Transformer Tutorial

Partial slides from:
Haibin Ling, Michael Ryoo, Stony Brook
Graham Neubig, CS11-747, CMU LIT,
Fei-Fei Li, Ranjay Krishna, Danfei Xu, cs231n, Stanford University
Ming Li, CS 886, Waterloo University
Jay Alammar, "The Illustrated Transformer"
Xipeng Qiu, "*A Tutorial of Transformers", Fudan University*
Aman Arora, https://amaarora.github.io/2021/01/18/ViT.html

1

Overview

- Attention
- Transformer Introduction
- Self-attention and Multi-head self-attention
- Position Encoding
- Residual and Decoder
- Transformer Variants in NLP
- Vision Transformer
- Codes for Vision Transformer
- Recent Vision Transformer Paper

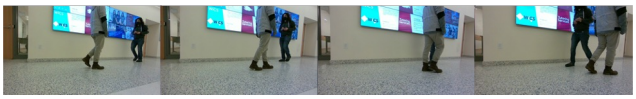
2

Overview

- Transformers were created for processing *sequential data*:
  - Natural language (sequence of words):

She ate a green apple

  - Video (sequence of images):



  - Output can be another sequence

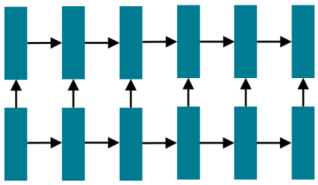
She ate a green apple → **Elle a mangé une pomme verte**

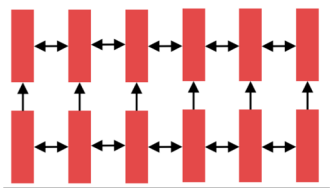
English French

3

Models for sequential data are not new

- Recurrent models:
  - Recurrent Neural Networks (RNNs)
    - Long Short-Term Memory Networks (LSTMs)
      - Bi-directional LSTMS ....



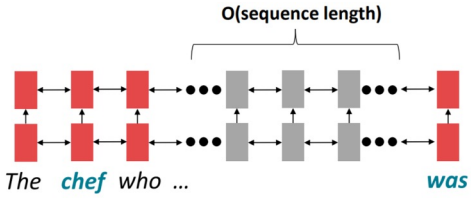


- Why are these methods surpassed by Transformer?

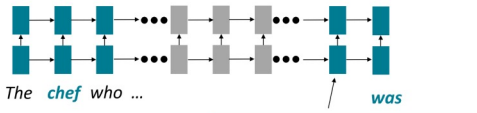
4

Recurrent Models have long interaction distance

- Taking O(sequence length) steps for distant units to interact



- Meaning that hard to long-distance relations
  - Both Information and gradient flow through so many layers

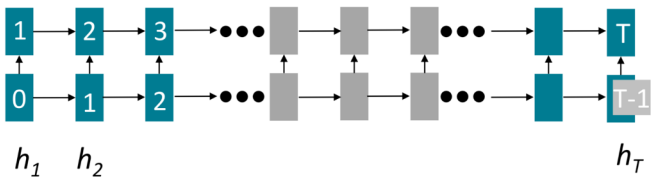


Info of ***chef*** has gone through O(sequence length) many layers!

5

Recurrent Models lack parallelizability

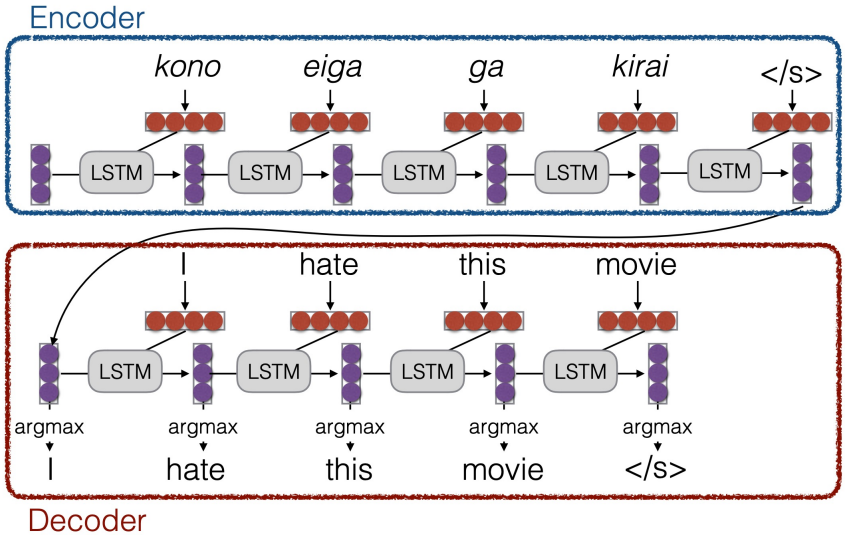
- Forward and backward passes have O(sequence length)
  - Later states are computed only after previous states are computed
  - Hard to train on long sequences, large datasets
- GPU is not good at this "serial" computation
  - Instead, it can do many independent computations in parallel



Numbers indicate min # of steps before a state can be computed

6

Encoder-Decoder Models



Sutskever et al. "Sequence to sequence learning with neural networks", NeurIPS 2014

7

Problem of Sentence Representation:

"You can't cram the meaning of a whole %&!$ing sentence into a single $&!*ing vector!" — Ray Mooney

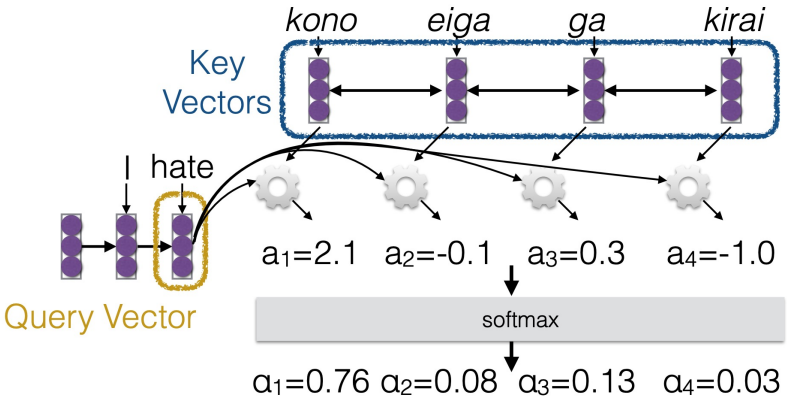
Basic idea of Attention:

- Encode each word in the sentence into a vector
- When decoding, perform a linear combination of these vectors, weighted by "attention weights"
- Use this combination in picking the next word

8

Calculating Attention

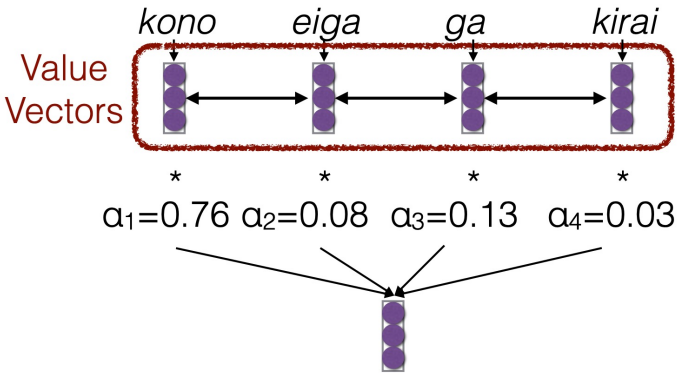
- Use “query” vector (decoder state) and “key” vectors (all encoder states)
- For each query-key pair, calculate weight
- Normalize to add to one using softmax



9

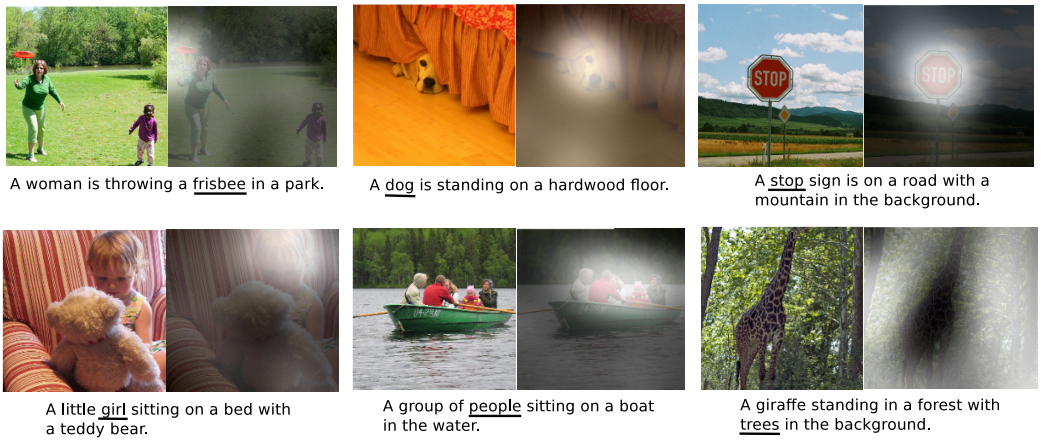
Calculating Attention

- Combine together value vectors (usually encoder states, like key vectors) by taking the weighted sum



10

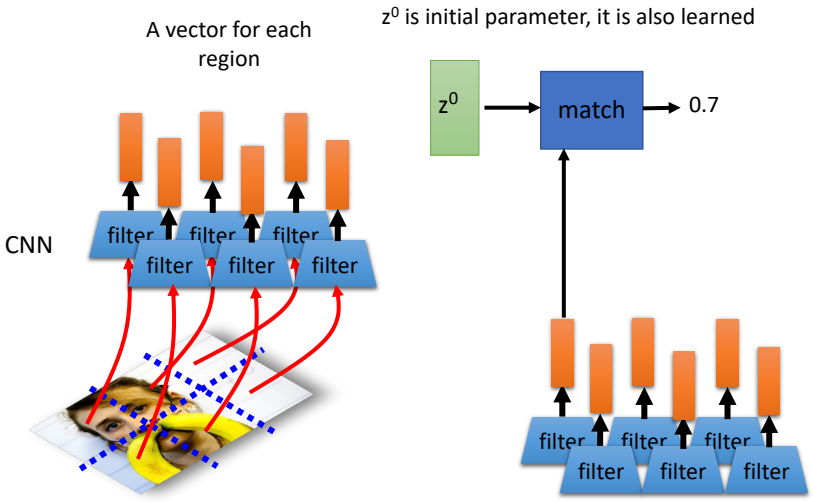
Image caption generation using attention



Kelvin Xu, et. al. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML, 2015

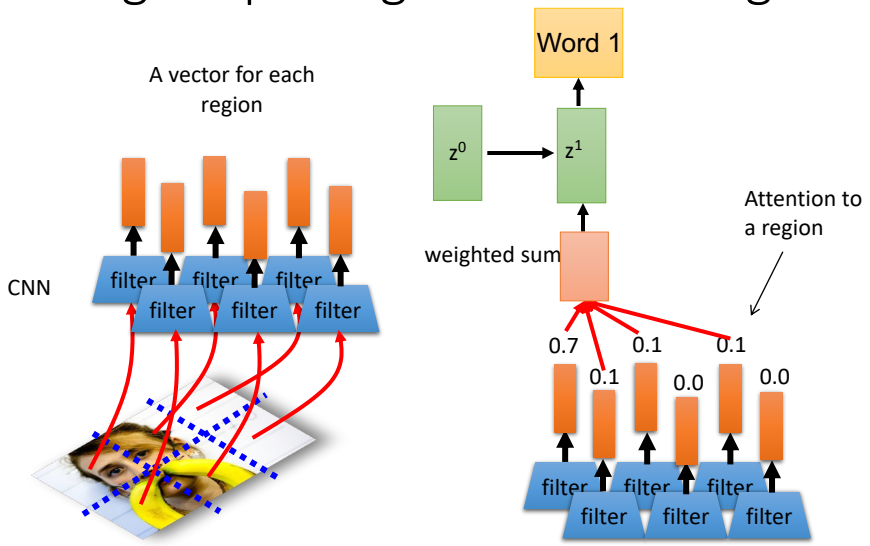
11

Image caption generation using attention



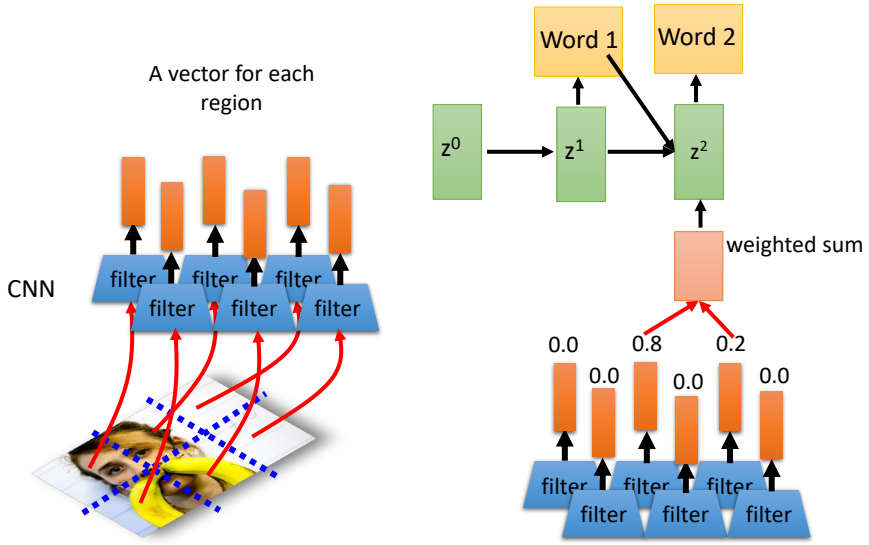
12

Image caption generation using attention



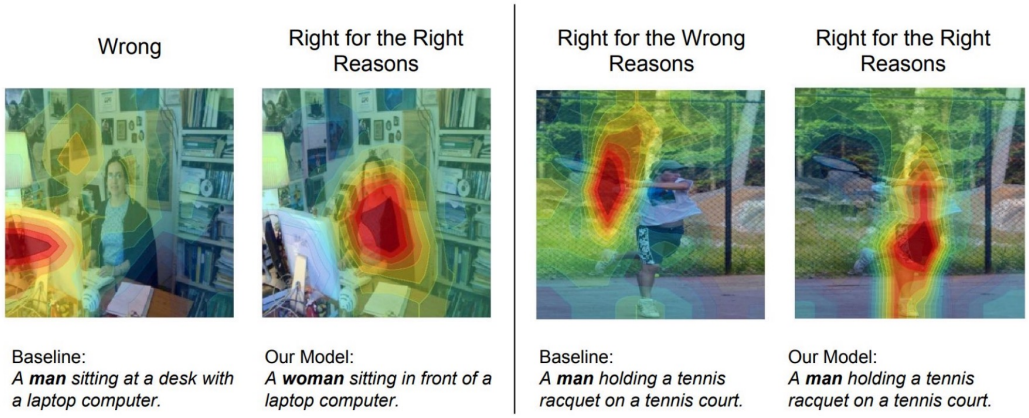
13

Image caption generation using attention



14

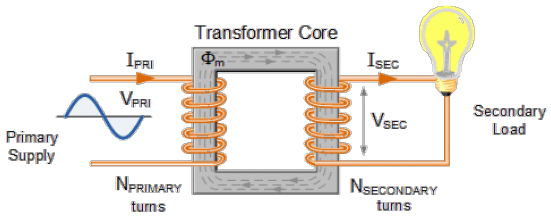
Attention can detect Gender Bias

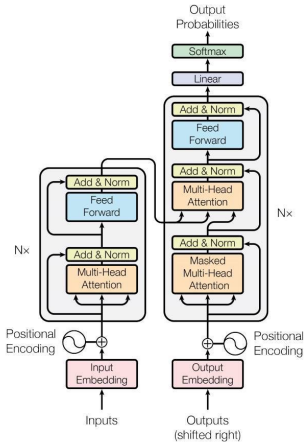


Burns et al. "Women also Snowboard: Overcoming Bias in Captioning Models" ECCV 2018

15

Transformers?





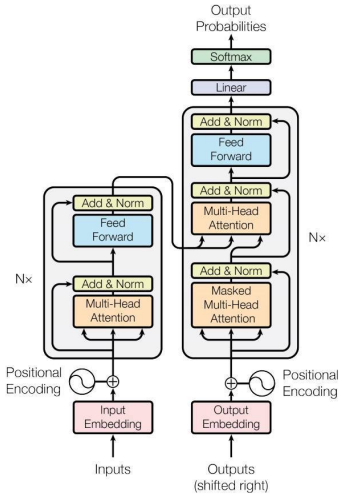
A neural network!!!

16

Introduction

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com



Vaswani, Ashish, et al. "Attention is All you Need." NeurIPS. 2017.

17

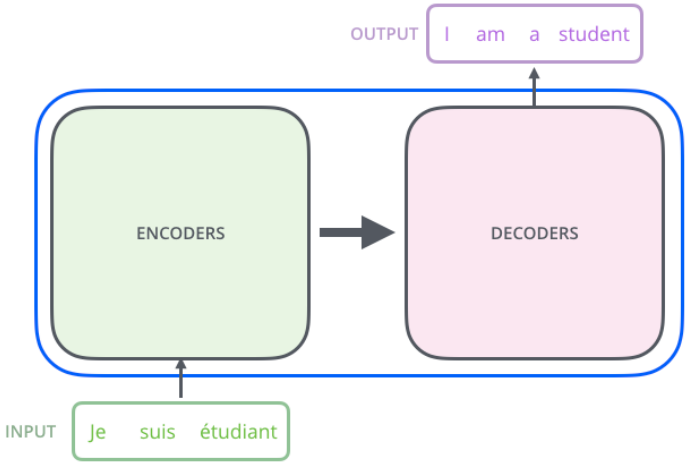
Overview

- Input: A sentence in one language,
- Output: Translation in another.

https://jalammar.github.io/illustrated-transformer/ (slides come from this source)

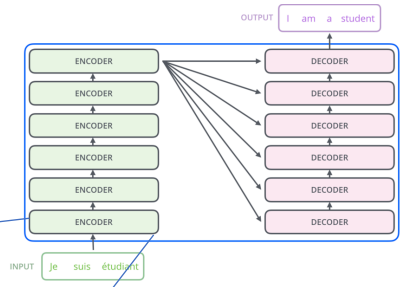
18

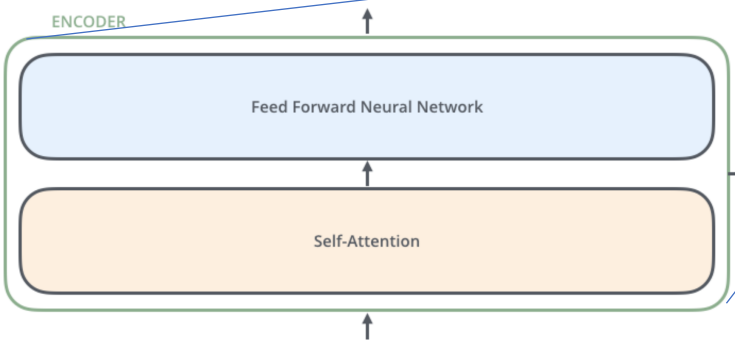
- A encoder-decoder model



19

The **encoders** are **all identical in structure** (yet they do not share weights). Each one is broken down into two sub-layers





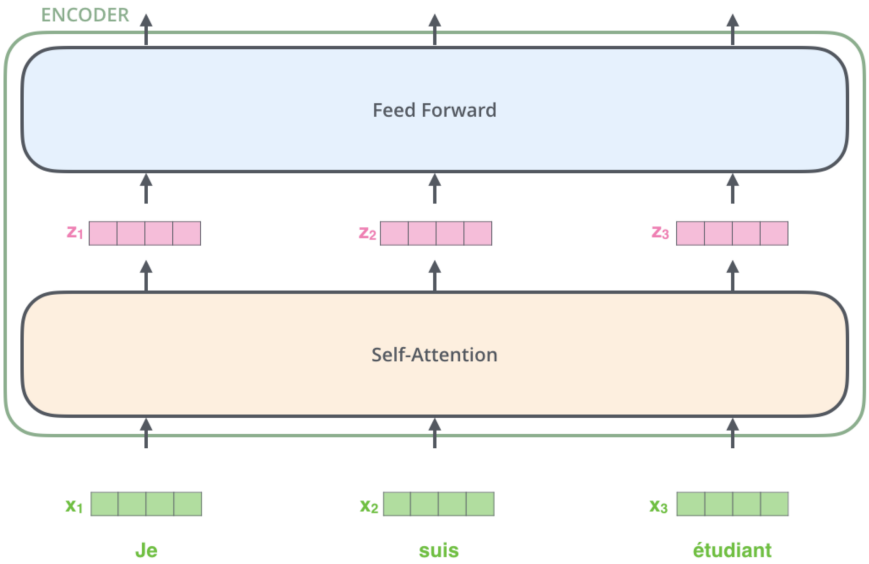
outputs of the self-attention are fed to a feed-forward neural network. The exact same one is independently applied to each position.

helps the encoder look at other words in the input sentence as it encodes a specific word.

20

Key property of Transformer: word in each position flows through its own path in the encoder.

- There are dependencies between these paths in the self-attention layer.
- Feed-forward layer does not have those dependencies => various paths can be executed in parallel !

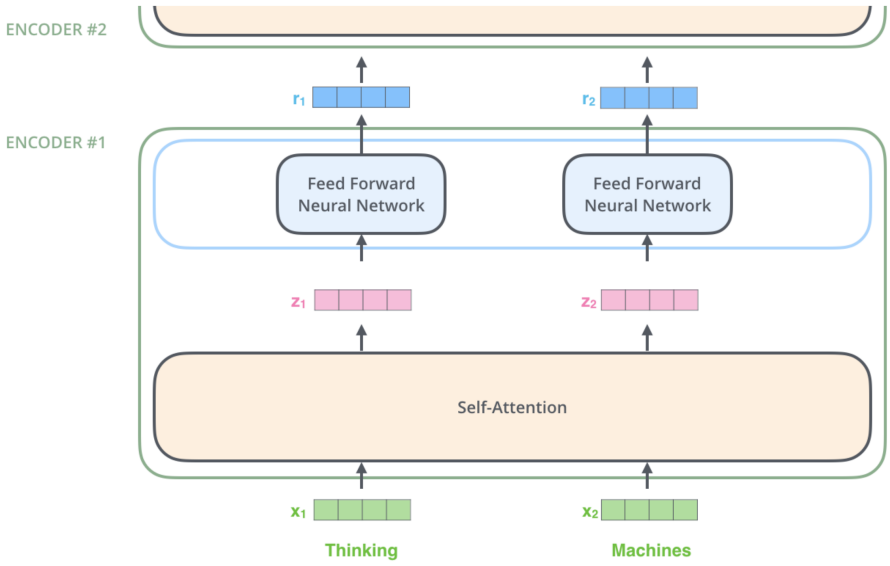


Word embeddings

21

Visually clearer on two words

- dependencies in self-attention layer.
- No dependencies in Feed-forward layer



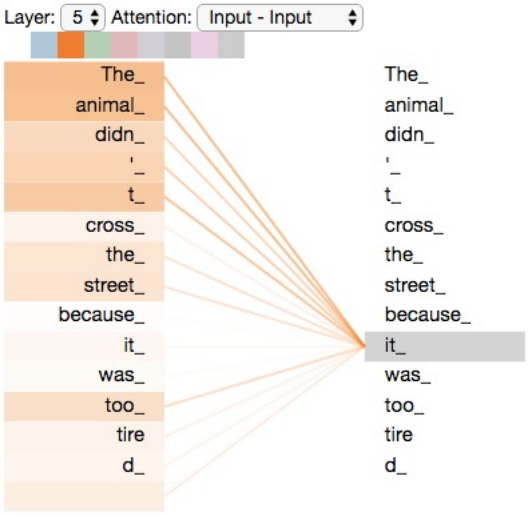
Word embeddings

22

Self-attention

”The animal didn't cross the street because it was too tired”

What does “it” in this sentence refer to?



23

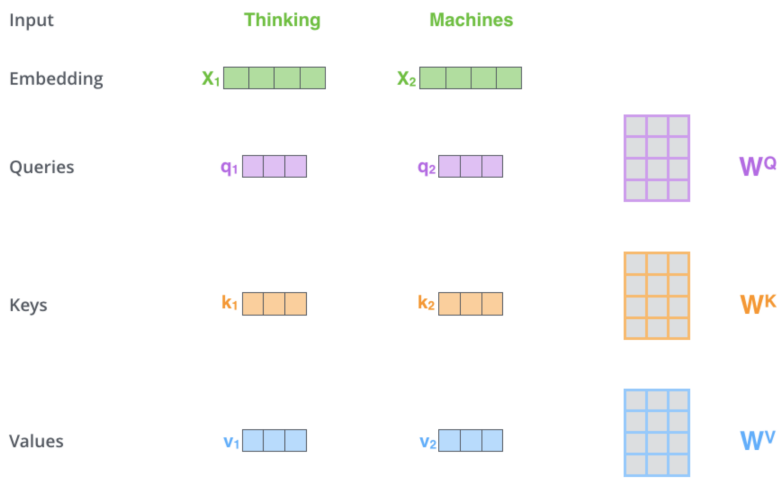
Self-Attention

While processing **each word** it allows to look at other positions in the input sequence for clues to build a better encoding for **this word**.

Step1: create three vectors from each of the encoder’s input vectors:

Query, Key, Value (typically smaller dimension).

by multiplying the embedding by three matrices that we **trained** during the training process.

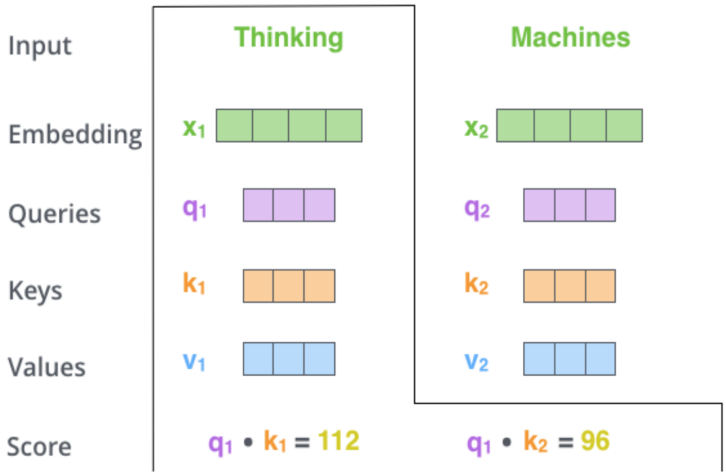


24

Self-Attention

Step 2: calculate a score (like we have seen for regular attention!) how much focus to place on other parts of the input sentence as we encode a word at a certain position.

Take dot product of the query vector with the key vector of the respective word we're scoring.

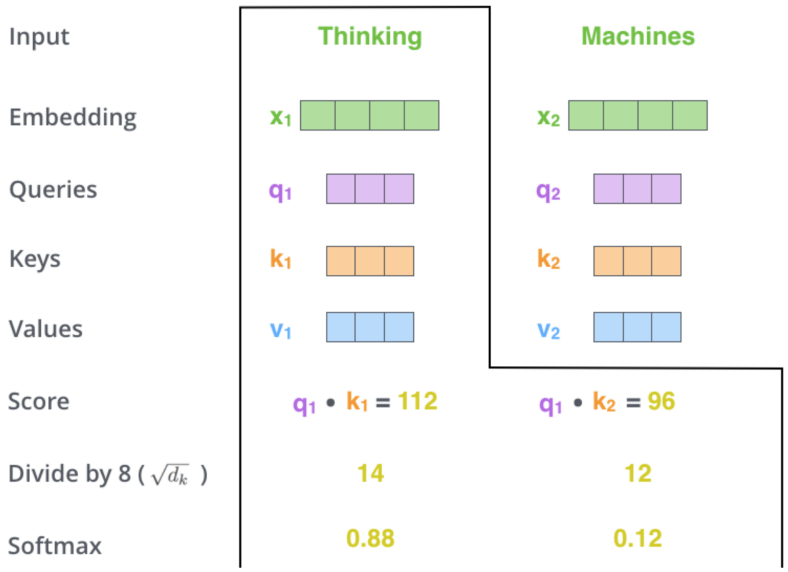


E.g., Processing the self-attention for word "Thinking" in position #1, the first score would be the dot product of q1 and k1. The second score would be the dot product of q1 and k2.

25

Self-Attention

- **Step 3** divide scores by the square root of the dimension of the key vectors (more stable gradients).
- **Step 4** pass result through a softmax operation. (all positive and add up to 1)



Intuition: softmax score determines how much each word will be expressed at this position.

26

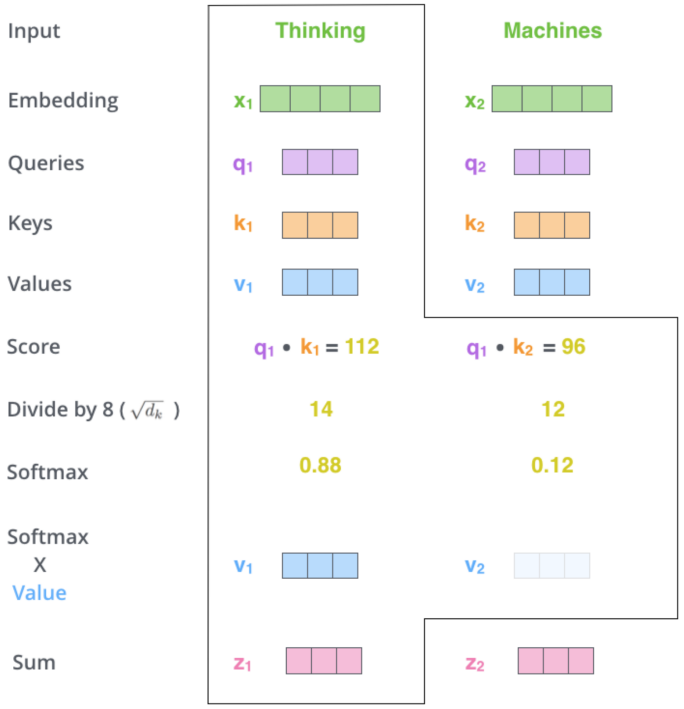
Self-Attention

- **Step5:** multiply each value vector by the softmax score (in preparation to sum them up).
- **Step6** : sum up the weighted value vectors. This produces the output of the self-attention layer at this position

More details:

- What we have seen for a word is done **for all words** (using matrices)
- Need to **encode position** of words
- And improved using a mechanism called "**multi-headed**" attention

(kind of like multiple filters for CNN)

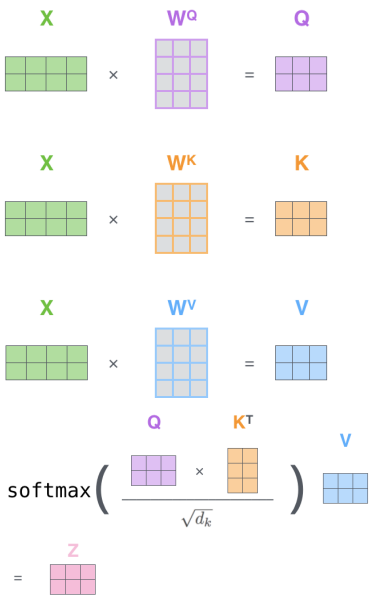


27

Matrix Calculation of Self-Attention

- **The first step** is to calculate the Query, Key, and Value matrices. We do that by packing our embeddings into a matrix X, and multiplying it by the weight matrices we've trained (WQ, WK, WV).

- **Then**, since we're dealing with matrices, we can condense steps two through six in one formula to calculate the outputs of the self-attention layer.

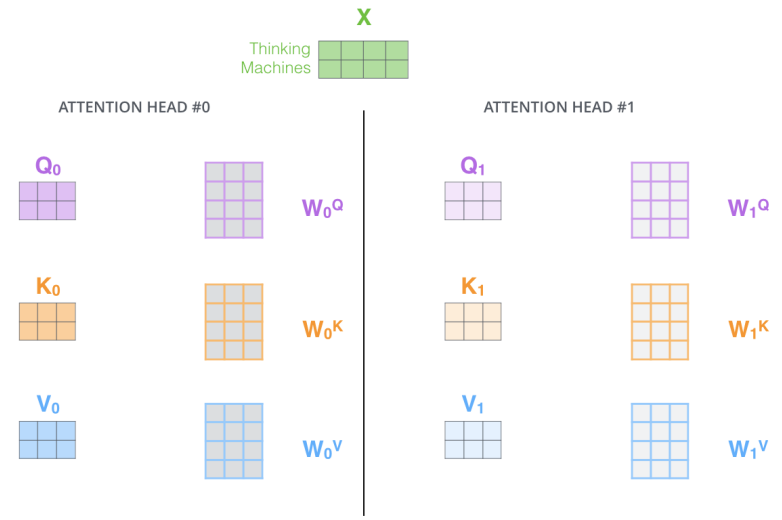


28

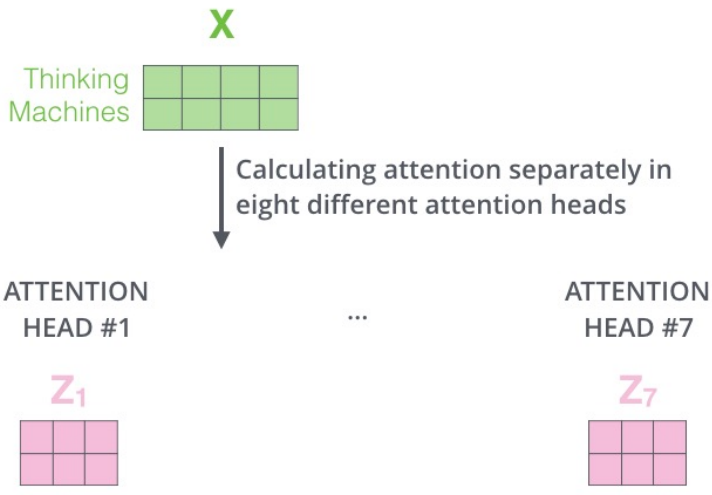
Multi-head self-attention

Why?

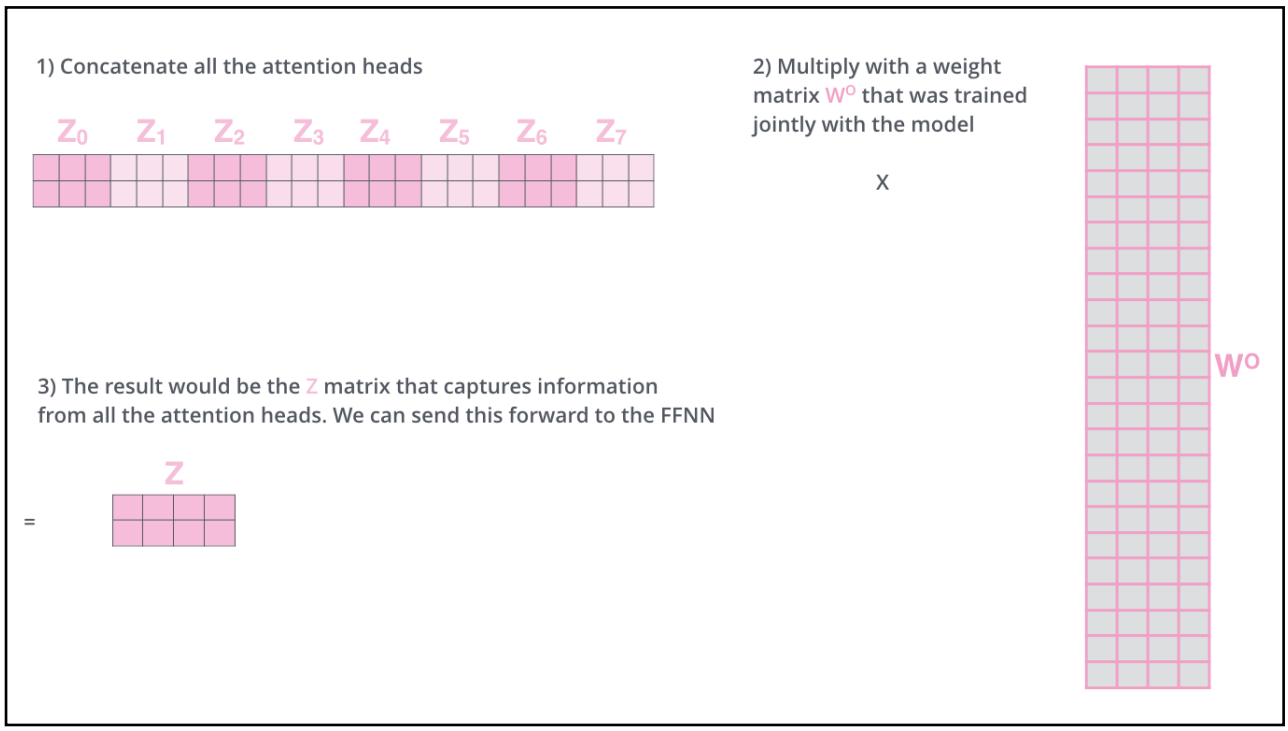
1. It expands the model's ability to focus on different positions.
2. It gives the attention layer multiple "representation subspaces".



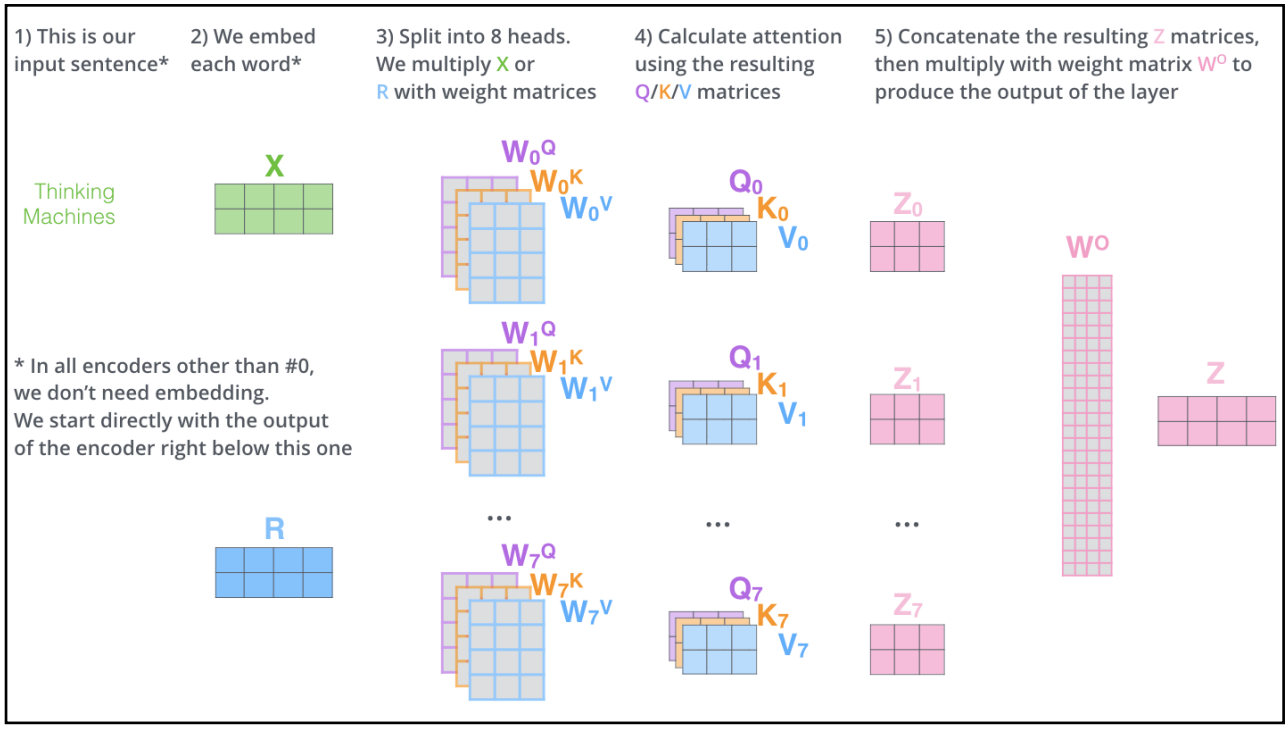
29



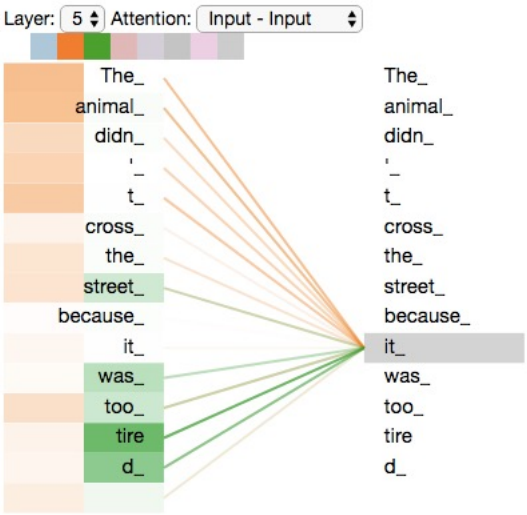
30


1) Concatenate all the attention heads
Z0 Z1 Z2 Z3 Z4 Z5 Z6 Z7
2) Multiply with a weight matrix WO that was trained jointly with the model
X
WO
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN
=
Z

31


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 WO to produce the output of the layer
Thinking Machines
X
W0Q W0K W0V
Q0 K0 V0
Z0
WO
Z
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one
W1Q W1K W1V
Q1 K1 V1
Z1
R
...
W7Q W7K W7V
Q7 K7 V7
Z7

32



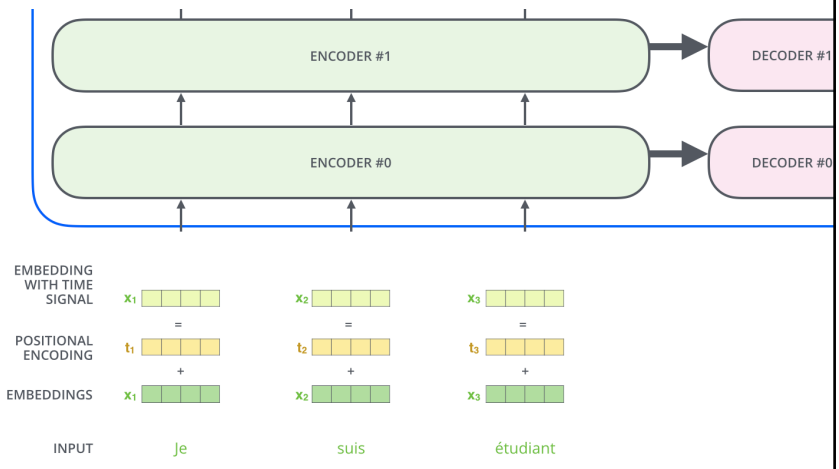
- As we encode the word "it" one attention head is focusing most on "the animal", while another is focusing on "tired" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

33

Position encoding

How to find a way to account for the order of the words in the input sequence?

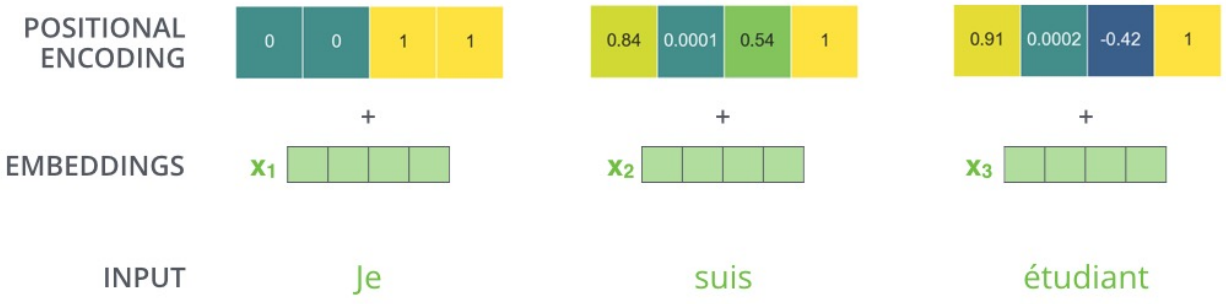
Adds a vector to each input embedding.



34

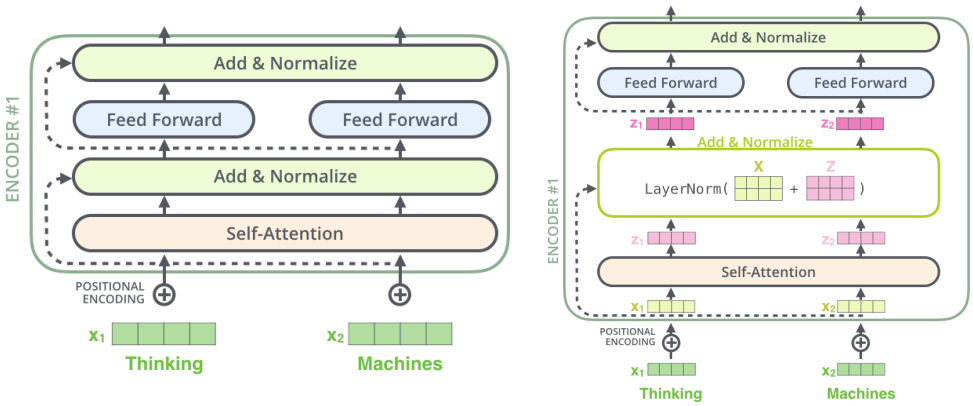
An example of position embedding

- Assume the embeddings has a dimensionality of 4:



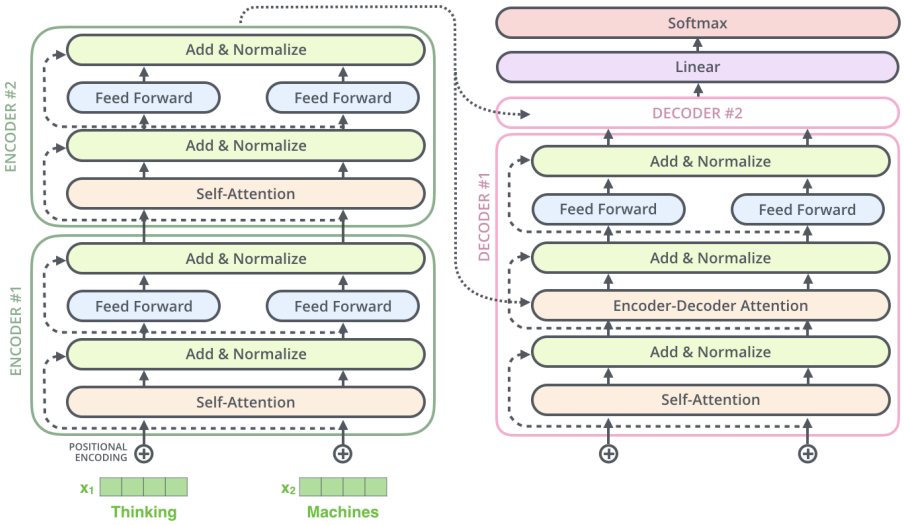
35

Residuals



37

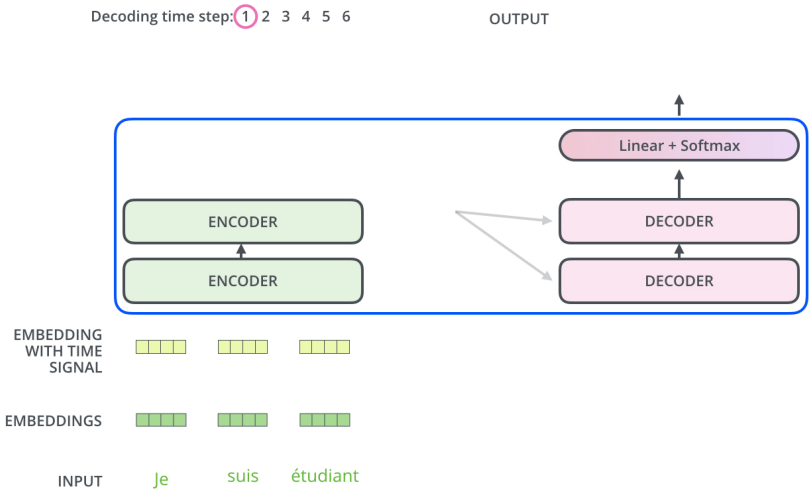
Residuals



38

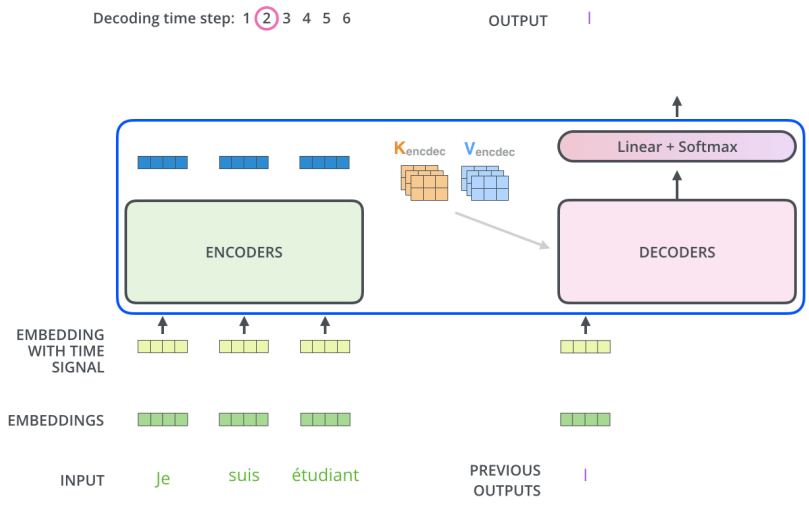
Decoder

In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence.



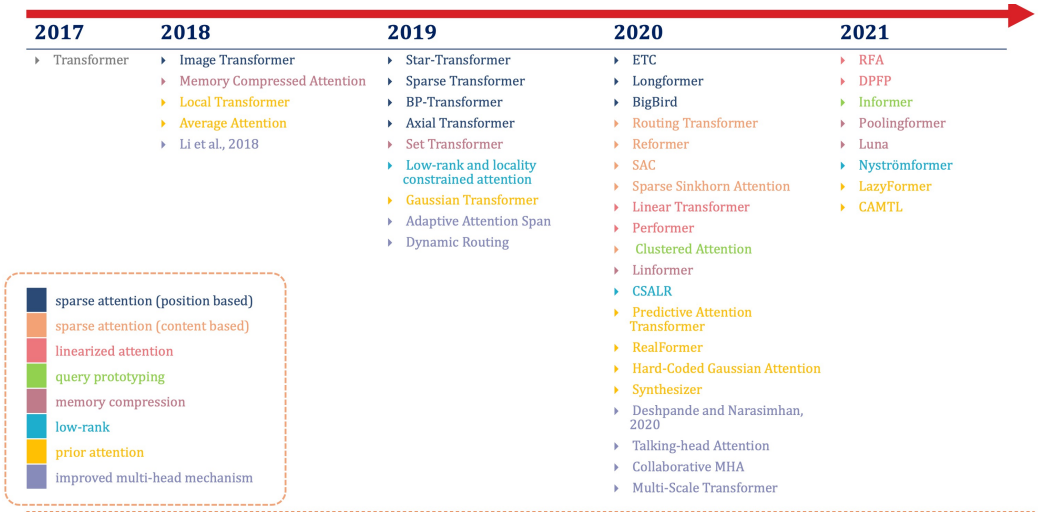
39

Decoder



40

Transformer Variants



Picture from " A Tutorial of Transformers ", Xipeng Qiu

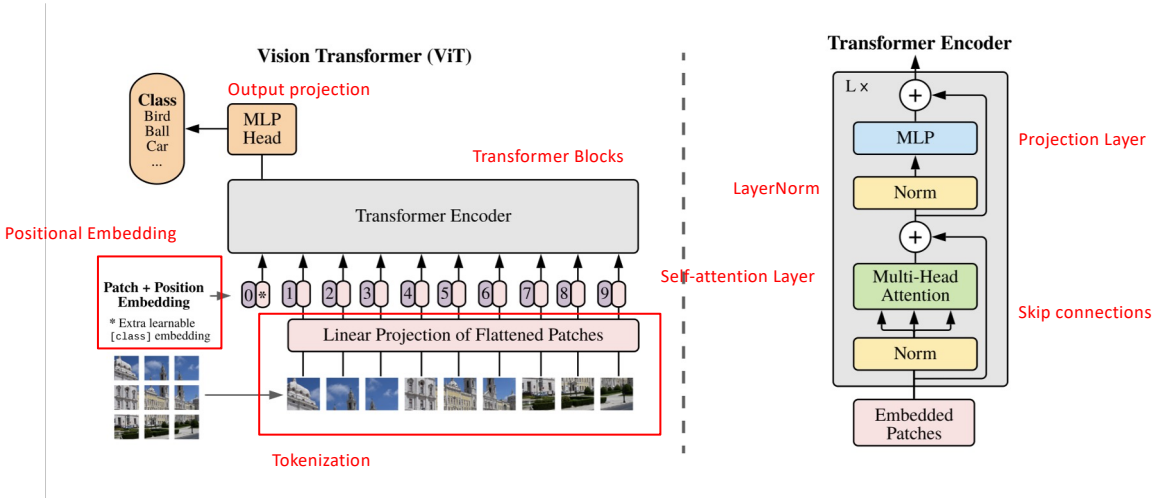
41

Vision Transformer

From github.com/lucidrains/vit-pytorch

42

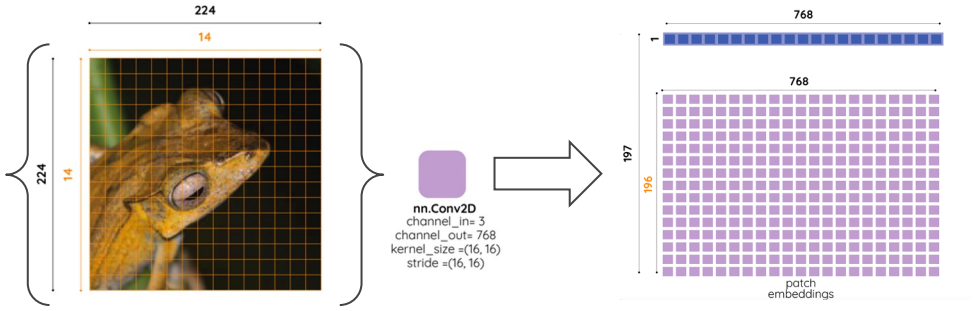
Vision Transformer



43

Tokenization

- Extracts non-overlapping image patches to create a **sequence** of tokens.
- Usually implemented as a Conv Layer (with stride equals kernel size), followed by Flattening
- [Optional] Class Token appended for classification tasks

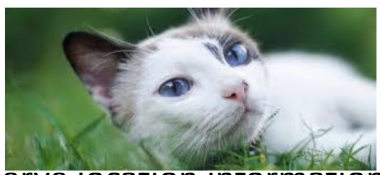


From https://tugot17.github.io/Vision-Transformer-Presentation

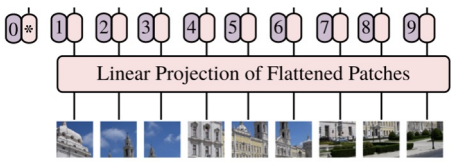
44

Positional Embedding

- The sequence of tokens are permutation invariant, a property which we do not want to have.



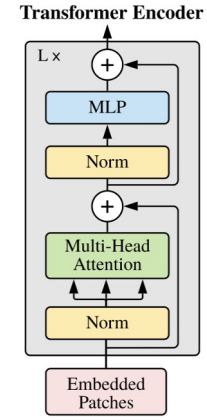
- To preserve location information of each token, we add positional embedding (Usually, a learnable tensor in vision)



45

Transformer Block

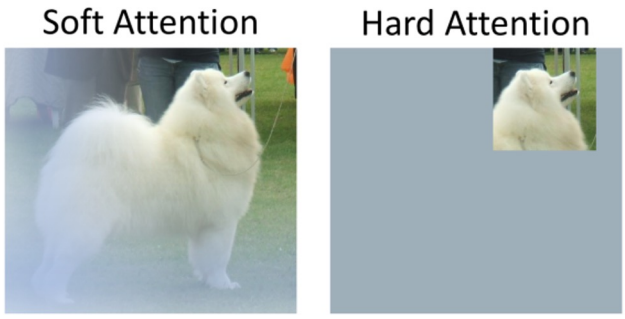
- LayerNorm normalizes each token in channel dimension to give stability (Also, makes training stable even with small batch sizes).
- Multi-head Self-attention (MSA) computes pairwise attention using projected Key, Query and Value representations.
- Skip-connections help preserve/propagate structure among tokens.
- Projection Layers (MLPs) mix information within each token (among channels).



46

Spatial attention for images

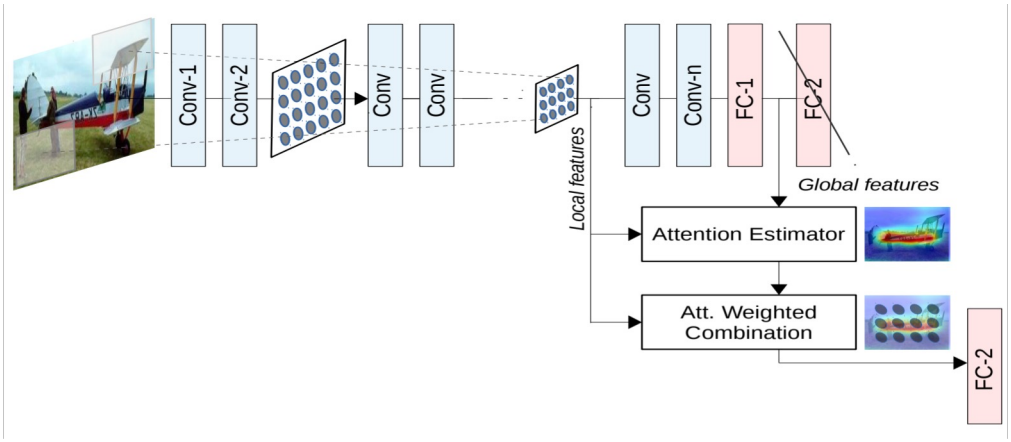
Constraining(?) what goes into the CNN



47

Spatial attention for images

Learn to generate a spatial weight matrix

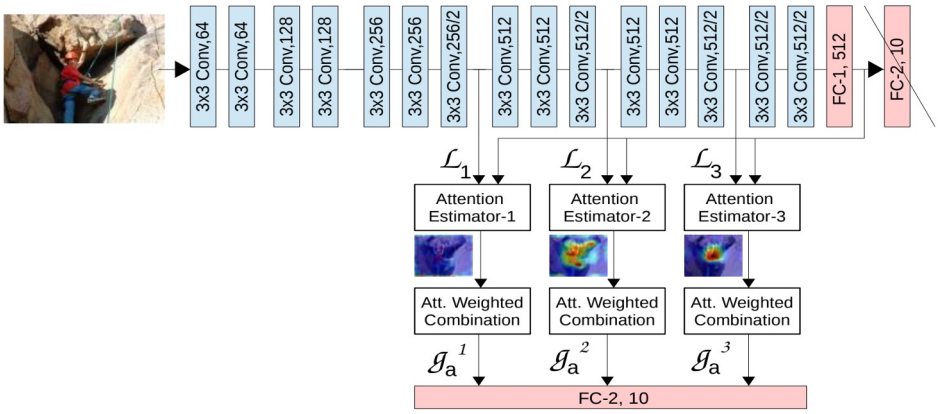


1804.02391.pdf (arxiv.org)

48

Spatial attention for images

Learn to generate a spatial weight matrix



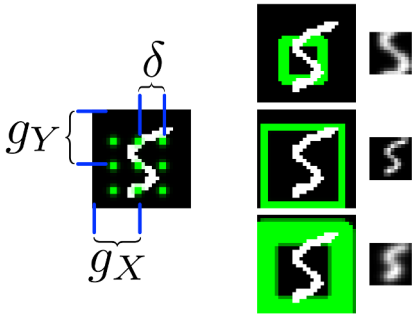
1804.02391.pdf (arxiv.org)

49

DRAW attention

Spatial attention for images

Representing attention in terms of few parameters



$$(\tilde{g}_X, \tilde{g}_Y, \log\sigma^2, \log\tilde{\delta}, \log\gamma) = W(h^{dec})$$

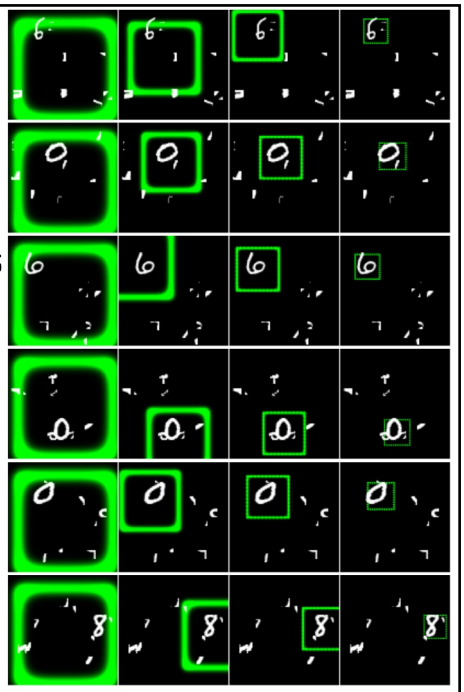
$$g_X = \frac{A+1}{2}(\tilde{g}_X + 1)$$

$$g_Y = \frac{B+1}{2}(\tilde{g}_Y + 1)$$

$$\delta = \frac{\max(A,B)-1}{N-1}\tilde{\delta}$$

$$F_X[i,a] = \frac{1}{Z_X}\exp\left(-\frac{(a-\mu_X^i)^2}{2\sigma^2}\right)$$

$$F_Y[j,b] = \frac{1}{Z_Y}\exp\left(-\frac{(b-\mu_Y^j)^2}{2\sigma^2}\right)$$



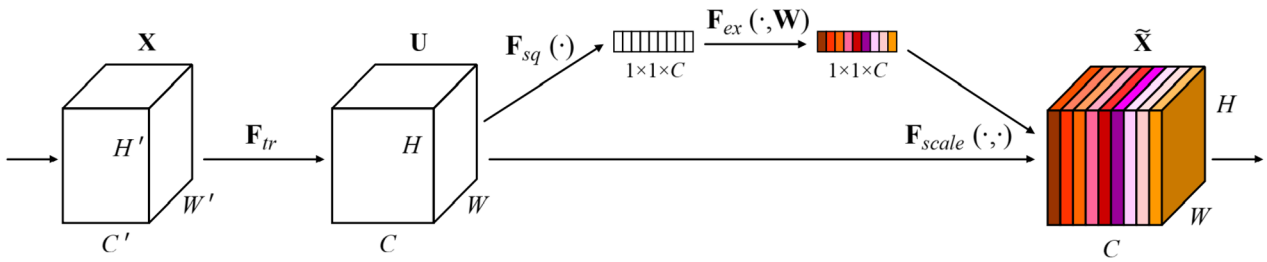
DRAW: A Recurrent Neural Network For Image Generation (arxiv.org)

50

Other forms of attention

Channel-wise attention: Squeeze-and-Excitation network

- Learn to weigh/select channels



51

Code for Vision Transformer

Pytorch Verision:

- https://github.com/rwightman/pytorch-image-models/blob/master/timm/models/vision_transformer.py
- https://github.com/lucidrains/vit-pytorch

TensorFlow Version:

- https://github.com/google-research/vision_transformer

59

Recent Research on Vision Transformer

- Github Website: https://github.com/dk-liang/Awesome-Visual-Transformer
- Representative works: DeiT, Swin Transformer (ICCV 2021 Best Paper), BEIT, MAE, DiNO
- Different Task:

  Detection: DETR, CT3D

  Tracking: TransT, TrackFormer, STARK

  Segmentation: VisTR, Swin Transformer, Segmenter

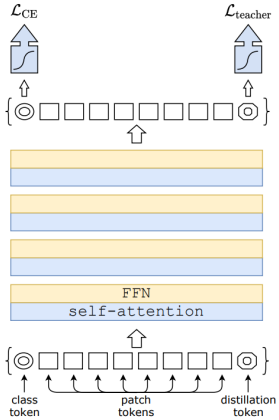
  …

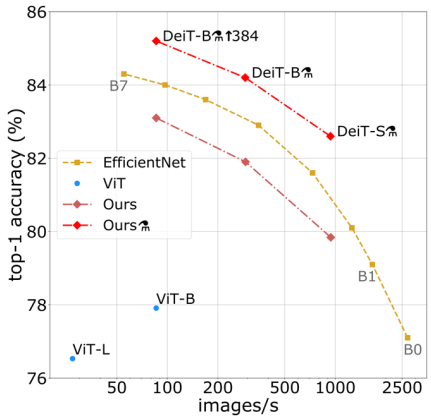
60

DeiT

Main idea:

Add a specific distillation token for Vision Transformer distillation Training





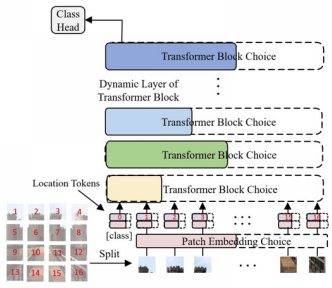
Touvron , Hugo, et al. "Training data-efficient image transformers & distillation through attention." ICML 2021.

61

AutoFormer

Main idea:

Search the optimal architecture of vision transformer with the proposed weight entanglement training strategy.



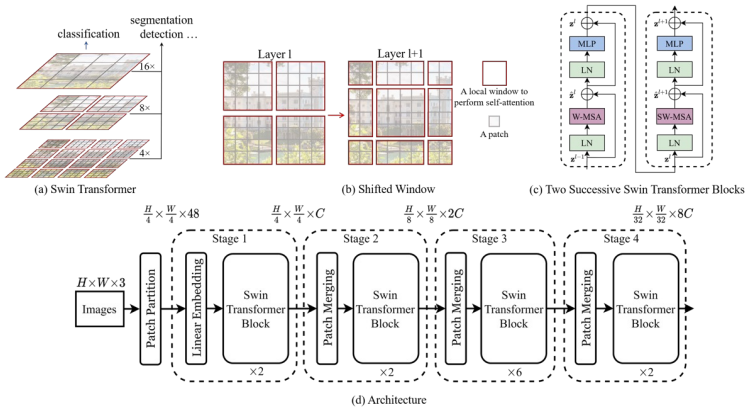
Chen , Minghao, et al. "AutoFormer: Searching Transformers for Visual Recognition." ICCV 2021.

62

Swin Transformer

Main idea:

Design a hierarchical vision transformer macro architecture and compute the attention only on predefined windows. At last, use shifted window for information communication.



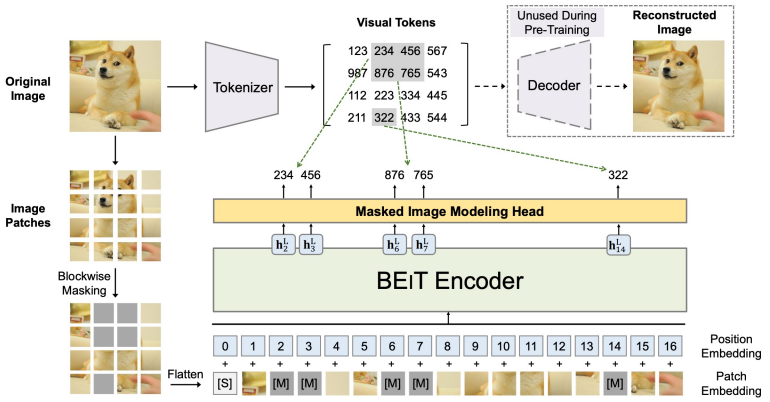
Liu, Zhe, et al. "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows." ICCV 2021.

63

BEIT

Main idea:

"tokenize" the original image into visual tokens. Then, randomly mask some image patches and predict the labels.



Bao, hangbo et al. "BEiT: BERT Pre-Training of Image Transformers." ICLR 2022.

64

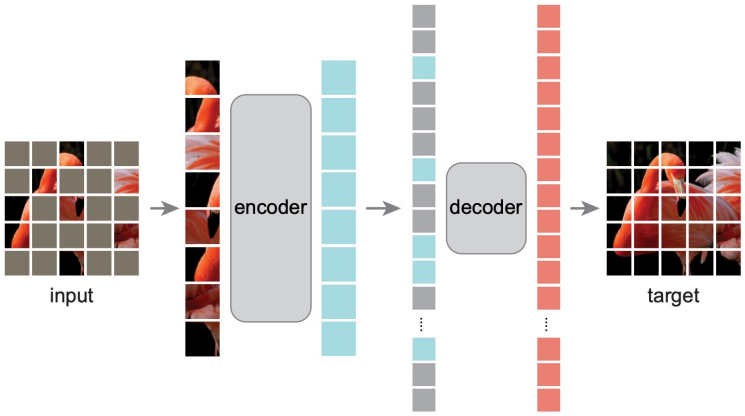
MAE

Main idea:

Mask random patches of the input image and reconstruct the missing pixels (pixel-level).

Encoder: Standard ViT

Decoder: Transformer Blocks



He, Kaiming et al. "Masked Autoencoders Are Scalable Vision Learners." CVPR 2022.

65

66