CSE440: Natural Language Processing II

Farig Sadeque
Assistant Professor
Department of Computer Science and Engineering
BRAC University

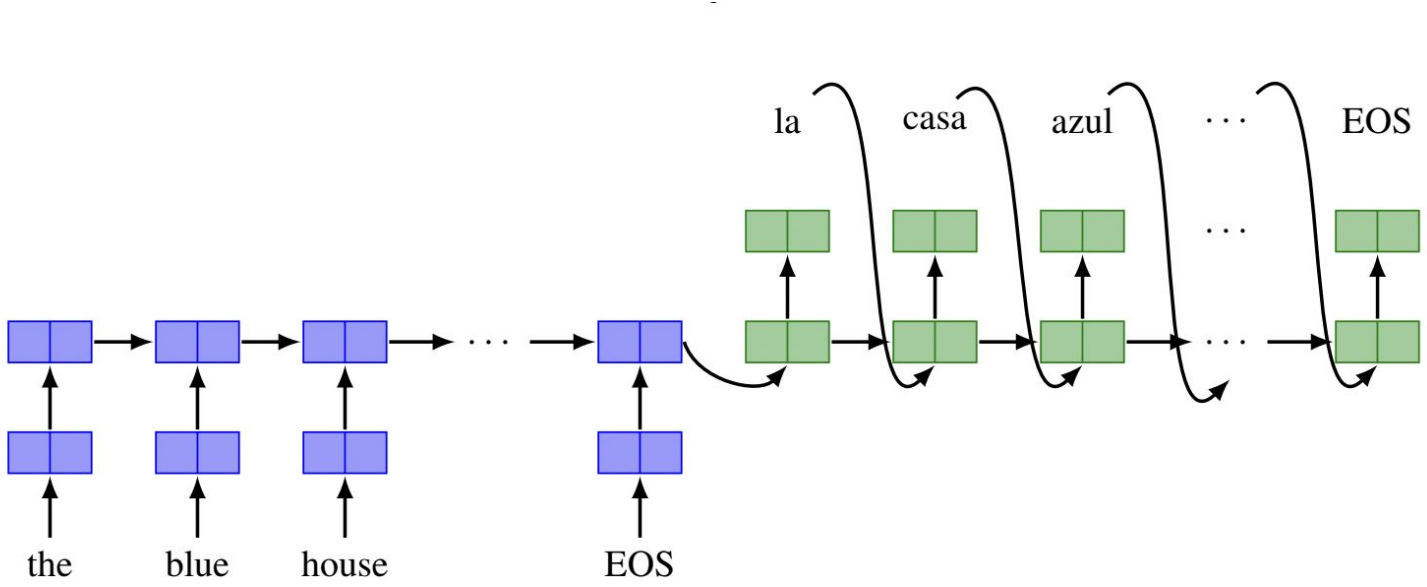
Lecture 7: Transformers

Overview

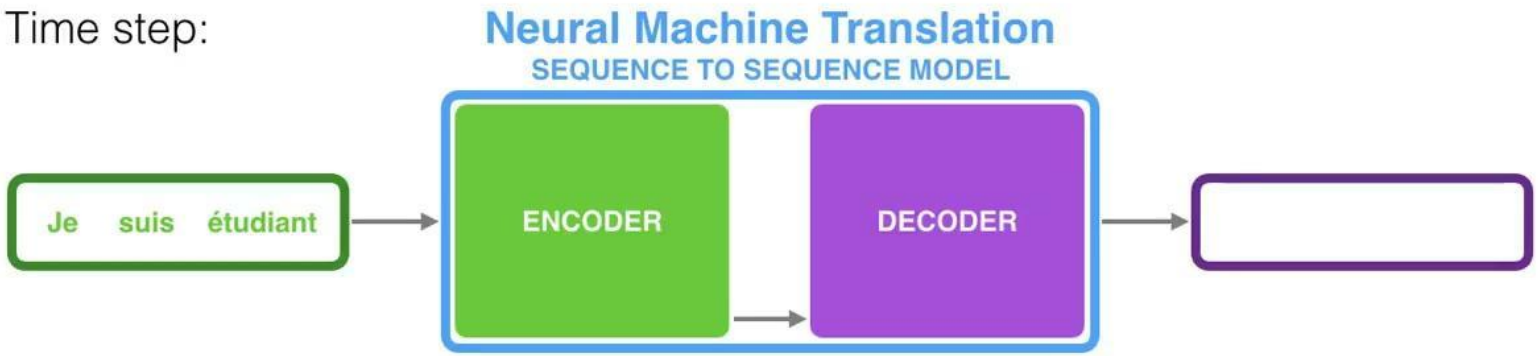
- Seq2Seq models
- Attention
- Transformers
  - Self attention
  - Byte-pair encoding
  - Positional encoding

Seq2Seq Model

- MT model using RNN

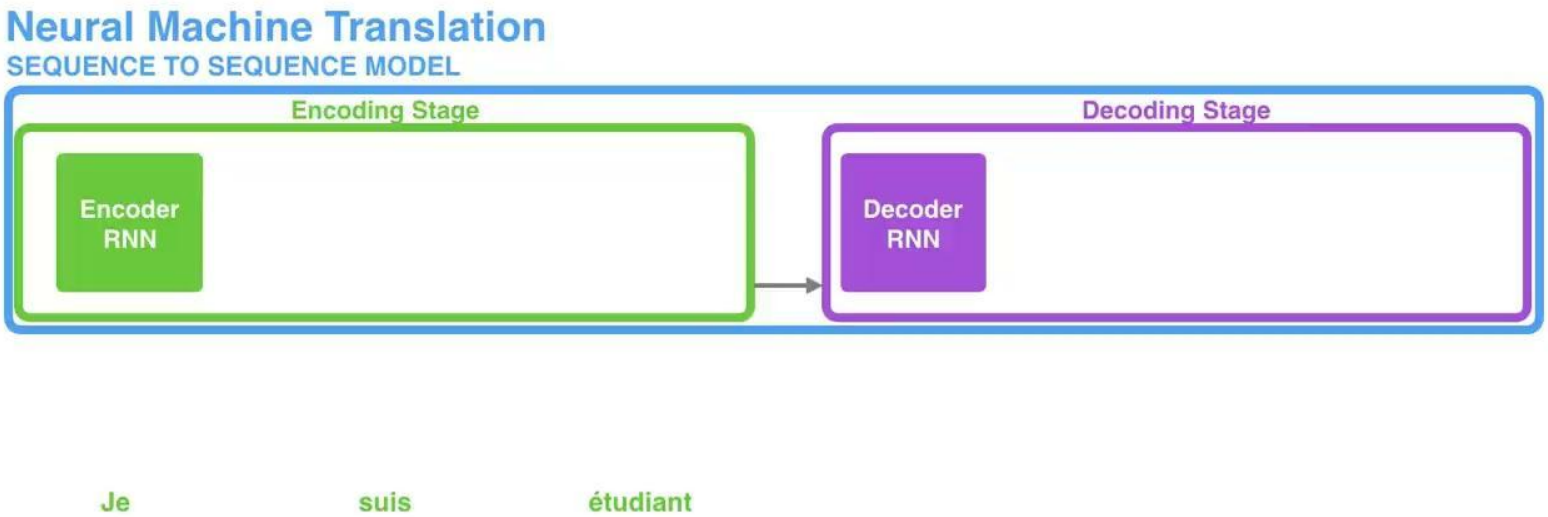


MT model using encoder-decoder



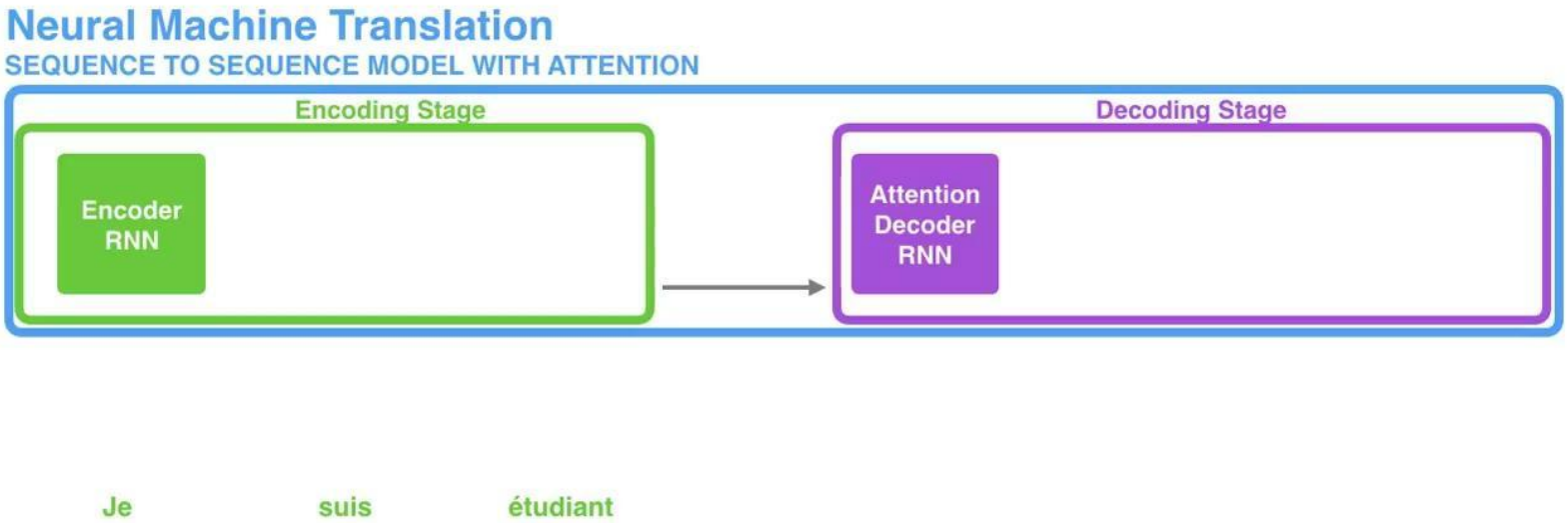
Video Courtesy: Jay Alammar

MT model using encoder-decoder



Video Courtesy: Jay Alammar

MT model using encoder-decoder

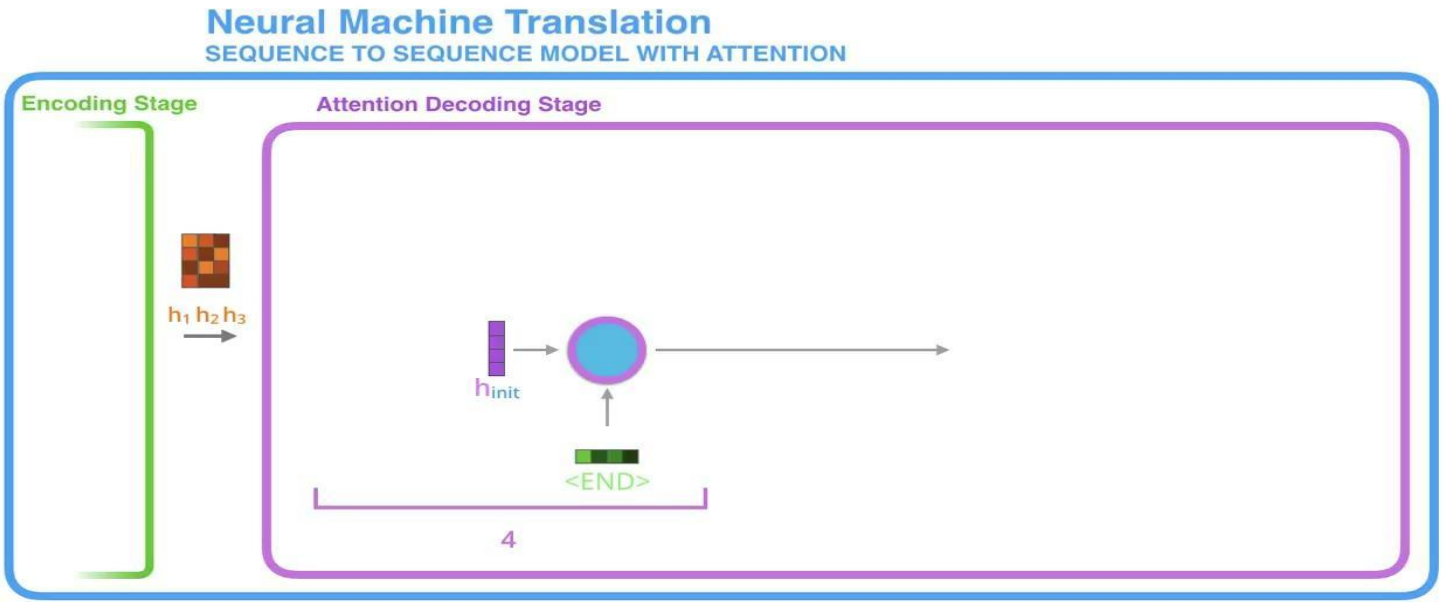


Video Courtesy: Jay Alammar

MT model using encoder-decoder

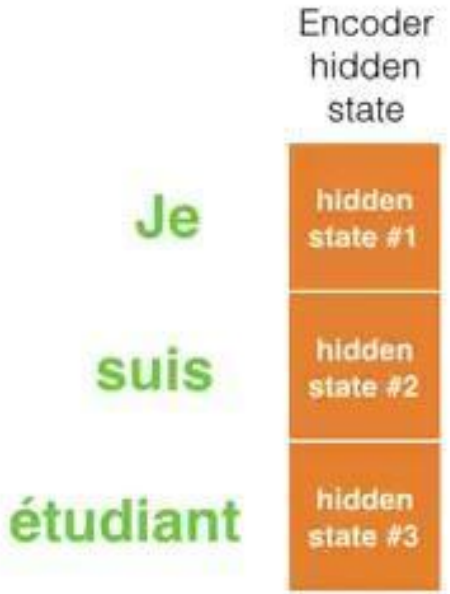
Video Courtesy: Jay Alammar

MT model using encoder-decoder



Video Courtesy: Jay Alammar

MT model using encoder-decoder



Video Courtesy: Jay Alammar

But…

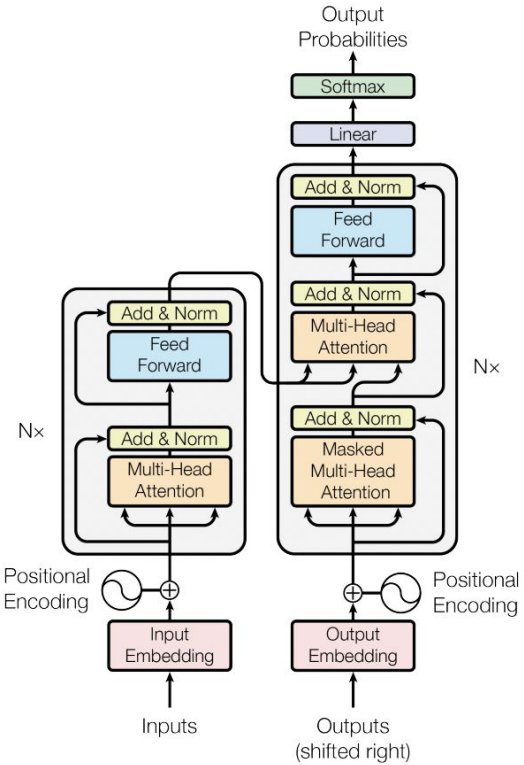
RNN-based encoder decoder works well, but:

- Backpropagation through time and infinite memory is still an issue, even with gated RNNs
- RNNs work sequentially, so parallelization is a challenge

Why do we need GPUs?

- CPUs are latency-optimized, GPUs are bandwidth-optimized
  - The more memory your computational operations require, the more significant the advantage of GPUs over CPUs: matrix multiplication requires more computational operations
- More computing units: better thread parallelism, can hide latency issues
- Faster access to RAMs (VRAMs)
- DNN computations just fit well with GPU architecture
  - Many identical neurons, doing the same computation

Transformer



We will see:

- Self-attention
- Multi-head attention
- Positional encoding
- Byte-pair encoding

Image Courtesy: Attention is all you need

High level view of a transformer

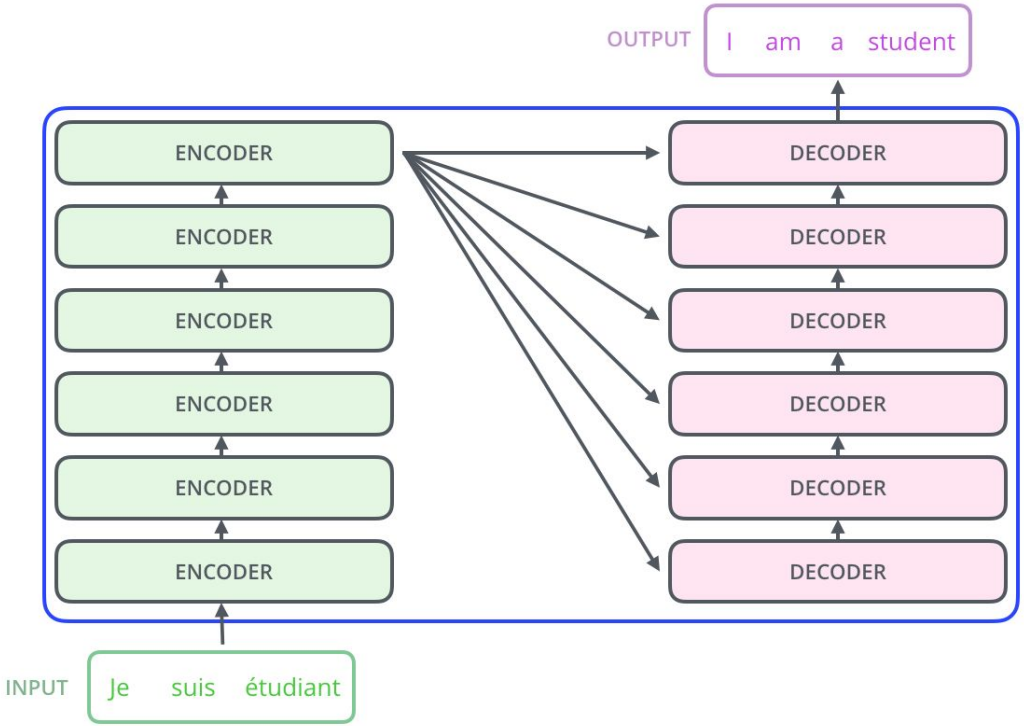


Image Courtesy: Jay Alammar

Self attention

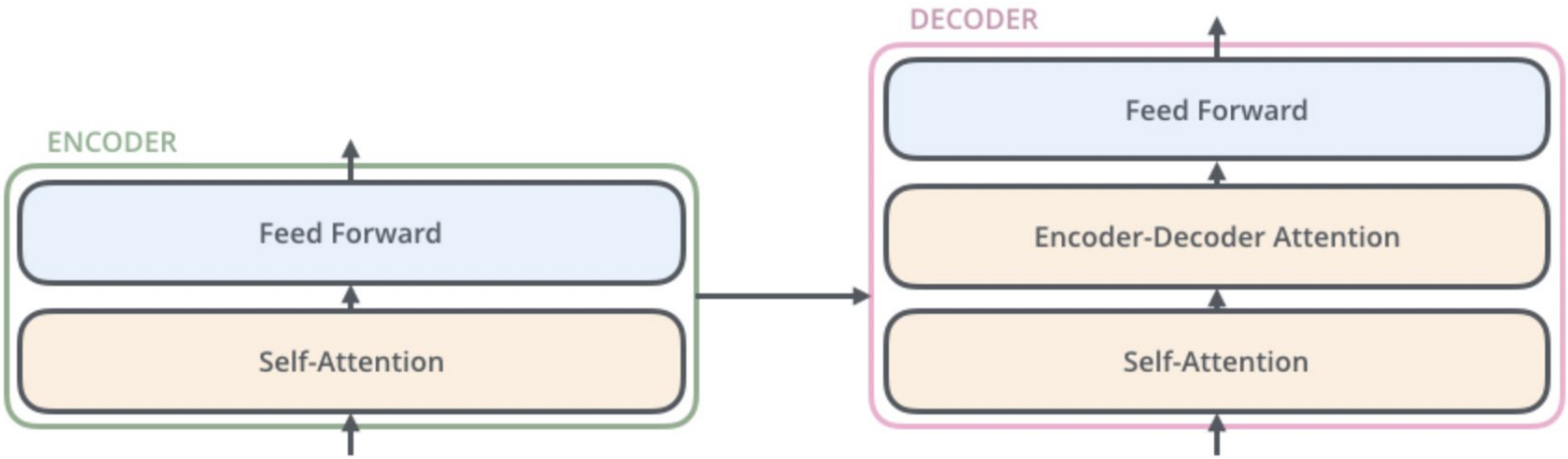


Image Courtesy: Jay Alammar

Self attention

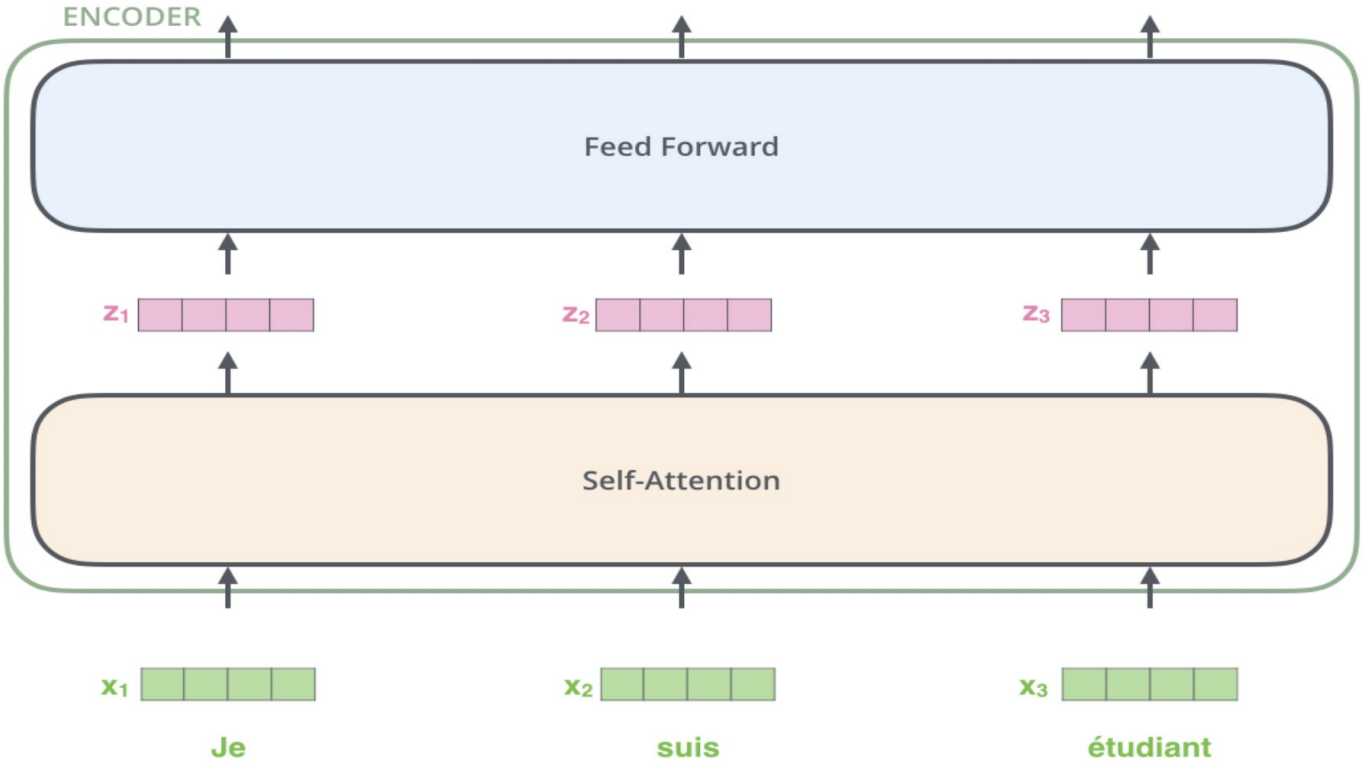


Image Courtesy: Jay Alammar

Self attention

Let’s look at it using a simpler sentence

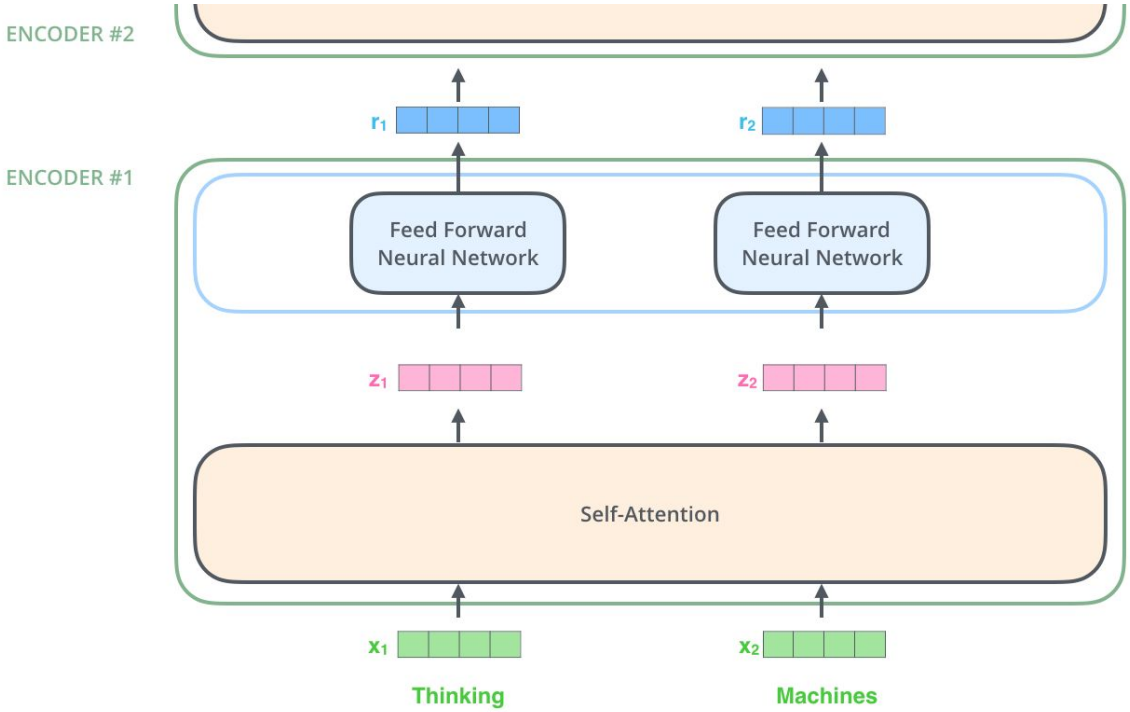


Image Courtesy: Jay Alammar

Self attention at a high level

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

It → animal? street?

For a human, pretty easy, for a machine, not so much.

Why is it called self attention?

- Simple attention focuses on context based on a query
  - For MT, the query could be the translation task
- Self attention takes the relationship of the words in the same sentence into account

Self attention

- An input goes through three linear transformations (mapping and resizing)
  - Query
  - Key
  - Value
- These idea came from information retrieval
  - Query: the string you search for (youtube search string)
  - Key: the string an info retrieval model matches your query with (video titles)
  - Value: the result presented to you (actual videos)
- The model returns (ranks) the best video based on the similarity of your query and the key it searches from– exactly what we want in self-attention
  - Rank the words (find their values) using your current word's representation (query) and all other words' representations (key)

Self attention internally

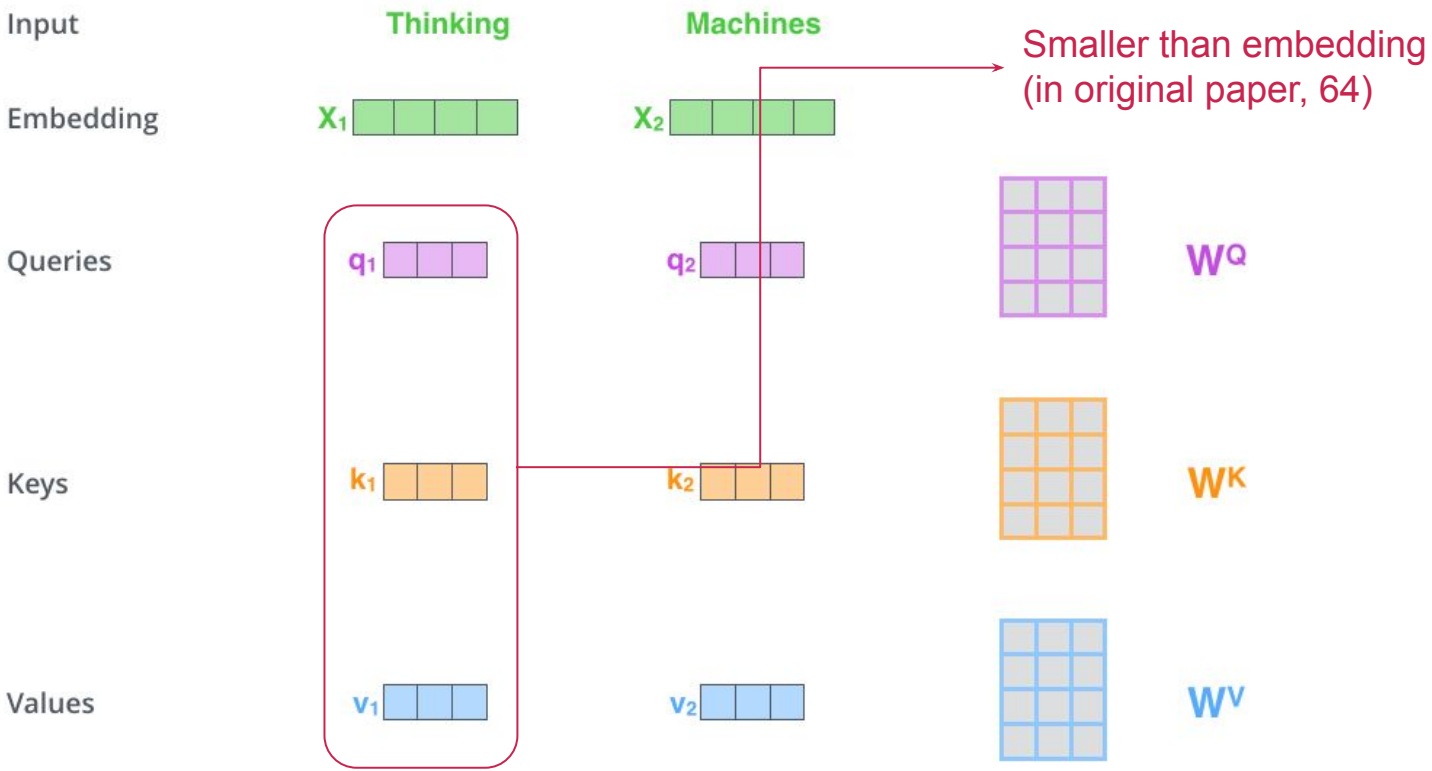


Image Courtesy: Jay Alammar

Self attention maths

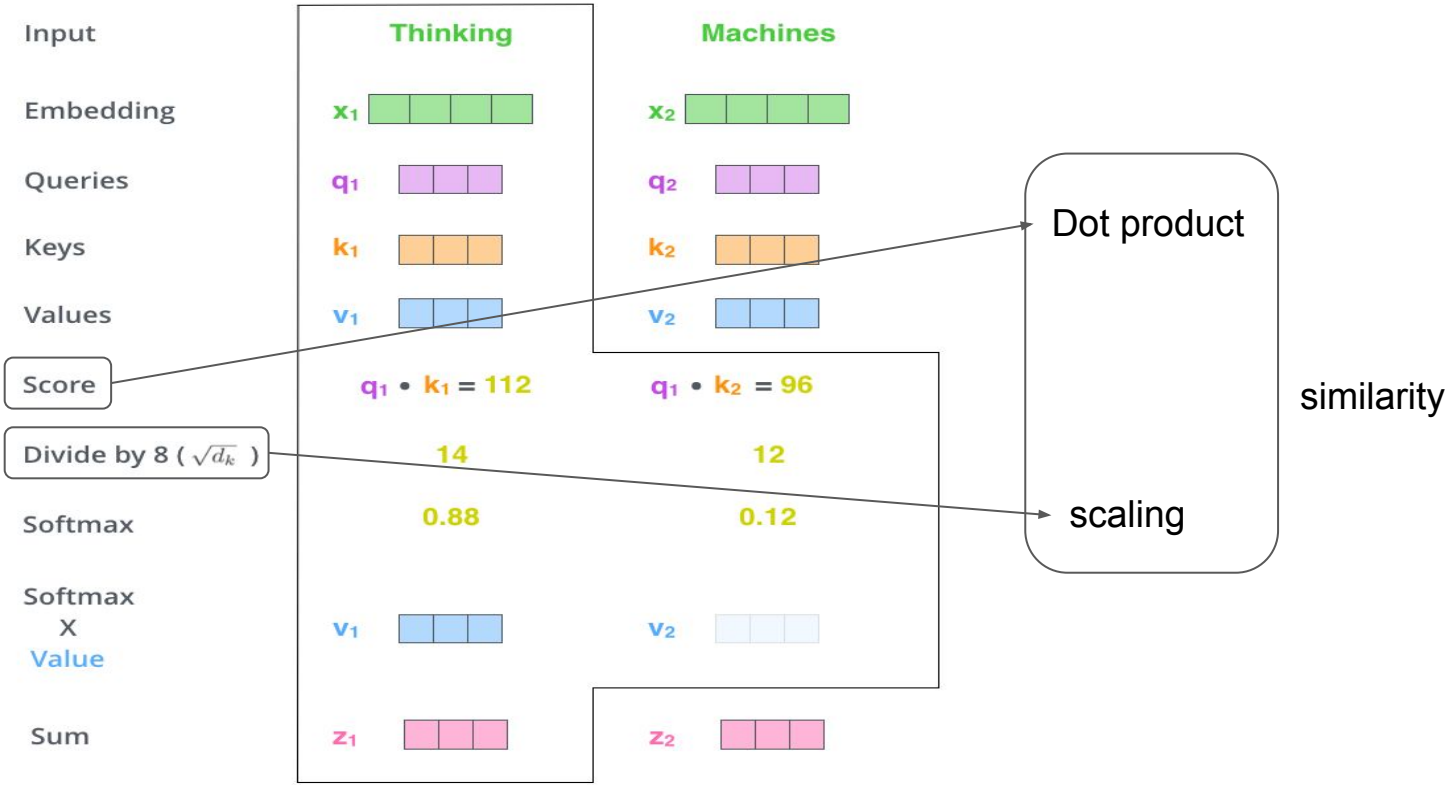


Image Courtesy: Jay Alammar

Multi-head attention

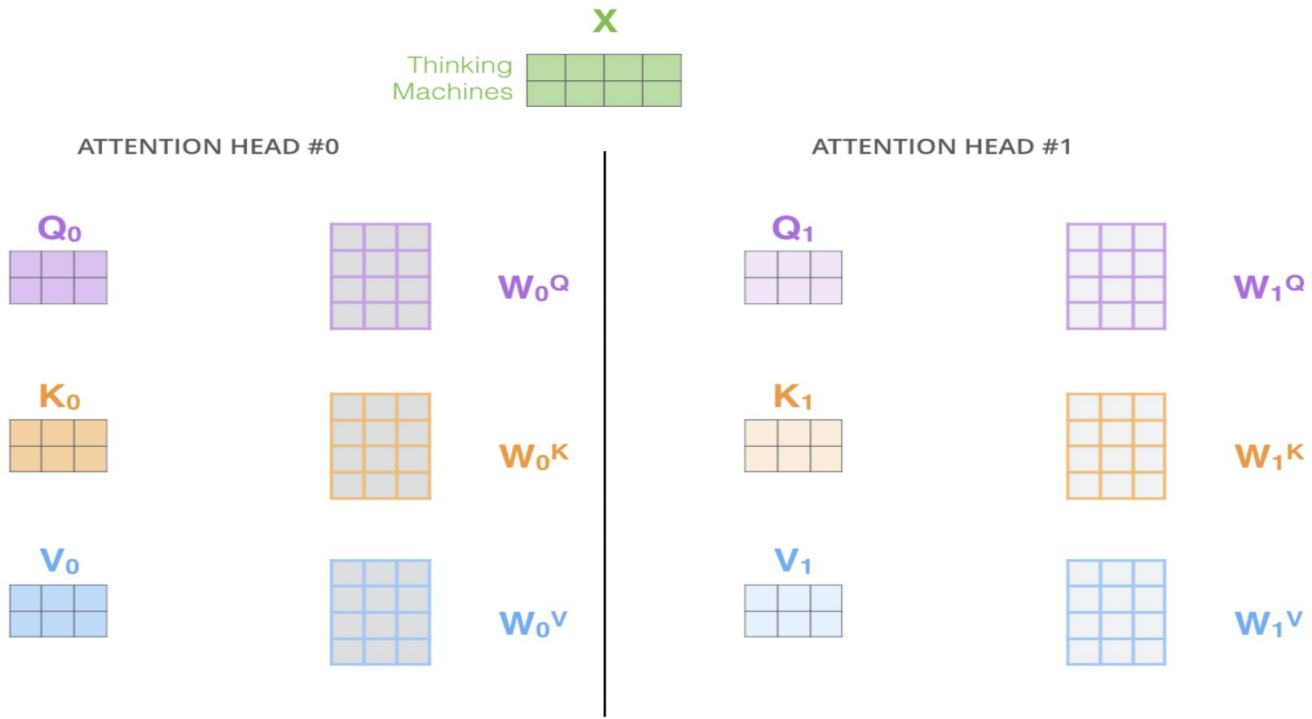


Image Courtesy: Jay Alammar

Multi-headed attention

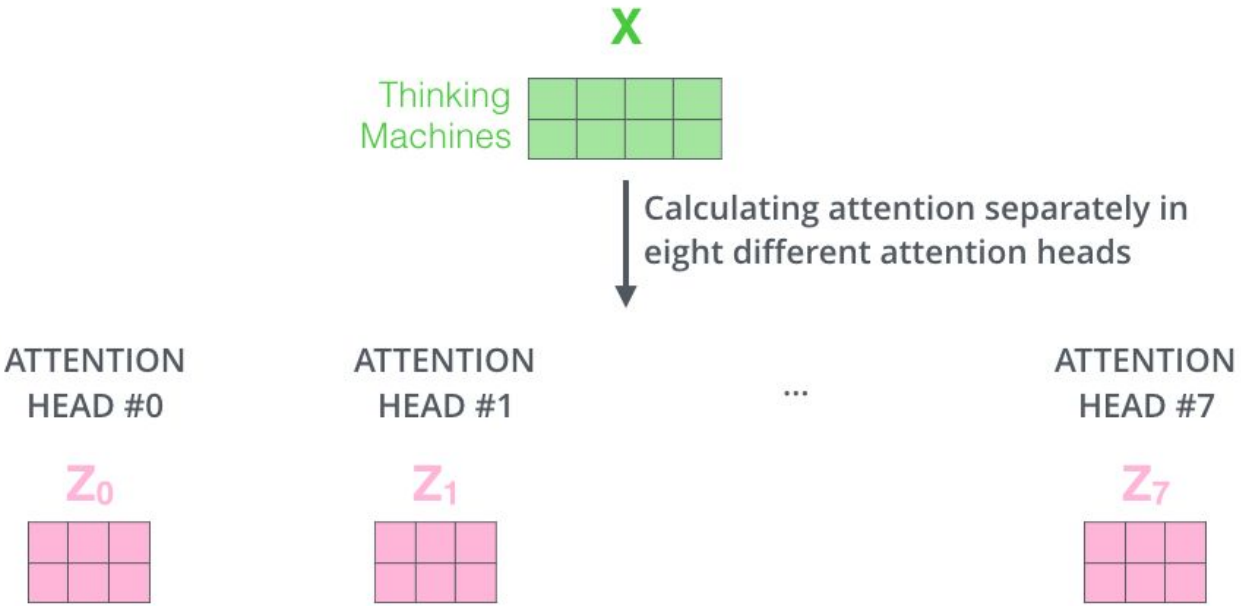


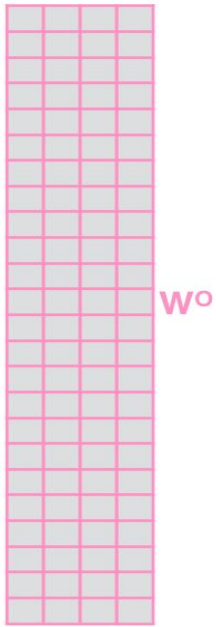
Image Courtesy: Jay Alammar

Multi-headed attention

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

=

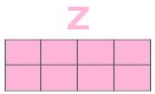


Image Courtesy: Jay Alammar

Multi-headed attention

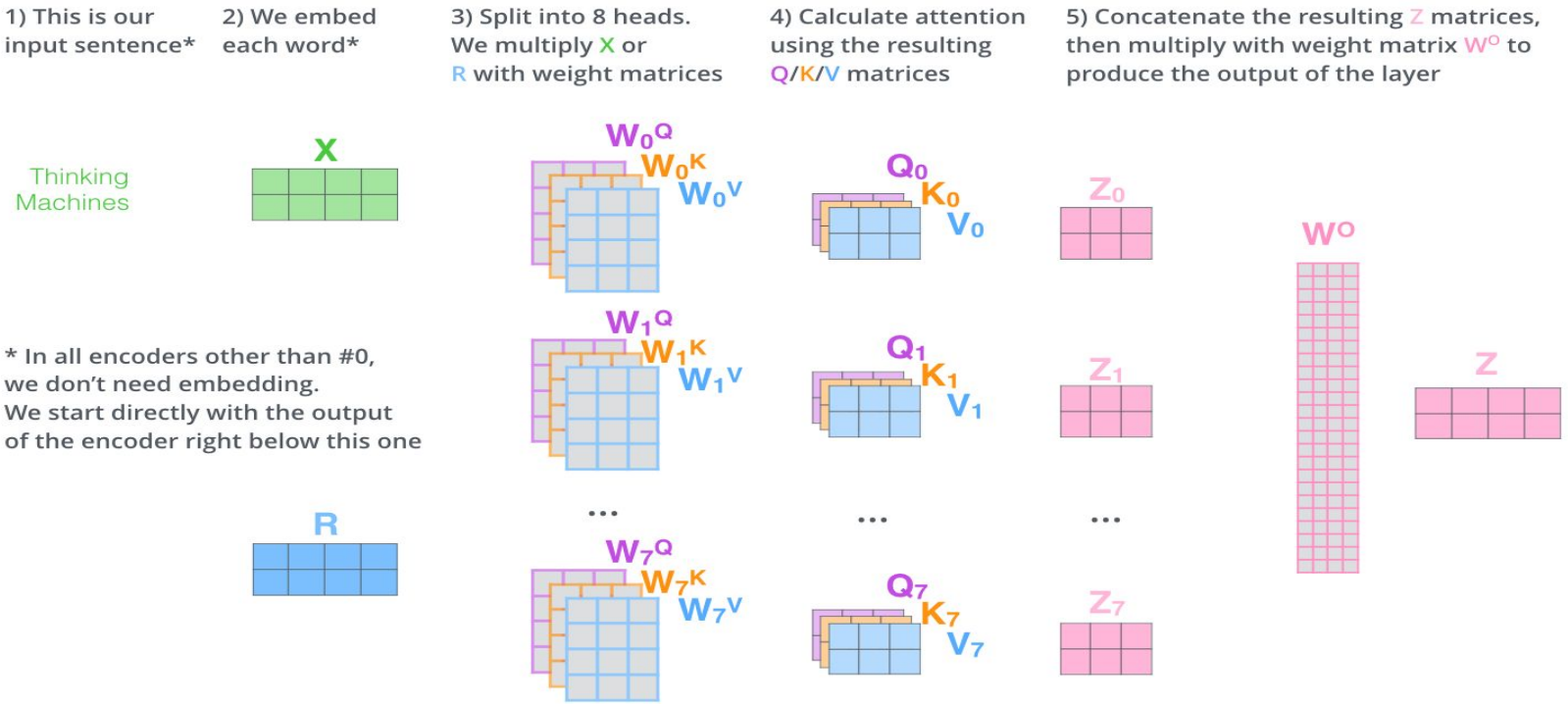


Image Courtesy: Jay Alammar

Why multi headed attention though?

- Because multi is better than one!
- No, really, that’s it!
- Multiple heads can (hopefully) focus on more things hat one head can.

Transformer

- Done! Yay!

Transformer

- Done! Yay!
- Nope!

Positional Encoding

- Need to encode the positional information of a token
- Transformer introduces and adds a vector to each input embedding
- These vectors follow a specific pattern that the model learns, which helps it determine the position of each word, or the distance between different words in the sequence
- Ideally, a positional encoding for a given position should remain the same for all examples, regardless the length
- Hence, a periodic function

Positional Encoding

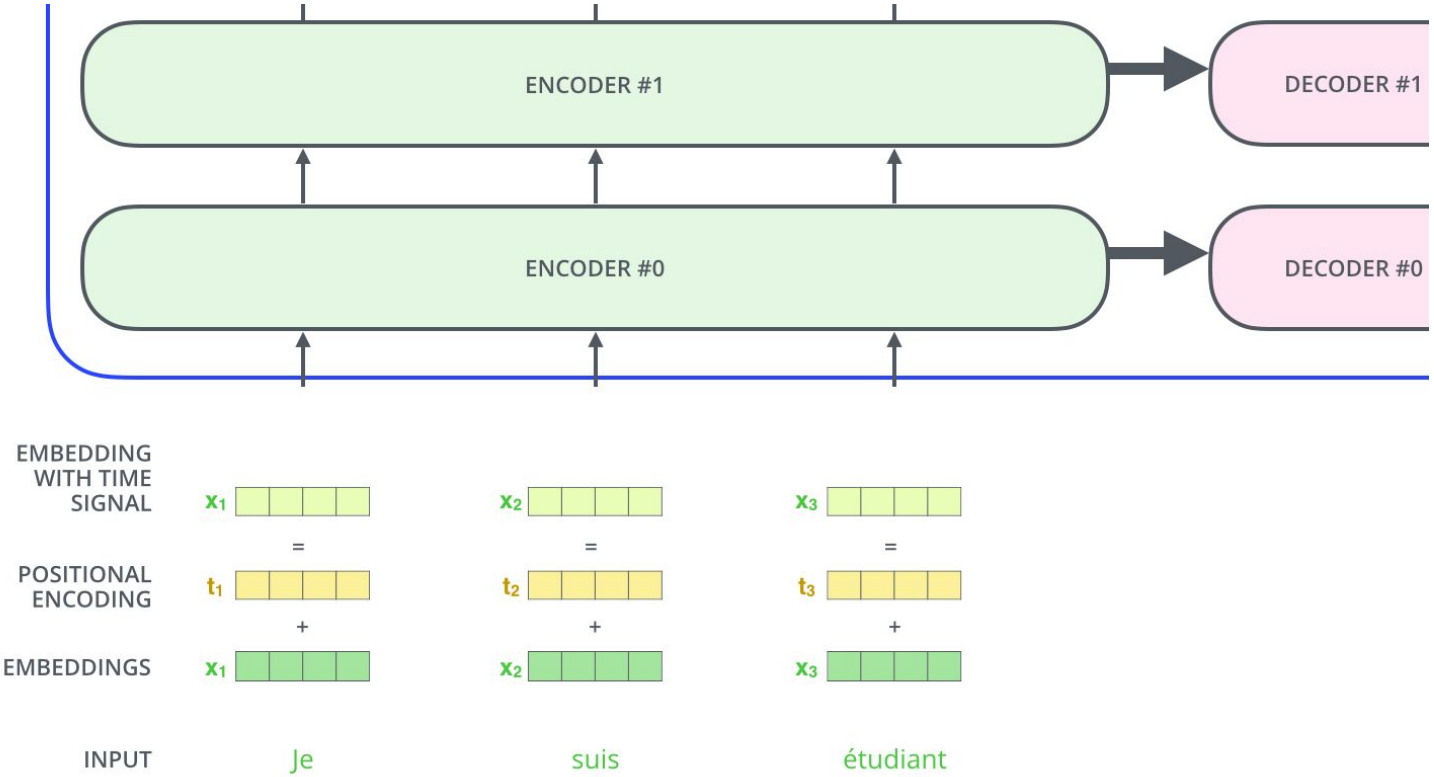


Image Courtesy: Jay Alammar

Transformer

- Done now! Yay!

Transformer

- Done now! Yay!
- Nope!

Input embeddings

- Much simpler than the other word embeddings
  - Creates a vocabulary (needs a limited sized vocabulary, BPE handles that)
  - Assigns each word an index number and a random vector representation
  - Original paper had 512 sized vectors

Transformers

- Surely done now? 😬

Transformers

- Surely done now? 😬
- Not yet! Haven’t seen the decoder!

Decoder during generation

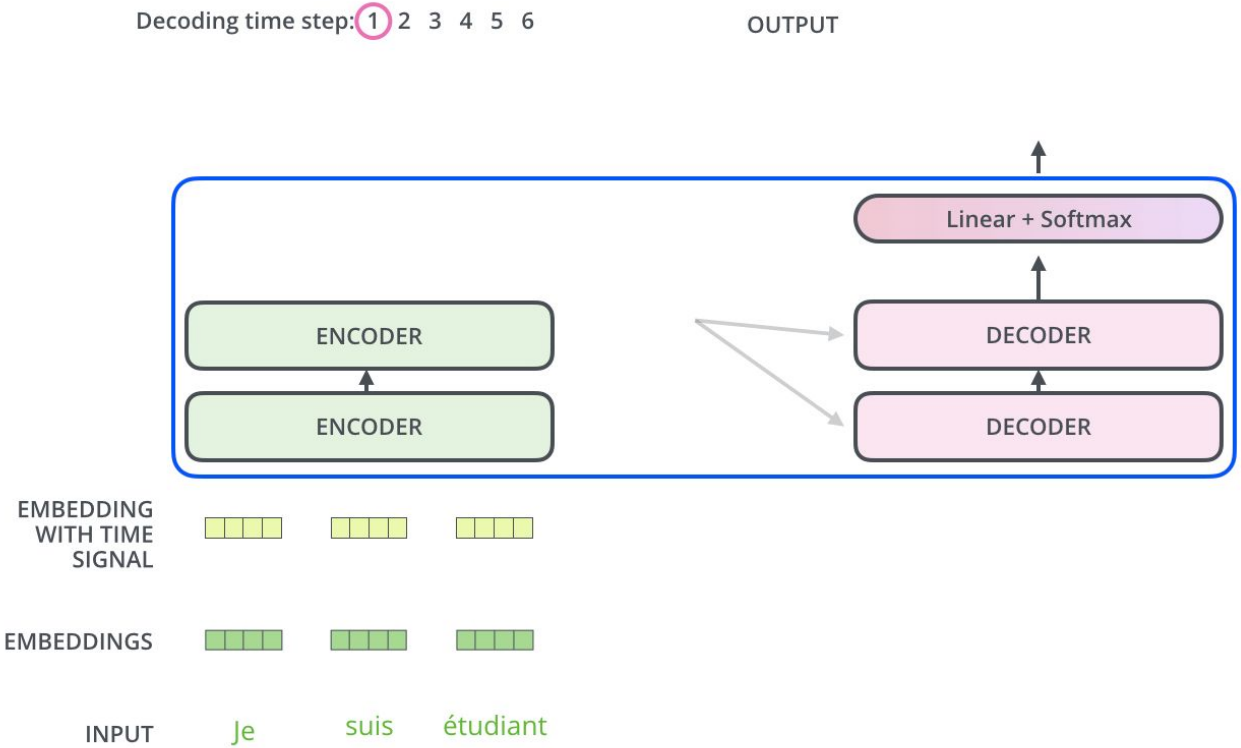


Image Courtesy: Jay Alammar

Decoder during generation

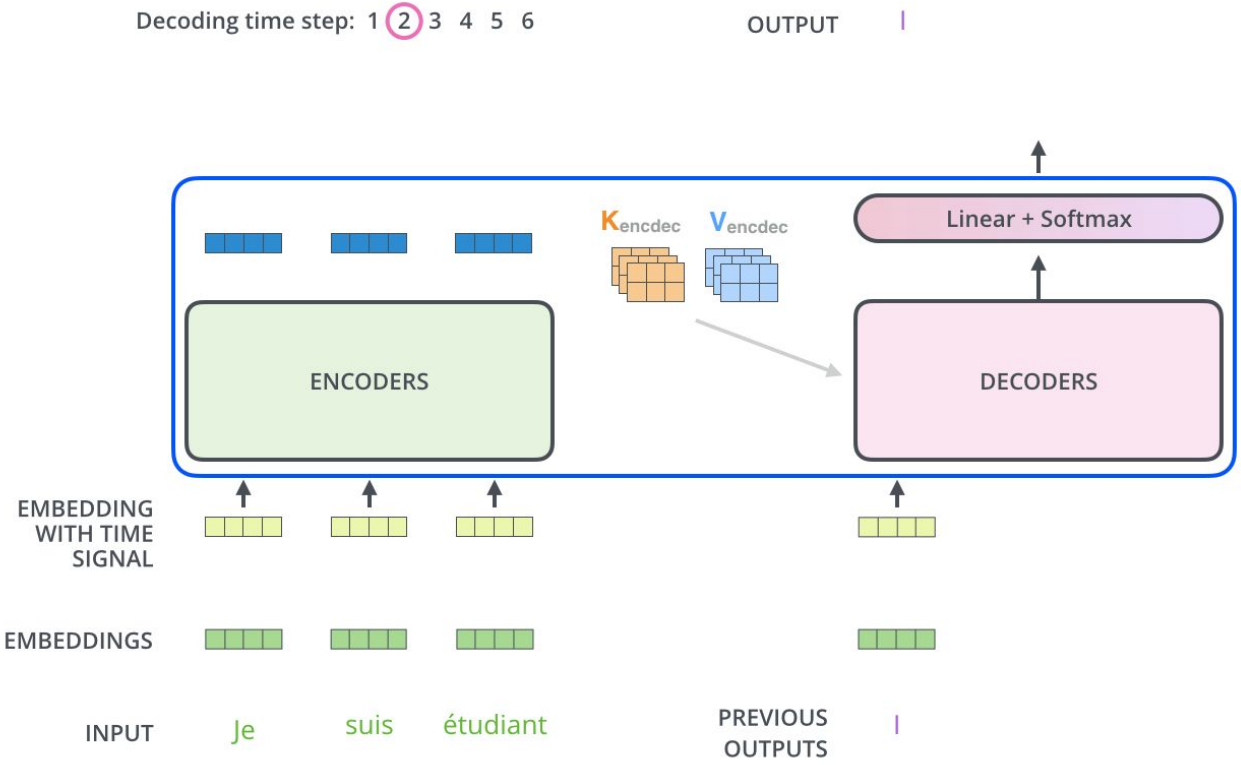
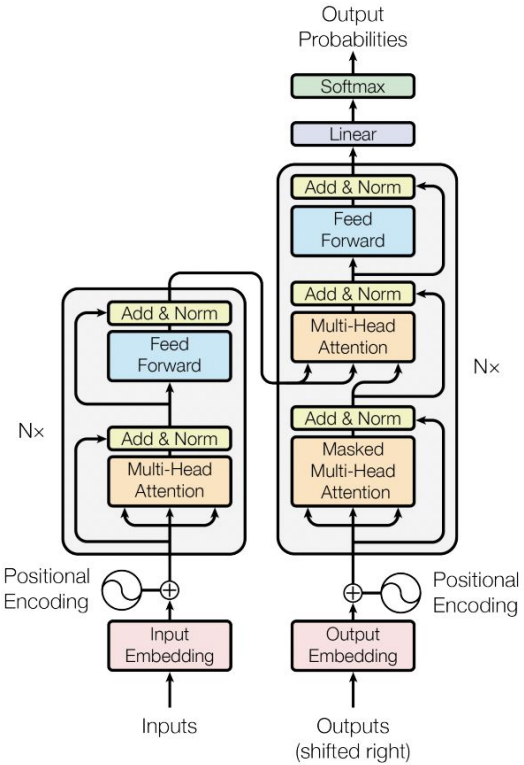


Image Courtesy: Jay Alammar

Final linear and softmax layer



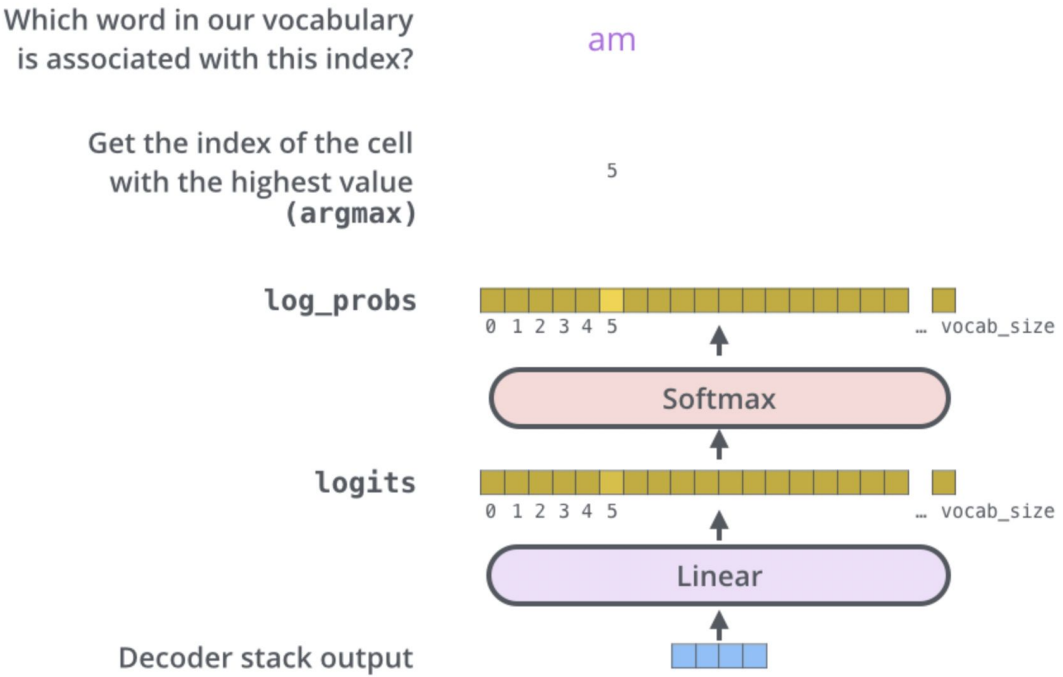


Image Courtesy: Jay Alammar

Decoder during inference

- Decoder multi-head attention layer takes input from the final encoder layer, splits that into two parts (key and value) using W_k and W_v from the encoder
- Uses the previous word's representation (from the deeper layers of the decoder) and uses it as the query to predict the current word
- This predicted current word is then used as the query to predict the next word

Decoder during training

- Masked multi-head attention layer masks the original current word during training and let's the model predict the current word
- After prediction, it unmasks the current word, calculates how wrong it was (binary cross entropy) and updates the weights accordingly. But…
- Instead of forcing the current prediction to generate the next word (like in inference), it uses the unmasked original word during training
    - This is also known as teacher forcing

Transformers

- Finally, done! 😁

Transformers

- Finally, done! 😁
- Naaah. Still need to figure out something else.

Tokenization

- Word tokenization is great, but there are more intelligent ways to tokenize
- Some languages like to combine words to create new words
  - German: Krankenhaus → Hospital (Kranken: sick, haus: house)
- English does inflections
  - Low, lower, lowest
- If we can focus on the core building words, we can reduce our vocabulary size significantly
  - Stemming and lemmatization is not going to be used

Byte-pair encoding

- We convert words into character (byte) pairs and encode them incrementally
- Example from Wikipedia:

  aaabdaaabac → ZabdZabac (Z=aa) → ZYdZYac (Y=ab) → XdXac (X=ZY)

- How can we use this in transformer? https://arxiv.org/pdf/1508.07909.pdf

Transformer

- Finally! 😫

Transformer

- Finally! 😫
- Wait! Let’s see an implementation of this.

Training a transformer

- Training a transformer is exceptionally difficult
- Not enough hardware, not enough time
- What to do?

Pretraining

- Transformers can be trained to learn through language representations
- Someone with enough hardware and time can (pre)train a transformer that can learn that language representation, and then we can use that representation for our NLP task

BERT

- BERT = Bidirectional Encoder Representations from Transformers
- Designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers
- Pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models
- Achieved state-of-the-art performance in almost all tasks when it came out
- Uses WordPiece tokenization
- Are massive
    - Base model has 12 encoders, 12 attention heads, 768 hidden units, large model has 24 encoders, 16 heads and 1024 hidden units
    - Base models counts up to 110 million parameters, large has 340 mils.

BERT

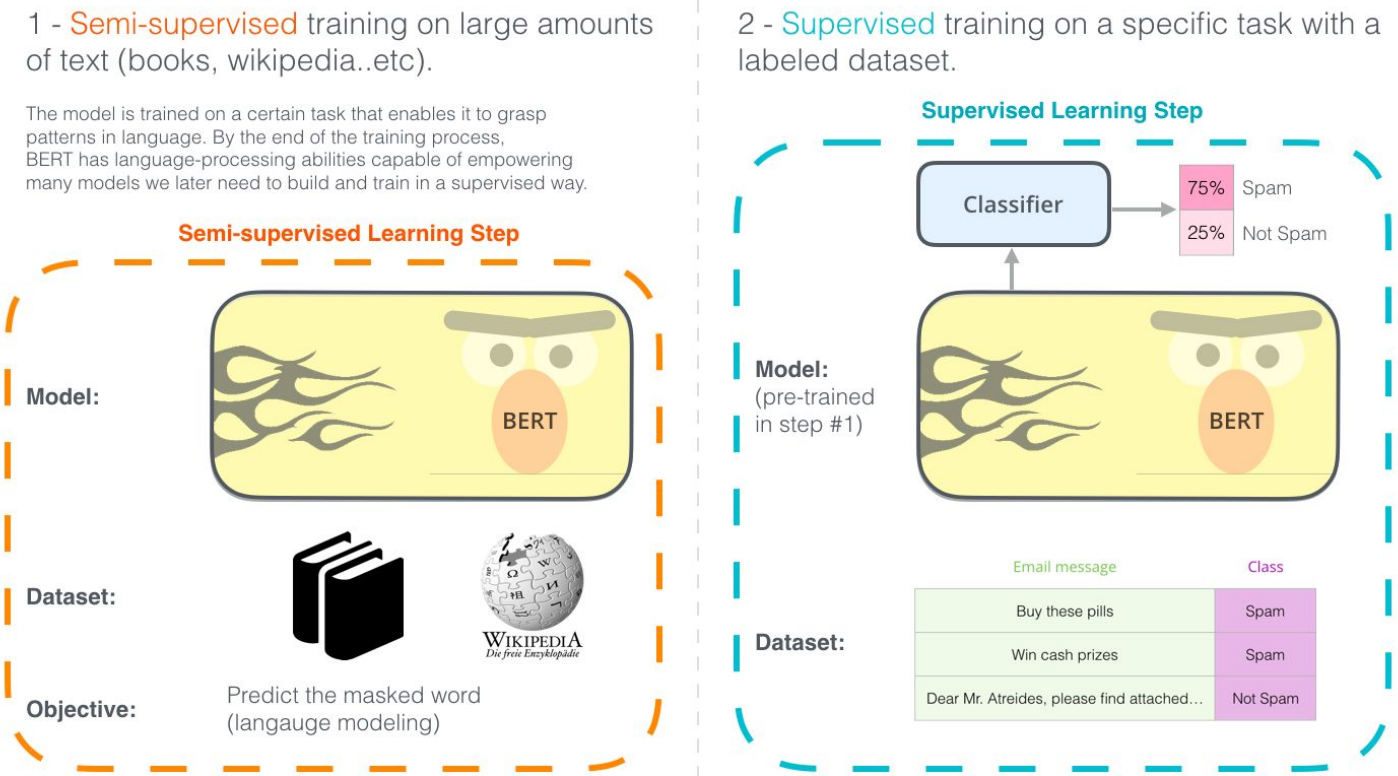


Image Courtesy: Jay Alammar

BERT

- Pretrained on two unsupervised task
  - Masked language modeling
  - Next sentence prediction
- Masked LM
  - Chooses 15% of tokens at random
  - Replaces a token with a [MASK] 80% of the time, with a random token 10% of the time and does not replace 10% of the time
- Next sentence prediction
  - Can model tasks that are not covered by language modeling (QA, inference)
  - Training data includes: 50% of the time B is the actual next sentence that follows A and 50% of the time it is a random sentence from the corpus

How to use BERT

- Task specific-Models (fine-tuning)
- Feature extraction
  - But which one should we use?

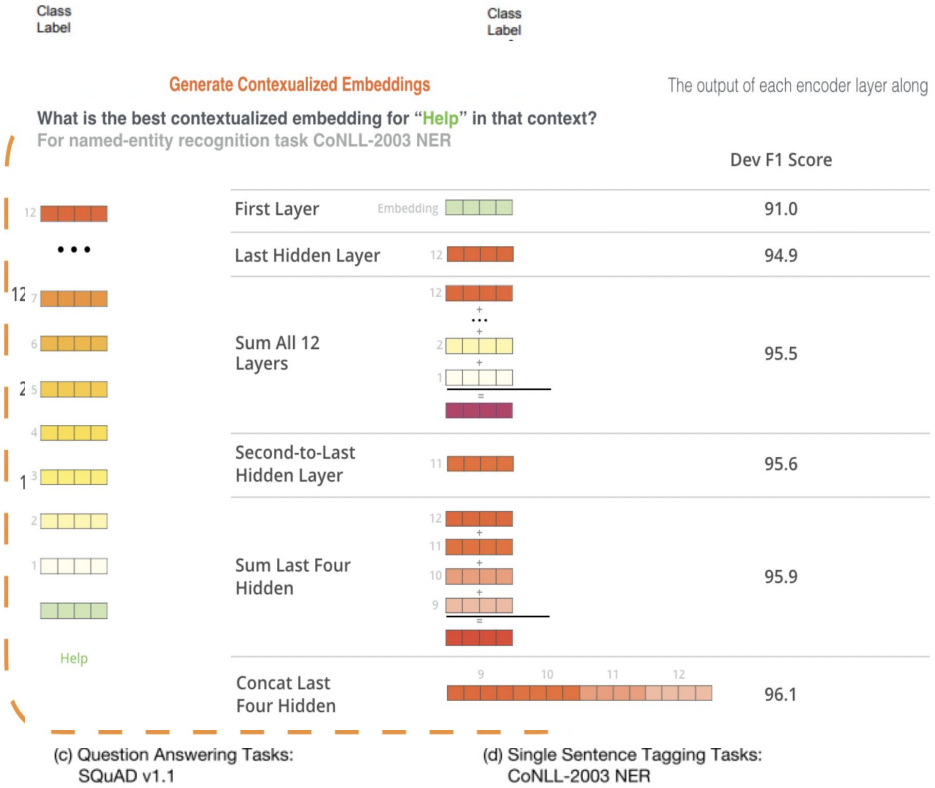


Image Courtesy: Jay Alammar