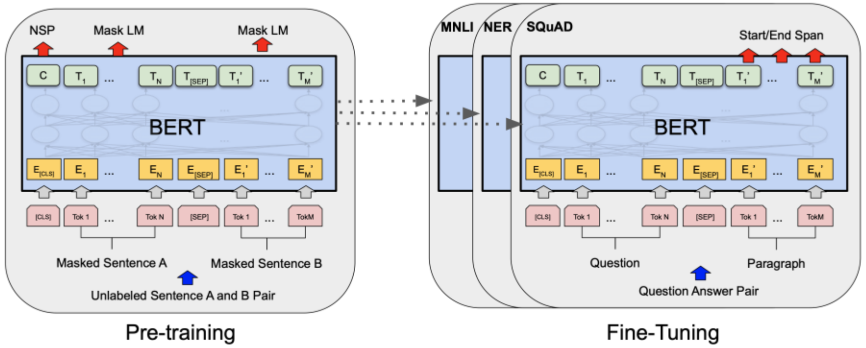


بنام خداوند جان و خرد

Transformers and BERT

Mohammad Taher Pilehvar

Deep Learning 99

https://teias-courses.github.io/dl99/

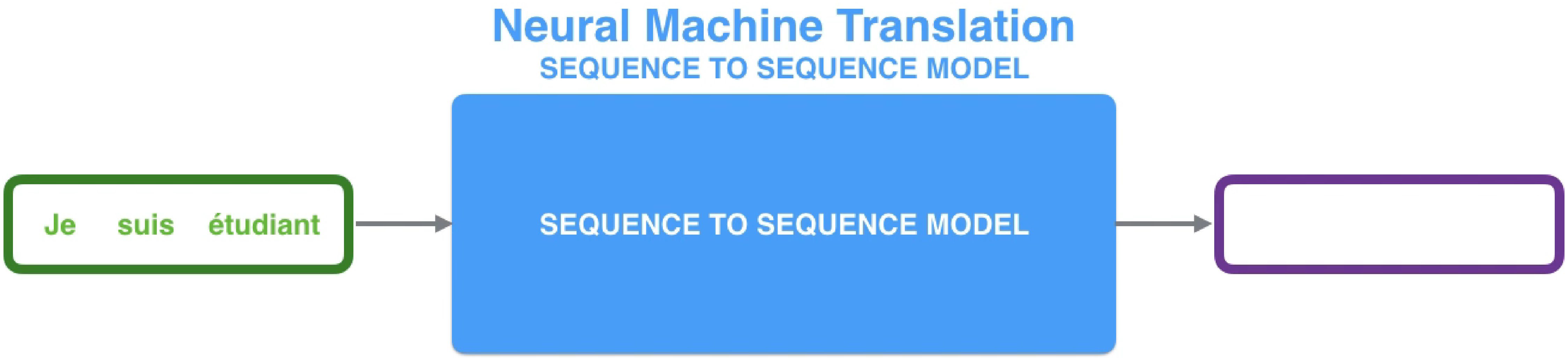
Most materials of these slides are taken from Jay Alammar's [blog post]

Attention

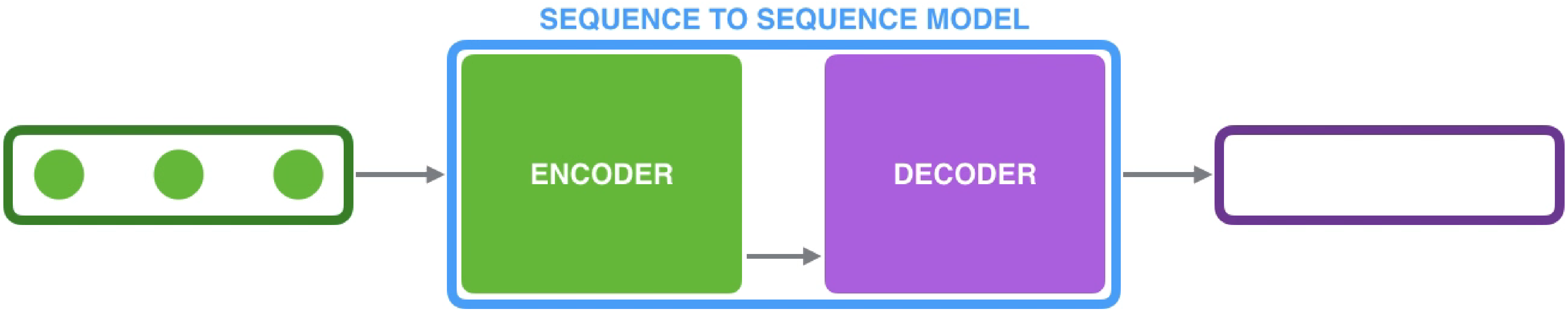
Sequence to Sequence model

Attention

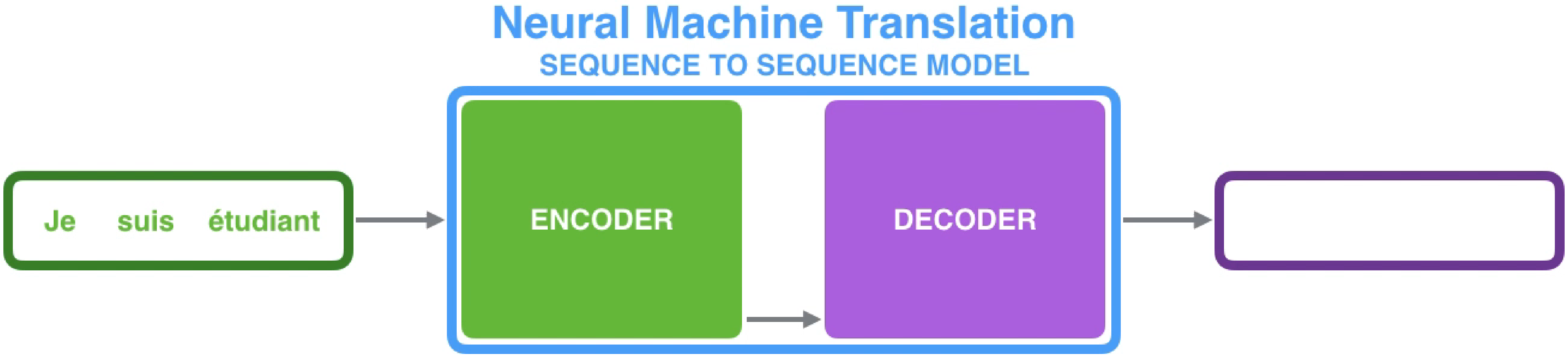
Sequence to Sequence model



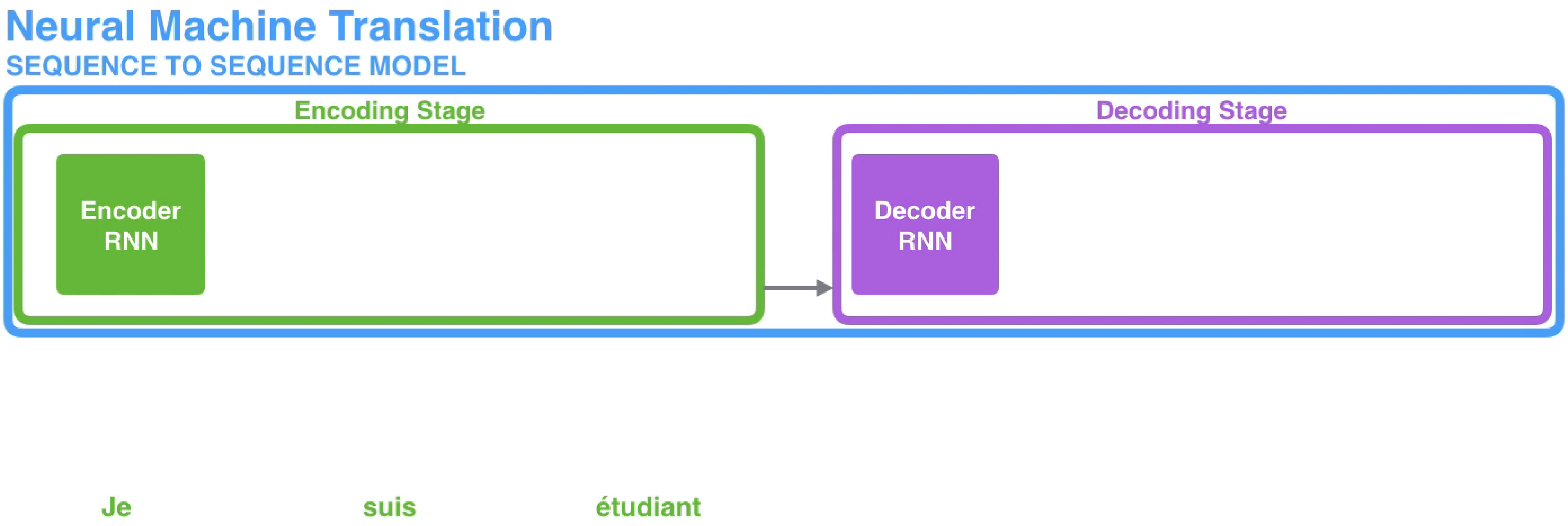
Attention



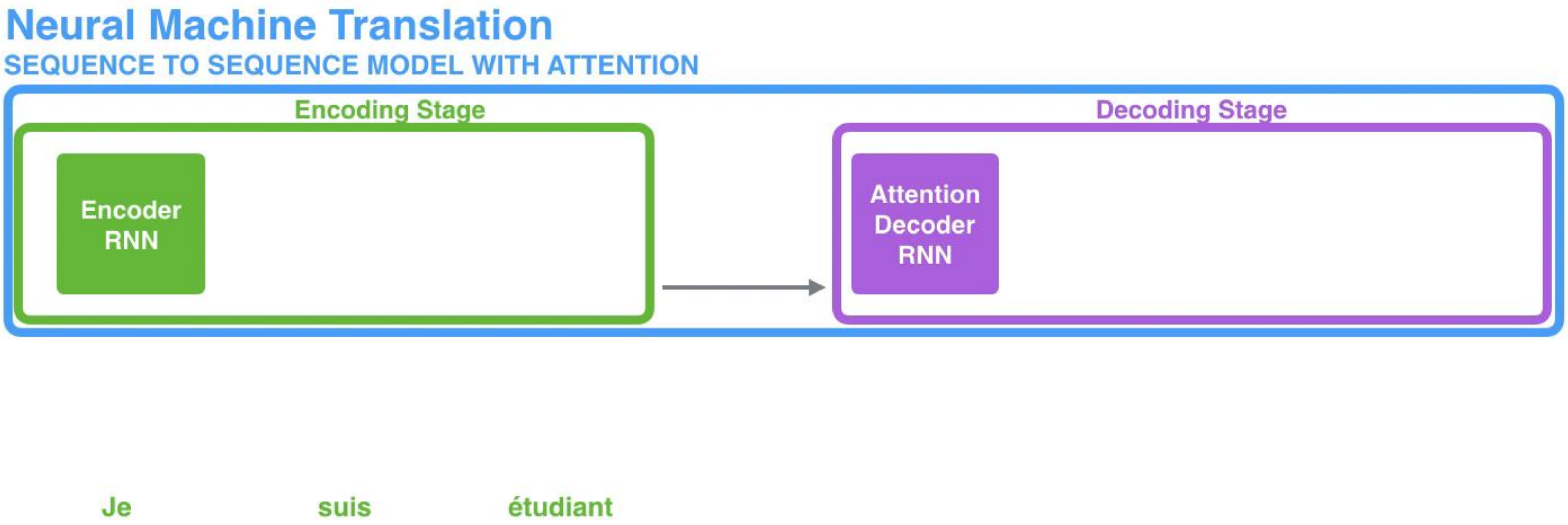
Attention


Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL
Je suis étudiant
ENCODER
DECODER

Attention


Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL
Encoding Stage
Decoding Stage
Encoder RNN
Decoder RNN
Je
suis
étudiant

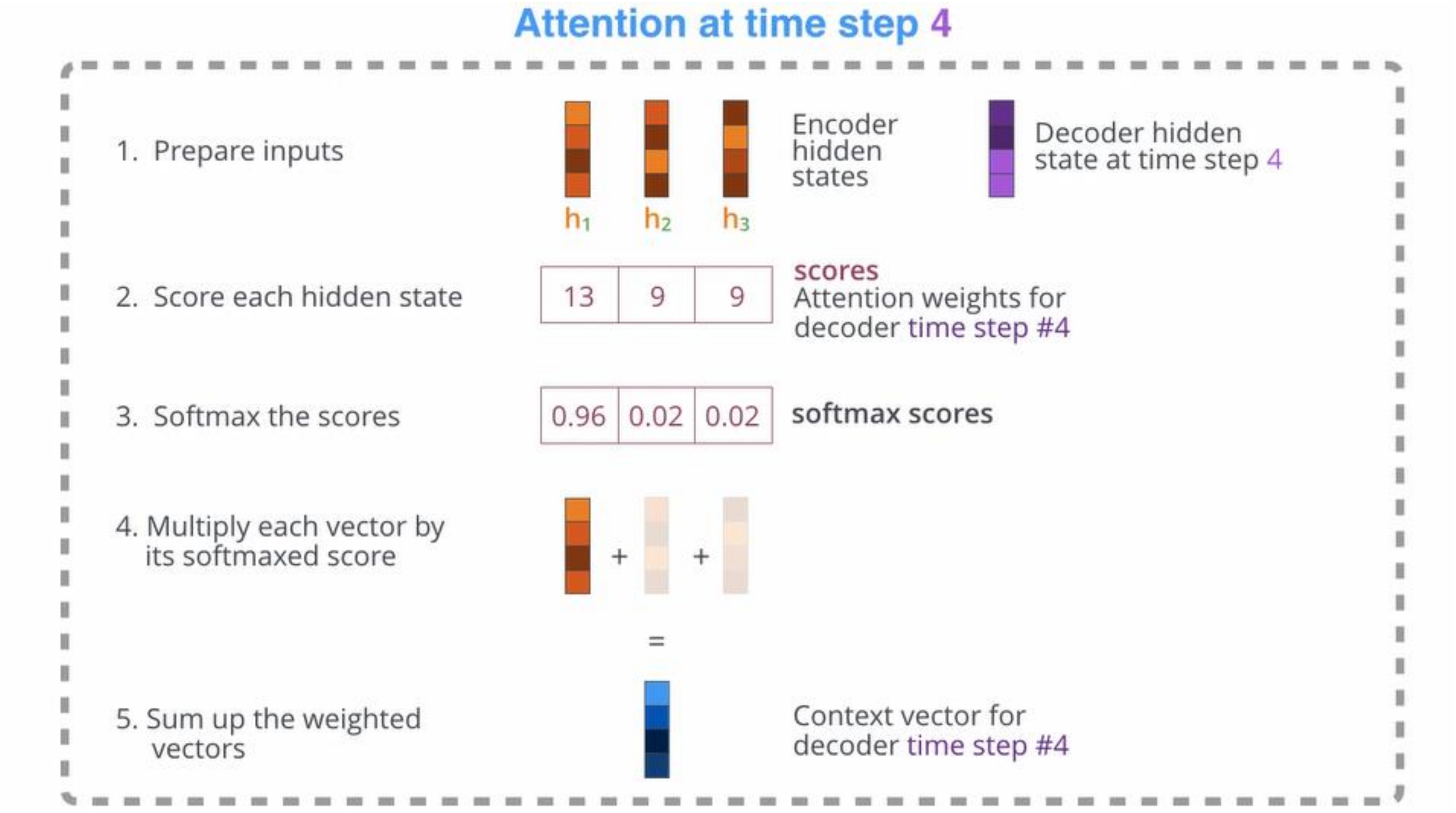
Attention


Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL WITH ATTENTION
Encoding Stage
Decoding Stage
Encoder RNN
Attention Decoder RNN
Je
suis
étudiant

Attention

1. Look at the set of encoder hidden states it received – each encoder hidden states is most associated with a certain word in the input sentence
2. Give each hidden states a score (let's ignore how the scoring is done for now)
3. Multiply each hidden state by its softmaxed score, thus amplifying hidden states with high scores, and drowning out hidden states with low scores

Attention

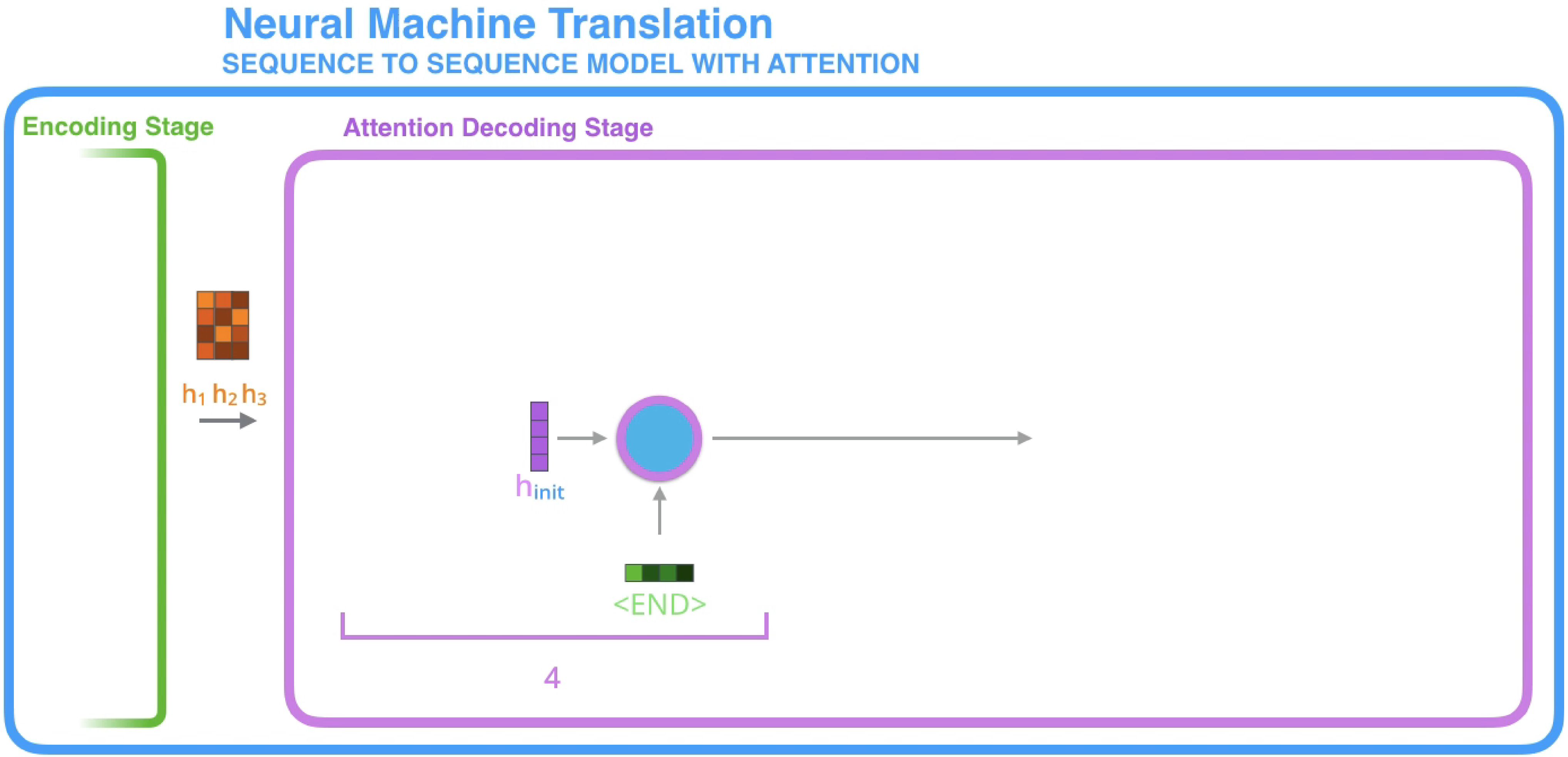


Attention at time step 4
1. Prepare inputs
Encoder hidden states
Decoder hidden state at time step 4
h1
h2
h3
2. Score each hidden state
13
9
9
scores
Attention weights for decoder time step #4
3. Softmax the scores
0.96
0.02
0.02
softmax scores
4. Multiply each vector by its softmaxed score
+
+
=
5. Sum up the weighted vectors
Context vector for decoder time step #4

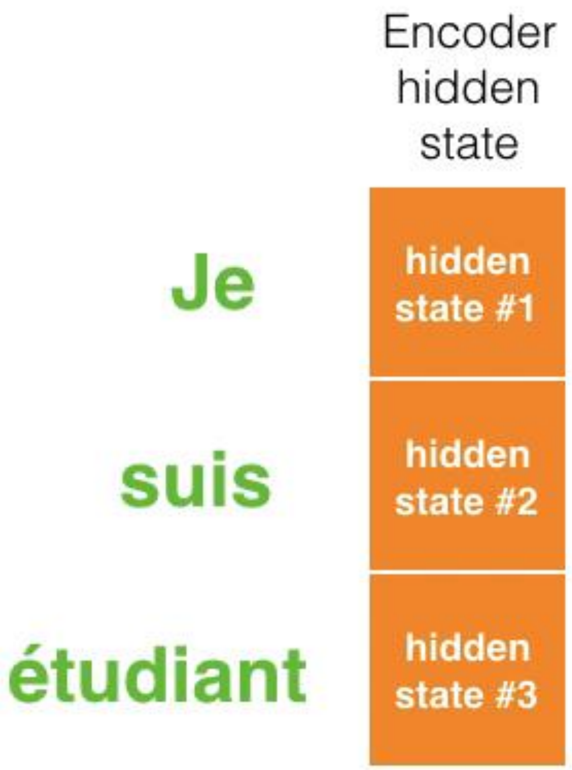
Attention

1. The attention decoder RNN takes in the embedding of the <END> token, and an initial decoder hidden state.
2. The RNN processes its inputs, producing an output and a new hidden state vector (h4). The output is discarded.
3. Attention Step: We use the encoder hidden states and the h4 vector to calculate a context vector (C4) for this time step.
4. We concatenate h4 and C4 into one vector.
5. We pass this vector through a feedforward neural network (one trained jointly with the model).
6. The output of the feedforward neural networks indicates the output word of this time step.
7. Repeat for the next time steps

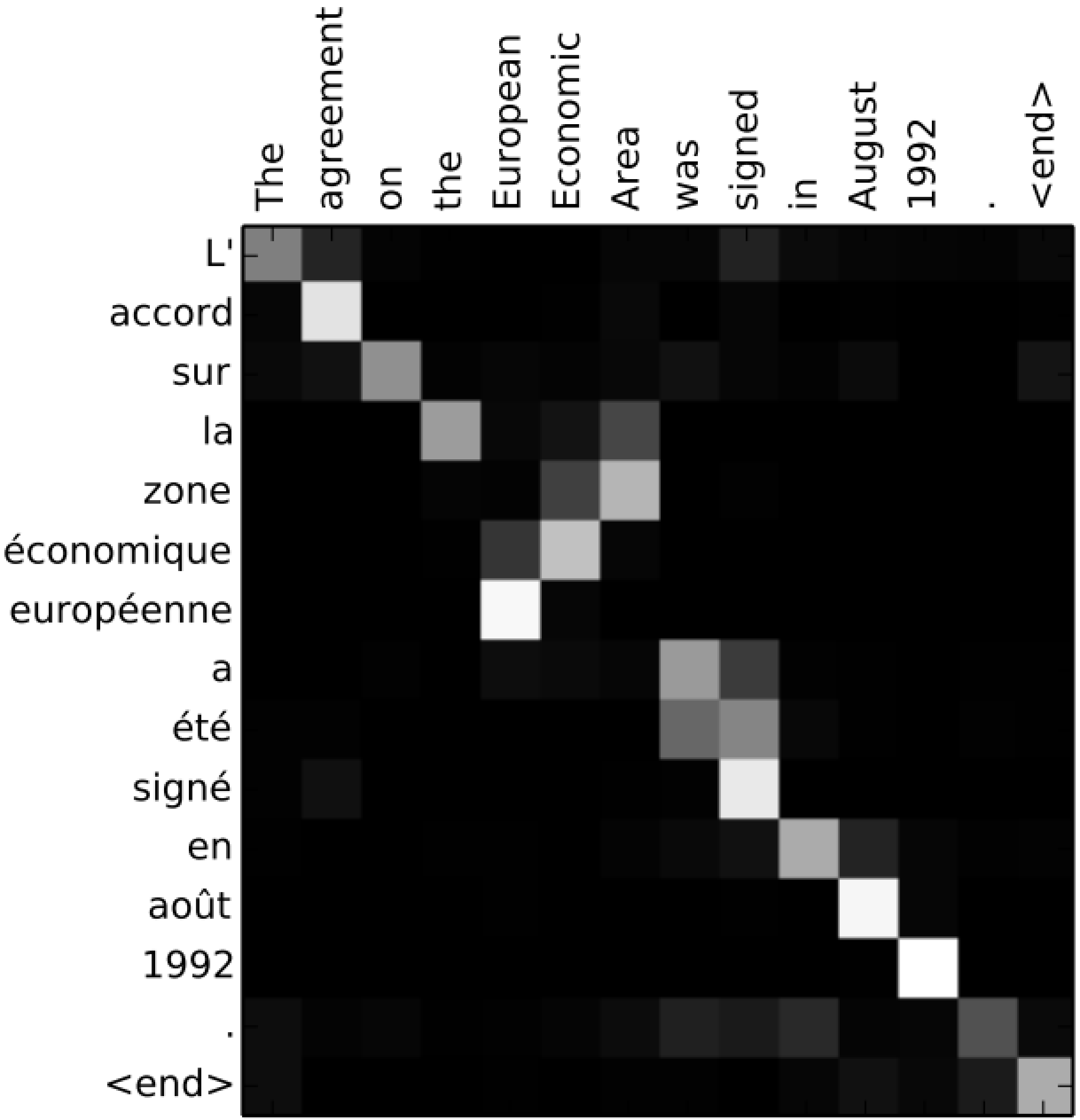
Attention


Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL WITH ATTENTION
Encoding Stage
Attention Decoding Stage
h1 h2 h3
hinit
<END>
4

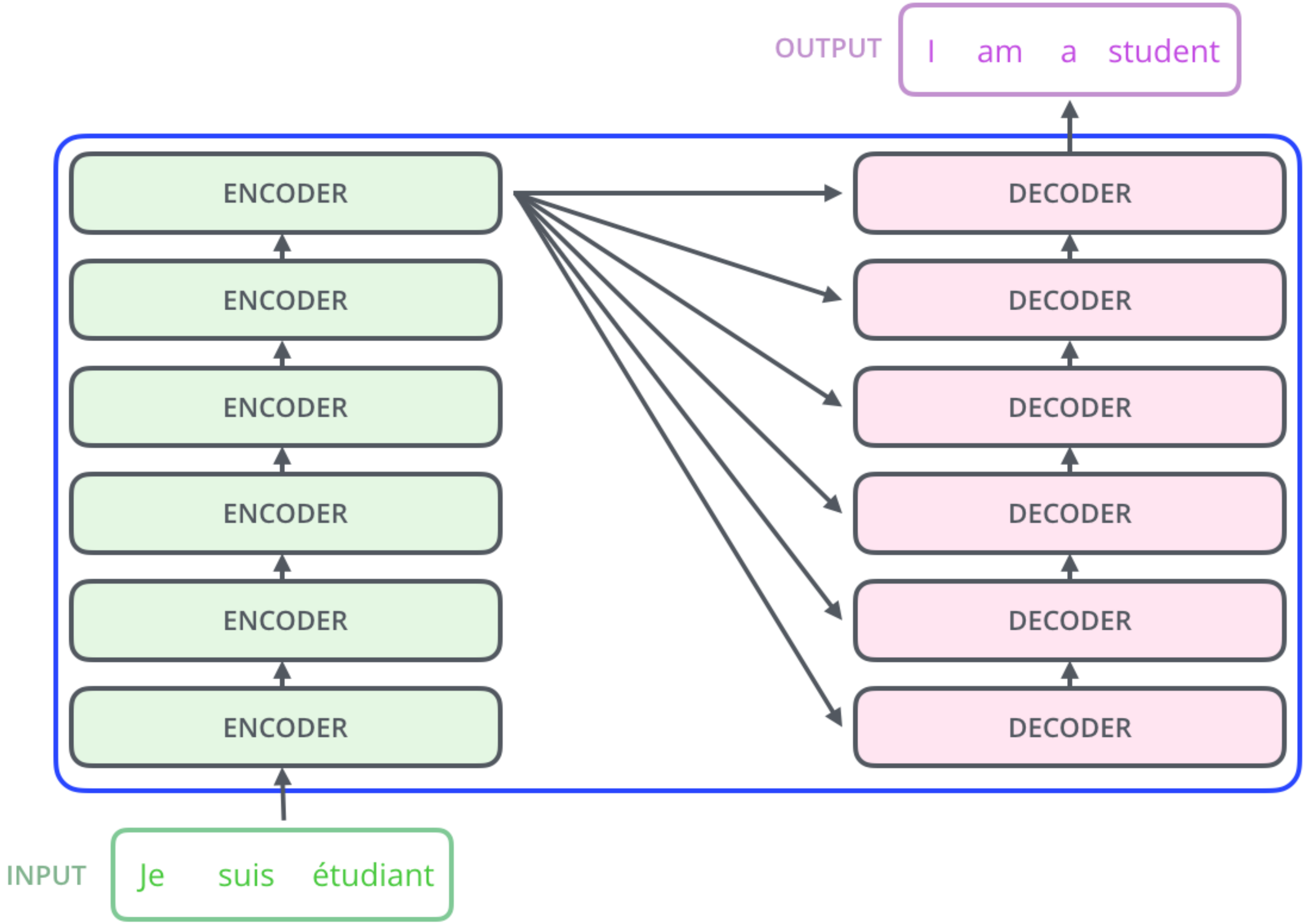
Attention


Encoder
hidden
state
Je
suis
étudiant
hidden
state #1
hidden
state #2
hidden
state #3

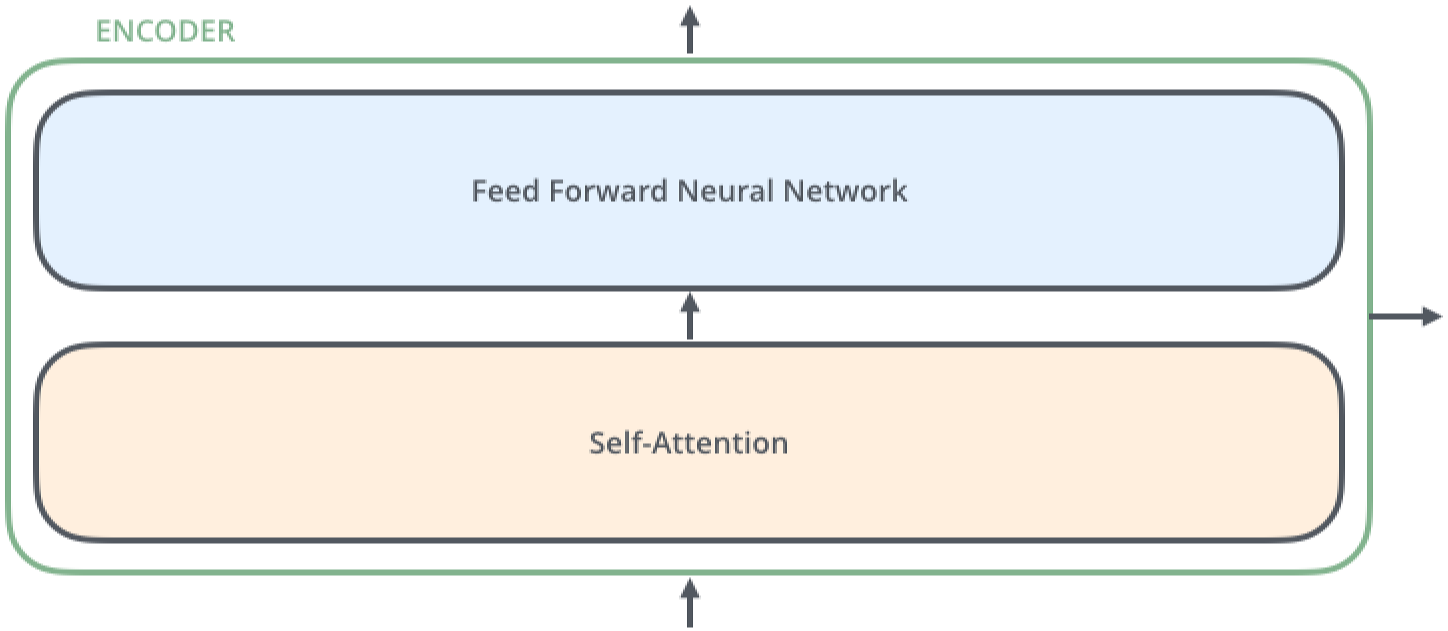
Attention


The
agreement
on
the
European
Economic
Area
was
signed
in
August
1992
.
<end>
L'
accord
sur
la
zone
économique
européenne
a
été
signé
en
août
1992
.
<end>

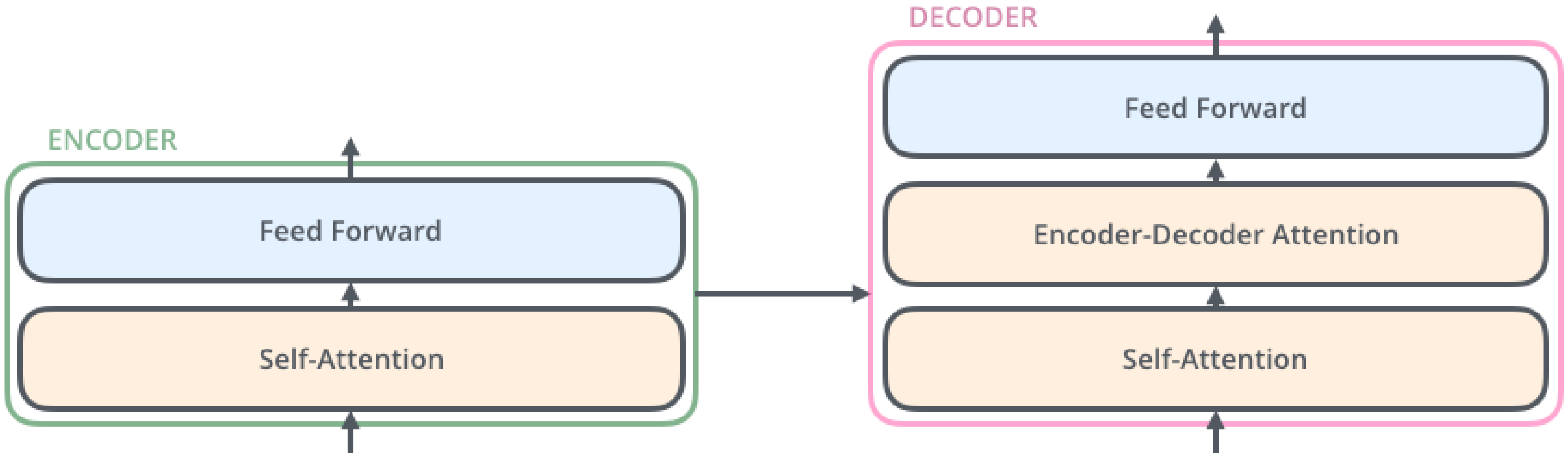
Transformers



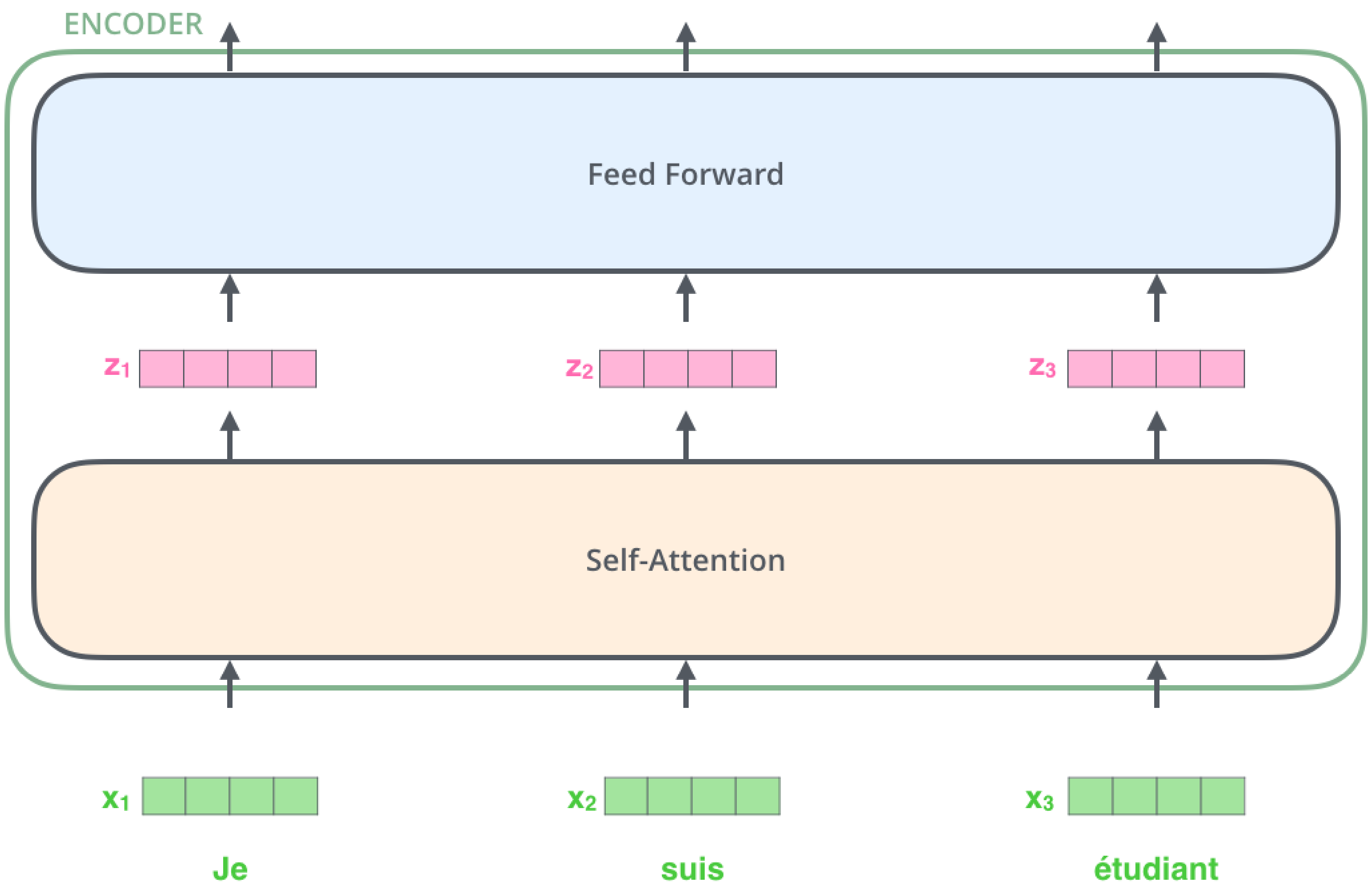
Transformer
Encoder



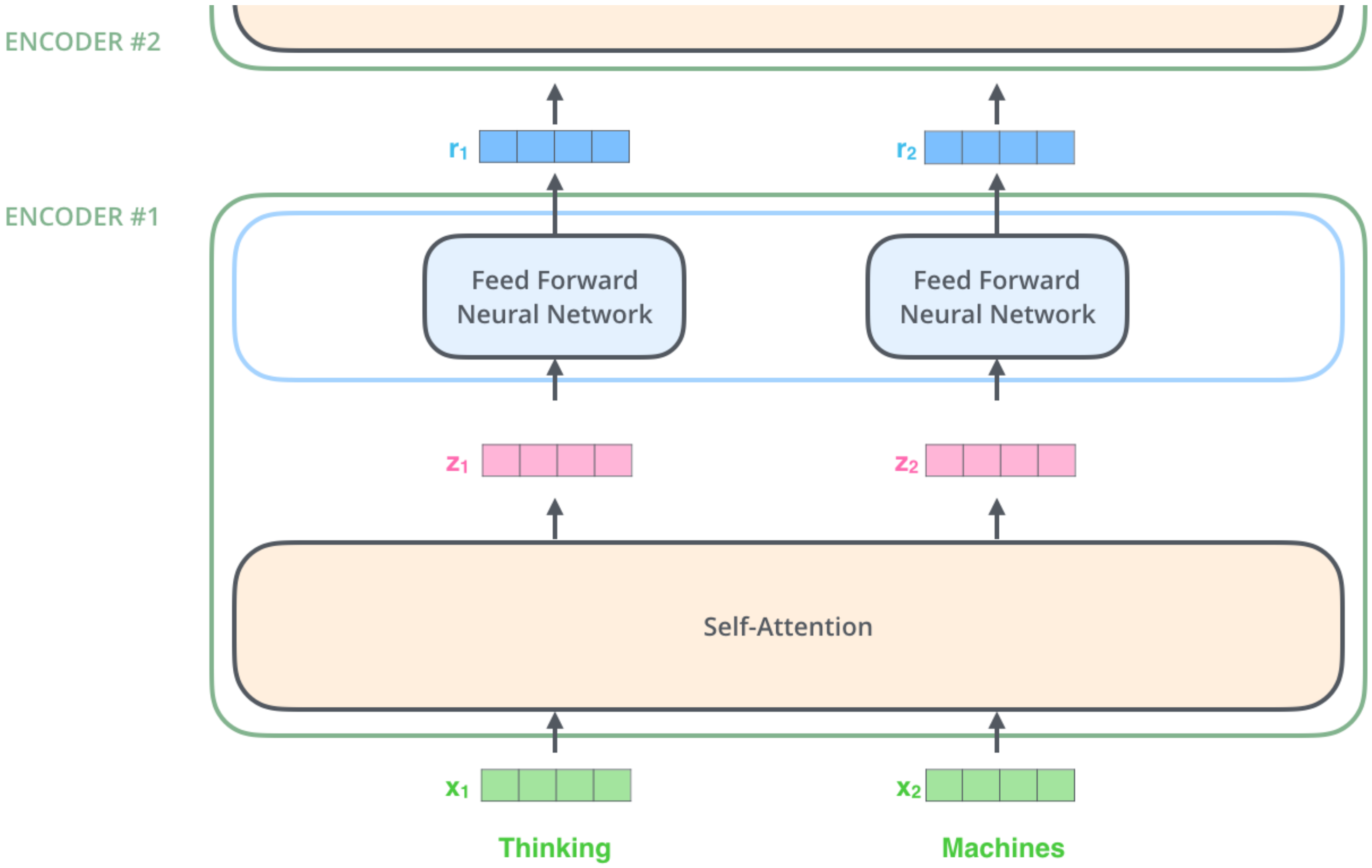
Transformer
Encoder - Decoder



Encoder

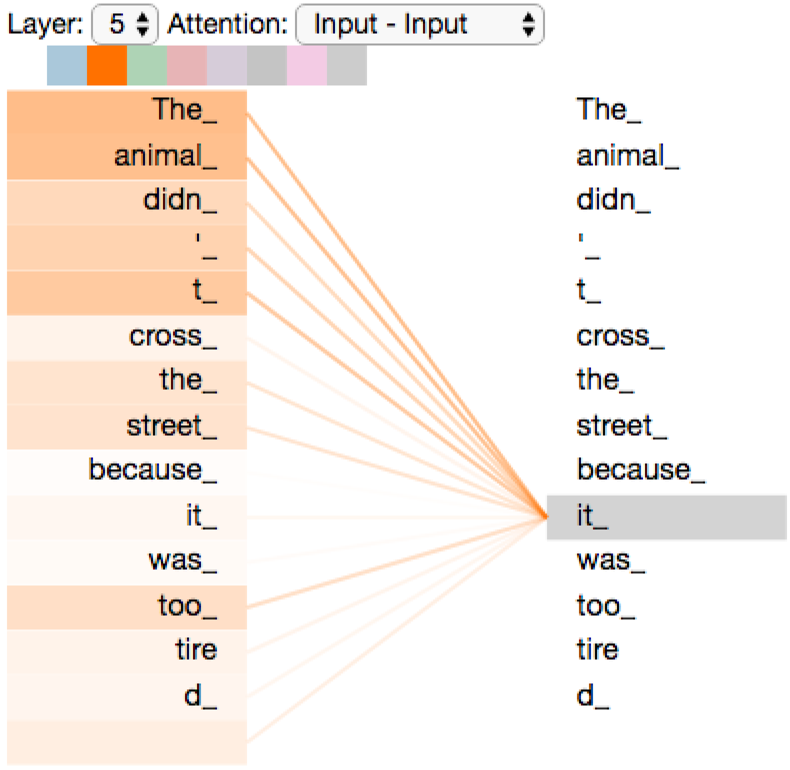

ENCODER
Feed Forward
z_1
z_2
z_3
Self-Attention
x_1
x_2
x_3
Je
suis
étudiant

Encoder

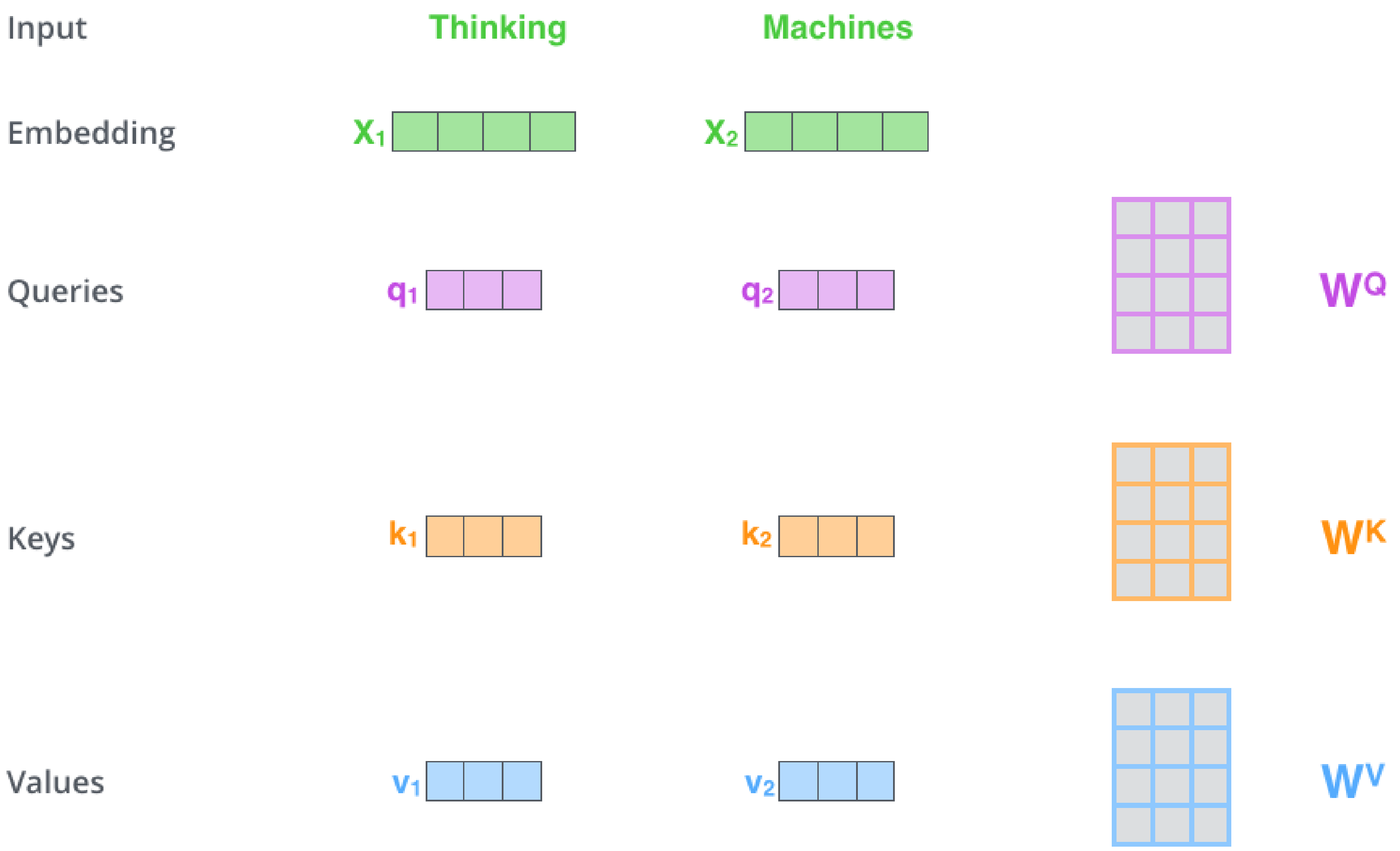

ENCODER #2
r_1
r_2
ENCODER #1
Feed Forward Neural Network
Feed Forward Neural Network
z_1
z_2
Self-Attention
x_1
x_2
Thinking
Machines

Self attention

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



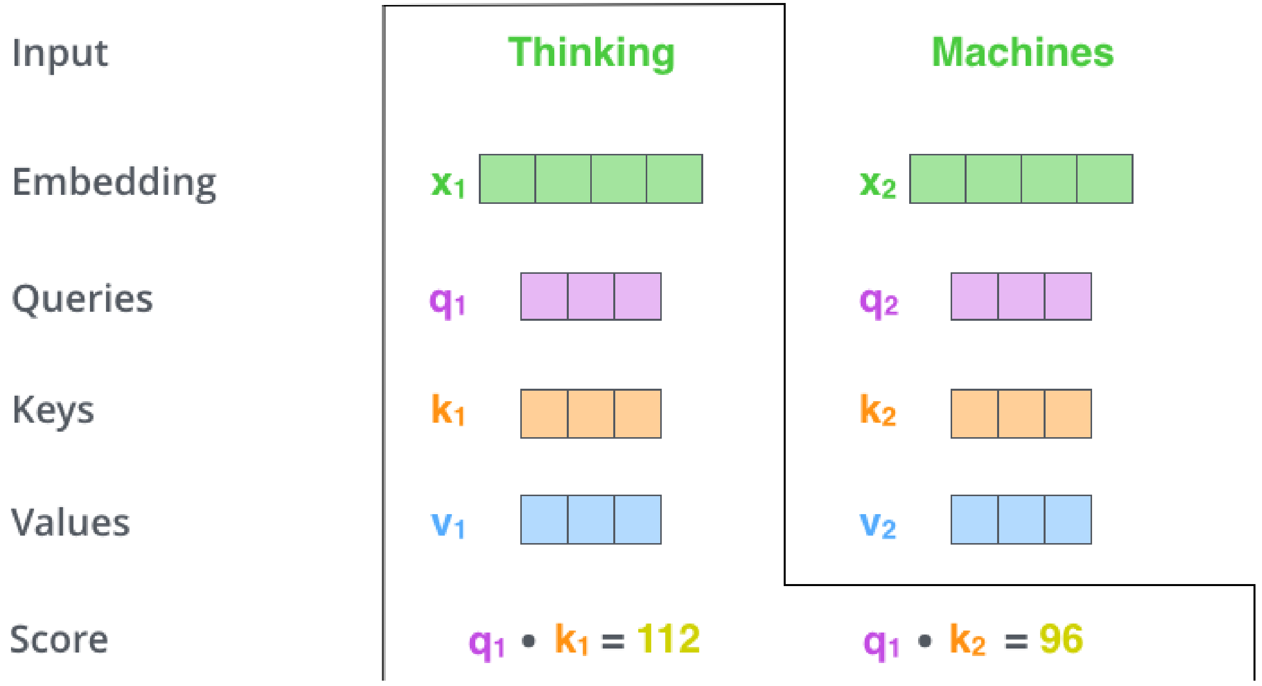
Self attention



Self attention

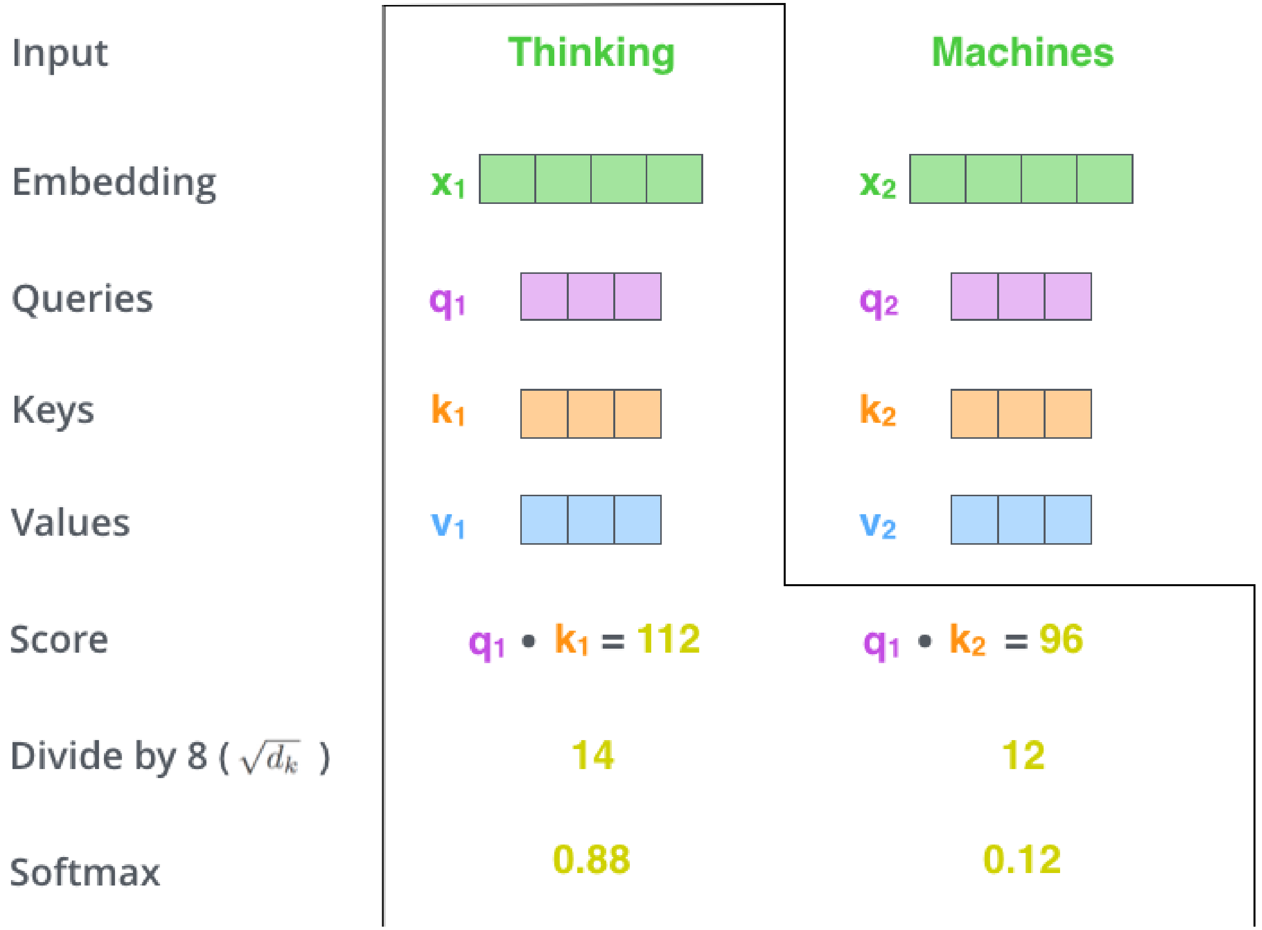
For first word: *thinking*

For each word, we create a *Query* vector, a *Key* vector, and a *Value* vector



Self attention

Divide the scores by 8, then pass through a softmax

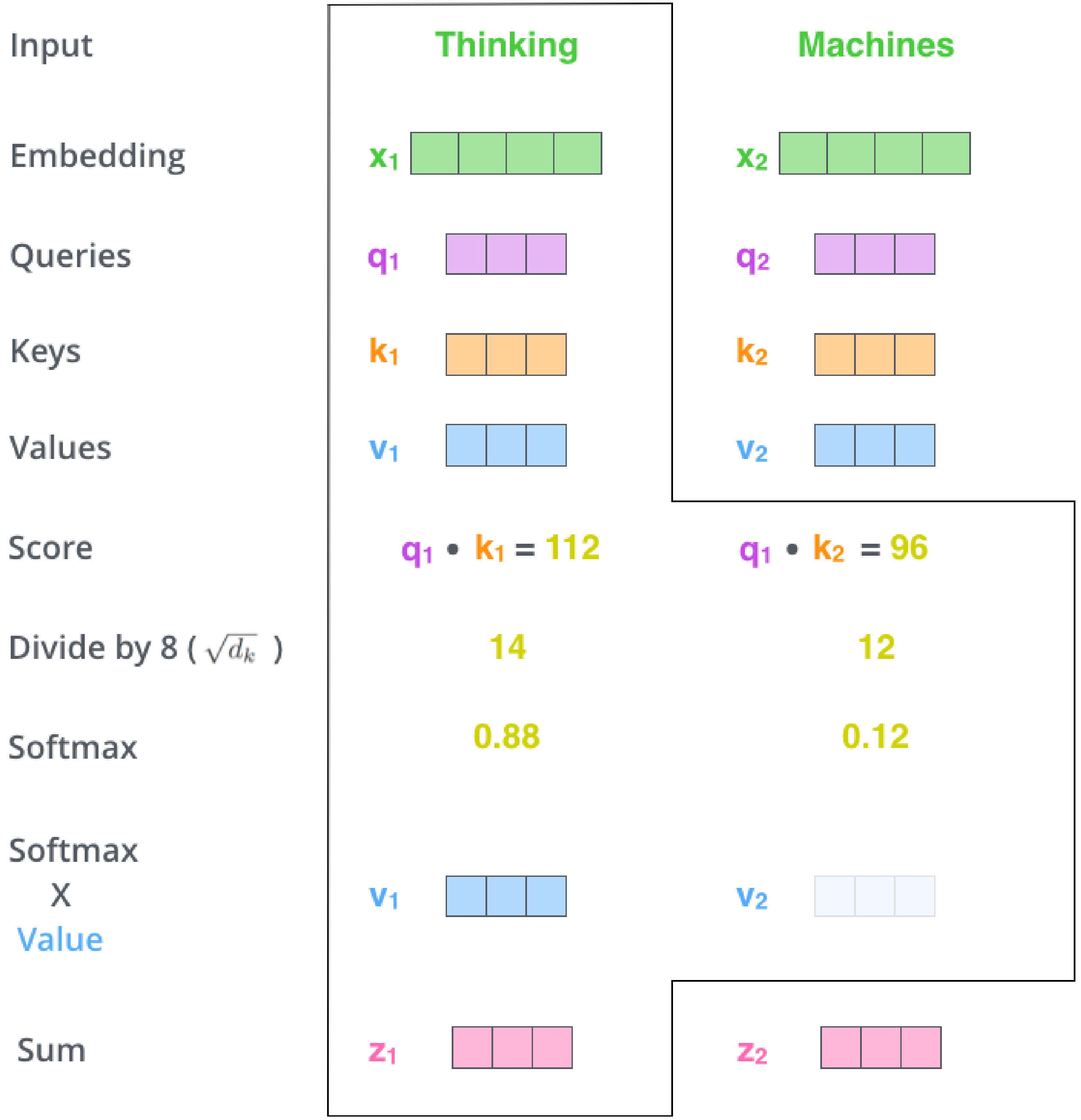


Self attention

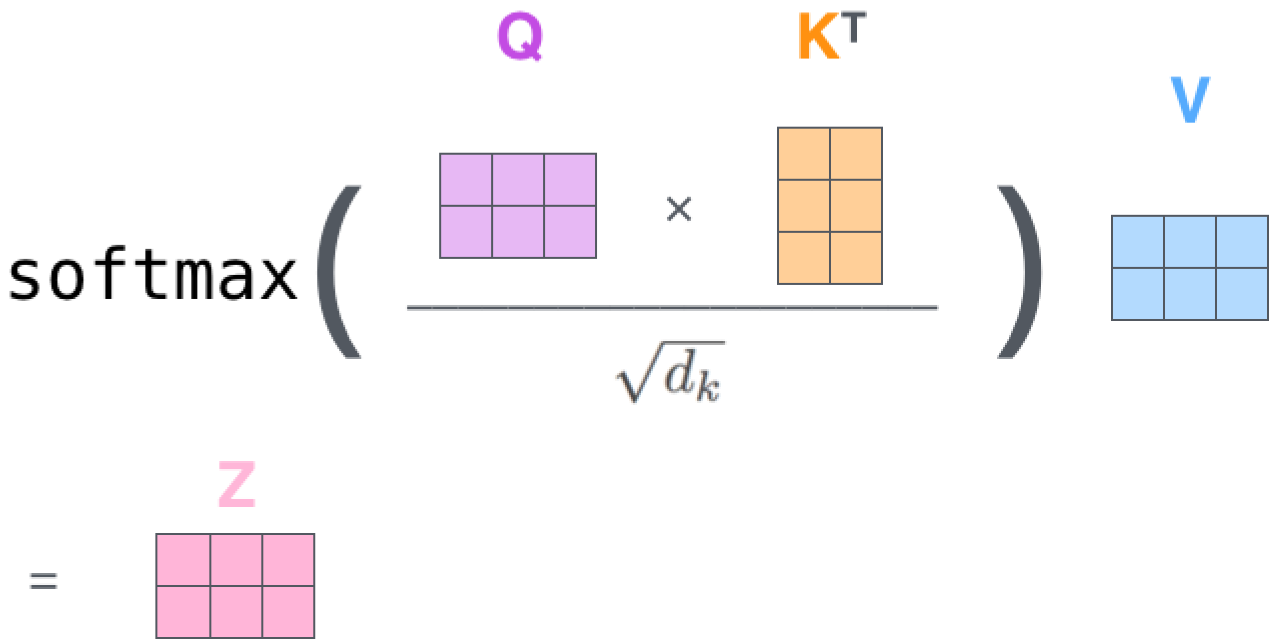
Multiply each value vector by the softmax score.

Sum up the weighted value vectors.

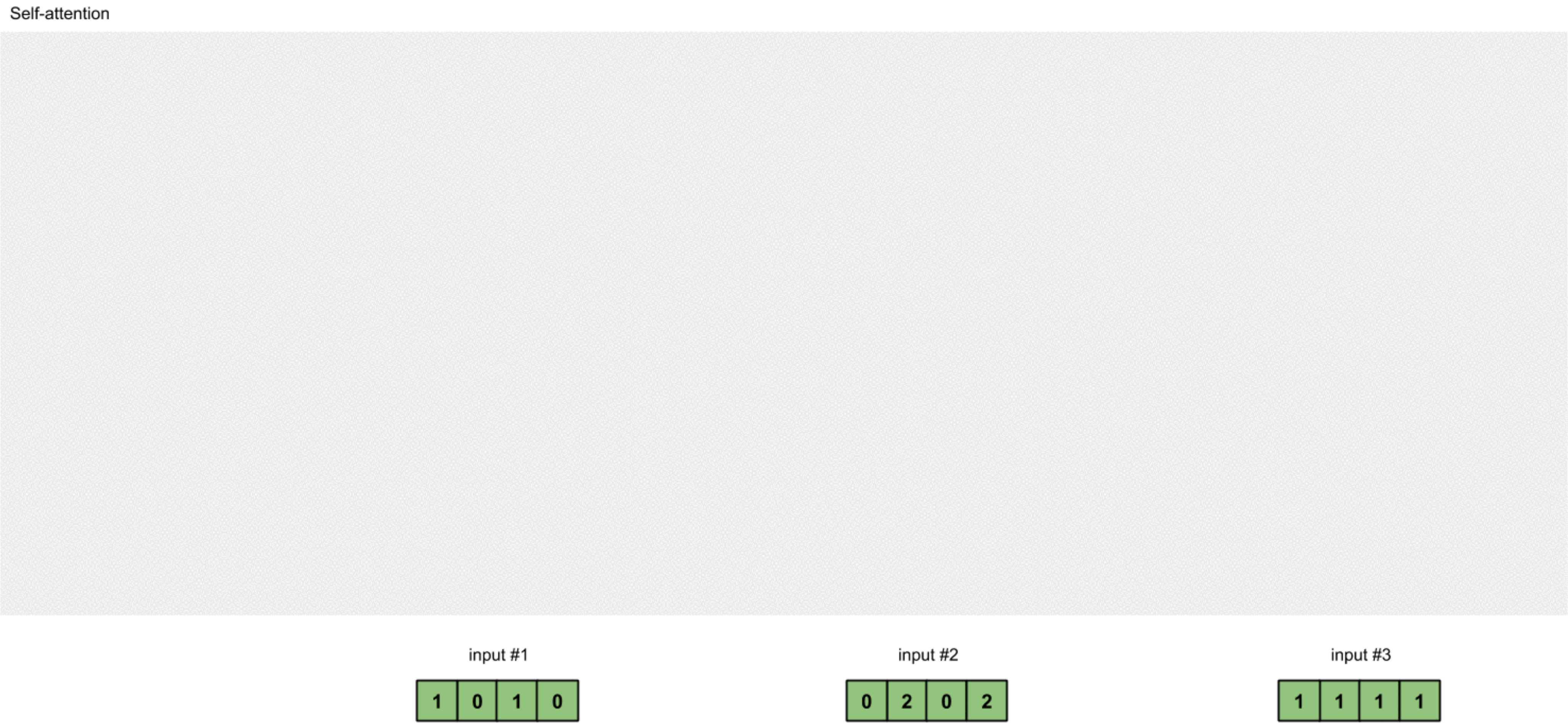
This gives the output for *thinking*

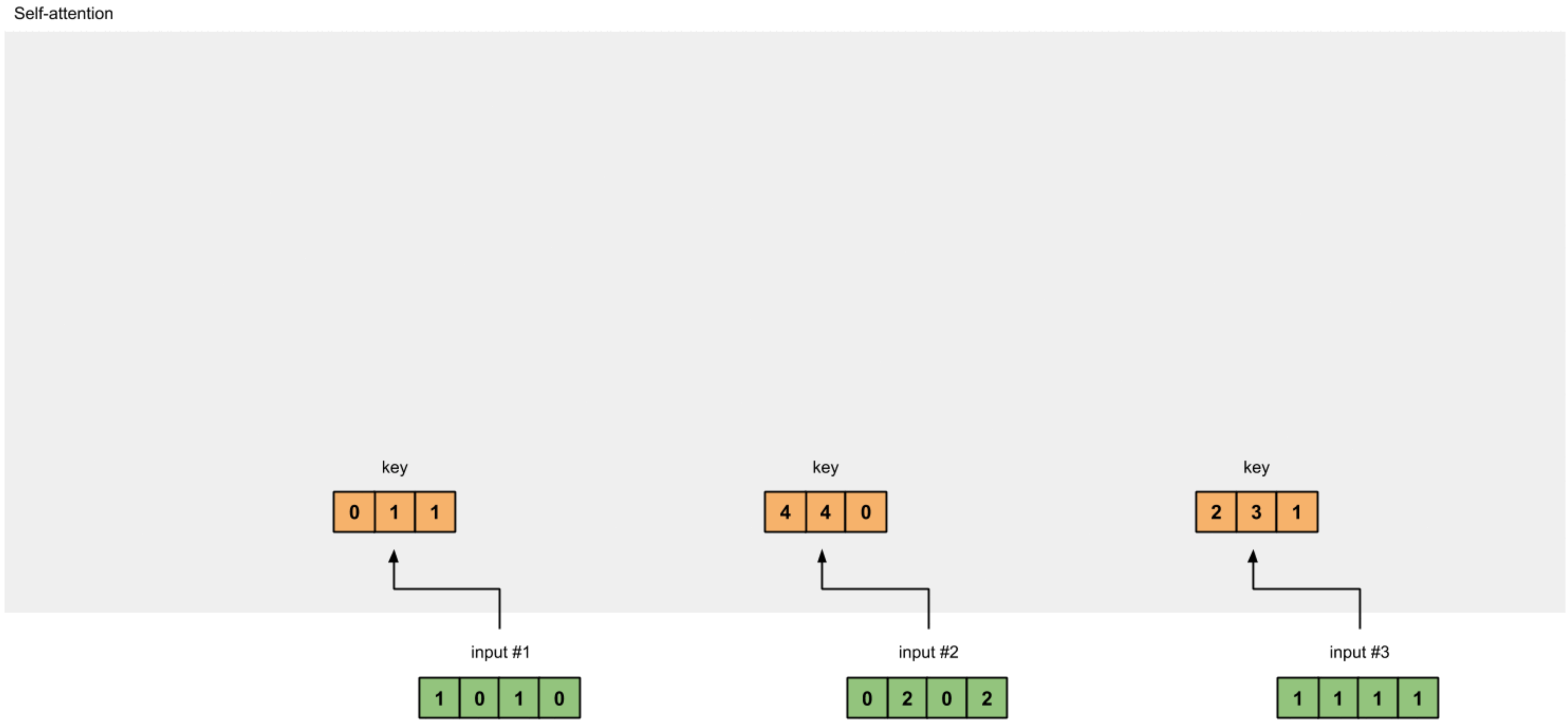


Self attention (matrix calculation)

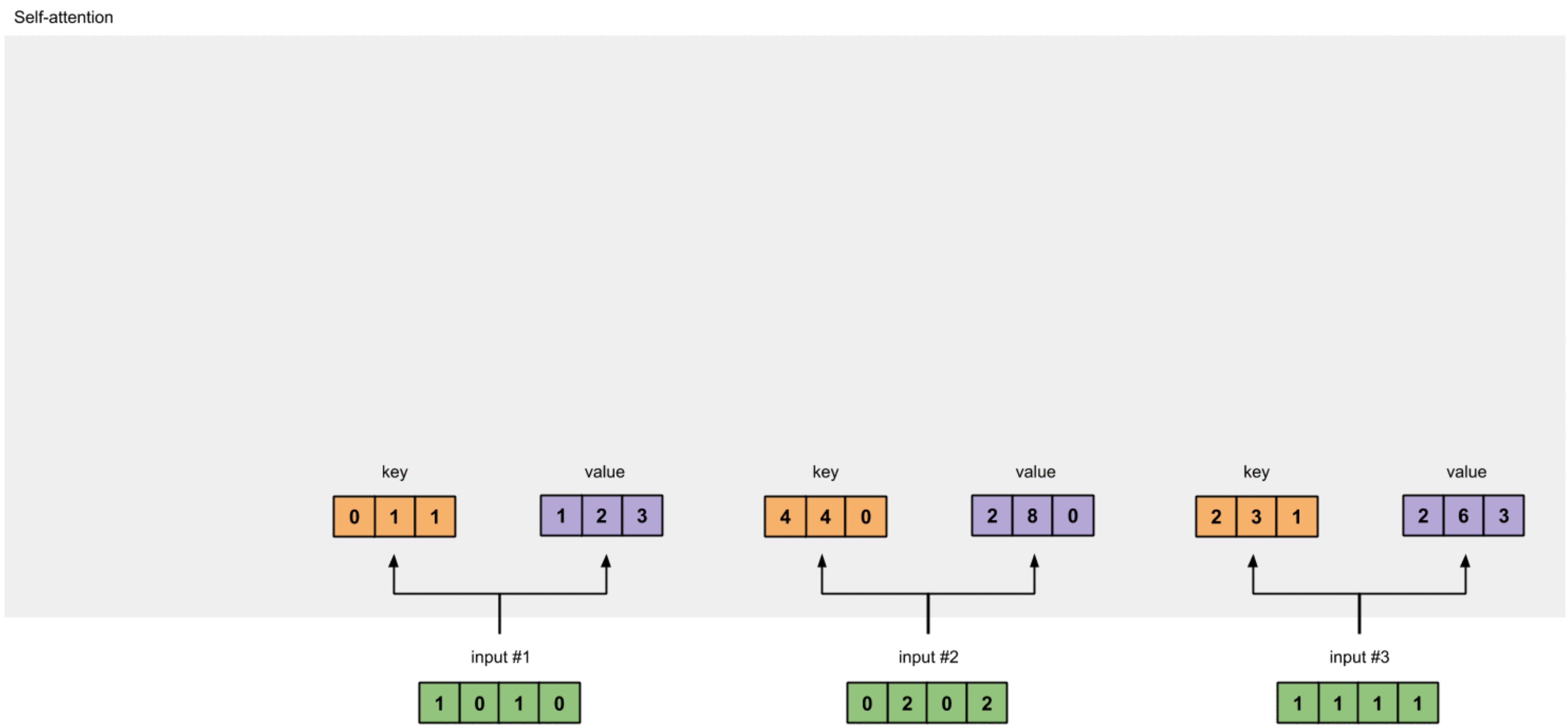


Self attention – from https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

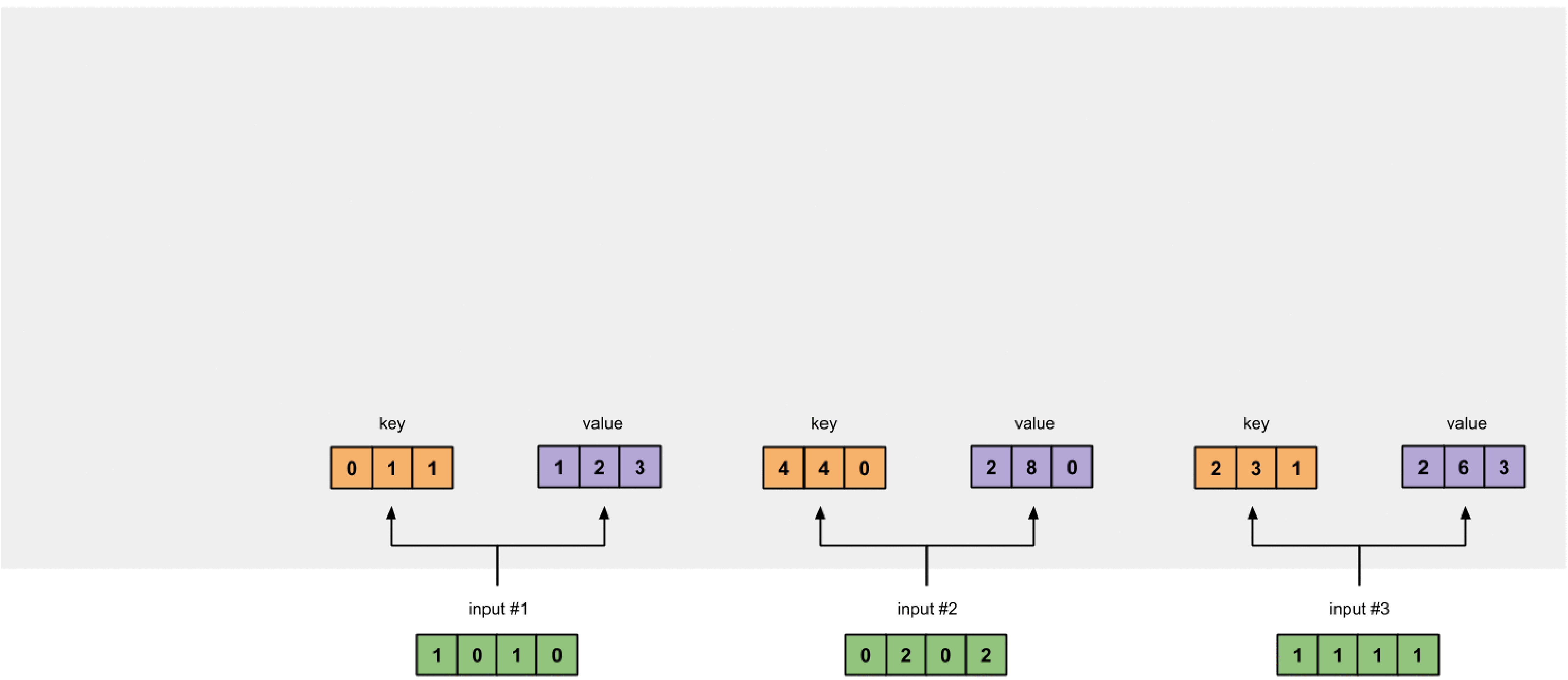




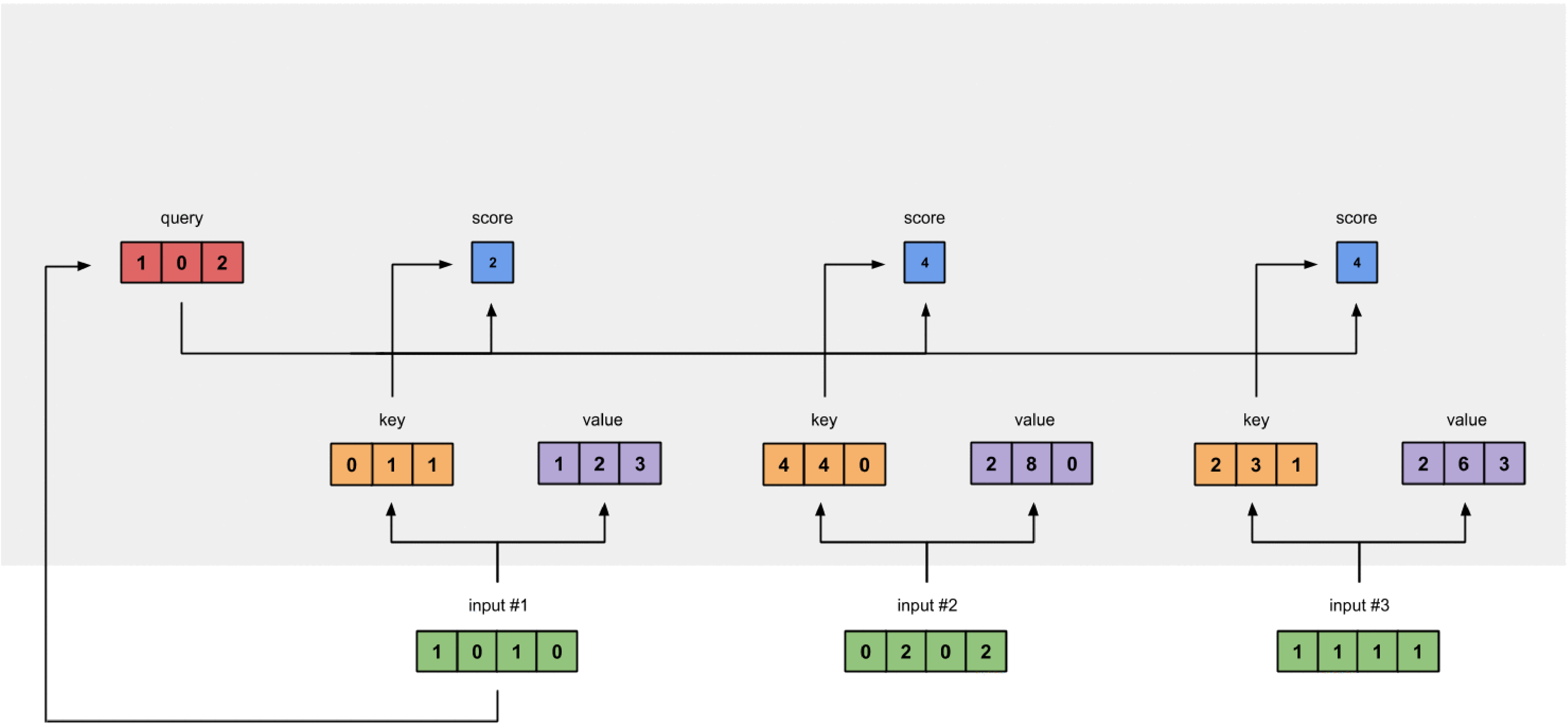
Self-attention
key
0 1 1
key
4 4 0
key
2 3 1
input #1
1 0 1 0
input #2
0 2 0 2
input #3
1 1 1 1


Self-attention
key
0 1 1
value
1 2 3
key
4 4 0
value
2 8 0
key
2 3 1
value
2 6 3
input #1
1 0 1 0
input #2
0 2 0 2
input #3
1 1 1 1

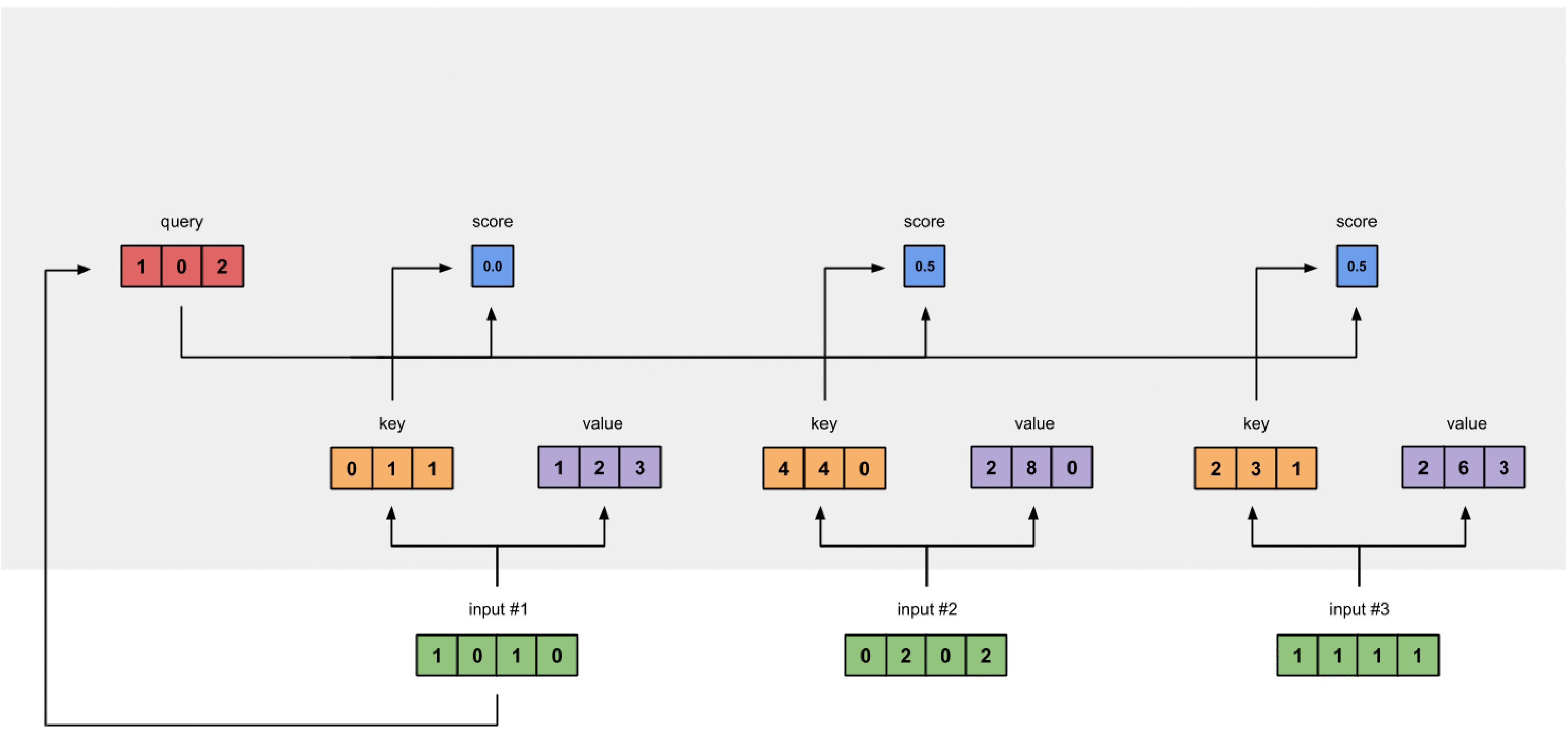
Self-attention



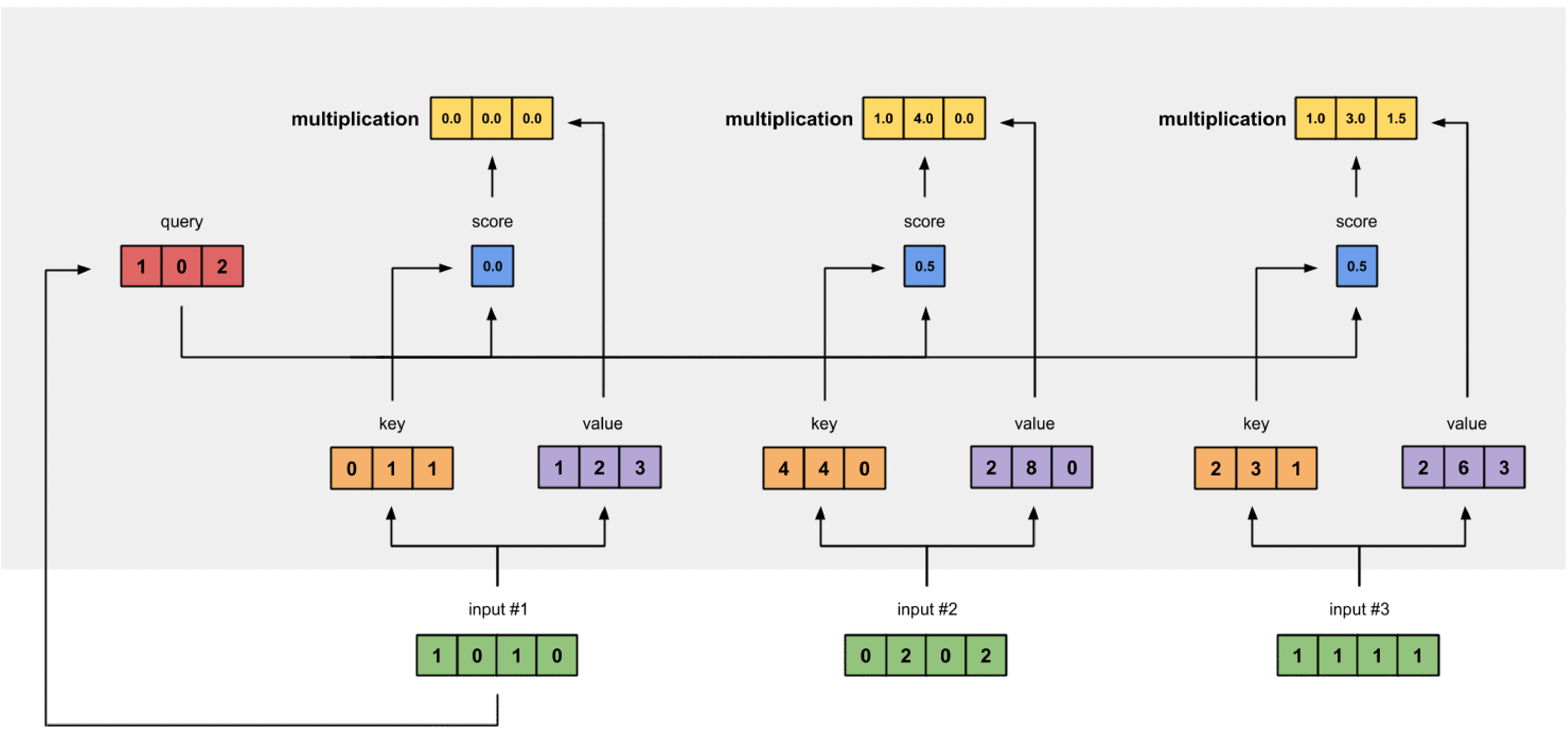
Self-attention

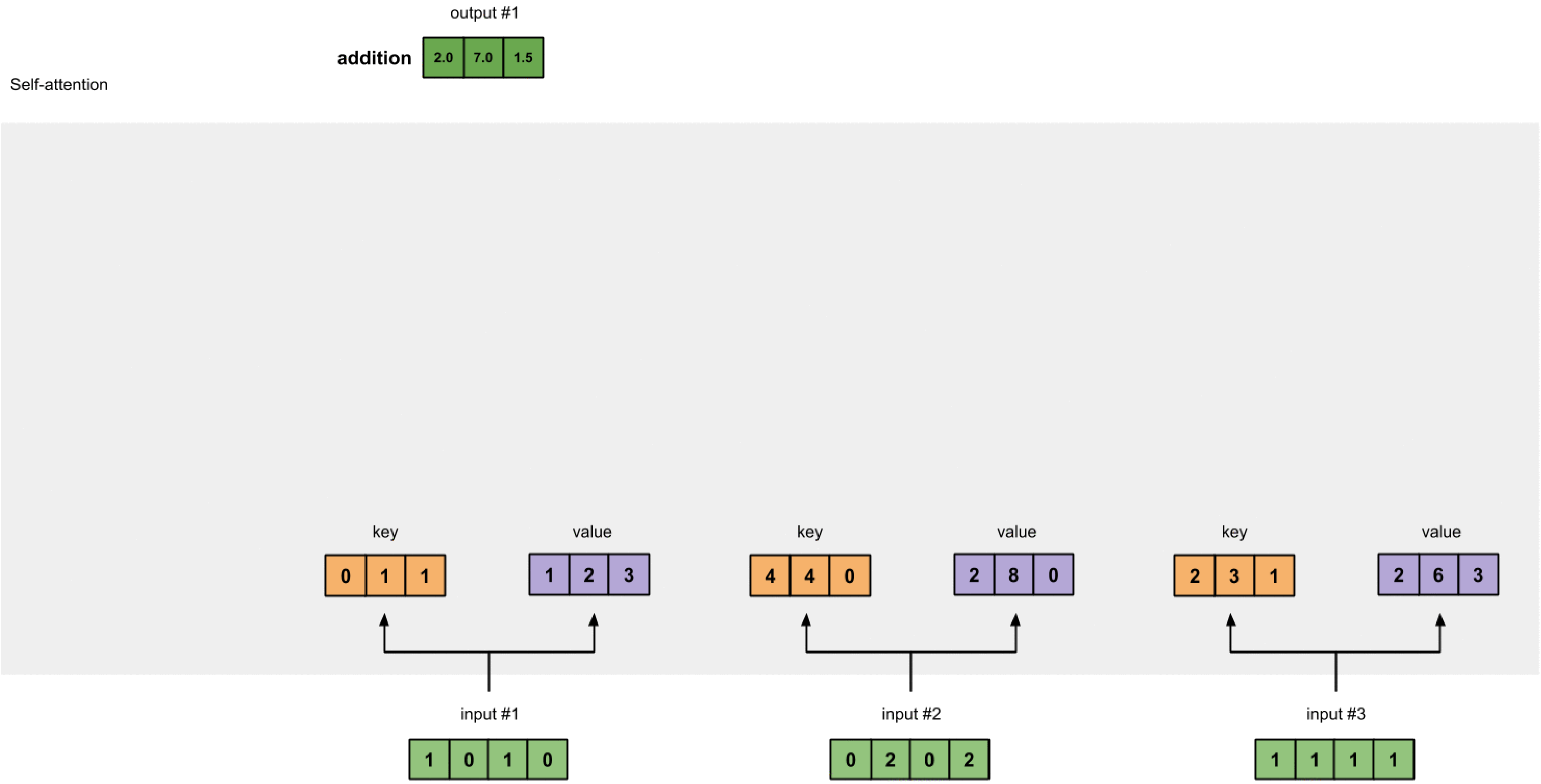


Self-attention



Self-attention

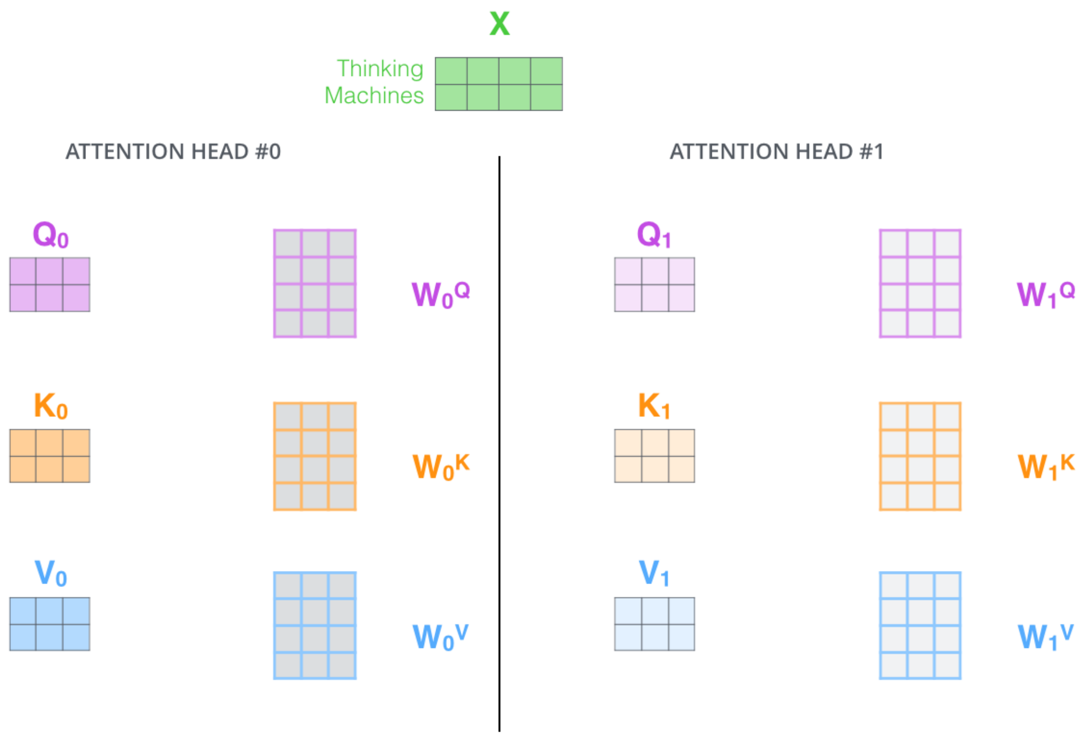




output #1
addition
2.0
7.0
1.5
Self-attention
key
0 1 1
value
1 2 3
key
4 4 0
value
2 8 0
key
2 3 1
value
2 6 3
input #1
1 0 1 0
input #2
0 2 0 2
input #3
1 1 1 1

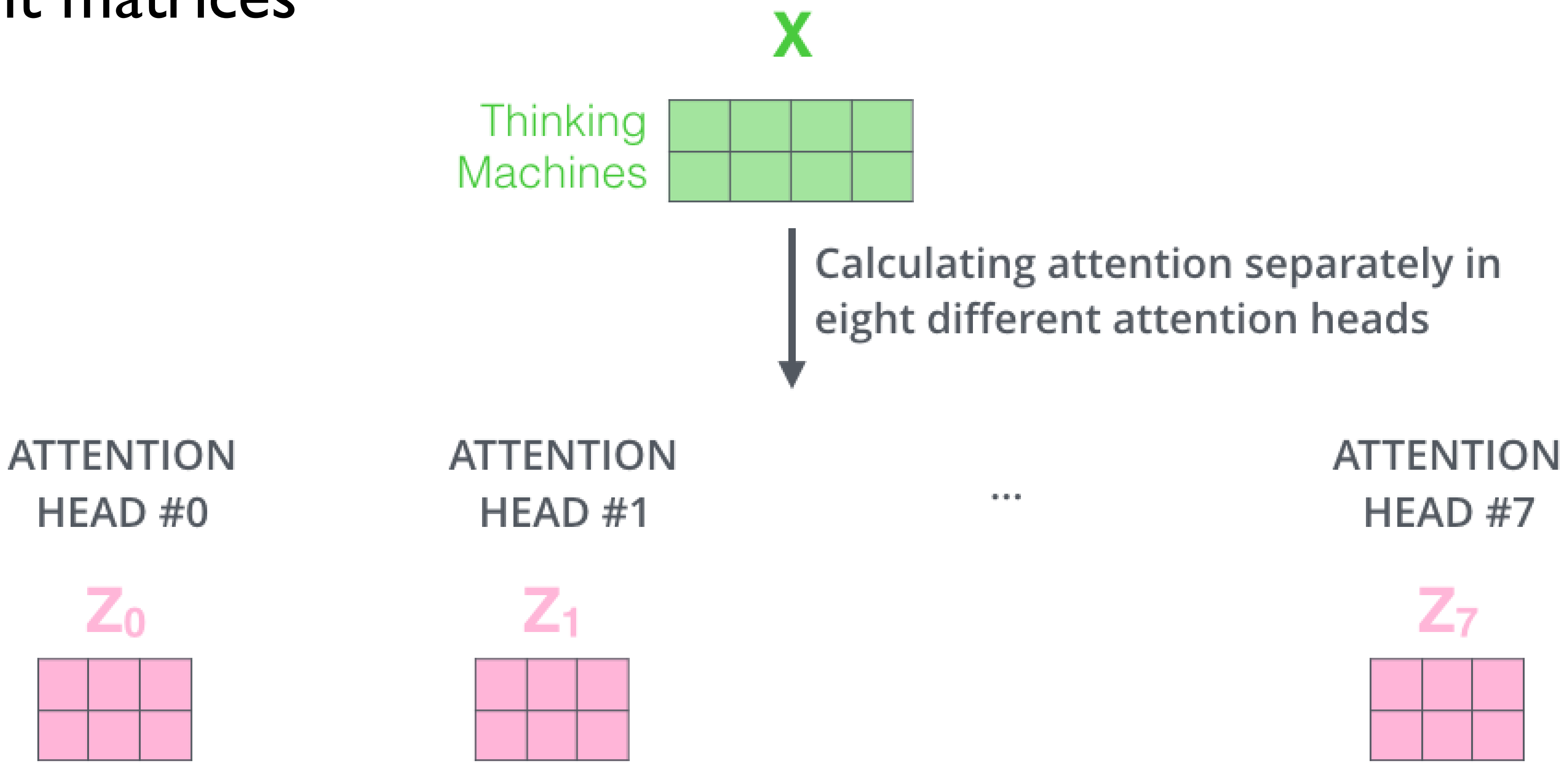
Multi-head attention

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



Multi-head attention

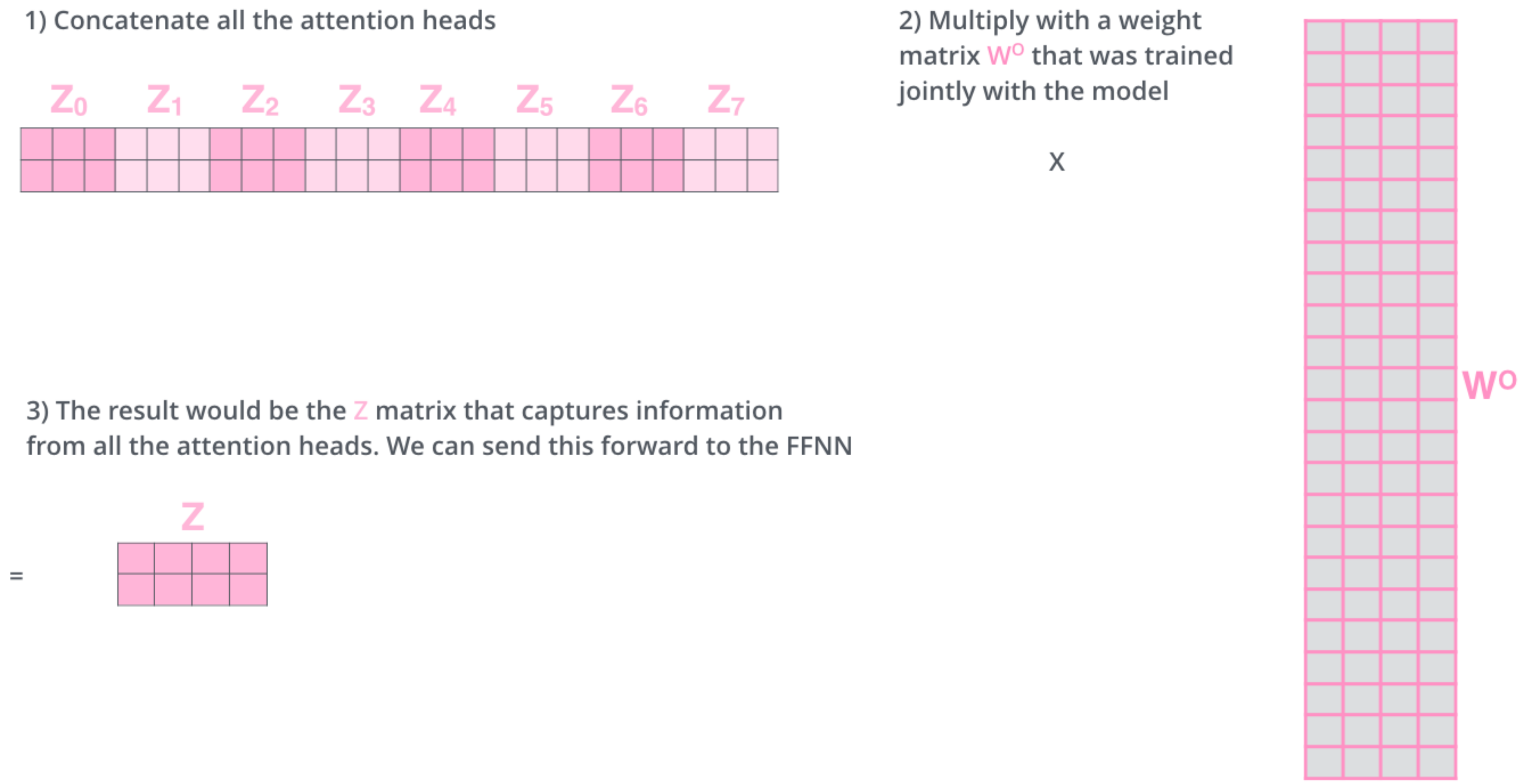
- Just do the same self-attention calculation (eight times) with different weight matrices



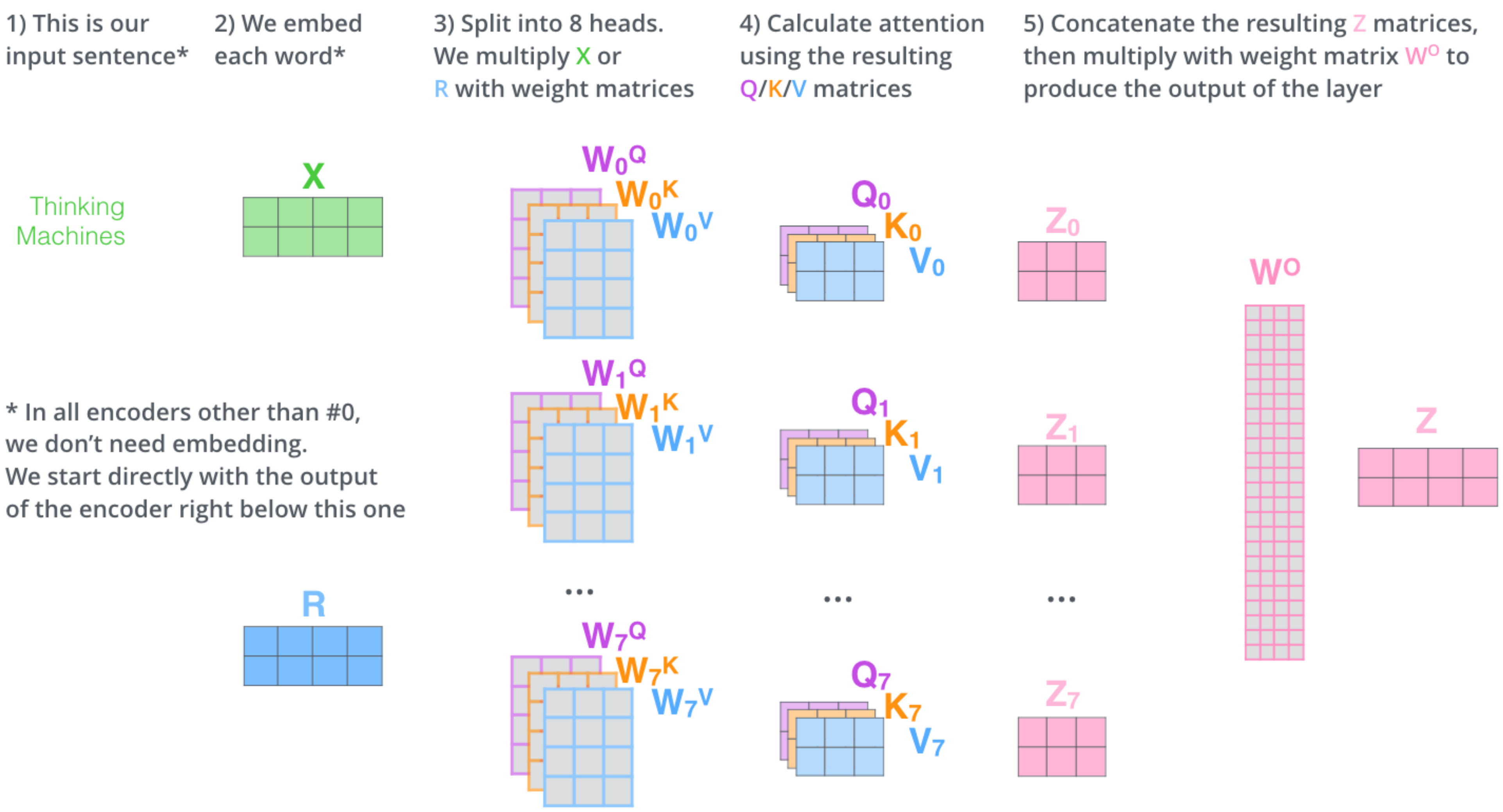
Multi-head attention

The feed-forward layer is not expecting eight matrices.

What to do?

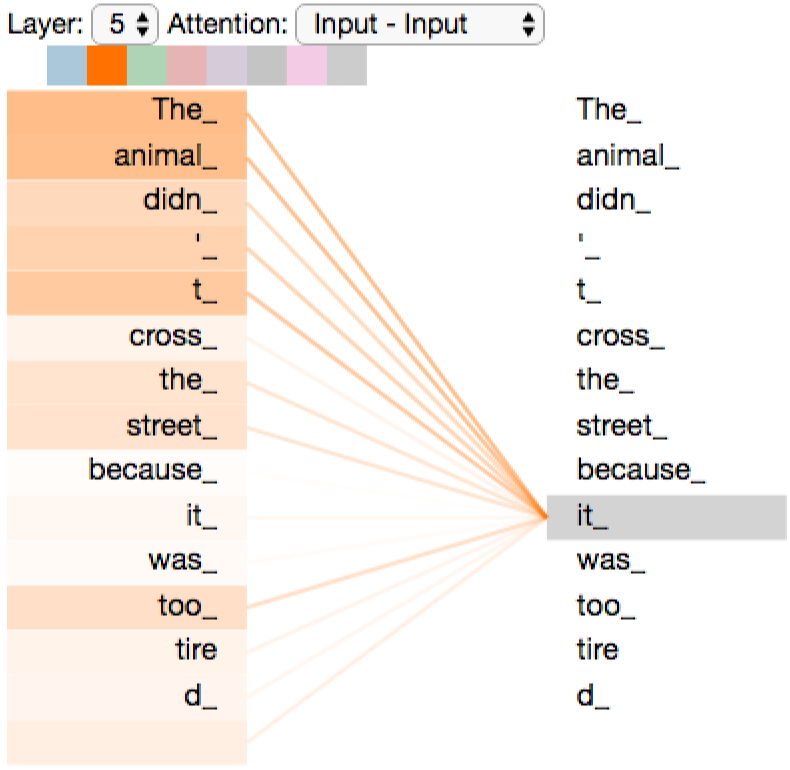


Multi-head attention



Self attention

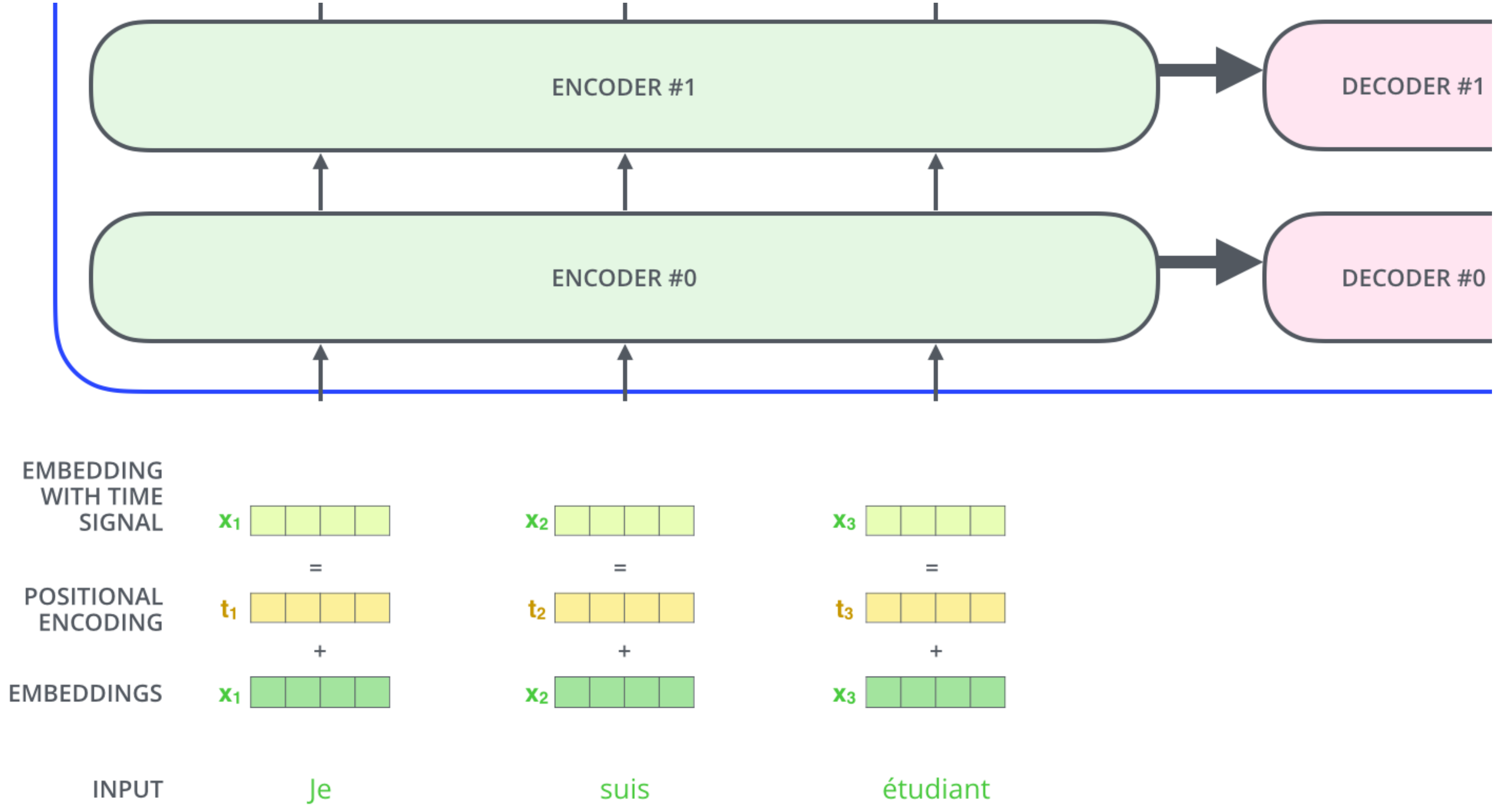
- “The animal didn't cross the street because it was too tired”



What about word order?

- Positional encoding

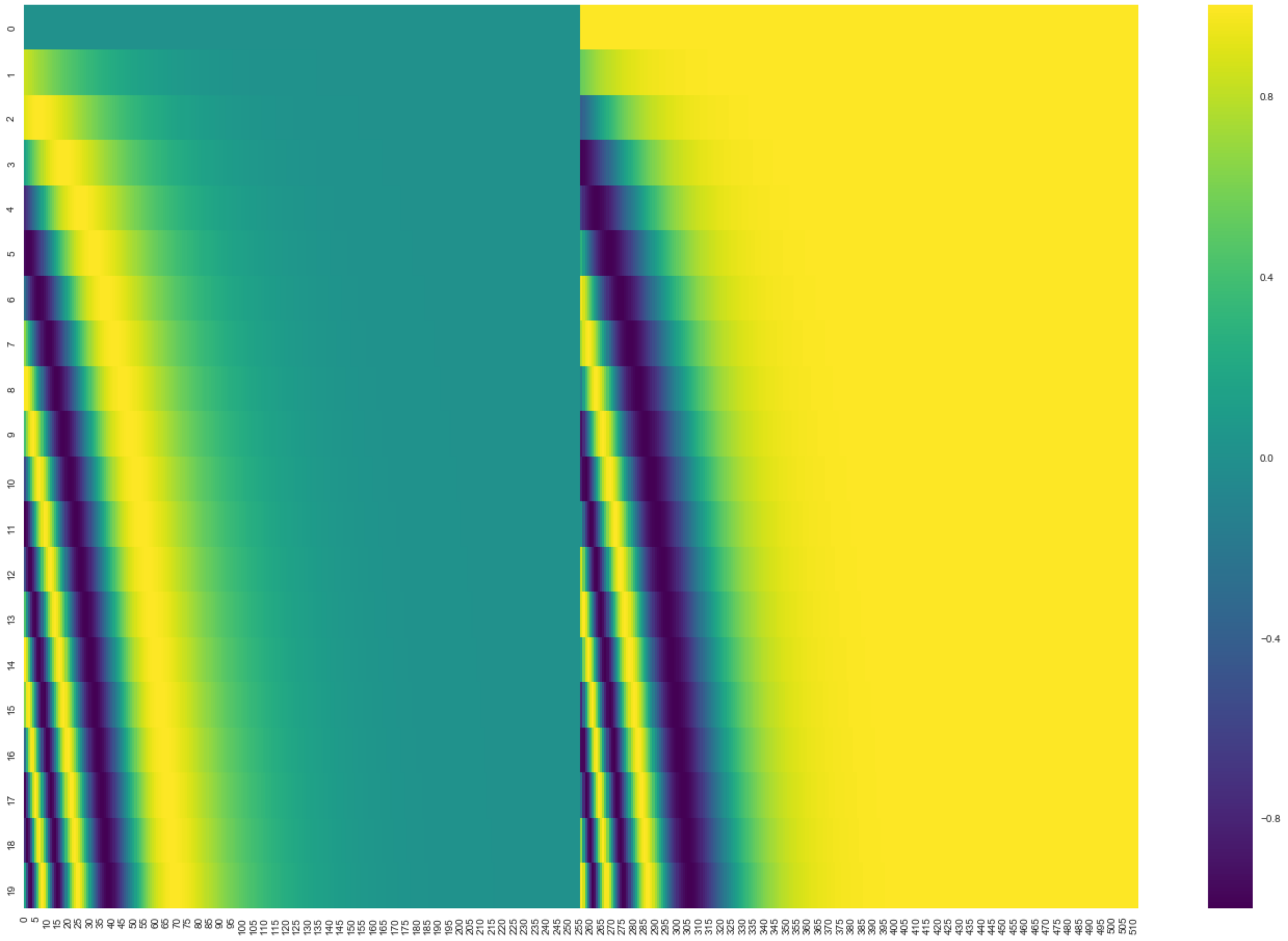
- To address this, the transformer adds a vector to each input embedding



Positional encoding

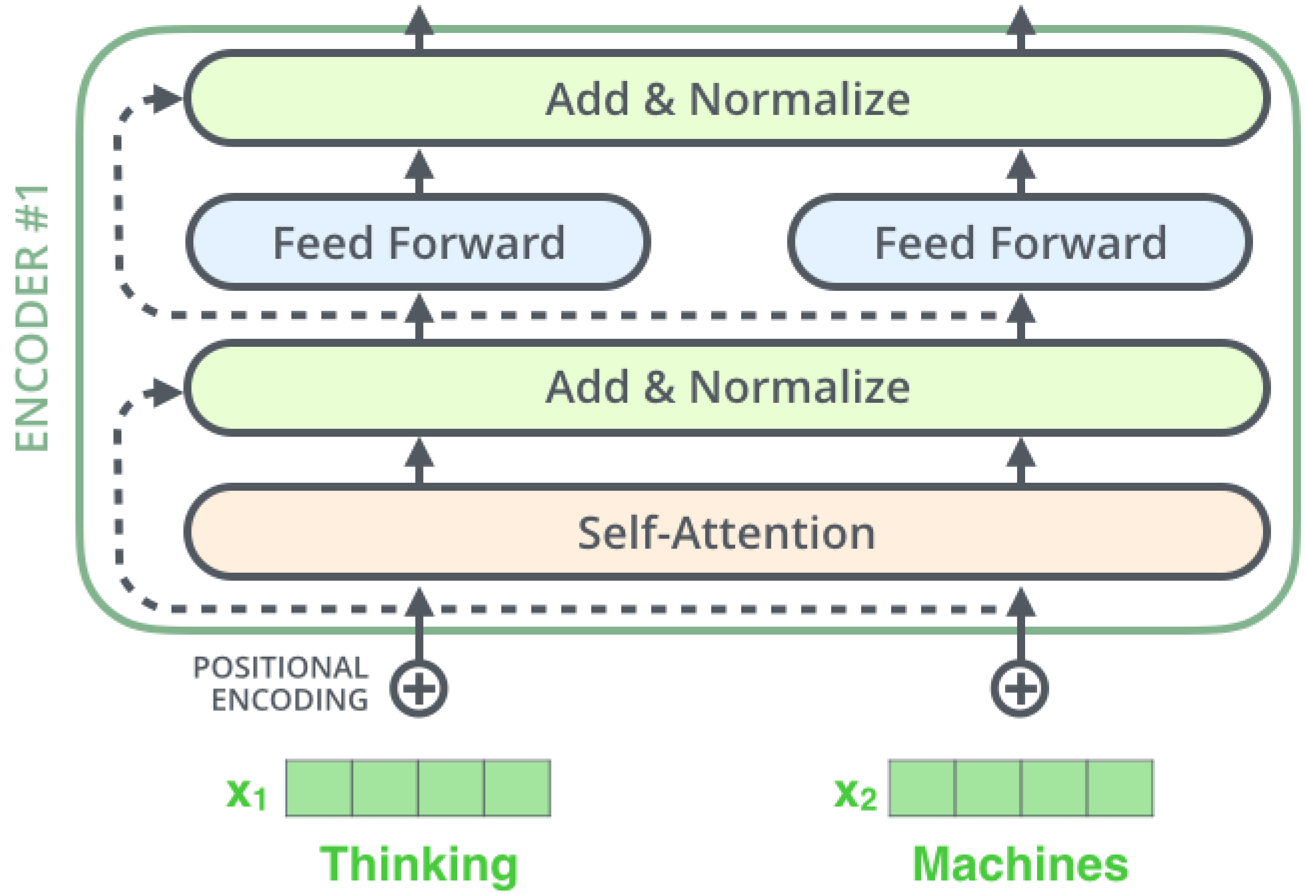
Each row corresponds the a positional encoding of a vector.

Each row contains 512 values – each with a value between 1 and -1.

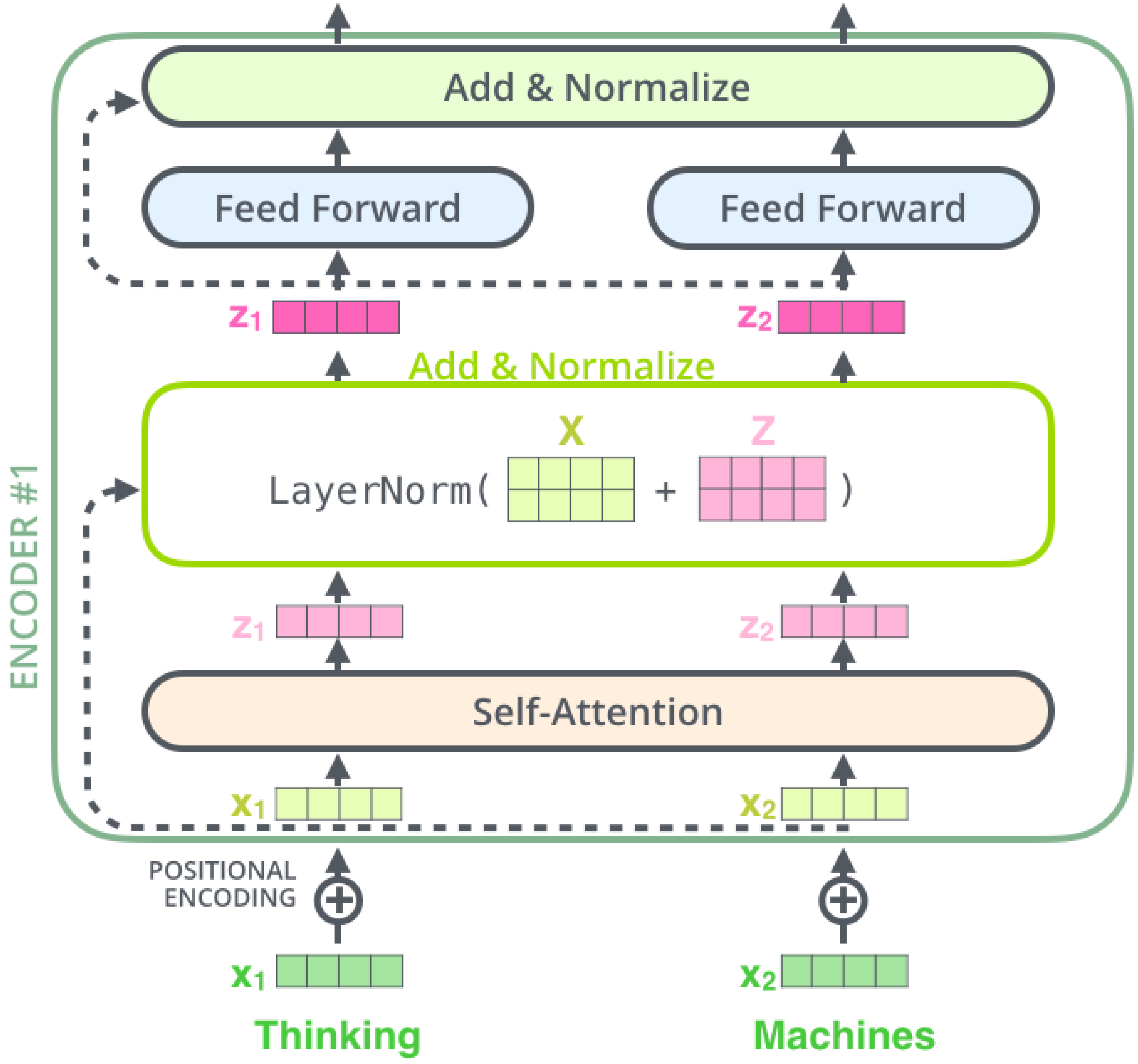


positional encoding for 20 words (rows) with an embedding size of 512 (columns)

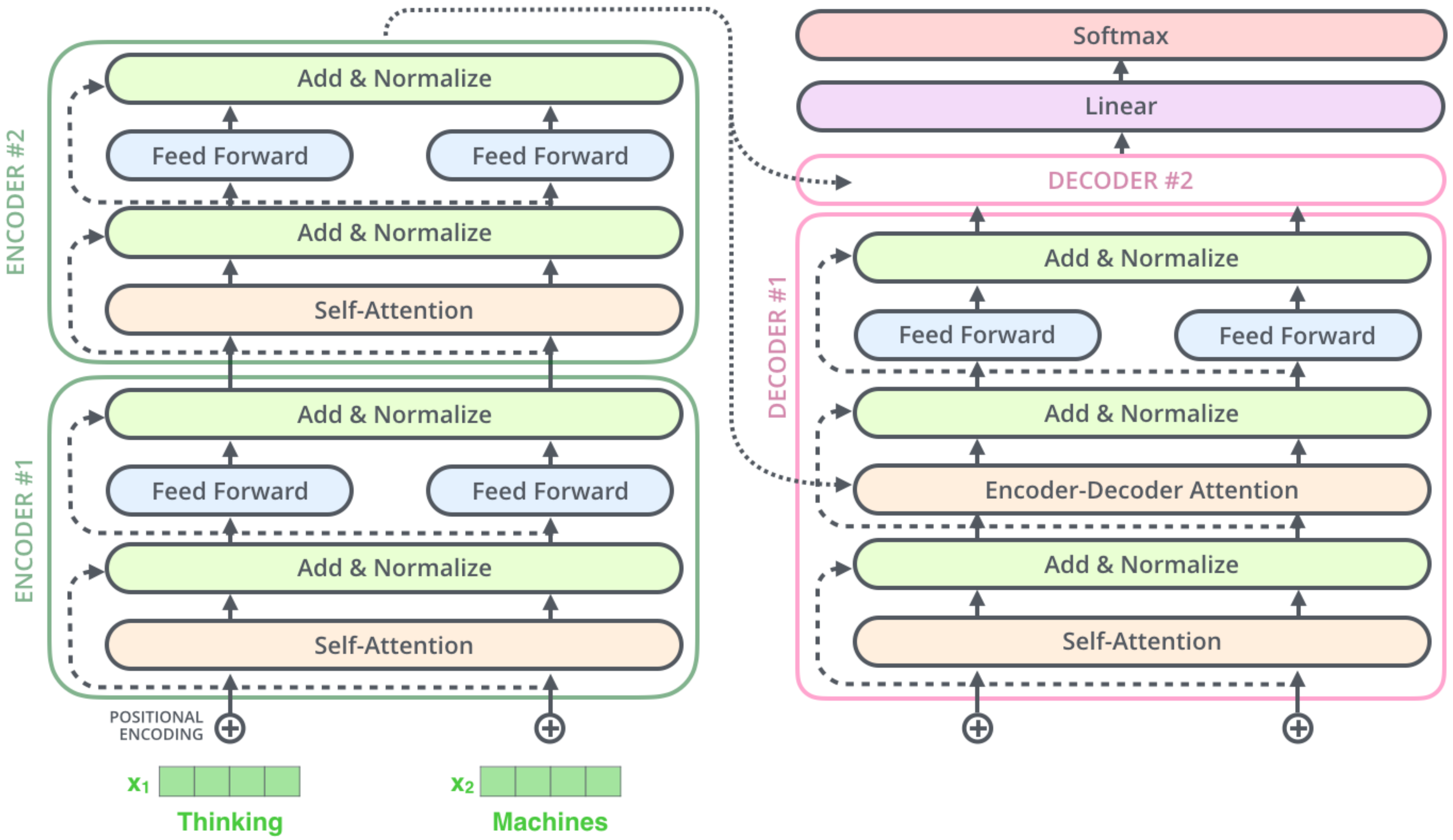
The residuals



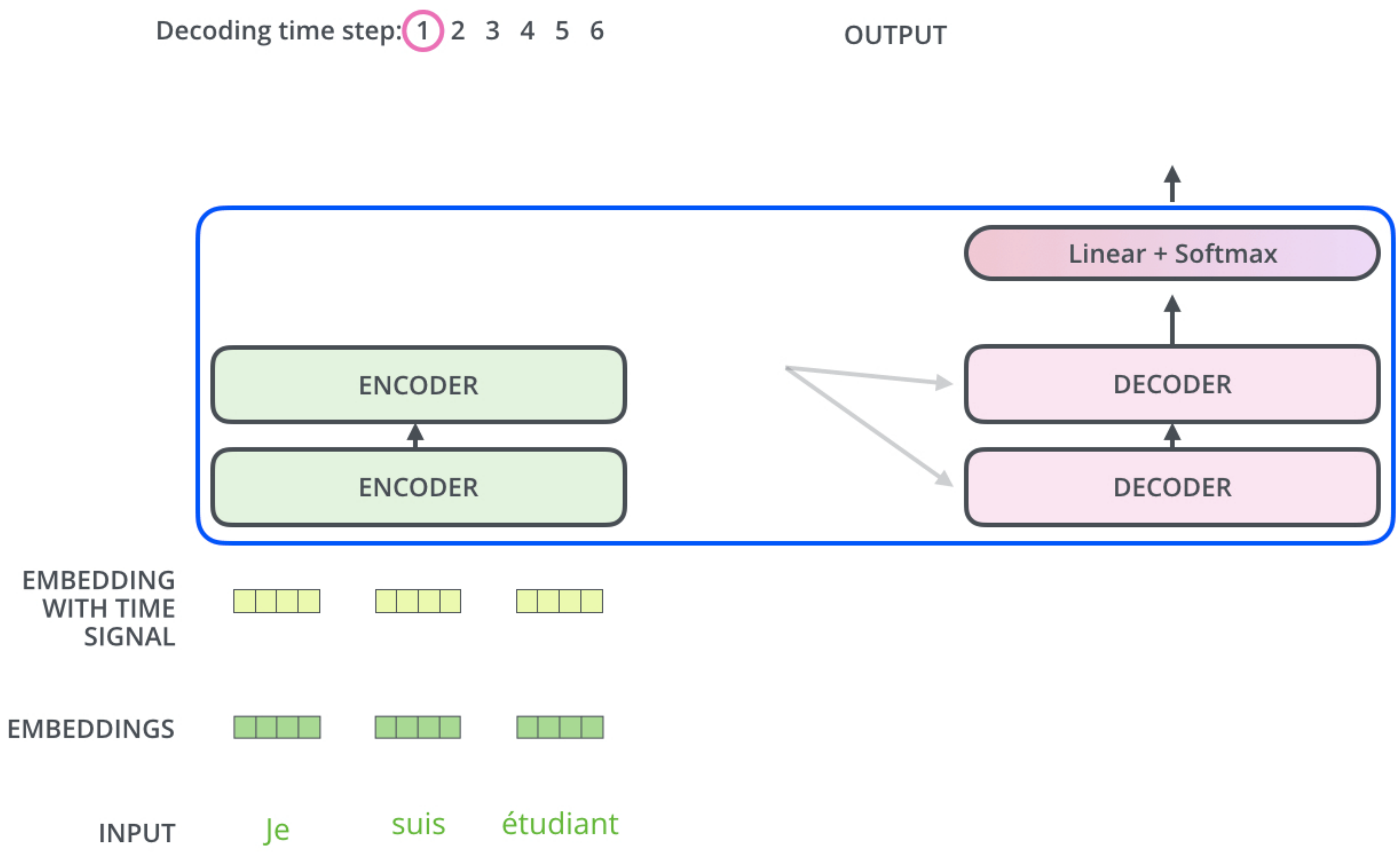
The encoder


Add & Normalize
Feed Forward
Feed Forward
z_1
z_2
Add & Normalize
X
Z
LayerNorm(+)
z_1
z_2
Self-Attention
x_1
x_2
ENCODER #1
POSITIONAL
ENCODING
x_1
x_2
Thinking
Machines

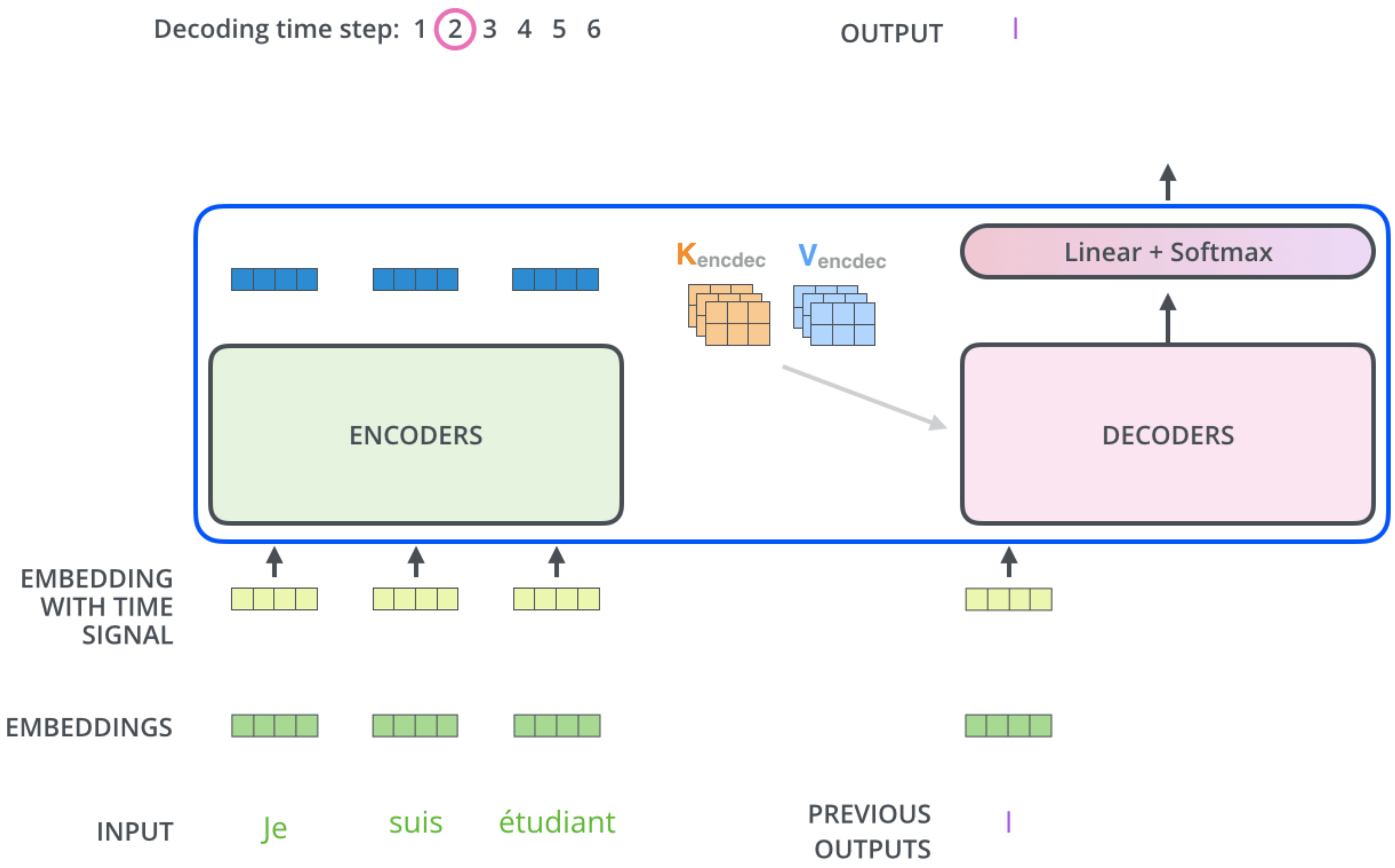
Transformer with 2 stacked encoders and decoders



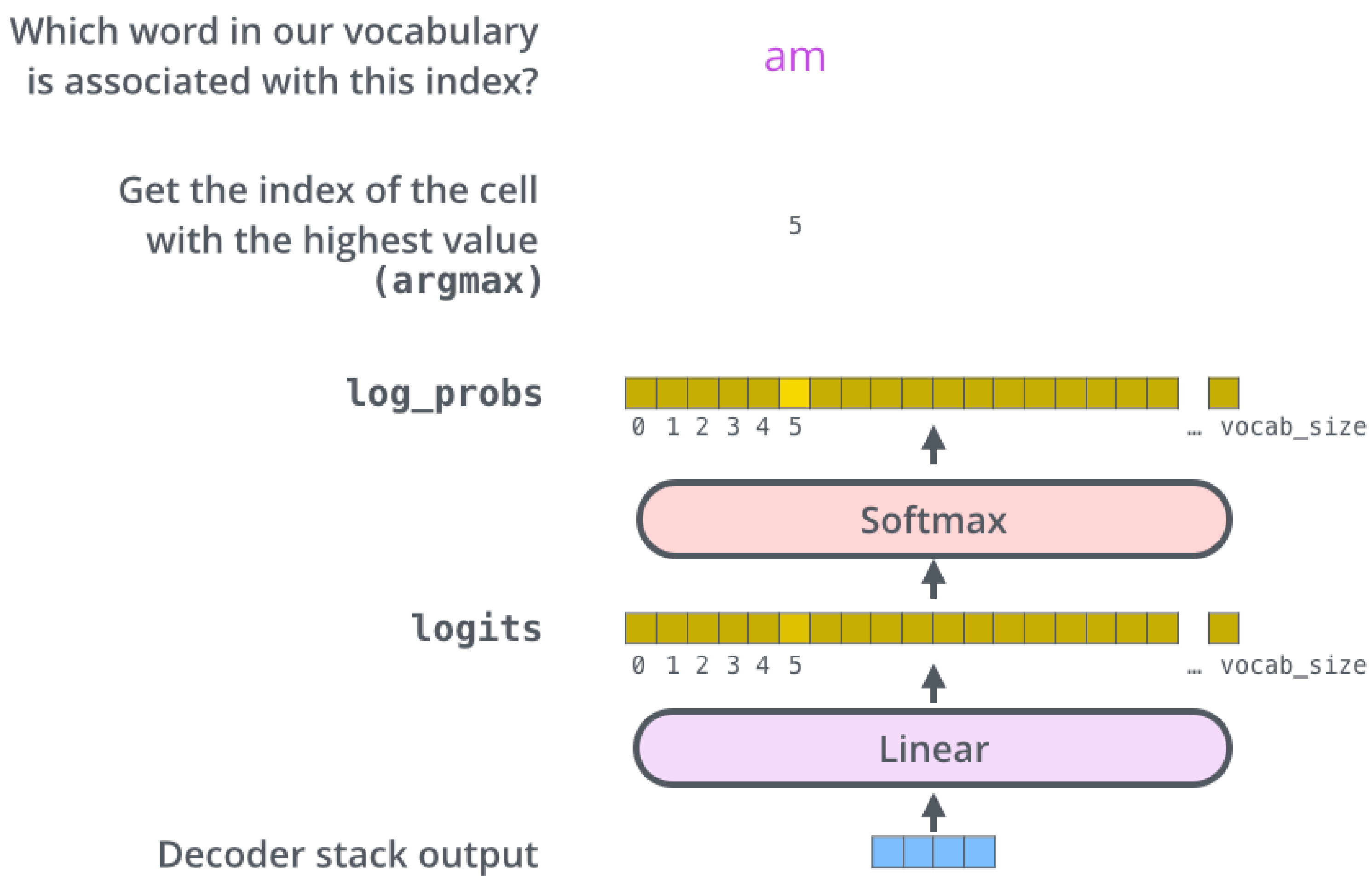
Decoder side



Decoder side



Final softmax layer



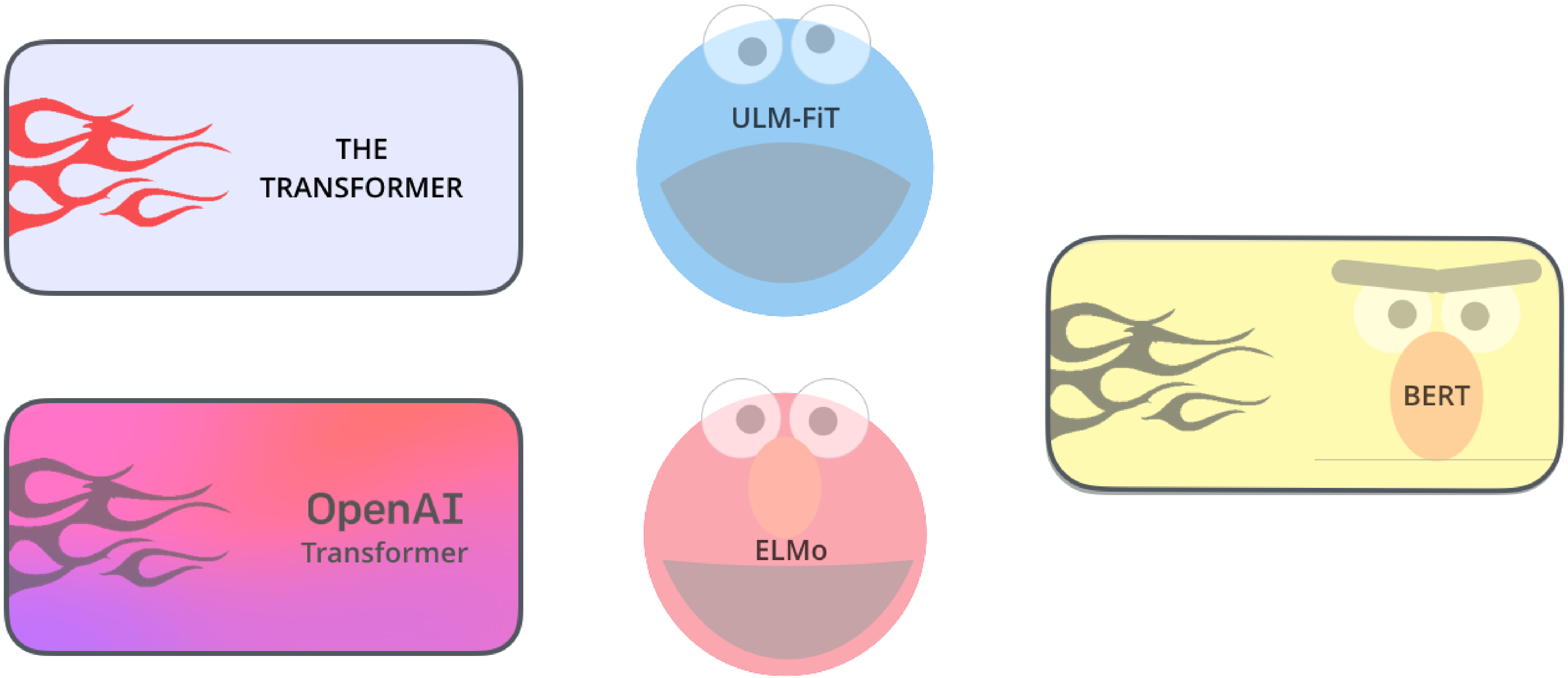
Transformers

- Impressive results on machine translation

- Replacement for LSTMs?
  - Better at capturing long-distance dependencies

- But, how to use encoder-decoder for sentence classification?
  - BERT solves this!

Contextualised word embeddings



Contextualised embeddings

A solution to both meaning conflation and itegration difficulty

Contextualised Embeddings

Word are **dynamic** in nature; they change role depending on the **context** in which they appear

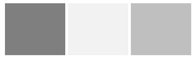
Contextualised Embeddings

Word are **dynamic** in nature; they change role depending on the **context** in which they appear

We need dynamic word embeddings!

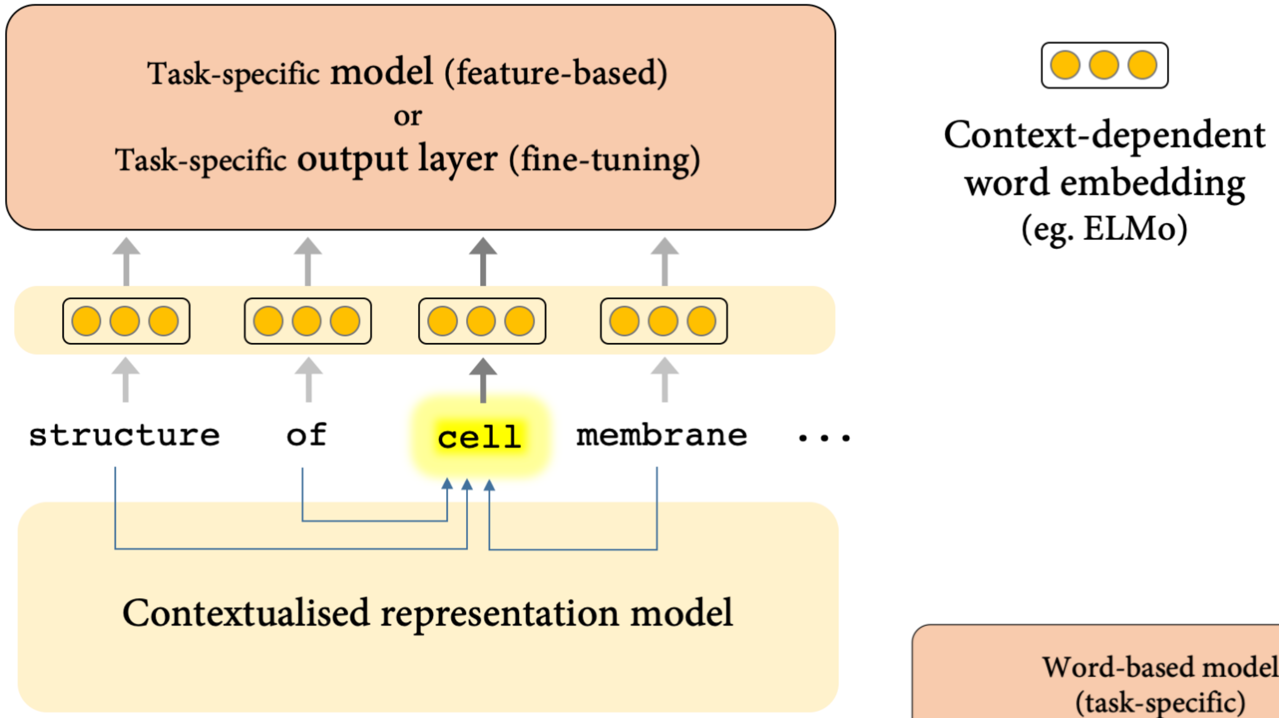
Contextualised Embeddings

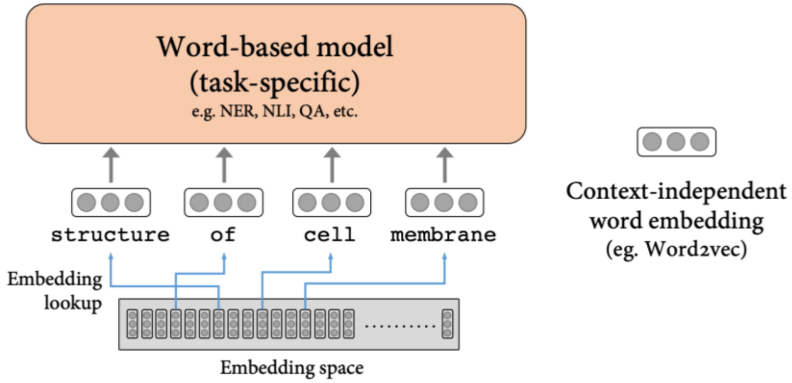
I ordered a wireless *mouse* from my laptop



Mouse has a high breeding rate

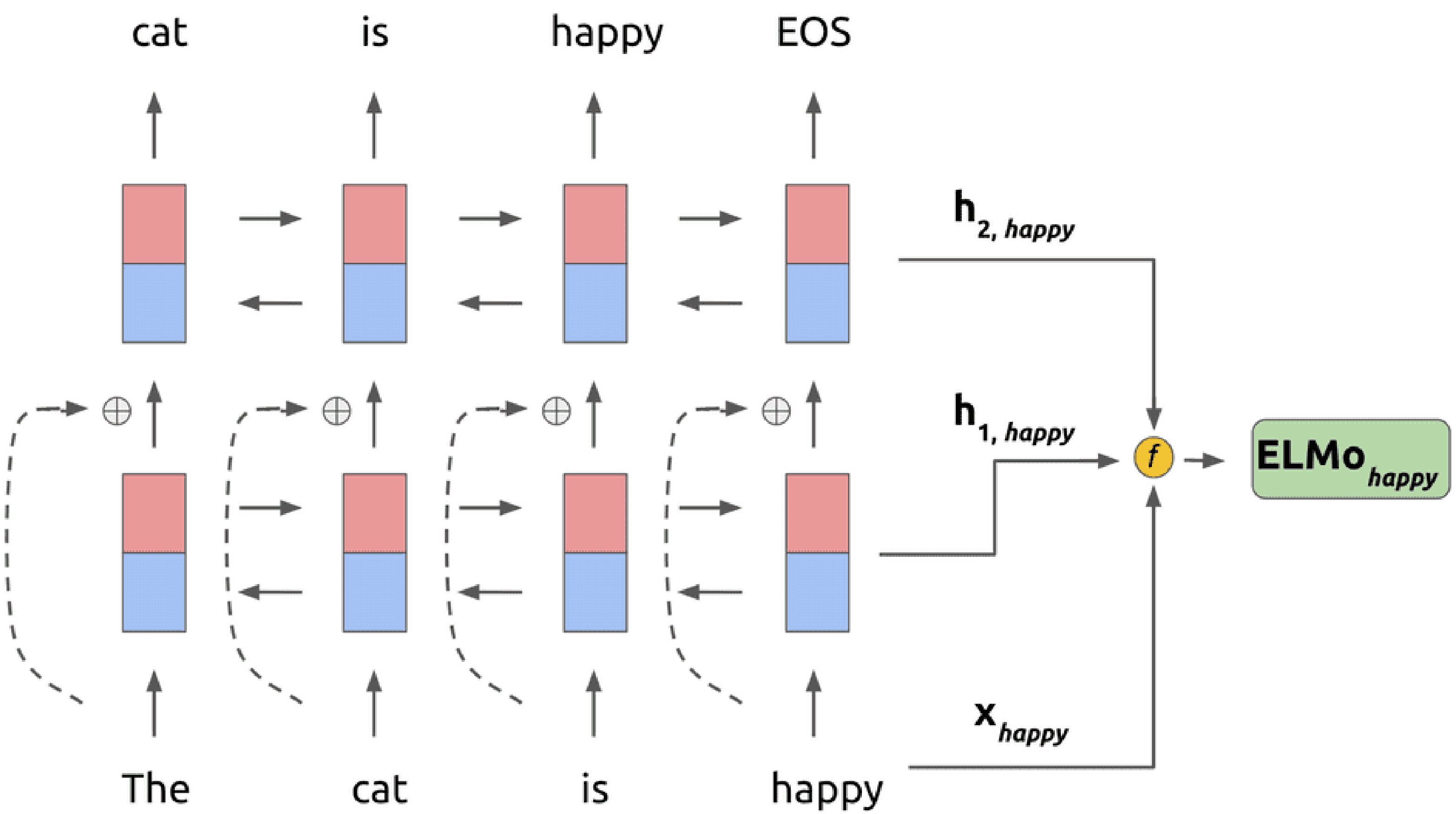
Contextualised Embeddings



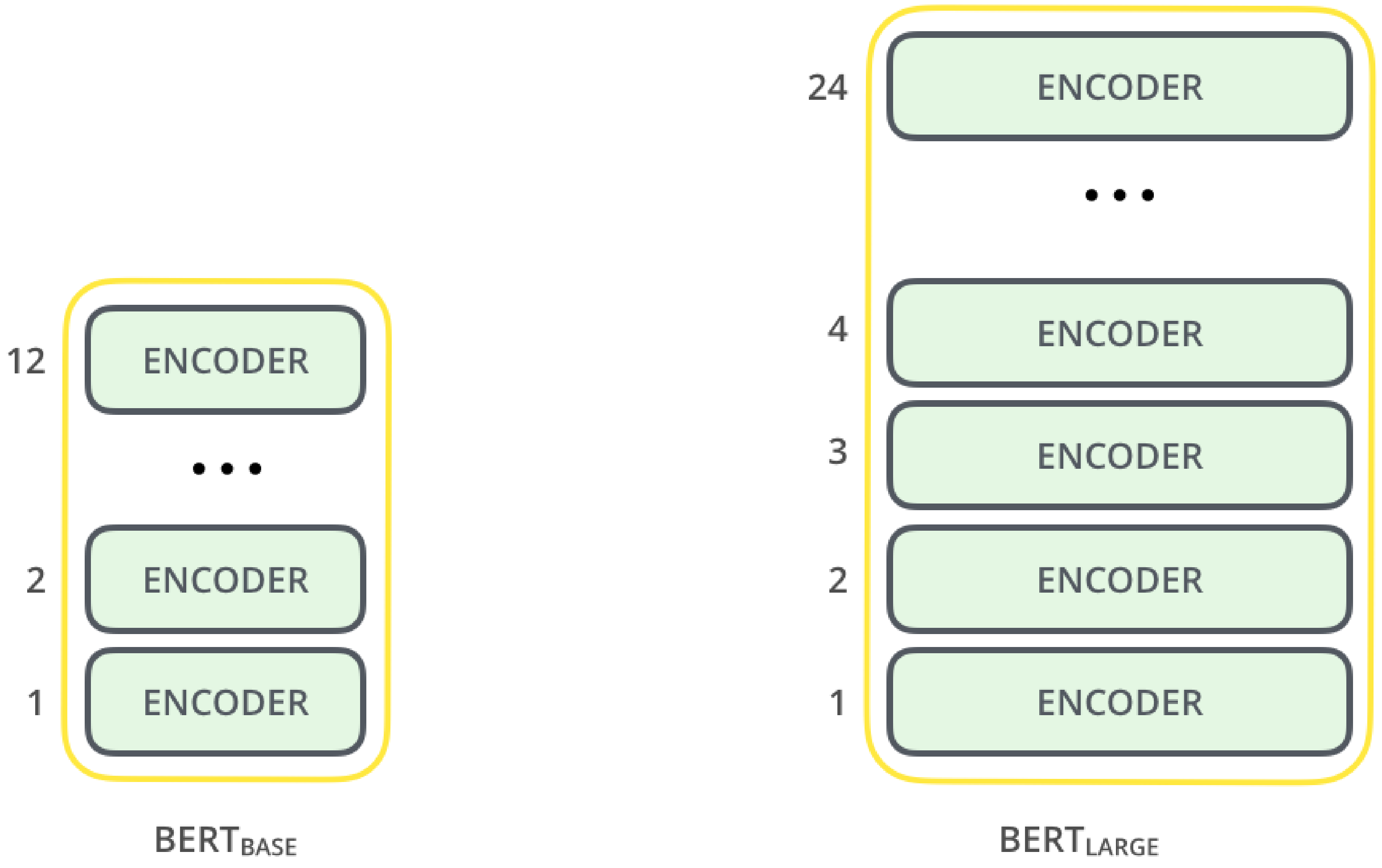


Contextualised Embeddings

ELMo: Embeddings from Language Models

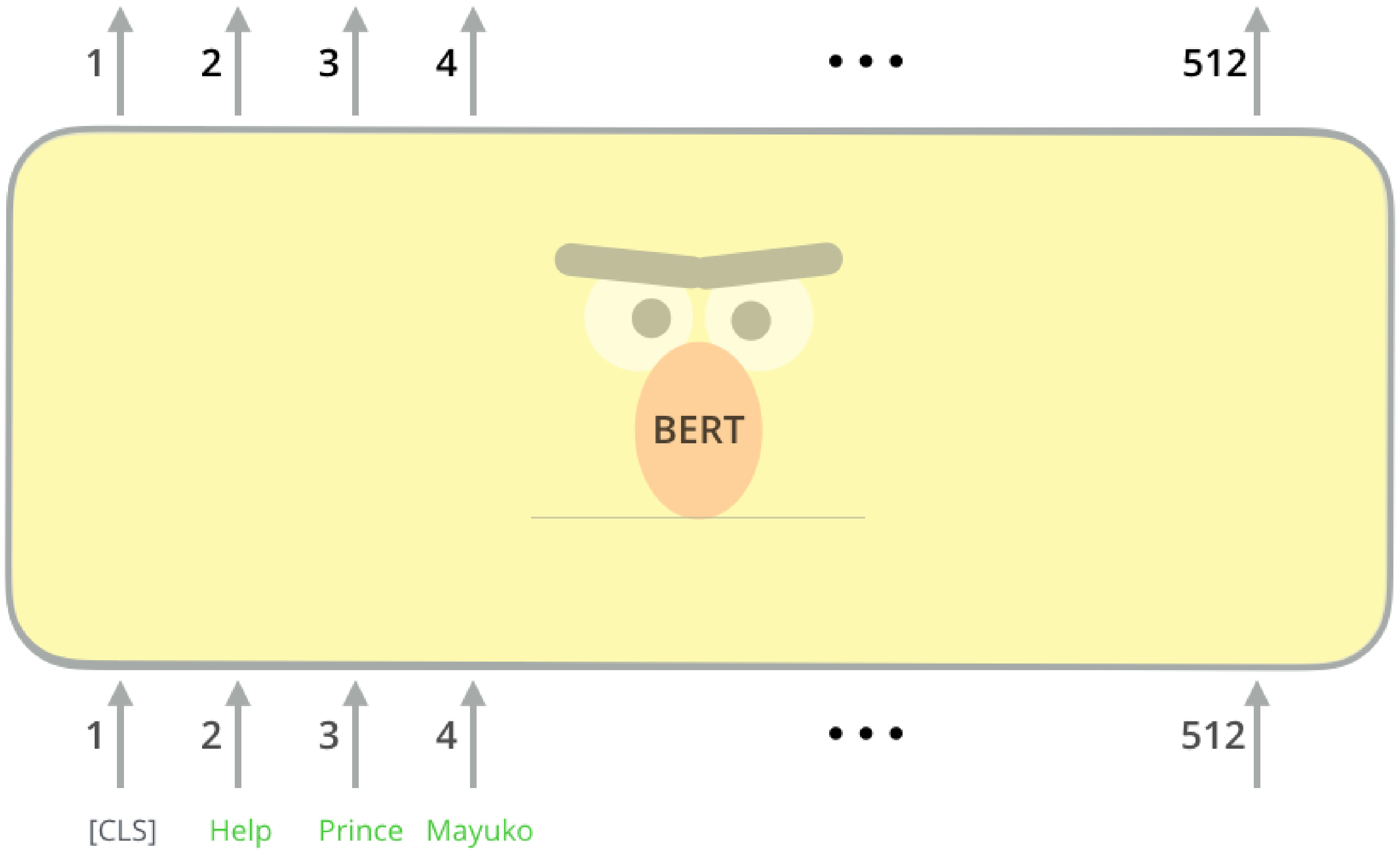


BERT

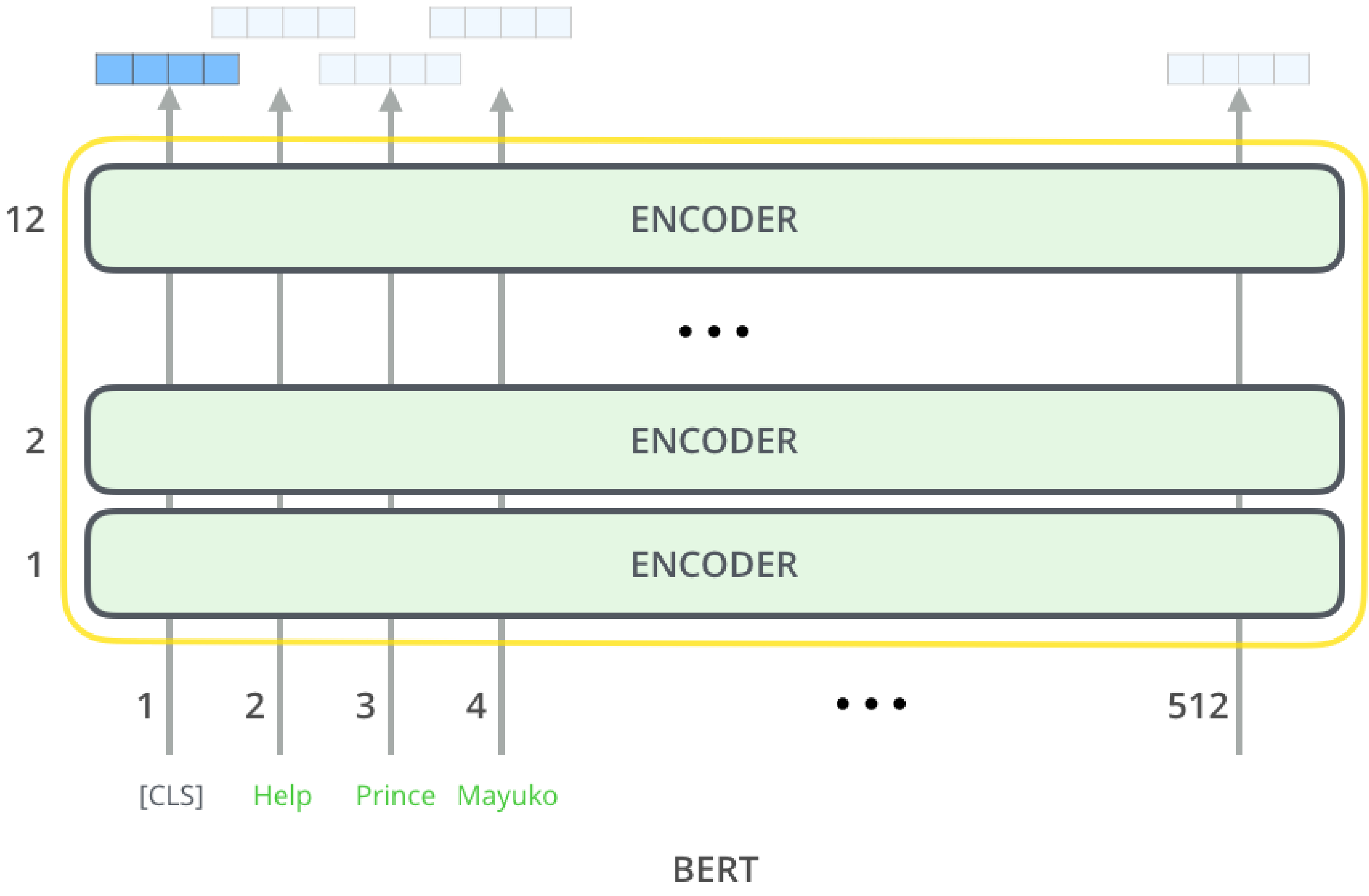


12
ENCODER
2
ENCODER
1
ENCODER
BERTBASE
24
ENCODER
4
ENCODER
3
ENCODER
2
ENCODER
1
ENCODER
BERTLARGE

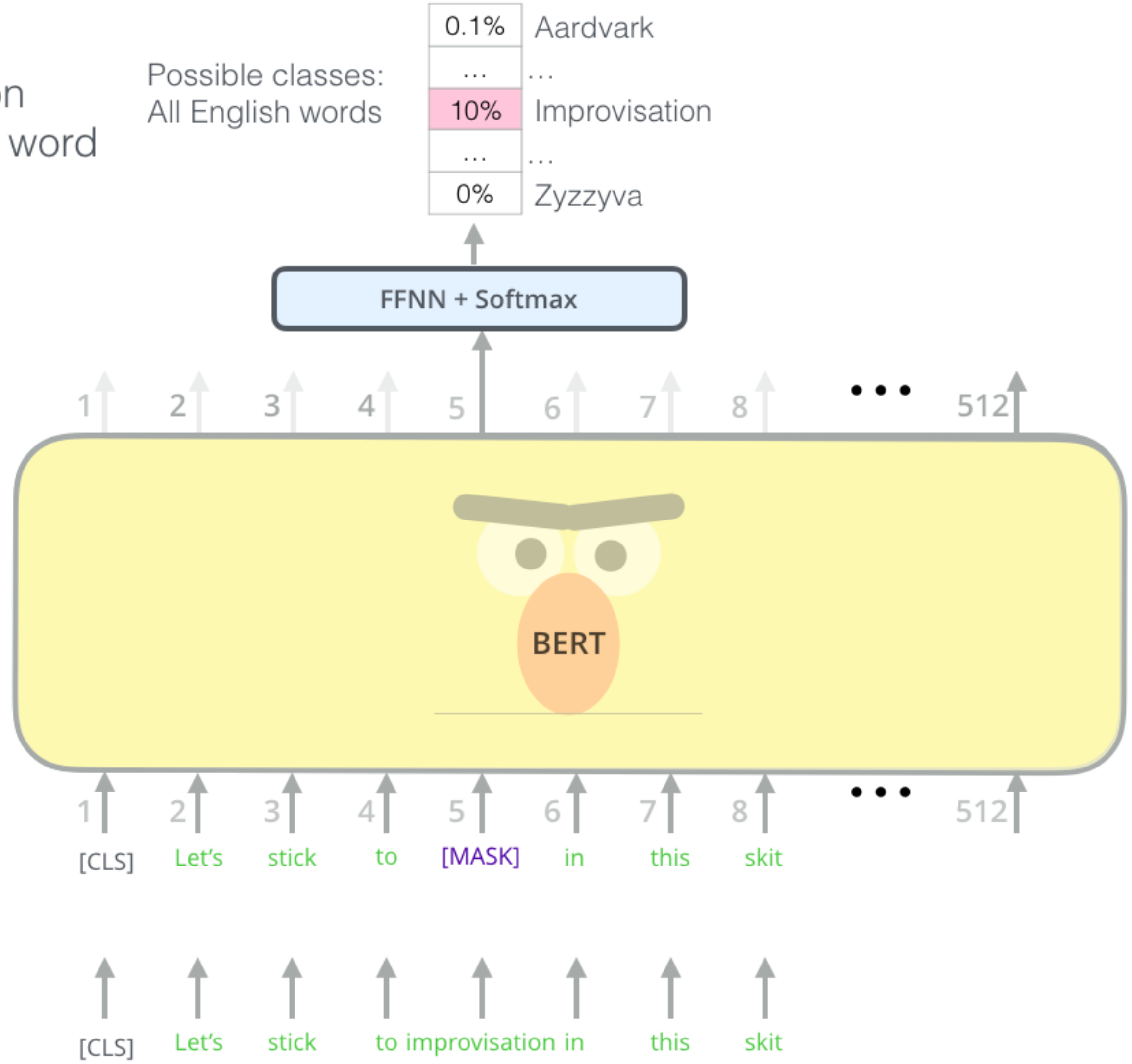
Input



Encoder (BERT-base)

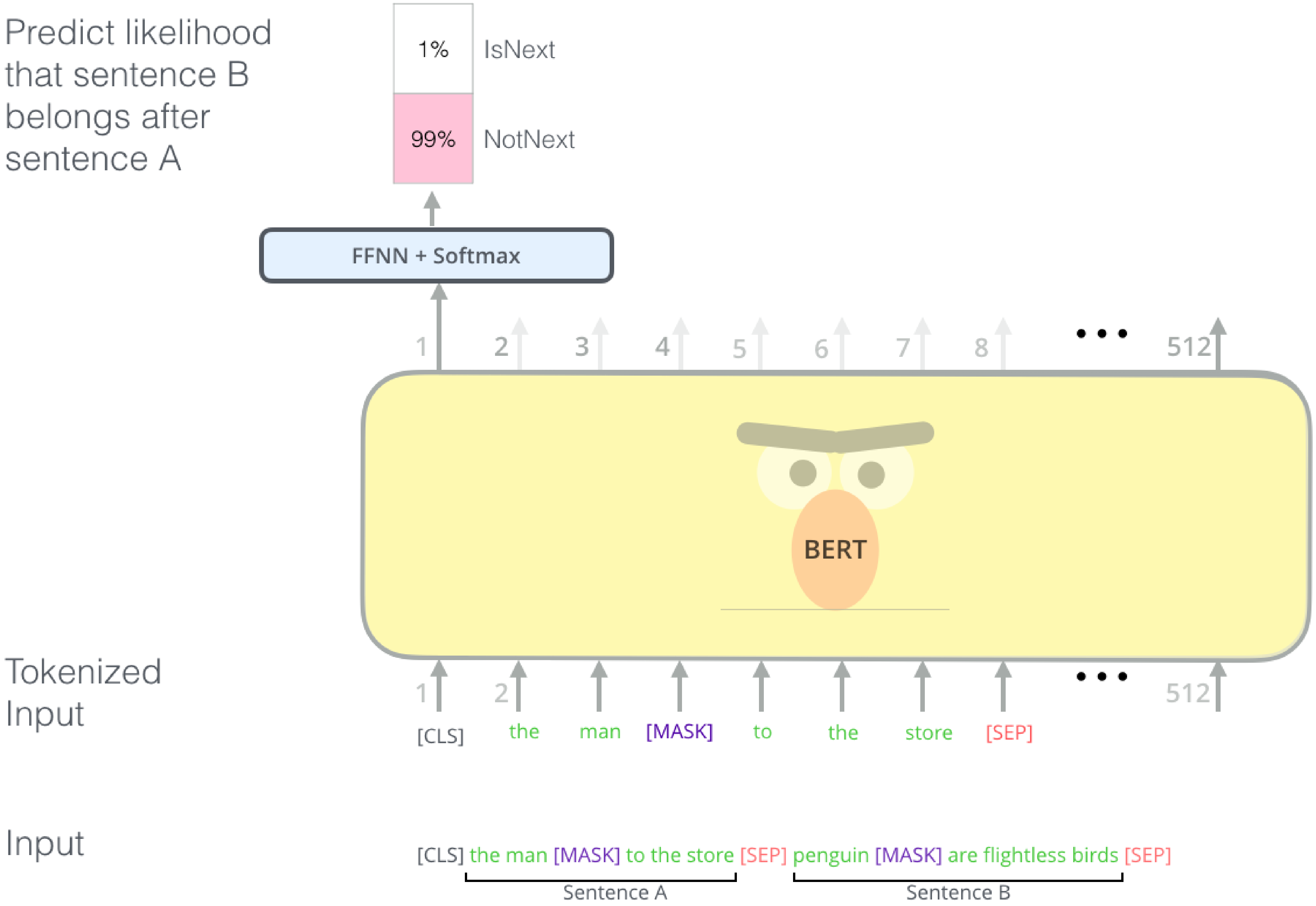


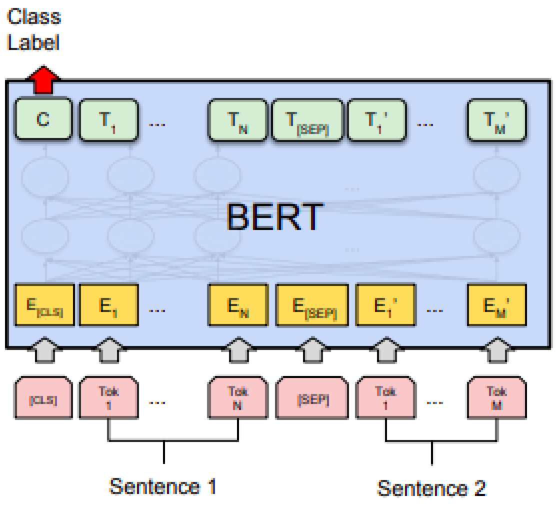
Use the output of the masked word's position to predict the masked word



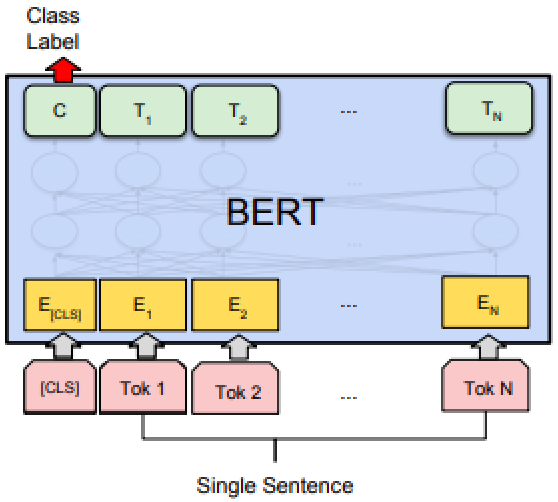
Randomly mask 15% of tokens

Input

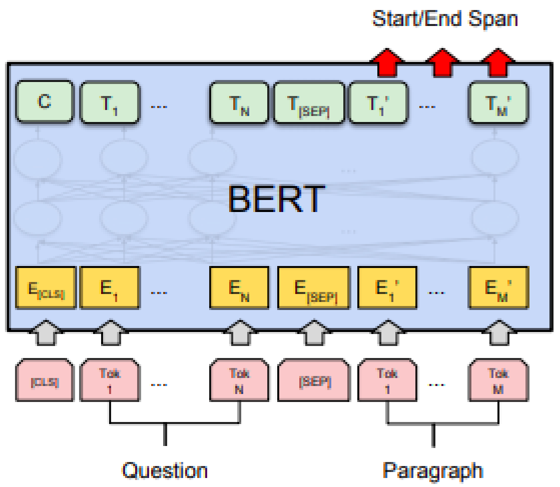

Predict likelihood that sentence B belongs after sentence A
1%
IsNext
99%
NotNext
FFNN + Softmax
1 2 3 4 5 6 7 8 ... 512
BERT
Tokenized Input
1 2 ... 512
[CLS] the man [MASK] to the store [SEP]
Input
[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]
Sentence A
Sentence B



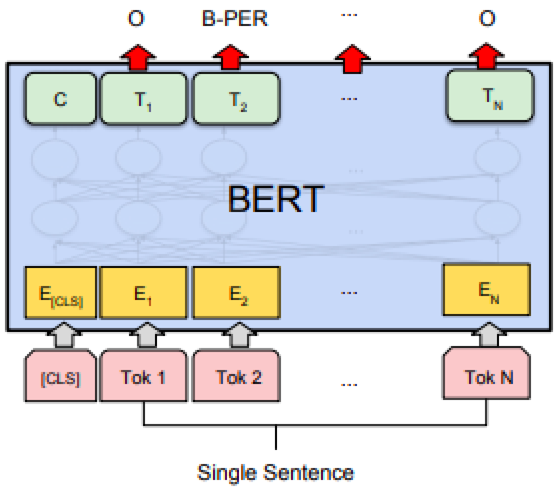
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA

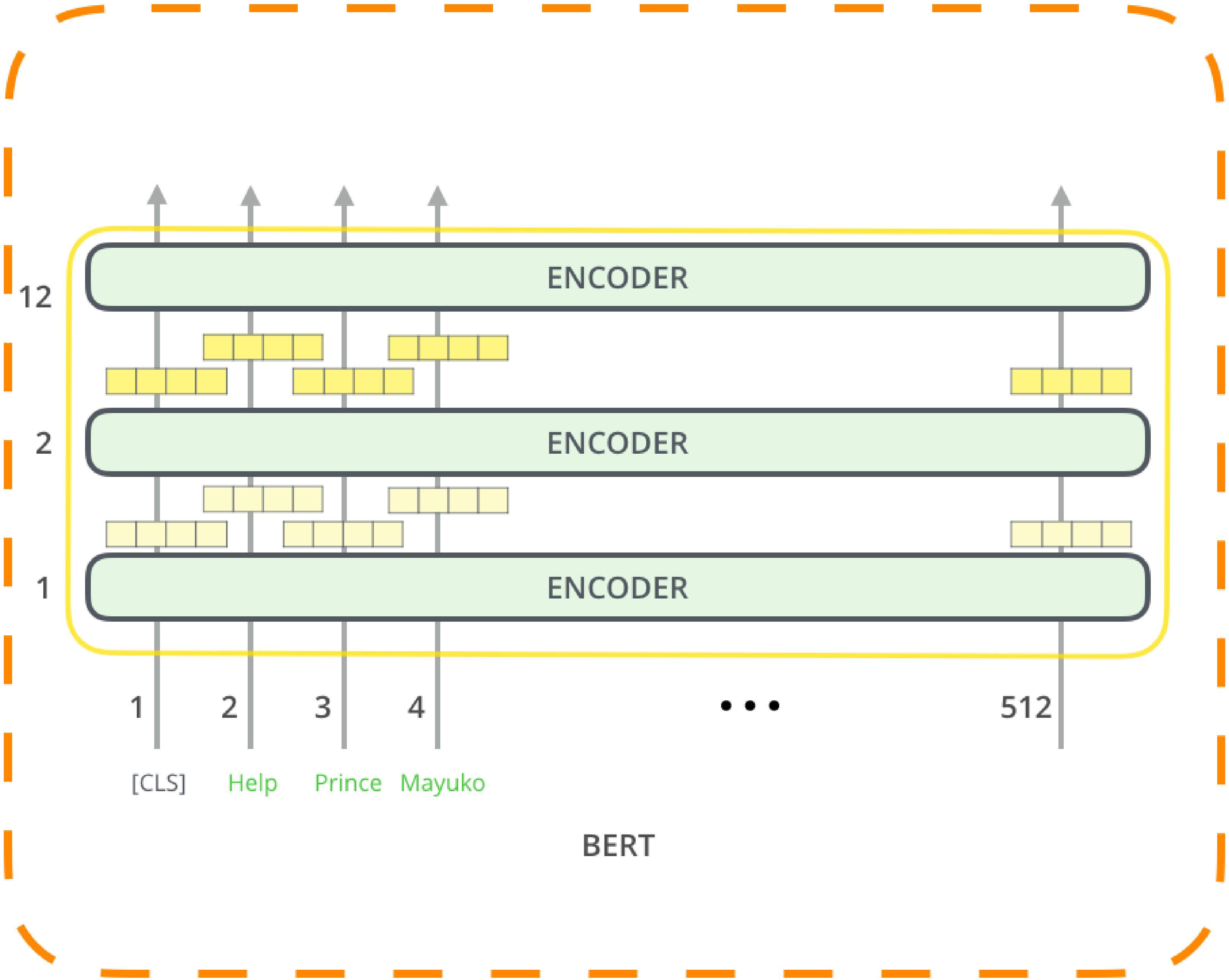


(c) Question Answering Tasks: SQuAD v1.1

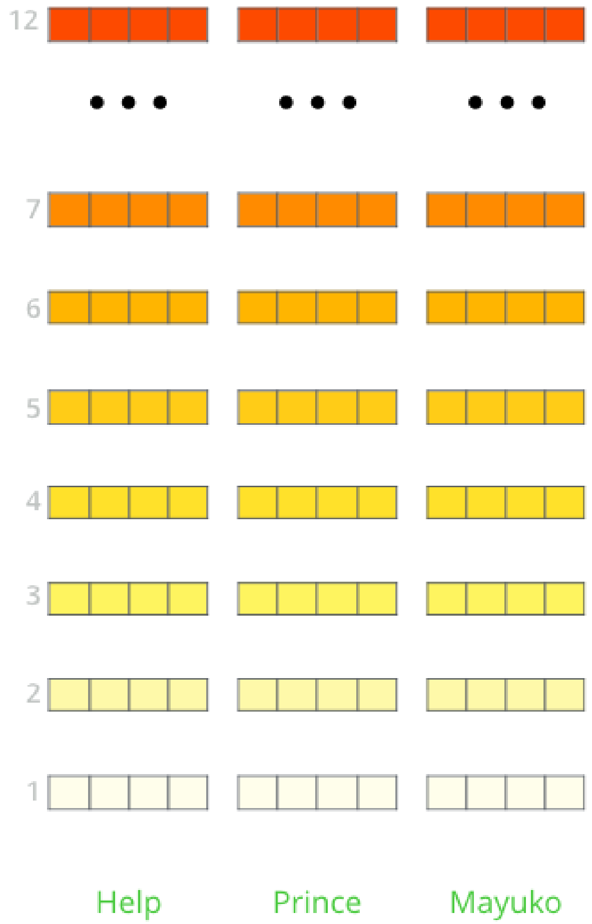


(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Generate Contexualized Embeddings



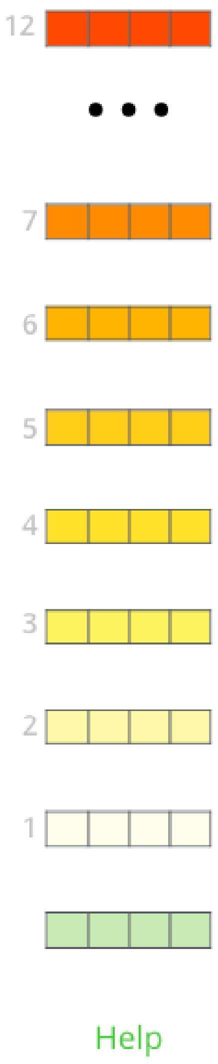
The output of each encoder layer along each token's path can be used as a feature representing that token.



But which one should we use?

What is the best contextualized embedding for “Help” in that context?

For named-entity recognition task CoNLL-2003 NER



	Layers used	Dev F1 Score
First Layer	Embedding	91.0
Last Hidden Layer	12	94.9
Sum All 12 Layers	12 + … + 2 + 1 =	95.5
Second-to-Last Hidden Layer	11	95.6
Sum Last Four Hidden	12 + 11 + 10 + 9 =	95.9
Concat Last Four Hidden	9 10 11 12	96.1

What's wrong with BERT?

- The [MASK] token used in training does not appear during fine-tuning

- BERT generates predictions independently
  - BERT predicts masked tokens in parallel.
  - During training, it does not learn to handle dependencies between predicting simultaneously masked tokens

Content from Keita Kurita

Newer models

- Transformer XL
- XLNet
- RoBERTa
- DistillBERT
- ALBERT
- XLM-Roberta
- …
- ParsBERT

GPT-3

175 billion parameters

GPT-2 had 1.5B

The largest so far (by Microsoft) had 17B

Training cost: $12M

Questions