# Deep Learning II: Transformer

COMP5318 Machine Learning and Data Mining
Semester 2, 2024, week 9
**Nguyen H. Tran (**Based on “The Illustrated Transformer” by Jay Alammar**)**

THE UNIVERSITY OF SYDNEY

# Transformer

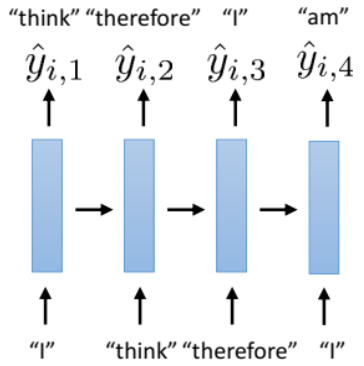
Based on “The Illustrated Transformer” by Jay Alammar

# A High-Level Look

- **Machine translation application**: take a sentence in one language, and output its translation in another

# A Basic Neural Language Model

- **Training data:** natural sentences
  - Nice to meet you!
  - Xin chào! Rất vui được quen bạn.
  - 만나서 반갑습니다
- In reality there could be several million of these. How are these represented?
  - Tokenize the sentence (each word is a token)
  - **Simplest:** one-hot vector
  - **More complex:** word embeddings




RNN-based model

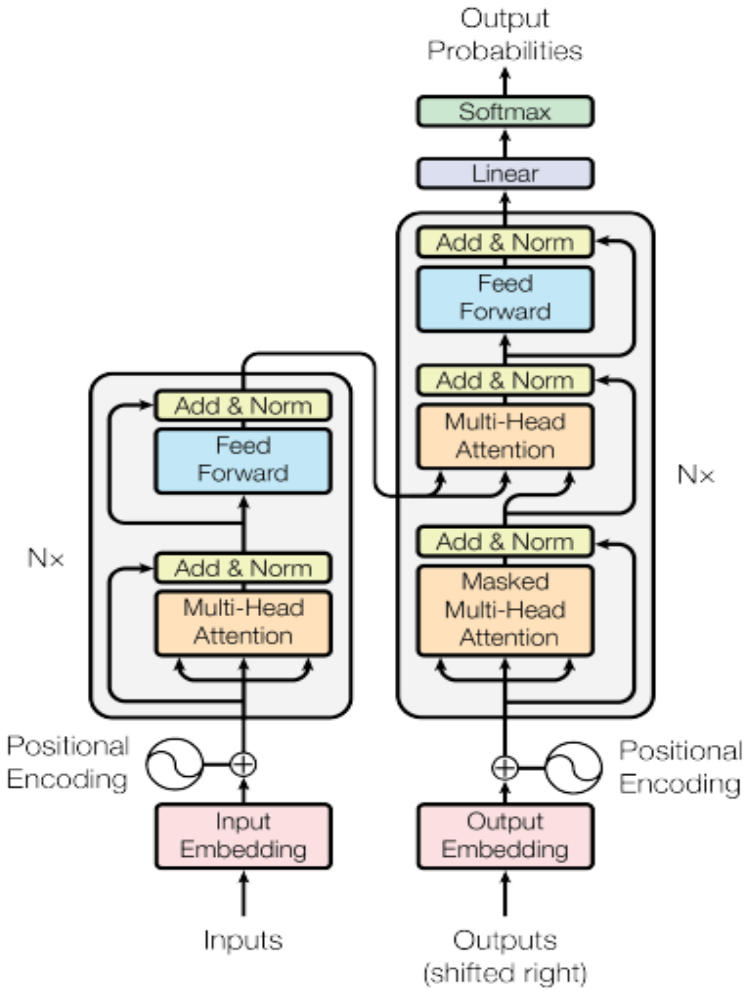
$$x_{1,1} = \begin{bmatrix} 0 \\ 0 \\ . \\ 0 \\ 1 \\ 0 \\ . \\ 0 \end{bmatrix}$$

dimensionality = number of possible words

index of this word

# Transformer

Encoder

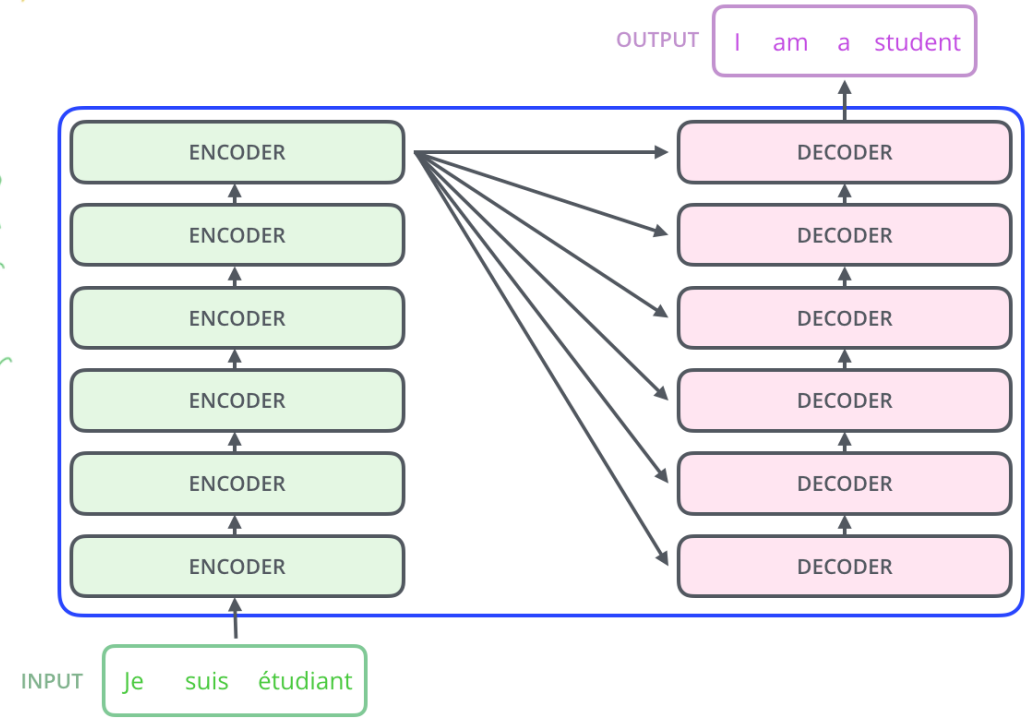


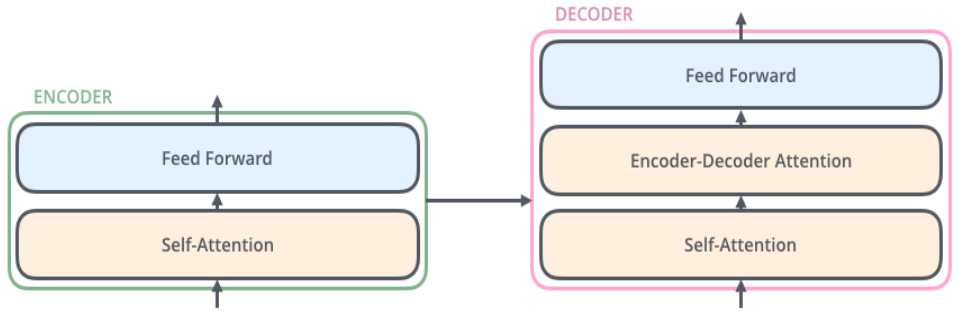

Decoder

Vaswani et al. (2017), *Attention is all you need.*

# A High-Level Look

- **Encoding component**: a stack of encoders identical in structure
  - *Self-attention layer*: helps look at other words in the input sentence as it encodes a specific word
- **Decoding component**: a stack of decoders of the same number of the encoders.
  - *Encoder-decoder attention layer:* helps focus on relevant parts of the input sentence

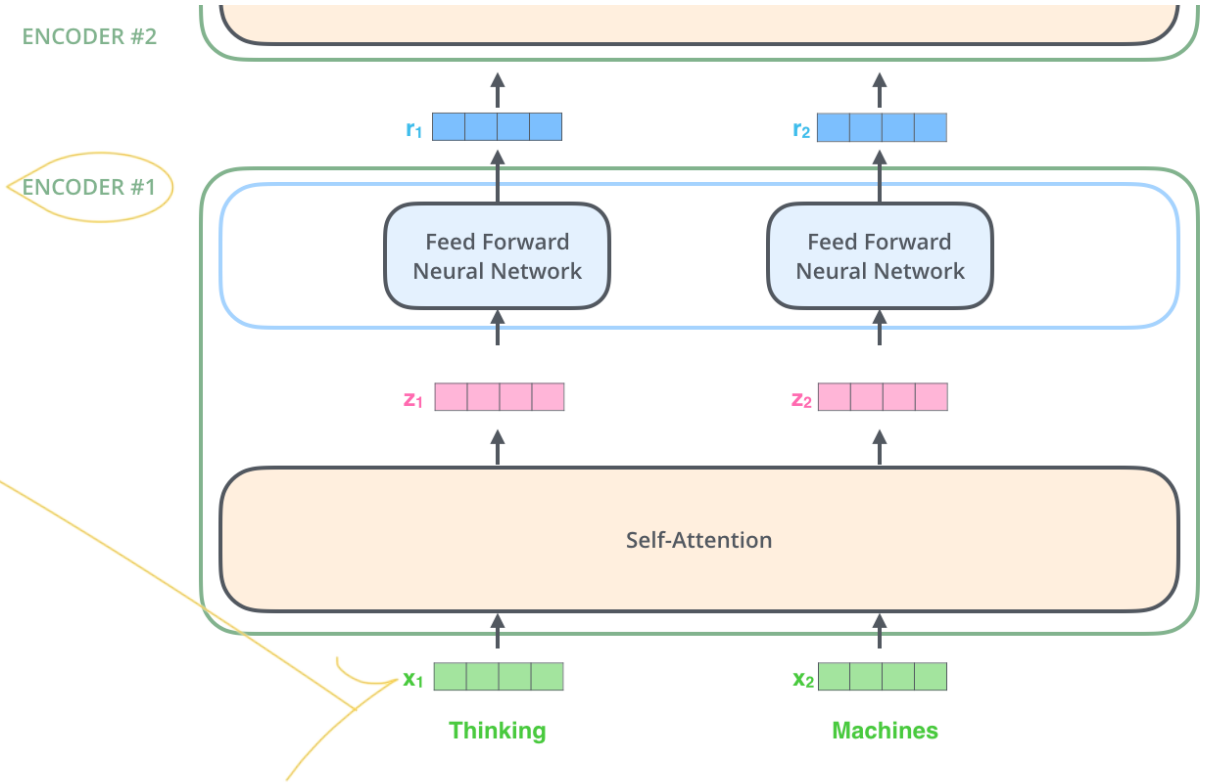


# The Encoder Side

Encoder 的方法

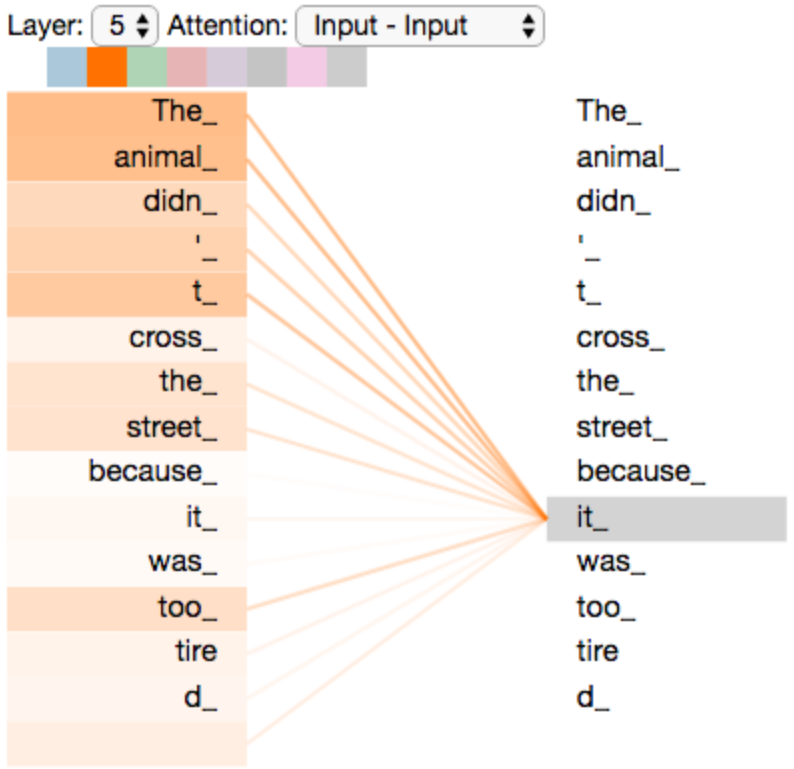
- **Embedding** happens in the bottom-most encoder
  - Turning each input word into a vector
  - Vector size would be the length of the longest sentence in training set
- Each vector flows through each of the two layers of the encoder
  - *dependencies* in the self-attention layer, but not in feed-forward layer
  - can be executed in *parallel* while flowing through the feed-forward layer



# Self-Attention at a High Level

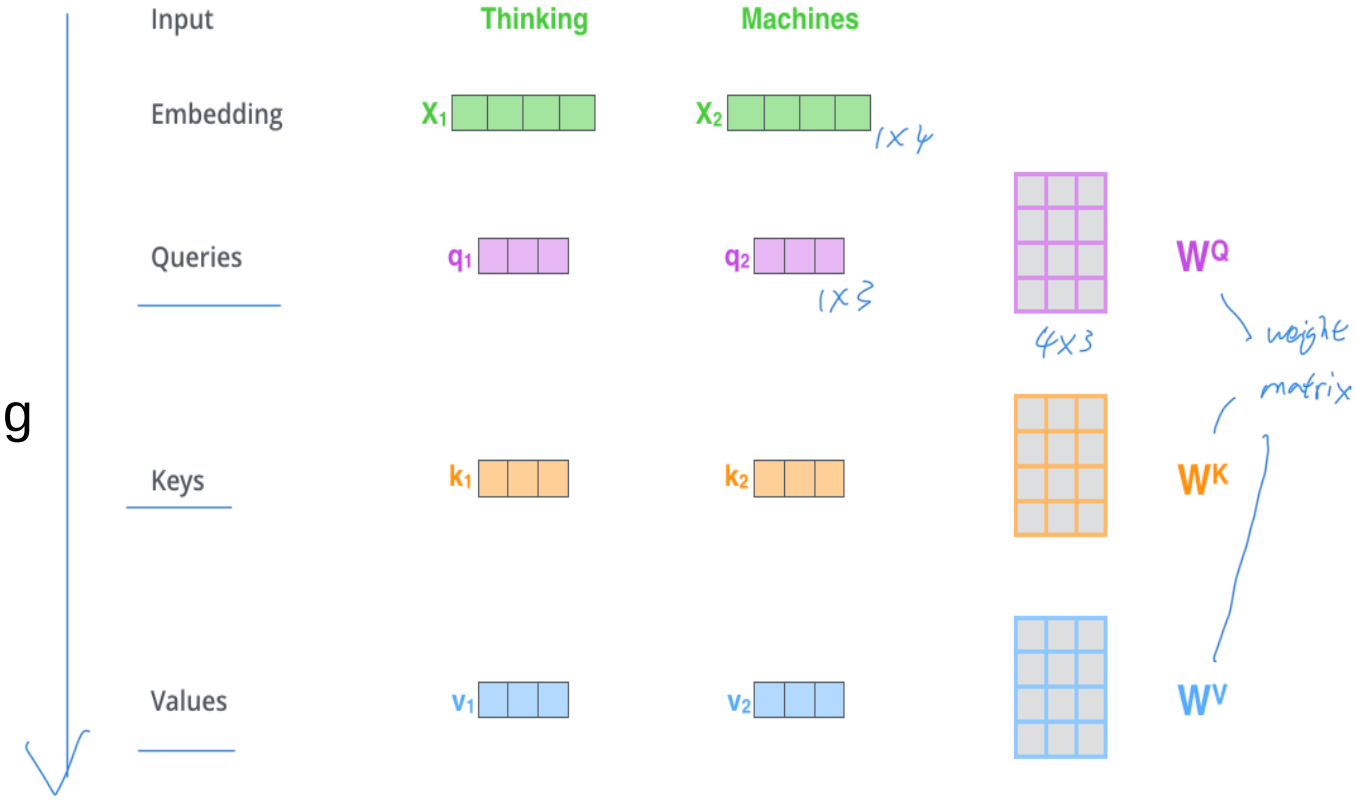
*"The animal didn't cross the street because it was too tired"*

- What does "it" in this sentence refer to?
  - Self-attention allows it to associate "it" with "animal".
  - Look at other positions in the input sequence for clues that can help lead to a better encoding for this word.



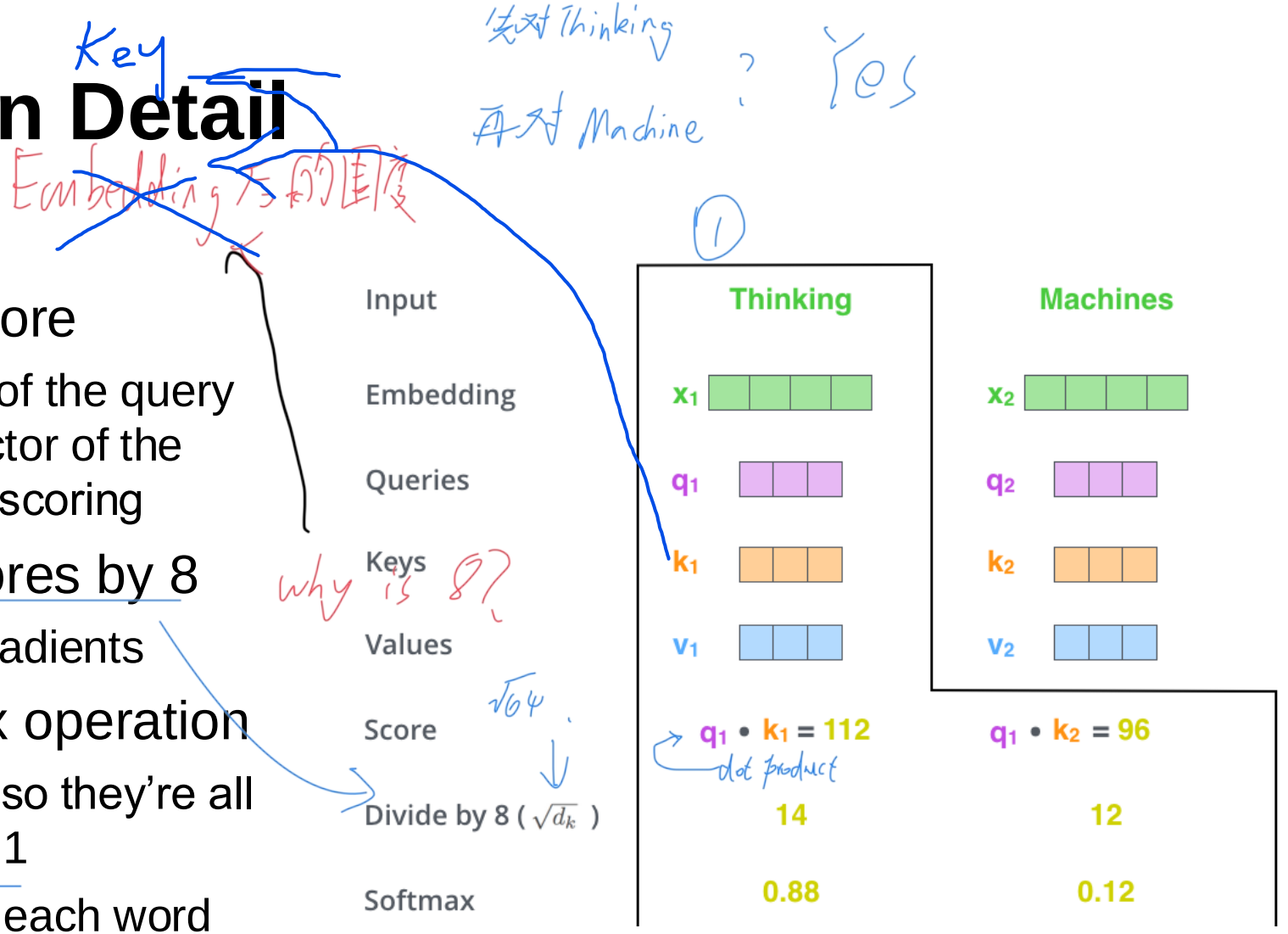
# Self-Attention in Detail

- **Step 1**: create a Query vector, a Key vector, and a Value vector from each of the encoder's input vectors
  - created by multiplying the embedding by three matrices that we trained during the training process



# Self-Attention in Detail

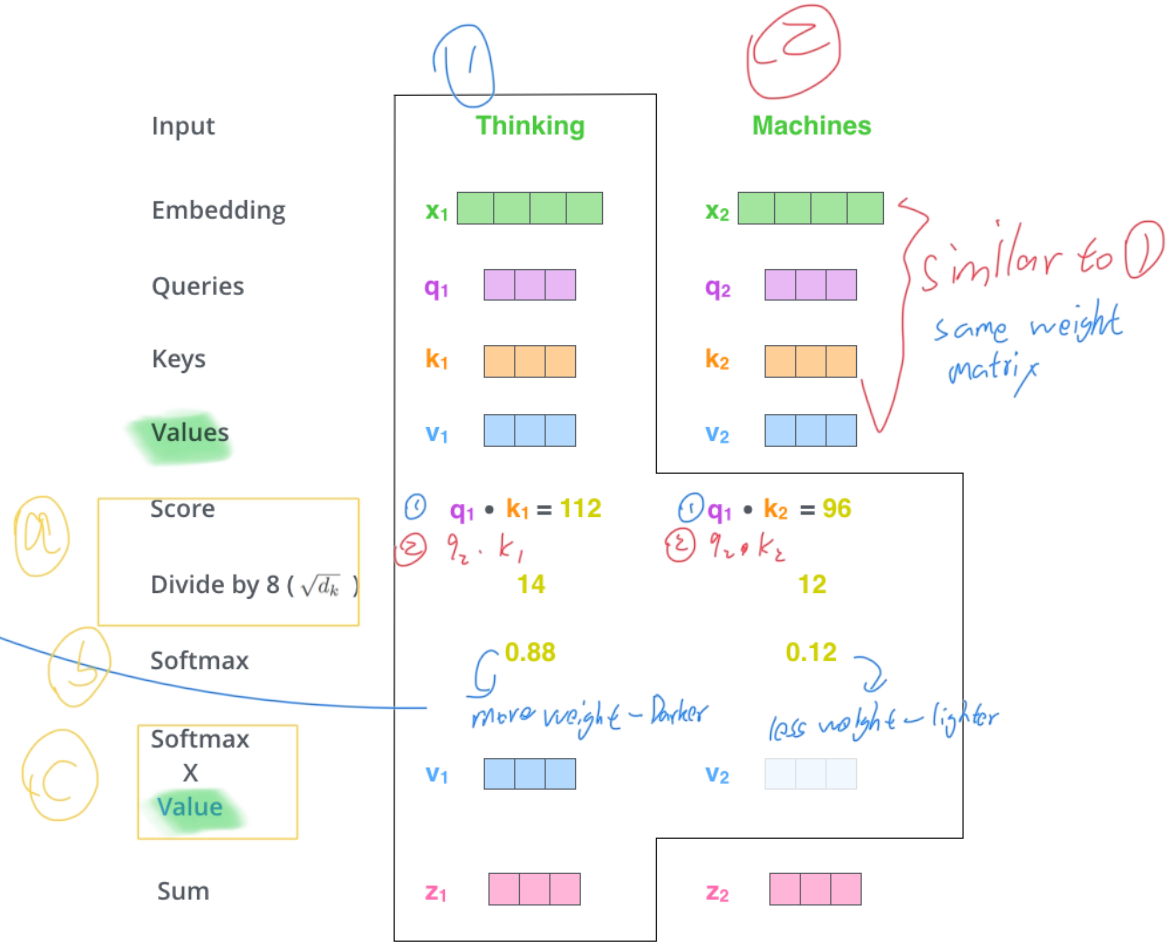
- **Step 2**: calculate a score
  - taking the dot product of the query vector with the key vector of the respective word we're scoring
- **Step 3:** divide the scores by 8
  - to have more stable gradients
- **Step 4:** apply softmax operation
  - normalizes the scores so they're all positive and add up to 1
  - determines how much each word will be expressed at this position



# Self-Attention in Detail

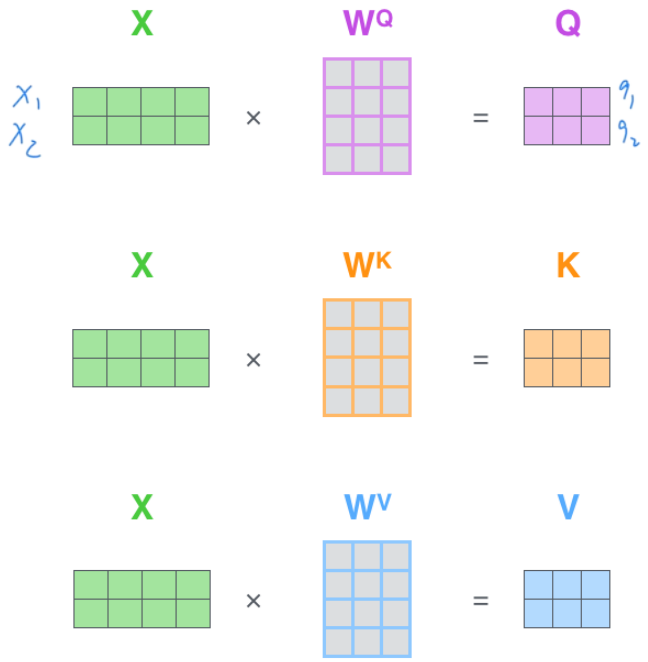
- **Step 5**: multiply each value vector by the softmax score
  - keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words
- **Step 6**: sum up the weighted value vectors
  - produces the output of the self-attention layer at this position (for the first word)

So $z_1 = v_1 + v_2$ ? ✓

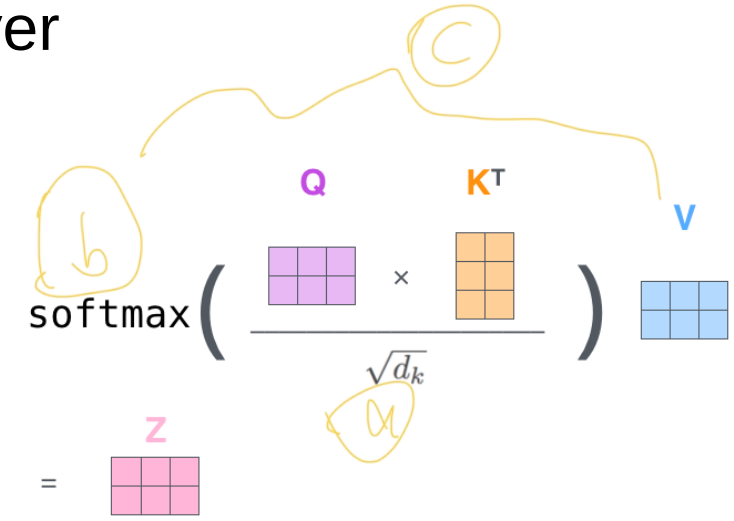


# Matrix Calculation of Self-Attention

- Packing embeddings into a matrix X and multiplying it by the weight matrices we've trained ($W^Q$, $W^K$, $W^V$)

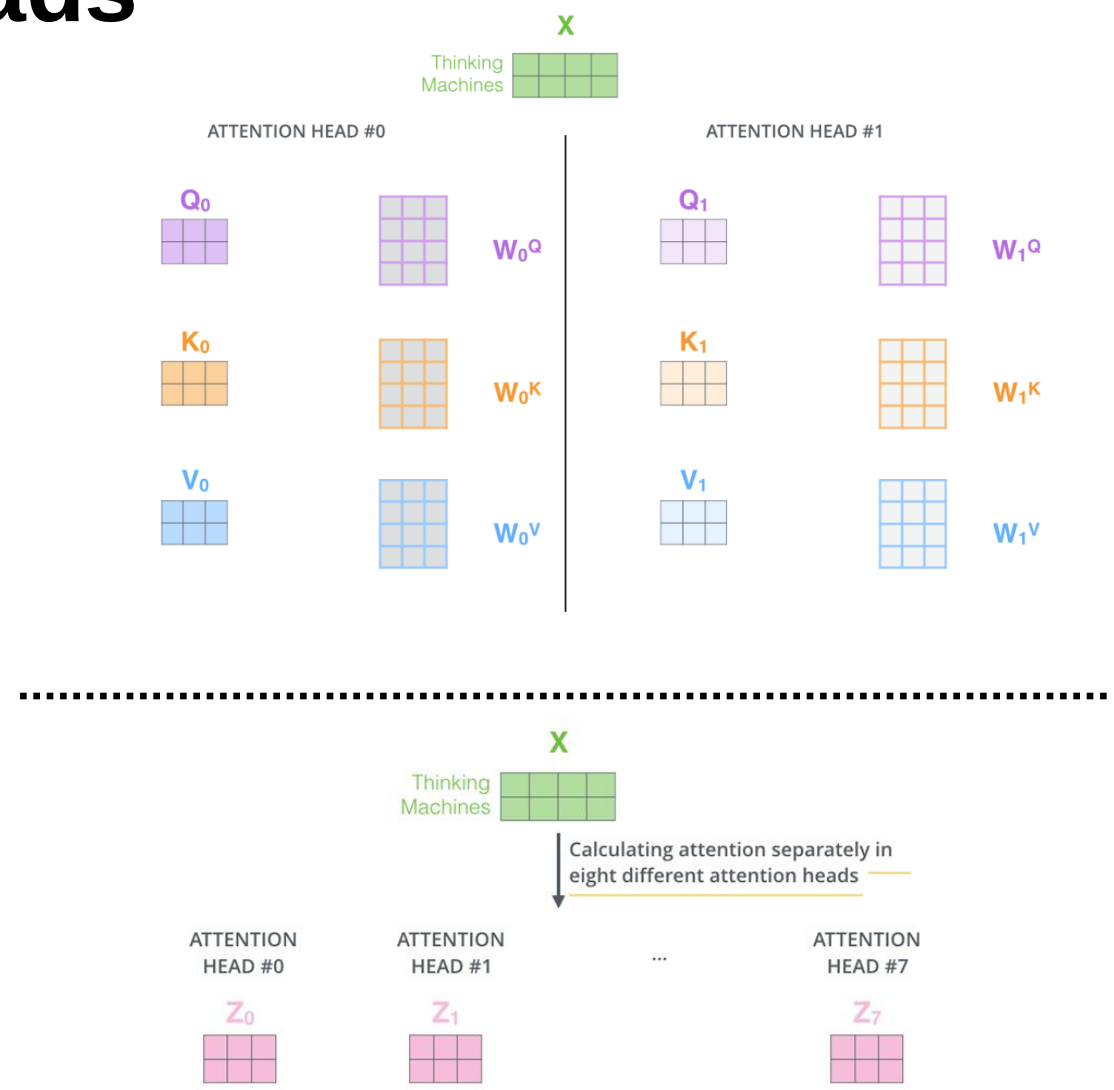



- Six-in-one formula to calculate the outputs of the self-attention layer



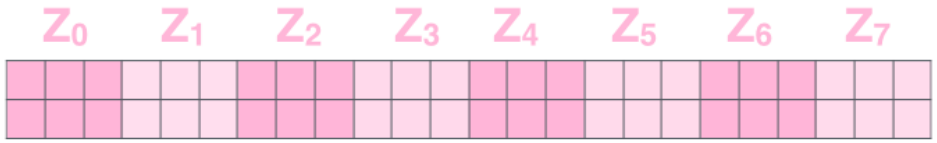
# The Beast With Many Heads

- “Multi-headed” attention
  - expands the model’s ability to focus on different positions.
  - gives the attention layer multiple “representation subspaces”.



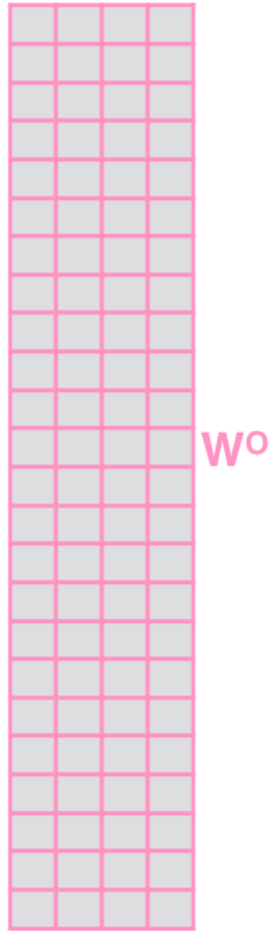
# The Beast With Many Heads

1) Concatenate all the attention heads



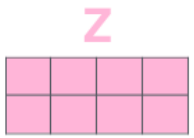

2) Multiply with a weight matrix $W^O$ that was trained jointly with the model

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

=



# The Beast With Many Heads

1) This is our input sentence*

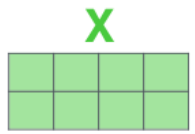
2) We embed each word*

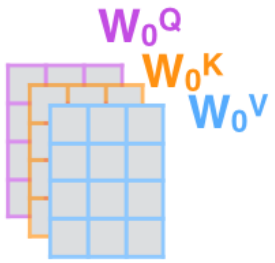
3) Split into 8 heads. We multiply X or R with weight matrices

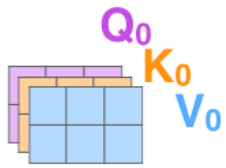
4) Calculate attention using the resulting Q/K/V matrices

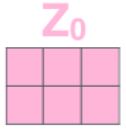
5) Concatenate the resulting Z matrices, then multiply with weight matrix $W^O$ to produce the output of the layer

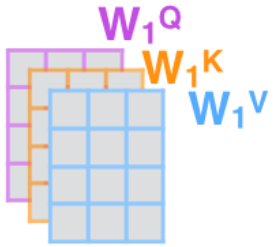
Thinking Machines

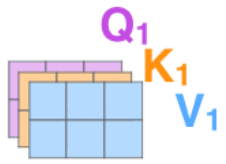


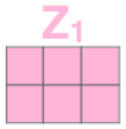



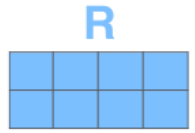



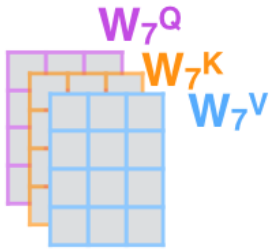



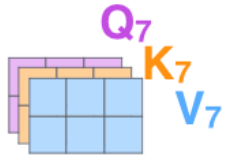

* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



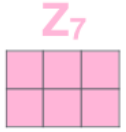

...

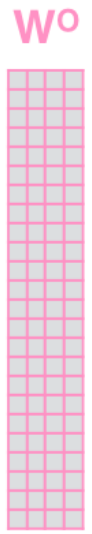


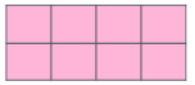

...



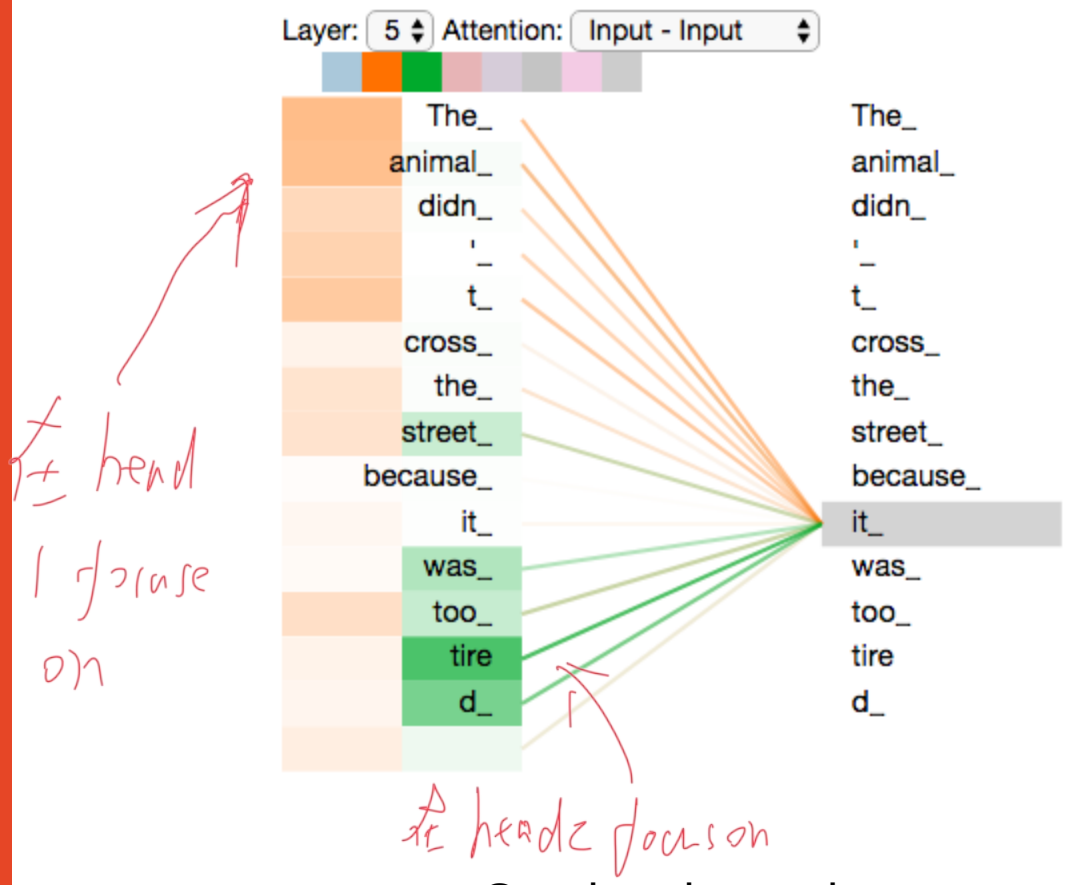

...



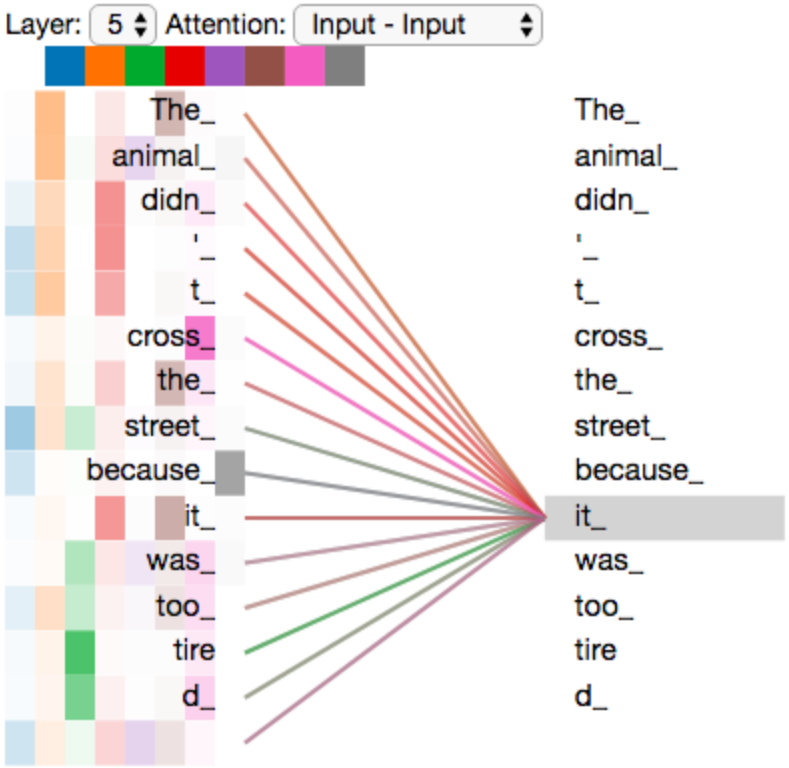



# The Beast With Many Heads



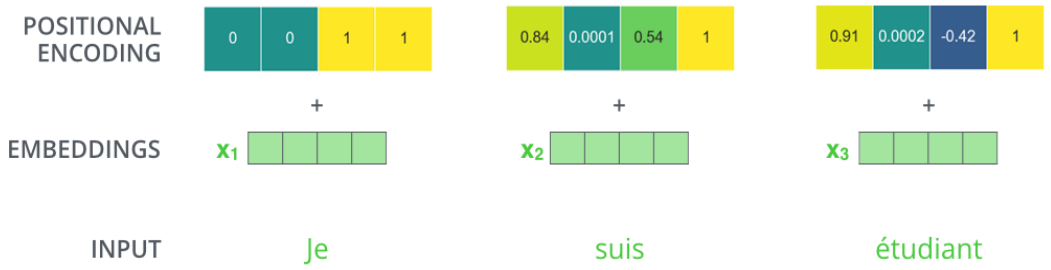

One-head attention

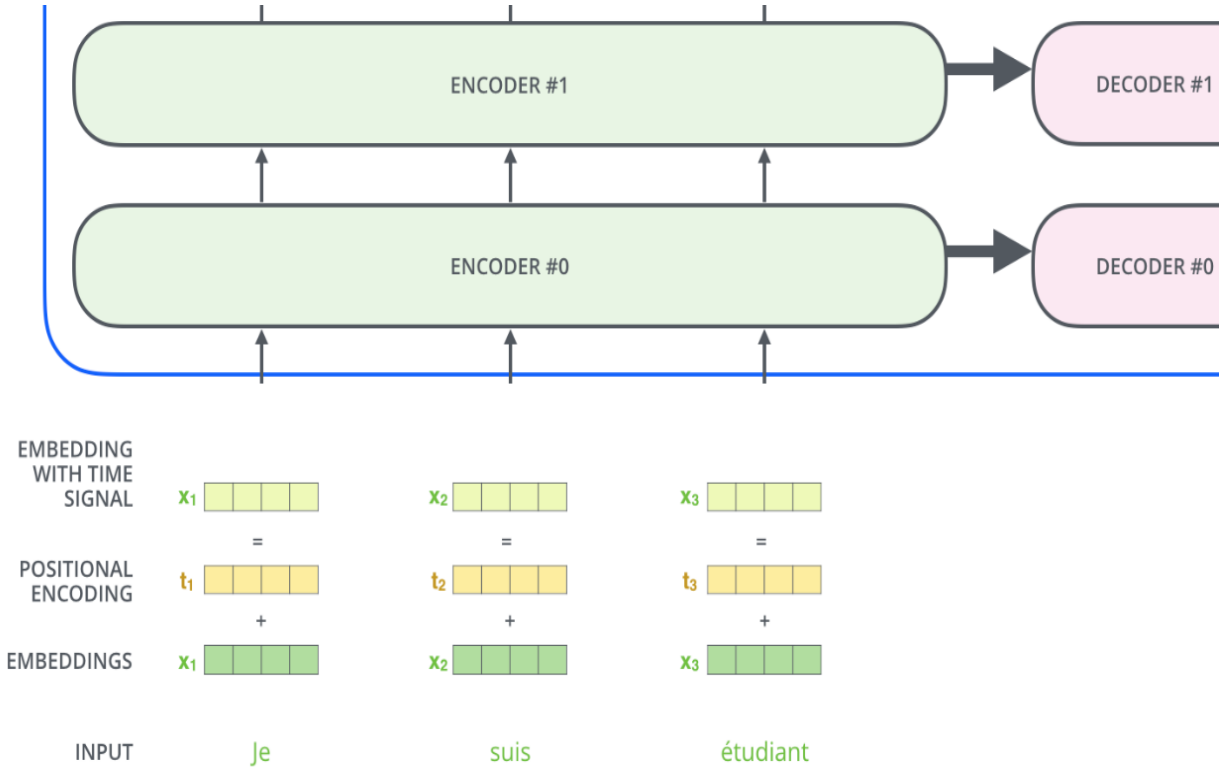



Multi-headed attention

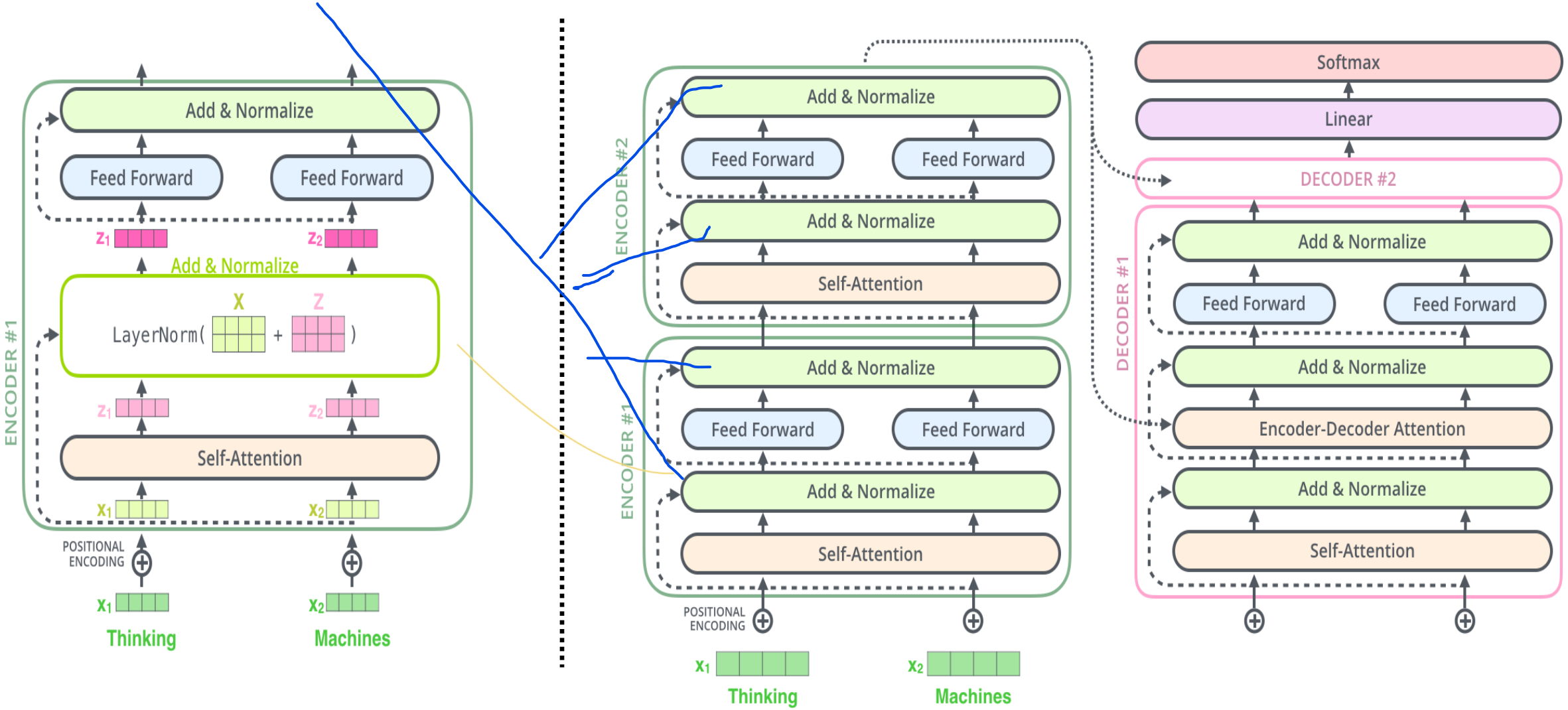
# Positional Encoding

- Provides meaningful distances between the embedding vectors once projected into Q/K/V vectors
- Determines the position of each word



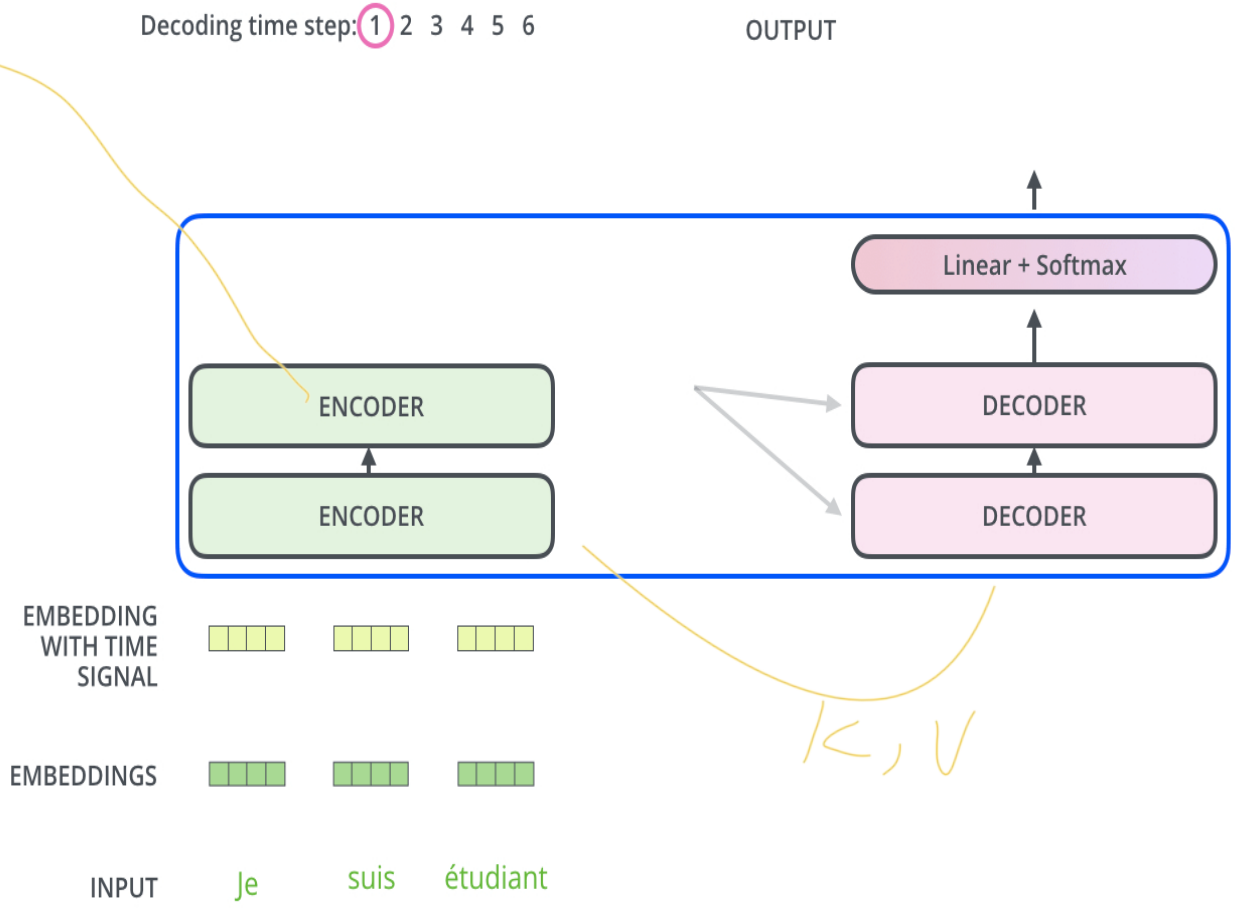



# The Residuals



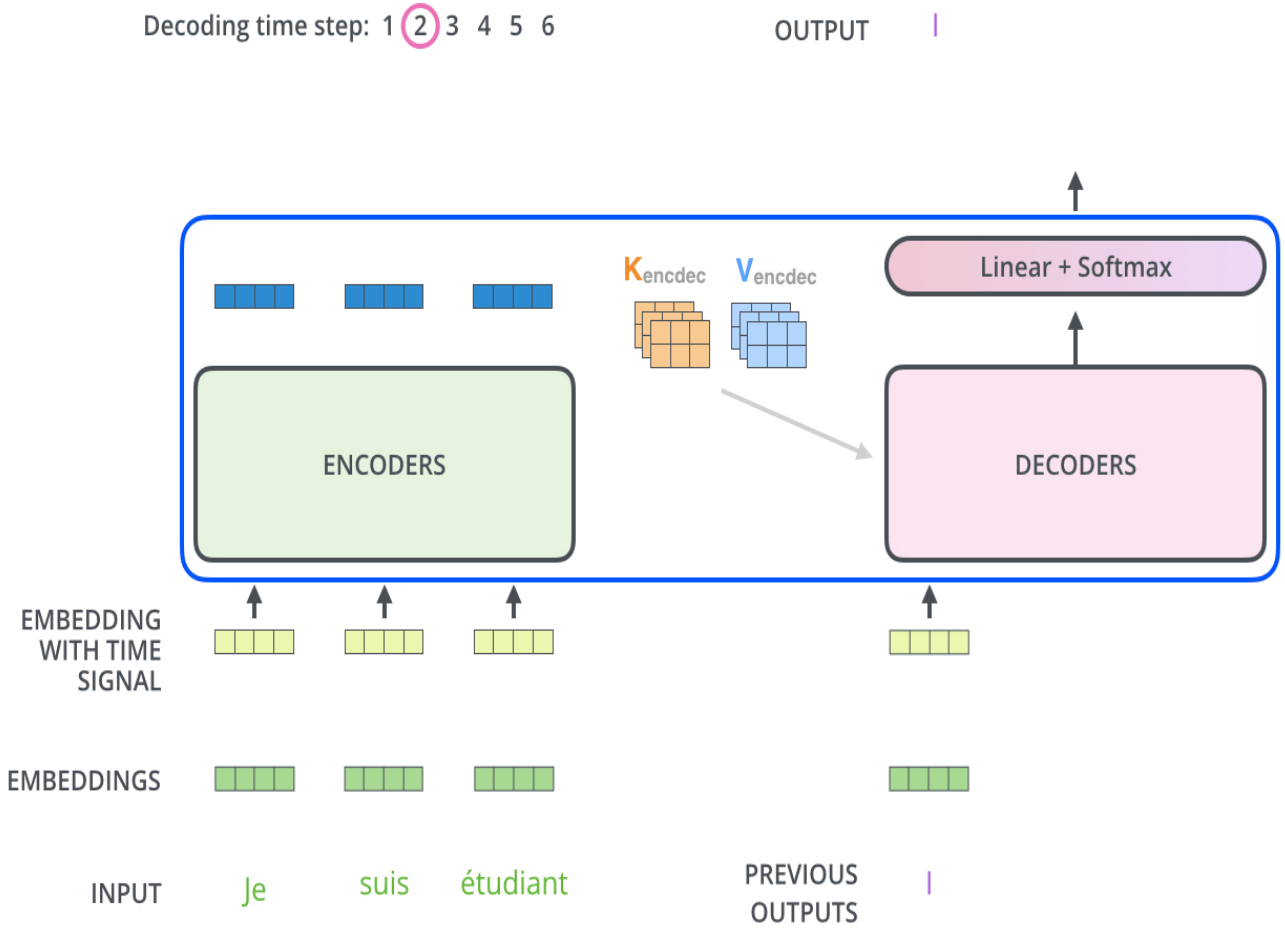
# The Decoder Side

- The output of the top encoder is transformed into a set of attention vectors K and V.
  - used in “encoder-decoder attention” layer of decoders
  - focus on appropriate places in the input sequence
- The output of each step is fed to the bottom decoder in the next time step

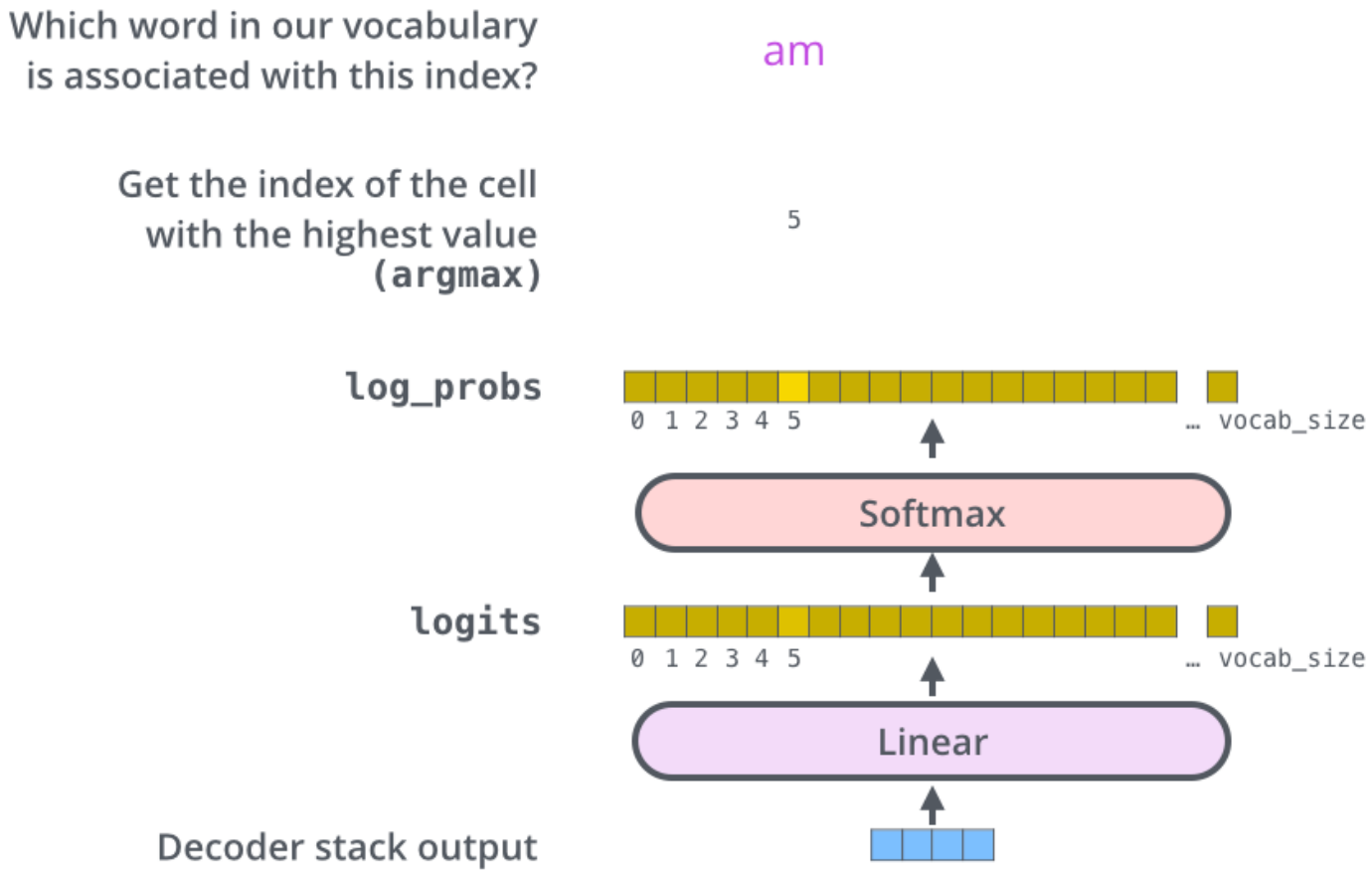


# The Decoder Side

- Embed and add positional encoding to decoder inputs to indicate the position of each word.
- Self-attention layer is only allowed to attend to earlier positions in the output sequence.
- “Encoder-Decoder Attention” layer works like multiheaded self-attention



# The Decoder Side: Final Linear and Softmax Layer



# Recap Of Training

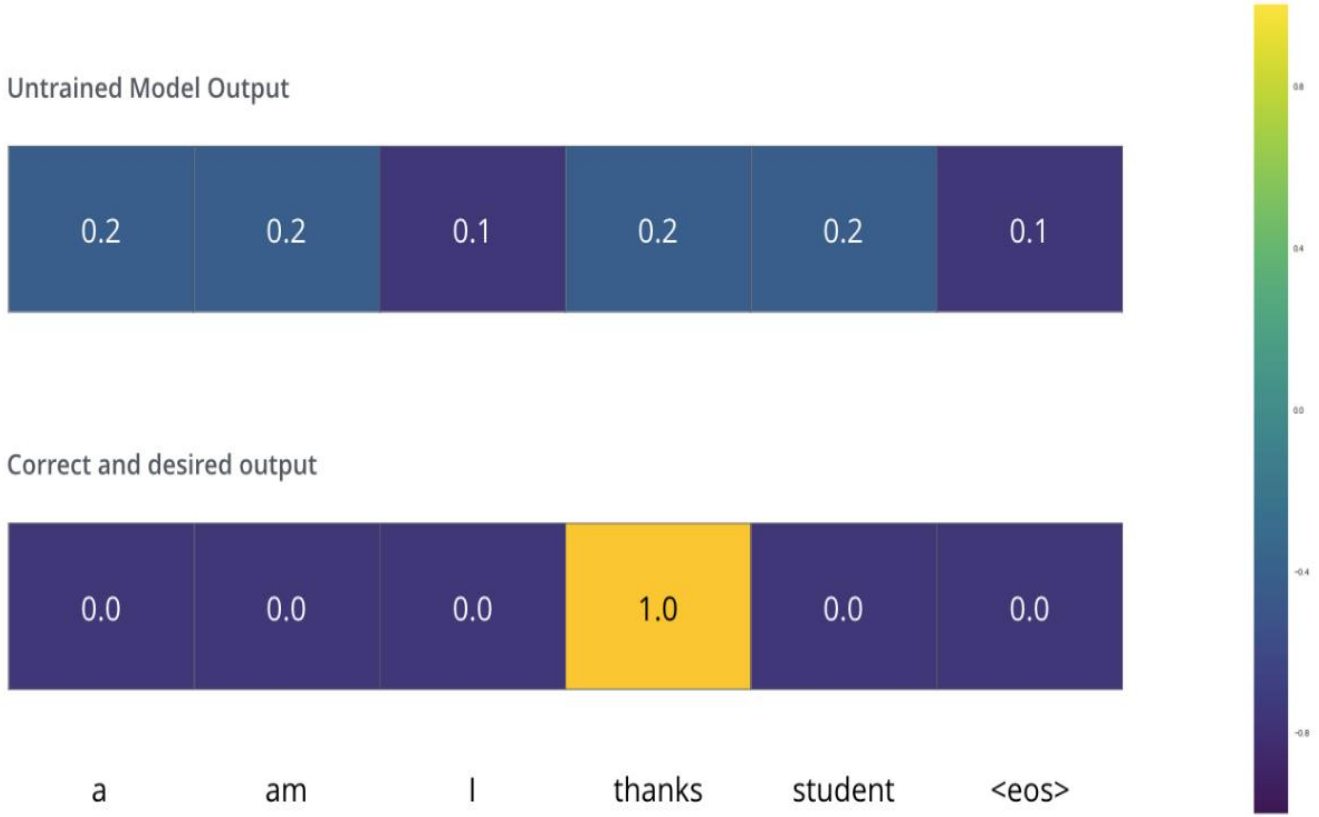
Output Vocabulary

| WORD | a | am | I | thanks | student | <eos> |
|---|---|---|---|---|---|---|
| INDEX | 0 | 1 | 2 | 3 | 4 | 5 |

One-hot encoding of the word “am”

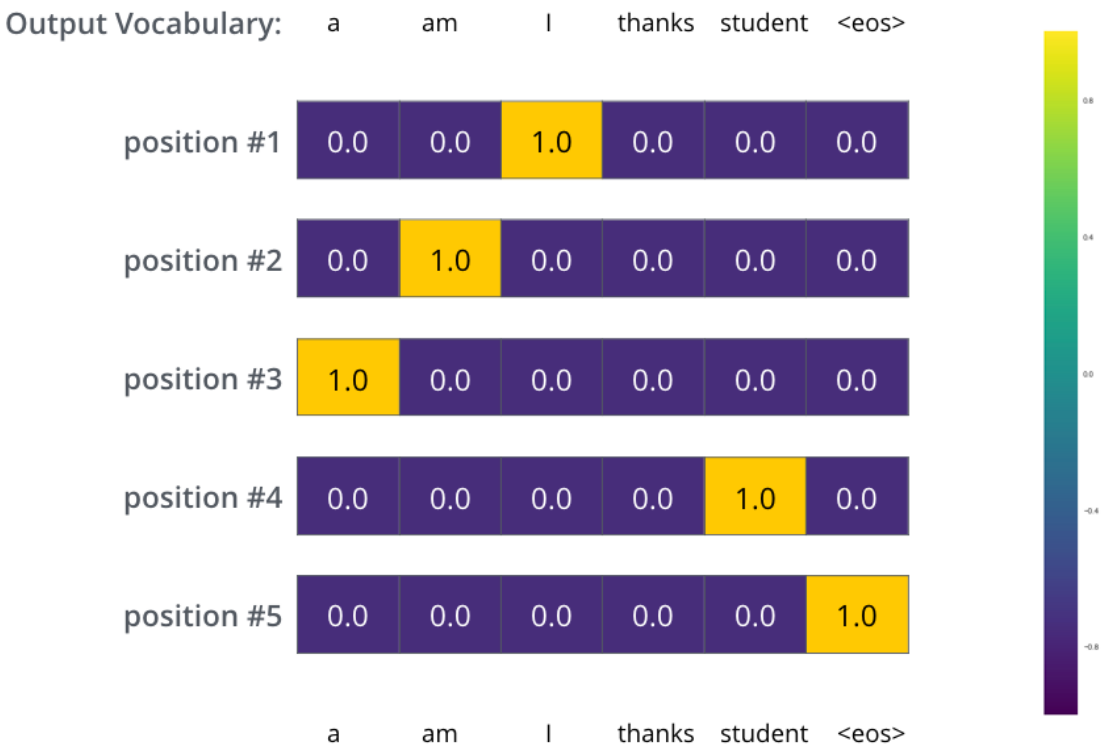
# The Loss Function

- The (untrained) model produces a probability distribution with arbitrary values for each cell/word
  - using backpropagation to make the output closer to the actual output.
- Comparing two probability distributions
  - Cross-Entropy
  - Kullback-Leiber Divergence

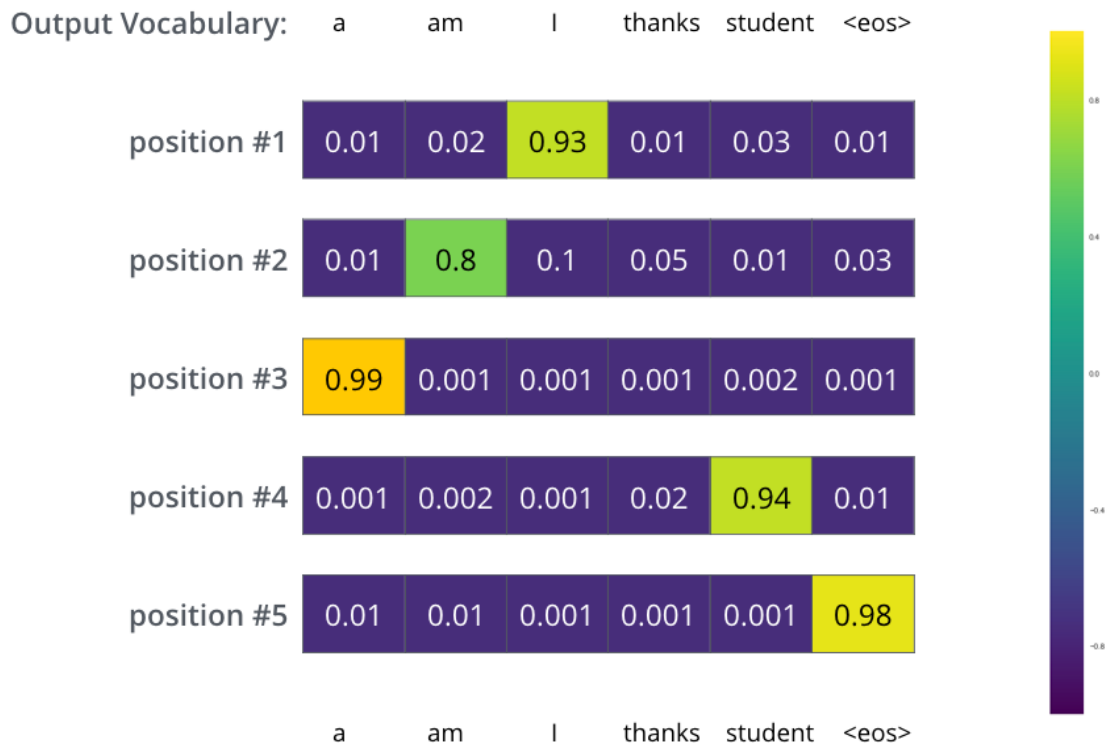


# The Loss Function

## Target Model Outputs

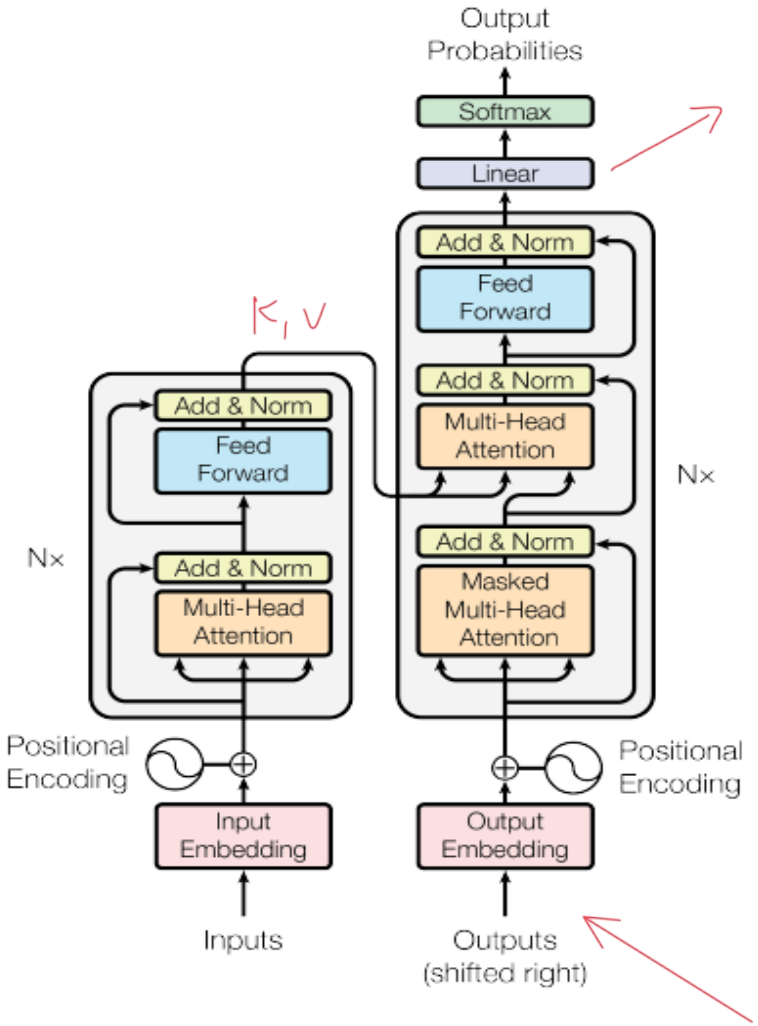



## Trained Model Outputs



# Transformer

Encoder




Decoder

Vaswani et al. (2017), *Attention is all you need.*

# Why Transformers?

### Downsides

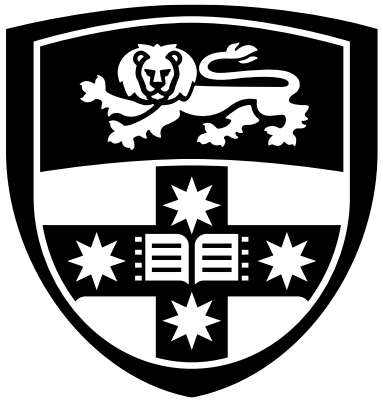
- Attention computations are technically $O(n^2)$
- Somewhat more complex to implement (positional encodings, etc.)

### Benefits

- Much better long-range connections
- Much easier to parallelize
- In practice, can make it much deeper (more layers) than RNN

The benefits seem to vastly outweigh the downsides, and transformers work much better than RNNs (and LSTMs) in many cases

Arguably one of the most important sequence modelling improvements of the past decade



THE UNIVERSITY OF
SYDNEY