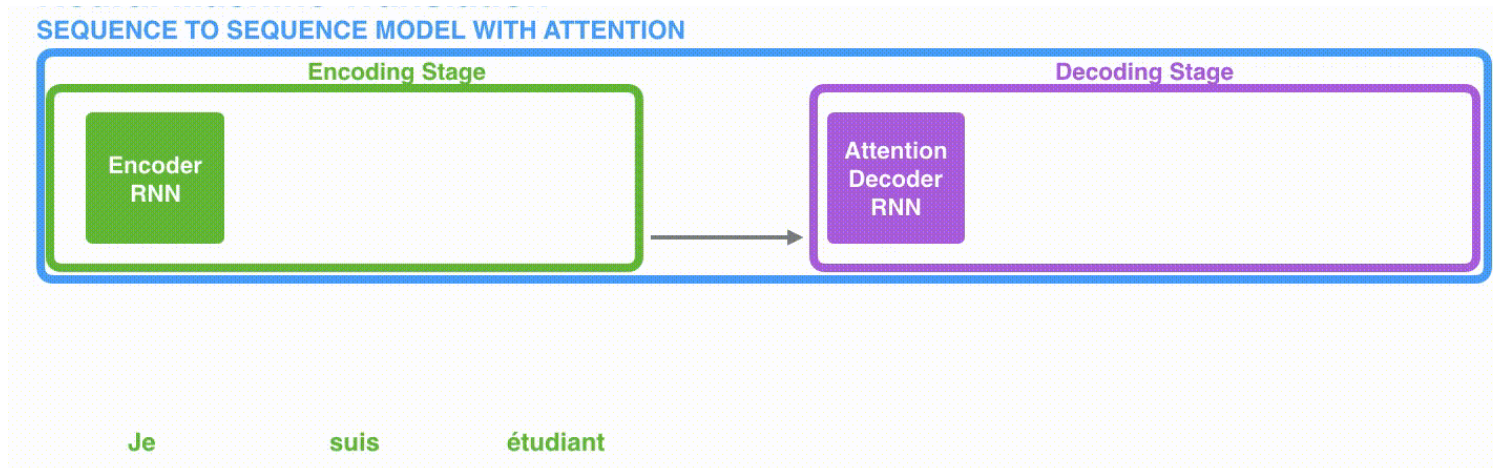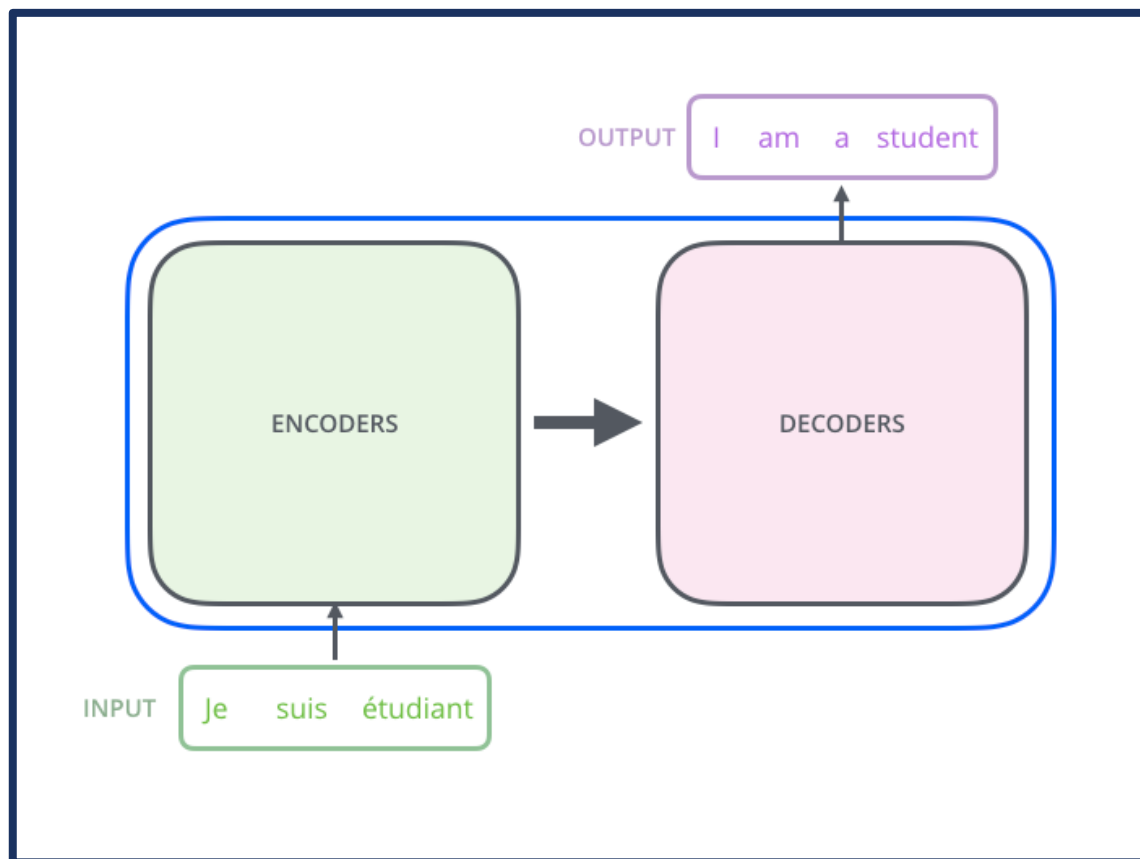# Lecture 09: Transformers

OVERVIEW

1. Transformer architecture
2. How transformers work
3. How to calculate self-attention
4. Multi-head attention
5. Transfer Learning

ATTENTION IS ALL YOU NEED

TRANSFORMERS

- **The Transformer** – proposed by Vaswani A., 2017 (Attention is all you need)

  - uses attention to boost the speed with which these models can be trained.

  - The transformer is based on a feedforward neural network rather than RNN.

  - The biggest benefit of the Transformer is the speed of computation due to parallelization.

  - All input sequences are processed simulataneously.

TRANSFORMER – ARCHITECTURE

- Architecture: stacked encoder – decoder architecture
  - The encoding component is a stack of encoders (original paper had six encoders)
    - encoders are all identical in structure
  - The decoding component is a stack of decoders of the same number as the encoder
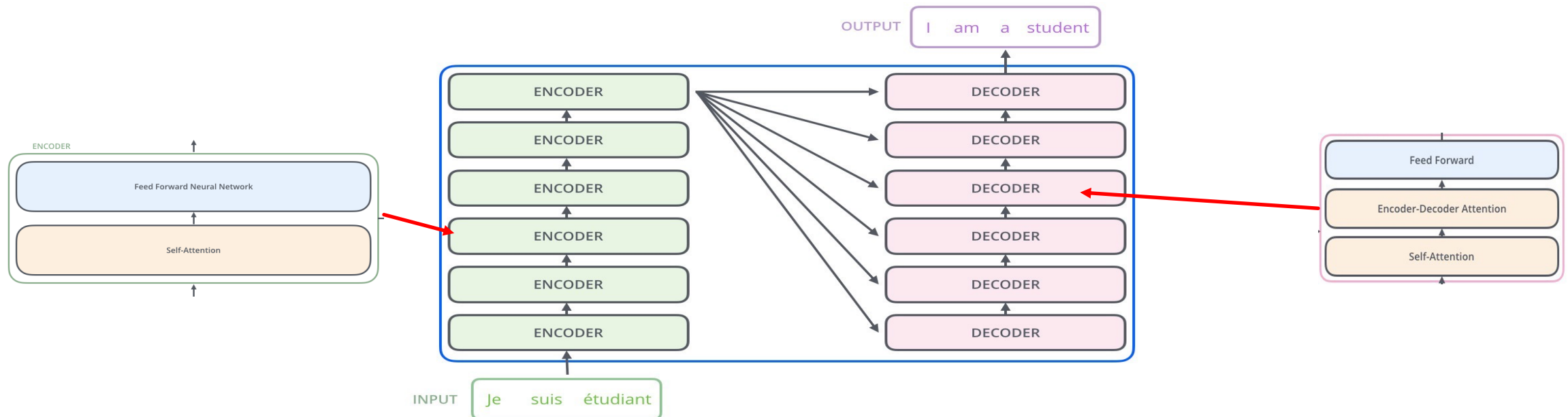
TRANSFORMER – ARCHITECTURE

Encoder

Inputs first flow through a self-attention layer

Outputs of the self-attention layer are fed to a feed-forward neural network.

Decoder

Input from encoder flows through self-attention layer then Encoder-Decoder Attention layer and finally through Feedforward layer

ENCODER

Feed Forward

Self-Attention

DECODER

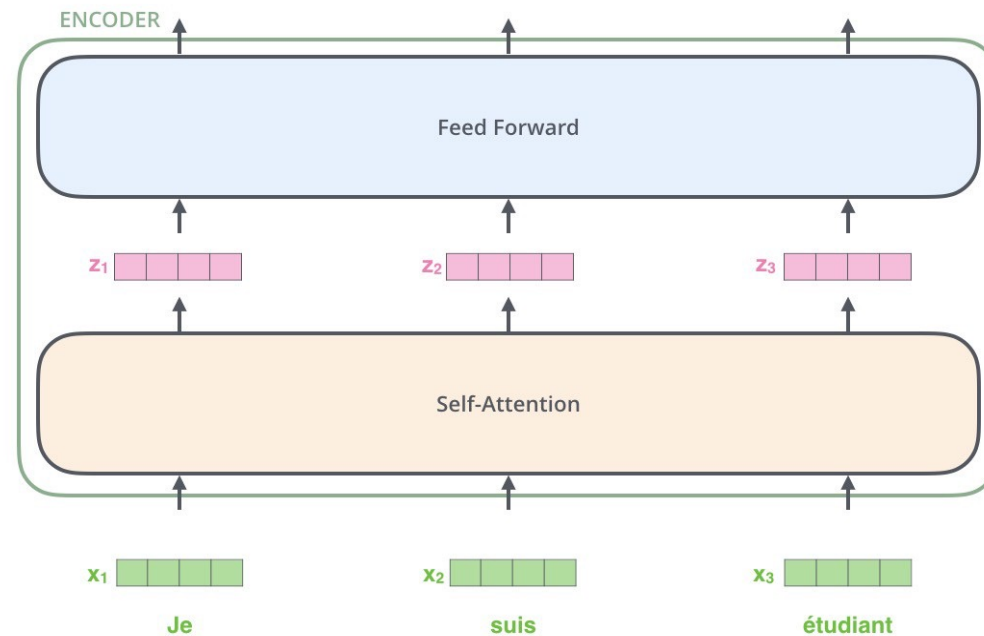Feed Forward

Encoder-Decoder Attention
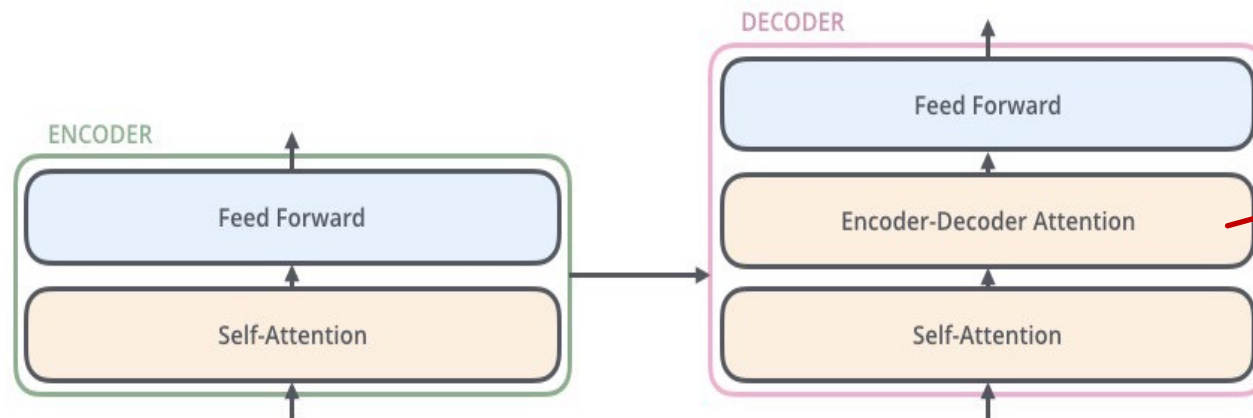
Self-Attention

Attention layer:
- helps decoder focus on relevant parts of the input

HOW THE TRANSFORMER WORKS

- Convert input word into a word embedding vectors
- Each word in the sentence flows through its own path in the encoder

ENCODER

Feed Forward

z_1 z_2 z_3

Self-Attention

x_1 Je x_2 suis x_3 étudiant

HOW THE TRANSFORMER WORKS – ENCODING

- Encoder receive input words as vectors
- Self attention processes the input and sent to the feedforward network
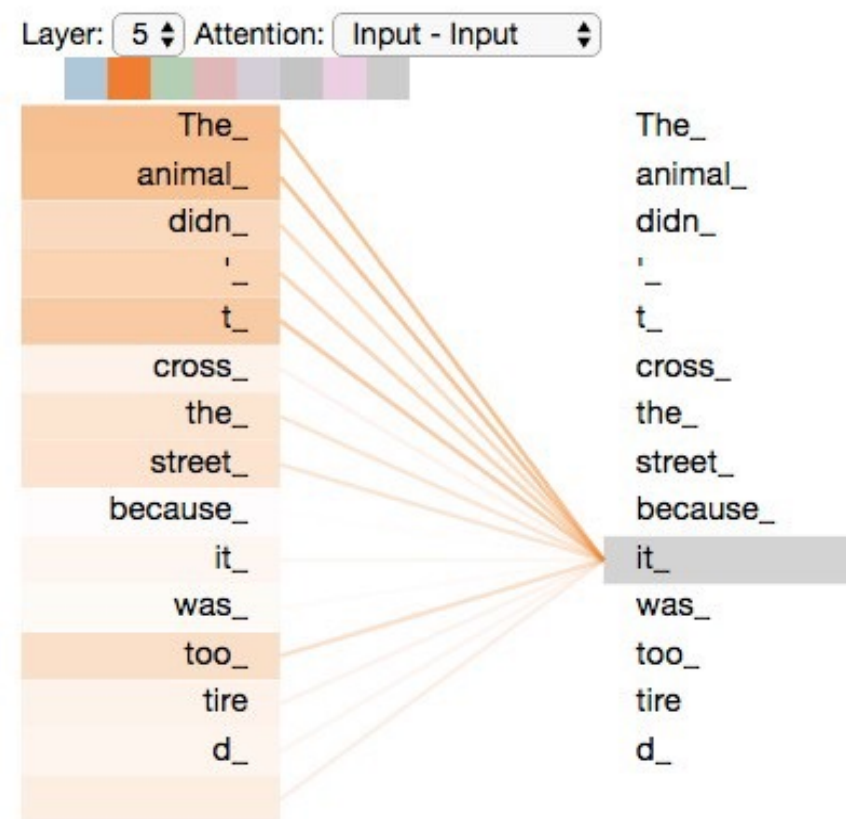- Feedforward sends output to next encoder layer.
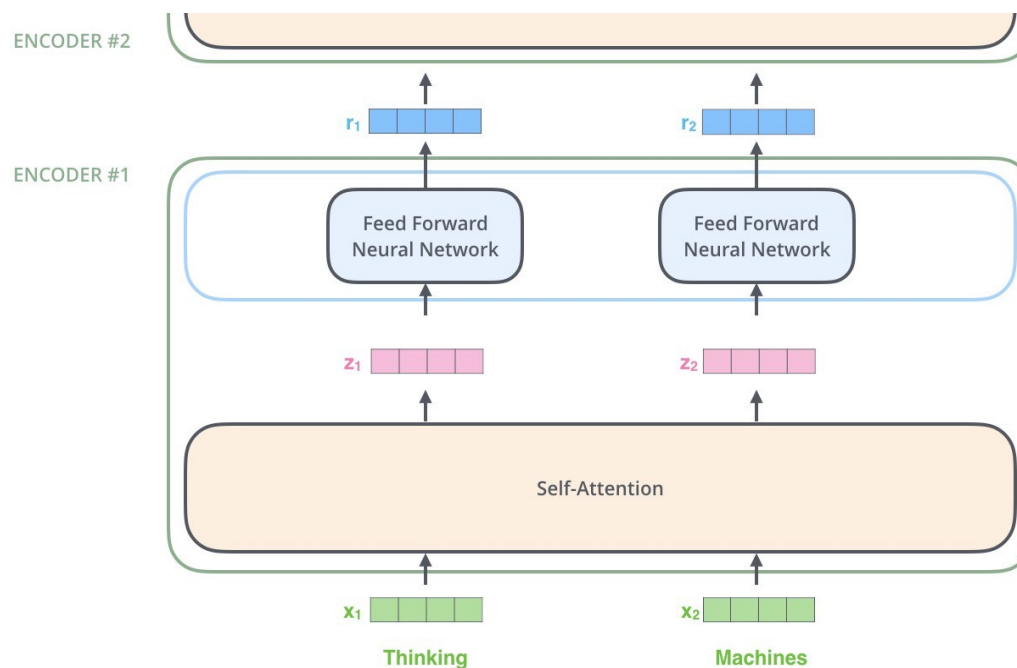
TRANSFORMER – WHAT IS SELF ATTENTION?

Example:

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

What does "*it*" in this sentence refer to?

- *Is it the street or*
- *the animal*

- self-attention allows the transformer model to associate "*it*" with "*animal*".

- self attention allows it to look at other positions for clues to help the model better encode the word *it*

https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb
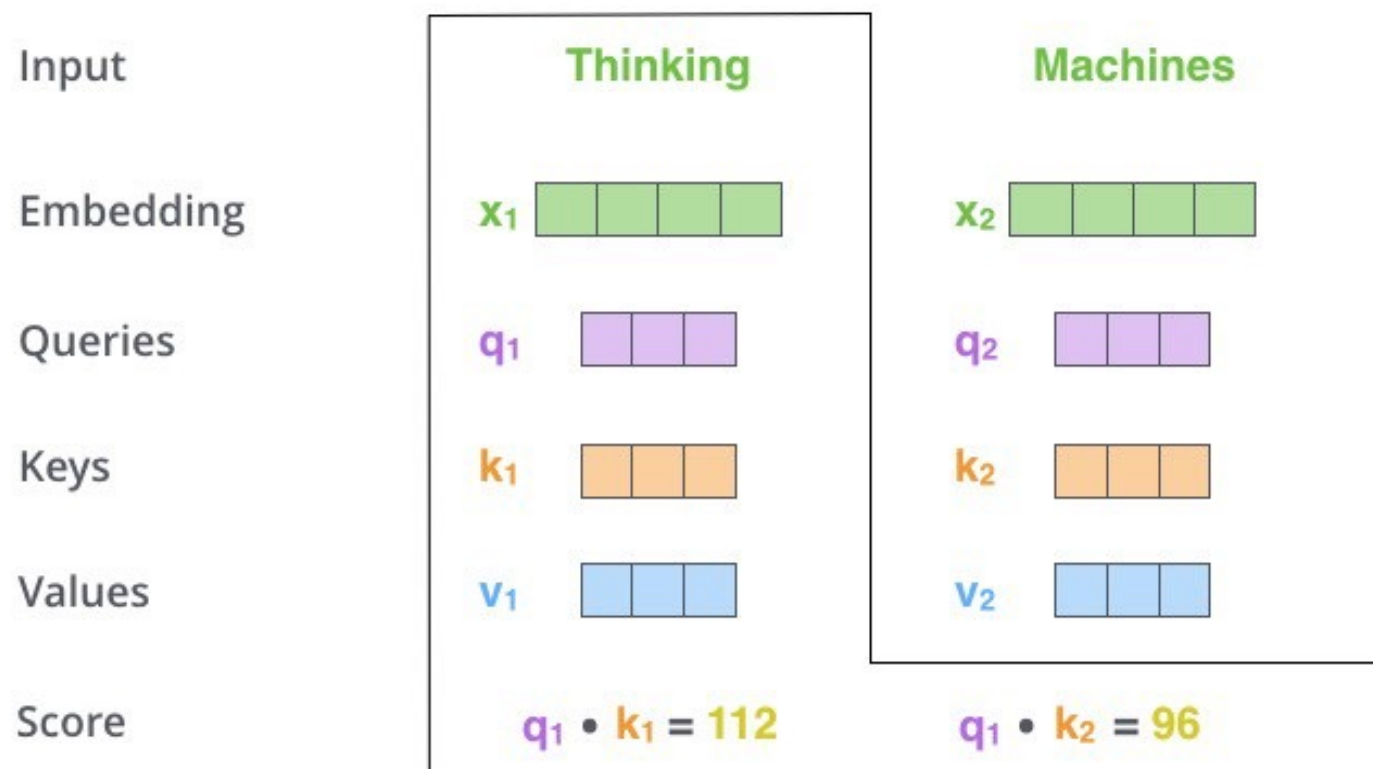
HOW TO CALCULATE SELF ATTENTION – STEP 1

Create 3 vectors (**Q**uery, **k**ey, **V**alue – dimension 64) from each encoder's input vector
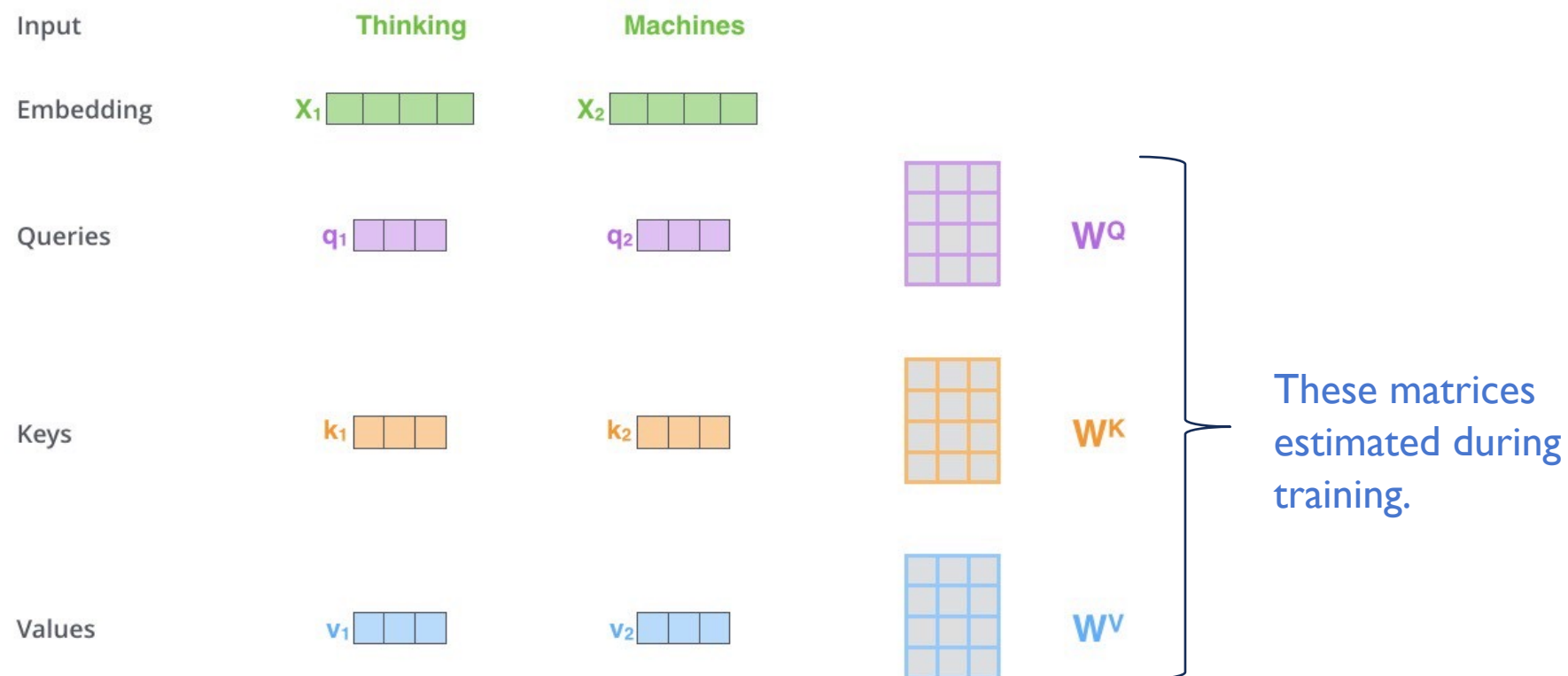
HOW TO CALCULATE SELF ATTENTION – STEP 2
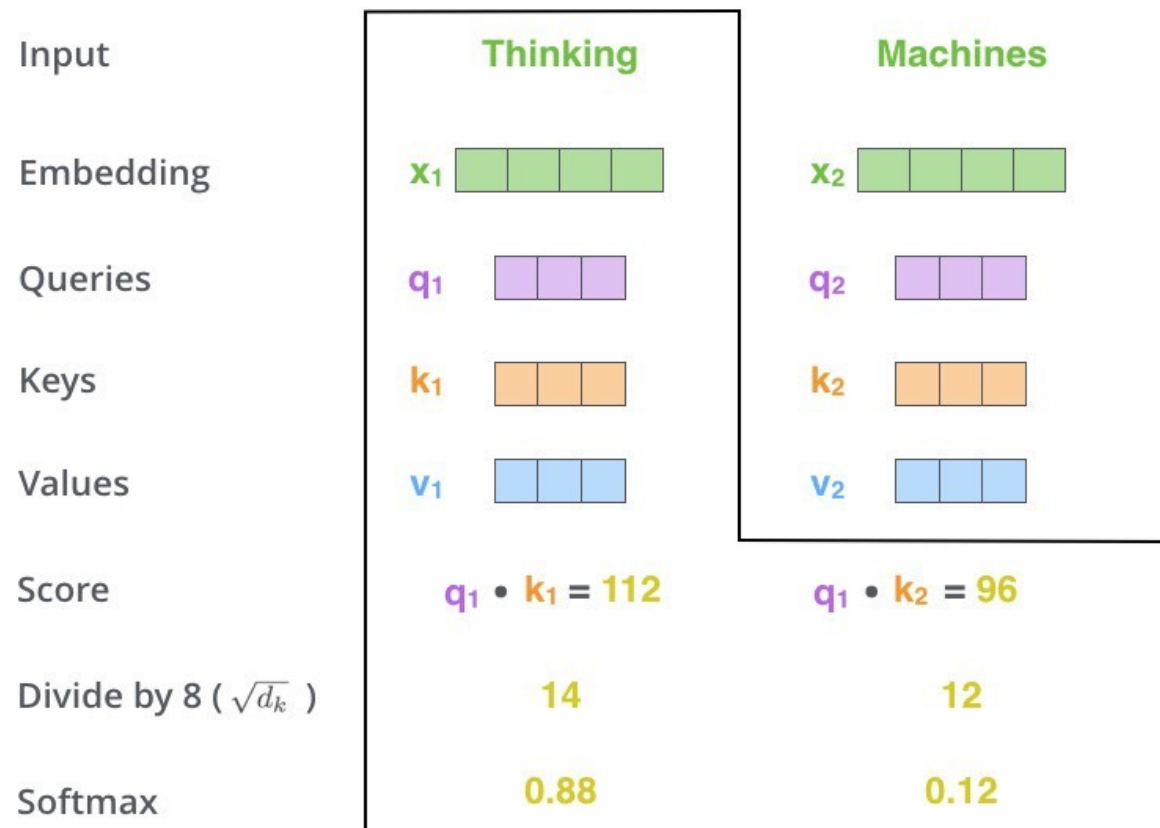
calculate self-attention score

Step 3:

- divide the core by 8
- square root of the dimension of the key vector

Step 4:

- apply SoftMax operation to normalizes scores

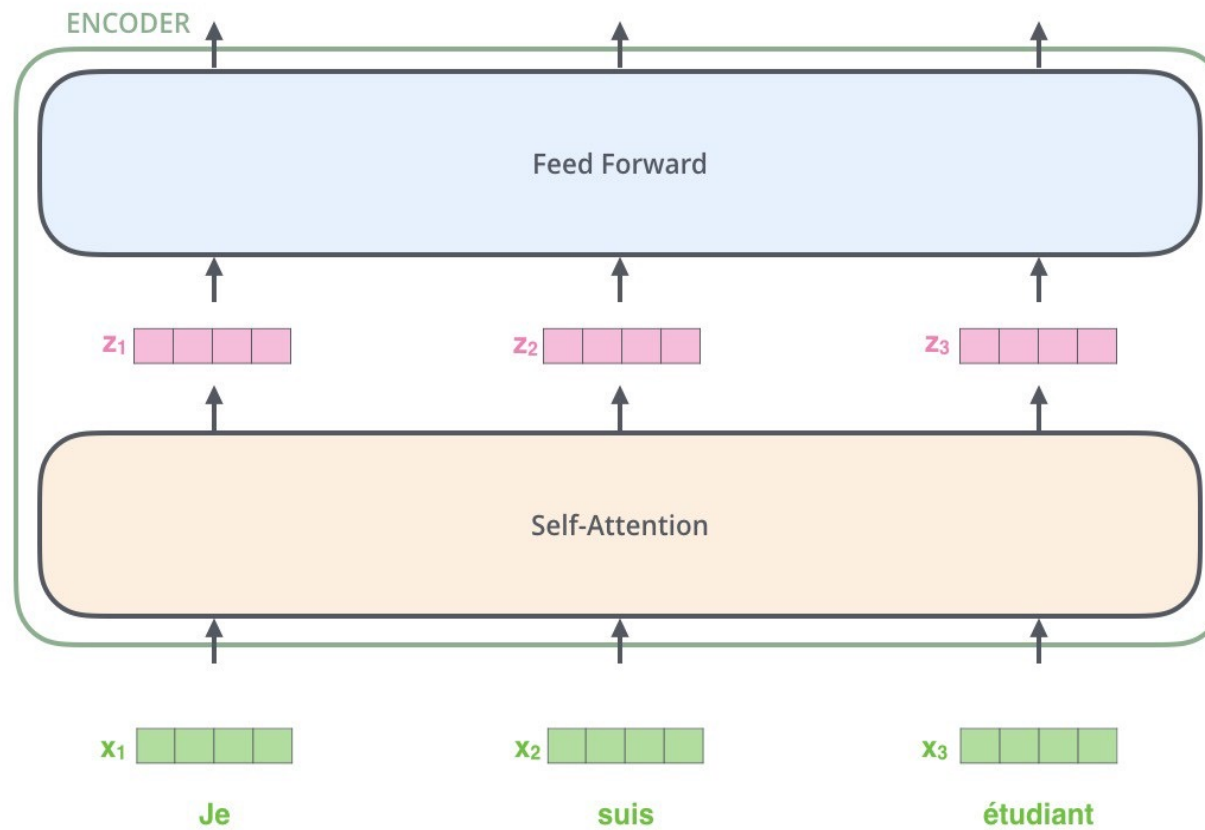| Input	Thinking	Machines
Embedding	x_1	x_2
Queries	q_1	q_2
Keys	k_1	k_2
Values	v_1	v_2
Score	$q_1 \cdot k_1 = 112$	$q_1 \cdot k_2 = 96$
Divide by 8 ($\sqrt{d_k}$)	14	12
Softmax	0.88	0.12

HOW TO CALCULATE SELF ATTENTION – STEP 5 & 6

- **Step 5:**
  - multiply each value vector by the SoftMax score
    - **Intuition:** to amplify relevant words and down grade irrelevant words

- **Step 6:**
  - sum up the weighted value vectors.
    - produces the output of the self- attention layer

	Thinking	Machines
Input		
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

13
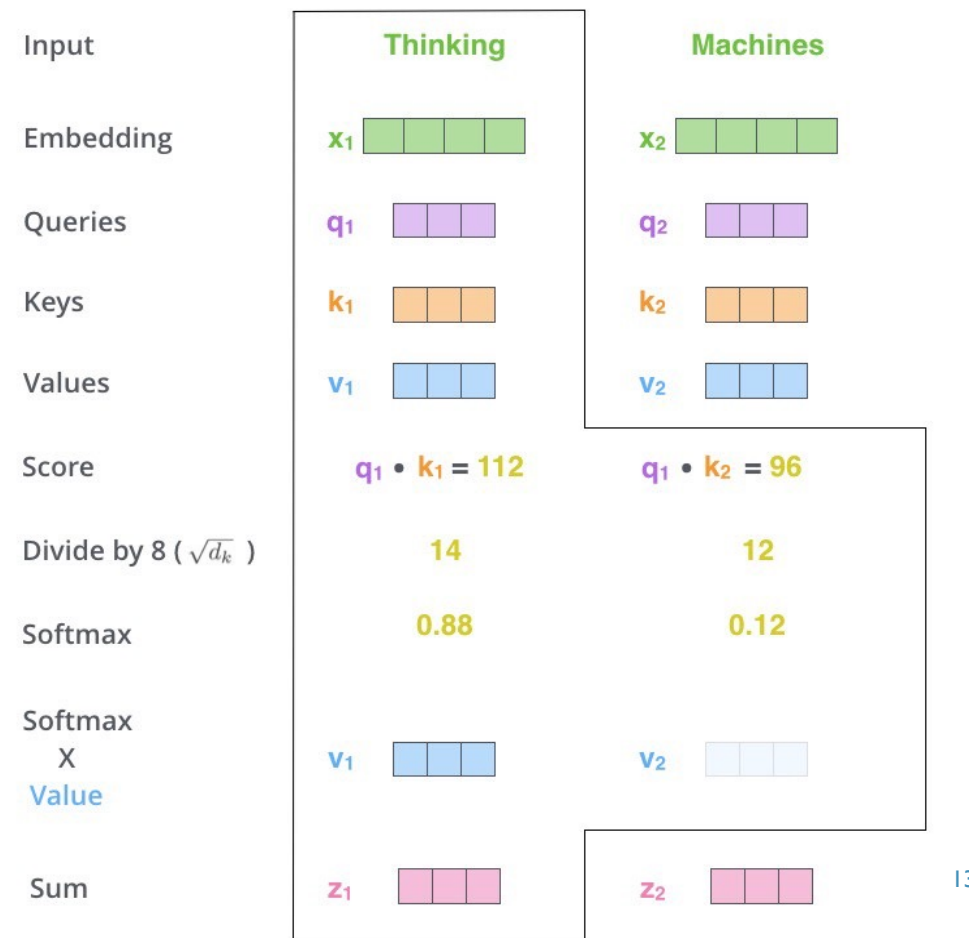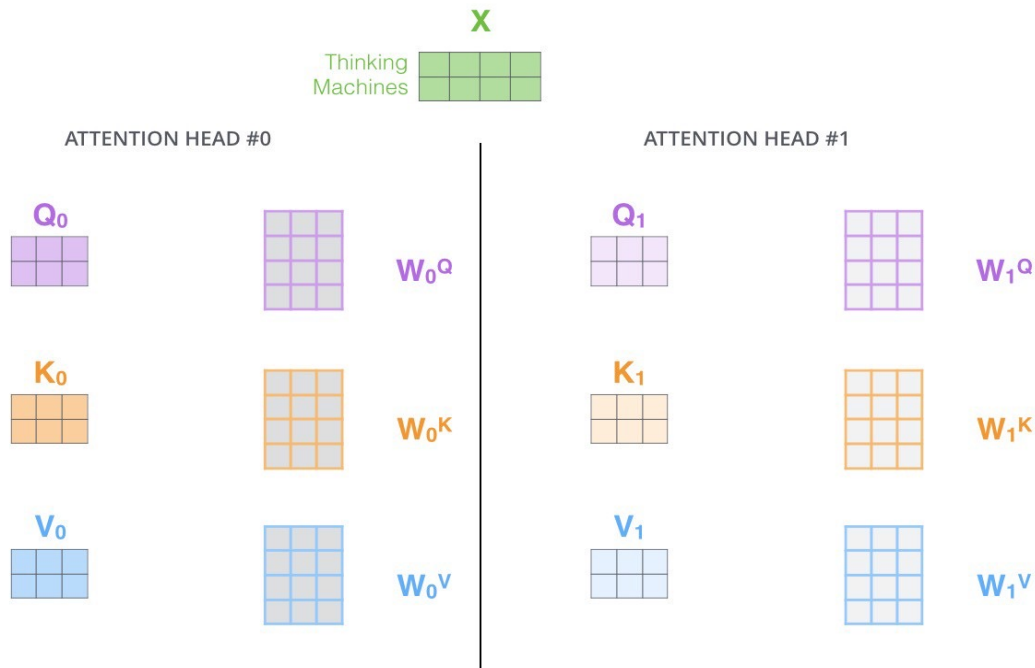
HOW TO CALCULATE SELF ATTENTION

Multi-headed attention improves the performance of the attention layer by allowing for:

1. It expands the model's ability to focus on different positions.

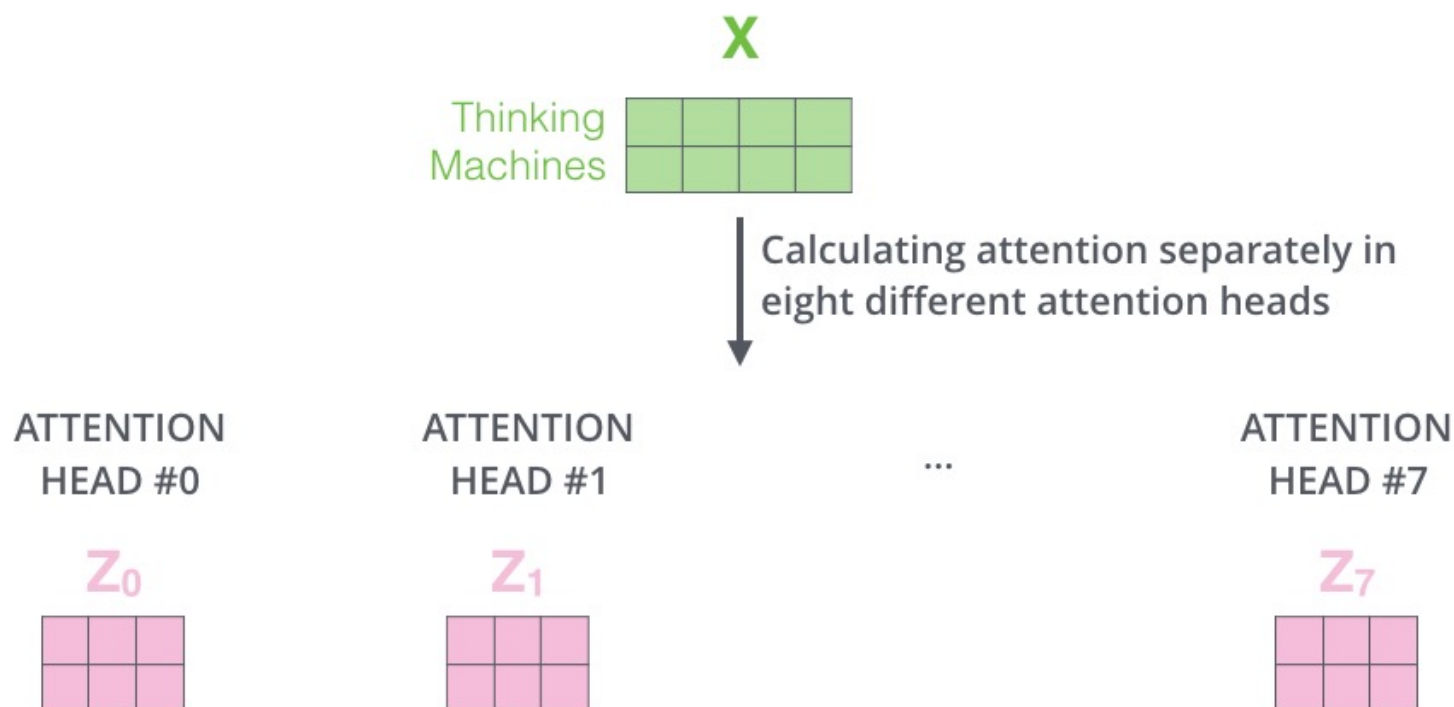2. It gives the attention layer multiple "representation subspaces".

MULTI-HEADED ATTENTION

Assume we have multi-headed attention with 8 self attention (eight Z matrices)

MULTI-HEADED ATTENTION

Assume we have multi-headed attention with 8 self attention (eight Z matrices)

- the feedforward layer expects a single matrix

- We concatenate the matrices then multiply them by an additional weight matrix W^O

1) Concatenate all the attention heads

Z_0 Z_1 Z_2 Z_3 Z_4 Z_5 Z_6 Z_7

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

X

W^O

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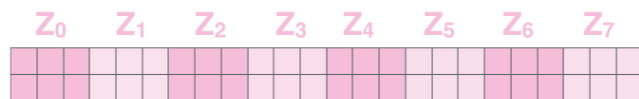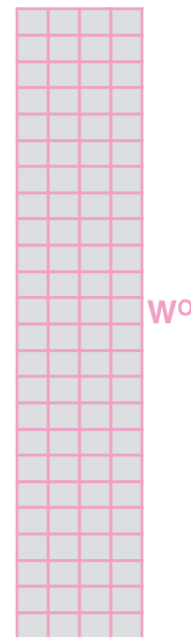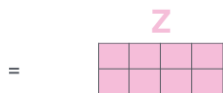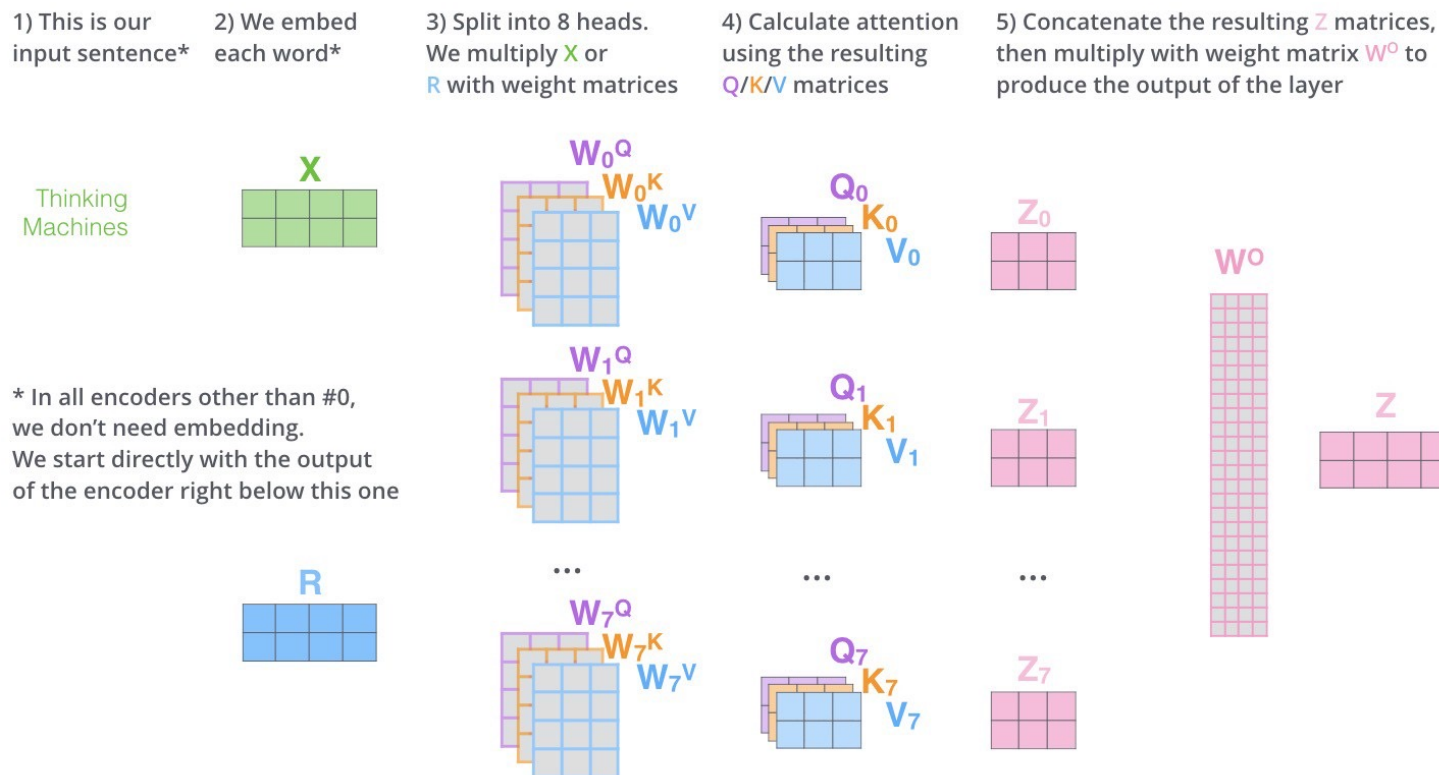
Z

=

MULTI-HEADED ATTENTION

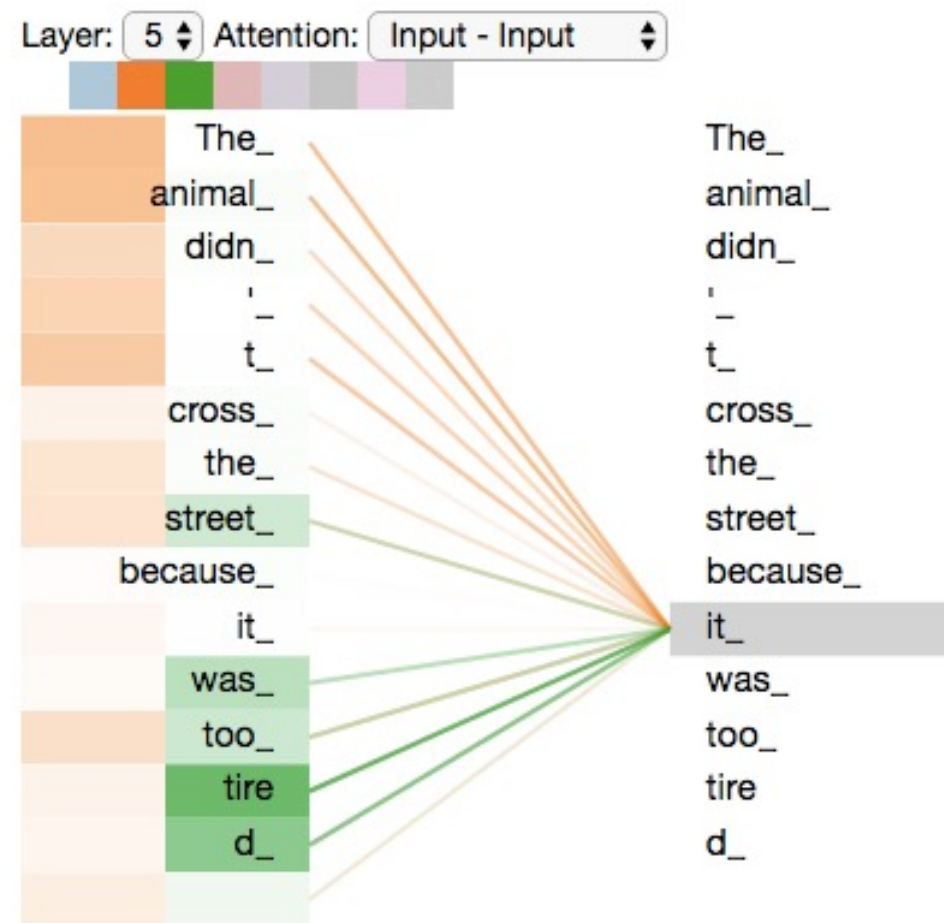Assume we have multi-headed attention with 8 self attention (eight Z matrices)

- the feedforward layer expects a single matrix

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

Thinking Machines

X

W_0^Q
W_0^K
W_0^V

Q_0
K_0
V_0

Z_0

W^O

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

W_1^Q
W_1^K
W_1^V

Q_1
K_1
V_1

Z_1

Z

R

...

W_7^Q
W_7^K
W_7^V

...

Q_7
K_7
V_7

...

Z_7

MULTI-HEADED ATTENTION

Multi head attention "*it*"

- one attention head is focusing on most on *the animal*
- another is focusing on *tired*

https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb

MULTI-HEADED ATTENTION

Multi head attention "*it*"

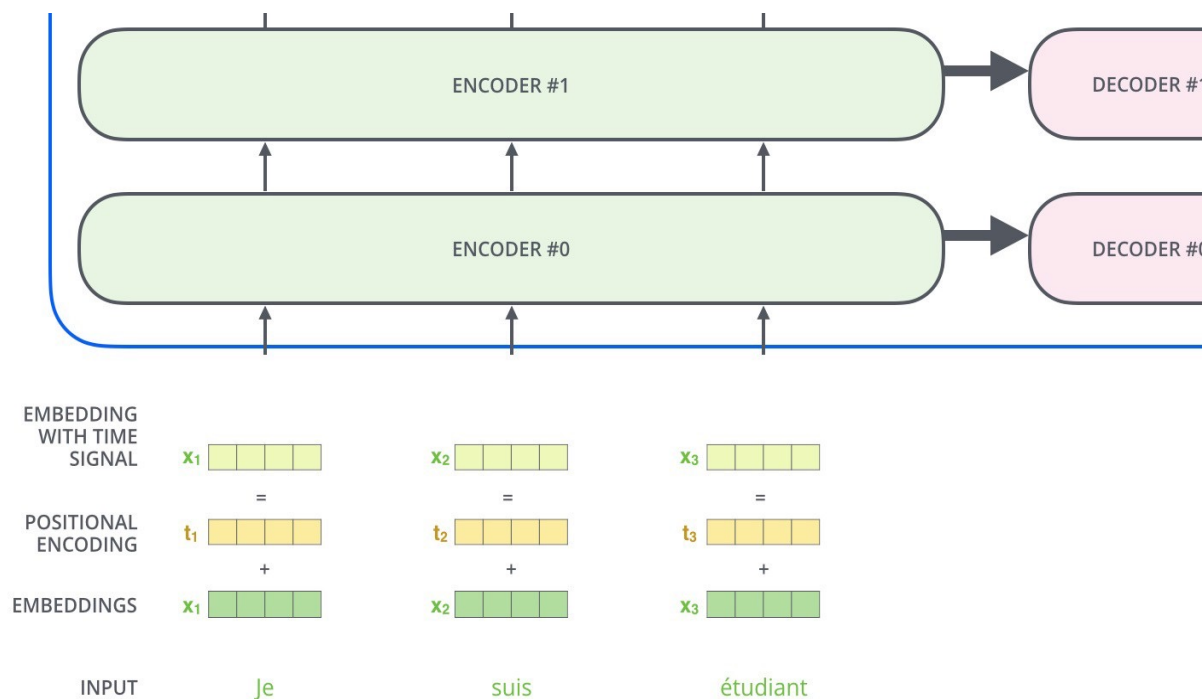- one attention head is focusing on most on *the animal*

- another is focusing on *tired*

- Visualizing all 8 attention heads
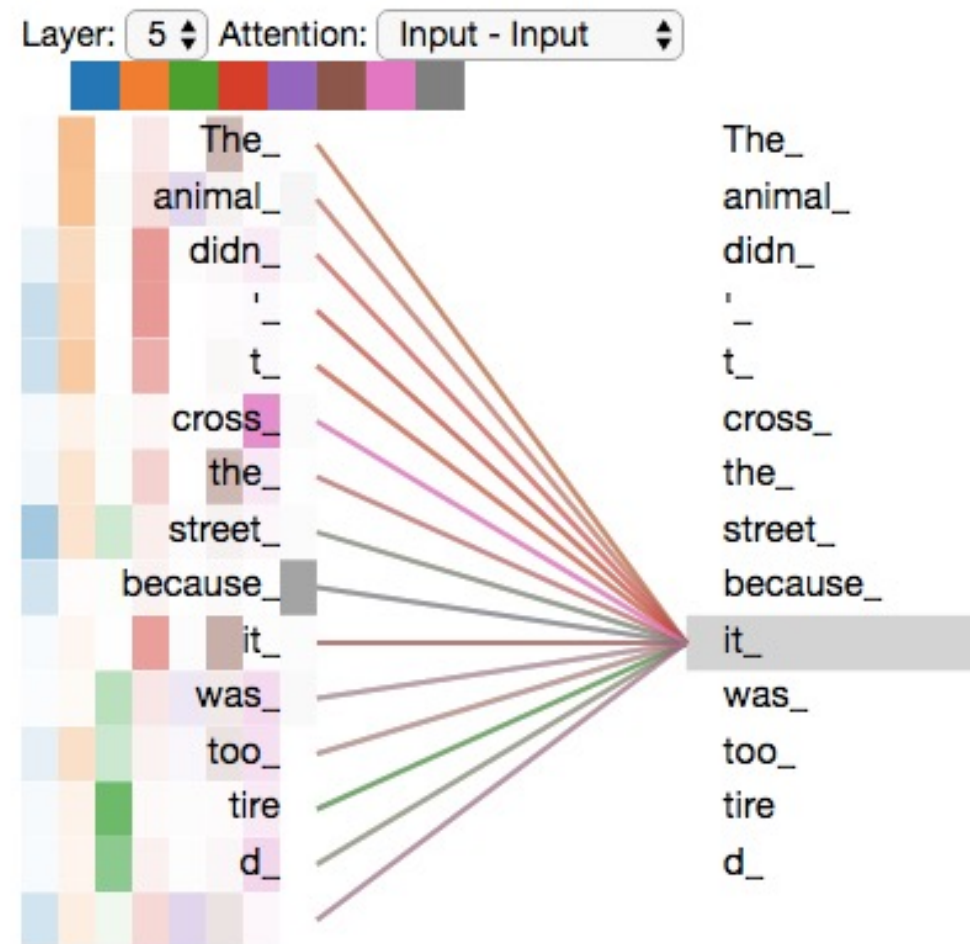
POSITIONAL ENCODING IN TRANSFORMERS

How do we account for the order of the words?

- Transformer adds a vector to each input embedding
- These vectors allow it to determine the position of each word

POSITIONAL ENCODING IN TRANSFORMERS

- **Intuition:**
  - adding these values to the embeddings provides meaningful distances between the embedding vectors once they are projected into Q/K/V vectors and during dot-product attention.
  - These positional encoding are generated during training (detail in the paper)
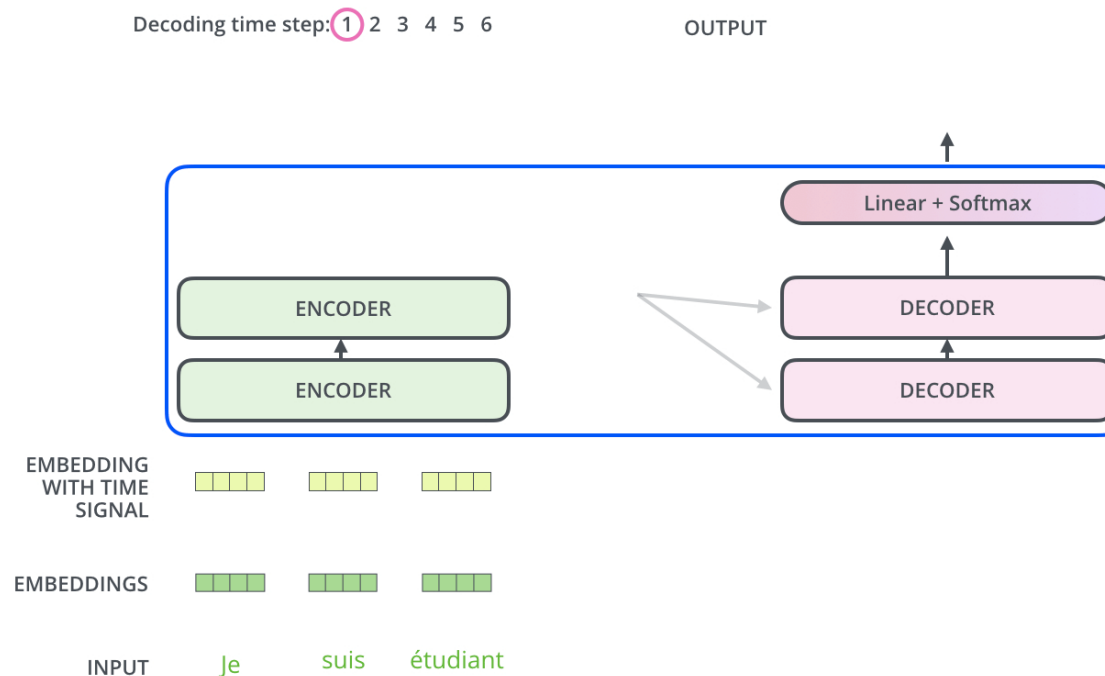- Assume encoding of dimension 4, the positioning will look like
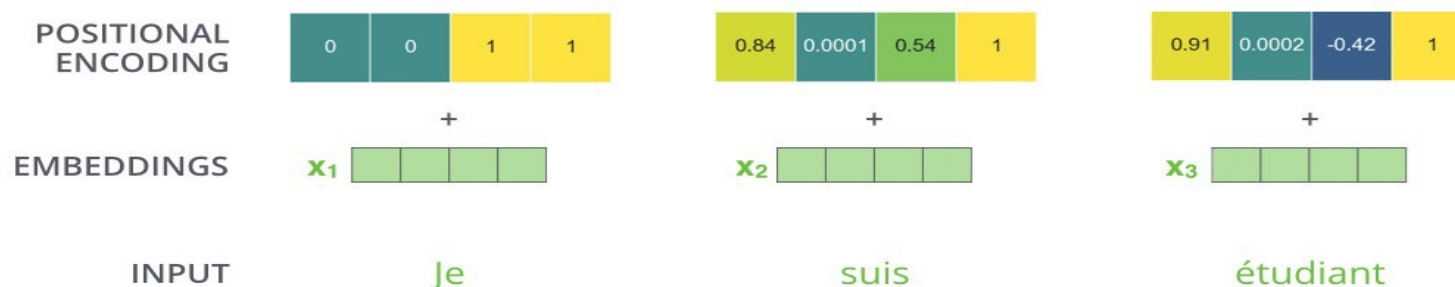
RESIDUAL CONNECTION

Residual connection: Each sub-layer (self-attention, feed forward neural network) in each encoder and decoder have a residual connection around it followed by a layer normalization

https://jalammar.github.io/illustrated-transformer/
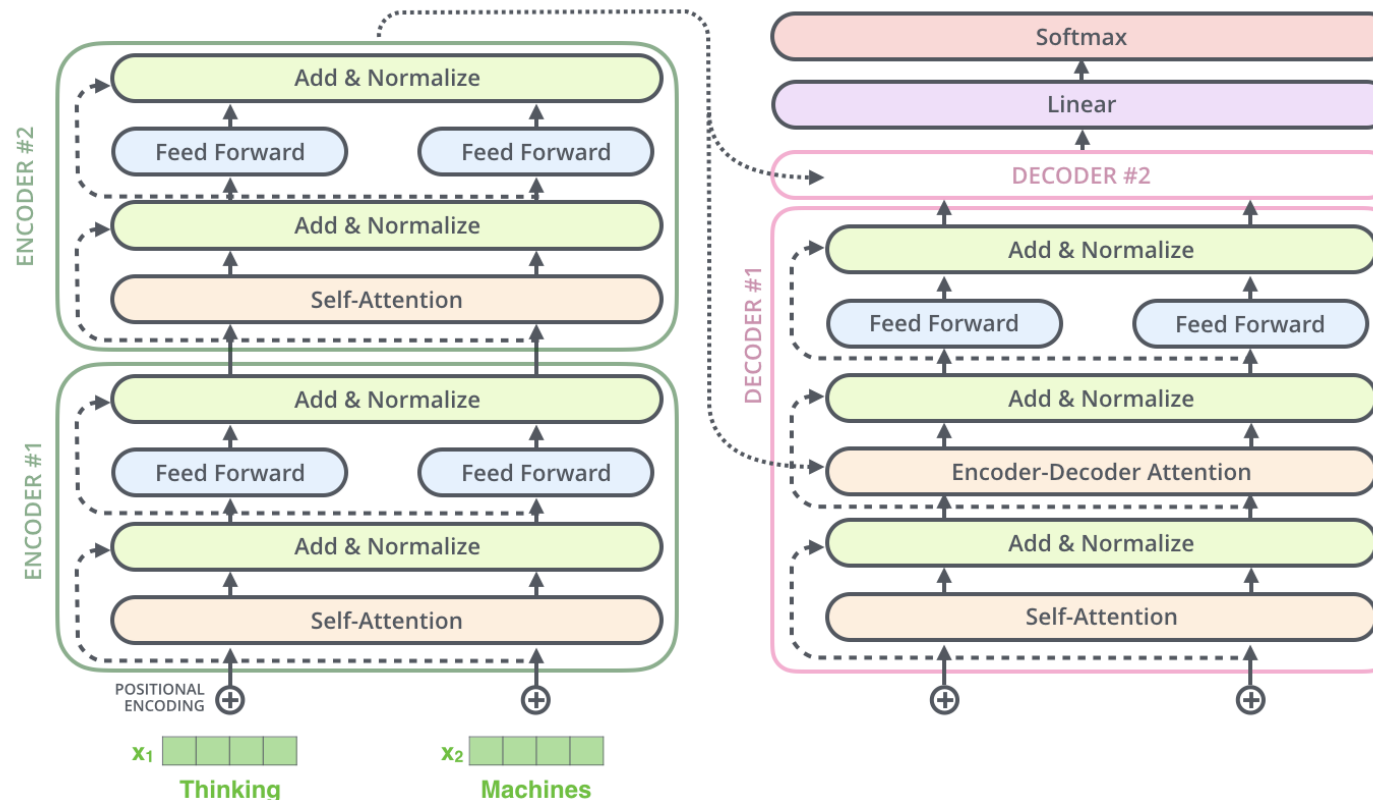
THE DECODER SIDE

- The encoder process the input sentence
- Output of top encoder layer is converted to attention vectors K and V.
- K & V vectors are used by each decoder in its "encoder-decoder attention" layer
- K & V helps decoder focus on appropriate words.

Decoding time step: (1) 2 3 4 5 6   OUTPUT

Linear + Softmax

ENCODER   DECODER

ENCODER   DECODER

EMBEDDING
WITH TIME
SIGNAL

EMBEDDINGS

INPUT   Je   suis   étudiant

24

THE DECODER SIDE

The following steps repeat the process until a special symbol is reached indicating the transformer decoder has completed its output
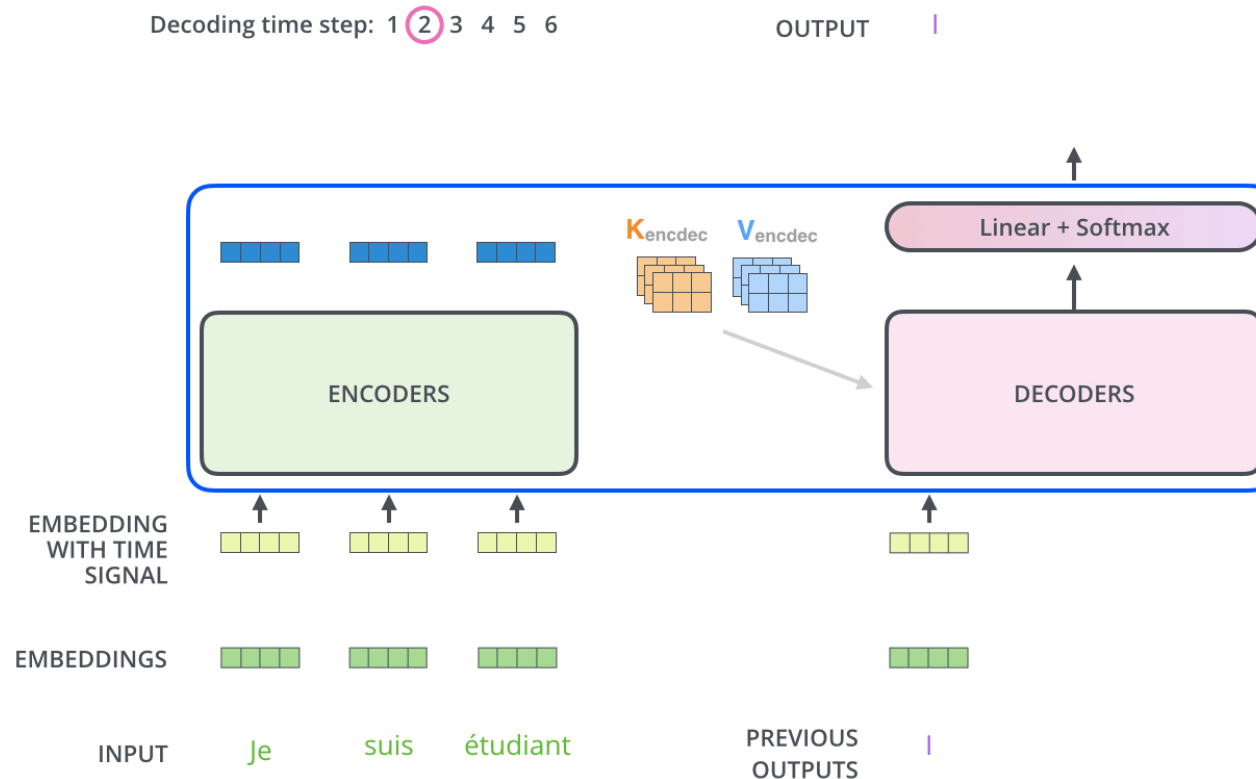
THE DECODER SIDE

The self attention layers is only allowed to focus on earlier positions in the output sequence.

- This is done by masking future positions (setting them to -inf) before the SoftMax step in the self-attention calculation.

The "Encoder-Decoder Attention" layer is like multiheaded self-attention

- except it creates its Queries matrix from the layer below it,
- takes the Keys and Values matrix from the output of the encoder stack.

FINAL LINEAR AND SOFTMAX LAYER

Final Linear Layer:
- fully connected connected neural network that projects the vector produced by the decoders into a logits vector.

Assume our model was trained on a 1000-word vocabulary

- Then logit vector has dimension 1000 cells

- Each cell the score of a unique word

- The SoftMax layers turns these scores into probabilities and the word with the highest probability is chosen.
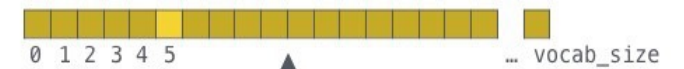
Which word in our vocabulary is associated with this index?

am

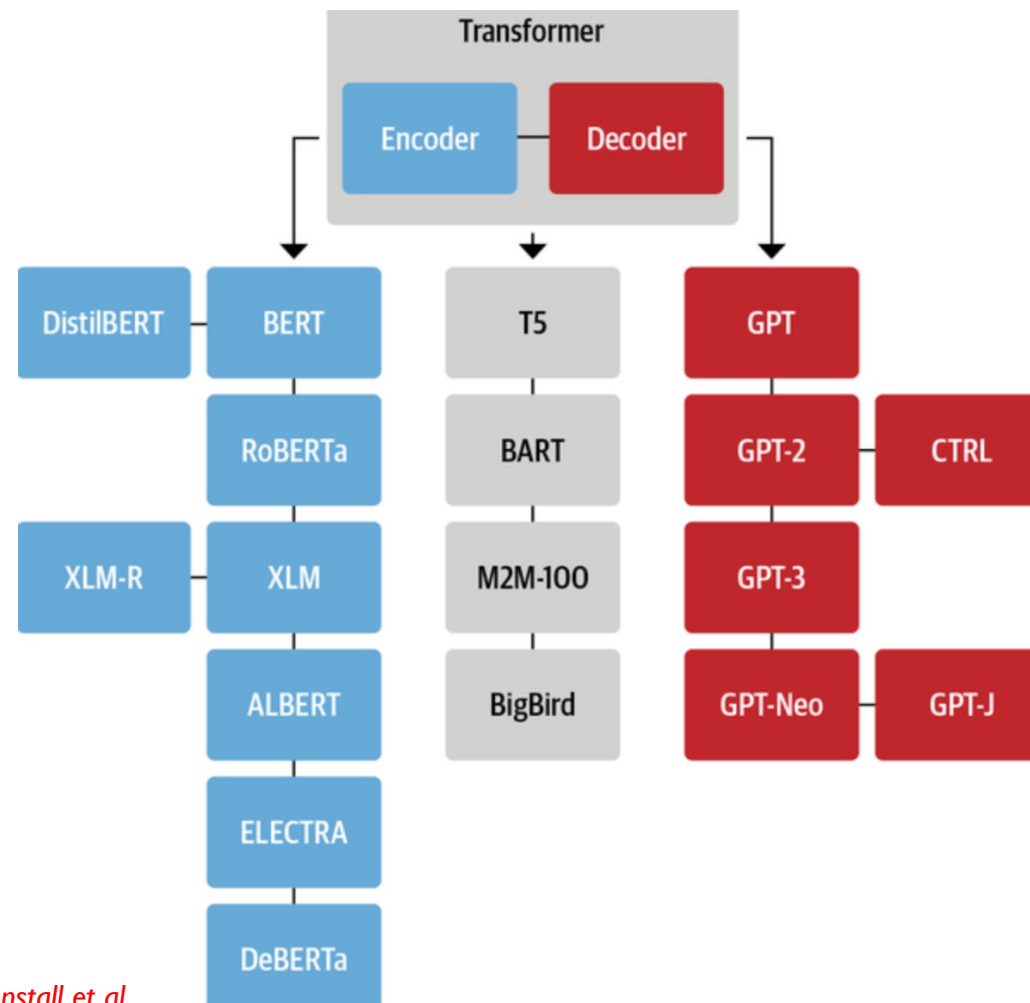Get the index of the cell with the highest value (argmax)

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

OVERVIEW OF MOST PROMINENT TRANSFORMERS

Natural language processing with transformers by Tunstall et al

Dual training of Masked language model and next sentence prediction.
Masked language model

https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270
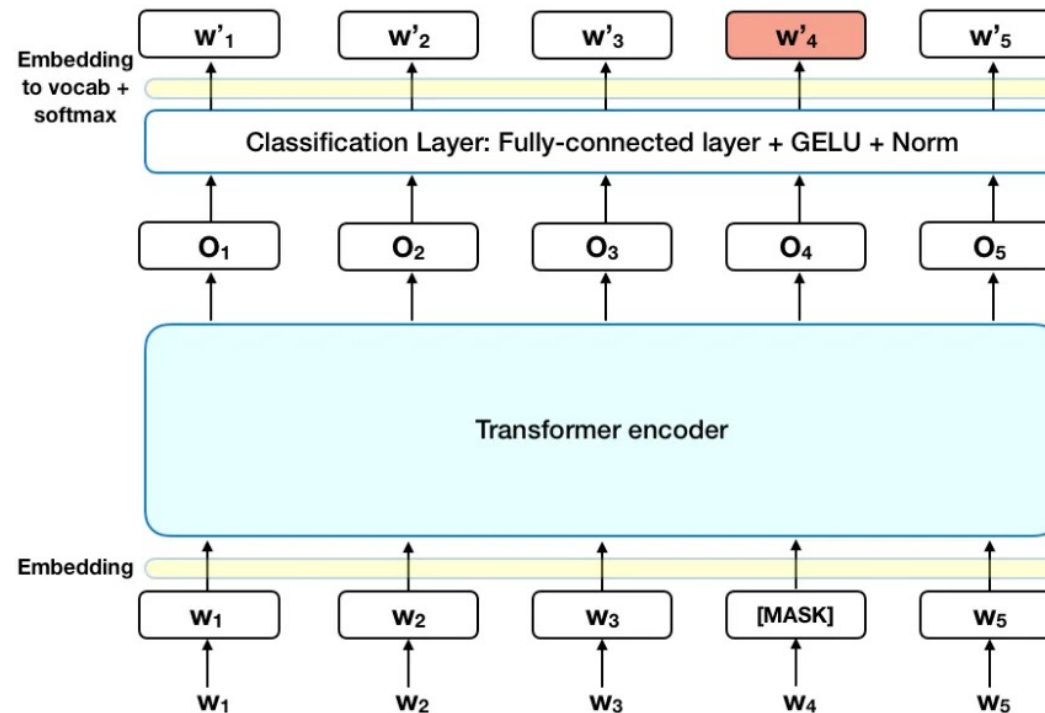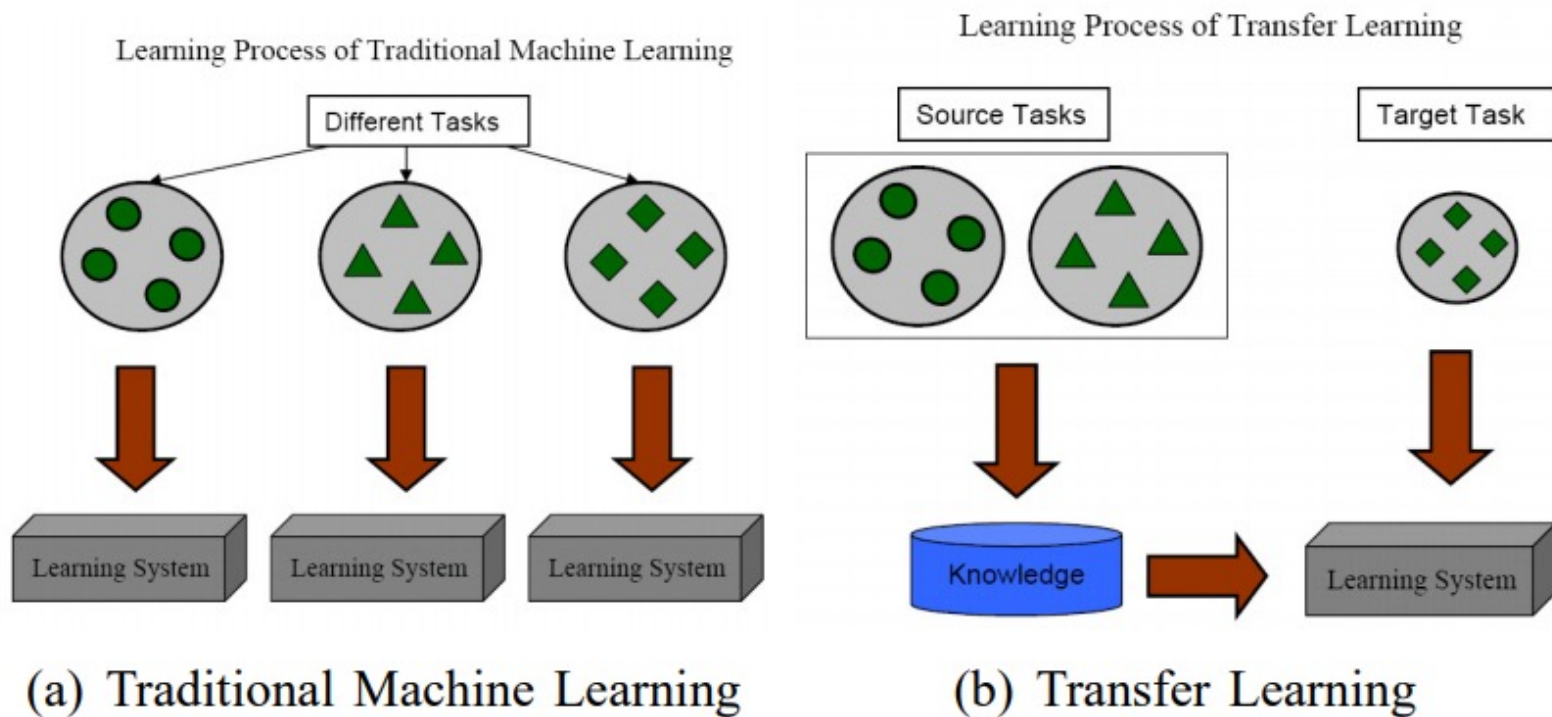
OVERVIEW OF BERT – NEXT SENTENCE PREDICTION
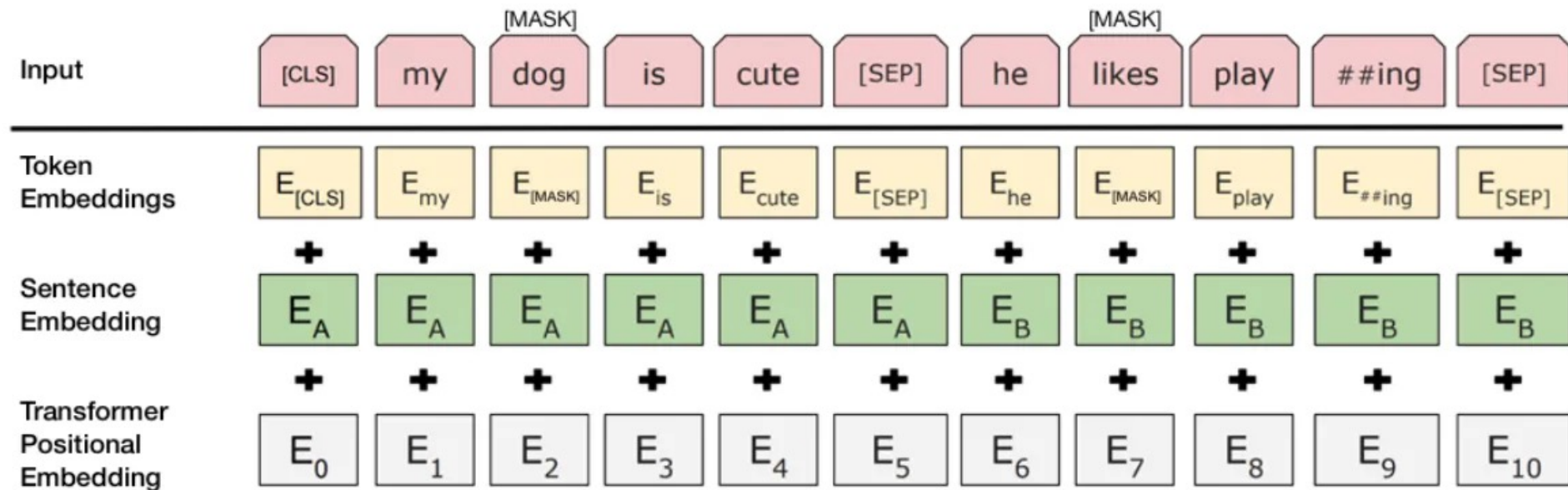
TRANSFER LEARNING

Transferring the knowledge of one model to perform a new task.

TRANSFER LEARNING

https://lisaong.github.io/mldds-courseware/03_TextImage/transfer-learning.slides.html
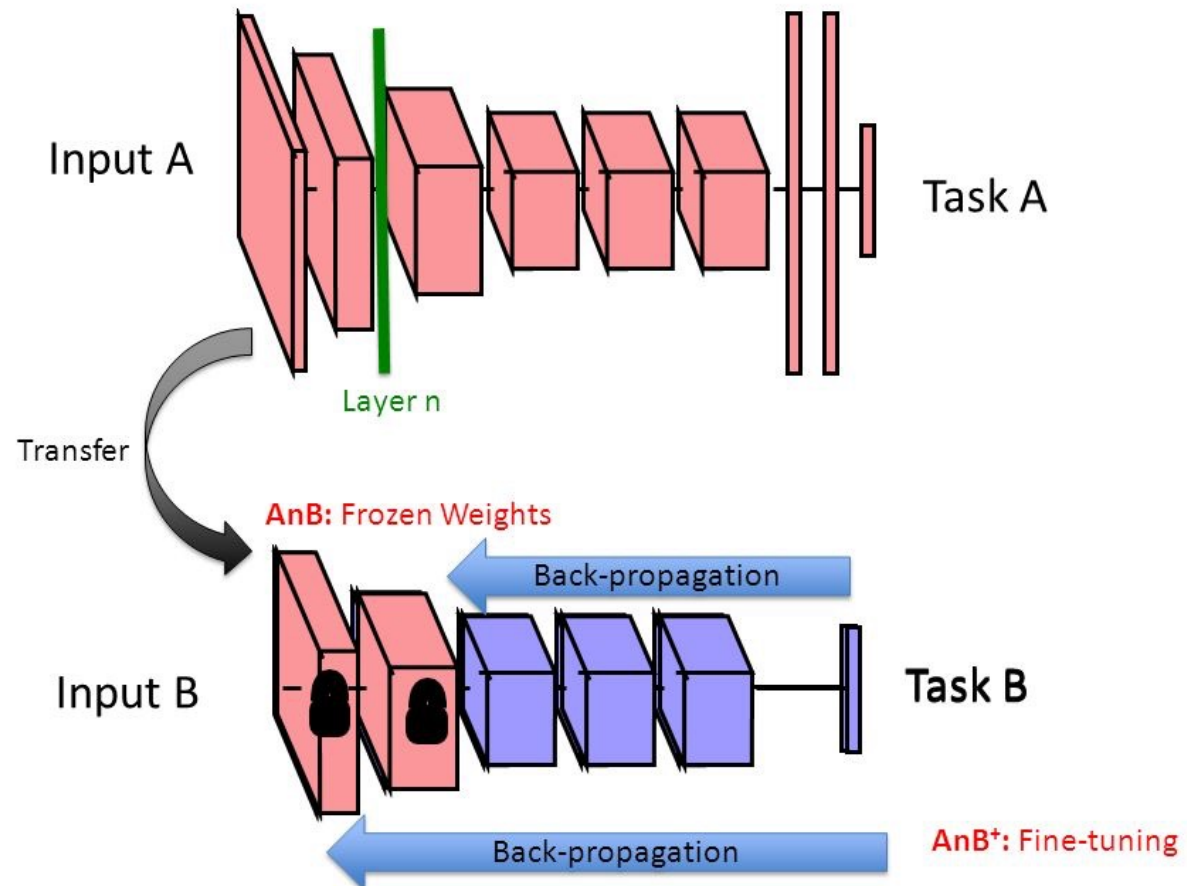
TRANSFER LEARNING

Diffferent types of transfer learning

Type	Description	Examples
Inductive	Adapt existing **supervised** training model on new **labeled** dataset	Classification, Regression
Transductive	Adapt existing **supervised** training model on new **unlabeled** dataset	Classification, Regression
Unsupervised	Adapt existing **unsupervised** training model on new **unlabeled** dataset	Clustering, Dimensionality Reduction

https://lisaong.github.io/mldds-courseware/03_TextImage/transfer-learning.slides.html

USEFUL LINKS

- Attention is all you need https://arxiv.org/abs/1706.03762
- BERT https://arxiv.org/pdf/1810.04805.pdf
- Transfer learning https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.147.9185&rep=rep1&type=pdf
- Hugging face
  - https://huggingface.co/transformers/index.html
  - https://github.com/huggingface/transformers
  - https://huggingface.co/docs/transformers/training
  - https://neptune.ai/blog/hugging-face-pre-trained-models-find-the-best