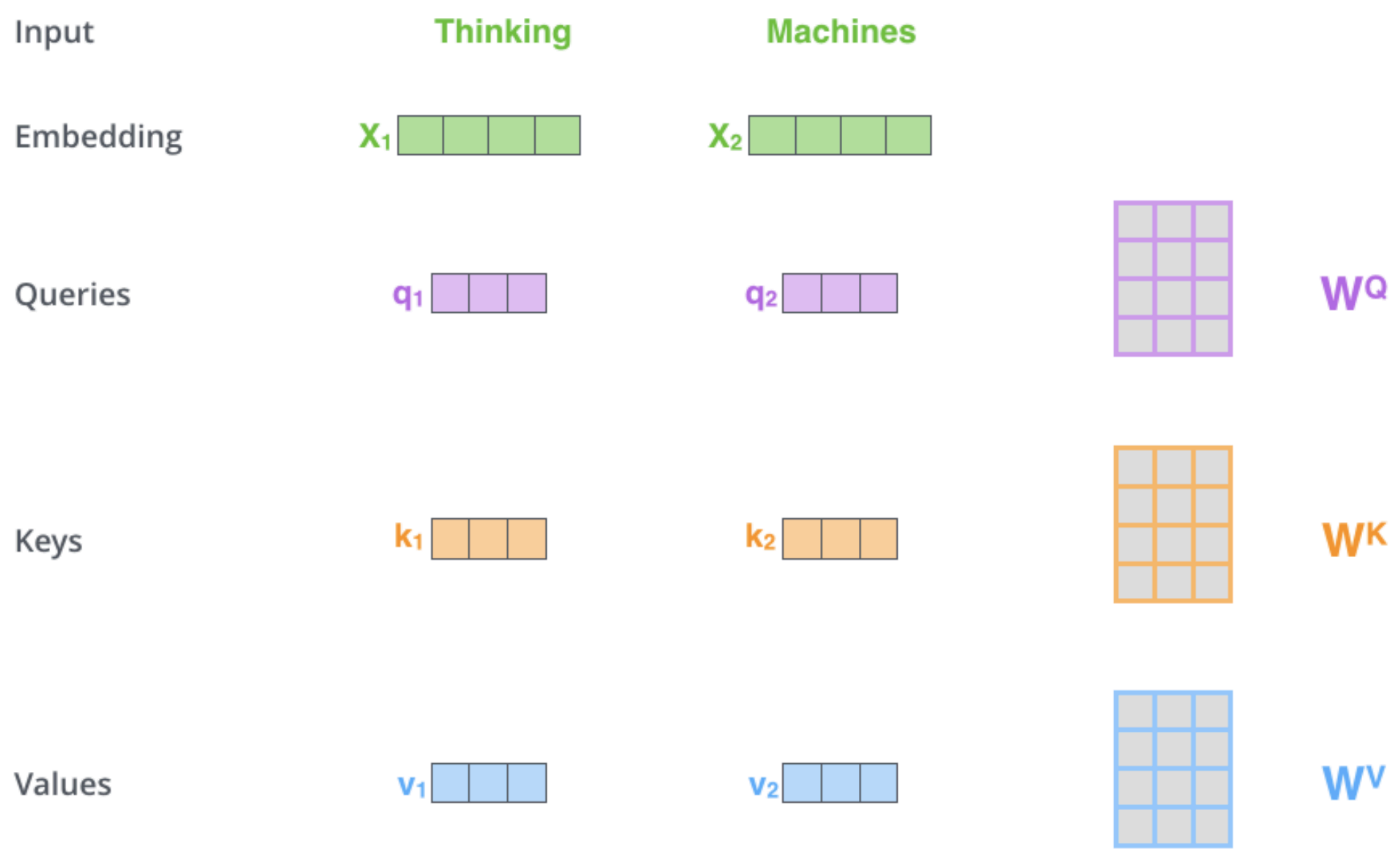
- Main idea: The actual meaning of each word may be related to other words in the sentence
- The actual meaning (latent vector) of each word is a weighted (attention) combination of other words (latent vectors) in the sentences



Transformer

Self-attention layer

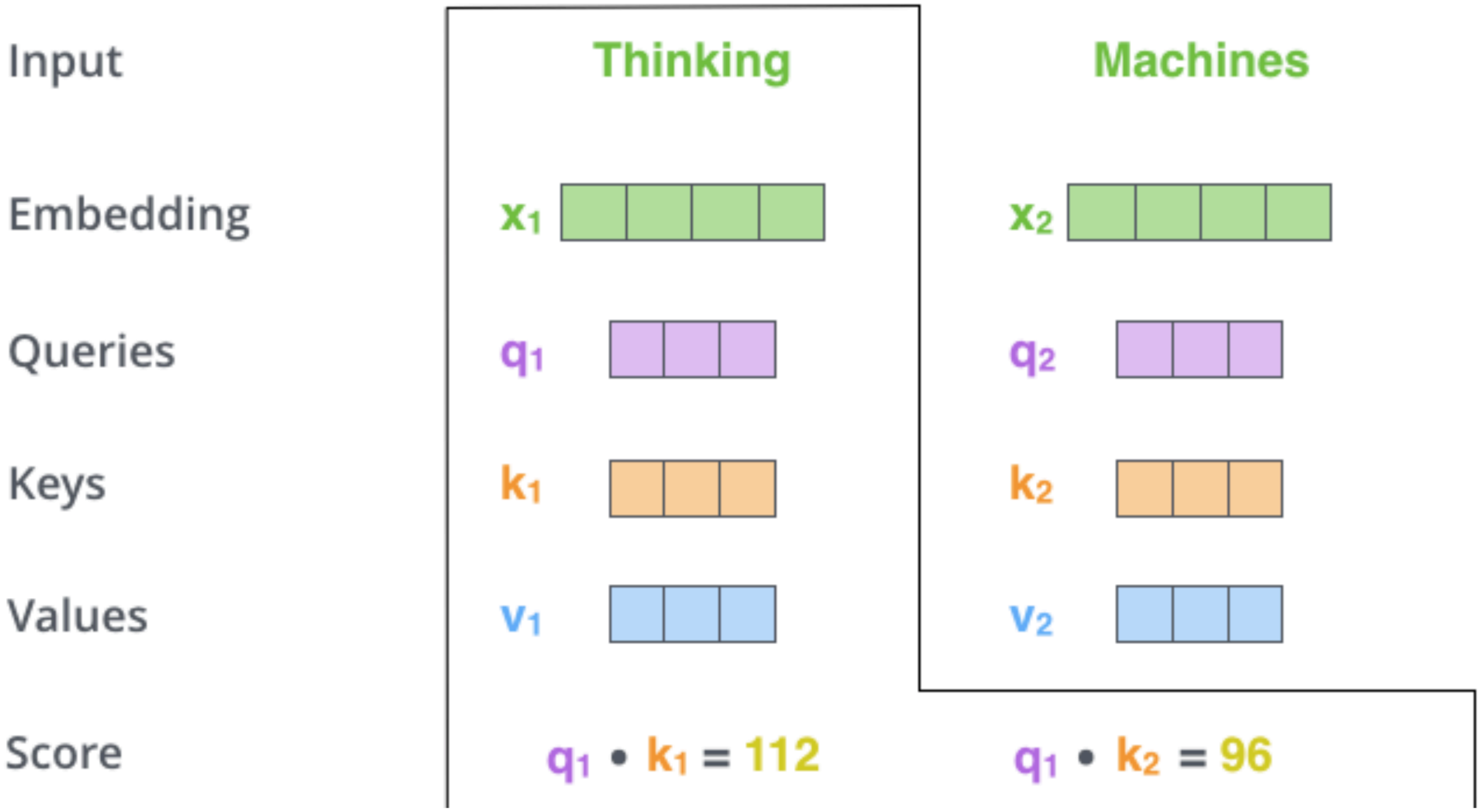
- Input latent vectors: $x_1, \ldots, x_n$
- Self-attention parameters: W^Q, W^K, W^V (weights for query, key, value)
- For each word i, compute
  - Query vector: $q_i = x_i W^Q$
  - Key vector: $k_i = x_i W^K$
  - Value vector: $v_i = x_i W^V$



Transformer

Self-attention layer

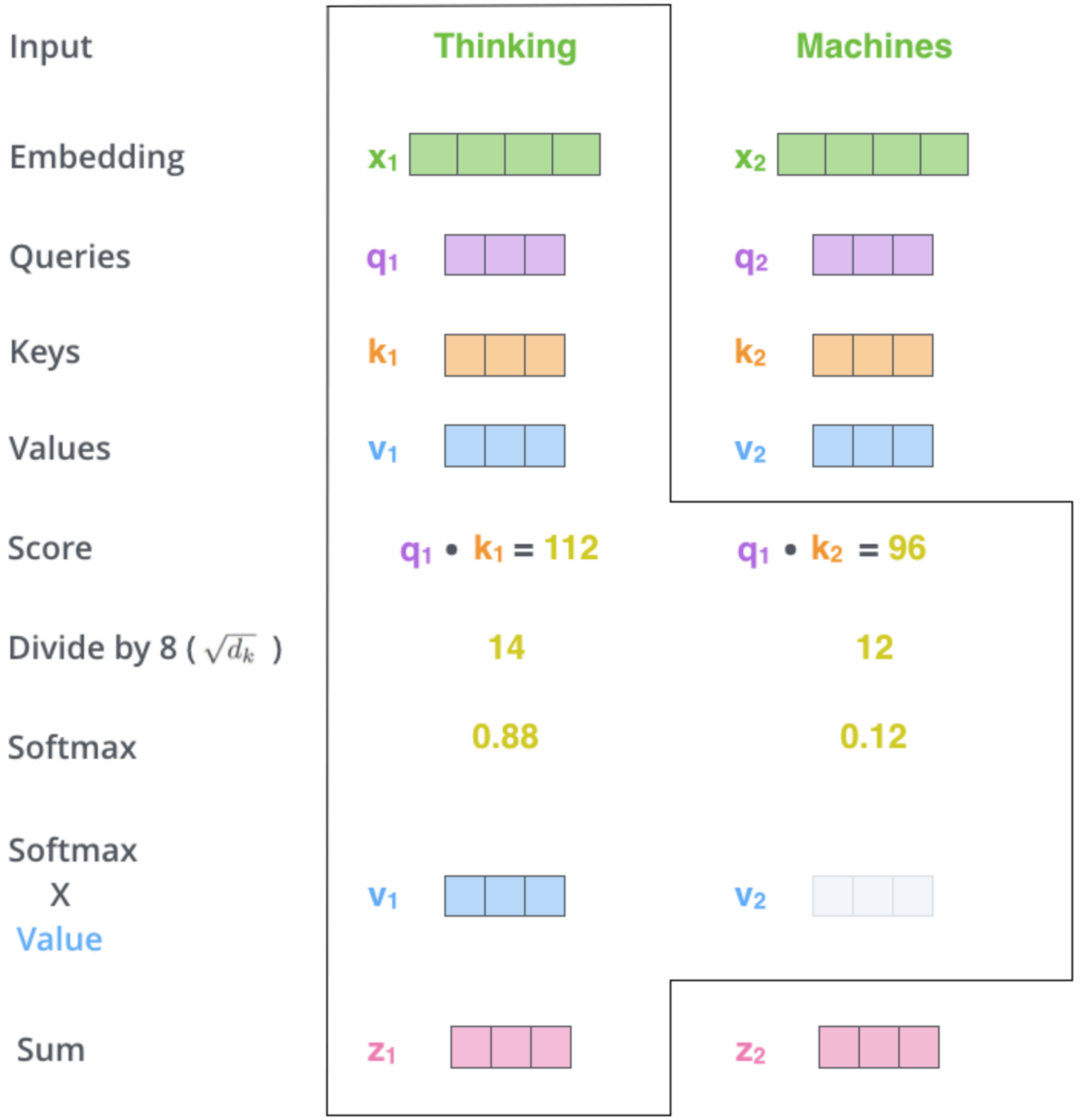
- For each word i, compute the scores to determine how much focus to place on other input words
  - The attention score for word j to word i: $q_i^T k_j$



Transformer

Self-attention layer

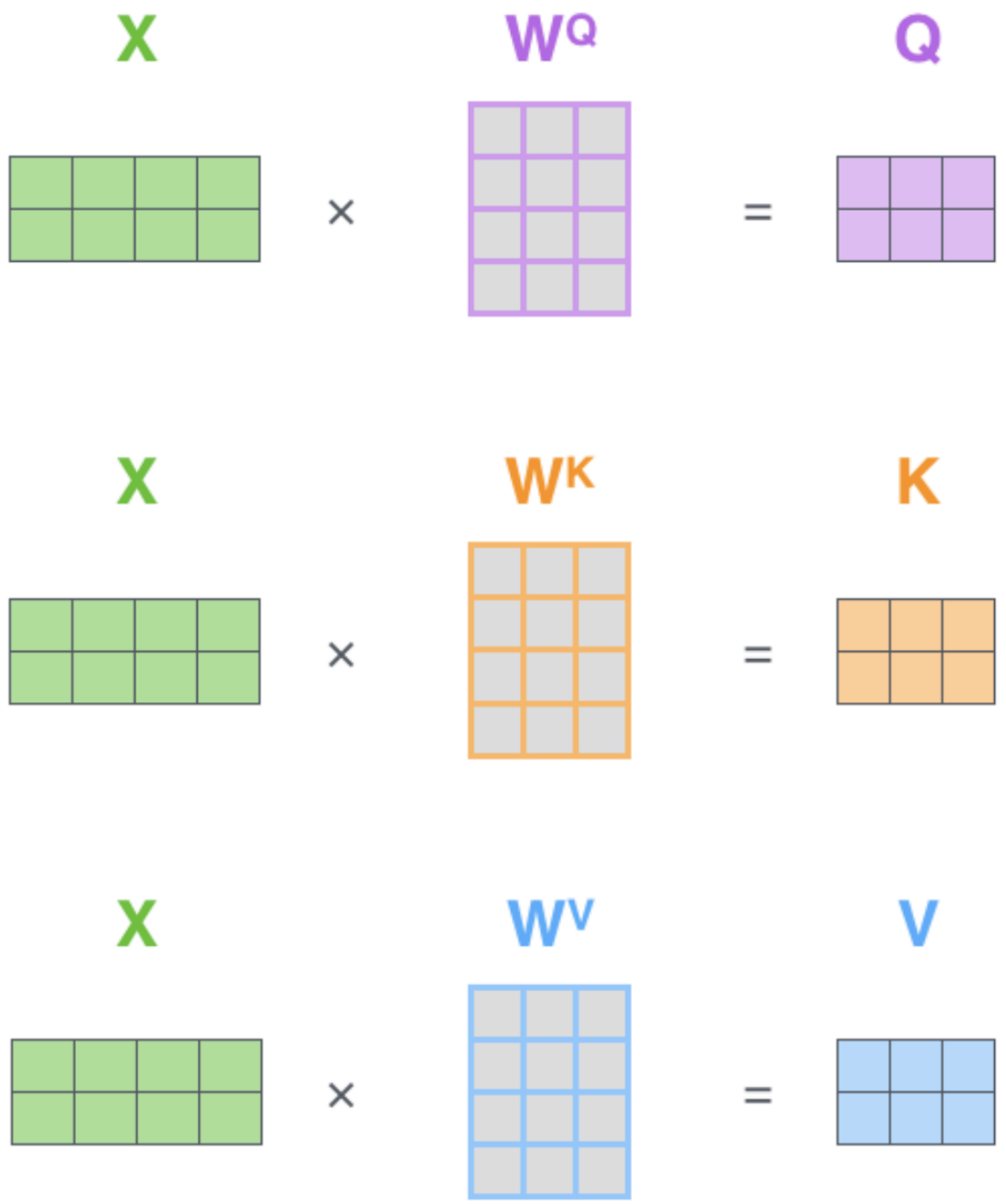
- For each word i, the output vector
- $\sum_j s_{ij} v_j, \quad s_i = \text{softmax}(q_i^T k_1, \ldots, q_i^T k_n)$

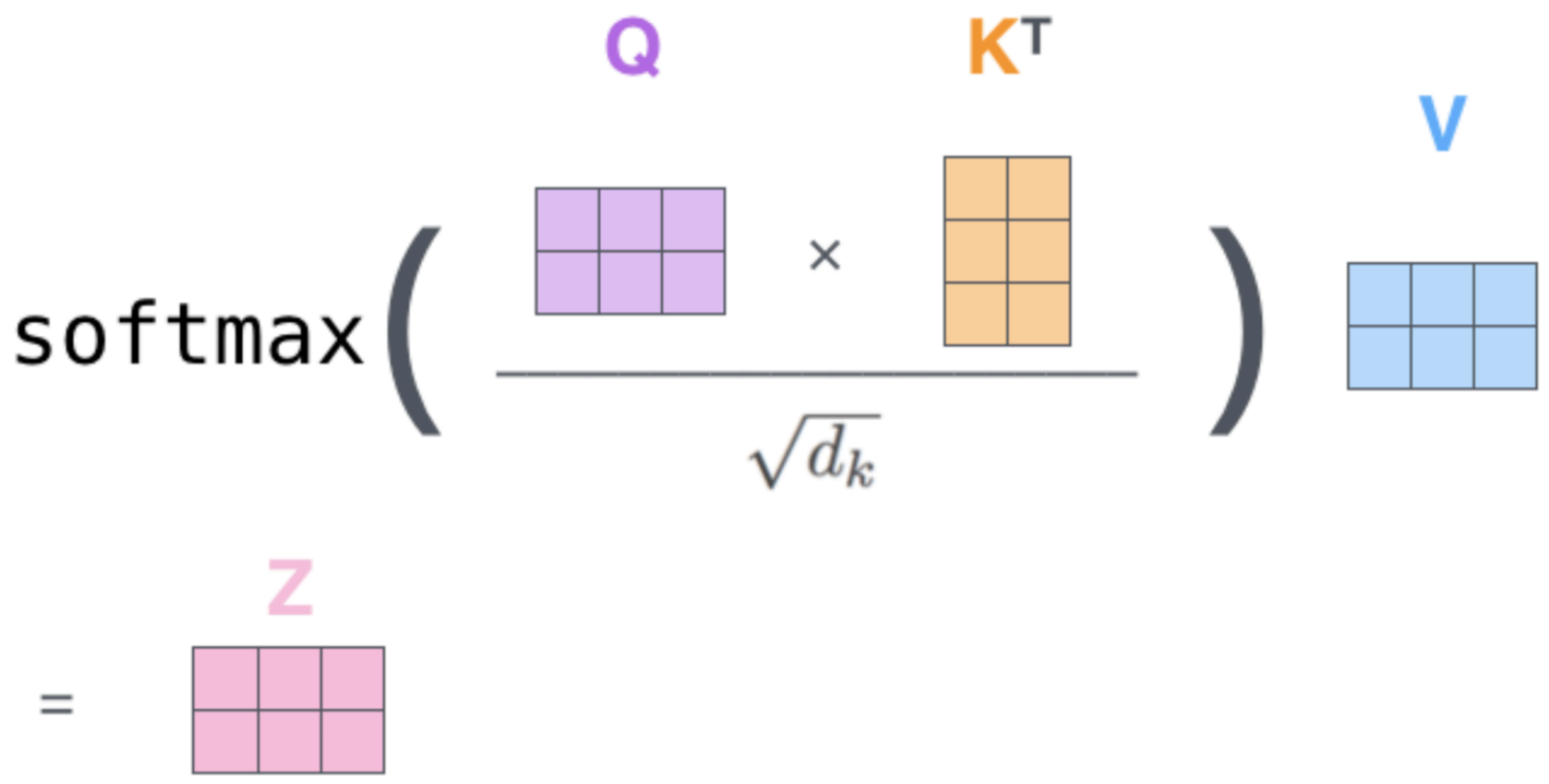


Transformer

Matrix form

- $Q = XW^Q,\ K = XW^K,\ V = XW^V,\quad Z = \text{softmax}(QK^T)V$

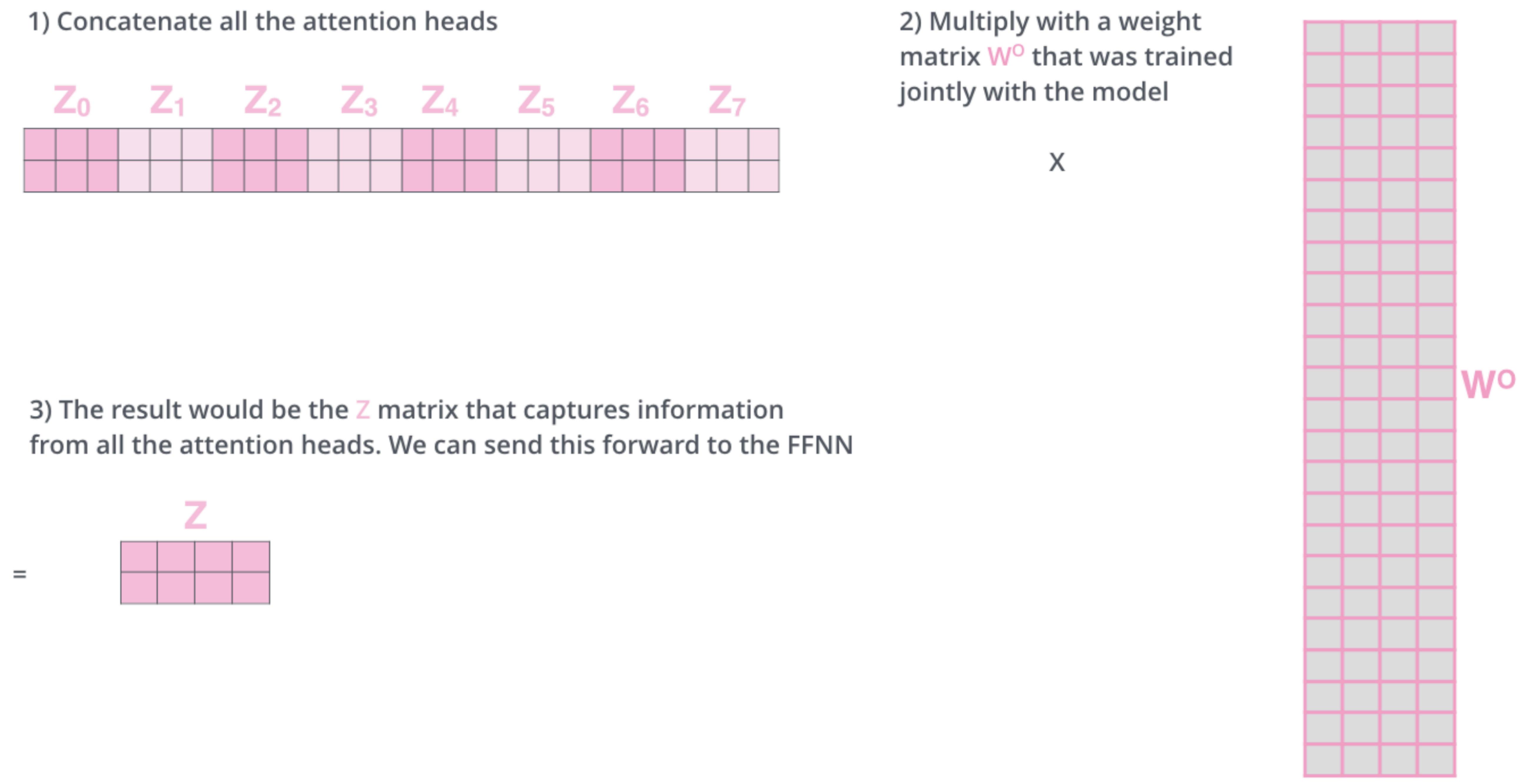




Transformer

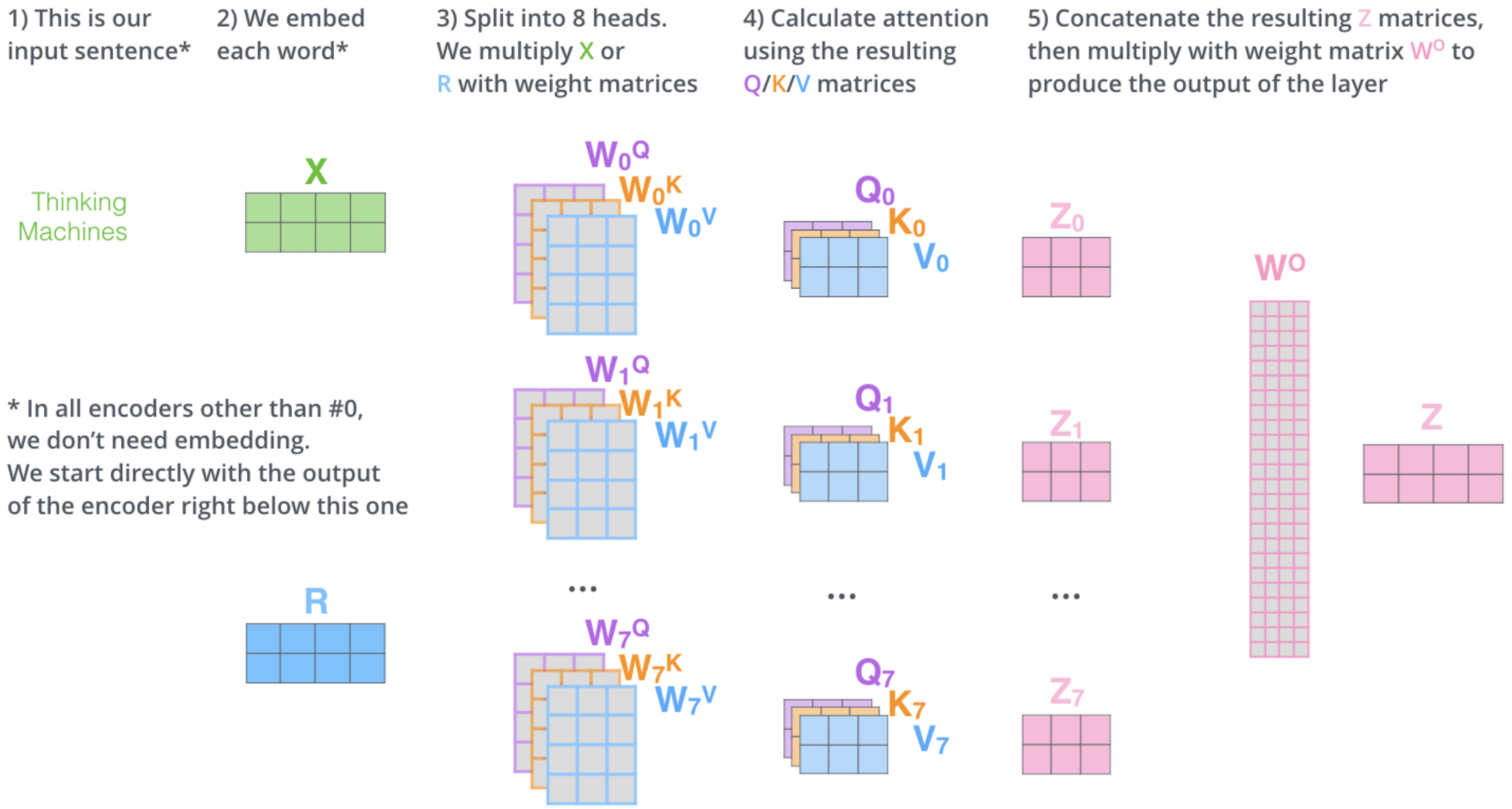
Multiply with weight matrix to reshape

- Gather all the outputs $Z_1, \ldots, Z_k$
- Multiply with a weight matrix to reshape
- Then pass to the next fully connected layer



Transformer

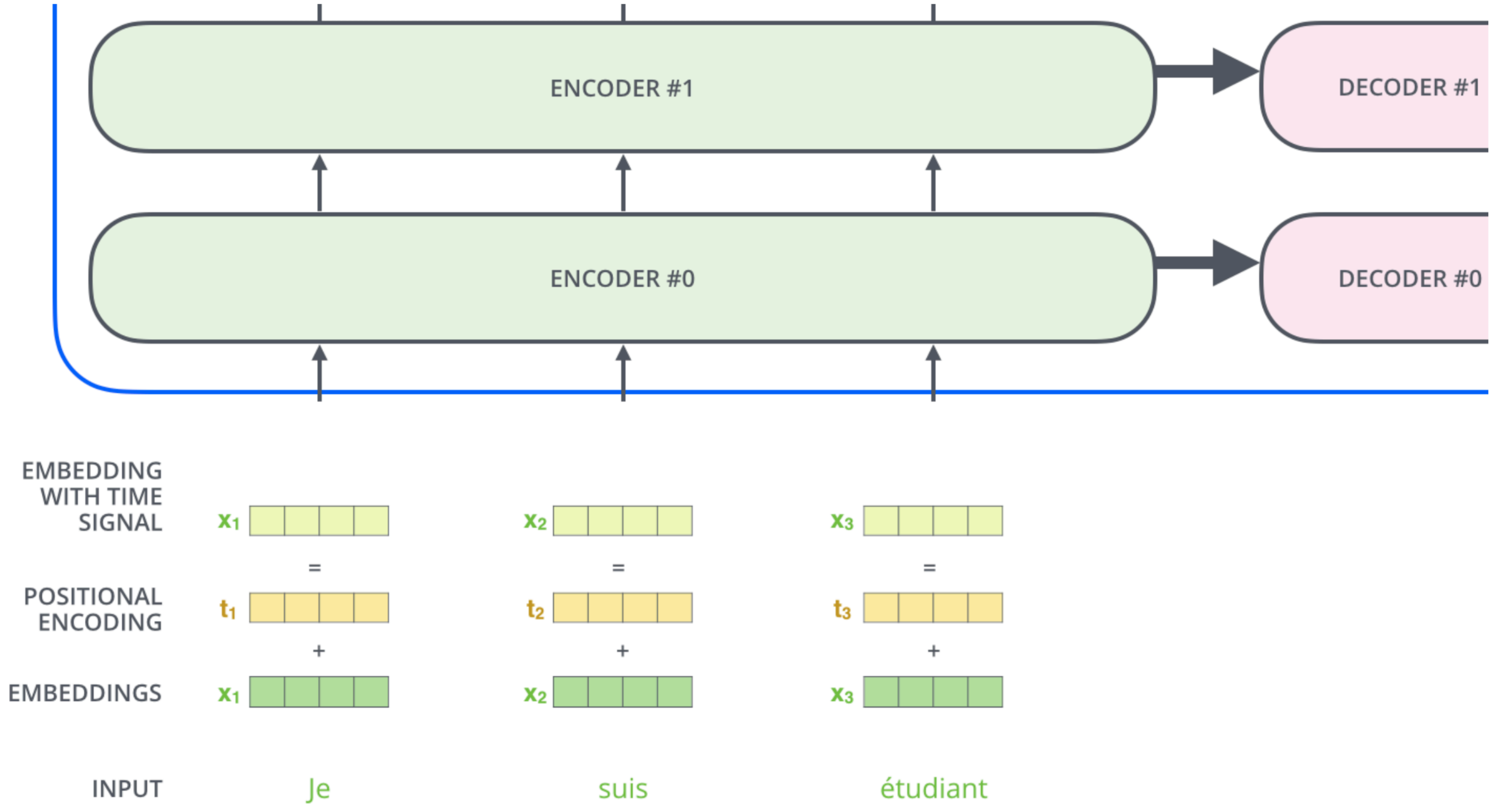
Overall architecture



Transformer

Sinusoidal Position Encoding

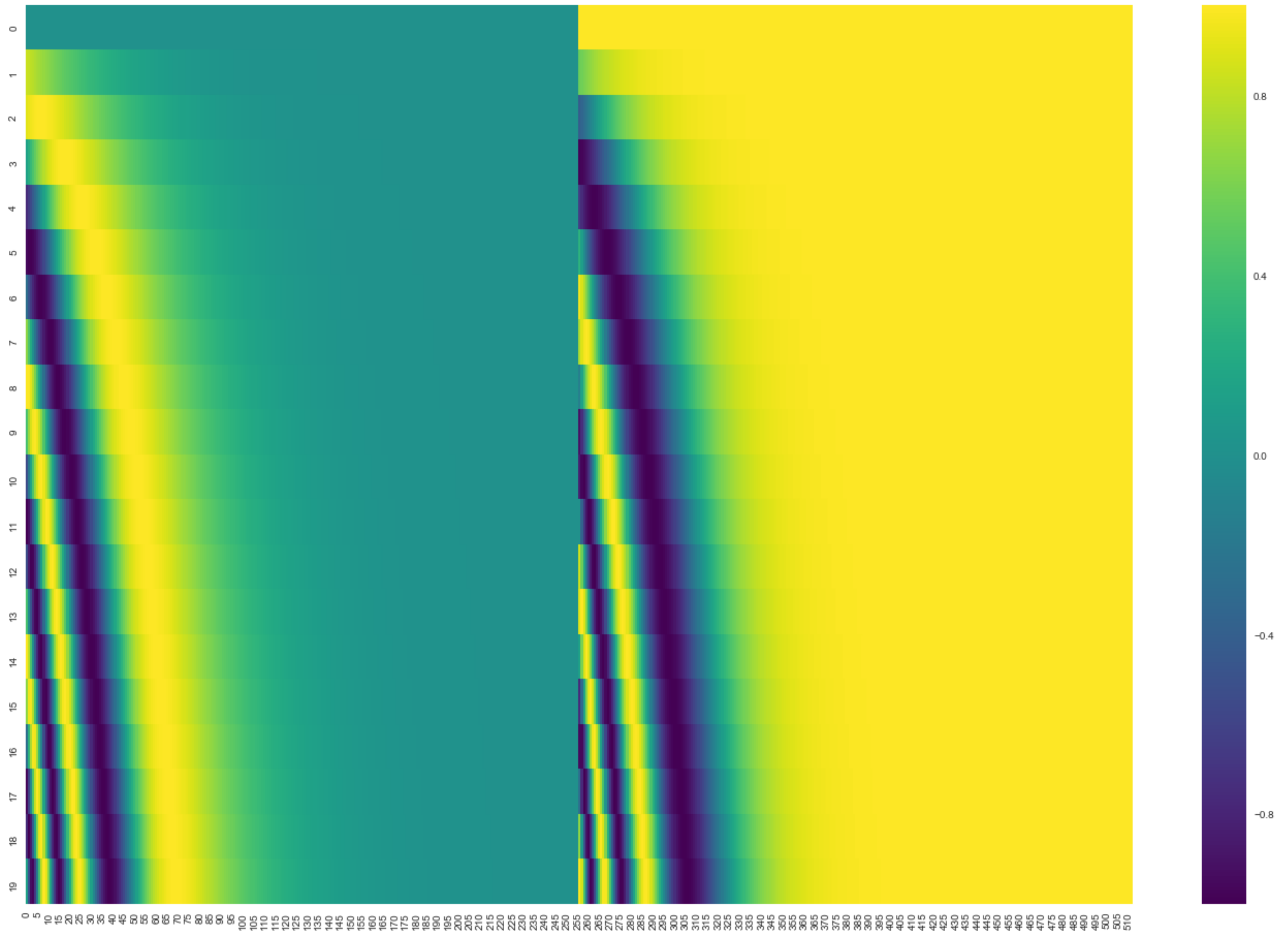
- The above architecture ignores the sequential information
- Add a positional encoding vector to each x_i (according to i)



Transformer

Positional Embedding

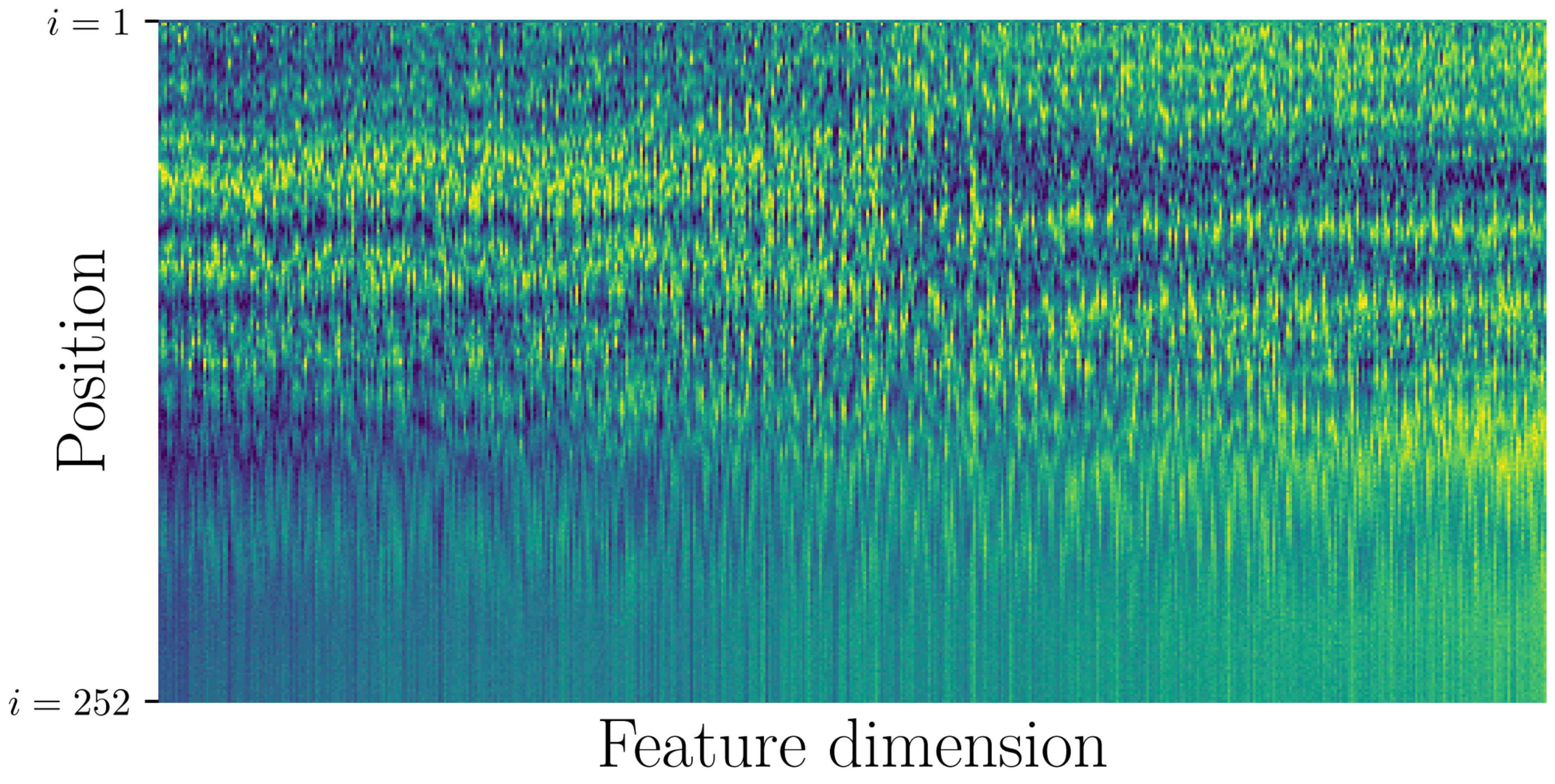
- Sin/cosine functions with different wavelengths (used in the original Transformer)
- The jth dimension of ith token $p_i[j] = \begin{cases} \sin(i \cdot c^{\frac{j}{d}}) \text{ if j is even} \\ \cos(i \cdot c^{\frac{j-1}{d}}) \text{ if j is odd} \end{cases}$
- smooth, parameter-free, inductive



Transformer

Types of positional encoding

- Position embedding: learn a latent vector for each position
  - non-smooth, data-driven (learnable), non-inductive
- Relative position embedding:
  - For each i, j, use the relative position embedding a_{j-i}
  - non-smooth, data-driven (learnable), (partial)-inductive



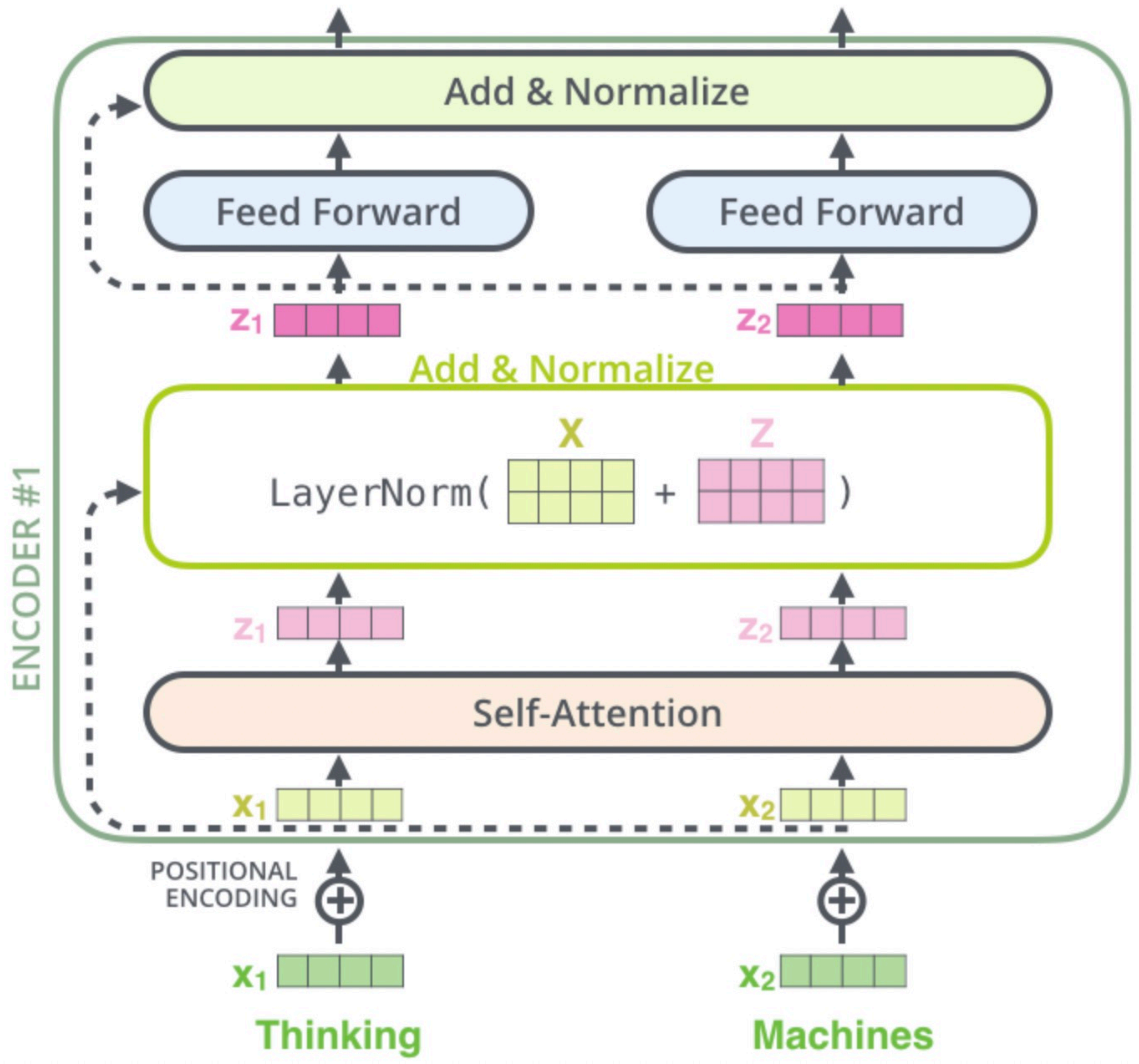
Transformer

Positional Encoding

- Neural ODE embedding :
  - Model positional embedding as a dynamic linear system
    - ``Learning to Encode Position for Transformer with Continuous Dynamical Model. Liu et al., 2020''
- Learnable Fourier Feature (Li et al., 2021):
  - $p_x = \phi(r_x, \theta)W_p$, where $r_x = \frac{1}{D}[\cos xW_r, \sin xW_r]$
  - W_r, W_p: learnable parameters (irrelevant to sequence length)
  - ``Learnable Fourier Features for Multi-Dimensional Spatial Positional Encoding. Li et al., 2021''
  - smooth, data-driven (learnable), inductive

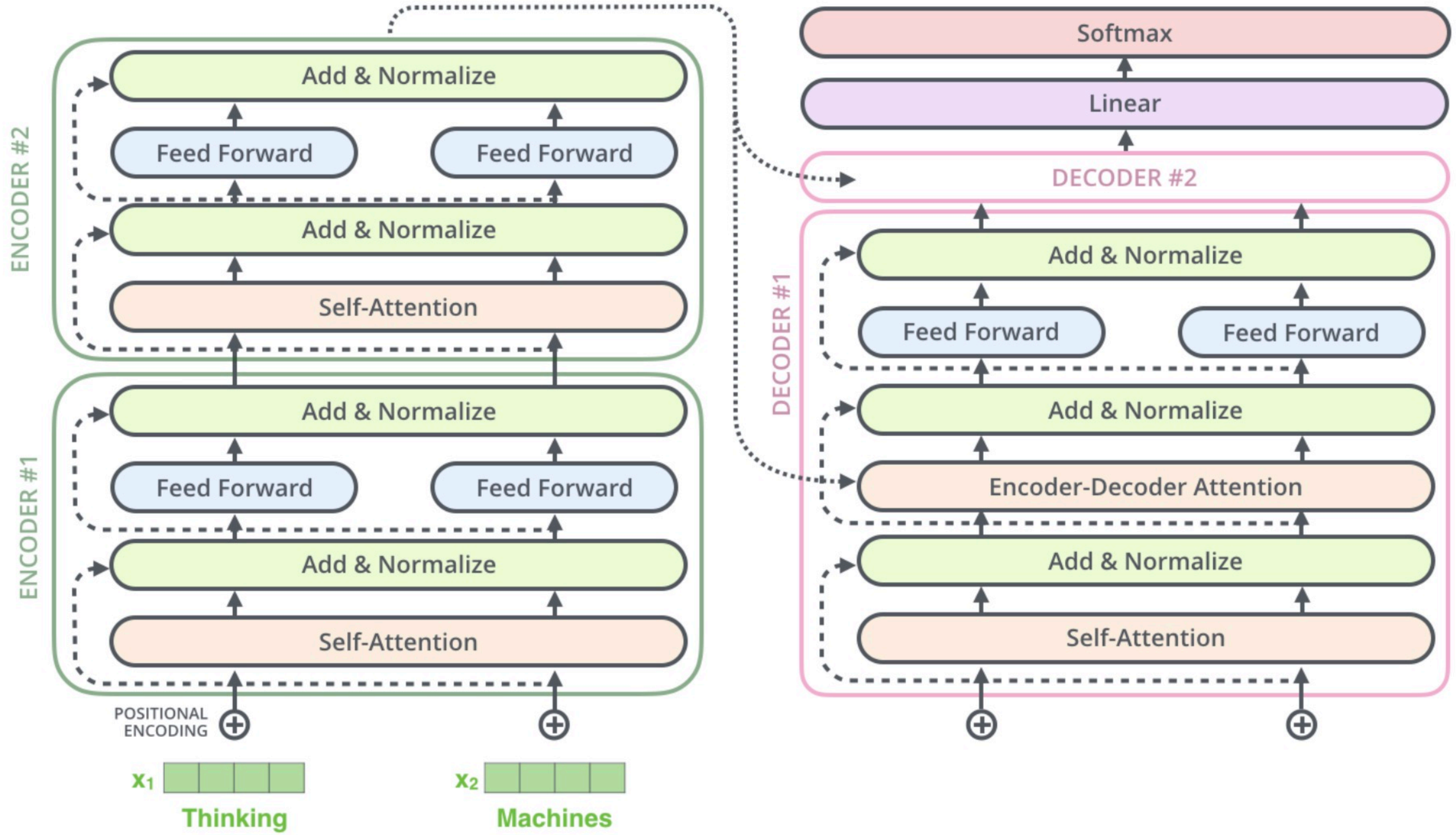
Transformer

Residual



Transformer

Whole framework



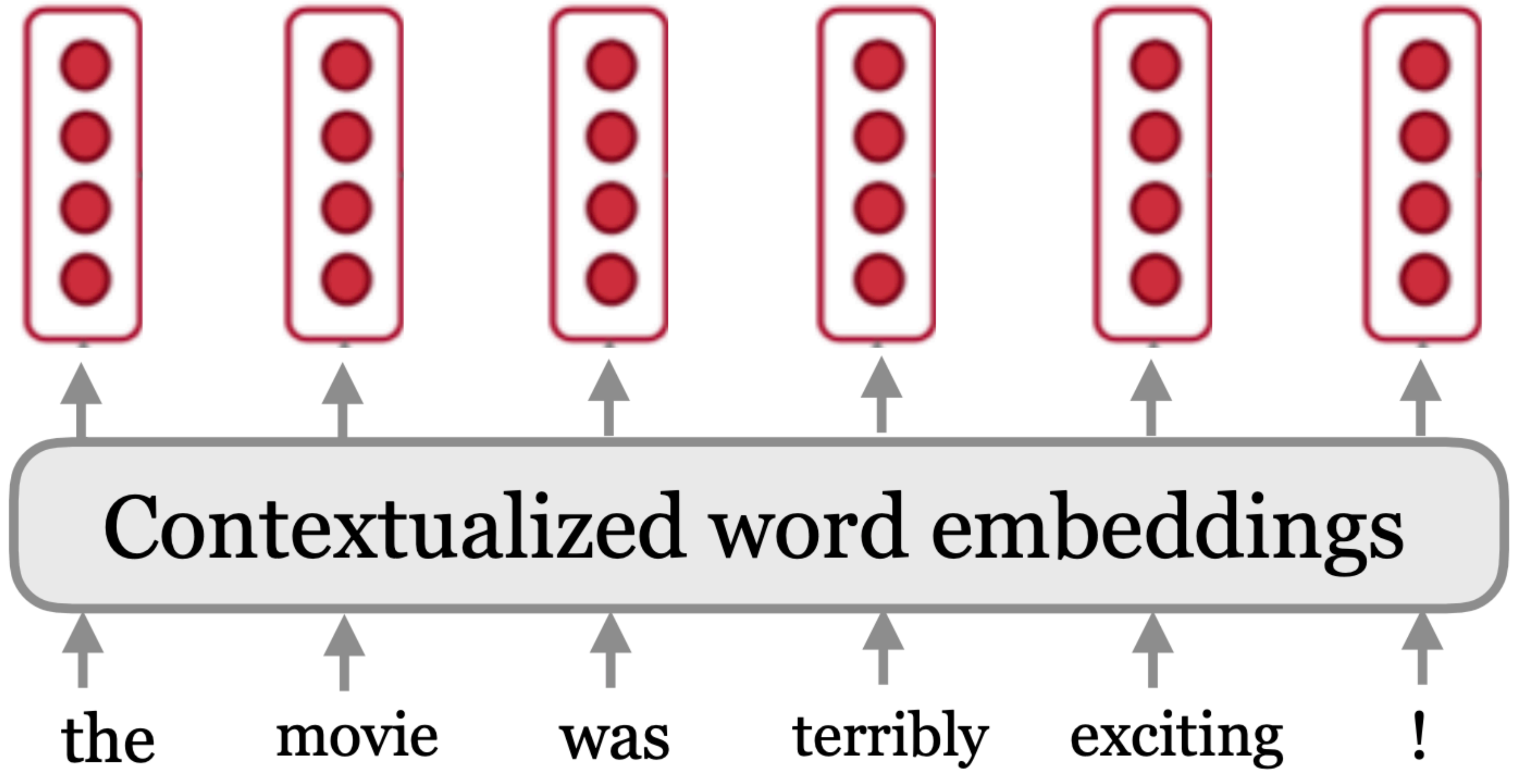
Contextual embedding

Contextual world representation

- The semantic meaning of a word should depend on its context

```
[0.9, -0.2, 1.6, …]                    [-1.9, -0.4, 0.1, …]
       ↑                                        ↑
open a bank account                  on the river bank
```

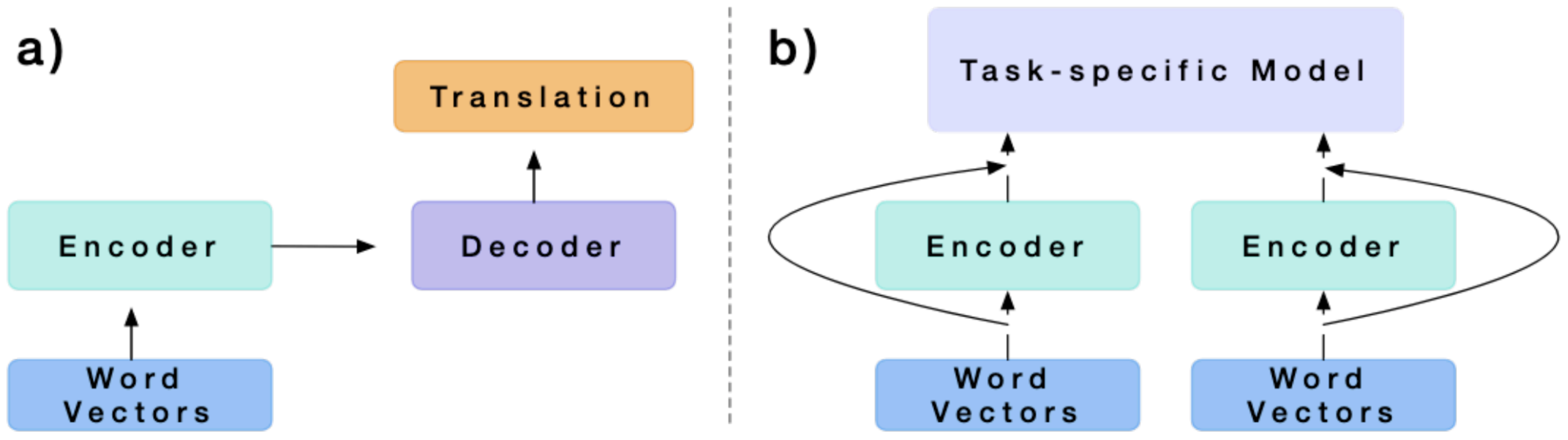
- Solution: Train a model to extract contextual representations on text corpus



Contextual embedding

CoVe (McCann et al., 2017)

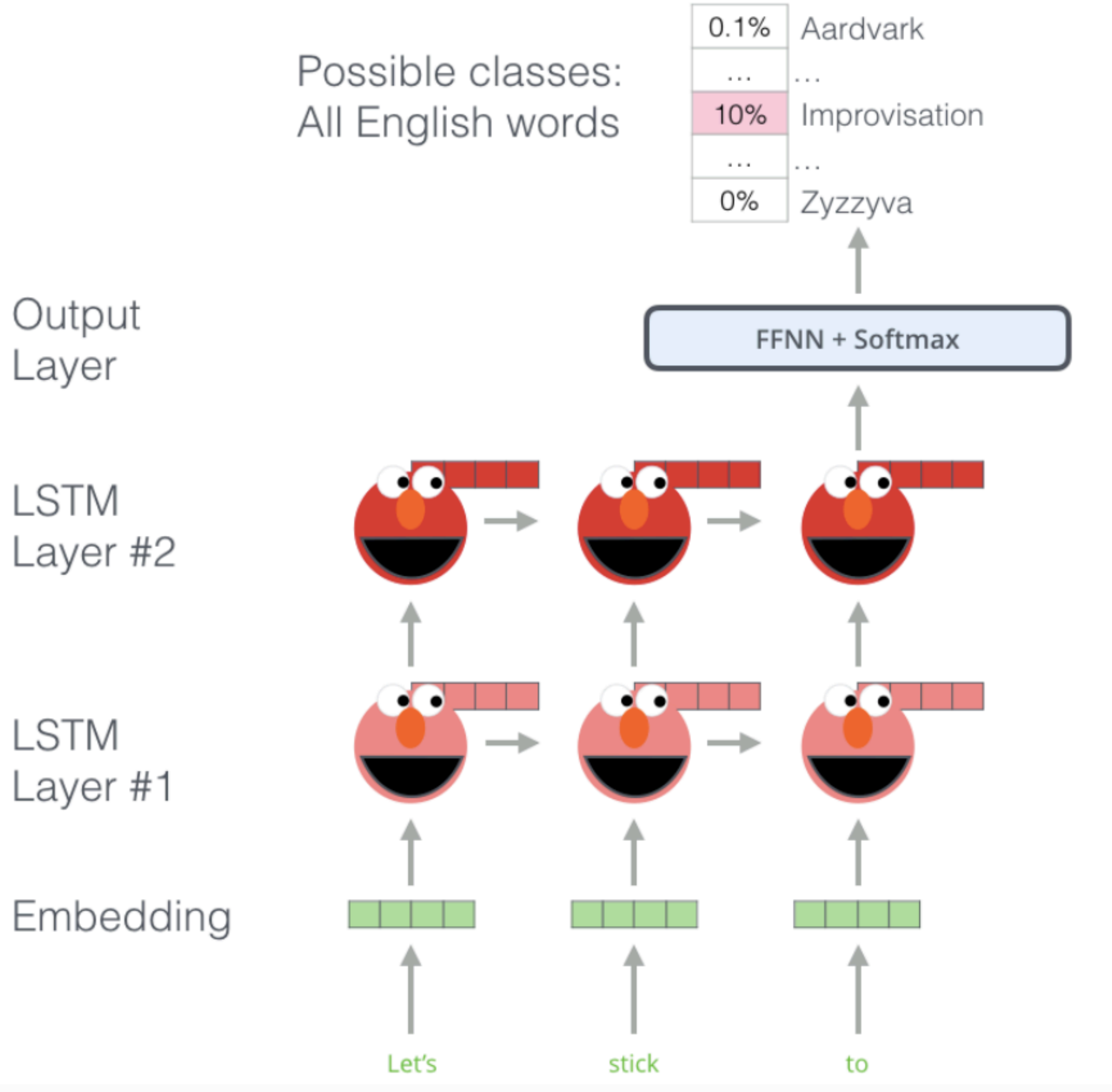
- Key idea: Train a standard neural machine translation model
- Take the encoder directly as contextualized word embeddings
- Problems:
    - Translation requires paired (labeled) data
    - The embeddings are tailored to particular translation corpuses



Contextual embedding

Language model pretraining task

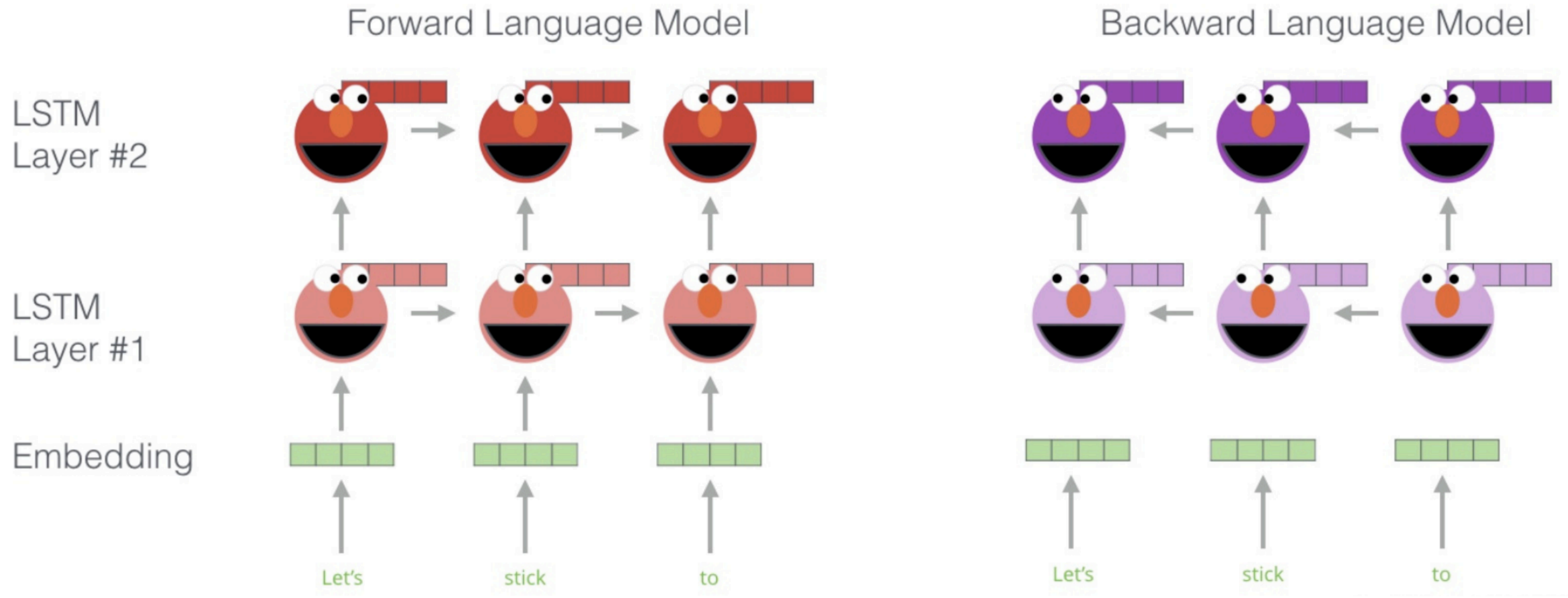
- Predict the next word given the prefix
- Can be defined on any unlabeled document



Contextual embedding

ELMo (Peter et al., 2018)

- Key ideas:
  - Train a foward and backward LSTM language model on large corpus
  - Use the hidden states for each token to compute a vector representation of each word
  - Replace the word embedding by Elmo's embedding (with fixed Elmo's LSTM weights)



Contextual embedding

ELMo results

TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

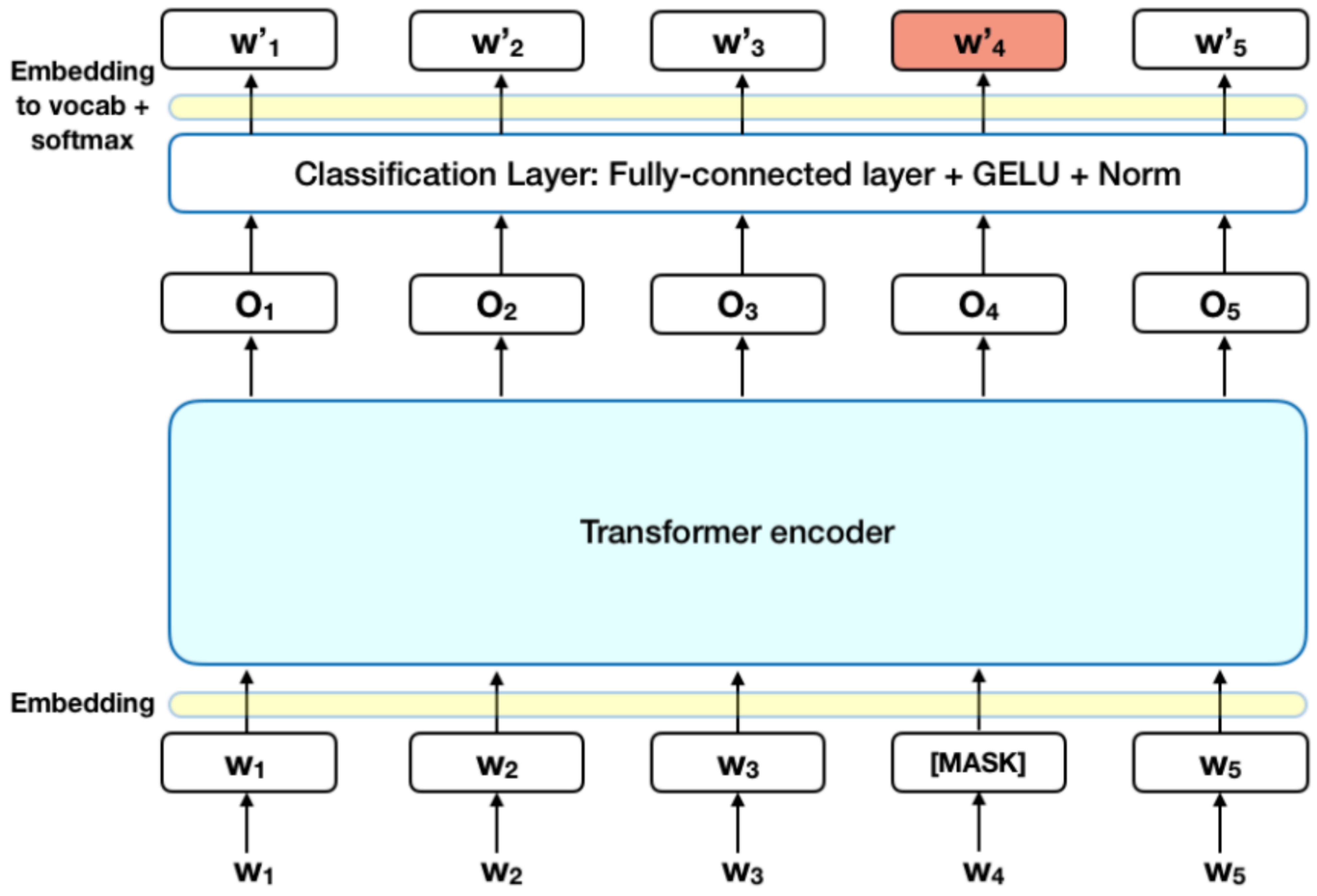
Contextual embedding
BERT

- Key idea: replace LSTM by Transformer
- Define the generated pretraining task by masked language model
- Two pretraining tasks
- Finetune both BERT weights and task-dependent model weights for each task

Contextual embedding

BERT pretraining loss

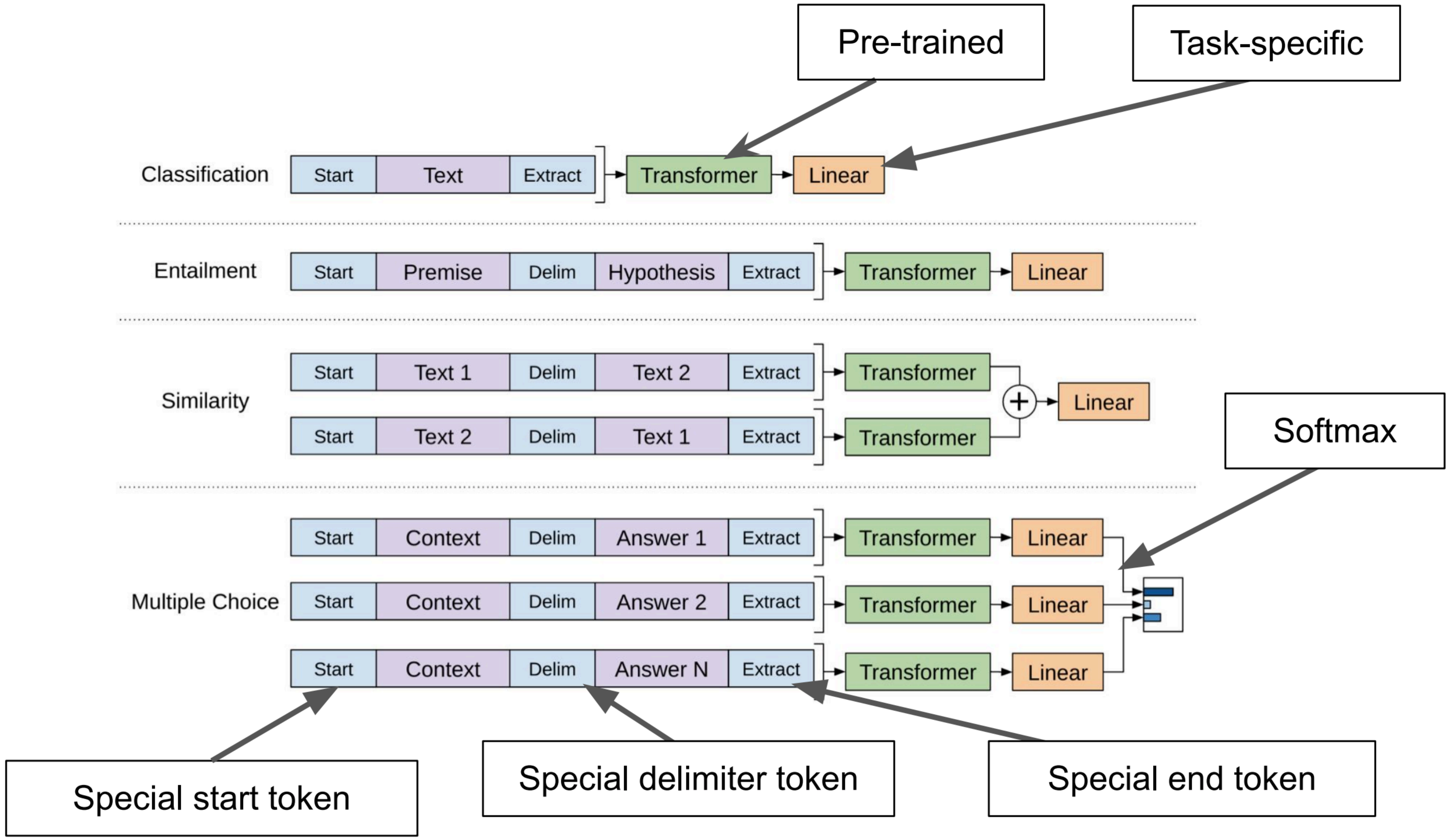
- Masked language model: predicting each word by the rest of sentence
- Next sentence prediction: the model receives pairs of sentences as input and learns to predict if the second sentence is the subsequent sentence in the original document.



Contextual embedding

BERT finetuning

- Keep the pretrained Transformers
- Replace or append a layer for the final task
- Train the whole model based on the task-dependent loss



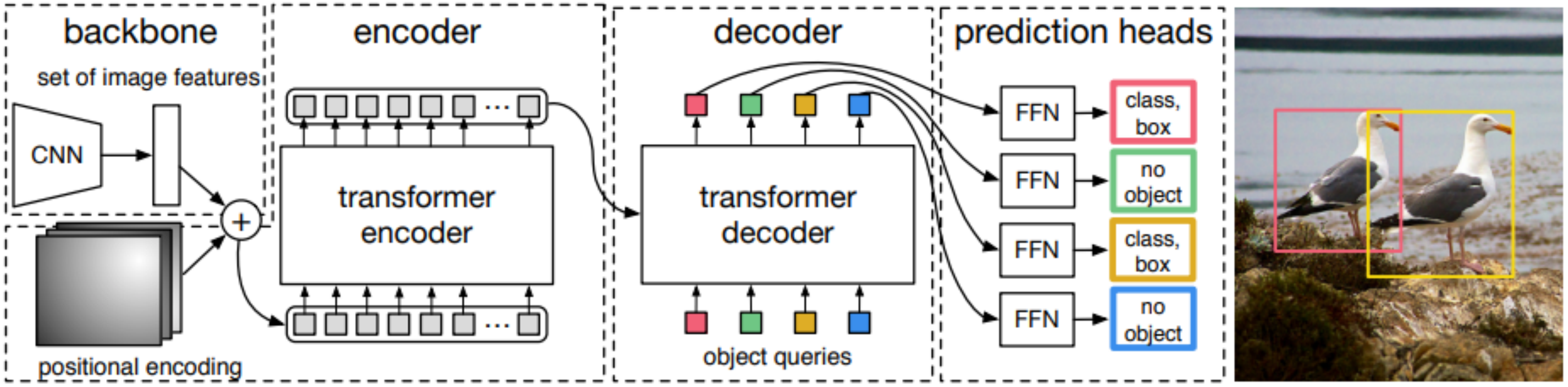
Contextual embedding
BERT results

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
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
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Vision Transformer (ViT)

Attemps on applying self-attention to vision

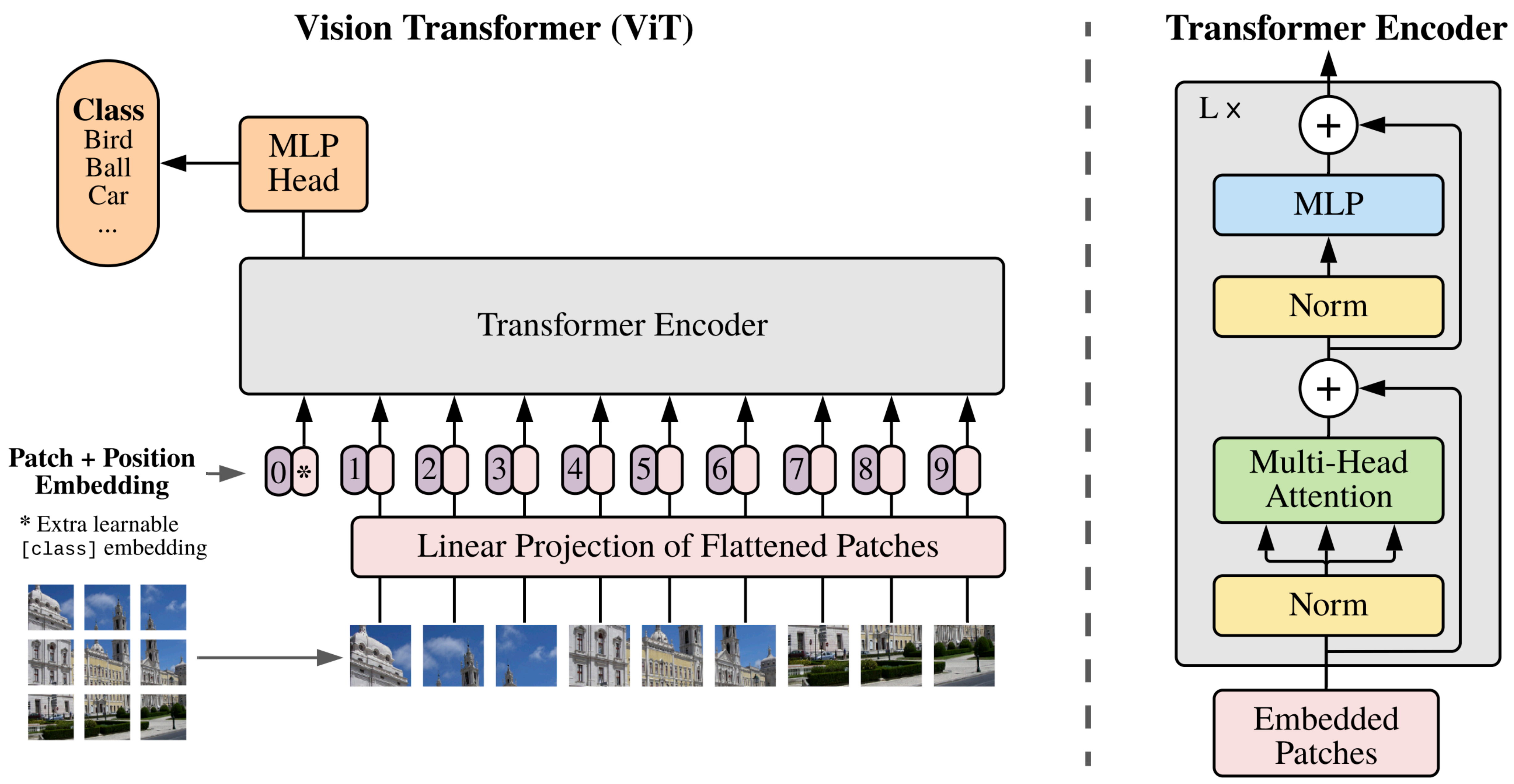
- DETR (Carion et al., 2020): CNN + Self-attention for object detection
- Stand-alone self-attention (Ramachandran et al., 2020)
- …



Vision Transformer (ViT)

Vision Transformer (ViT)

- Partition input image into $K \times K$ patches
- A linear projection to transform each patch to feature (no convolution)
- Pass tokens into Transformer



Vision Transformer (ViT)

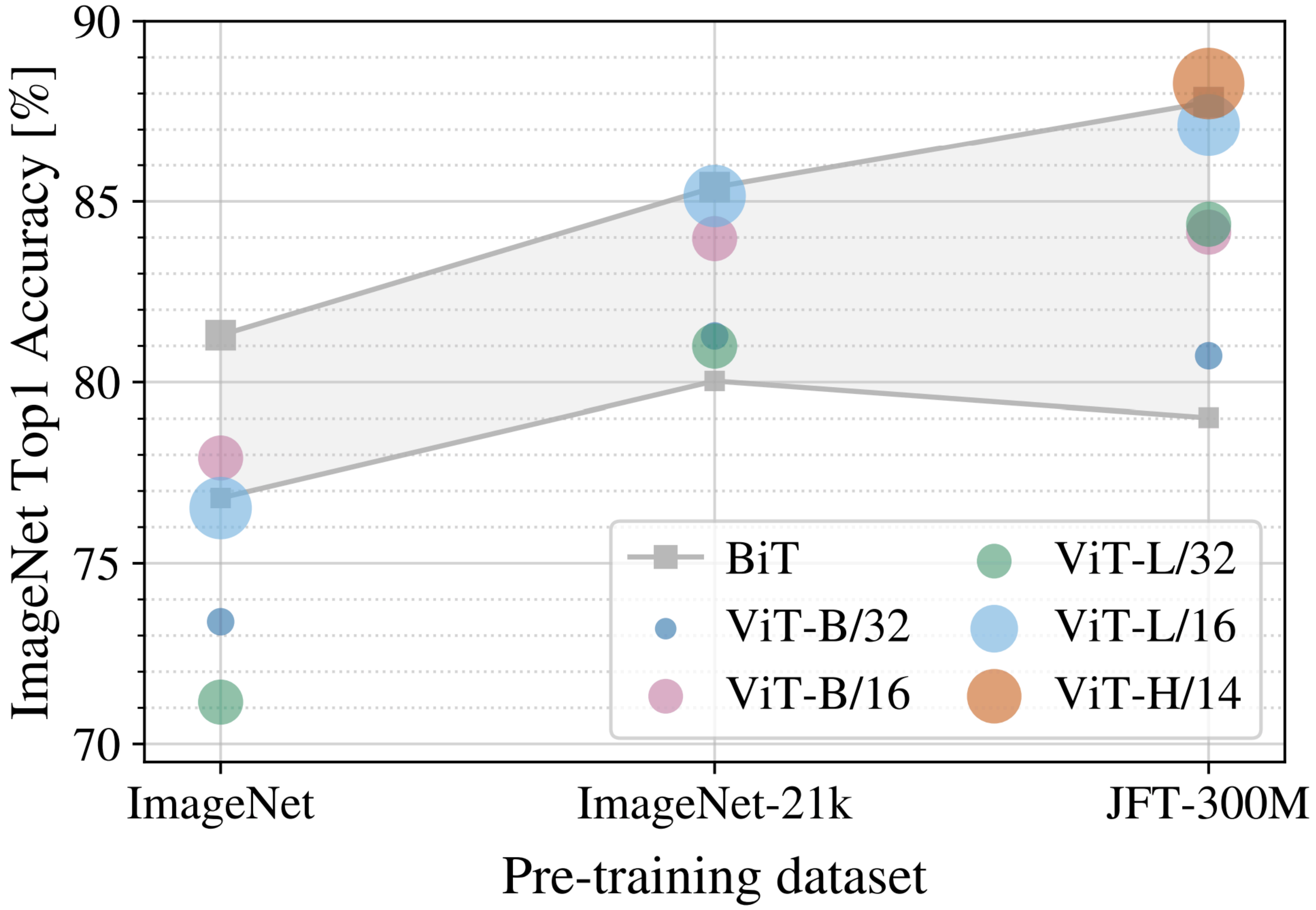
Vision Transformer (ViT)

- Patches are non-overlapping in the original ViT
- $N \times N$ image $\Rightarrow$ $(N/K)^2$ tokens
- Smaller patch size $\Rightarrow$ more input tokens
  - Higher computation (memory) cost, (usually) higher accuracy
- Use 1D (learnable) positional embedding
- Inference with higher resolution:
  - Keep the same patch size, which leads to longer sequence
  - Interpolation for positional embedding

Vision Transformer (ViT)

ViT Performance

- ViT outperforms CNN with large pretraining



Vision Transformer (ViT)

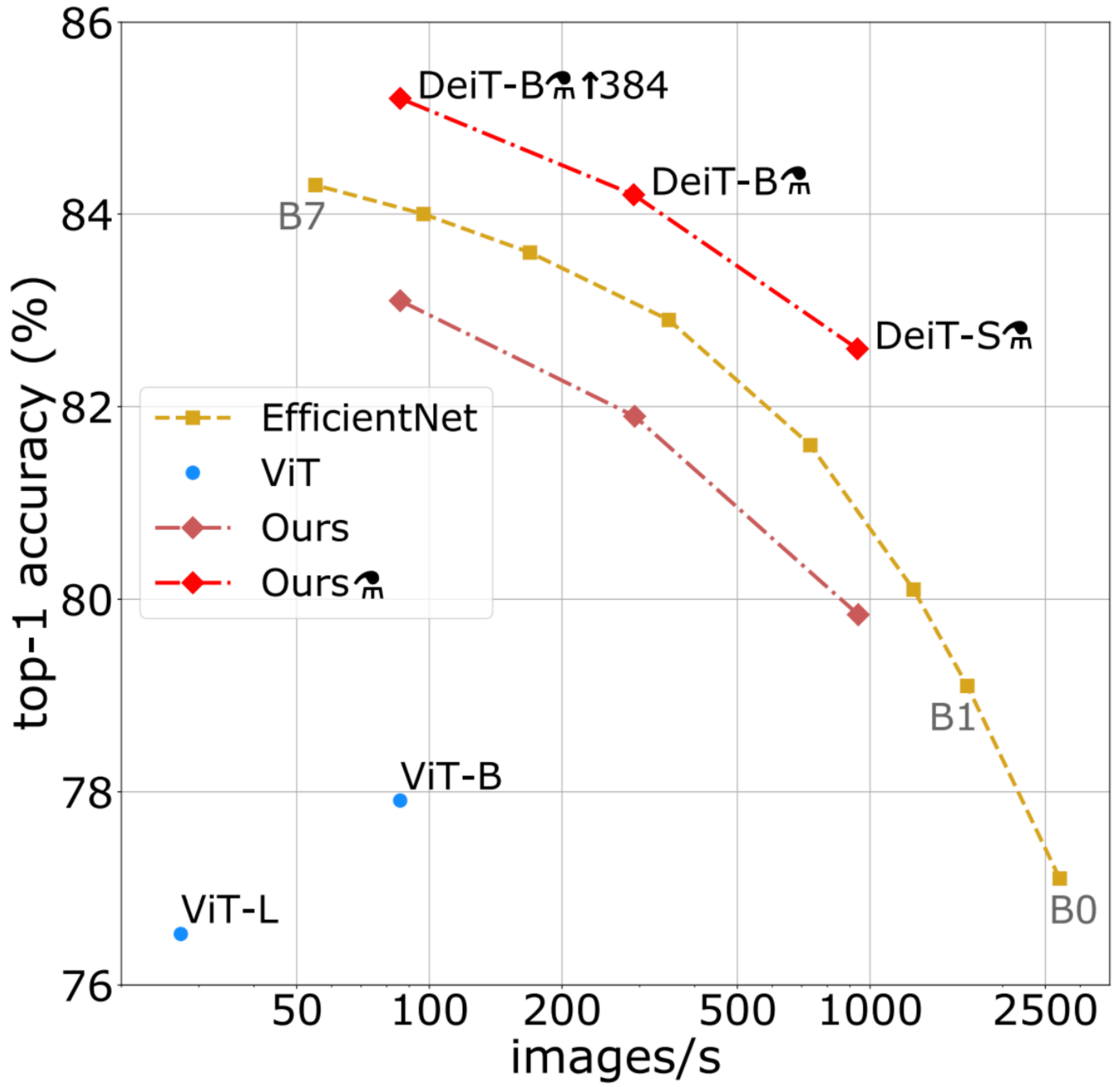
ViT Performance

- Attention maps of ViT (to input)

Vision Transformer (ViT)

ViT v.s. ResNet

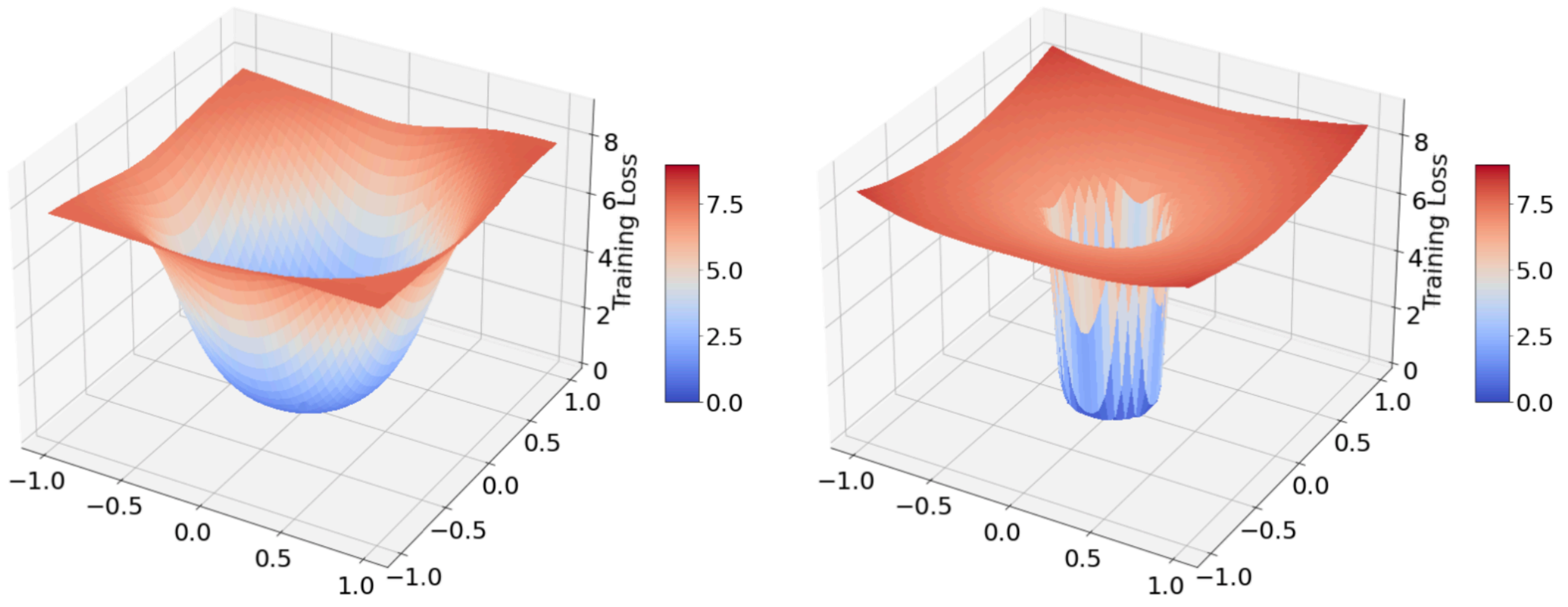
- Can ViT outperform ResNet on ImageNet without pretraining?
- Deit (Touvron et al., 2021):
  - Use very strong data augmentation
  - Use a ResNet teacher and distill to ViT



Vision Transformer (ViT)

ViT v.s. ResNet

- ViT tends to converge to sharper regions than ResNet



(a) ResNet

Leading eigenvalue of Hessian: 179.8

(b) ViT

Leading eigenvalue of Hessian: 738.8

Vision Transformer (ViT)

``Sharpness'' is related to generalization

- Testing can be viewed as a slightly perturbed training distribution
- Sharp minimum ⇒ performance degrades significantly from training to testing

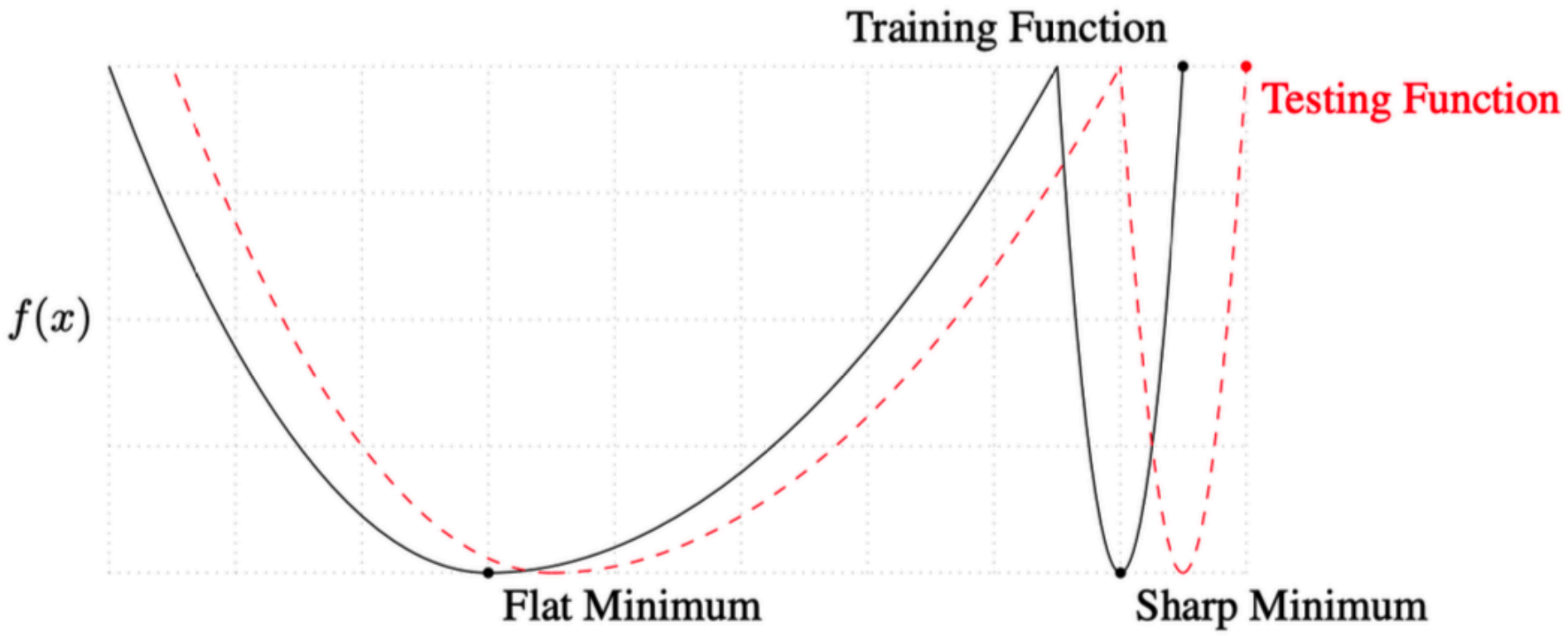


Figure from (Keskar et al., 2017)

Vision Transformer (ViT)

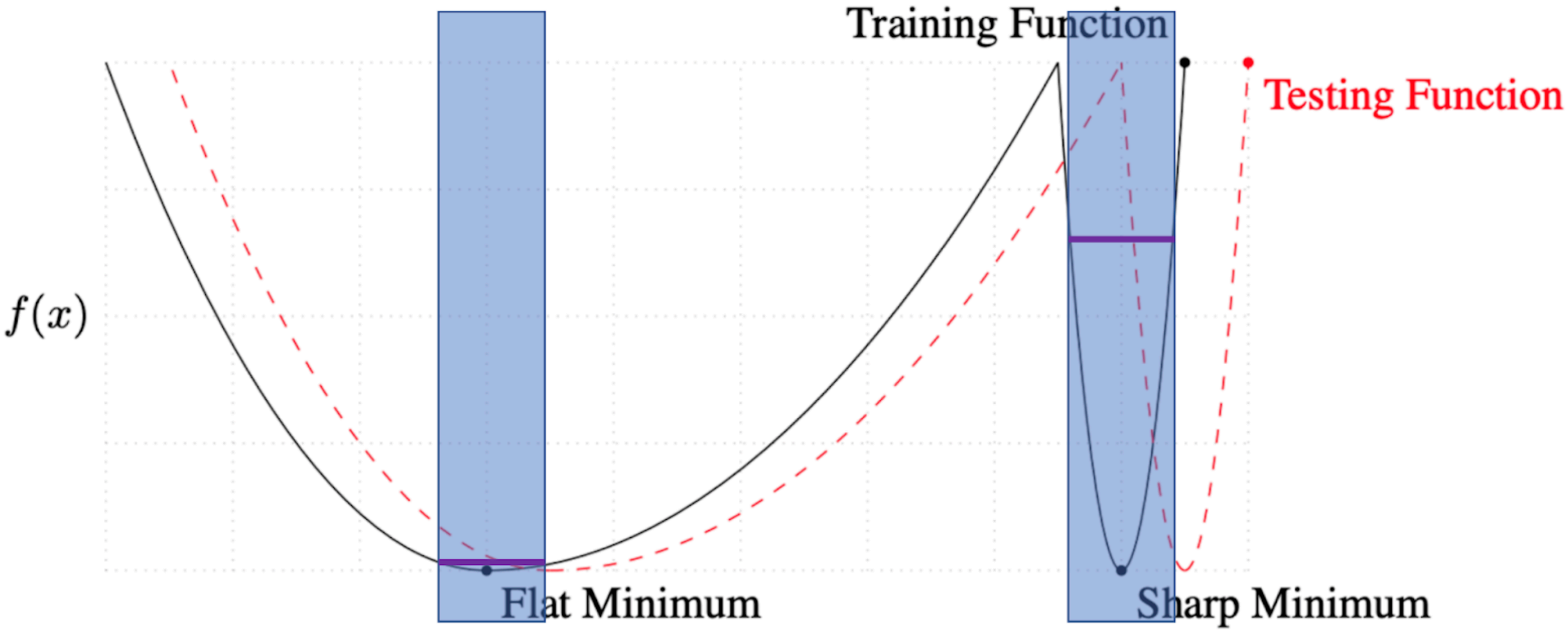
Sharpness Aware Minimization (SAM)

- Optimize the worst-case loss within a small neighborhood
  - $\min_{w} \max_{\|\delta\|_2 \le \epsilon} L(w + \delta)$
  - ϵ is a small constant (hyper-parameter)
- Use 1-step gradient ascent to approximate inner max:
  - $\hat{\delta} = \arg \max_{\|\delta\|_2 \le \epsilon} L(w) + \nabla L(w)^T \delta = \epsilon \frac{\nabla L(w)}{\|\nabla L(w)\|}$
- Conduct the following update for each iteration:
  - $w \leftarrow w - \alpha \nabla L(w + \hat{\delta})$

Vision Transformer (ViT)

Sharpness Aware Minimization (SAM)

- SAM is a natural way to penalize sharpness region (but requires some computational overhead)



Vision Transformer (ViT)

ViT v.s. ResNet

Model	#params	Throughput (img/sec/core)	ImageNet	Real	V2	ImageNet-R	ImageNet-C
ResNet							
ResNet-50-SAM	25M	2161	76.7 (+0.7)	83.1 (+0.7)	64.6 (+1.0)	23.3 (+1.1)	46.5 (+1.9)
ResNet-101-SAM	44M	1334	78.6 (+0.8)	84.8 (+0.9)	66.7 (+1.4)	25.9 (+1.5)	51.3 (+2.8)
ResNet-152-SAM	60M	935	79.3 (+0.8)	84.9 (+0.7)	67.3 (+1.0)	25.7 (+0.4)	52.2 (+2.2)
ResNet-50x2-SAM	98M	891	79.6 (+1.5)	85.3 (+1.6)	67.5 (+1.7)	26.0 (+2.9)	50.7 (+3.9)
ResNet-101x2-SAM	173M	519	80.9 (+2.4)	86.4 (+2.4)	69.1 (+2.8)	27.8 (+3.2)	54.0 (+4.7)
ResNet-152x2-SAM	236M	356	81.1 (+1.8)	86.4 (+1.9)	69.6 (+2.3)	28.1 (+2.8)	55.0 (+4.2)
Vision Transformer							
ViT-S/32-SAM	23M	6888	70.5 (+2.1)	77.5 (+2.3)	56.9 (+2.6)	21.4 (+2.4)	46.2 (+2.9)
ViT-S/16-SAM	22M	2043	78.1 (+3.7)	84.1 (+3.7)	65.6 (+3.9)	24.7 (+4.7)	53.0 (+6.5)
ViT-S/14-SAM	22M	1234	78.8 (+4.0)	84.8 (+4.5)	67.2 (+5.2)	24.4 (+4.7)	54.2 (+7.0)
ViT-S/8-SAM	22M	333	81.3 (+5.3)	86.7 (+5.5)	70.4 (+6.2)	25.3 (+6.1)	55.6 (+8.5)
ViT-B/32-SAM	88M	2805	73.6 (+4.1)	80.3 (+5.1)	60.0 (+4.7)	24.0 (+4.1)	50.7 (+6.7)
ViT-B/16-SAM	87M	863	79.9 (+5.3)	85.2 (+5.4)	67.5 (+6.2)	26.4 (+6.3)	56.5 (+9.9)
MLP-Mixer							
Mixer-S/32-SAM	19M	11401	66.7 (+2.8)	73.8 (+3.5)	52.4 (+2.9)	18.6 (+2.7)	39.3 (+4.1)
Mixer-S/16-SAM	18M	4005	72.9 (+4.1)	79.8 (+4.7)	58.9 (+4.1)	20.1 (+4.2)	42.0 (+6.4)
Mixer-S/8-SAM	20M	1498	75.9 (+5.7)	82.5 (+6.3)	62.3 (+6.2)	20.5 (+5.1)	42.4 (+7.8)
Mixer-B/32-SAM	60M	4209	72.4 (+9.9)	79.0 (+10.9)	58.0 (+10.4)	22.8 (+8.2)	46.2 (12.4)
Mixer-B/16-SAM	59M	1390	77.4 (+11.0)	83.5 (+11.4)	63.9 (+13.1)	24.7 (+10.2)	48.8 (+15.0)
Mixer-B/8-SAM	64M	466	79.0 (+10.4)	84.4 (+10.1)	65.5 (+11.6)	23.5 (+9.2)	48.9 (+16.9)

Vision Transformer (ViT)

ViT v.s. ResNet

- Let's compare one ViT layer vs one convolution layer
- Reception field: (which input neurons can affect an output neuron)
  - CNN: some subarea of image (kernel size)
  - Self-attention: the whole image
  - $\Rightarrow$ there exists self-attention function that cannot be captured by convolution
- Is the function set of self-attention strictly larger than convolution?
  - Yes, given enough attention heads

Vision Transformer (ViT)

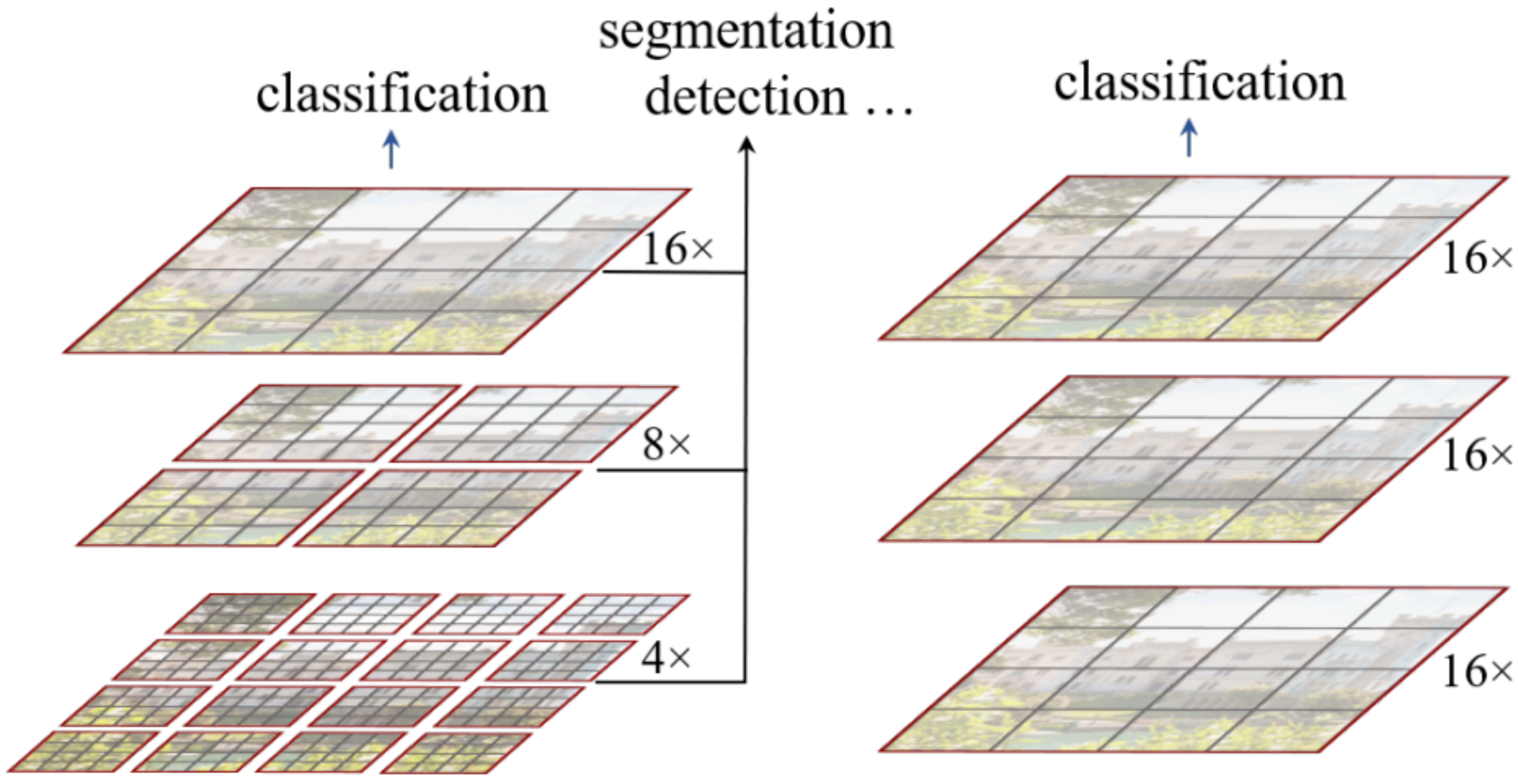
How can self-attention do convolution?

- Consider self-attention with relative positional encoding
  - $$\text{Output} = \text{Softmax}(\underbrace{\frac{QK^T}{\sqrt{d}}}_{\text{context aware}} + \underbrace{B}_{\text{context agnostic}})V$$
  - Q, K, V: query, key, value matrices
  - $B_{i,j} = b_{(x_i - x_j, y_i - y_j)}$: relative positional encoding (trainable scalars)
- To perform convolution: Set $Q, K = 0$ and purely rely on B
- Implication: the positional encoding can capture CNN; the query/key matrices can capture context-aware information beyond convolution

Vision Transformer (ViT)

Swin Transformer (Liu et al., 2021)

- Problems of the original ViT:
  - Non-overlapping partition
  - Only a single resolution
  - Quadratic complexity for attention computation
- Swin Transformer: hierarchical and sliding window partitions

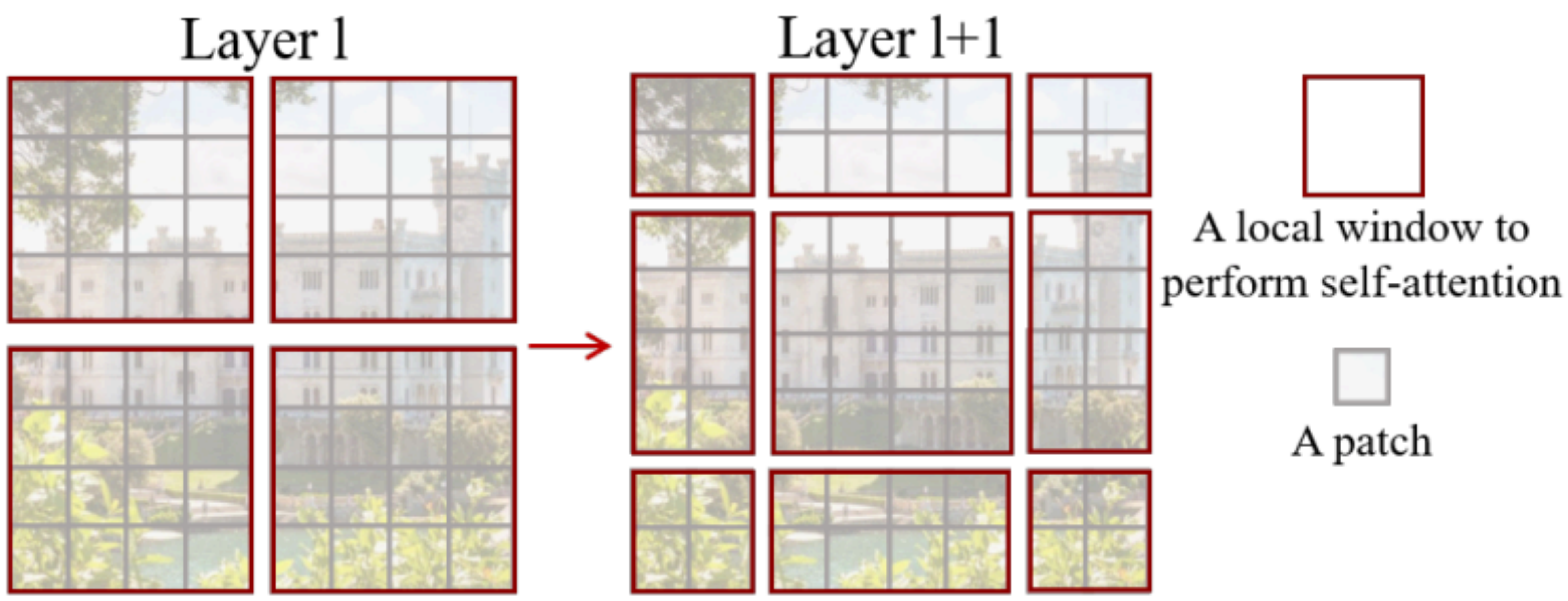


(a) Swin Transformer (ours)   (b) ViT

Vision Transformer (ViT)

Swin Transformer

- Attention within sub-blocks with shifts to avoid huge attention matrix



Vision Transformer (ViT)

Swin Transformer

(a) Regular ImageNet-1K trained models

method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
RegNetY-4G [48]	224^2	21M	4.0G	1156.7	80.0
RegNetY-8G [48]	224^2	39M	8.0G	591.6	81.7
RegNetY-16G [48]	224^2	84M	16.0G	334.7	82.9
EffNet-B3 [58]	300^2	12M	1.8G	732.1	81.6
EffNet-B4 [58]	380^2	19M	4.2G	349.4	82.9
EffNet-B5 [58]	456^2	30M	9.9G	169.1	83.6
EffNet-B6 [58]	528^2	43M	19.0G	96.9	84.0
EffNet-B7 [58]	600^2	66M	37.0G	55.1	84.3
ViT-B/16 [20]	384^2	86M	55.4G	85.9	77.9
ViT-L/16 [20]	384^2	307M	190.7G	27.3	76.5
DeiT-S [63]	224^2	22M	4.6G	940.4	79.8
DeiT-B [63]	224^2	86M	17.5G	292.3	81.8
DeiT-B [63]	384^2	86M	55.4G	85.9	83.1
Swin-T	224^2	29M	4.5G	755.2	81.3
Swin-S	224^2	50M	8.7G	436.9	83.0
Swin-B	224^2	88M	15.4G	278.1	83.5
Swin-B	384^2	88M	47.0G	84.7	84.5

(b) ImageNet-22K pre-trained models

method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
R-101x3 [38]	384^2	388M	204.6G	-	84.4
R-152x4 [38]	480^2	937M	840.5G	-	85.4
ViT-B/16 [20]	384^2	86M	55.4G	85.9	84.0
ViT-L/16 [20]	384^2	307M	190.7G	27.3	85.2
Swin-B	224^2	88M	15.4G	278.1	85.2
Swin-B	384^2	88M	47.0G	84.7	86.4
Swin-L	384^2	197M	103.9G	42.1	87.3