Transformer

Boonserm Kijsirikul

Dept. of Computer Engineering

Chulalongkorn University

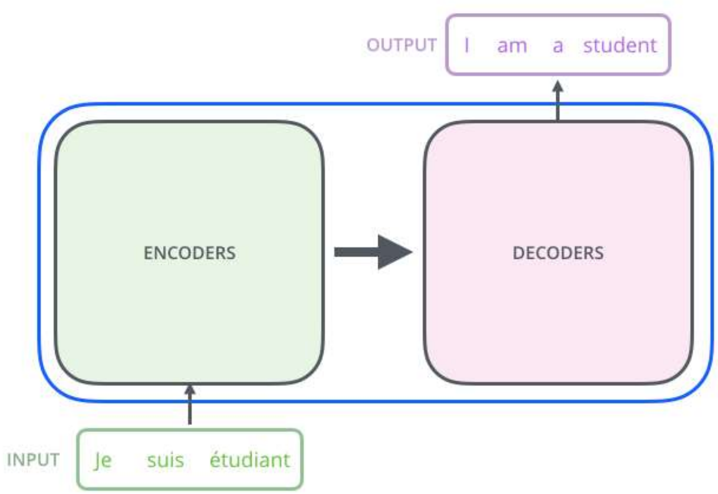
Features of Transformer

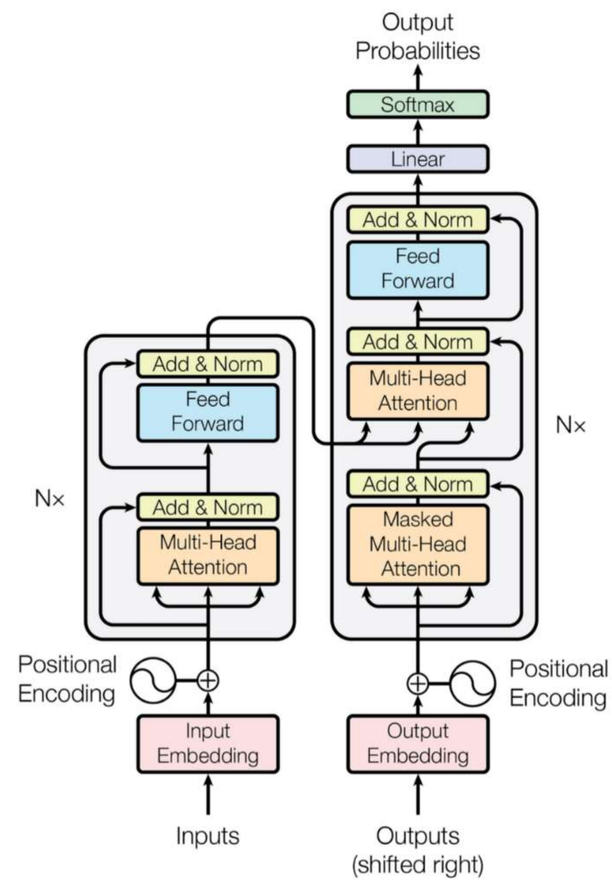
- Parallelization ability
- Short and long range dependencies
- Constant ‘path length’ between any two positions
- Multiple “heads” (multiple attention distributions and multiple outputs for a single input)
- Layer normalization and residual connections to make optimization easier

– Explicit position encodings

Transformer – Model Architecture

- As same as most neural sequence transduction models, Transformer have an encoder-decoder structure.

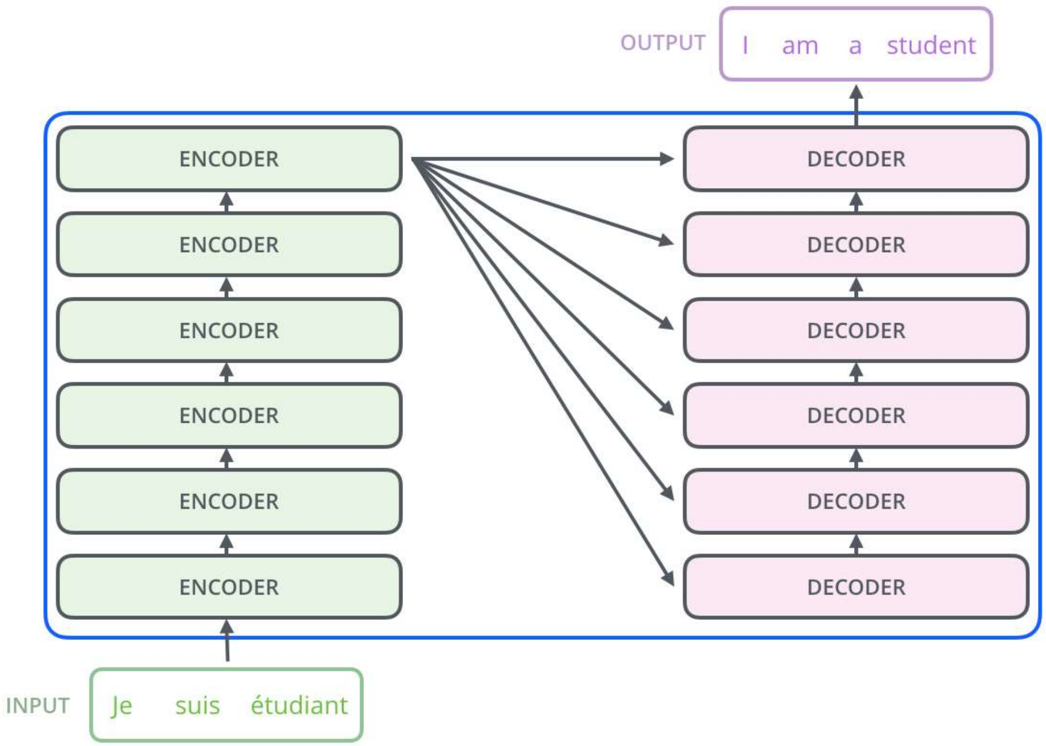




http://jalammar.github.io/illustrated-transformer/

Transformer – Model Architecture (cont.)

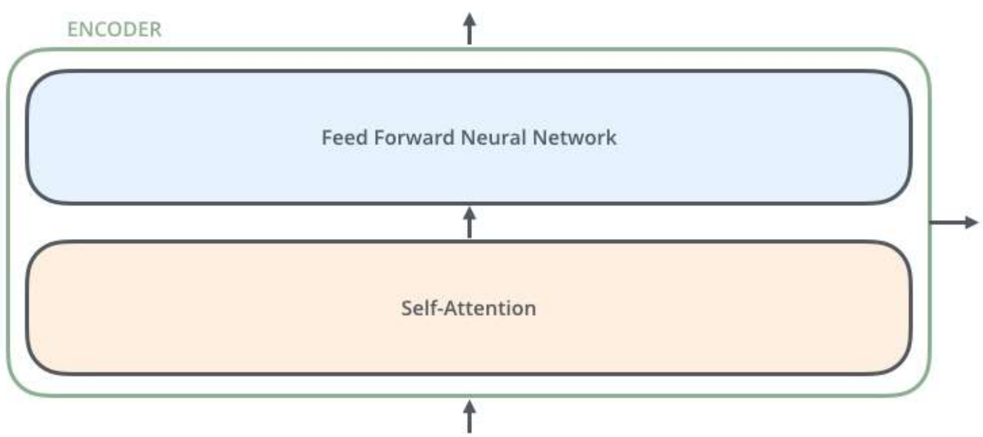
- The encoding component is a stack of encoders (6 encoders). The decoding component is a stack of decoders of the same number.

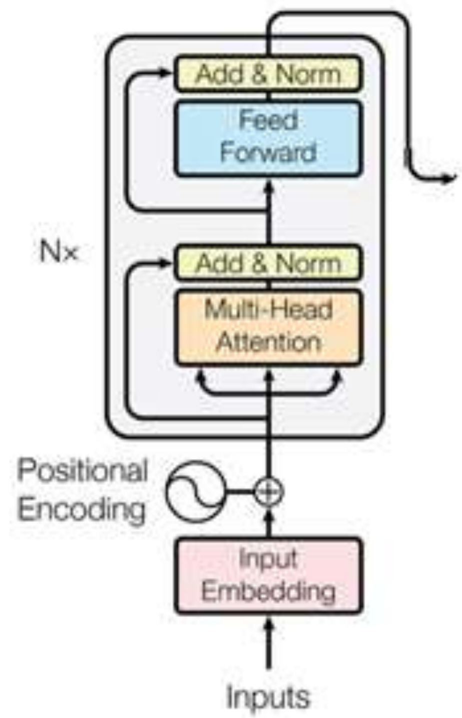


http://jalammar.github.io/illustrated-transformer/

Encoder

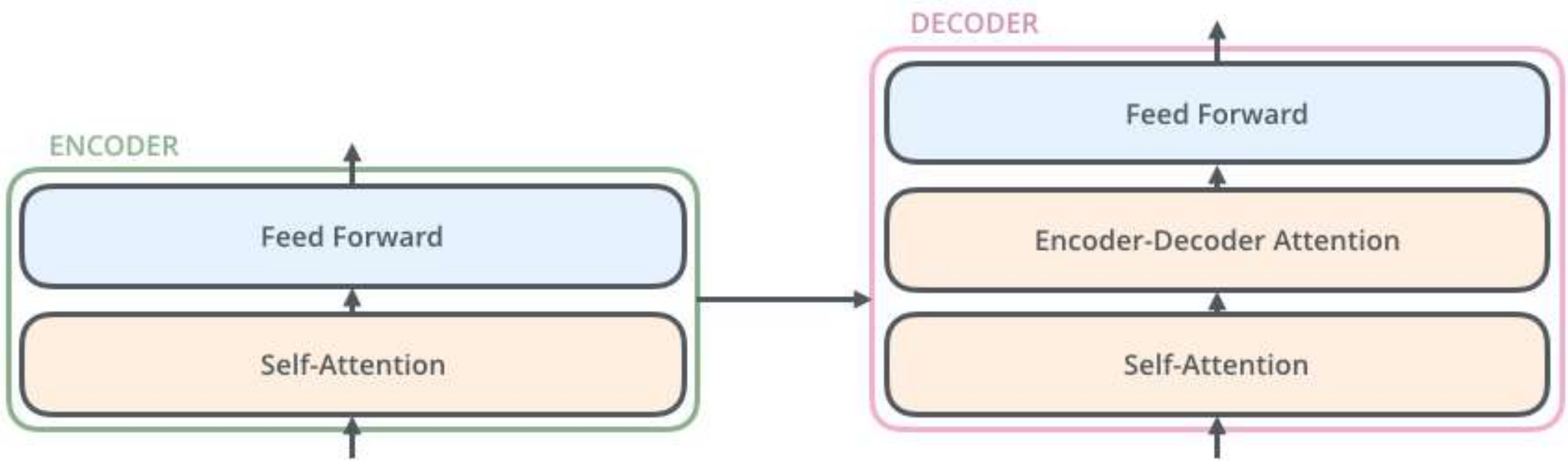
- The encoders all the same in structure (do not share weights). Each encoder is composed of two sub-layers: (1) Self-Attention (2) FNN





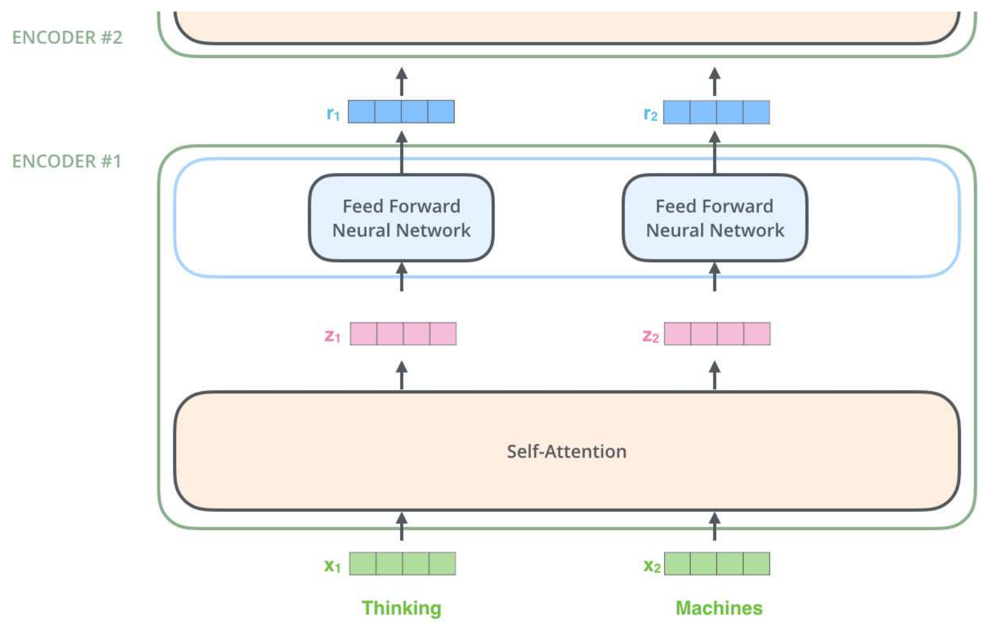
Decoder

- The decoder has (1) Self-Attention and (2) FNN layers, and additional (3) Encoder-Decoder Attention layer that helps the decoder focus on relevant parts of the input sentence.



Example

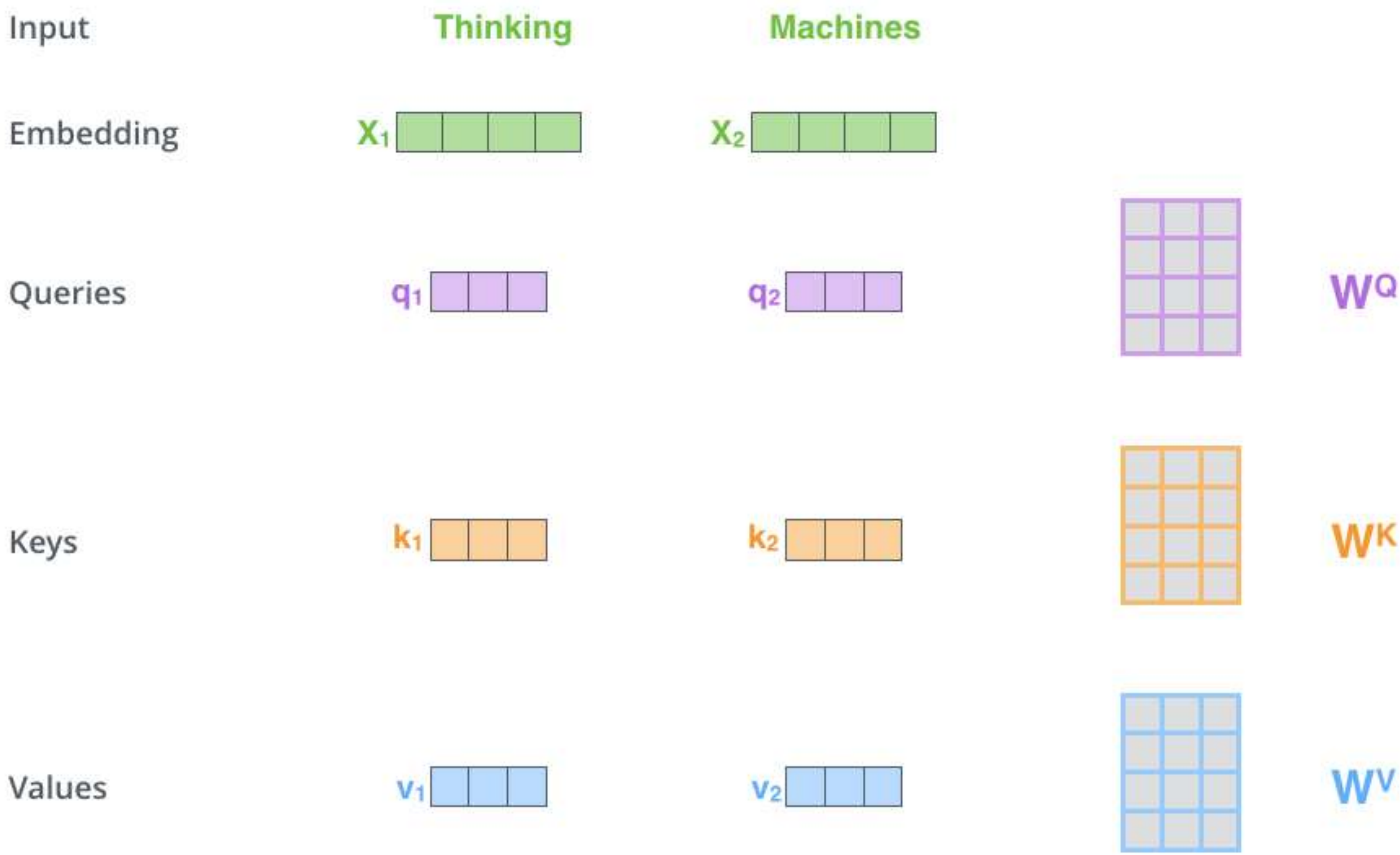
- Given an input text, Transformer turns each input word into a vector (of size 512) using an embedding algorithm.
- The embedding only happens in the bottom-most encoder (in other encoders, it would be the output of the encoder that is directly below).
- The size of the input list is hyperparameter we can set – basically it would be the length of the longest sentence in our training dataset.



Self-Attention

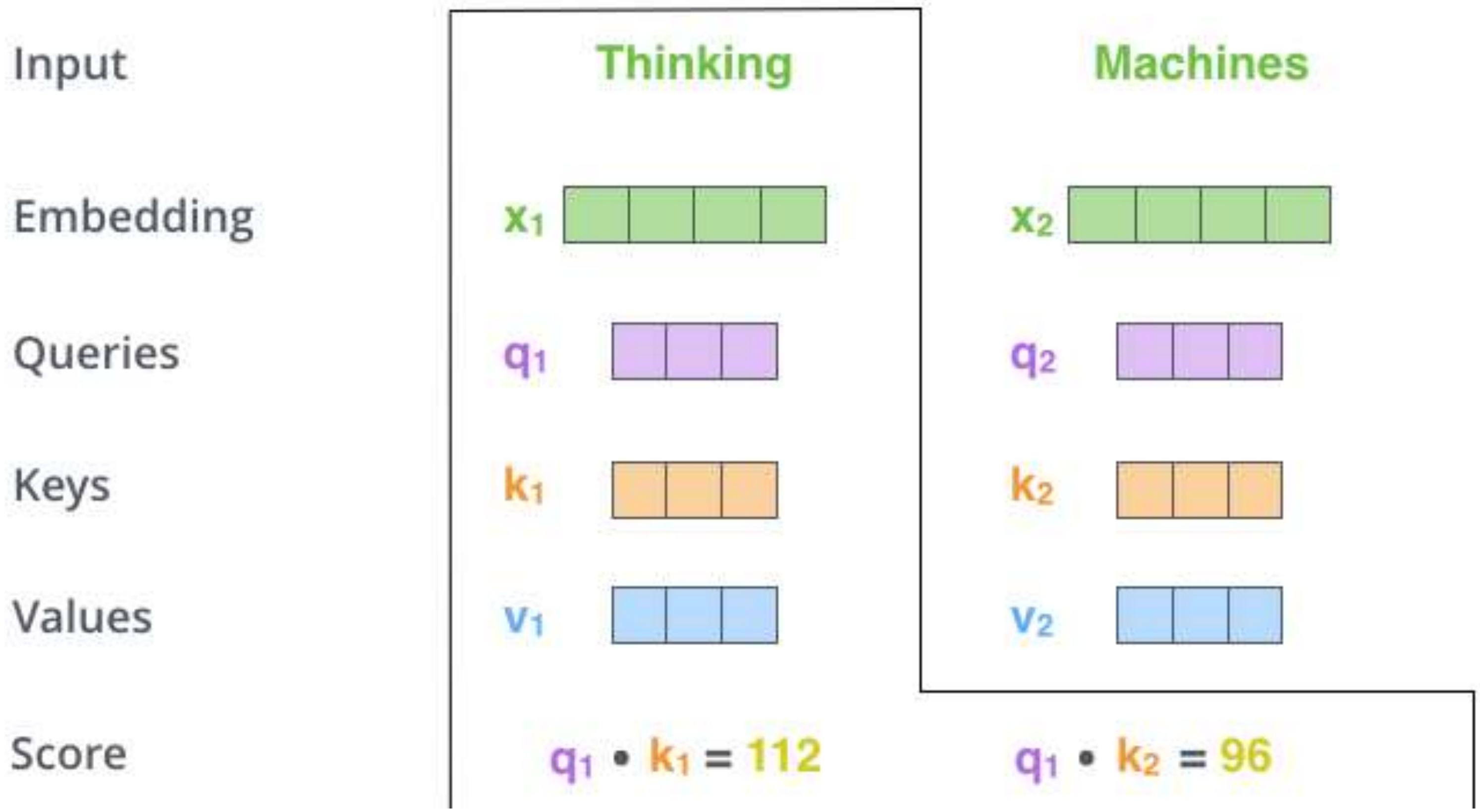
- The first step creates three vectors from each of the encoder's input vector (in this case, the embedding of each word):
  - Query vector,
  - Key vector, and
  - Value vector.
- These vectors are created by multiplying the embedding by three matrices that are trained during the training process.

Query, Key and Value Vectors



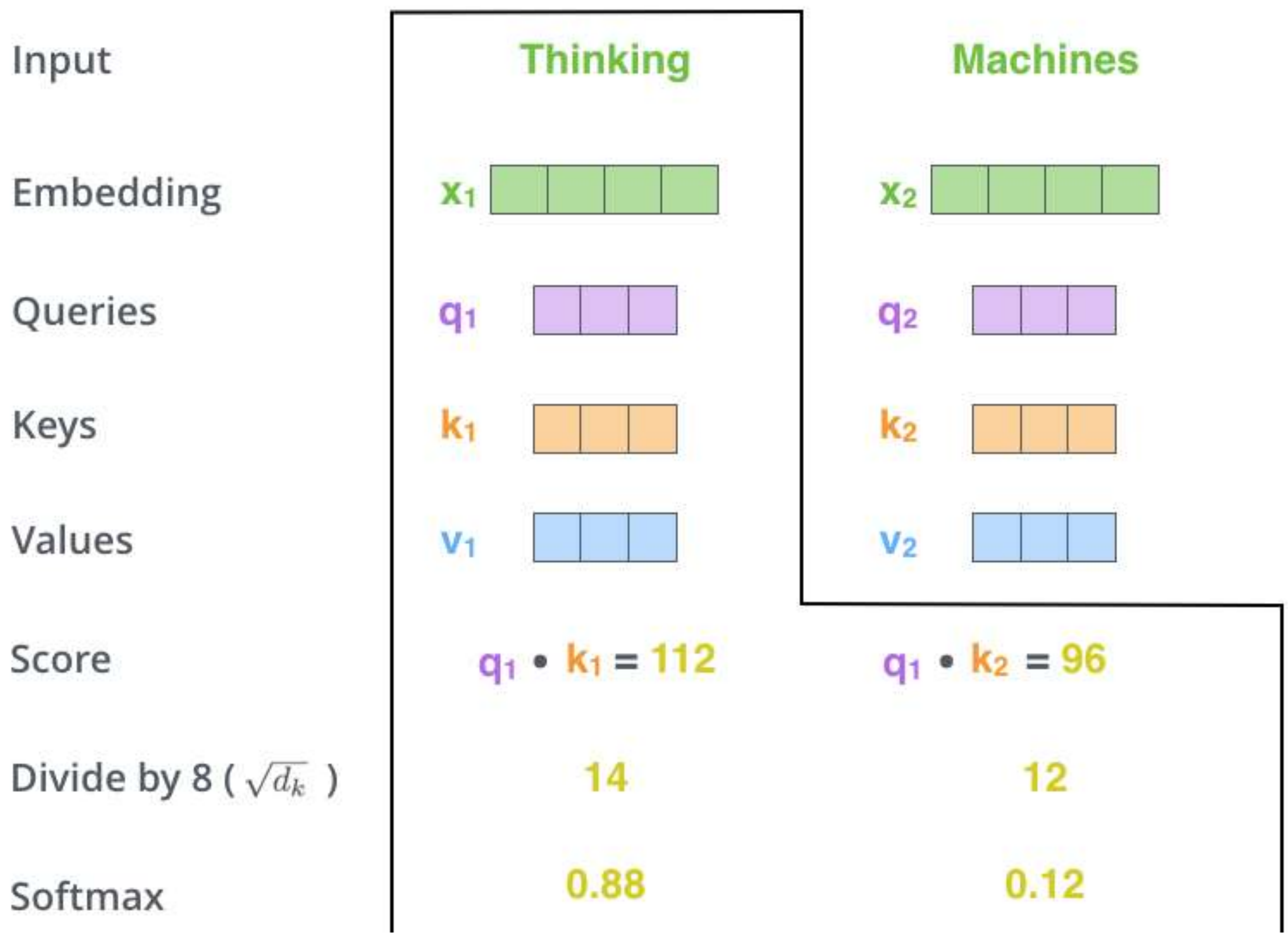
http://jalammar.github.io/illustrated-transformer/

- **The second step** calculates a score. To calculate the self-attention for the first word "Thinking", we need to score each word of the input sentence against this word.
- The score determines how much focus to place on other parts of the input sentence as we encode a word at a certain position.



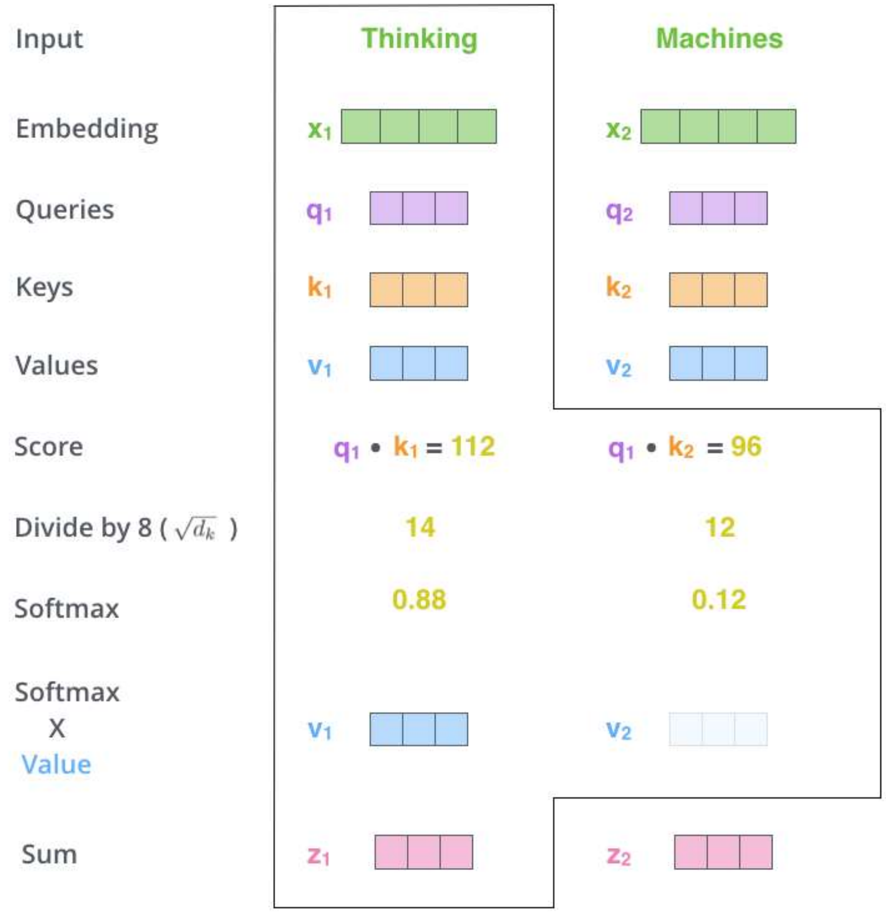
http://jalammar.github.io/illustrated-transformer/

- The **third and forth steps** are to divide the scores by 8 (the square root of the dimension of the key vectors used in the paper – 64. This leads to having more stable gradients.
- Softmax normalizes the scores so they are all positive and add up to 1.



http://jalammar.github.io/illustrated-transformer/

- The **fifth step** is to multiply each value vector by the softmax score (in preparation to sum them up).
- The **sixth step** is to sum up the weighted value vectors. This produces the output of the self-attention layer at this position (for the first word).



http://jalammar.github.io/illustrated-transformer/

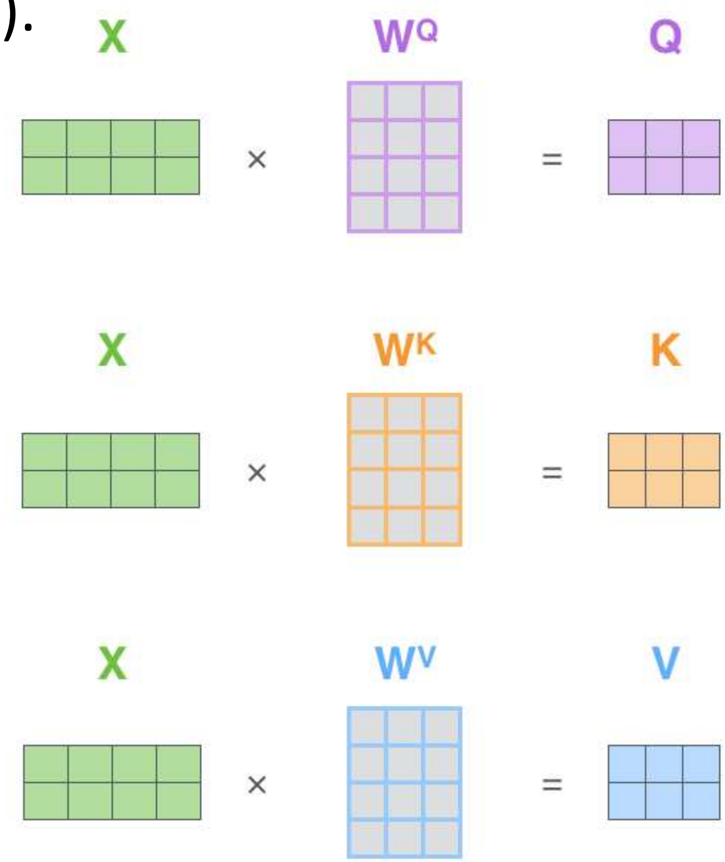
Attention - Three Inputs & One Output

- INPUT
  - Keys: A sequence of vectors also known as the memory. It is the contextual information that we want to look at. In traditional sequence-to-sequence learning they are usually the RNN encoder outputs.
  - Values: A sequence of vectors from which we aggregate the output through a weighted linear combination.
  - Query: A single vector that we use to probe the Keys. By probing we mean the Query is independently combined with each key to arrive at a single probability. The type of attention determines how the combination is done. Usually Query is the decoder RNN state at a given time step in traditional sequence-to-sequence learning
- OUTPUT

  A single vector which is derived from a linear combination of the Values using the probabilities from the previous step as weights.

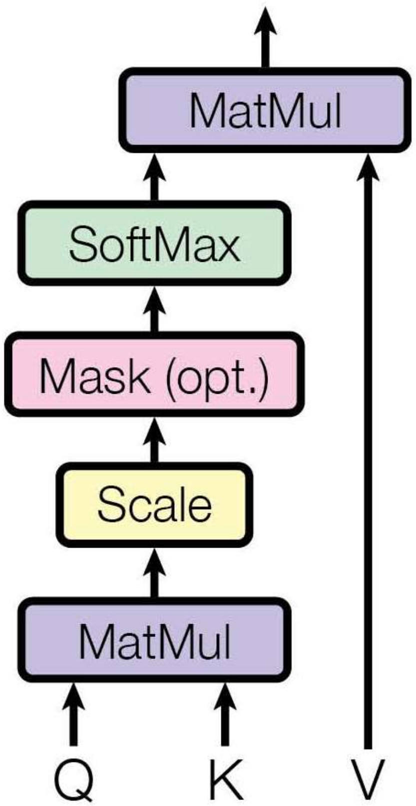
Matrix Calculation of Self-Attention

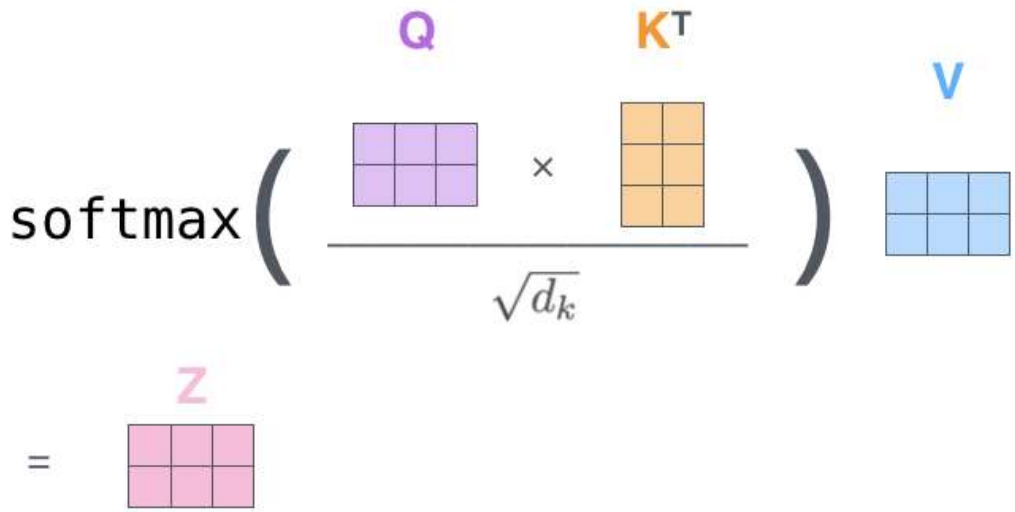
- As the first step, we calculate the Query, Key, and Value matrices, by packing our embeddings into a matrix X, and multiplying it by the weight matrices we have trained (WQ, WK, WV).



Matrix Calculation of Self-Attention (cont.)

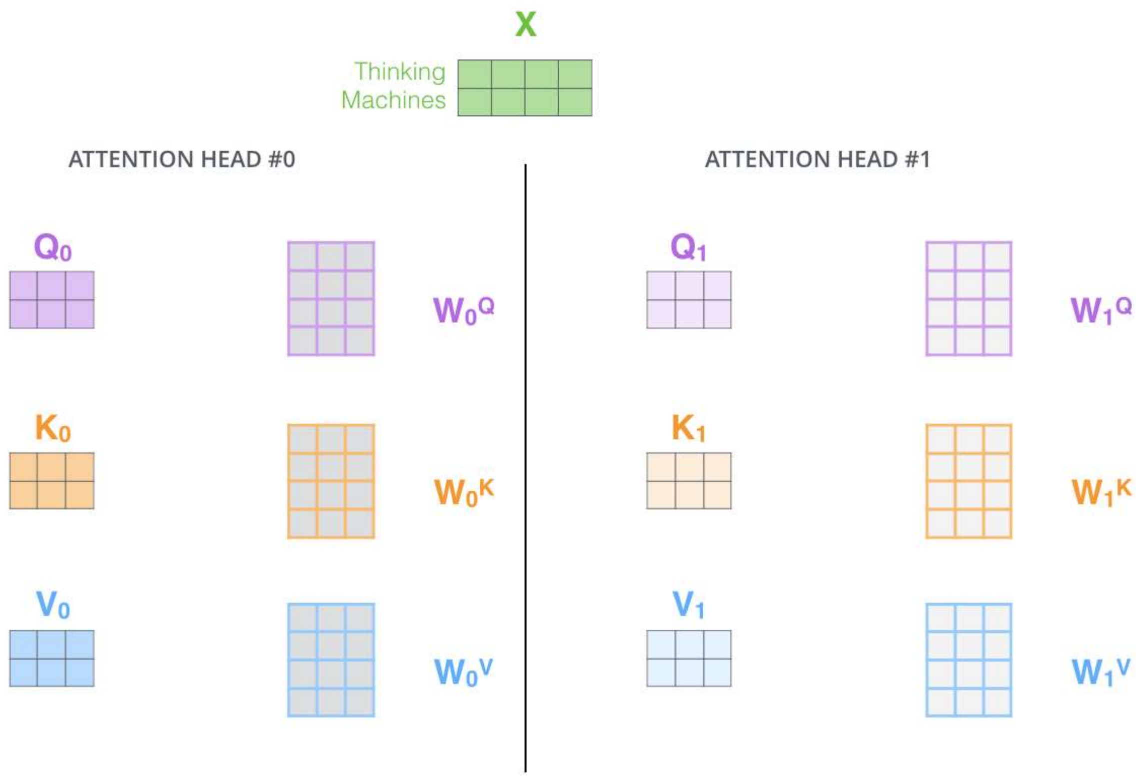
- We then condense steps two through six in one formula to calculate the outputs of the self-attention layer.



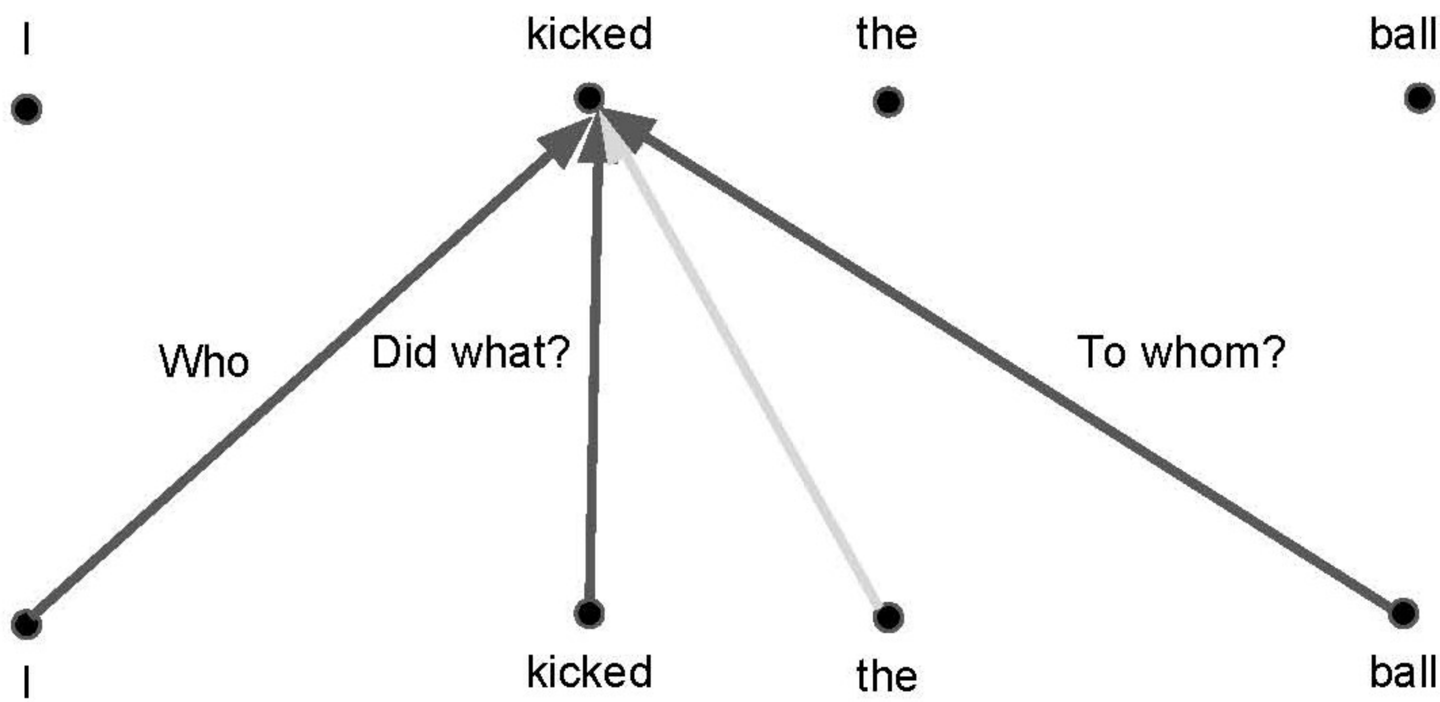


Multi-Headed Attention

- Multi-headed attention improves the performance of the attention layer in two ways:
  - It expands the model's ability to focus on different positions.
  - It gives the attention layer multiple "representation subspaces".

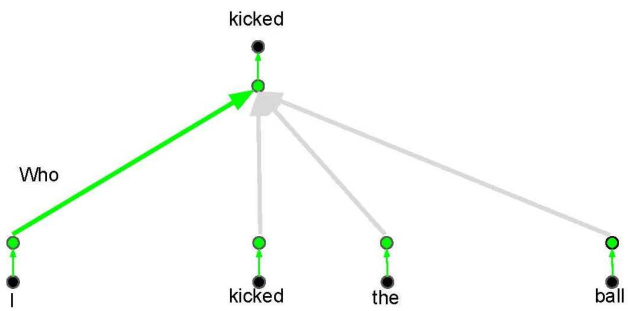


Self-Attention

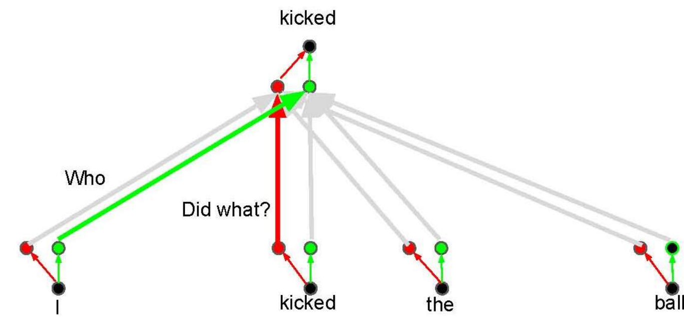


Self-Attention

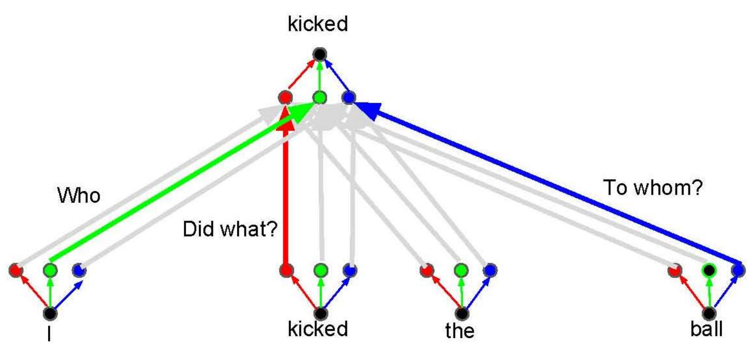
Attention head: Who



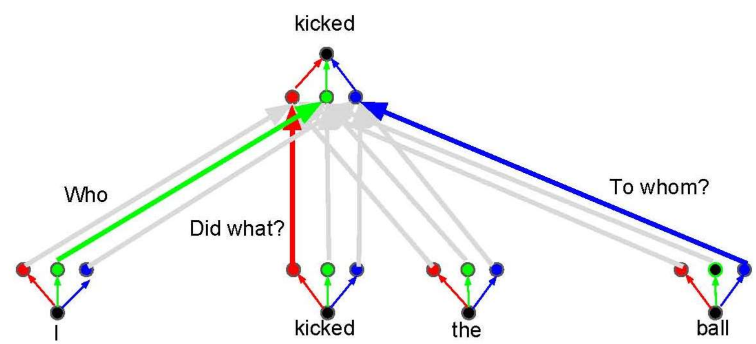
Attention head: Did What?



Attention head: To Whom?

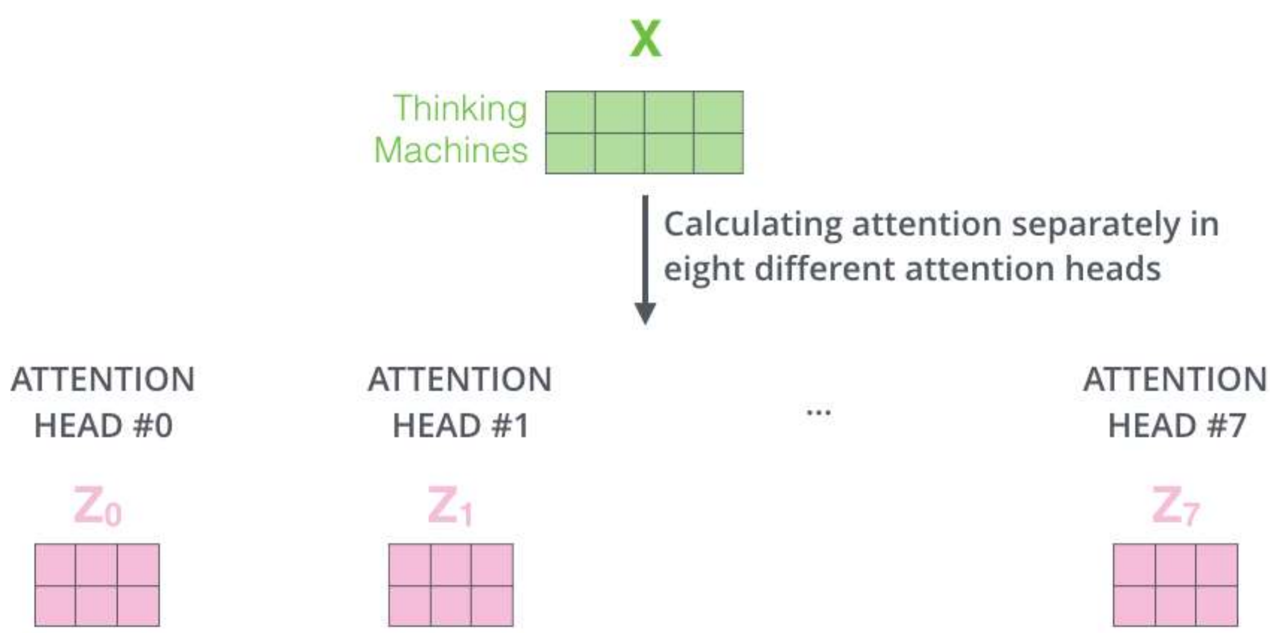


Multihead Attention



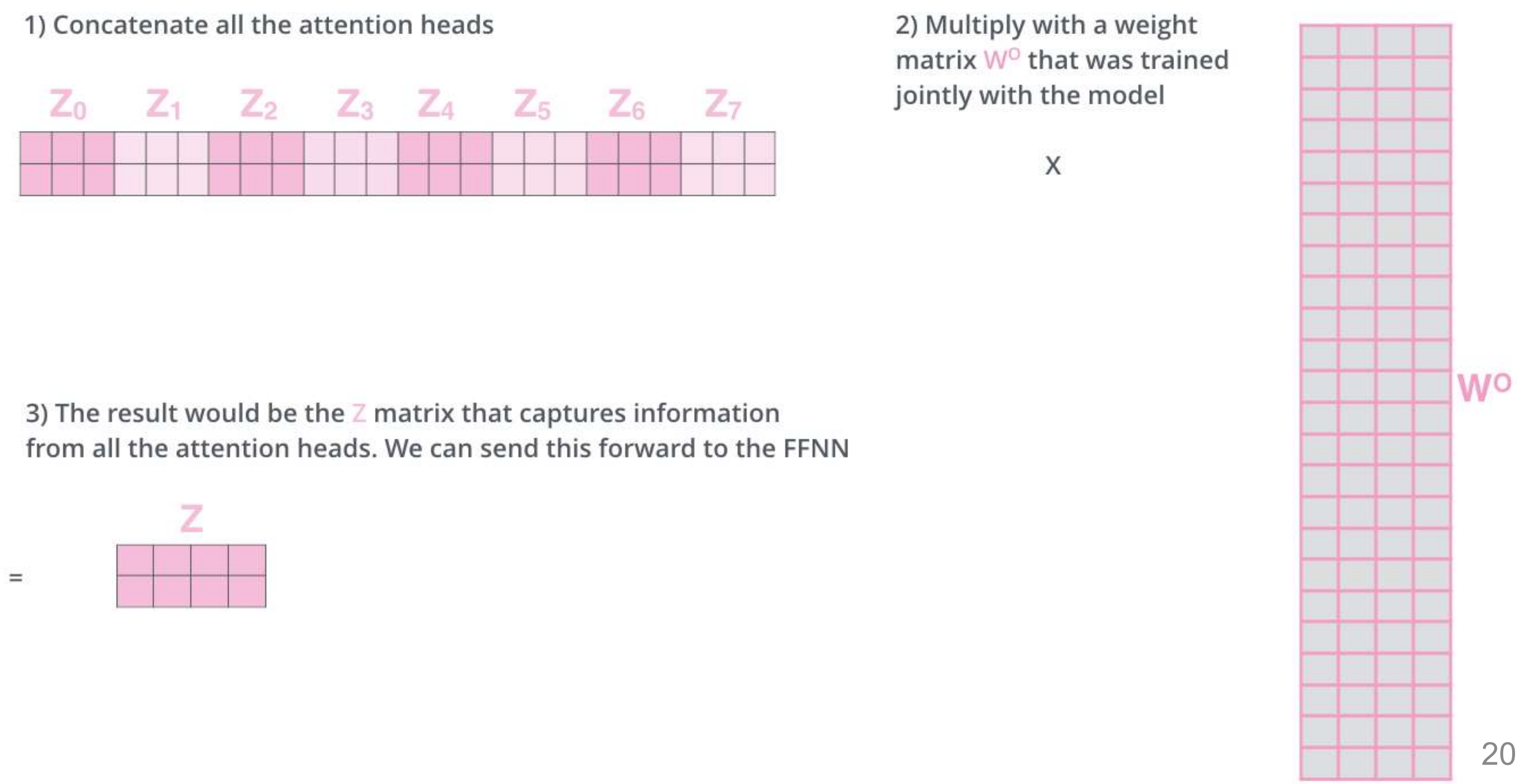
Multi-Headed Attention (cont.)

- If we do the same self-attention calculation as outlined above, just eight different times with different weight matrices, we end up with eight different Z matrices.

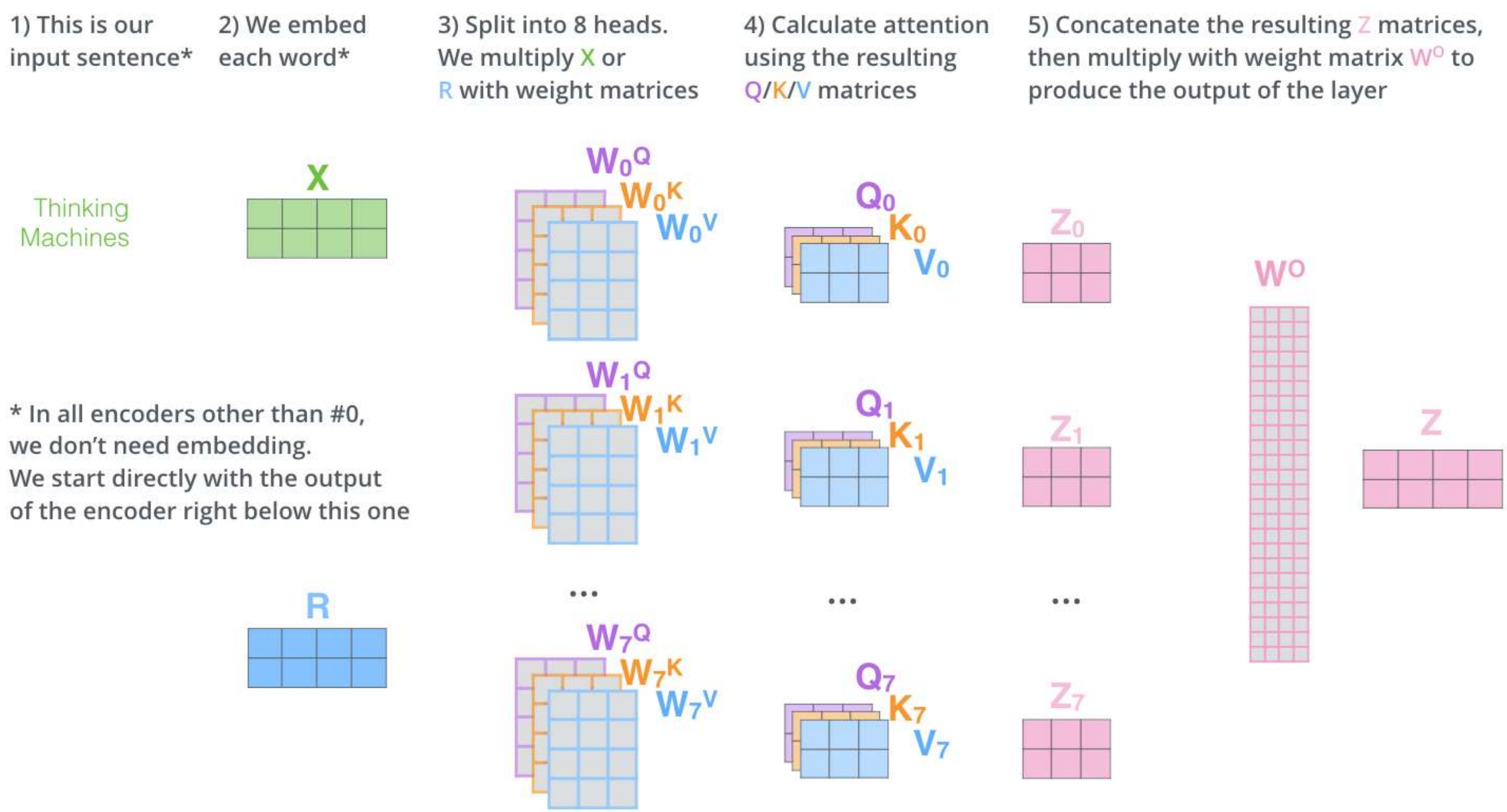


Multi-Headed Attention (cont.)

- We then condense these eight down into a single matrix, by first concatenating the matrices and then multiplying them by an additional weights matrix WO.



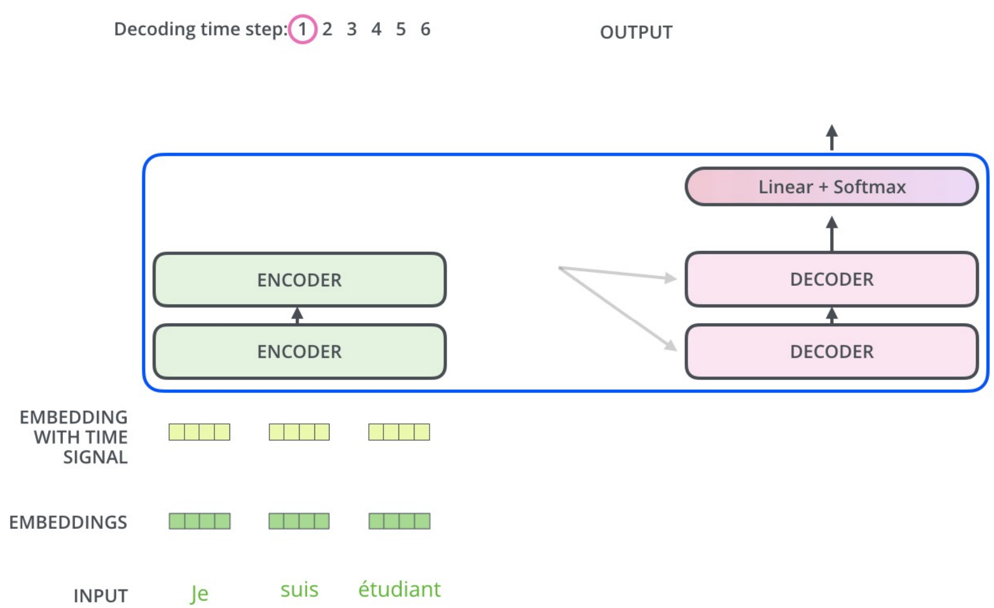
Recap



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

The Decoder Side

- The encoder starts by processing the input sequence. The output of the top encoder is then transformed into a set of attention vectors K and V. These are to be used by each decoder in its “encoder-decoder attention” layer which helps the decoder focus on appropriate places in the input sequence:



The Decoder Side (cont.)

- The following steps repeat the process until a special symbol is reached indicating the transformer decoder has completed its output.