Chapter 6

Transformers

1. **Attention Layers-Modules**
2. **Transformers (Vanilla)**
3. **Vision Based Transformers**

Attention Layers-Modules

An improving mechanism for RNNs by applying an interface connecting the encoder and decoder

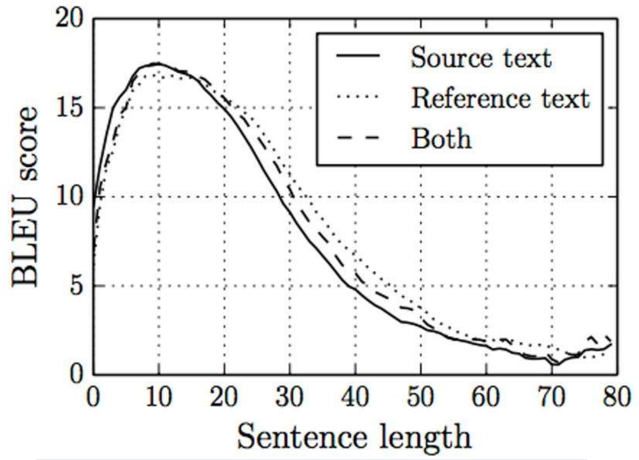
- In translation machines, LSTM/GRU as a seq2seq model can not predict successful translation for a sentence, when its length increases.

What is attention layer in Deep Learning?

- Every RNN/LSTM can be interpreted as an Encoder and Decoder.
  - Encoder encodes the input samples of a chronological signal to hidden state.
  - Decoder decodes the hidden state to the output.

Attention is an interface connecting the encoder and decoder that provides the decoder with information from every encoder hidden state.

With this framework, the model is able to selectively focus on valuable parts of the input sequence and hence, learn the association between them.



BLEU:
Bilingual Evaluation Understudy

*Neural Machine Translation by Jointly Learning to Align and Translate

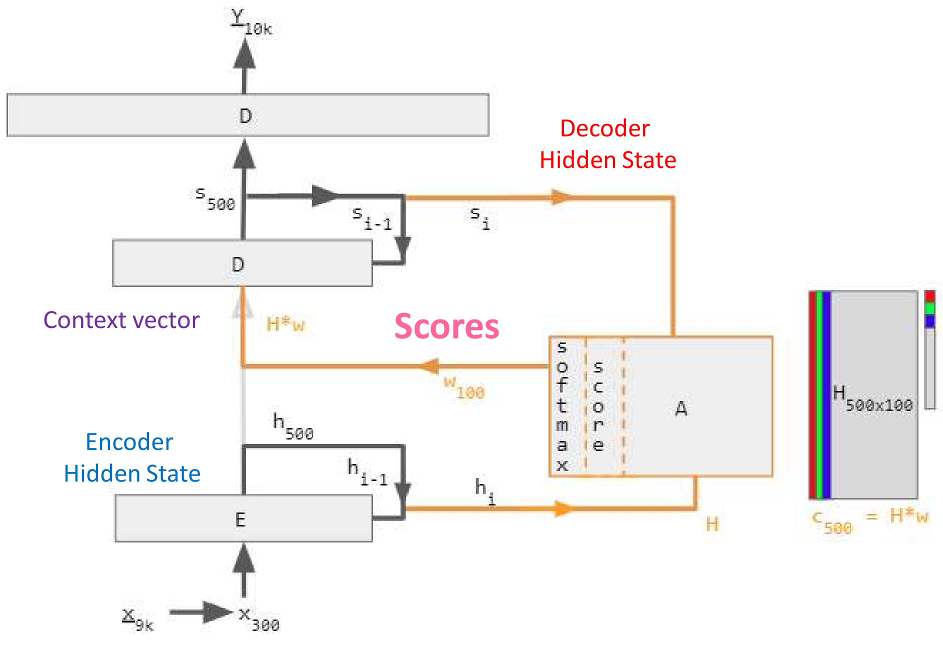
Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio

A language translation example

To build a machine that translates English-to-French (see the shown diagram), one starts with an Encoder-Decoder and grafts an attention unit to it. In the simplest case such as the shown example, the attention unit is just lots of dot products of recurrent layer states and does not need training. In practice, the attention unit consists of 3 fully connected neural network layers that needs to be trained. The 3 layers are called Query, Key, and Value.

Encoder-Decoder with attention. This diagram uses specific values to relieve an already cluttered notation alphabet soup. The left part (in black) is the Encoder-Decoder, the middle part (in orange) is the attention unit, and the right part (in grey & colors) is the computed data. Grey regions in H matrix and w vector are zero values.

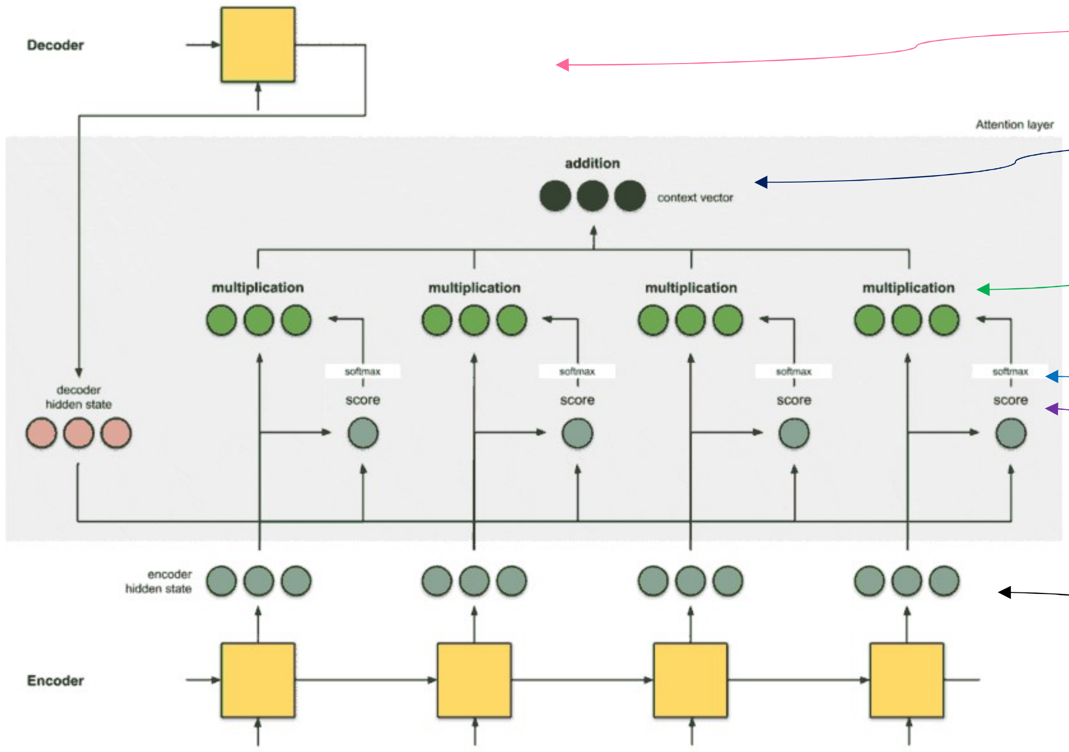
Numerical subscripts are examples of vector sizes. Lettered subscripts i and i-1 indicate time step.



label	description
100	max sentence length
300	embedding size (word dimension)
500	length of hidden vector
9k, 10k	dictionary size of input & output languages respectively.
x, Y	9k and 10k 1-hot dictionary vectors. x → x implemented as a lookup table rather than vector multiplication. Y is the 1-hot maximizer of the linear Decoder layer D; that is, it takes the argmax of D's linear layer output.
x	300-long word embedding vector. The vectors are usually pre-calculated from other projects such as GloVe or Word2Vec.
h	500-long encoder hidden vector. At each point in time, this vector summarizes all the preceding words before it. The final h can be viewed as a "sentence" vector, or a thought vector as Hinton calls it.
s	500-long decoder hidden state vector.
E	500 neuron RNN encoder. 500 outputs. Input count is 800–300 from source embedding + 500 from recurrent connections. The encoder feeds directly into the decoder only to initialize it, but not thereafter; hence, that direct connection is shown very faintly.
D	2-layer decoder. The recurrent layer has 500 neurons and the fully connected linear layer has 10k neurons (the size of the target vocabulary).[7] The linear layer alone has 5 million (500 * 10k) weights -- ~10 times more weights than the recurrent layer.
score	100-long alignment score
w	100-long vector attention weight. These are "soft" weights which changes during the forward pass, in contrast to "hard" neuronal weights that change during the learning phase.
A	Attention module — this can be a dot product of recurrent states, or the Query-Key-Value fully connected layers. The output is a 100-long vector w.
H	500x100. 100 hidden vectors h concatenated into a matrix
c	500-long context vector = H * w. c is a linear combination of h vectors weighted by w.

Unfolded attention layer

The below figure demonstrates an Encoder-Decoder architecture with an attention layer (All time steps).



5-Feed the context vector into the decoder.

4-Sum the alignment vectors (as context vector)

3- Multiply each encoder hidden state by its softmax score.

2-Run all the scores through a softmax layer

1-Obtain a score for every encoder hidden state.

0-Prepare hidden states.

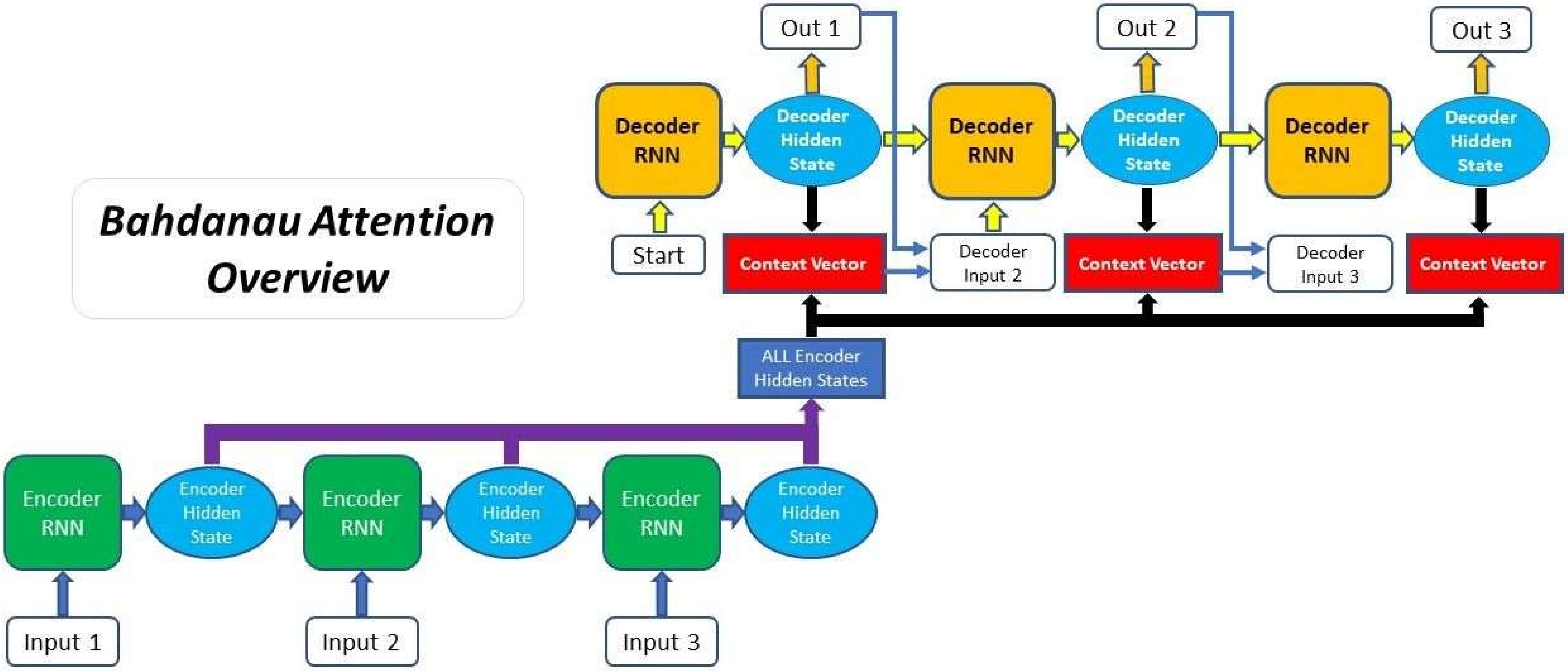
***Encoder-Decoder Recurrent Neural Network Models for Neural Machine Translation**
by Jason Brownlee on January 1, 2018 in Deep Learning for Natural Language Processing

The implementations of an attention layer can be broken down into 4 steps.

For each time step:

- **Step 0: Prepare hidden states.**
- First, prepare all the available encoder hidden states (green) and the first decoder hidden state (red). In our example, we have 4 encoder hidden states and the current decoder hidden state. (Note: the last consolidated encoder hidden state is fed as input to the first time step of the decoder. The output of this first time step of the decoder is called the first decoder hidden state.)
- **Step 1: Obtain a score for every encoder hidden state.**
- A score (scalar) is obtained by a score function (also known as alignment score function or alignment model). In this example, the score function is a dot product between the decoder and encoder hidden states.
- **Step 2: Run all the scores through a softmax layer.**
- We put the scores to a softmax layer so that the softmax scores (scalar) add up to 1. These softmax scores represent the attention distribution.
- **Step 3: Multiply each encoder hidden state by its softmax score.**
- By multiplying each encoder hidden state with its softmax score (scalar), we obtain the alignment vector or the annotation vector. This is exactly the mechanism where alignment takes place.
- **Step 4: Sum the alignment vectors.**
- The alignment vectors are summed up to produce the context vector. A context vector is an aggregated information of the alignment vectors from the previous step.
- **Step 5: Feed the context vector into the decoder.**

Bahdanau Attention Overview



*Neural Machine Translation by Jointly Learning to Align and Translate (2014-2016)
Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio

Attention variants

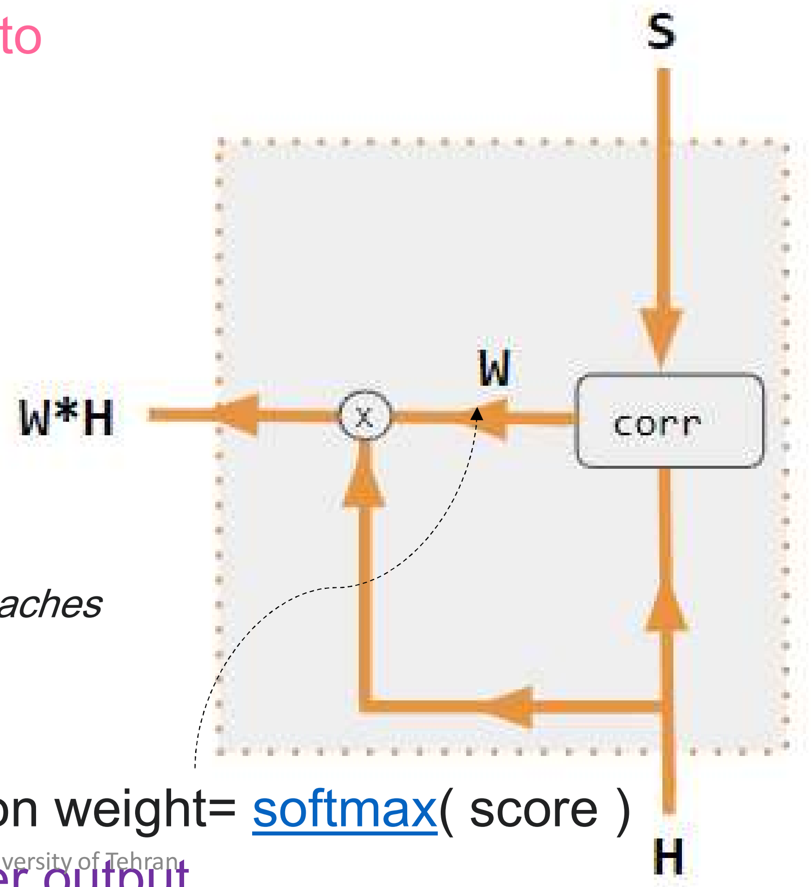
1. encoder-decoder dot product
2. encoder-decoder QKV (Query-Key-Value)
3. encoder-only dot product
4. encoder-only QKV
5. Pytorch tutorial

1. encoder-decoder dot product

s = decoder input to Attention
=Decoder hidden state

Both Encoder & Decoder are needed to calculate Attention.

c = context vector = H*w



1.*Luong, Minh-Thang (2015-09-20). "Effective Approaches to Attention-based Neural Machine Translation". arXiv:1508.04025v5 [cs.CL].*

w = attention weight= softmax(score)

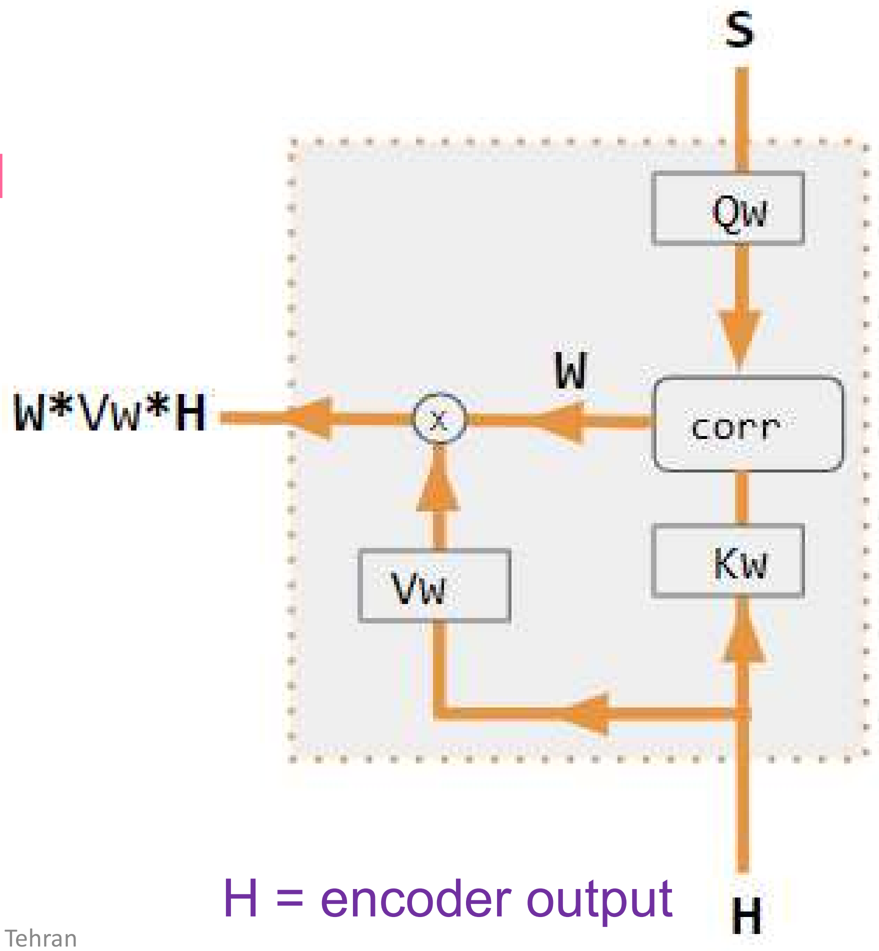
H = encoder output

2. encoder-decoder QKV

Both Encoder & Decoder are needed to calculate Attention.

s = decoder input to Attention

c = context vector

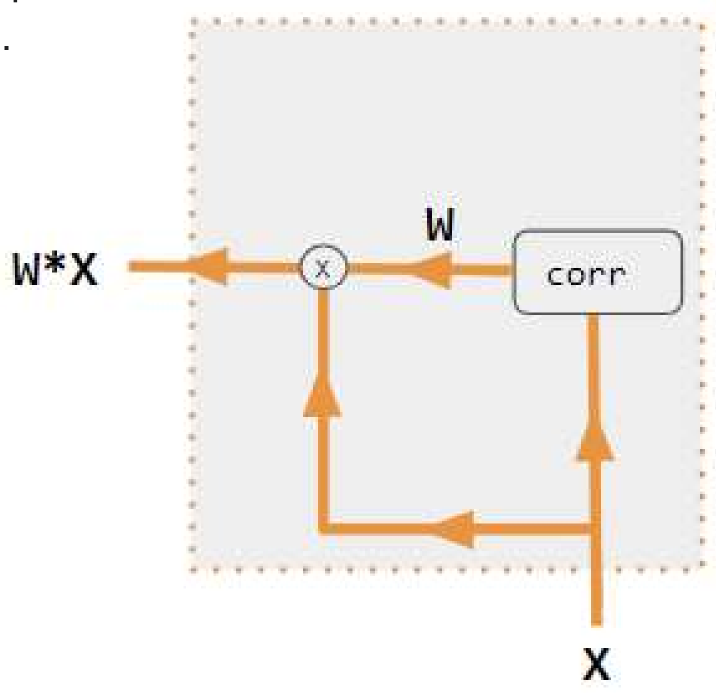


H = encoder output

1. *Neil Rhodes (2021). CS 152 NN—27: Attention: Keys, Queries, & Values. Event occurs at 06:30. Retrieved 2021-12-22.*

3. encoder-only dot product

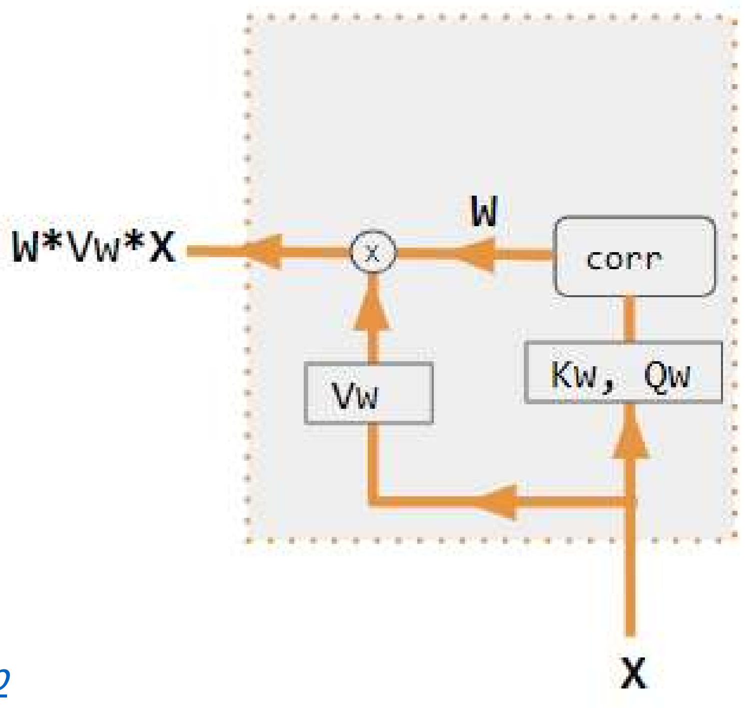
Decoder is NOT used to calculate Attention. With only 1 input into corr, W is an auto-correlation of dot products. $w_{ij} = x_i * x_j$ [10]



1. *Alfredo Canziani & Yann Lecun (2021). NYU Deep Learning course, Spring 2020. Event occurs at 05:30. Retrieved 2021-12-22.*

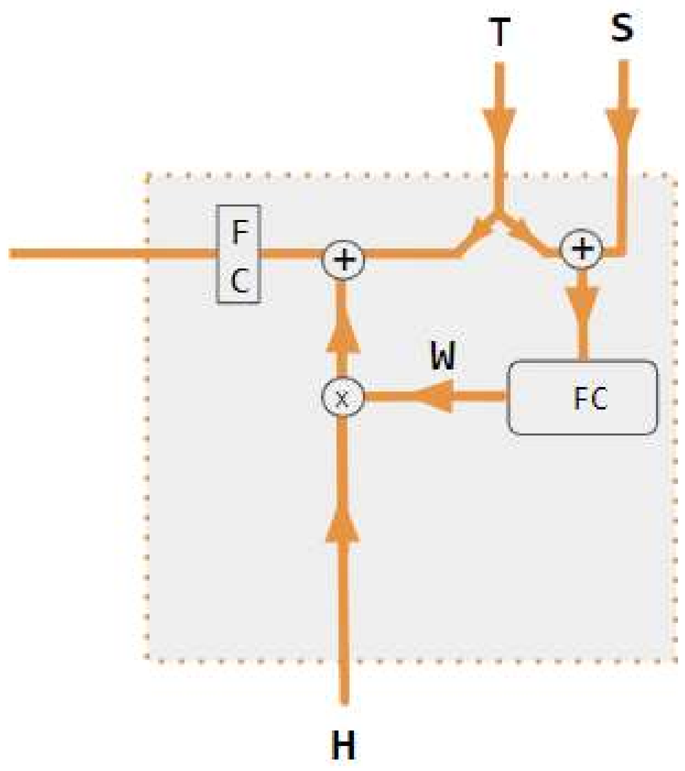
4. encoder-only QKV

Decoder is NOT used to calculate Attention.[11]



1. *Alfredo Canziani & Yann Lecun (2021). NYU Deep Learning course, Spring 2020. Event occurs at 20:15. Retrieved 2021-12-22.*

5. Pytorch tutorial



A FC layer is used to calculate Attention instead of dot product correlation.

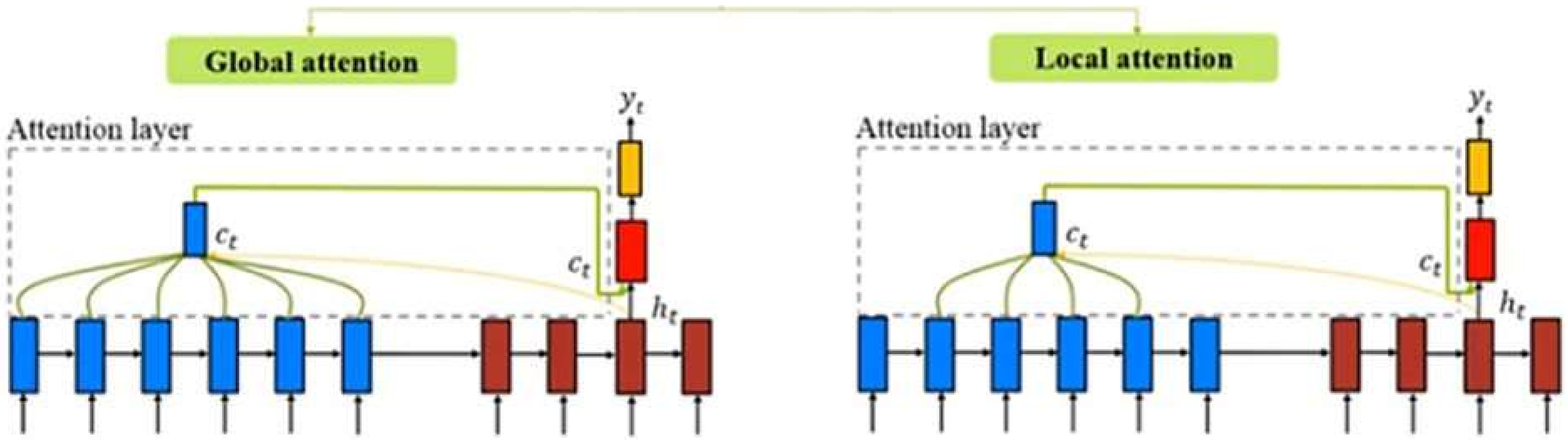
S = decoder hidden state, T = target word embedding.

H = encoder hidden state, X = input word embedding

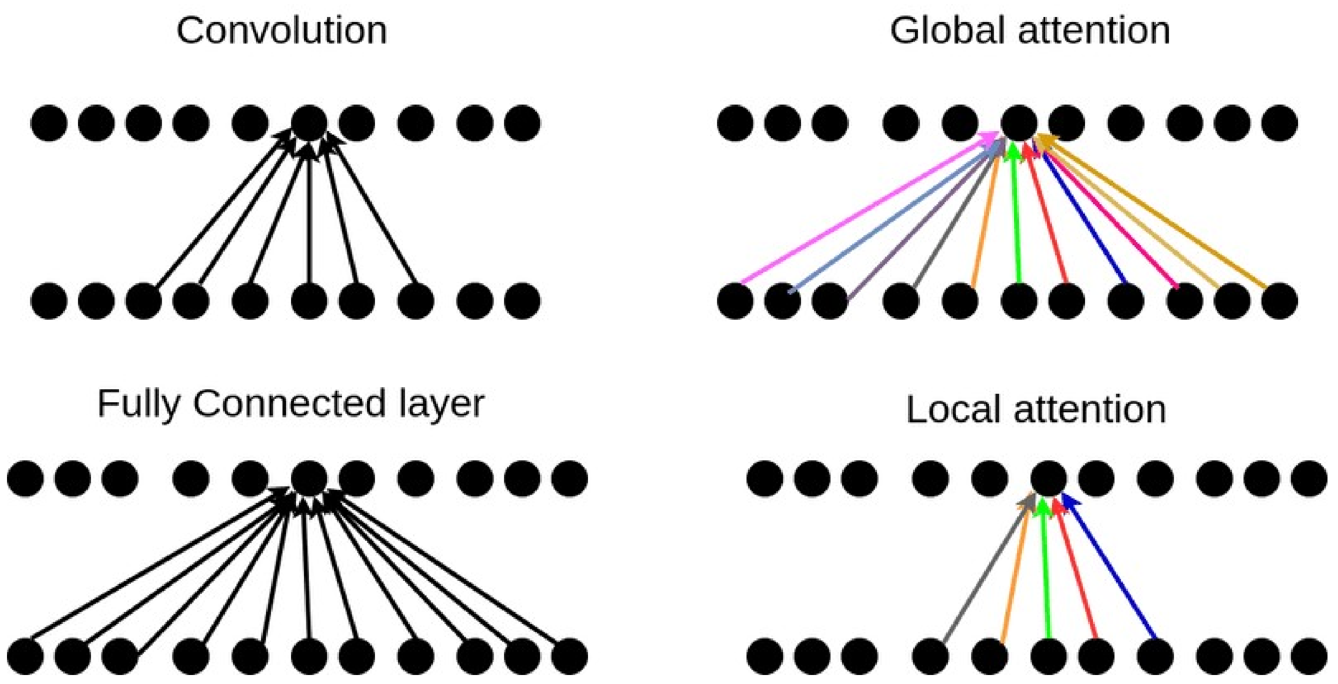
1. *Robertson, Sean. "NLP From Scratch: Translation With a Sequence To Sequence Network and Attention". pytorch.org. Retrieved 2021-12-22.*

Types of attention

Depending on how many source states contribute while deriving the attention vector (α), there can be three types of attention mechanisms:



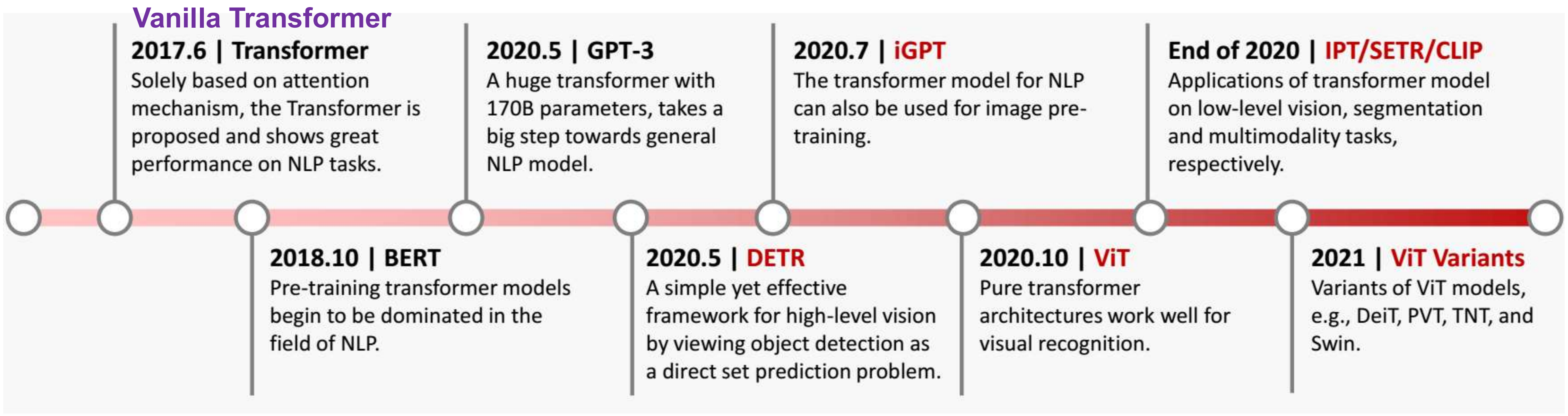
•**Global Attention**: When attention is placed on all source states. In global attention, we require as many weights as the source sentence length.
•**Local Attention**: When attention is placed on a few source states.
•**Hard Attention**: When attention is placed on only one source state.
You can check out my Kaggle notebook or GitHub repo to implement NMT with the attention mechanism using TensorFlow.


Convolution
Global attention
Fully Connected layer
Local attention

1.2.4 Transformers

A transformer is a deep learning model that generates significant weights to each part of the input data through using mechanisms of self-attention and cross-attention.
It is used primarily in the fields of natural language processing (NLP), computer vision (CV), and speech processing.

- TimeLine of Key milestones in the development of transformer * Kai Han Feb2022



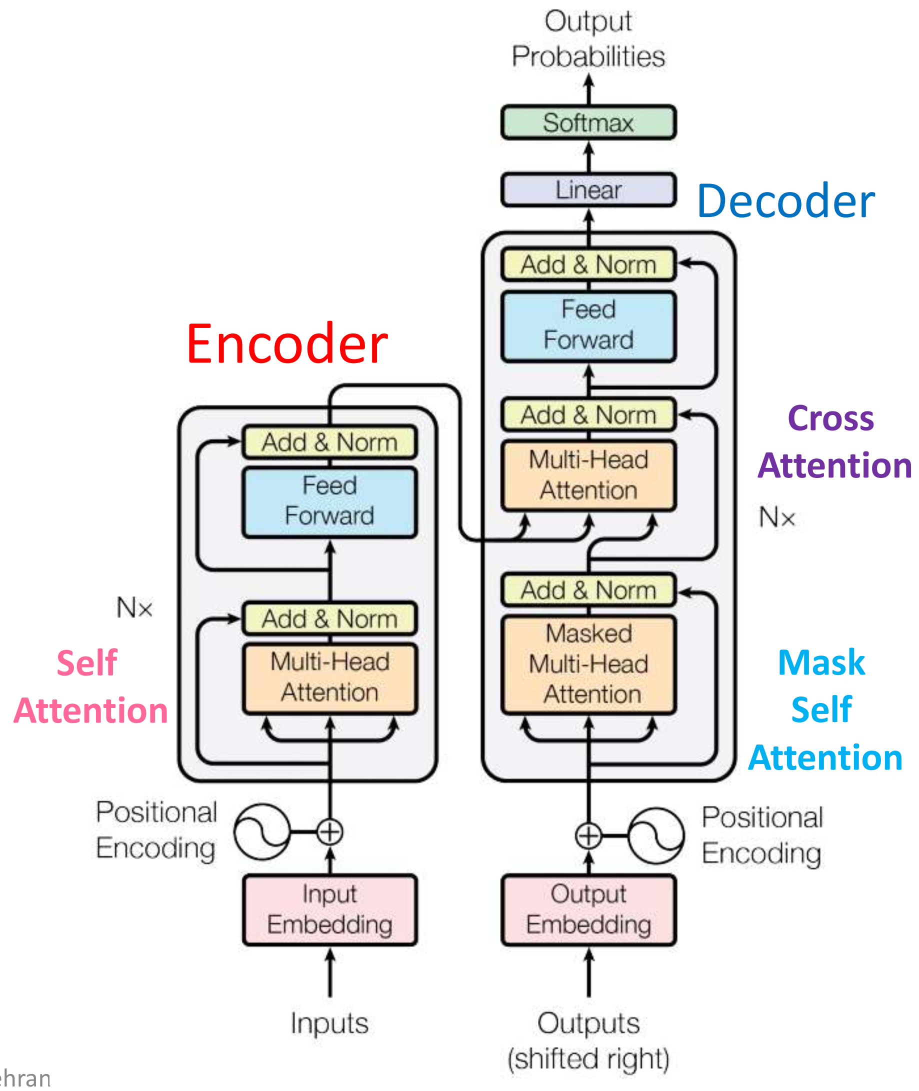
Vanilla Transformer
2017.6 | Transformer
Solely based on attention mechanism, the Transformer is proposed and shows great performance on NLP tasks.
2018.10 | BERT
Pre-training transformer models begin to be dominated in the field of NLP.
2020.5 | GPT-3
A huge transformer with 170B parameters, takes a big step towards general NLP model.
2020.5 | DETR
A simple yet effective framework for high-level vision by viewing object detection as a direct set prediction problem.
2020.7 | iGPT
The transformer model for NLP can also be used for image pre-training.
2020.10 | ViT
Pure transformer architectures work well for visual recognition.
End of 2020 | IPT/SETR/CLIP
Applications of transformer model on low-level vision, segmentation and multimodality tasks, respectively.
2021 | ViT Variants
Variants of ViT models, e.g., DeiT, PVT, TNT, and Swin.

The vision transformer models are marked in red.

Vanilla Transformer

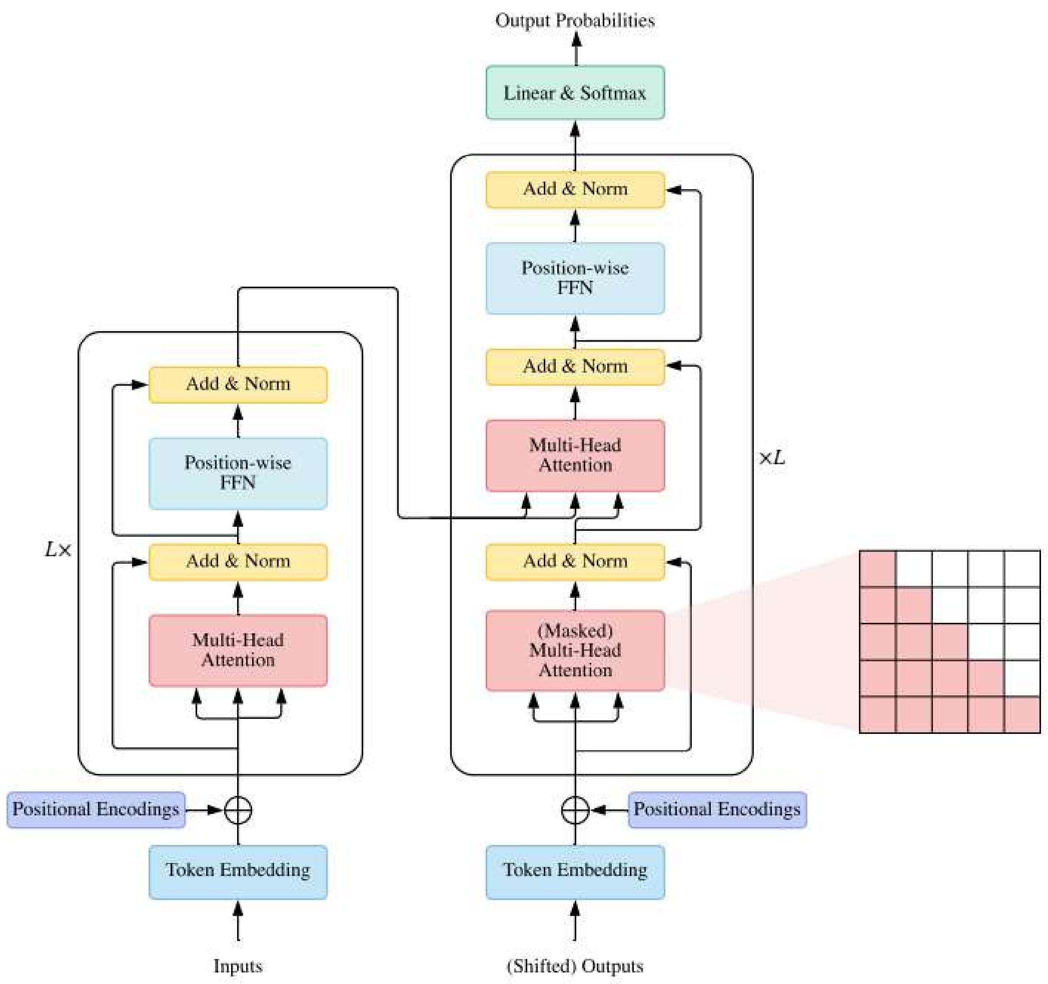
*Vaswani...2017 (Attention is all you need)

- a sequence-to-sequence model and consists of an encoder and a decoder, each of which is a stack of "N" identical blocks each *encoder block* is mainly composed of a multi-head self-attention module and a position-wise feed-forward network (FFN). In this work, the encoder is composed of a stack of N=6 identical layers.
- For building a deeper model, a residual connection is employed around each module, followed by Layer Normalization module.
- Compared to the encoder blocks, decoder blocks additionally insert cross-attention modules between the multi-head self-attention modules and the position-wise FFNs. In this work, the decoder is also composed of a stack of N=6 identical layers.
- Furthermore, the self-attention modules in the decoder are adapted to prevent each position from attending to subsequent positions.



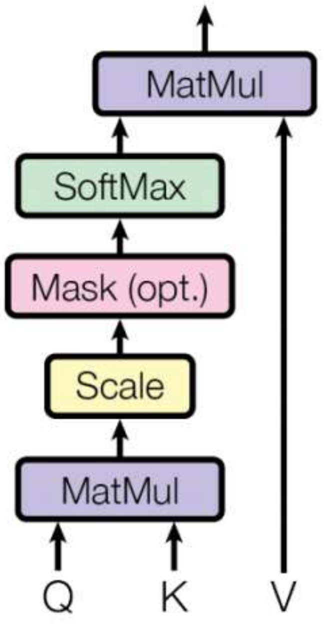
The key modules of the vanilla Transformer

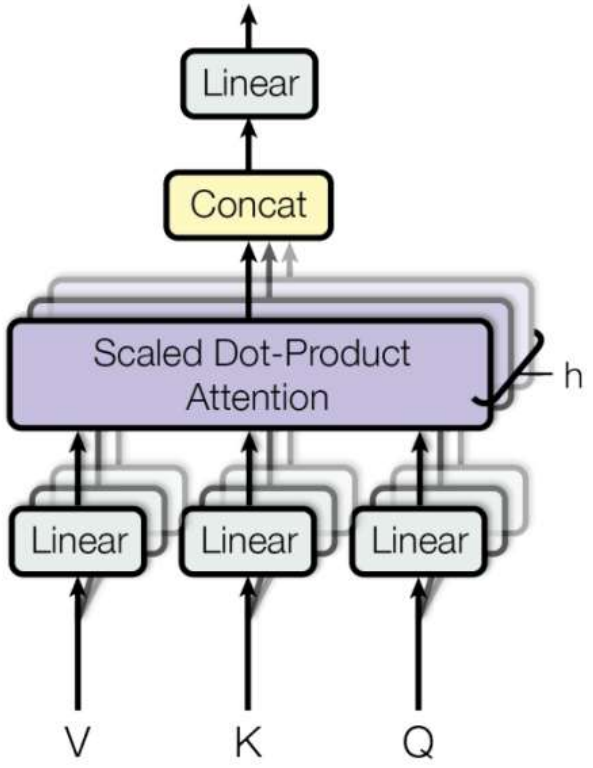
1. **Attention Modules**
   - Single Attention
   - Multi Head-Attention
     - Self-attention
     - Masked Self-attention
     - Cross-attention
2. **Position-wise FFN** (Feed Forward Network)
3. **Residual Connection and Normalization.**
4. **Position Encodings**.



Overview of vanilla Transformer architecture

In this work we employ h=8 parallel attention layers, or heads





Single-attention function Multi-head attention function.

1. Attention Modules

- **Single attention function**

Transformer adopts attention mechanism with Query-Key-Value (QKV) model. Given the packed matrix representations of queries , keys , and values , the scaled dot-product attention used by Transformer is given by

$$A = softmax(\frac{QK^T}{\sqrt{D_k}}) \text{ , Q} \in R^{N\times D_k} \text{ ,K} \in R^{M\times D_k}\text{,V} \in R^{M\times D_v}$$

N: length of Query, M: length of Keys(or values)- D_k: Dimension of query and key D_v: Dimension of values

- Softmax is applied in a row-wise manner.
- The dot-products of queries and keys are divided by $\sqrt{D_k}$ to alleviate gradient vanishing problem of the softmax function.

- **Multi head attention function**

Instead of simply applying a single attention function, Transformer uses multi-head attention, where the original queries, keys and values are with H different sets of learned projections.

$$MultiHeadAtn(Q,K,V) \ = \ Concat(head_1,\cdots,head_H)W^o$$

$$head_i = Attention\left(QW_i^Q,KW_i^K,VW_i^V\right) \ i = 1,2,\ldots,H$$

The model then concatenates all the outputs.

Three types of attention (in terms of the source of queries and key-value pairs).

- *Self-attention*

  In Transformer encoder, we set Q = K = V = X, where X is the outputs of the previous layer.

- *Masked Self-attention*

  In the Transformer decoder, the self-attention is restricted such that queries at each position can only attend to all key-value pairs up to and including that position.

  To enable parallel training, this is typically done by applying a mask function to the un-normalized attention matrix $\hat{A} = \exp(\frac{QK^T}{\sqrt{D_k}})$ where the illegal positions are masked out by setting $\hat{A}_{ij} = -\infty \; if \; i < j$ This kind of self-attention is often referred to as autoregressive or causal attention

- *Cross-attention*

  The queries are projected from the outputs of the previous (decoder) layer, whereas the keys and values are projected using the outputs of the encoder.

2. Position-wise FFN.

The position-wise FFN is a fully connected feed-forward module that operates separately and identically on each position

$$\mathrm{FFN}(\mathrm{H}') = \mathrm{ReLU}(\mathrm{H}'\mathrm{W}^1 + \mathrm{b}^1)\mathrm{W}^2 + \mathrm{b}^2$$

where H' is the outputs of previous layer, and $W^1 \in R^{D_m \times D_f}$, $W^2 \in R^{D_f \times D_m}$, $b^1 \in R^{D_f}$, $b^2 \in R^{D_m}$ are trainable parameters. Typically the intermediate dimension D_ff the FFN is set to be larger than D_m

3. Residual Connection and Normalization

In order to build a deep model, Transformer employs a residual connection [49] around each module, followed by Layer Normalization [4]. For instance, each Transformer encoder block may be written as

$$\mathrm{H}' = \mathrm{LayerNorm}(\mathrm{SelfAtention}(\mathrm{X}) + \mathrm{X})$$

$$\mathrm{H} = \mathrm{LayerNorm}(\mathrm{FFN}(\mathrm{H}') + \mathrm{H}')$$

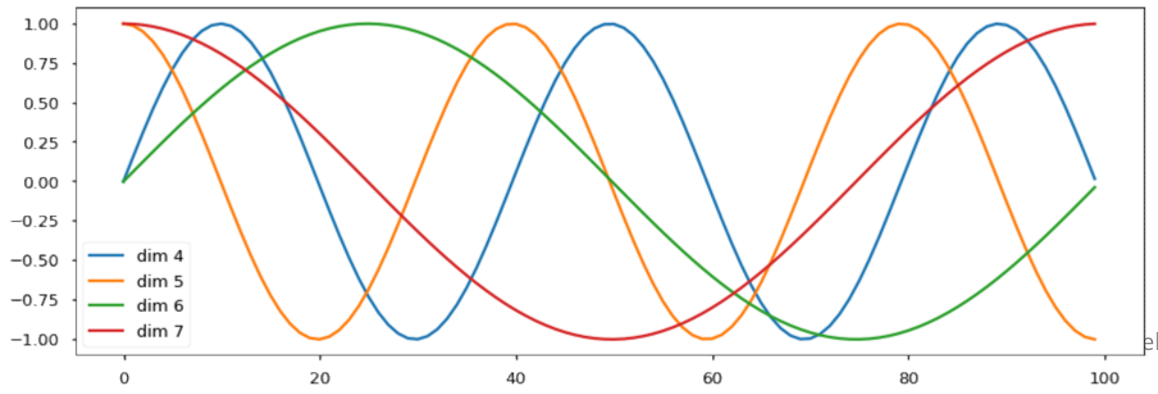
where SelfAtention(·) denotes self attention module and LayerNorm(·) denotes the layer normalization operation

4. Position Encodings.

Since Transformer doesn't introduce recurrence or convolution, it is ignorant of positional information (especially for the encoder). Thus additional positional representation is needed to model the ordering of tokens

Positional Encoding

- Since our model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, we must inject some information about the relative or absolute position of the tokens in the sequence. To this end, we add "positional encodings" to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension dmodel
- as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed (cite).
- In this work, we use sine and cosine functions of different frequencies: PE(pos,2i)=sin(pos/100002i/dmodel)
- PE(pos,2i+1)=cos(pos/100002i/dmodel)
- where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to 10000·2π. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PEpos+k can be represented as a linear function of PEpos.
- In addition, we apply dropout to the sums of the embeddings and the positional encodings in both the encoder and decoder stacks. For the base model, we use a rate of Pdrop=0.1
- .



Position-wise Feed-Forward Networks

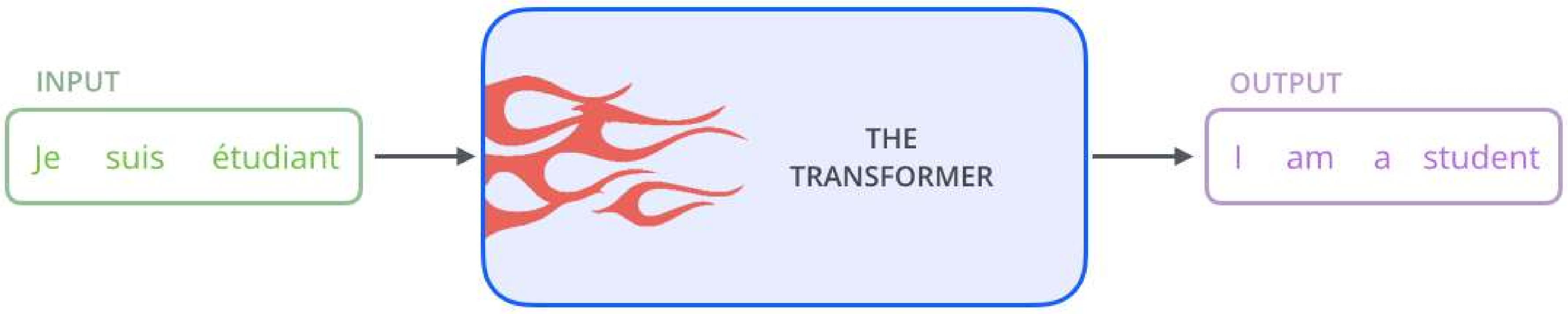
In addition to attention sub-layers, each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically.

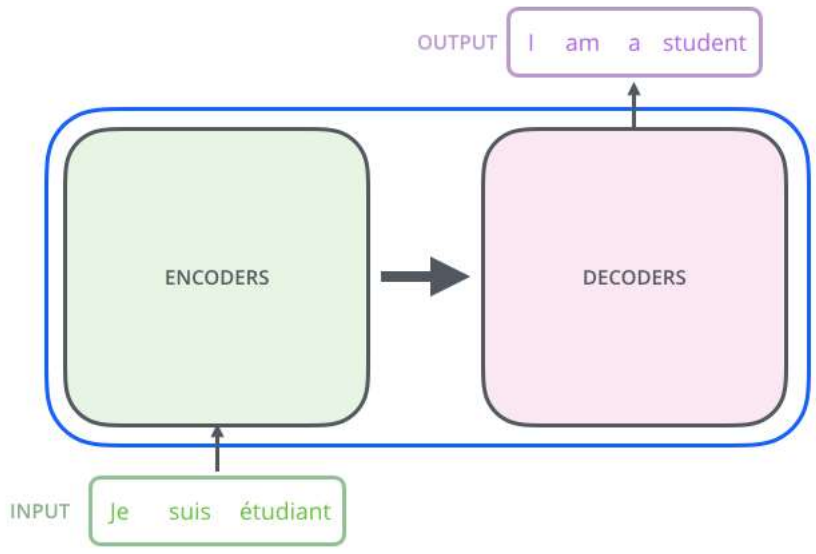
This consists of two linear transformations with a ReLU activation in between.

$$FFN(x)=\max(0,xW1+b1)W2+b2$$

While the linear transformations are the same across different positions, they use different parameters from layer to layer.

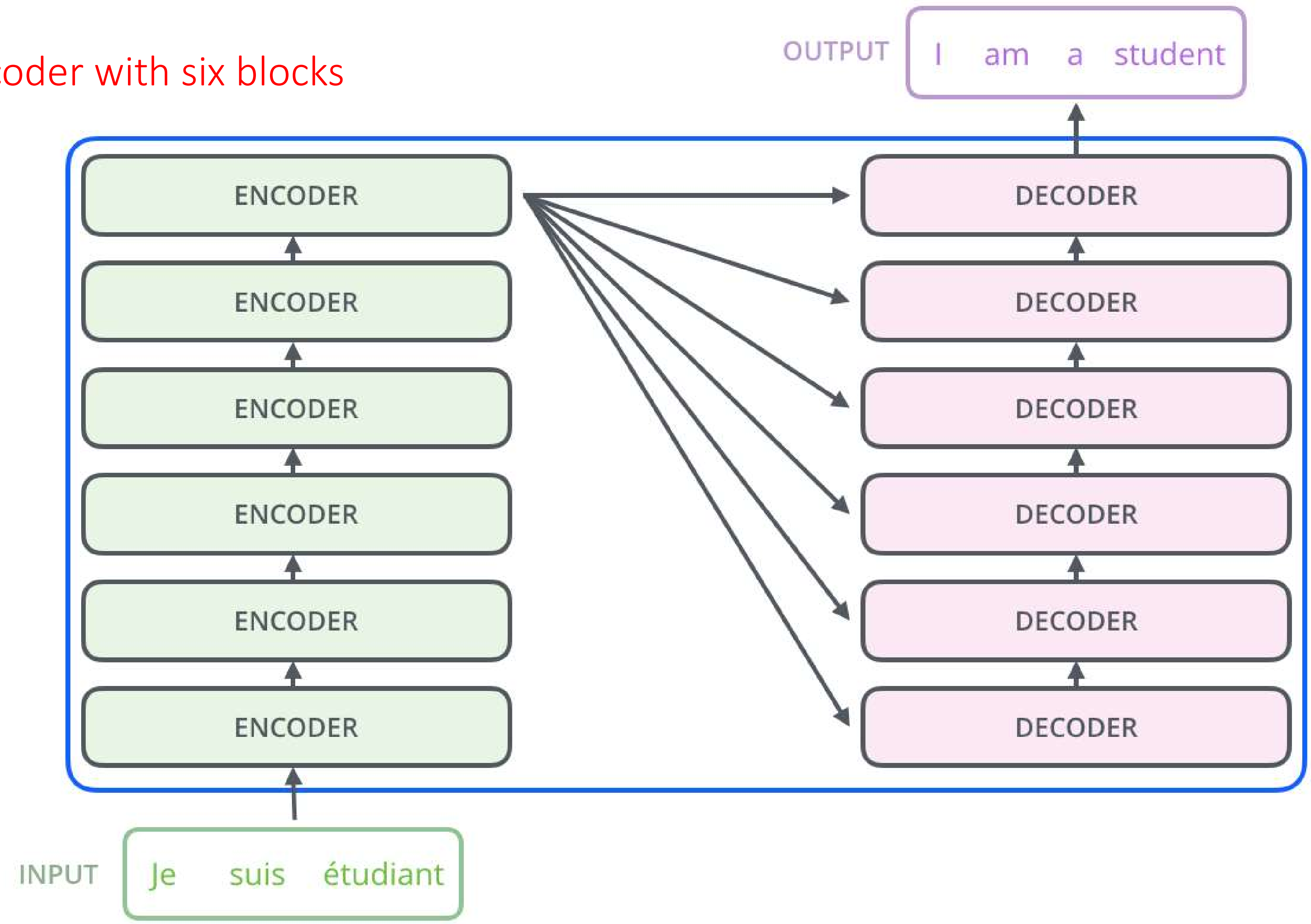
Another way of describing this is as two convolutions with kernel size 1.
The dimensionality of input and output is dmodel=512 , and the inner-layer has dimensionality dff=2048.



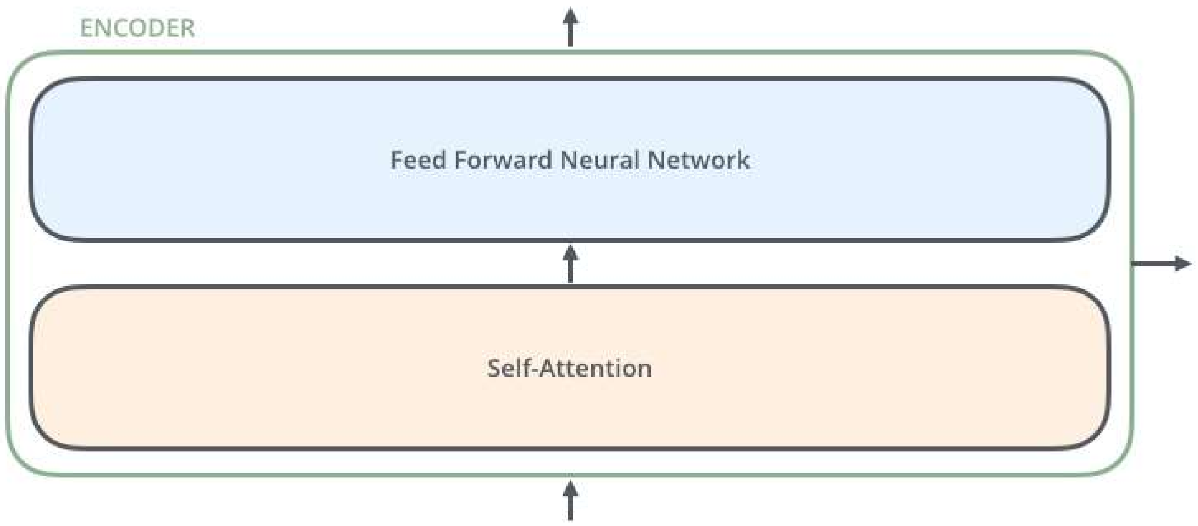


Jay Alammar
Visualizing machine learning one concept at a time.
@JayAlammar on Twitter. YouTube Channel

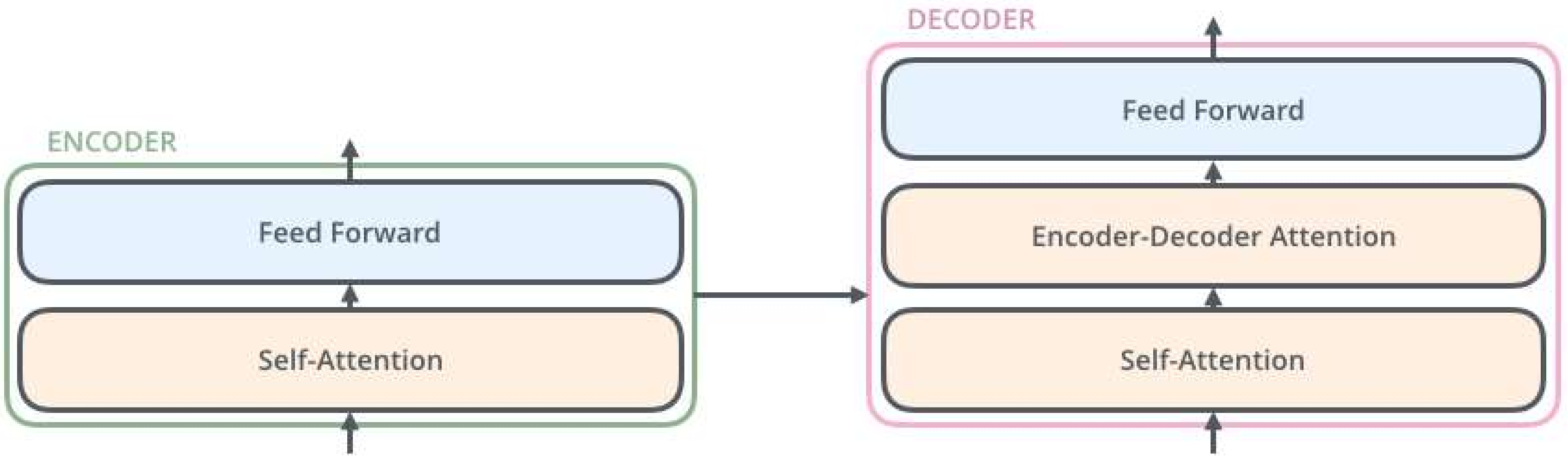
Encoder and Decoder with six blocks



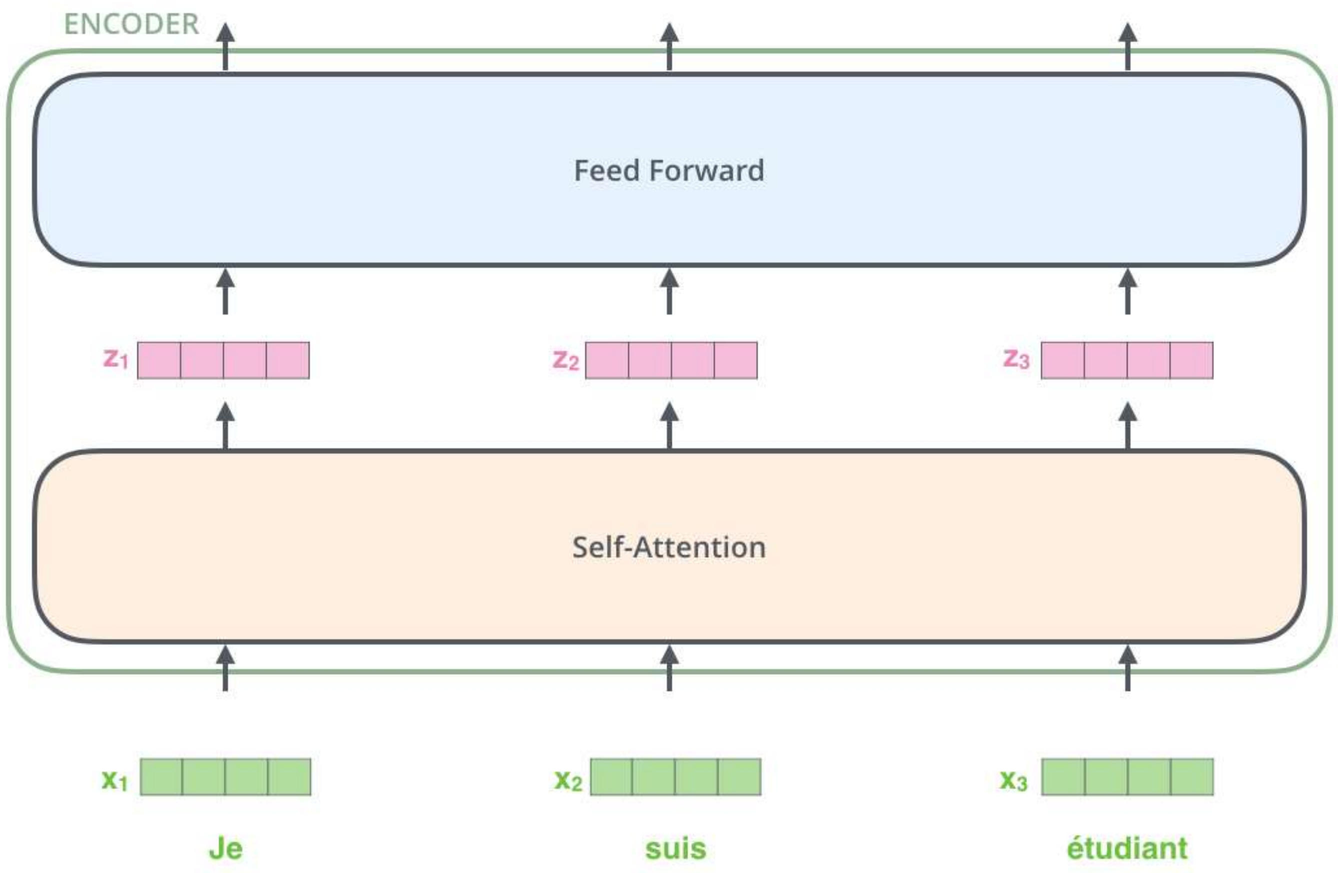
Encoder Block



Encoder and Decoder with one block

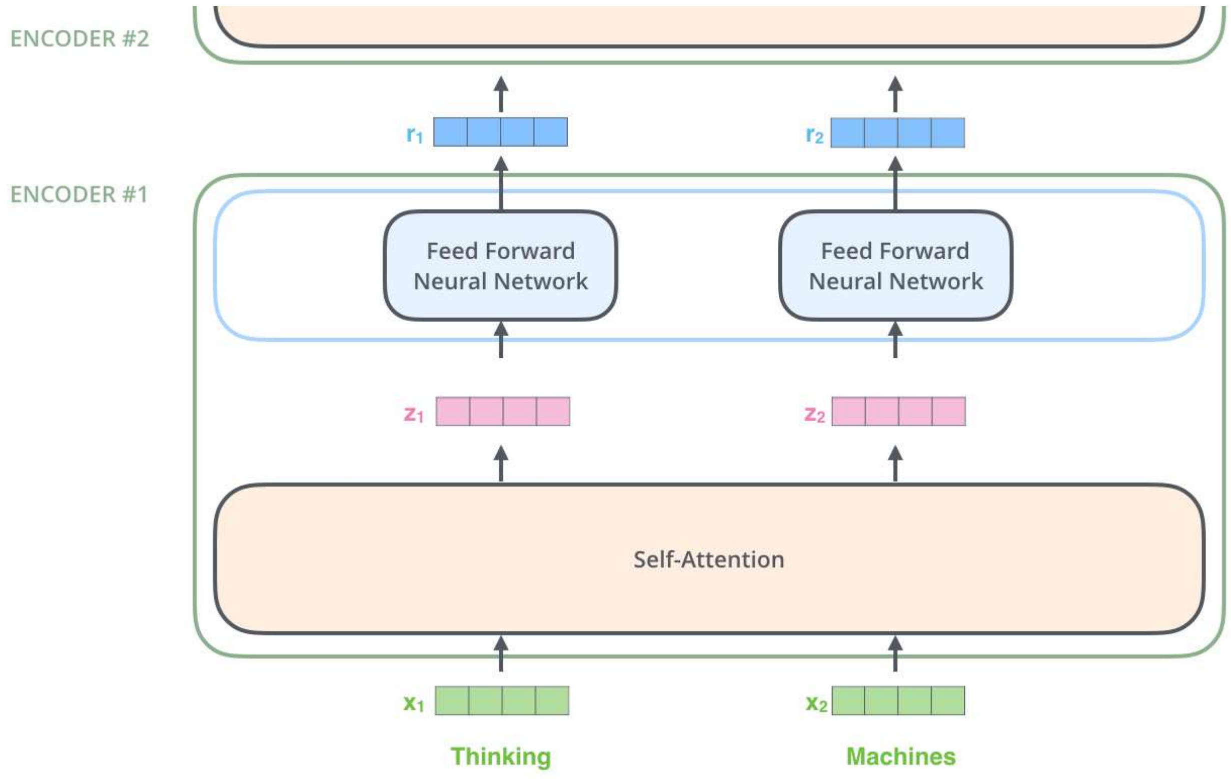


Bringing The Tensors Into The Picture



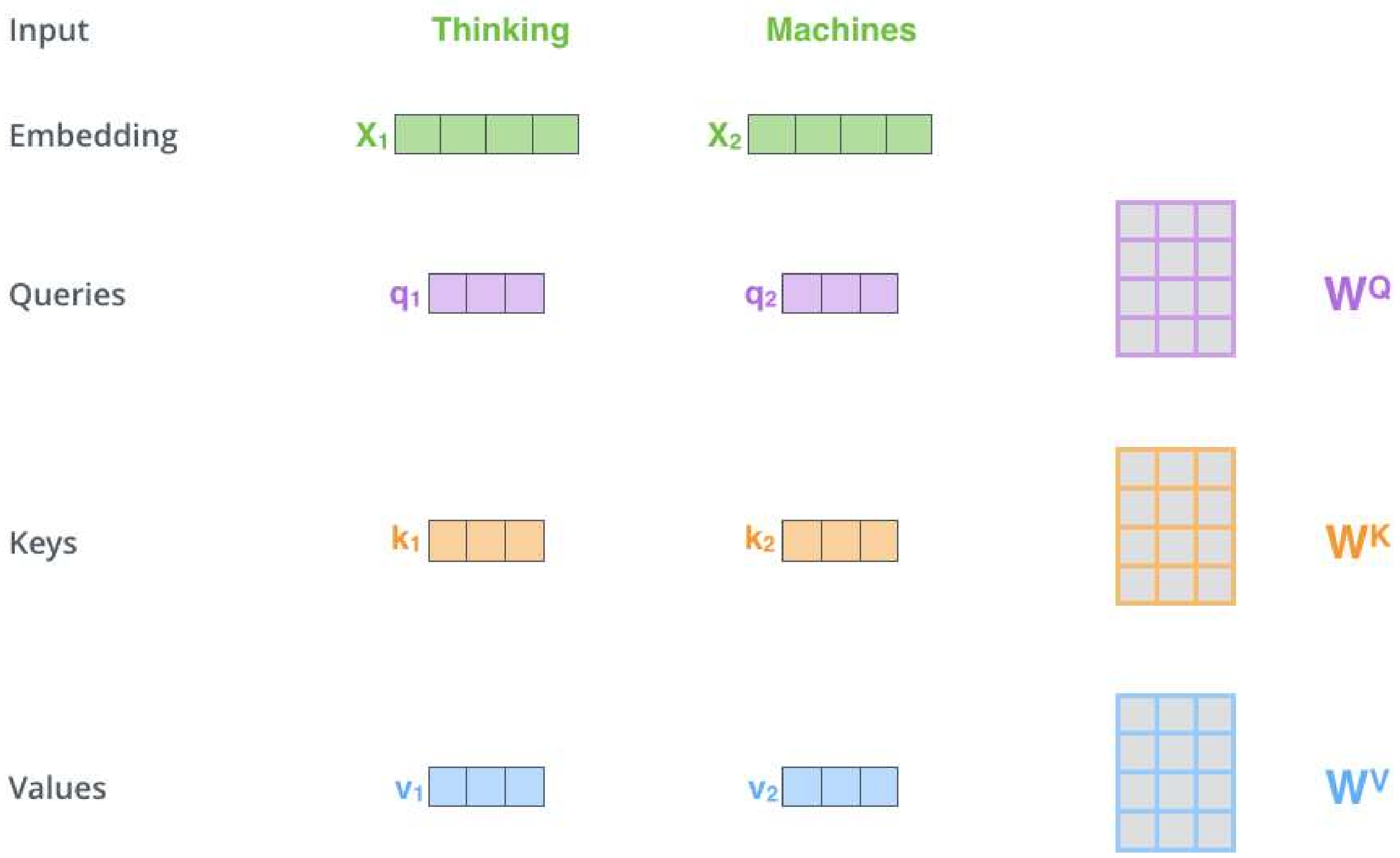
Now We’re Encoding!

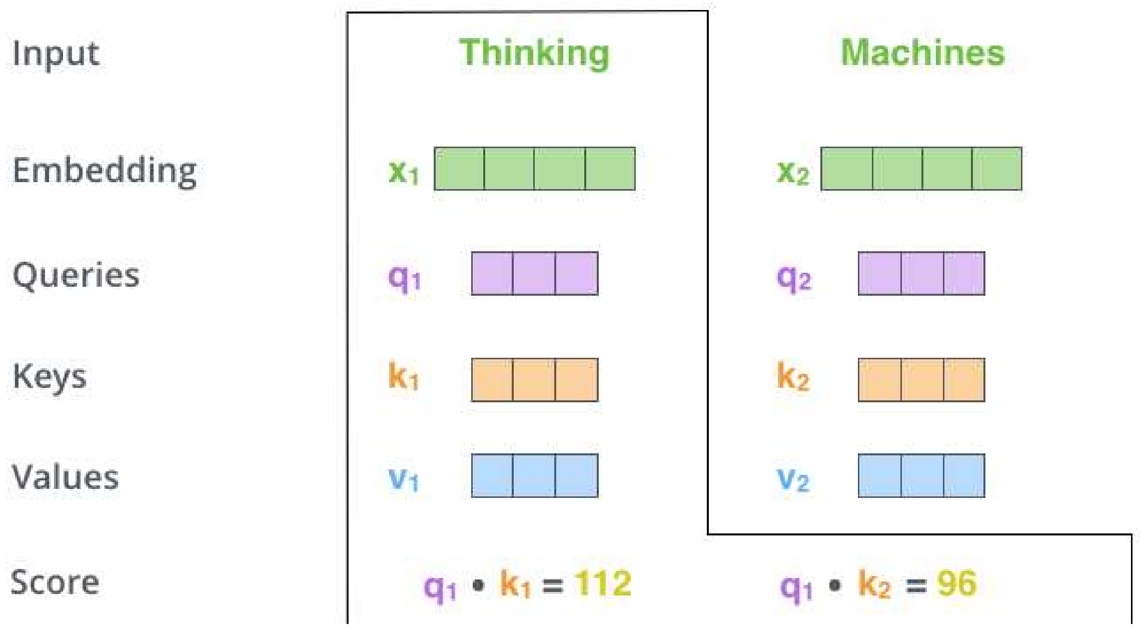
The word at each position passes through a self-attention process. Then, they each pass through a feed-forward neural network -- the exact same network with each vector flowing through it separately.

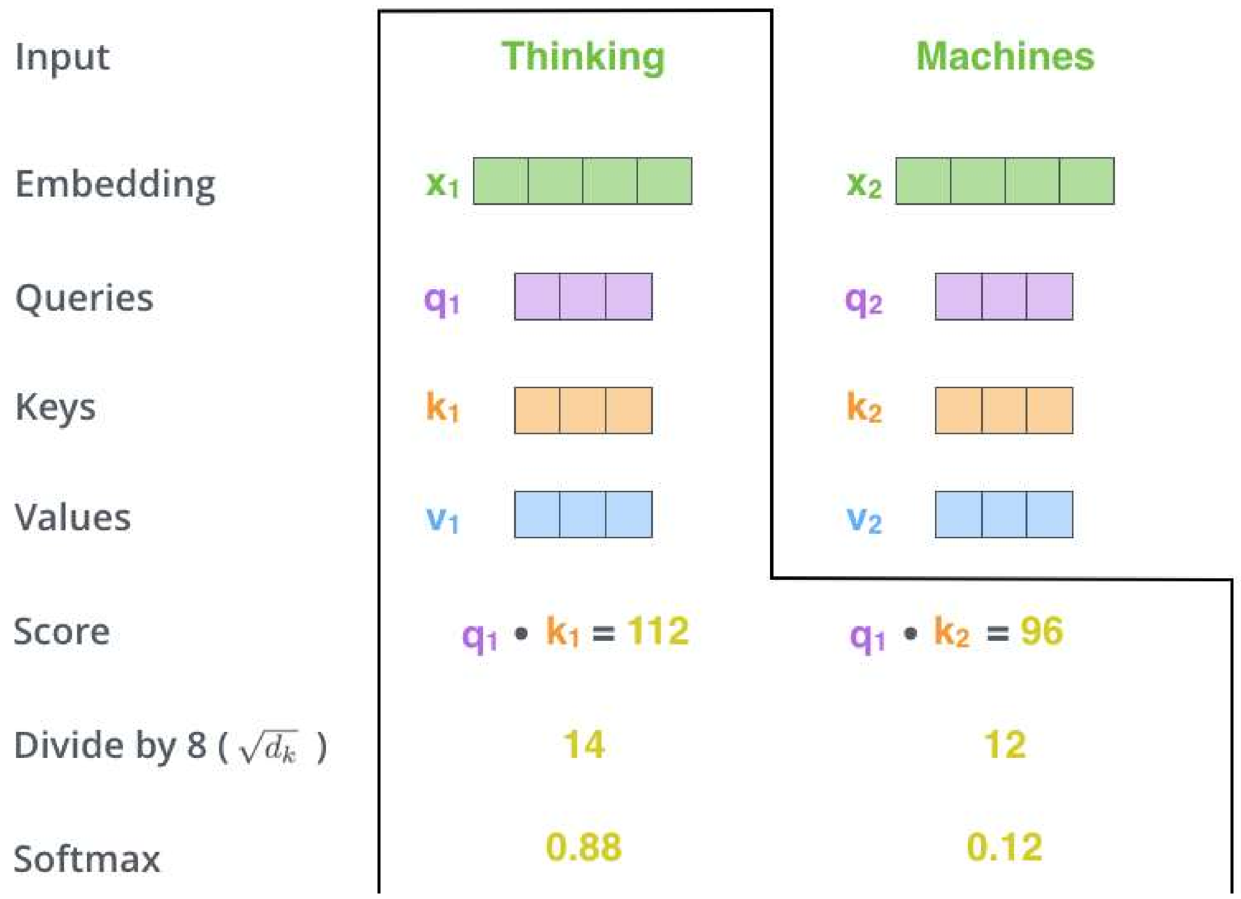


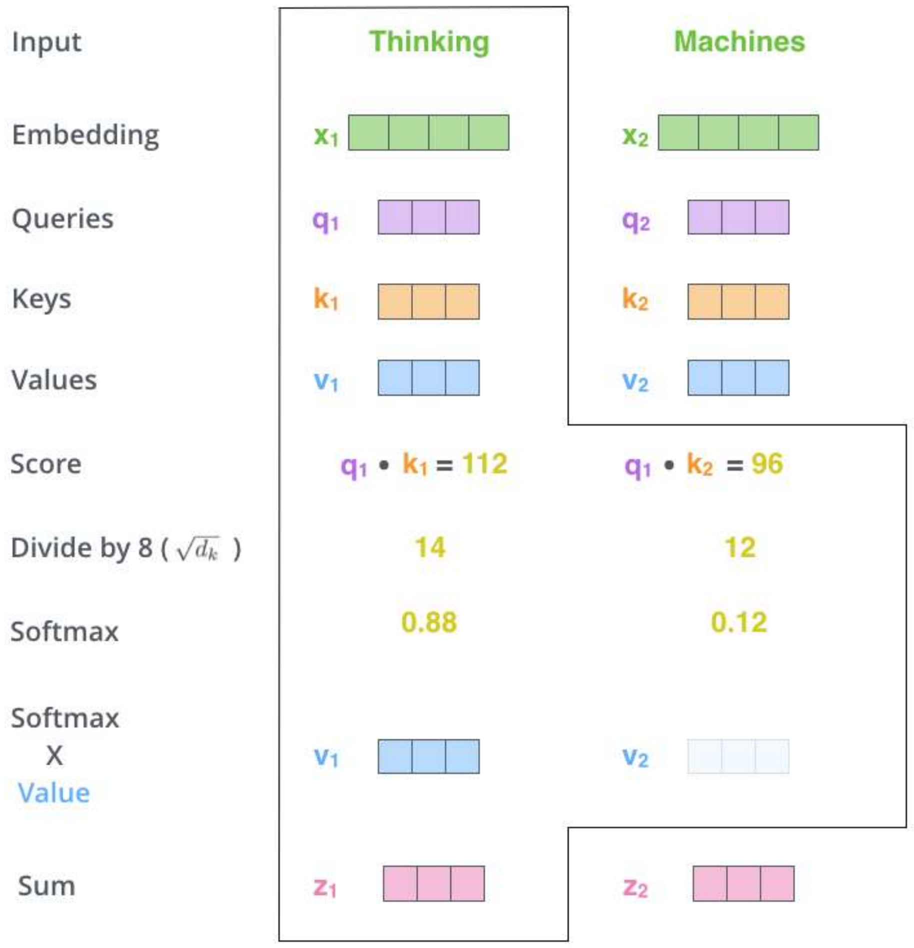
Self-Attention in Detail

Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.



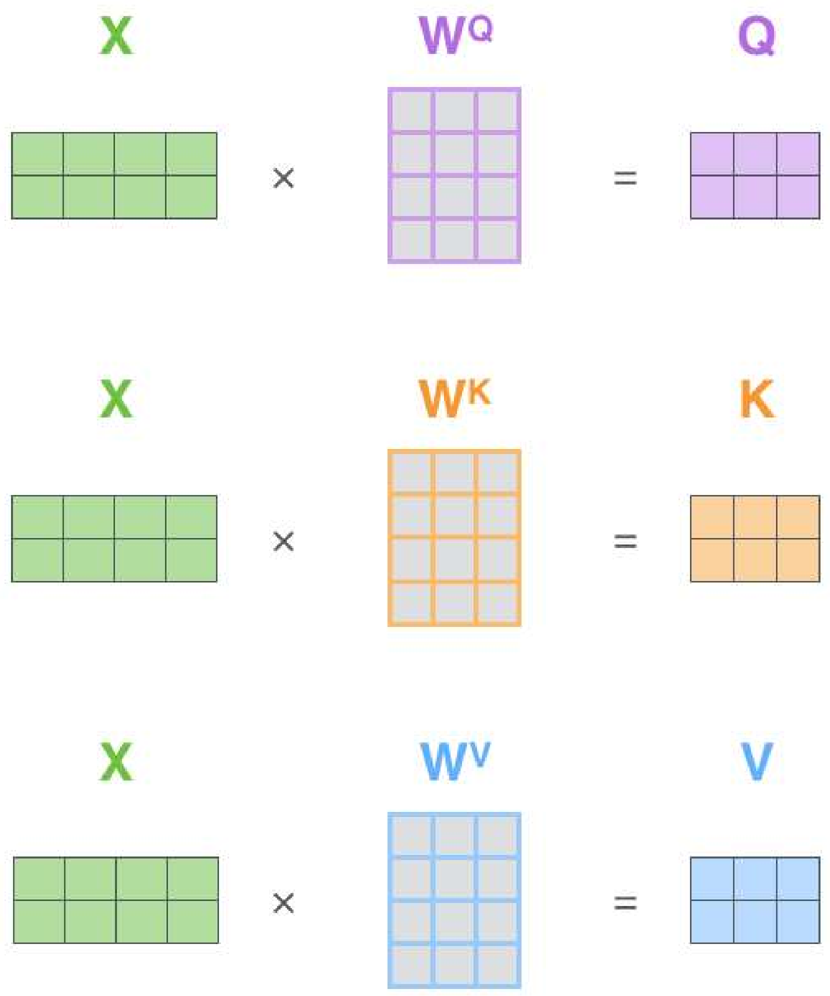

Input
Embedding
Queries
Keys
Values
Score
Thinking
Machines
x_1
x_2
q_1
q_2
k_1
k_2
v_1
v_2
$q_1 \bullet k_1 = 112$
$q_1 \bullet k_2 = 96$


Input
Thinking
Machines
Embedding
x_1
x_2
Queries
q_1
q_2
Keys
k_1
k_2
Values
v_1
v_2
Score
$q_1 \cdot k_1 = 112$
$q_1 \cdot k_2 = 96$
Divide by 8 ($\sqrt{d_k}$)
14
12
Softmax
0.88
0.12



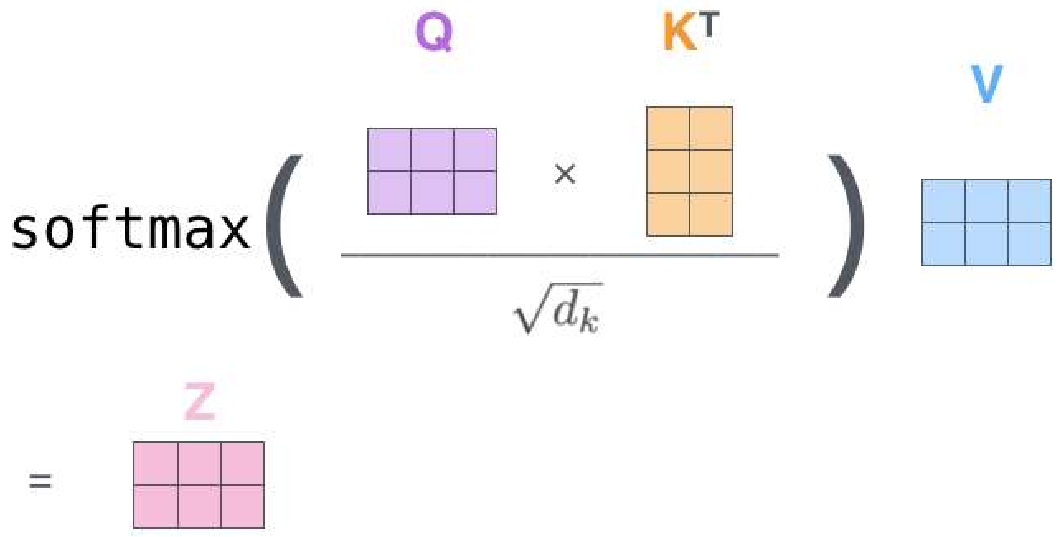
Input
Thinking
Machines
Embedding
x_1
x_2
Queries
q_1
q_2
Keys
k_1
k_2
Values
v_1
v_2
Score
$q_1 \bullet k_1 = 112$
$q_1 \bullet k_2 = 96$
Divide by 8 ($\sqrt{d_k}$)
14
12
Softmax
0.88
0.12
Softmax X Value
v_1
v_2
Sum
z_1
z_2

Matrix Calculation of Self-Attention

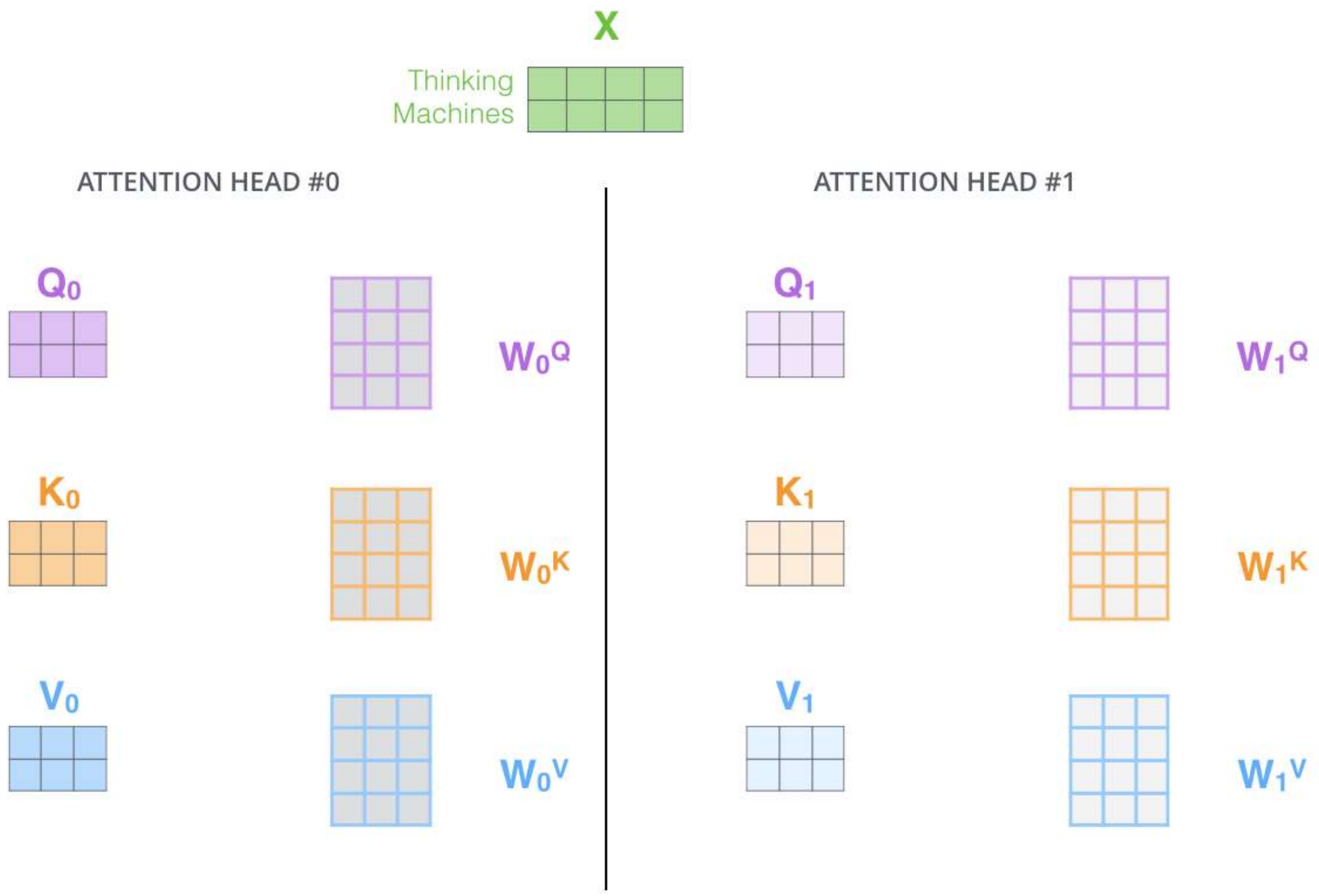


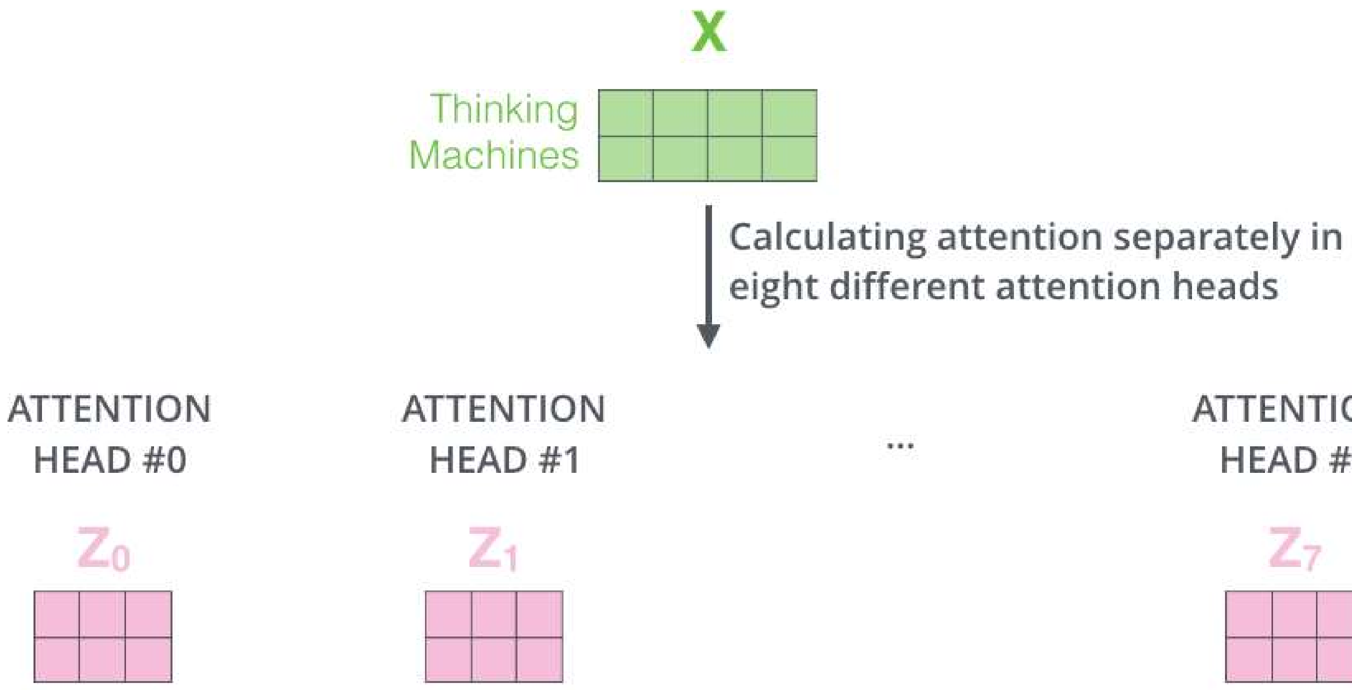
Every row in the X matrix corresponds to a word in the input sentence. We again see the difference in size of the embedding vector (512, or 4 boxes in the figure), and the q/k/v vectors (64, or 3 boxes in the figure)

The self-attention calculation in matrix form

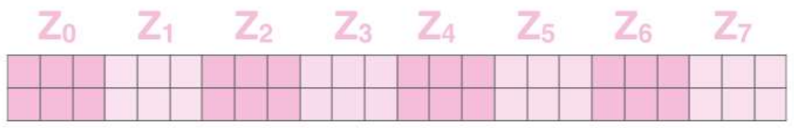


The Beast With Many Heads



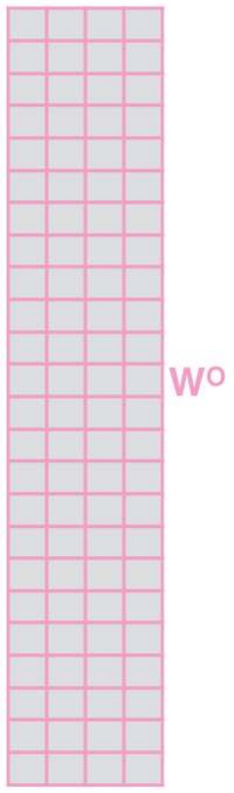

X
Thinking
Machines
Calculating attention separately in eight different attention heads
ATTENTION HEAD #0
ATTENTION HEAD #1
...
ATTENTION HEAD #7
Z0
Z1
Z7

1) Concatenate all the attention heads



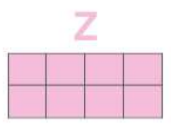
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



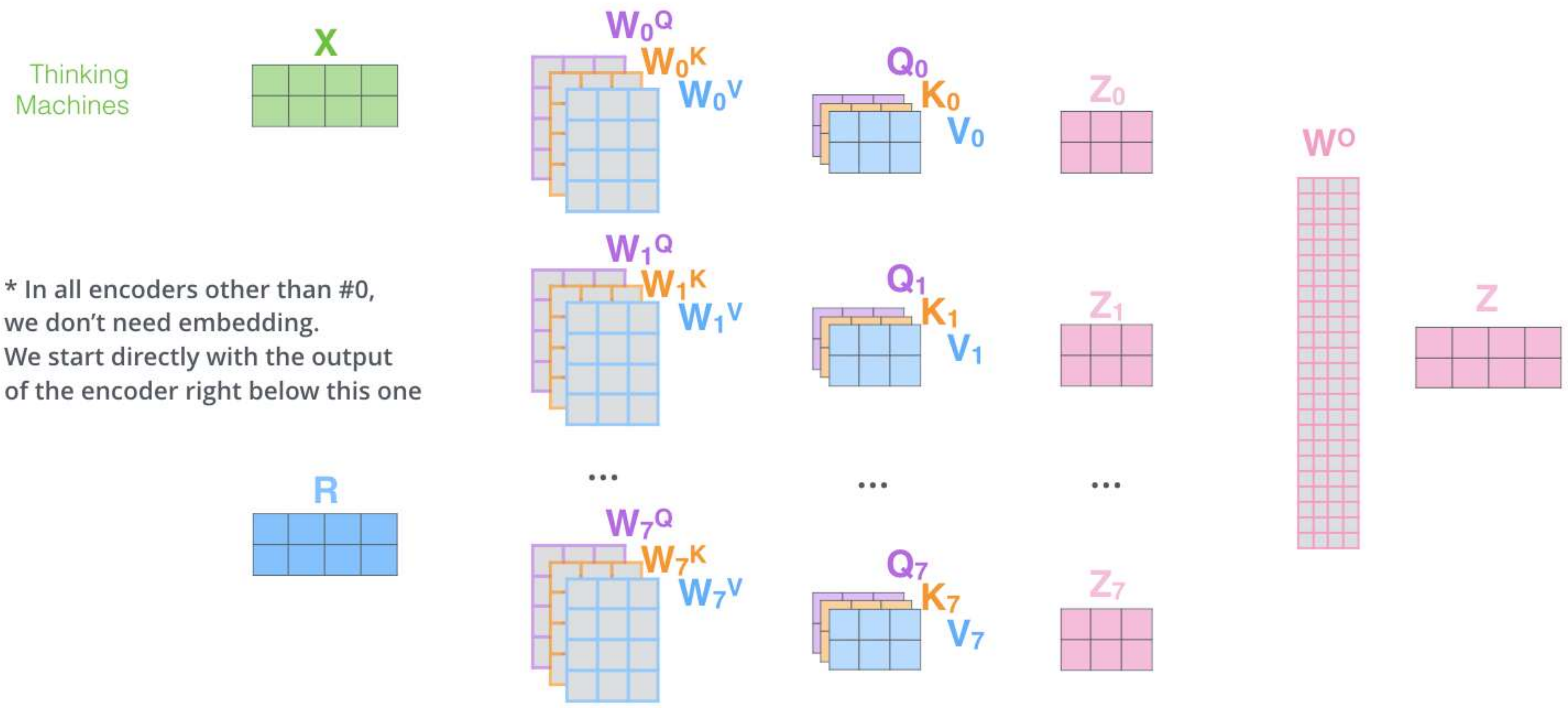
1) This is our input sentence*

2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

4) Calculate attention using the resulting Q/K/V matrices

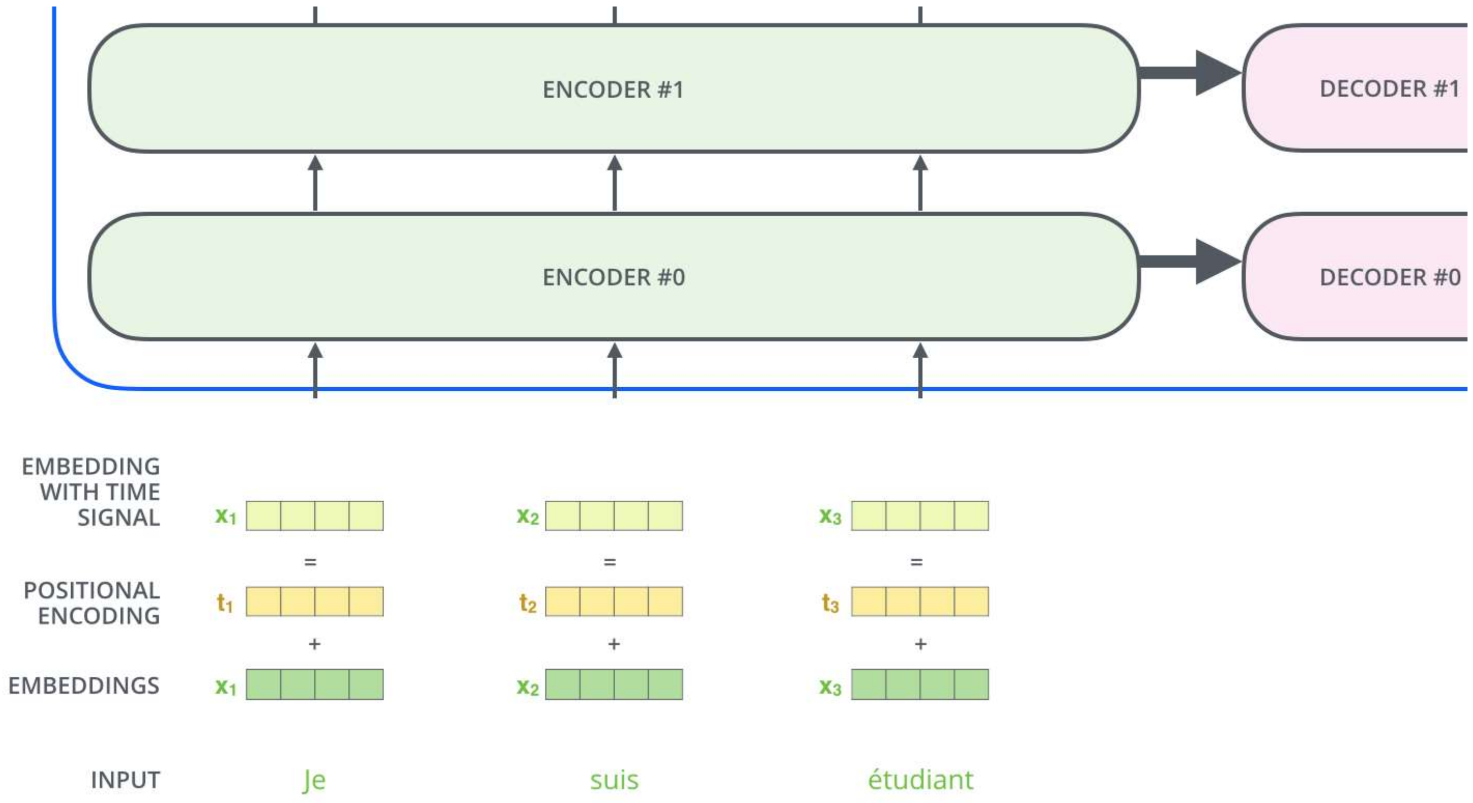
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

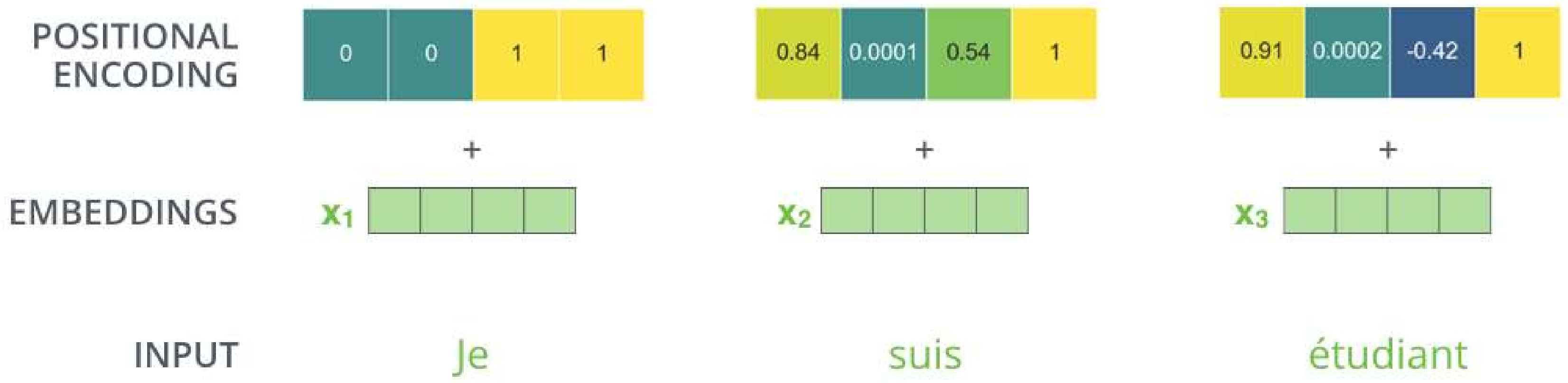


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Representing The Order of The Sequence Using Positional Encoding

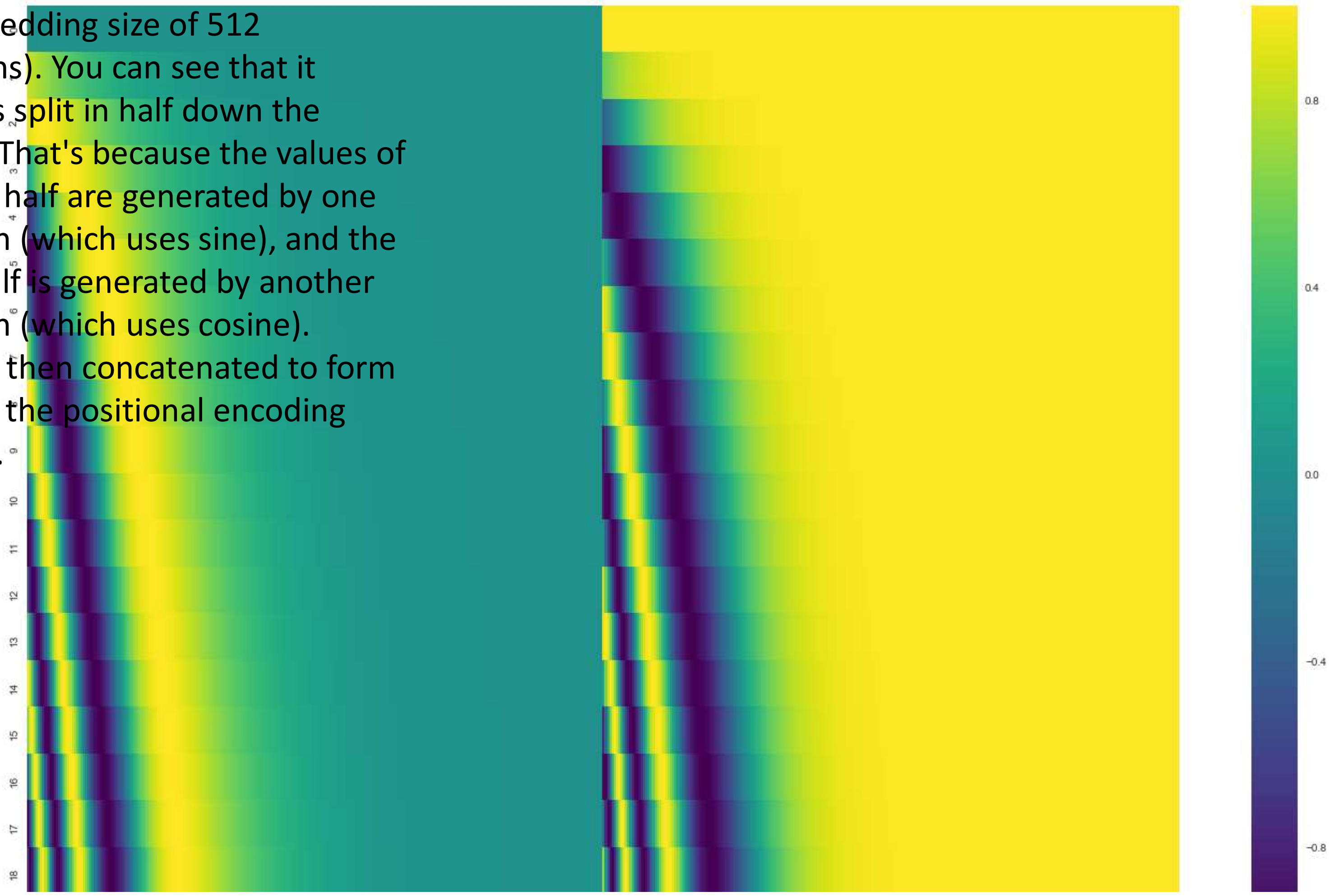
To give the model a sense of the order of the words, we add positional encoding vectors -- the values of which follow a specific pattern.



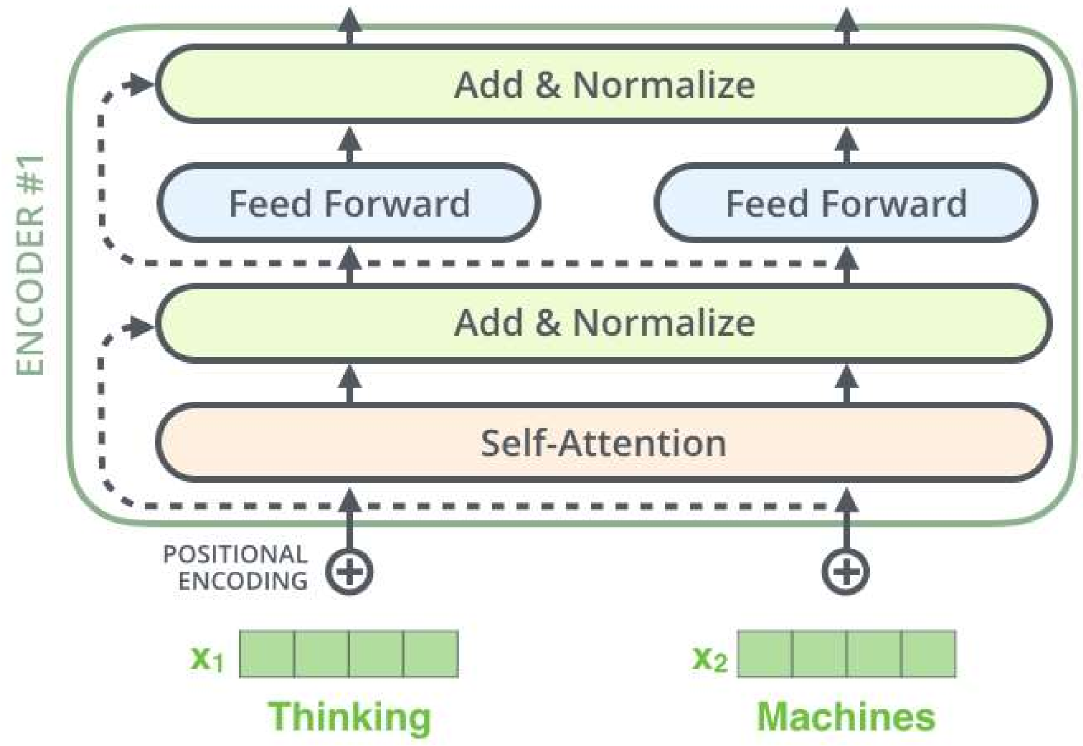


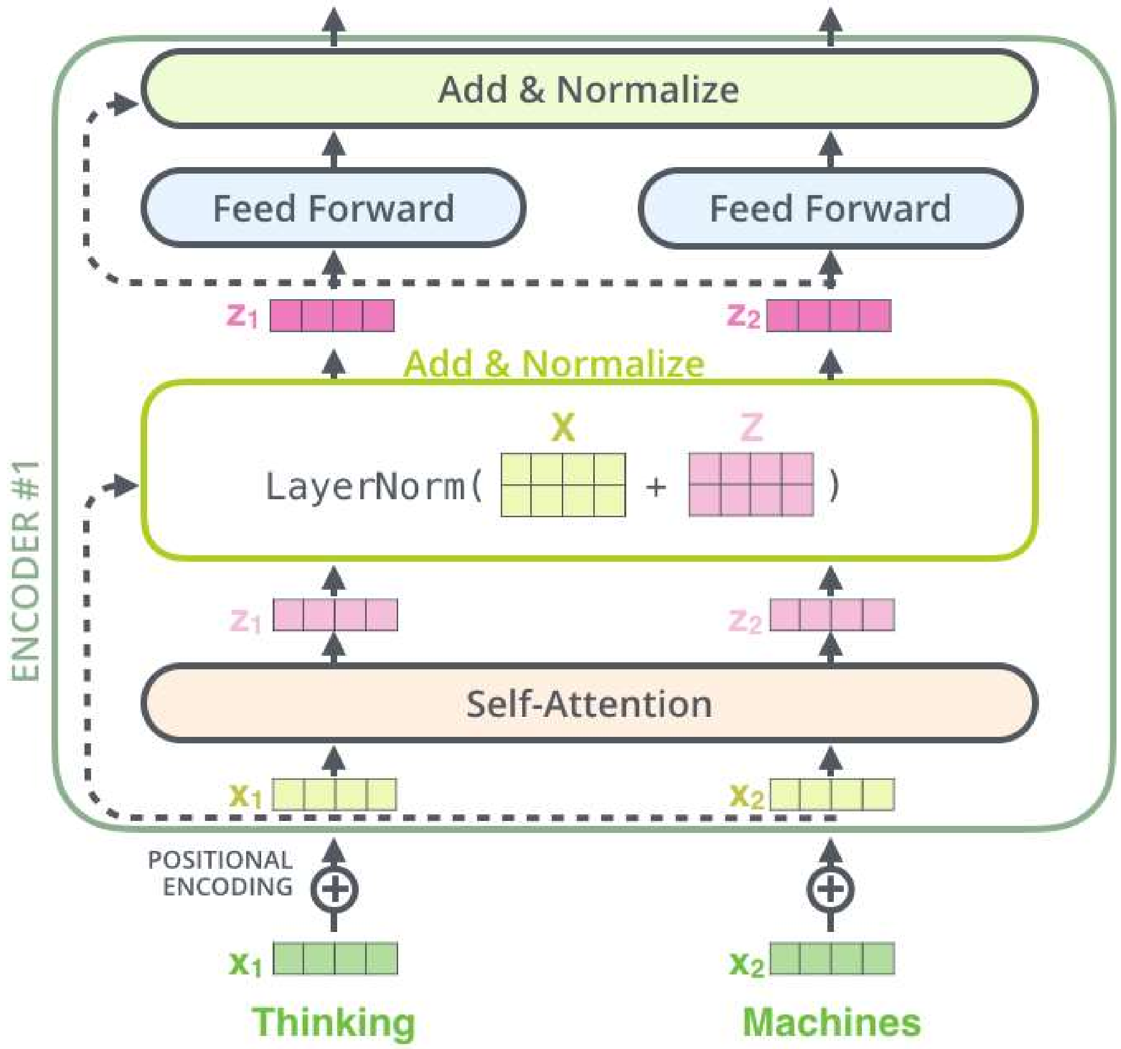
A real example of positional encoding with a toy embedding size of 4

A real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns). You can see that it appears split in half down the center. That's because the values of the left half are generated by one function (which uses sine), and the right half is generated by another function (which uses cosine). They're then concatenated to form each of the positional encoding vectors.

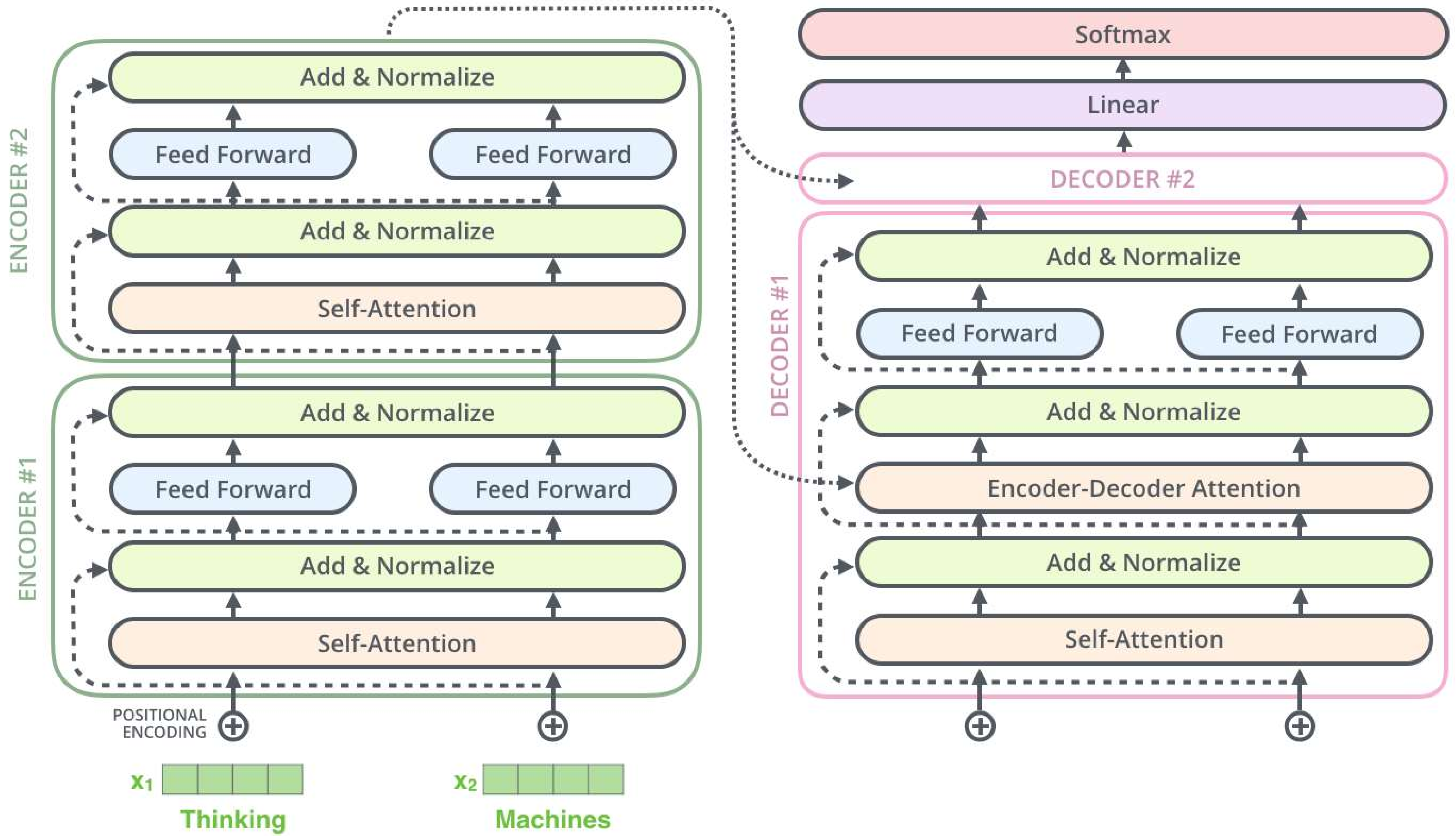


The Residuals

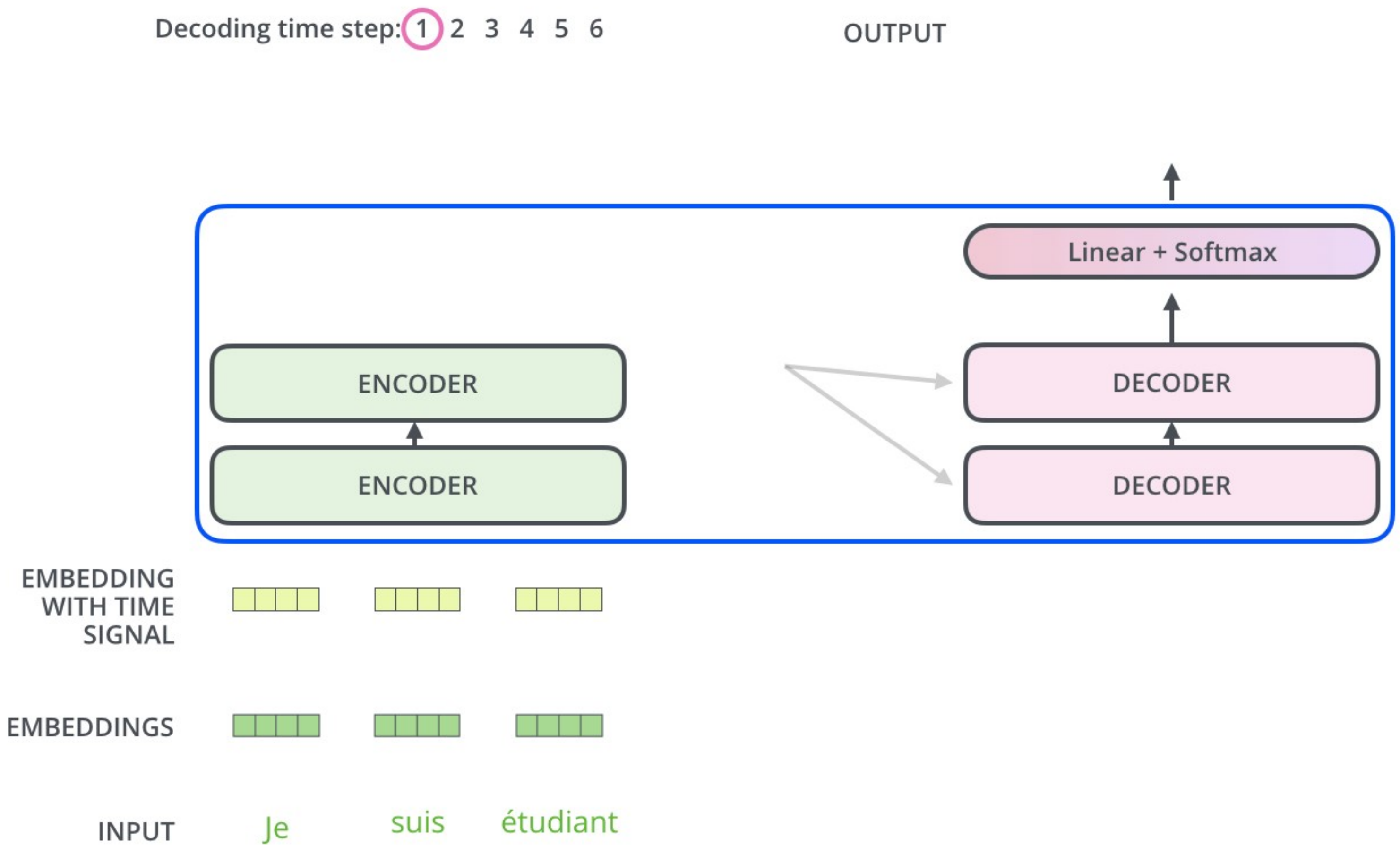


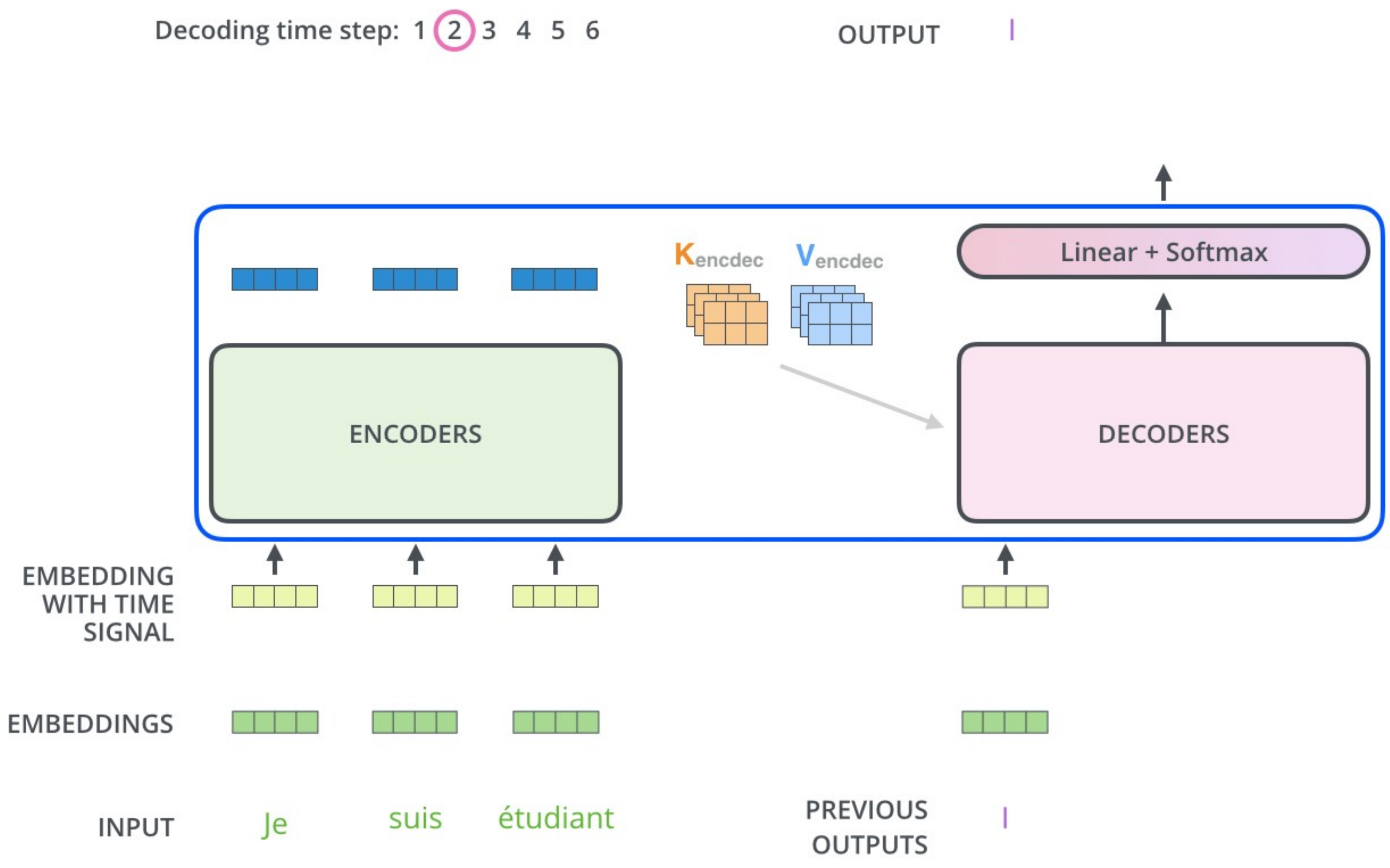


Add & Normalize
Feed Forward
Feed Forward
z1
z2
Add & Normalize
X
Z
LayerNorm(+)
z1
z2
Self-Attention
x1
x2
ENCODER #1
POSITIONAL ENCODING
x1
x2
Thinking
Machines

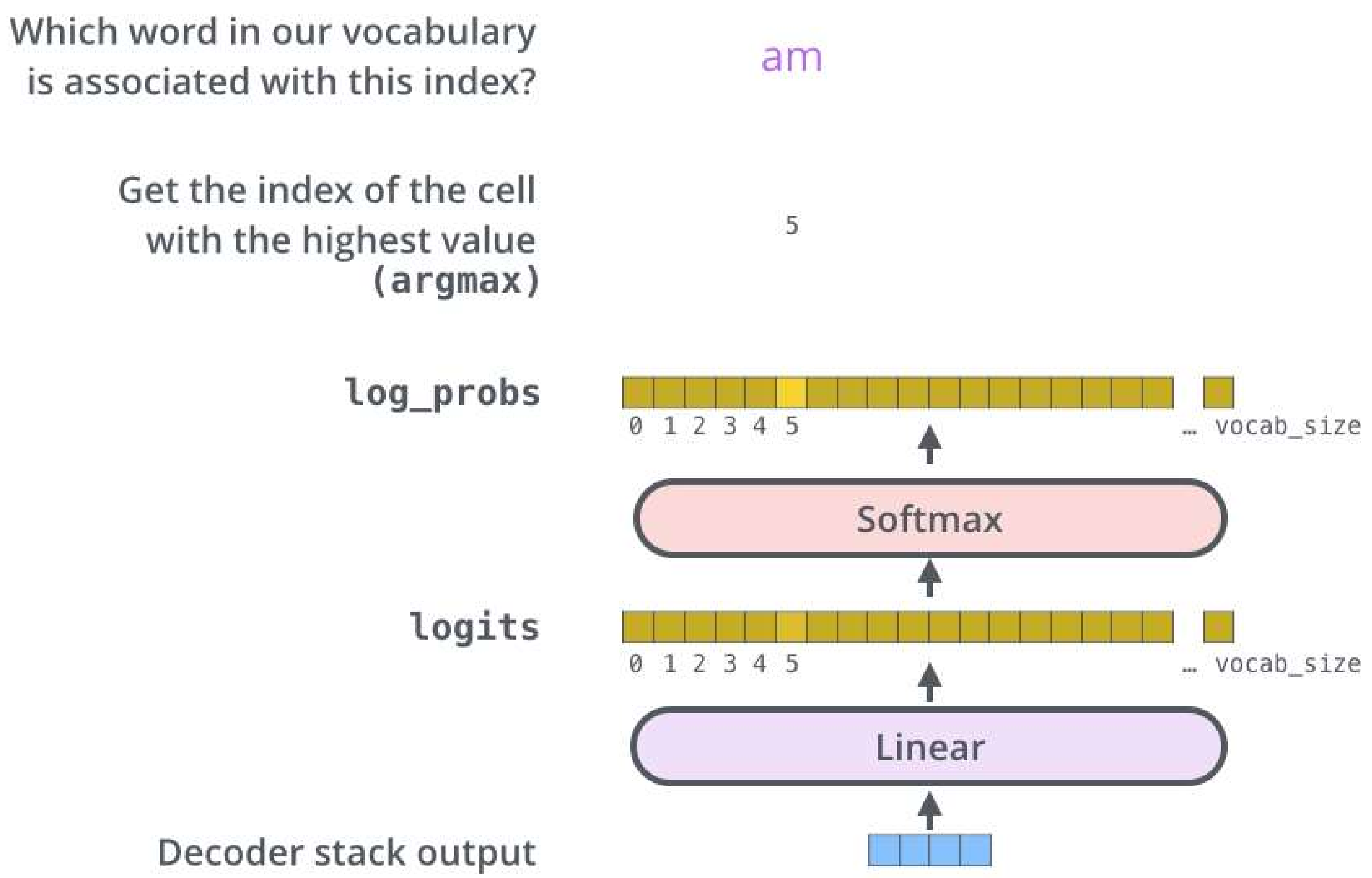

ENCODER #2
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
ENCODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
POSITIONAL ENCODING
x1
Thinking
x2
Machines
Softmax
Linear
DECODER #2
DECODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Encoder-Decoder Attention
Add & Normalize
Self-Attention

After finishing the encoding phase, we begin the decoding phase. Each step in the decoding phase outputs an element from the output sequence (the English translation sentence in this case).




Decoding time step: 1 2 3 4 5 6
OUTPUT
I
Kencdec
Vencdec
Linear + Softmax
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

The Final Linear and Softmax Layer


Which word in our vocabulary is associated with this index?
am
Get the index of the cell with the highest value (argmax)
5
log_probs
0 1 2 3 4 5
… vocab_size
Softmax
logits
0 1 2 3 4 5
… vocab_size
Linear
Decoder stack output

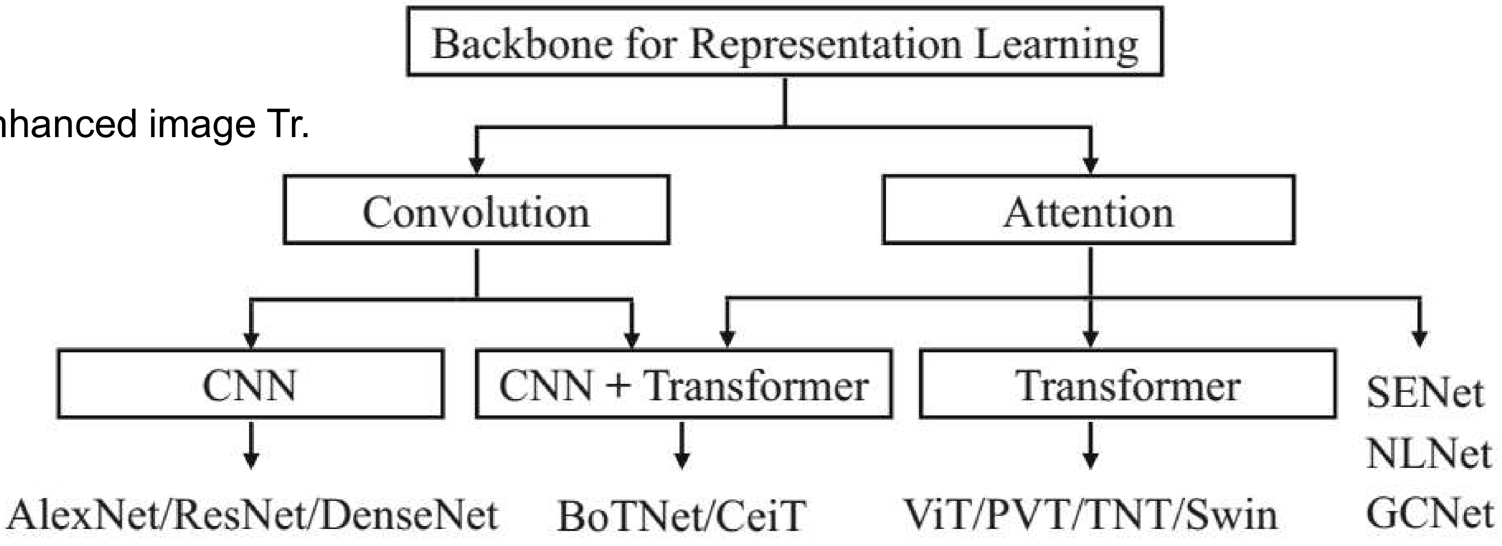
Vision Transformer (ViT)

The applications of transformer based models in computer vision, including image classification, high/mid-level vision, low-level vision and video processing.

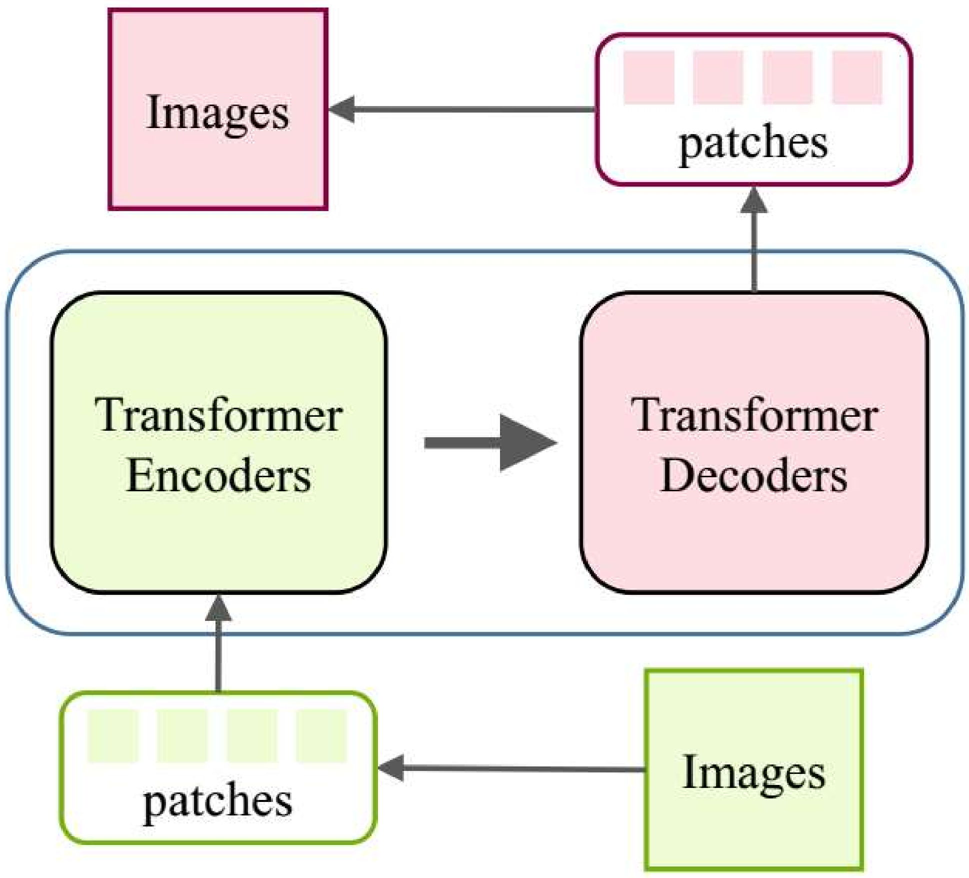
Convolution and Attention in Computer Vision Problems:

[Bottleneck Tr.]

Convolution-enhanced image Tr.



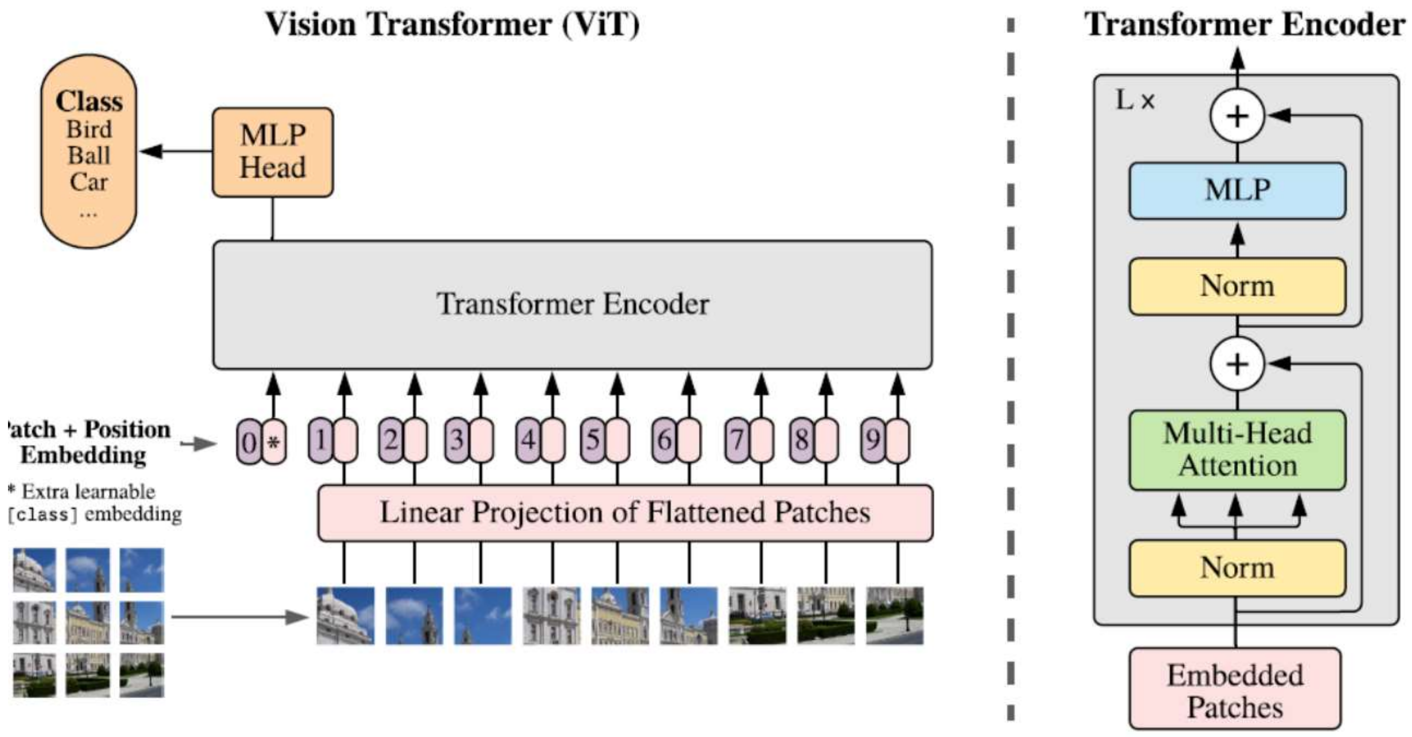
A generic framework for using transformer in image processing



Vision Transformer (ViT)

Dosovitskiy...2021

Vision Transformer (ViT) is a pure transformer directly applies to the sequences of image patches for image classification task.
It follows transformer's original design as much as possible.



The framework of ViT

How the Vision Transformer works

* Nikolas Adaloglou

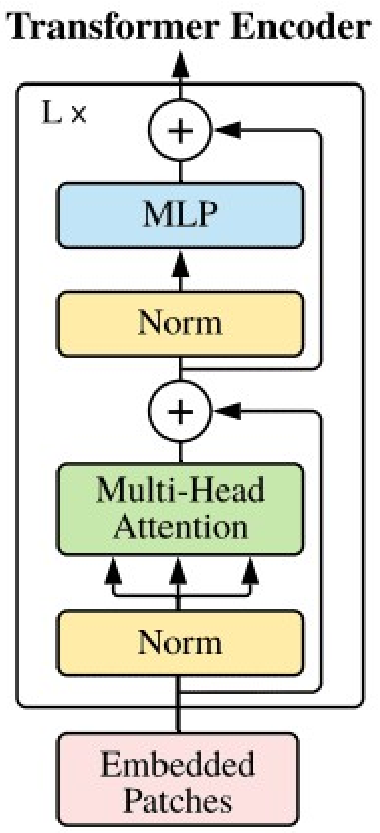
The total architecture is called Vision Transformer (ViT in short). Let’s examine it step by step.

1. Split an image into patches
2. Flatten the patches
3. Produce lower-dimensional linear embeddings from the flattened patches
4. Add positional embedding **(they model positional embeddings with trainable linear layers)**
5. Feed the sequence as an input to a standard transformer encoder
6. Pretrain the model with image labels (fully supervised on a huge dataset)
7. Fine tune on the downstream dataset for image classification

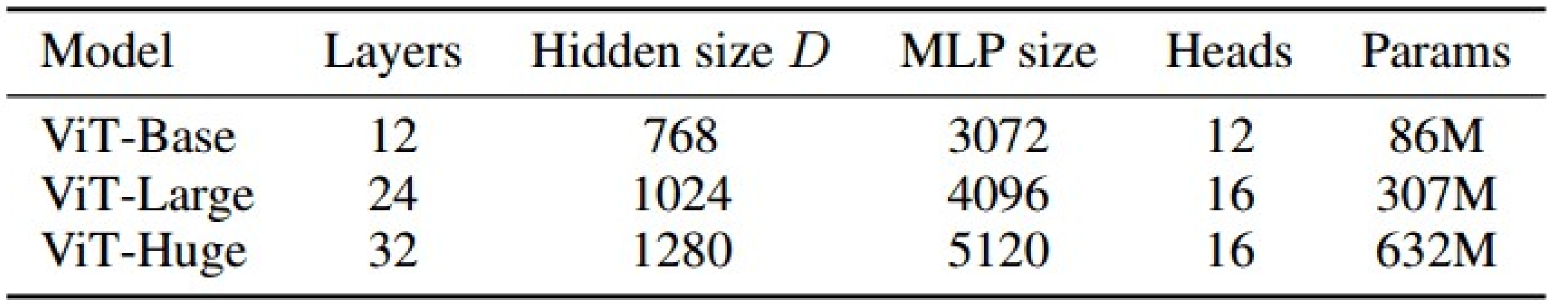
Image patches are basically the sequence tokens (like words).

In fact, the encoder block is identical to the original transformer proposed by Vaswani et al. (2017)

The only thing that changes is the number of those blocks.



To this end, and to further prove that with more data they can train larger ViT variants, 3 models were proposed:



Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

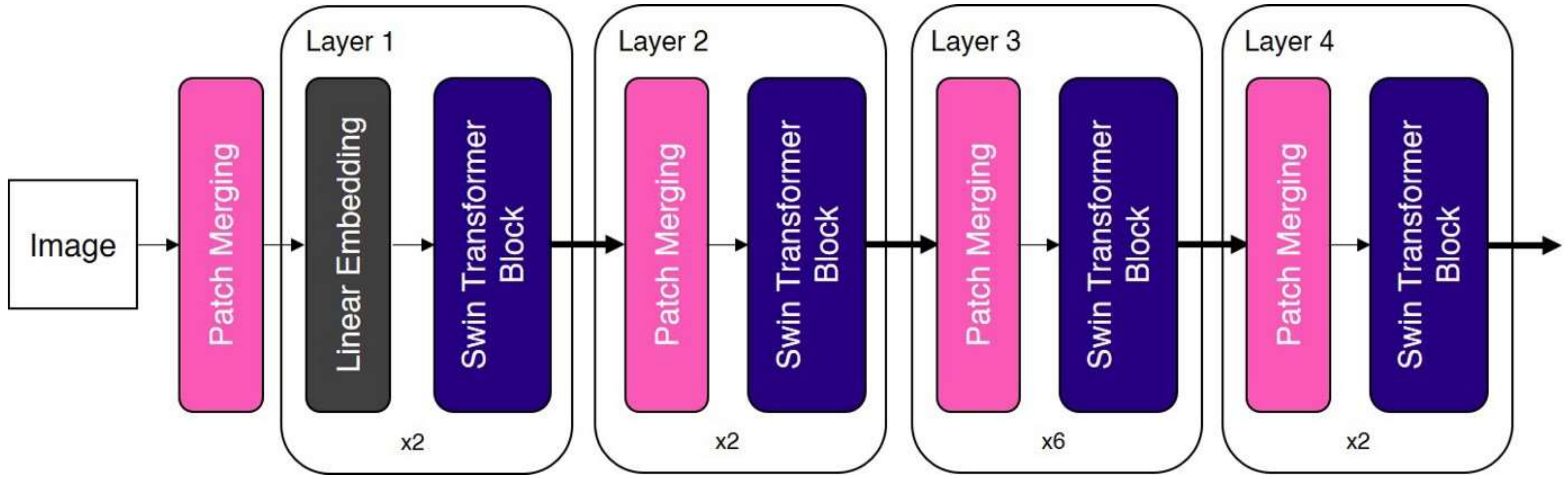
Heads refer to multi-head attention, while the MLP size refers to the blue module in the figure.
MLP stands for multi-layer perceptron but it's actually a bunch of linear transformation layers.
Hidden size D is the embedding size, which is kept fixed throughout the layers. Why keep it fixed? So that we can use short residual skip connections.
But is this enough?
Yes and no. Actually, we need a massive amount of data and as a result computational resources.

Swin Transformer

- Swin Transformer (Liu et al., 2021) is **a transformer-based deep learning model with state-of-the-art performance in vision tasks**. Unlike the Vision Transformer (ViT) (Dosovitskiy et al., 2020) which precedes it, Swin Transformer is highly efficient and has greater accuracy.
- In 2020, the Vision Transformer (ViT) garnered much attention from the AI community, notable for its pure transformer architecture with promising results in vision tasks. Despite its promise,
- ViTs suffer from several shortcomings:
  1. Most notably, ViTs struggle with high resolution images as its computational complexity is *quadratic* to the image size.
  2. Furthermore, the fixed scale tokens in ViTs are unsuitable in vision tasks where the visual elements are of variable scale.
- A flurry of research work followed ViT, and most of them made enhancements to the standard transformer architecture in order to address the above-mentioned shortcomings.
- In 2021, Microsoft researchers published the Swin Transformer (Liu et al., 2021), arguably one of the most exciting piece of research following up from the original ViT

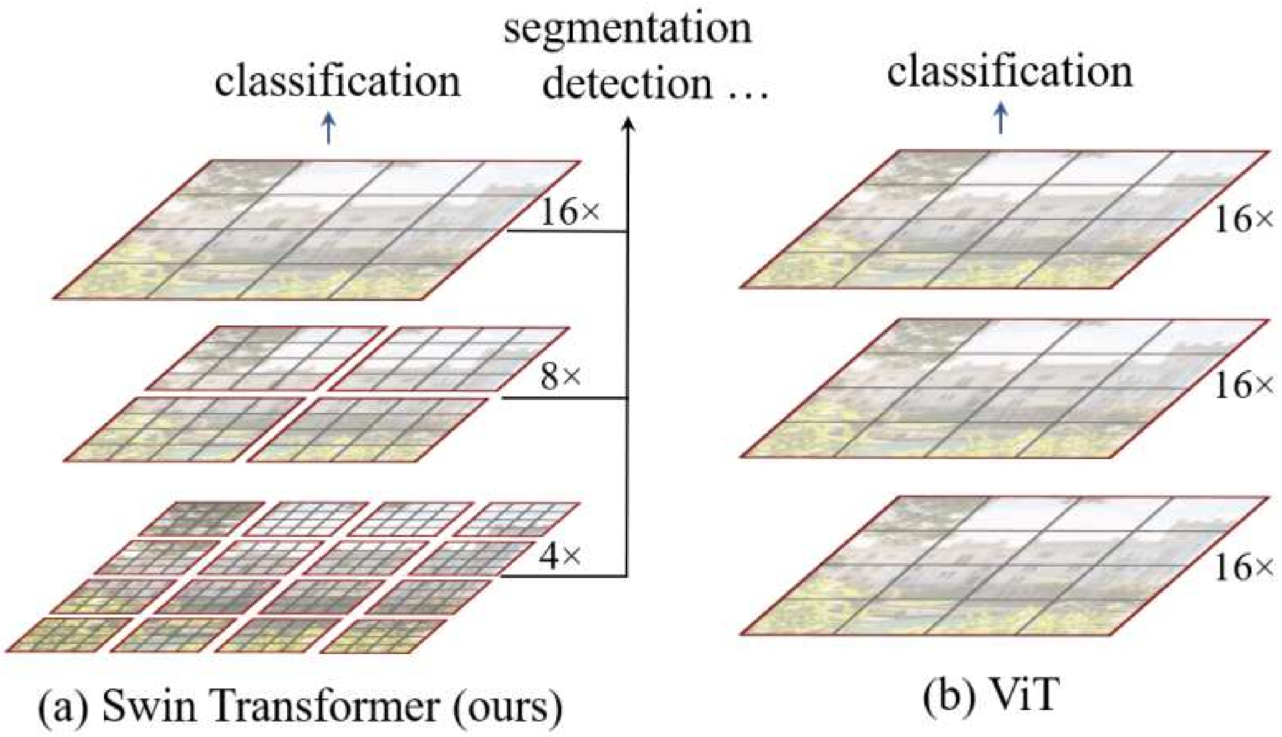
Swin Transformers

The Swin Transformer introduced two key concepts to address the issues faced by the original ViT — **hierarchical feature maps** and **shifted window attention.** In fact, the name of Swin Transformer comes from "**S**hifted **win**dow **Transformer**". The overall architecture of the Swin Transformer is shown below.



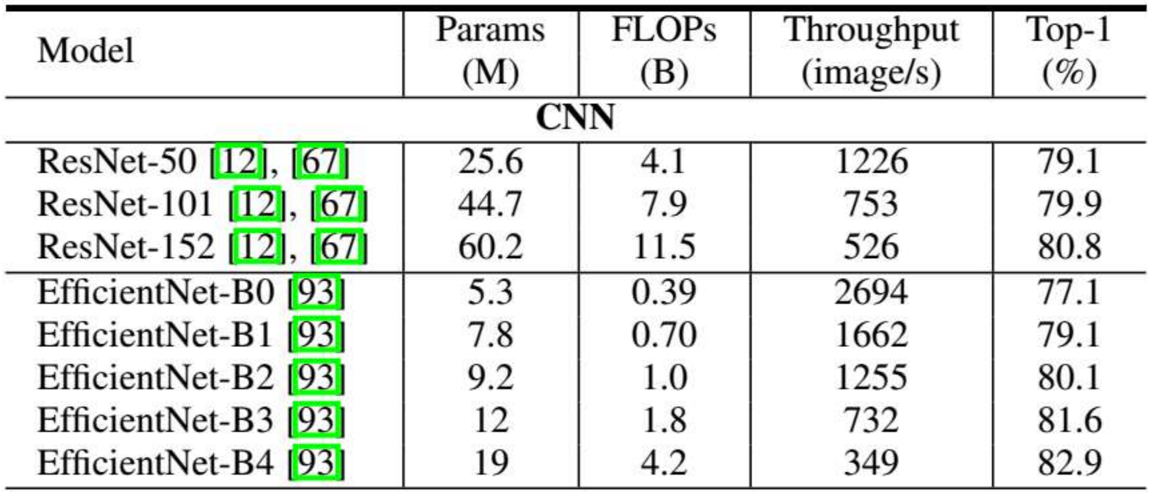
As we can see, the 'Patch Merging' block and the 'Swin Transformer Block' are the two key building blocks in Swin Transformer.

The shifted windowing scheme brings greater efficiency by limiting self-attention computation to non-overlapping local windows while also allowing for cross-window connection.

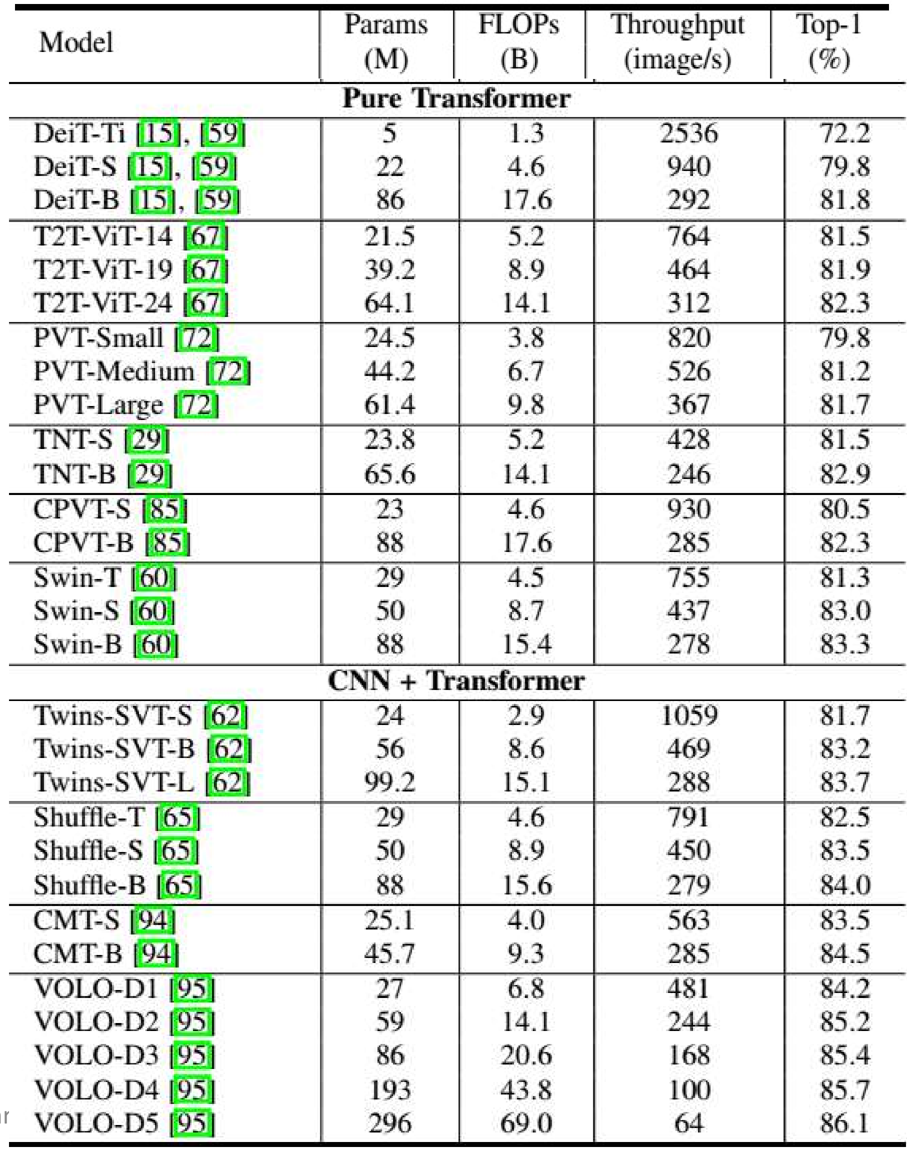


ImageNet result comparison of representative CNN and vision transformer models

Pure transformer means only using a few convolutions in the stem stage.
CNN + Transformer means using convolutions in the intermediate layers.

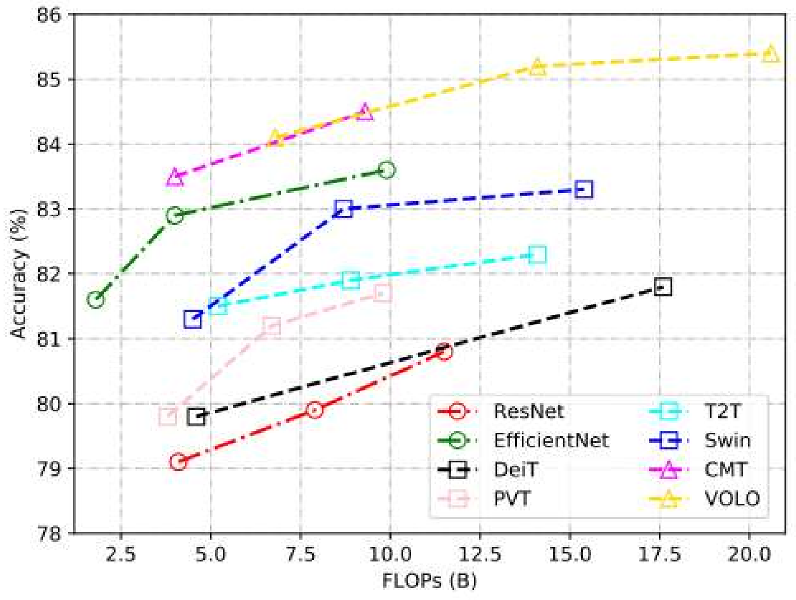


Model	Params (M)	FLOPs (B)	Throughput (image/s)	Top-1 (%)
CNN				
ResNet-50 [12], [67]	25.6	4.1	1226	79.1
ResNet-101 [12], [67]	44.7	7.9	753	79.9
ResNet-152 [12], [67]	60.2	11.5	526	80.8
EfficientNet-B0 [93]	5.3	0.39	2694	77.1
EfficientNet-B1 [93]	7.8	0.70	1662	79.1
EfficientNet-B2 [93]	9.2	1.0	1255	80.1
EfficientNet-B3 [93]	12	1.8	732	81.6
EfficientNet-B4 [93]	19	4.2	349	82.9

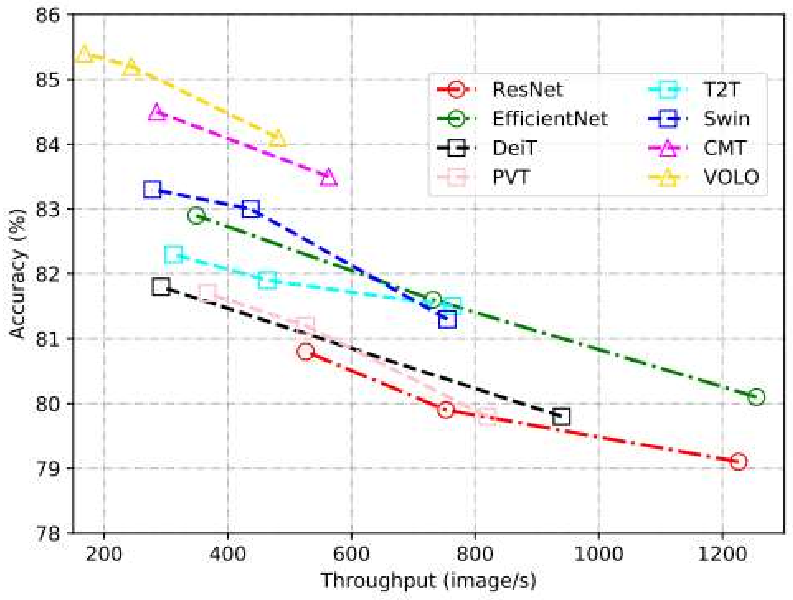


Model	Params (M)	FLOPs (B)	Throughput (image/s)	Top-1 (%)
Pure Transformer				
DeiT-Ti [15], [59]	5	1.3	2536	72.2
DeiT-S [15], [59]	22	4.6	940	79.8
DeiT-B [15], [59]	86	17.6	292	81.8
T2T-ViT-14 [67]	21.5	5.2	764	81.5
T2T-ViT-19 [67]	39.2	8.9	464	81.9
T2T-ViT-24 [67]	64.1	14.1	312	82.3
PVT-Small [72]	24.5	3.8	820	79.8
PVT-Medium [72]	44.2	6.7	526	81.2
PVT-Large [72]	61.4	9.8	367	81.7
TNT-S [29]	23.8	5.2	428	81.5
TNT-B [29]	65.6	14.1	246	82.9
CPVT-S [85]	23	4.6	930	80.5
CPVT-B [85]	88	17.6	285	82.3
Swin-T [60]	29	4.5	755	81.3
Swin-S [60]	50	8.7	437	83.0
Swin-B [60]	88	15.4	278	83.3
CNN + Transformer				
Twins-SVT-S [62]	24	2.9	1059	81.7
Twins-SVT-B [62]	56	8.6	469	83.2
Twins-SVT-L [62]	99.2	15.1	288	83.7
Shuffle-T [65]	29	4.6	791	82.5
Shuffle-S [65]	50	8.9	450	83.5
Shuffle-B [65]	88	15.6	279	84.0
CMT-S [94]	25.1	4.0	563	83.5
CMT-B [94]	45.7	9.3	285	84.5
VOLO-D1 [95]	27	6.8	481	84.2
VOLO-D2 [95]	59	14.1	244	85.2
VOLO-D3 [95]	86	20.6	168	85.4
VOLO-D4 [95]	193	43.8	100	85.7
VOLO-D5 [95]	296	69.0	64	86.1

FLOPs and throughput comparison of representative CNN and vision transformer models (ImageNet Top-1 (%)).



(a) Acc v.s. FLOPs.

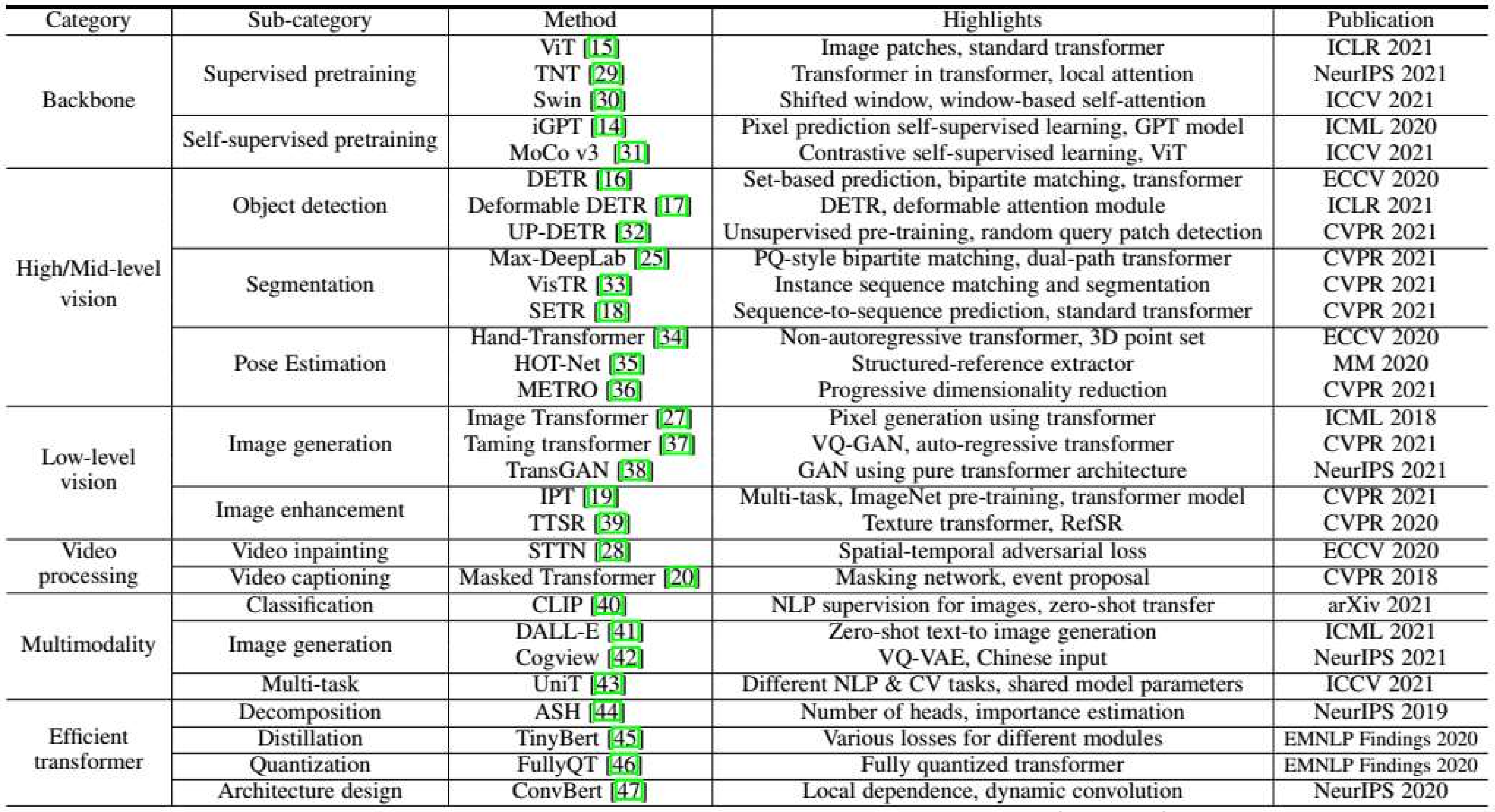


(b) Acc v.s. throughput.

Throughput is how much information actually gets delivered in a certain amount of time.

FLOP: Floating point operations per second

TABLE 1: Representative works of vision transformers.



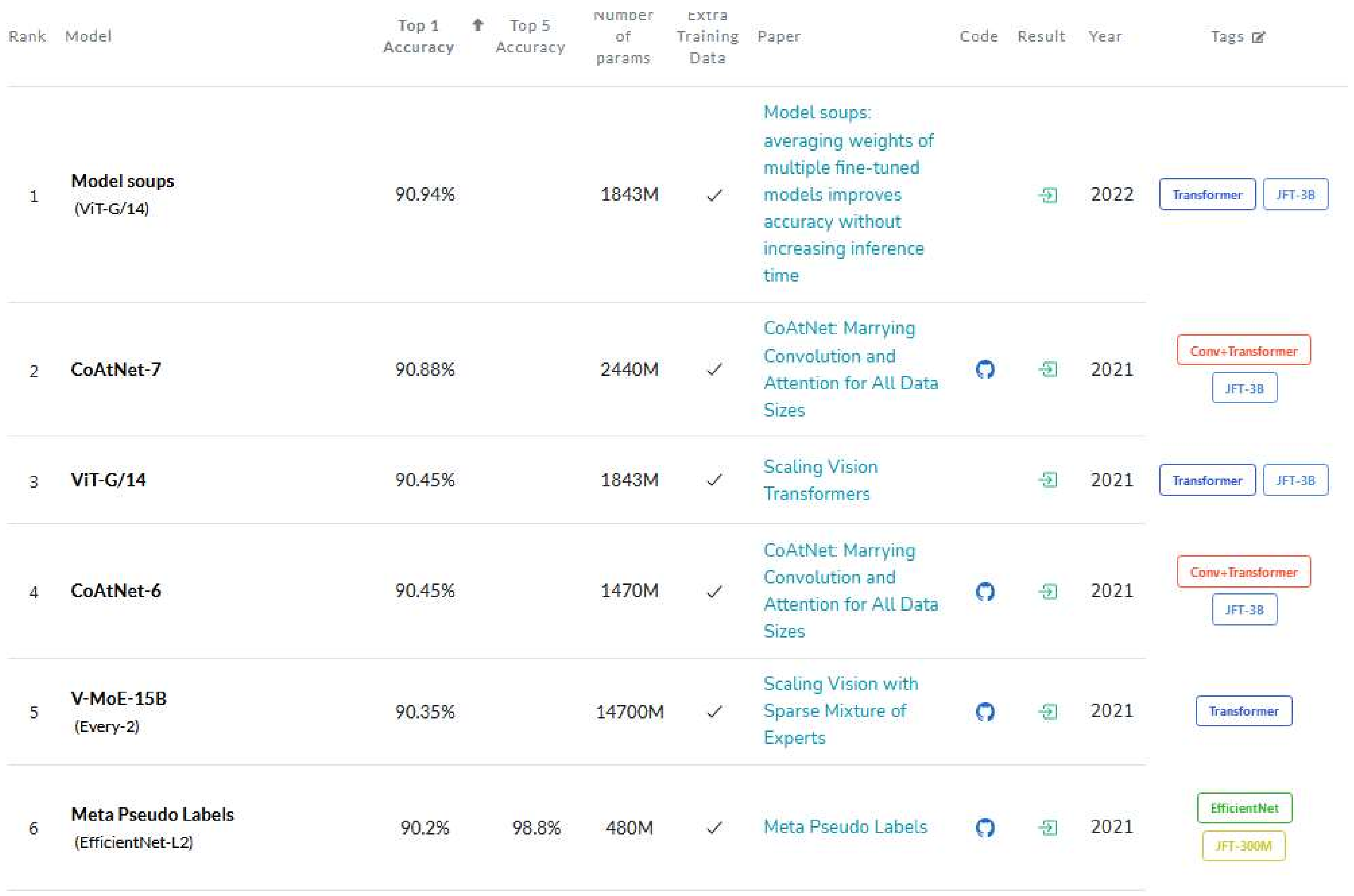
Category	Sub-category	Method	Highlights	Publication
Backbone	Supervised pretraining	ViT [15]	Image patches, standard transformer	ICLR 2021
		TNT [29]	Transformer in transformer, local attention	NeurIPS 2021
		Swin [30]	Shifted window, window-based self-attention	ICCV 2021
	Self-supervised pretraining	iGPT [14]	Pixel prediction self-supervised learning, GPT model	ICML 2020
		MoCo v3 [31]	Contrastive self-supervised learning, ViT	ICCV 2021
High/Mid-level vision	Object detection	DETR [16]	Set-based prediction, bipartite matching, transformer	ECCV 2020
		Deformable DETR [17]	DETR, deformable attention module	ICLR 2021
		UP-DETR [32]	Unsupervised pre-training, random query patch detection	CVPR 2021
	Segmentation	Max-DeepLab [25]	PQ-style bipartite matching, dual-path transformer	CVPR 2021
		VisTR [33]	Instance sequence matching and segmentation	CVPR 2021
		SETR [18]	Sequence-to-sequence prediction, standard transformer	CVPR 2021
	Pose Estimation	Hand-Transformer [34]	Non-autoregressive transformer, 3D point set	ECCV 2020
		HOT-Net [35]	Structured-reference extractor	MM 2020
		METRO [36]	Progressive dimensionality reduction	CVPR 2021
Low-level vision	Image generation	Image Transformer [27]	Pixel generation using transformer	ICML 2018
		Taming transformer [37]	VQ-GAN, auto-regressive transformer	CVPR 2021
		TransGAN [38]	GAN using pure transformer architecture	NeurIPS 2021
	Image enhancement	IPT [19]	Multi-task, ImageNet pre-training, transformer model	CVPR 2021
		TTSR [39]	Texture transformer, RefSR	CVPR 2020
Video processing	Video inpainting	STTN [28]	Spatial-temporal adversarial loss	ECCV 2020
	Video captioning	Masked Transformer [20]	Masking network, event proposal	CVPR 2018
Multimodality	Classification	CLIP [40]	NLP supervision for images, zero-shot transfer	arXiv 2021
	Image generation	DALL-E [41]	Zero-shot text-to image generation	ICML 2021
		Cogview [42]	VQ-VAE, Chinese input	NeurIPS 2021
	Multi-task	UniT [43]	Different NLP & CV tasks, shared model parameters	ICCV 2021
Efficient transformer	Decomposition	ASH [44]	Number of heads, importance estimation	NeurIPS 2019
	Distillation	TinyBert [45]	Various losses for different modules	EMNLP Findings 2020
	Quantization	FullyQT [46]	Fully quantized transformer	EMNLP Findings 2020
	Architecture design	ConvBert [47]	Local dependence, dynamic convolution	NeurIPS 2020

*High-level vision deals with the interpretation and use of what is seen in the image **(events)**

*Mid-level vision deals with how this information is organized as objects and their attributes **(classification, detection, segmentation)**

Low-level vison deals with how this information is organized as edges, blobs, and… (In image generation and enhancement)

ImageNet Benchmark (Image Classification) | Papers With Code



Rank	Model	Top 1 Accuracy	Top 5 Accuracy	Number of params	Extra Training Data	Paper	Code	Result	Year	Tags
1	**Model soups** (ViT-G/14)	90.94%		1843M	✓	Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time			2022	Transformer JFT-3B
2	**CoAtNet-7**	90.88%		2440M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	Conv+Transformer JFT-3B
3	**ViT-G/14**	90.45%		1843M	✓	Scaling Vision Transformers			2021	Transformer JFT-3B
4	**CoAtNet-6**	90.45%		1470M	✓	CoAtNet: Marrying Convolution and Attention for All Data Sizes			2021	Conv+Transformer JFT-3B
5	**V-MoE-15B** (Every-2)	90.35%		14700M	✓	Scaling Vision with Sparse Mixture of Experts			2021	Transformer
6	**Meta Pseudo Labels** (EfficientNet-L2)	90.2%	98.8%	480M	✓	Meta Pseudo Labels			2021	EfficientNet JFT-300M

End of Chapter 6

Thank you