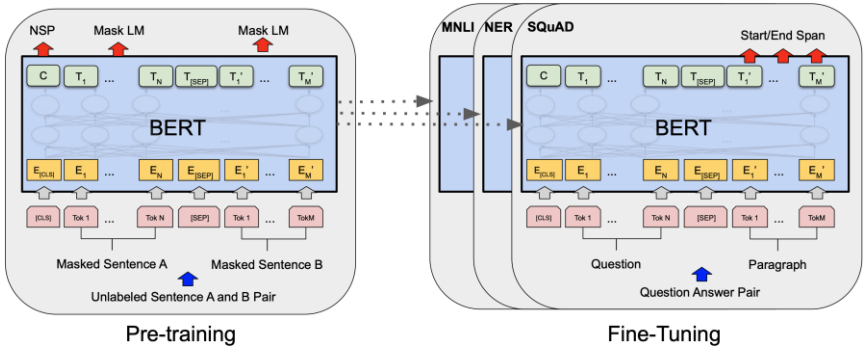


بنام خداوند جان و خرد

Transformers and BERT

Mohammad Taher Pilehvar

Natural Language Processing 1400
https://teias-courses.github.io/nlp00/

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

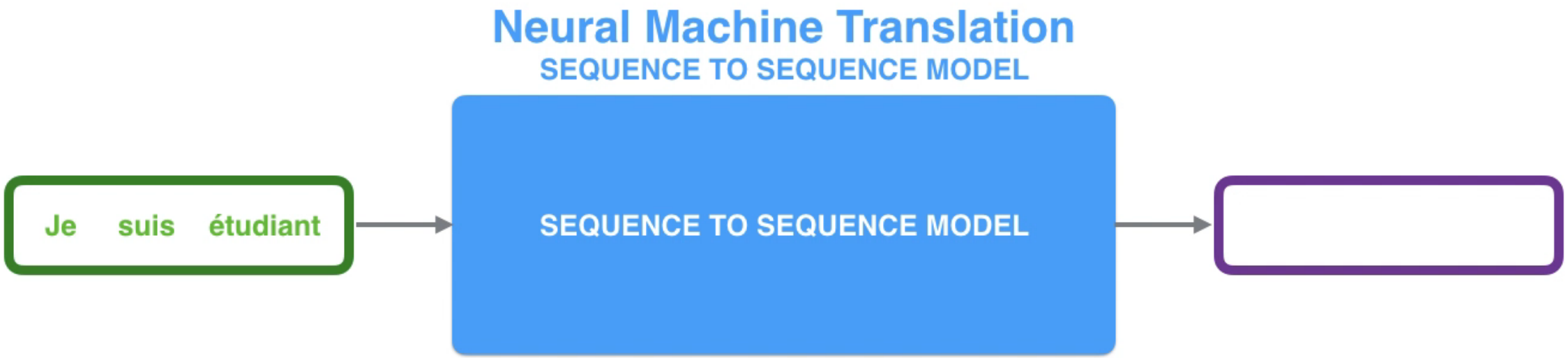
TeIAS | Tehran Institute for Advanced Studies

Attention

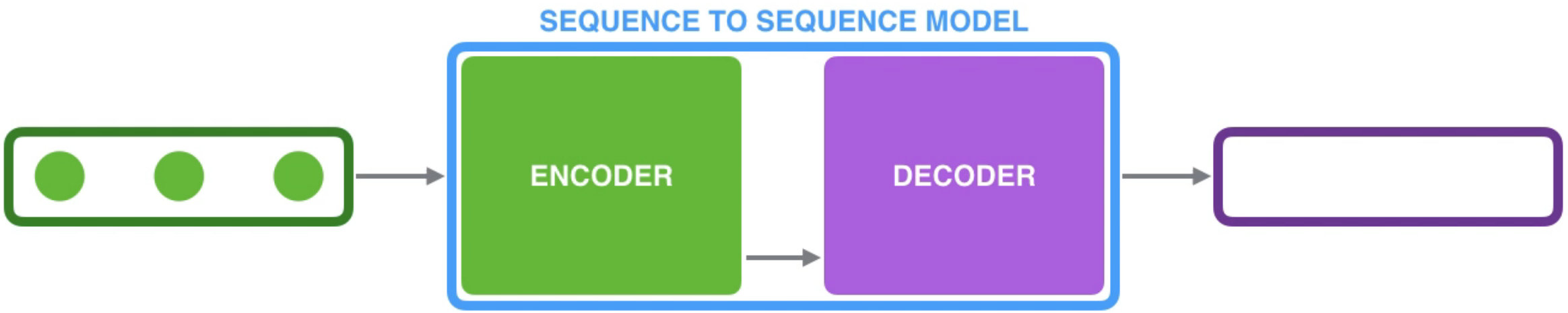
Sequence to Sequence model

Attention

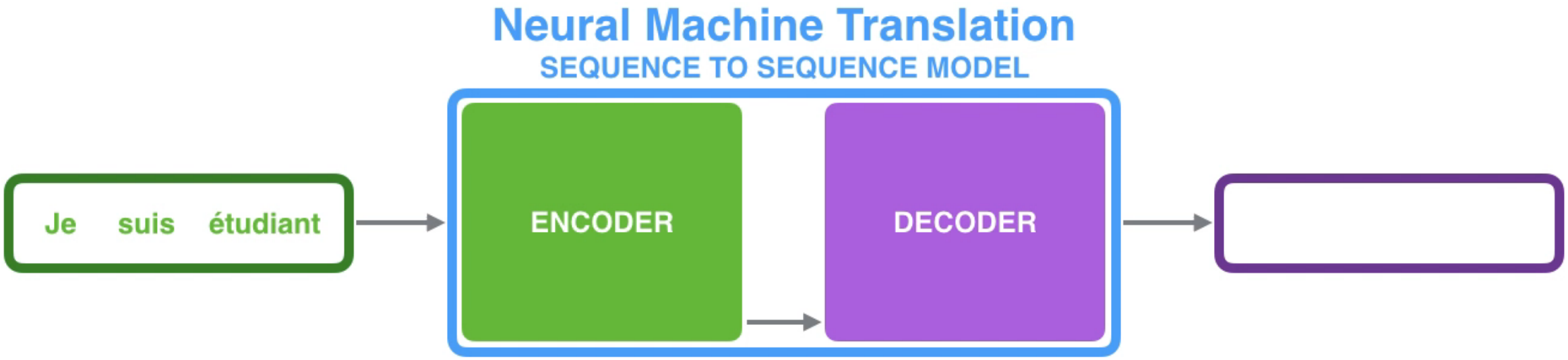
Sequence to Sequence model



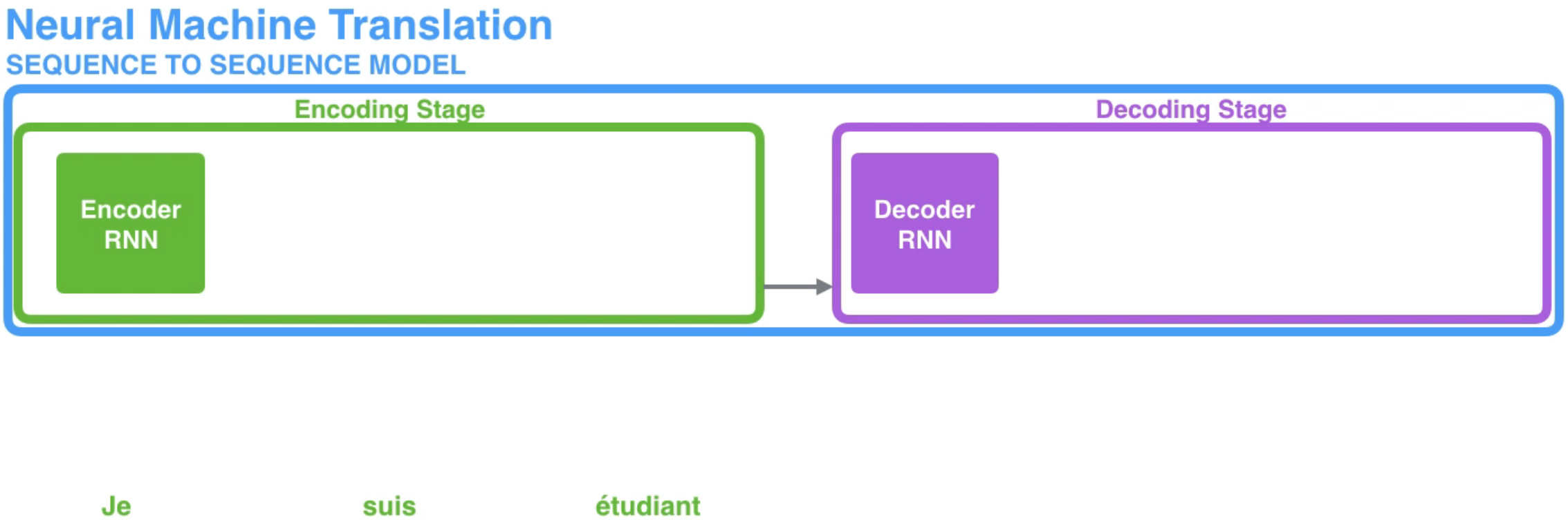
Attention



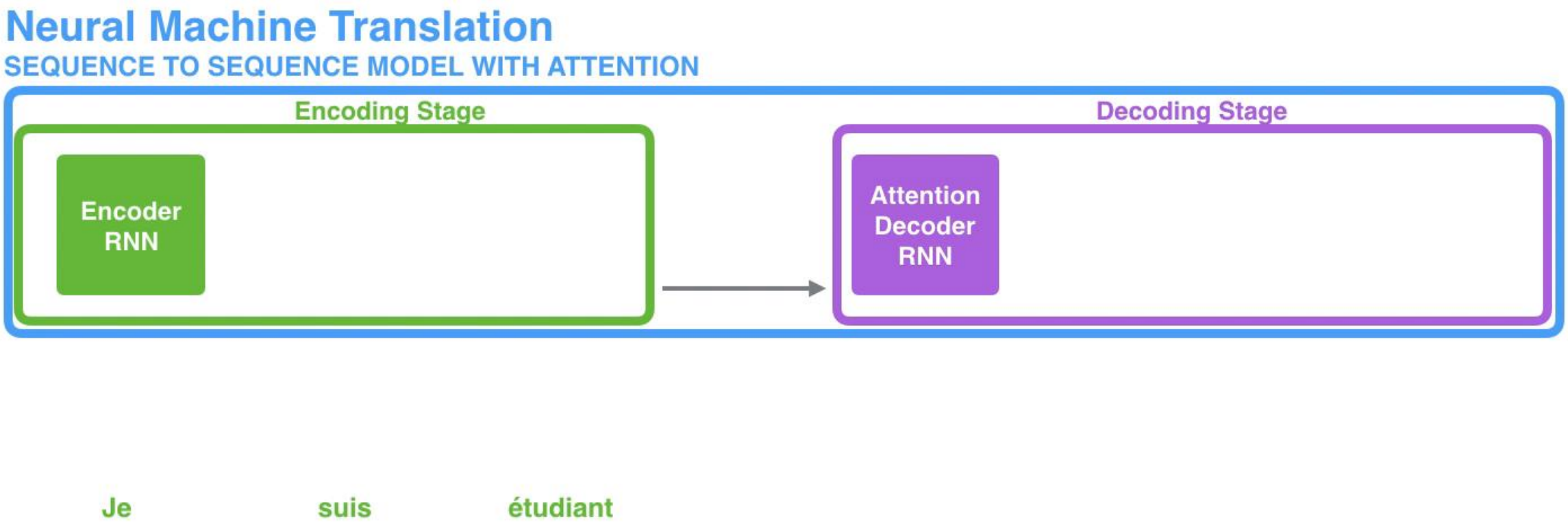
Attention


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

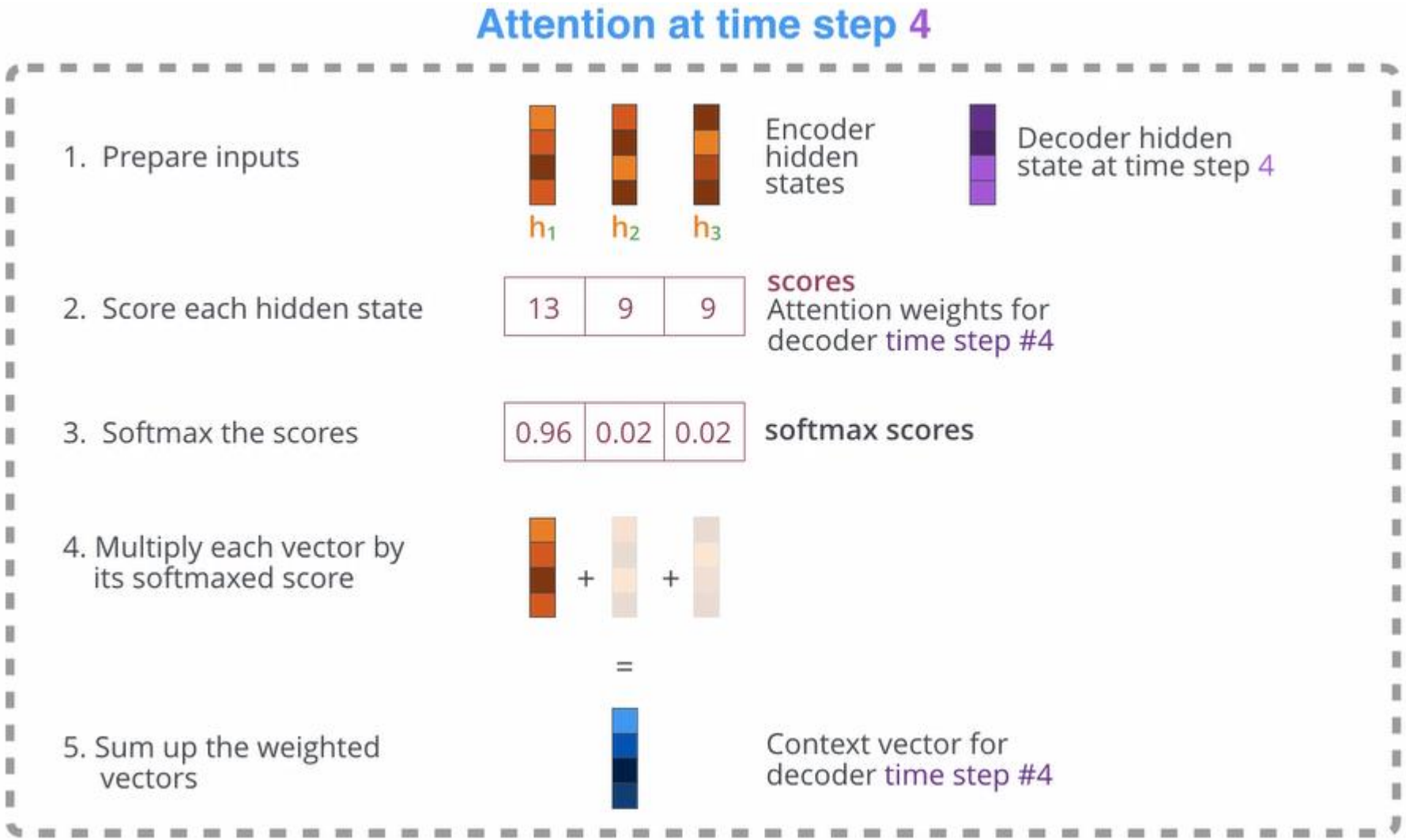
Attention


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

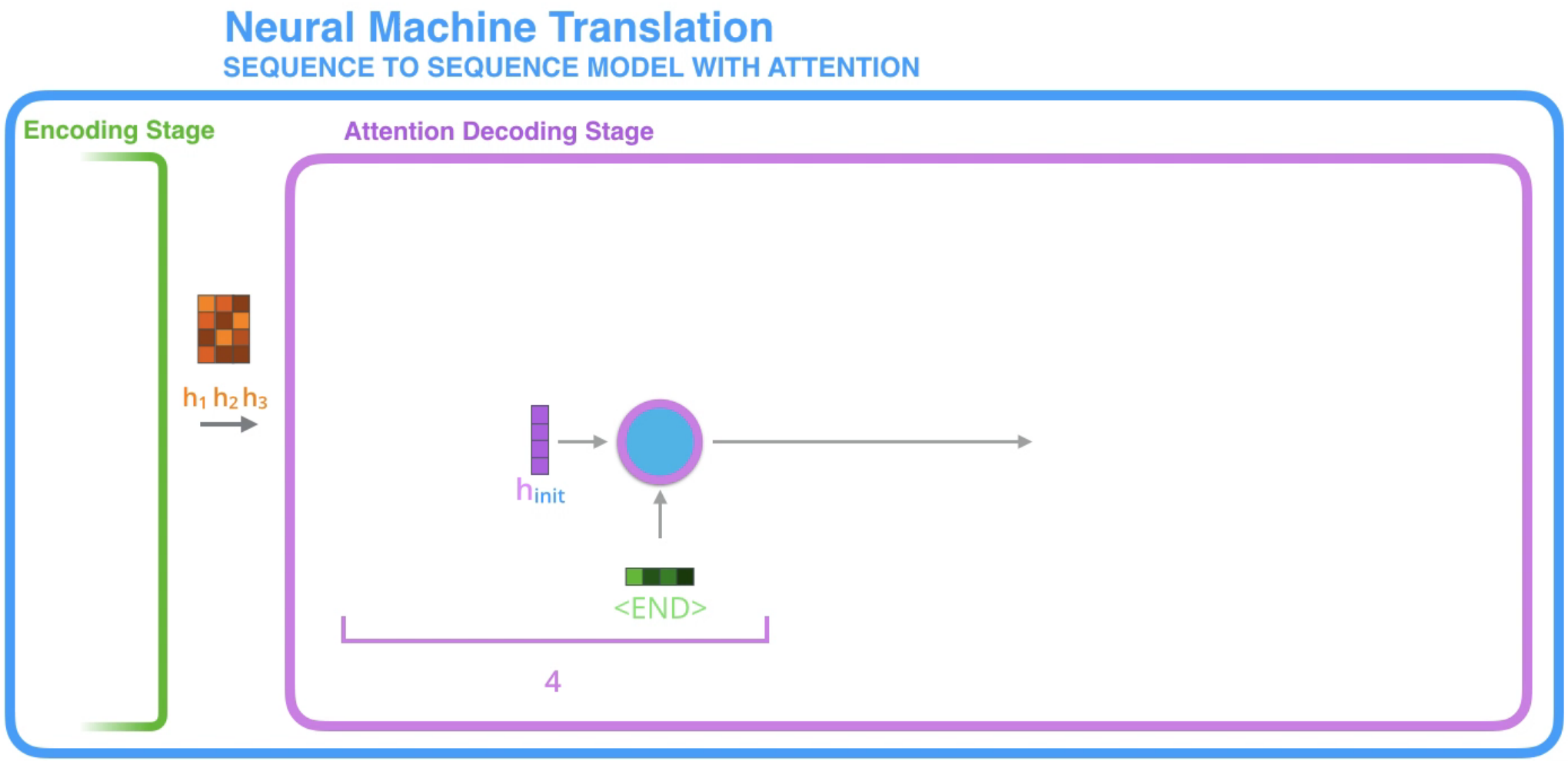
Attention


Attention at time step 4
1. Prepare inputs
h1
h2
h3
Encoder hidden states
Decoder hidden state at time step 4
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

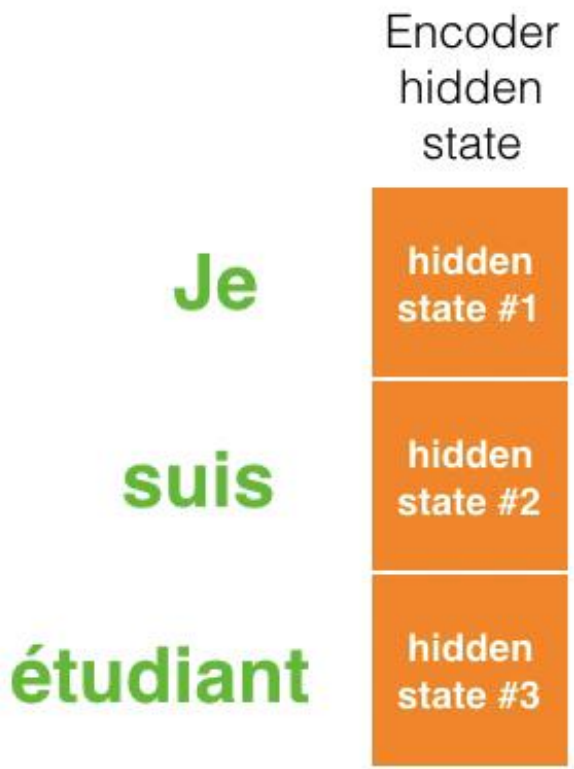
Attention

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

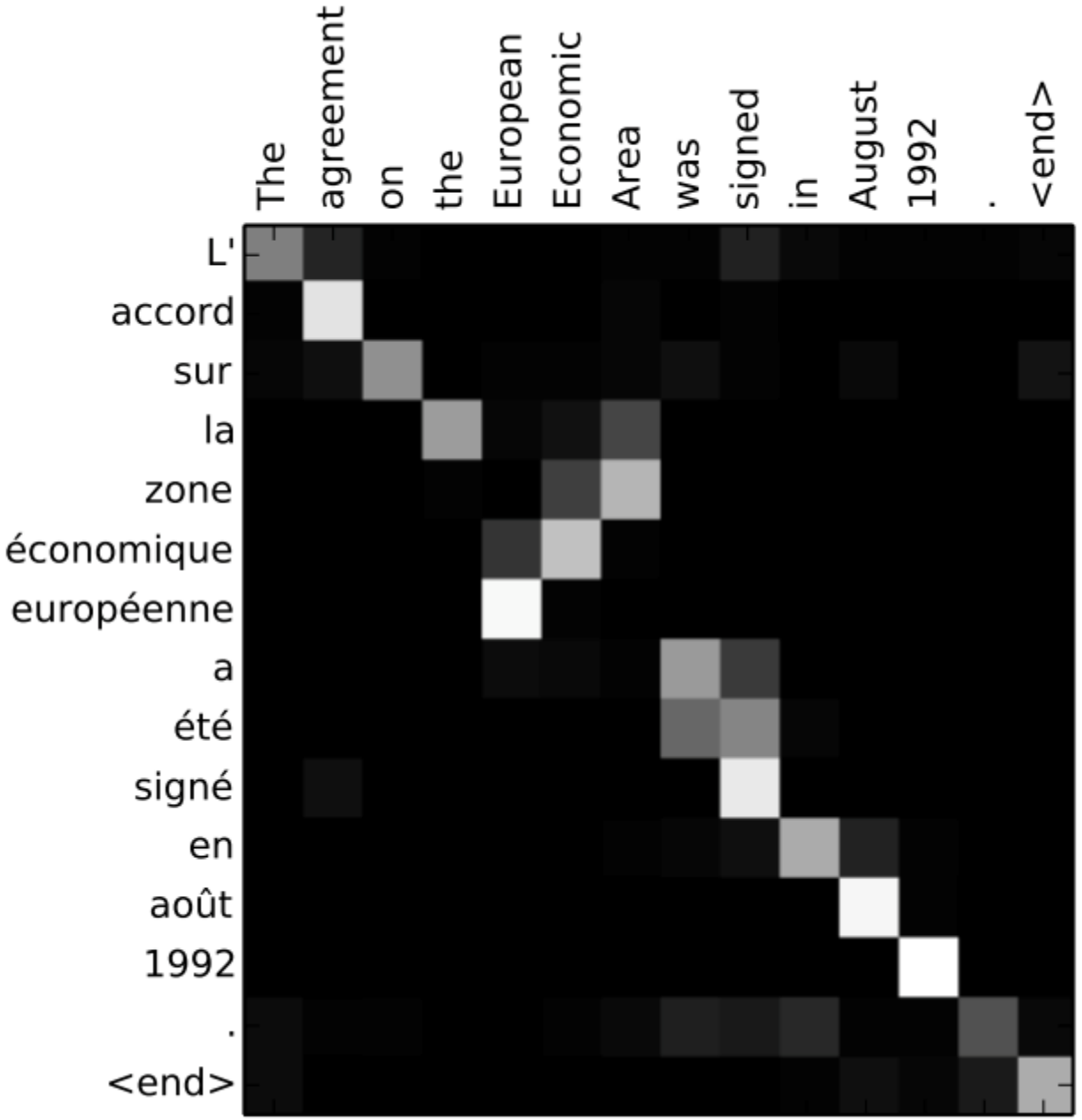
Attention


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

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

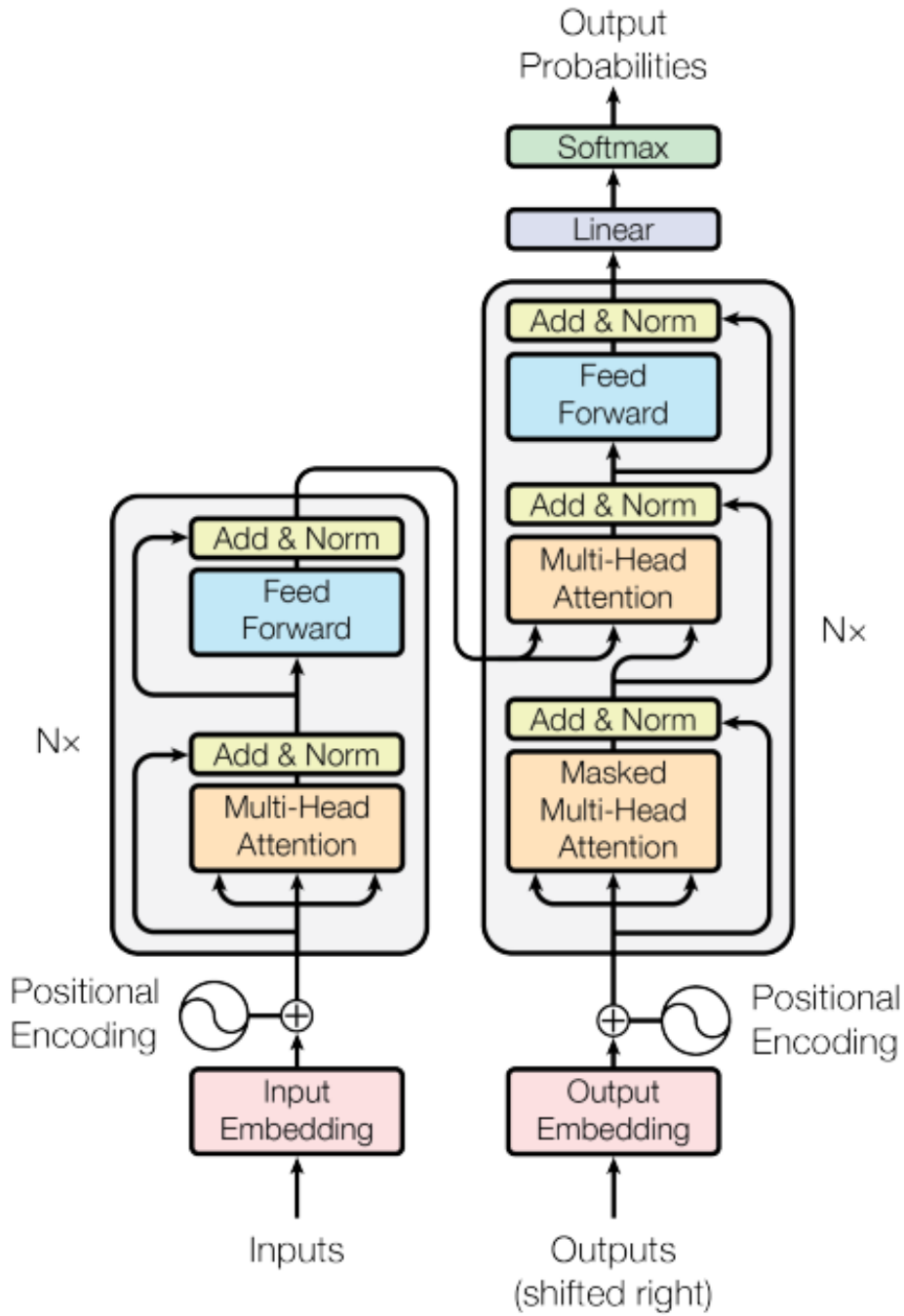
Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

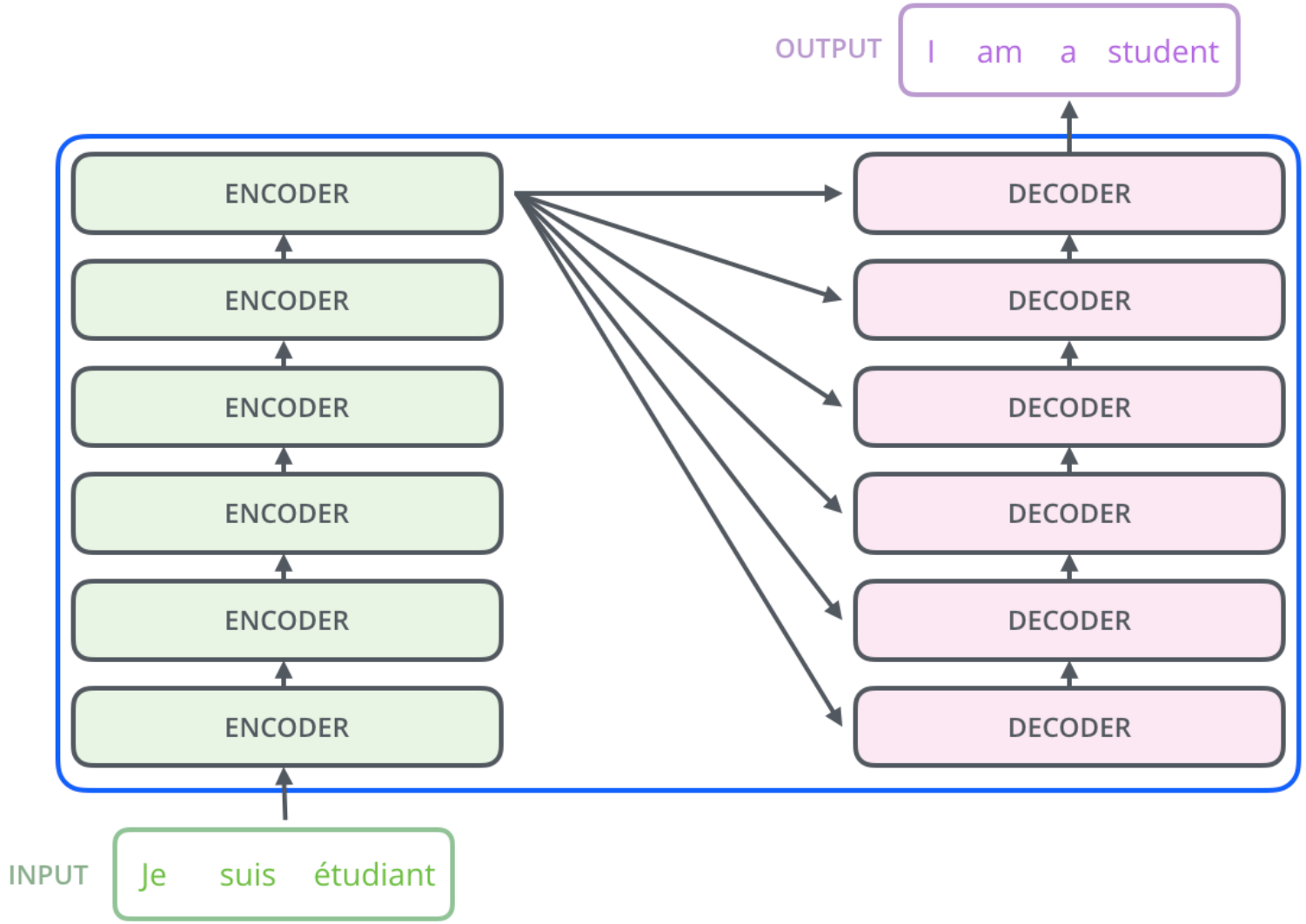
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

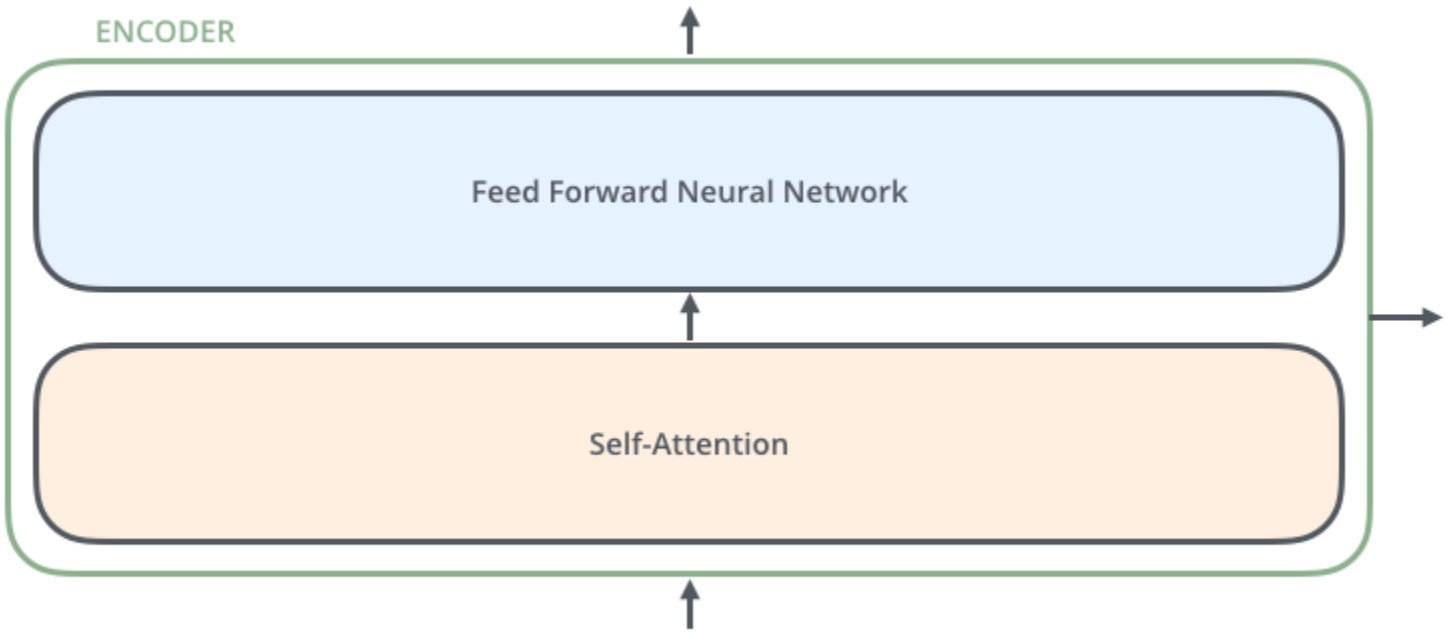
Transformers



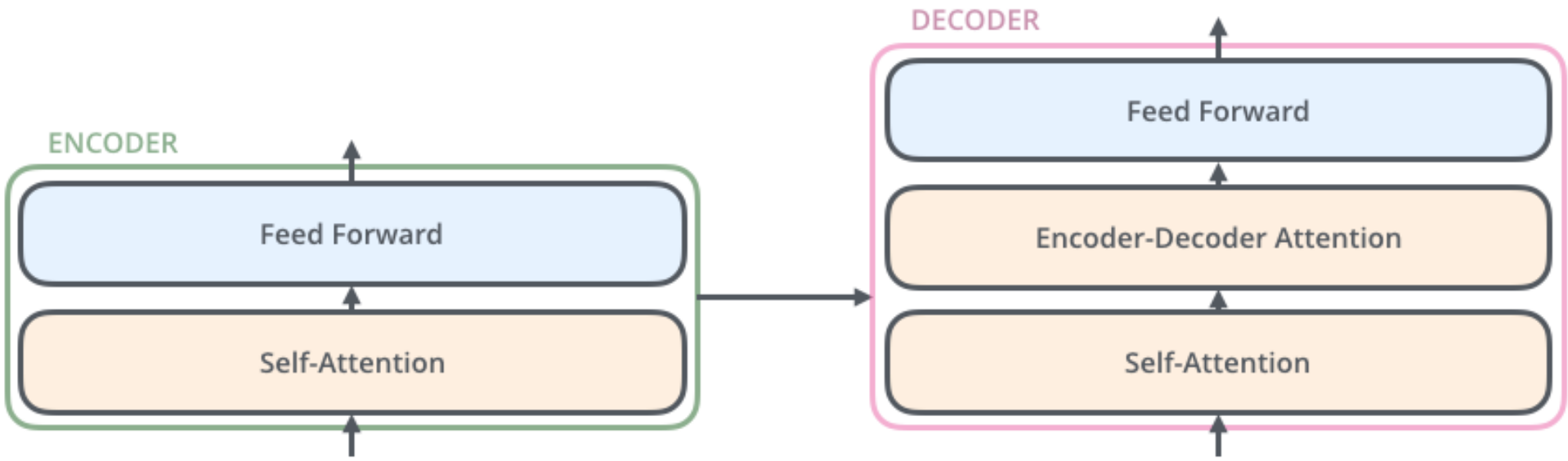
Transformers



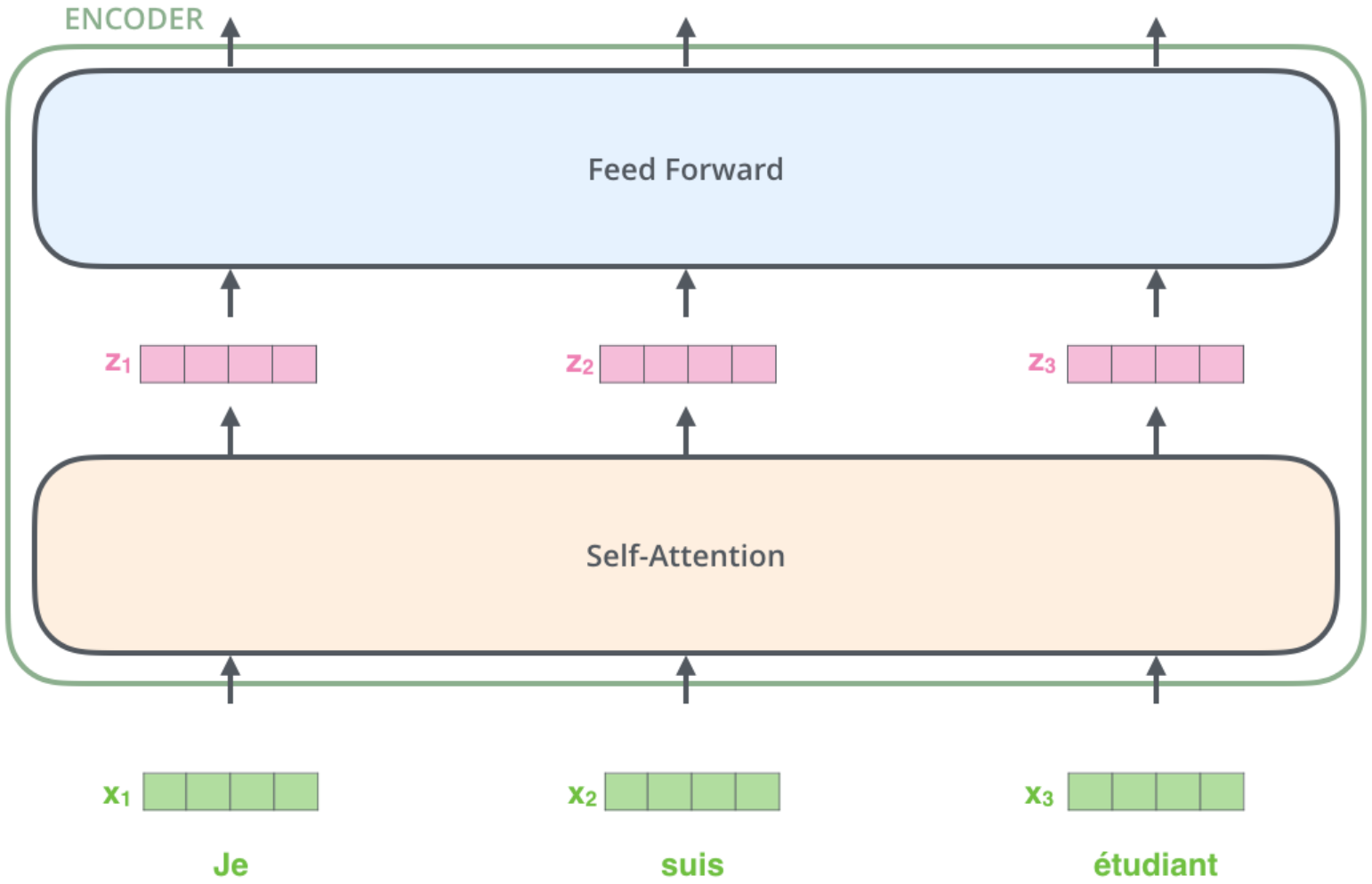
Transformer
Encoder



Transformer
Encoder - Decoder

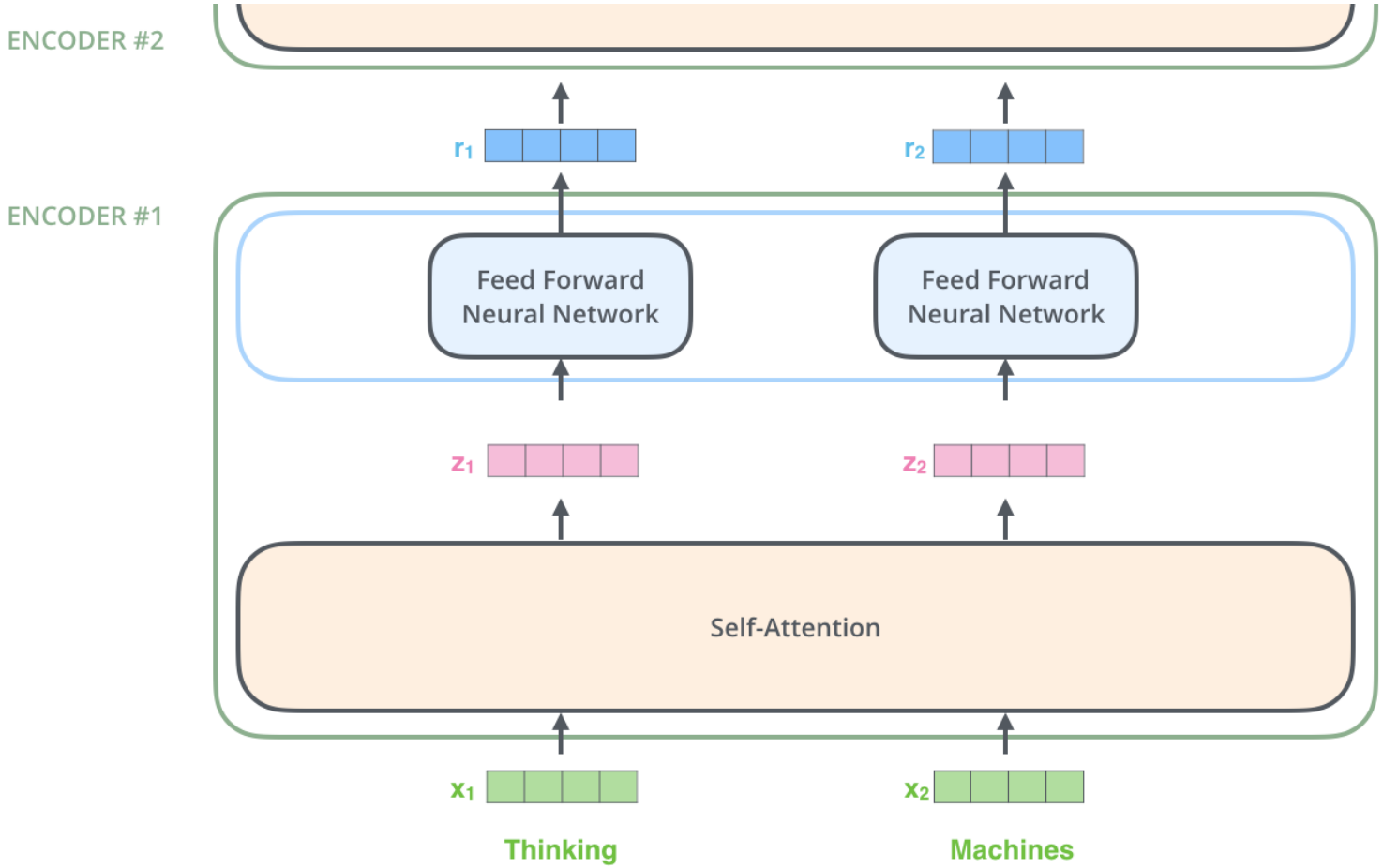


Encoder


ENCODER
Feed Forward
z1
z2
z3
Self-Attention
x1
x2
x3
Je
suis
étudiant

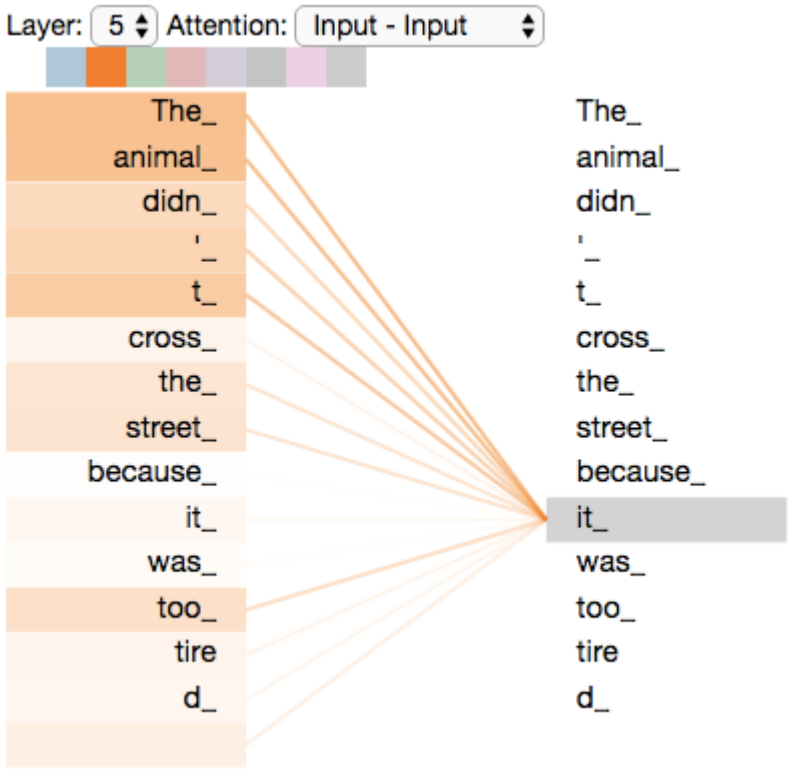
Encoder

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

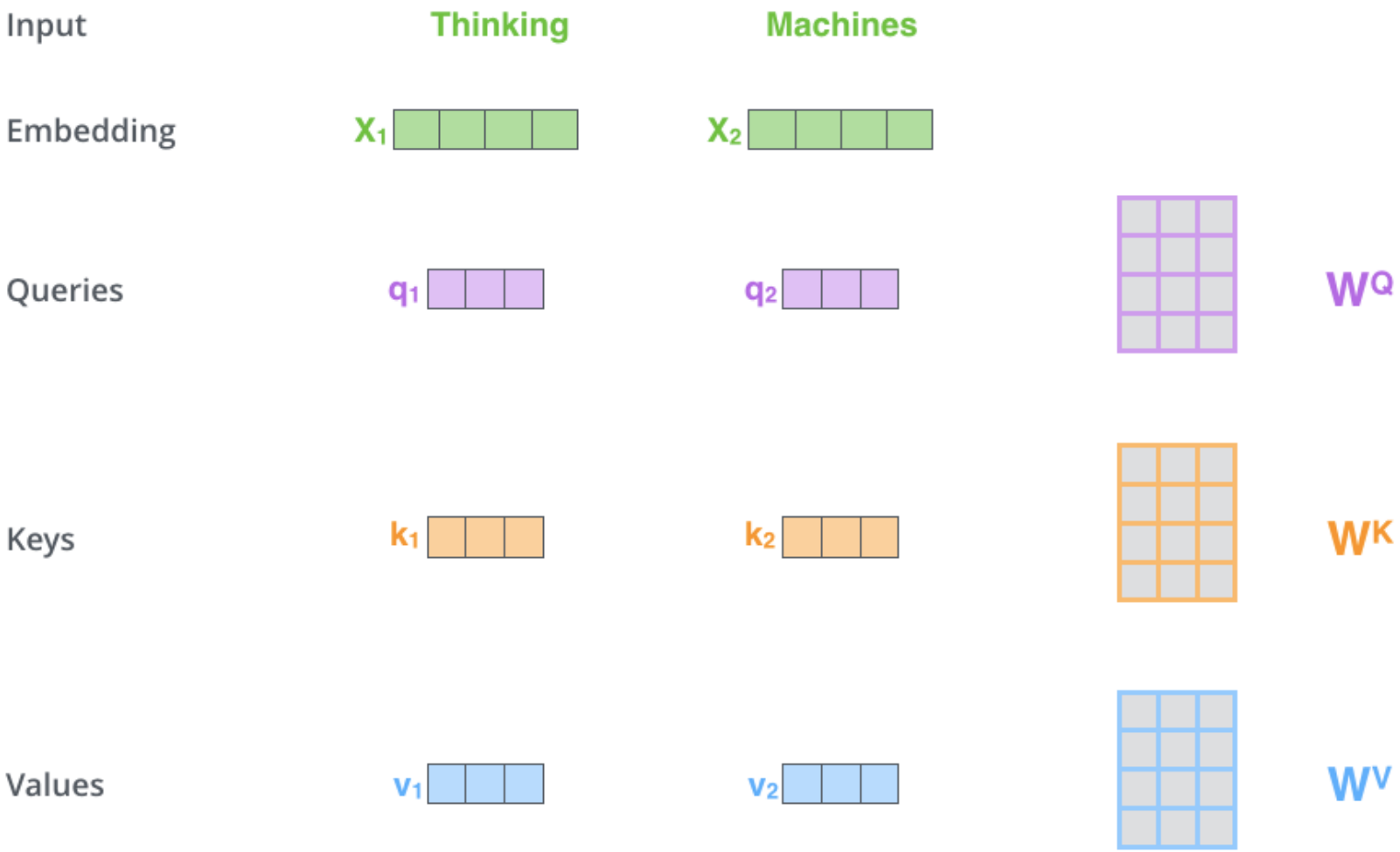


Self attention

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



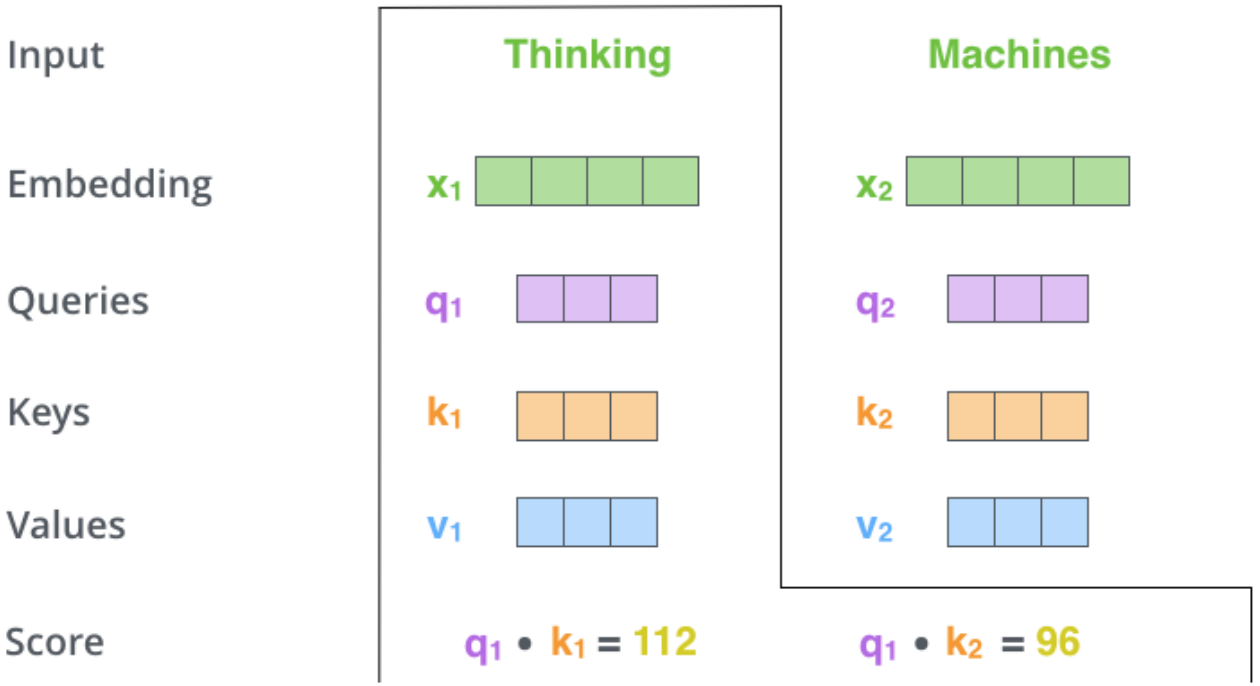
Self attention



Self attention

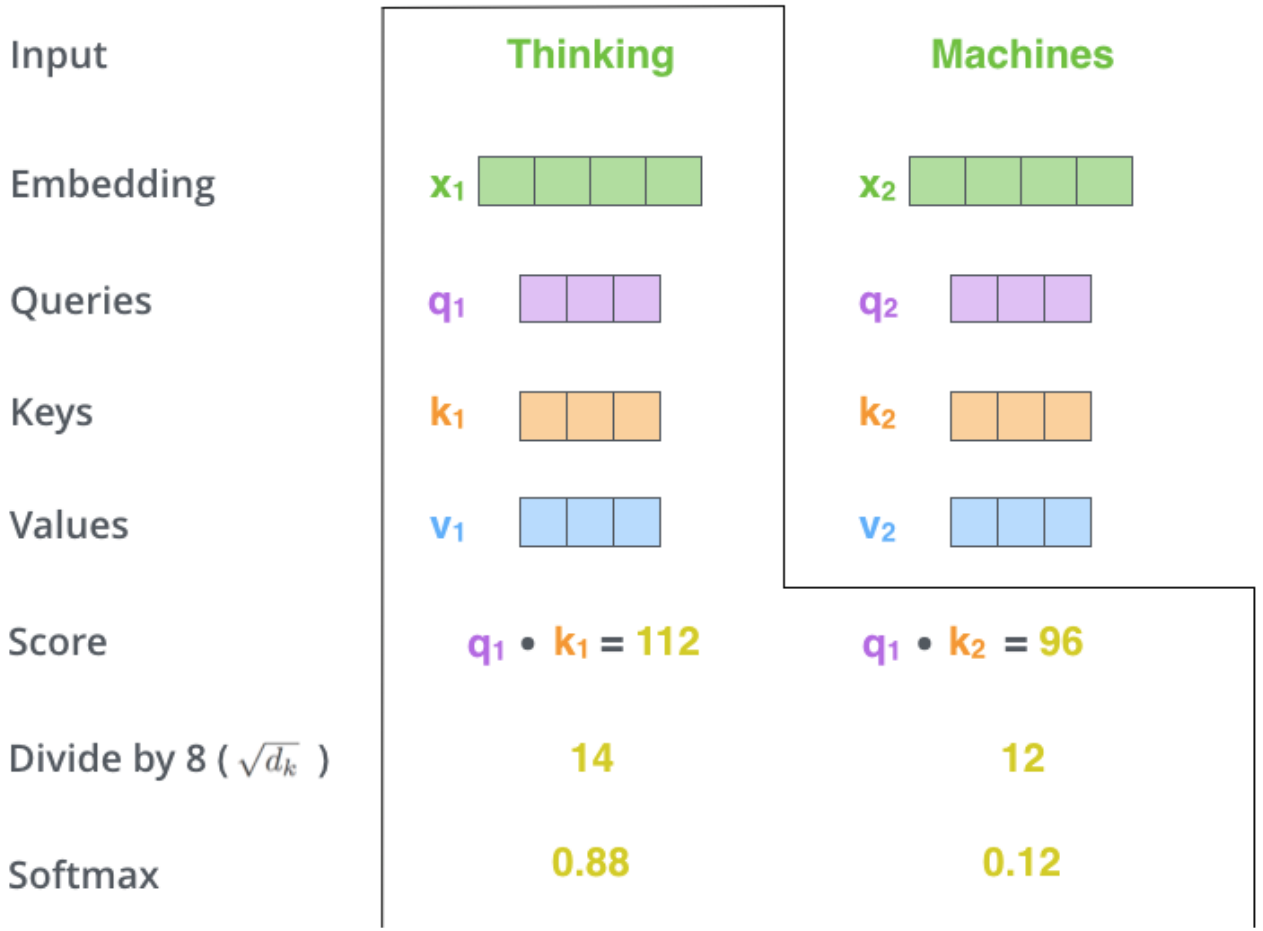
For first word: *thinking*

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

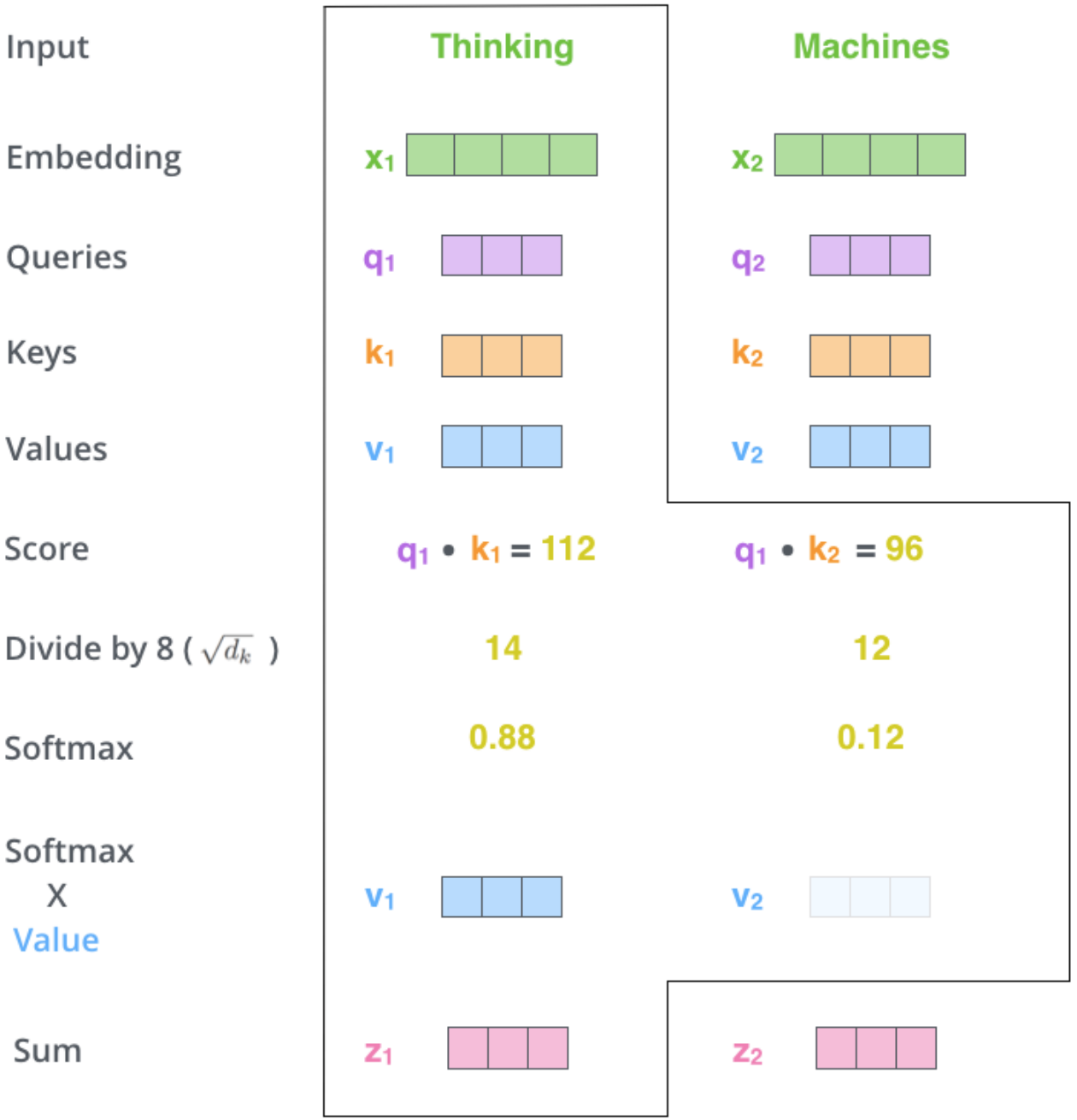


Self attention

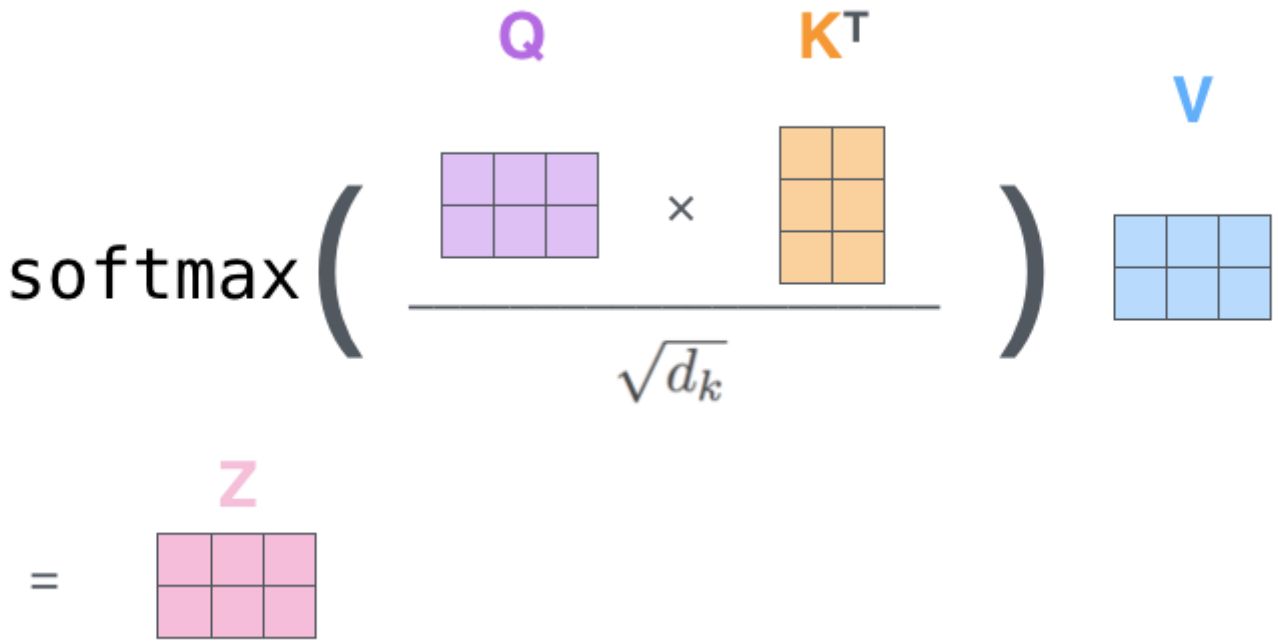
Multiply each value vector by the softmax score.

Sum up the weighted value vectors.

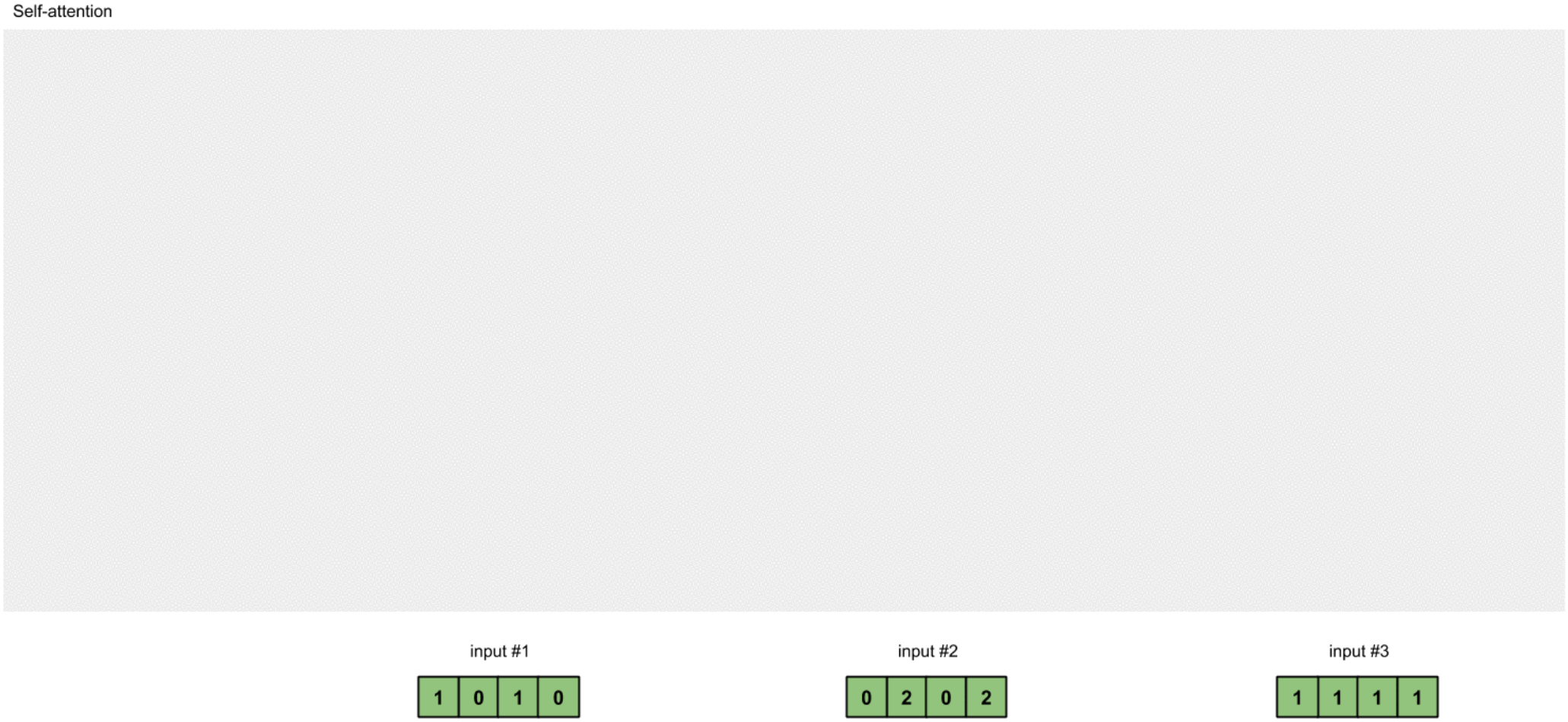
This gives the output for *thinking*



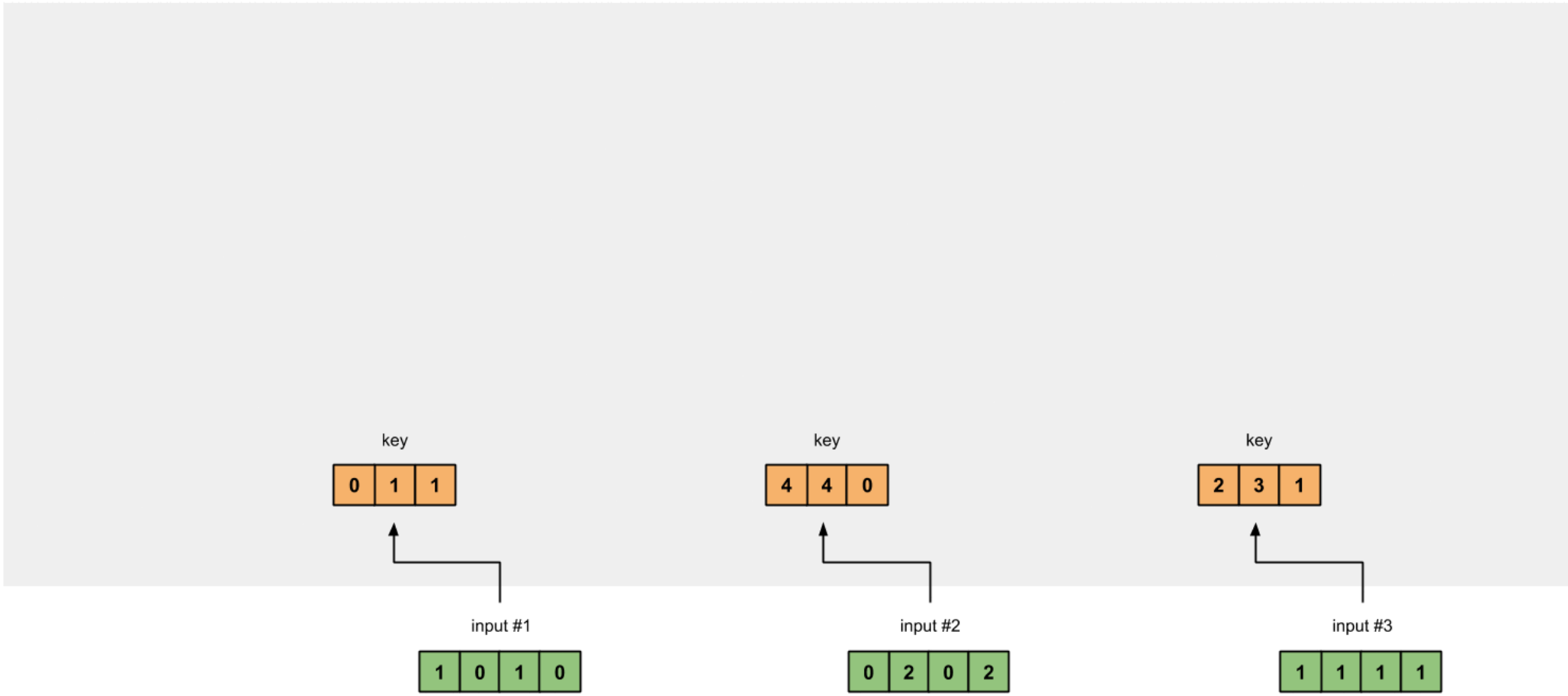
Self attention (matrix calculation)



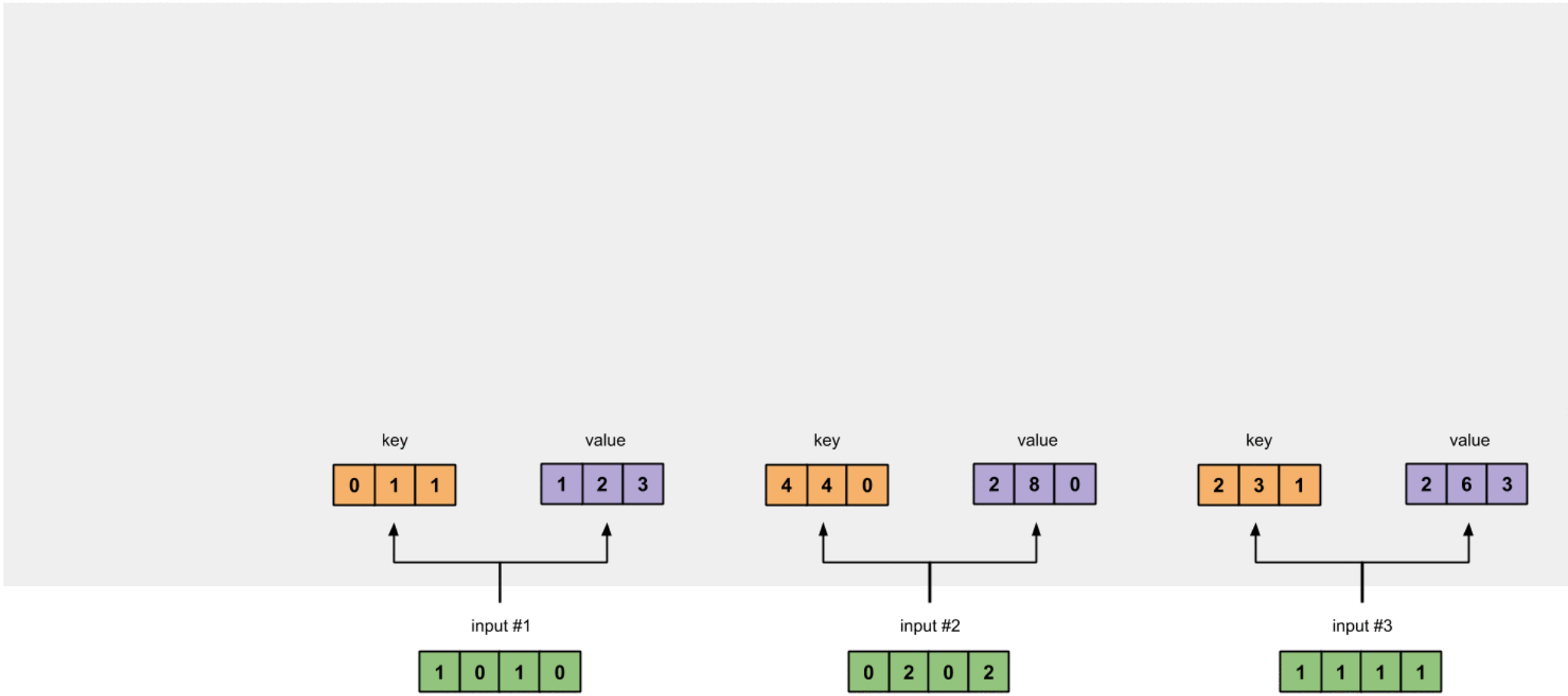
Self attention – from https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a



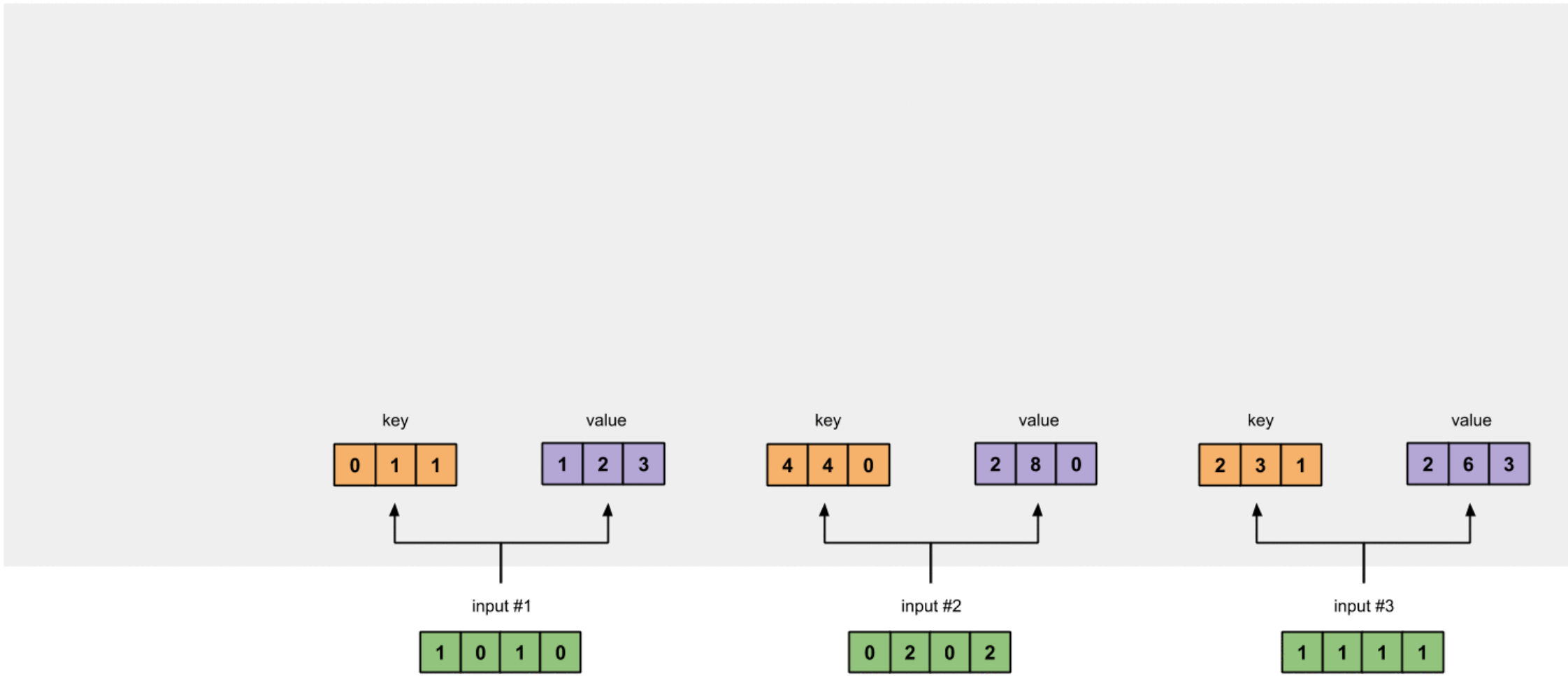
Self-attention



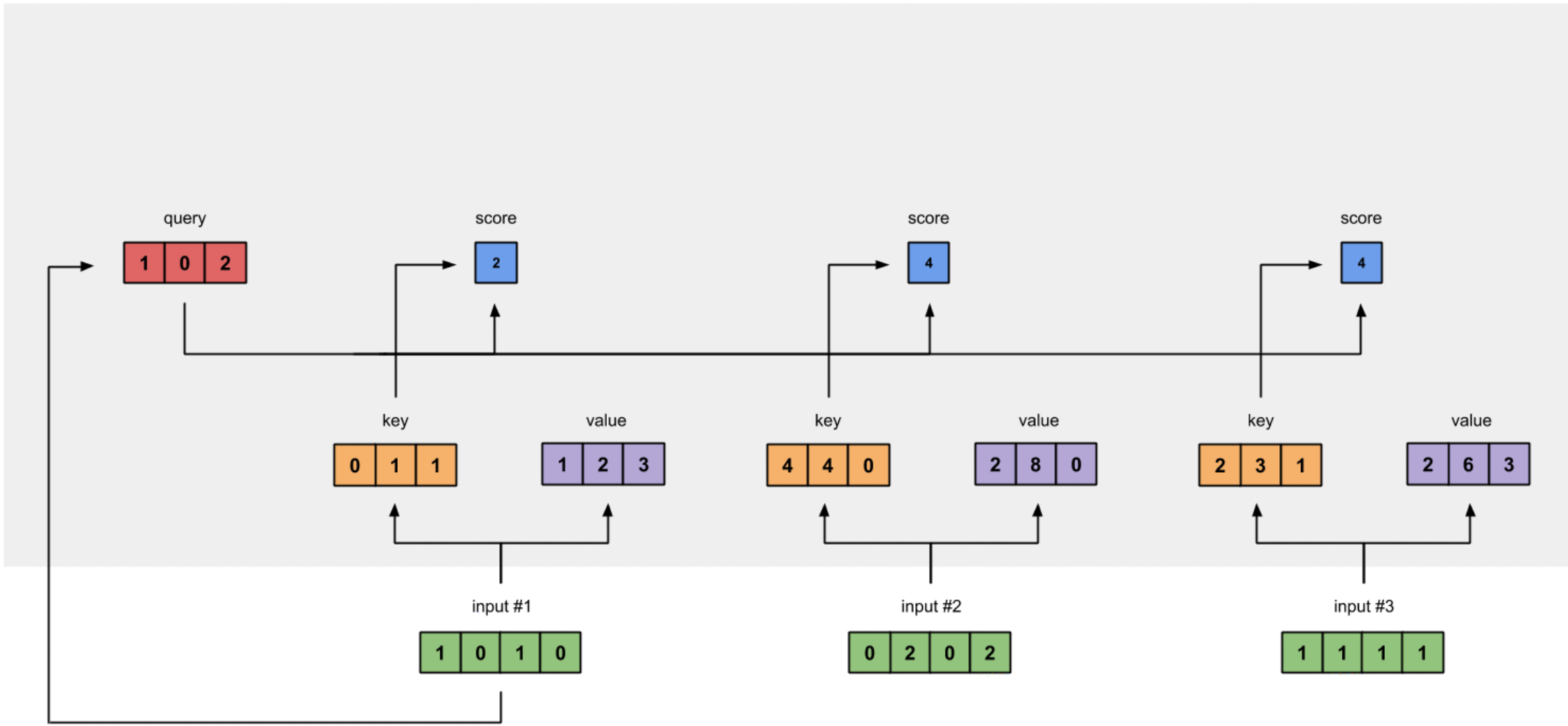
Self-attention



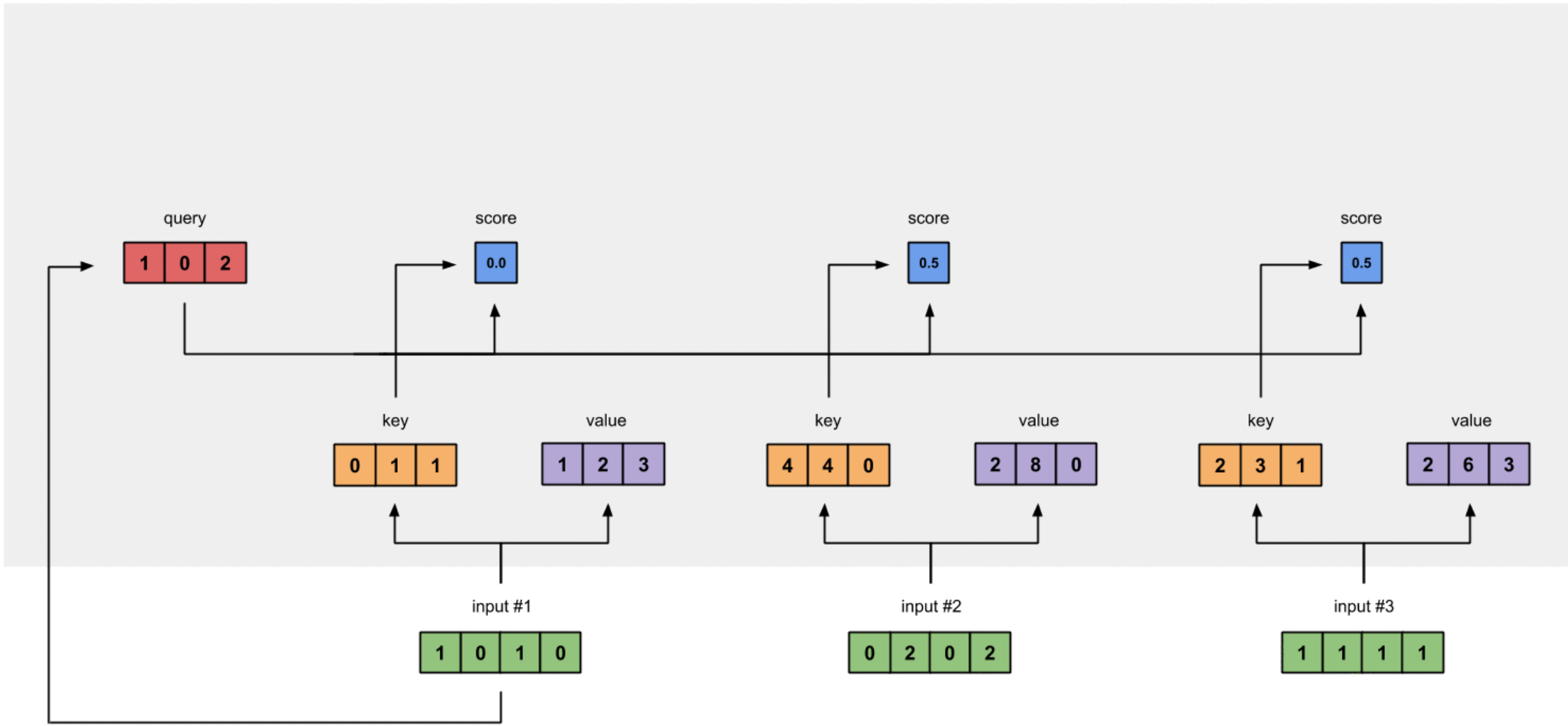
Self-attention



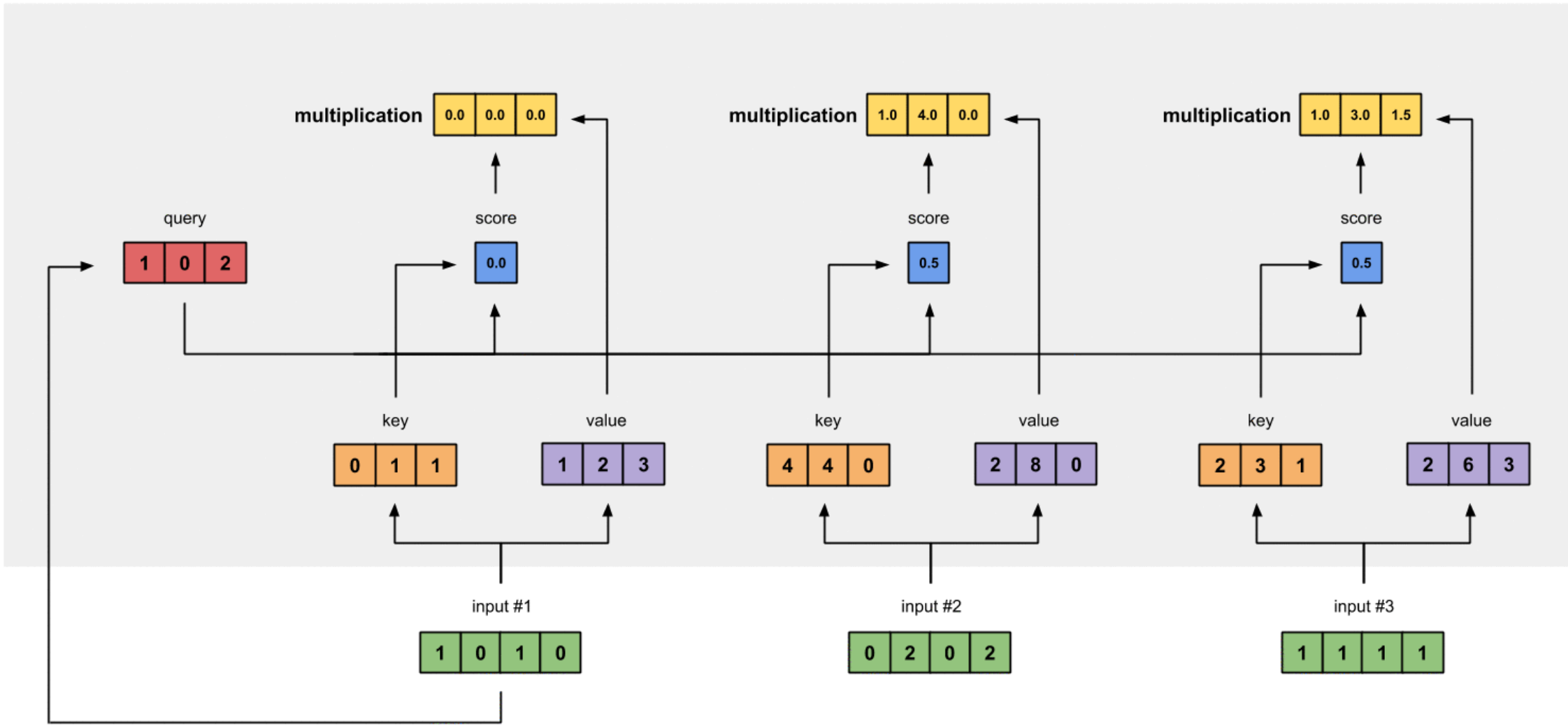
Self-attention

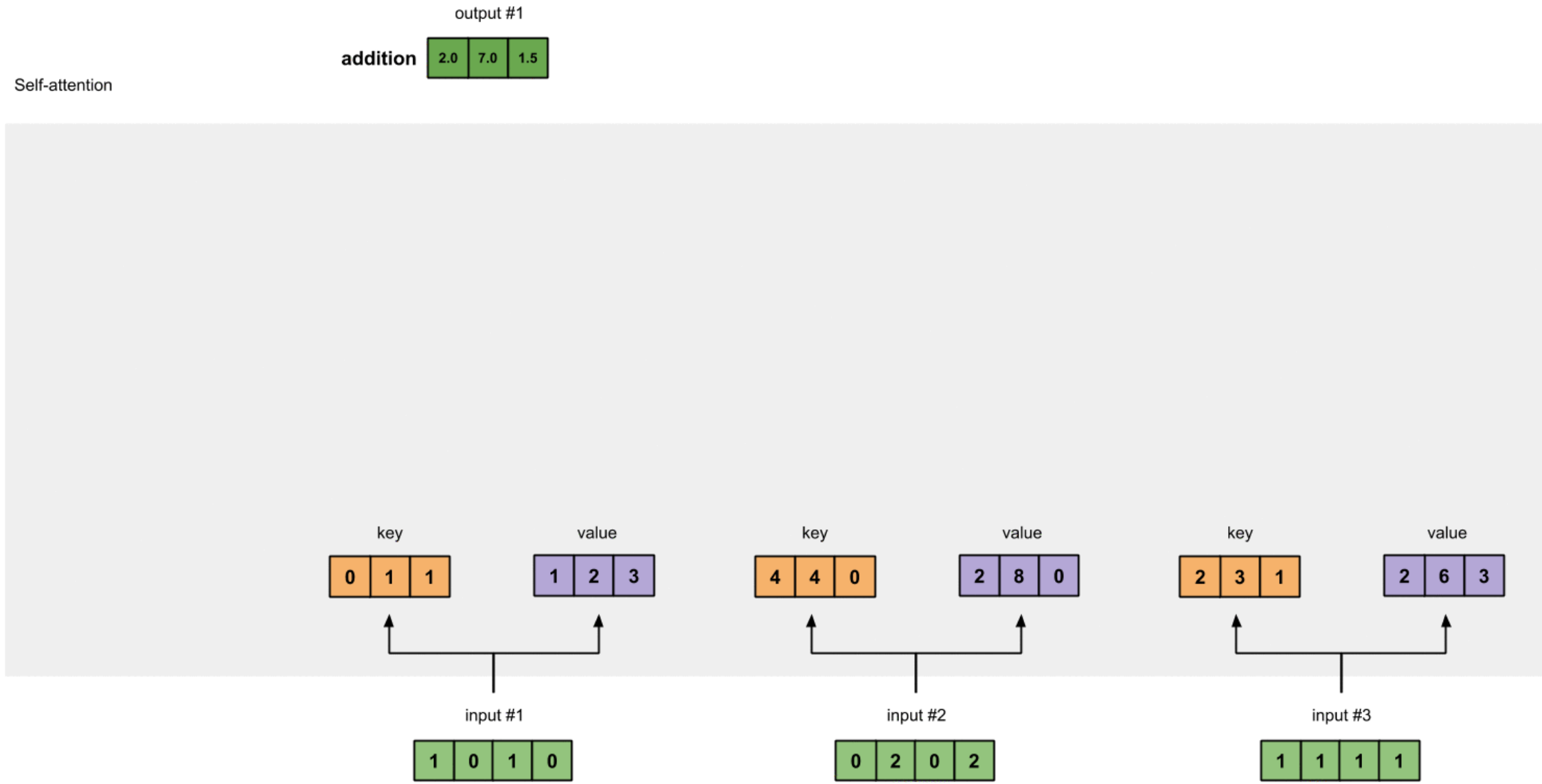


Self-attention



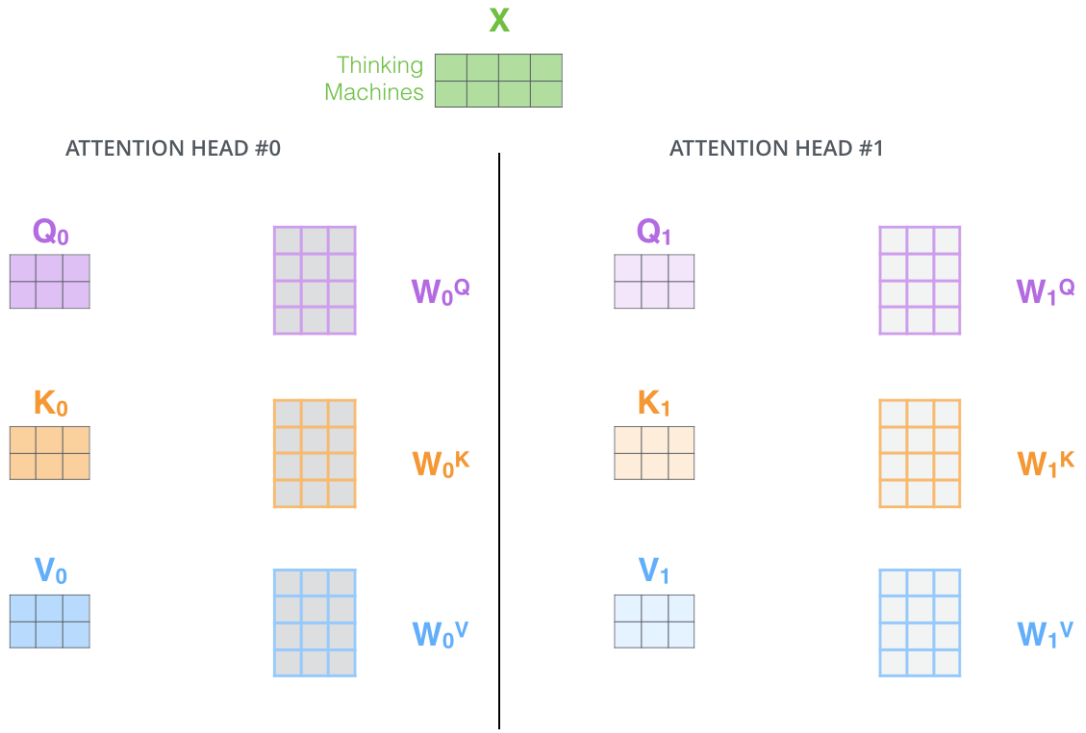
Self-attention




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

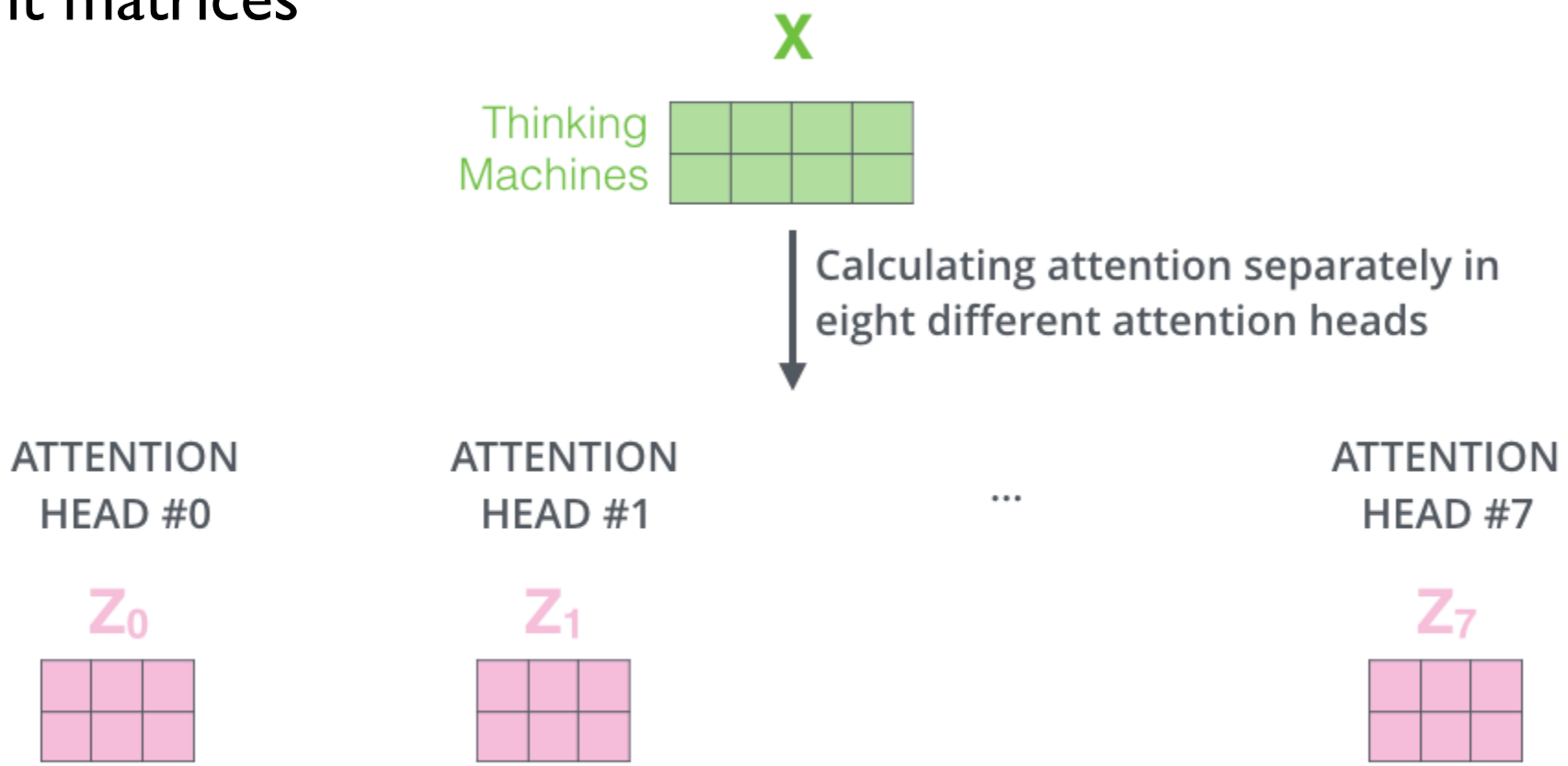
Multi-head attention

- 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

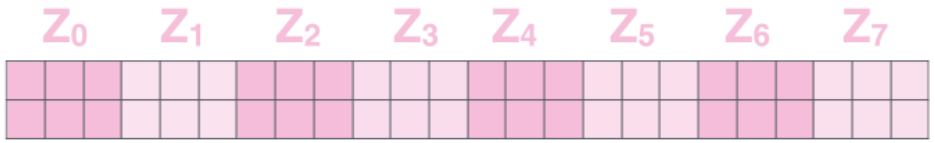


Multi-head attention

The feed-forward layer is not expecting eight matrices.

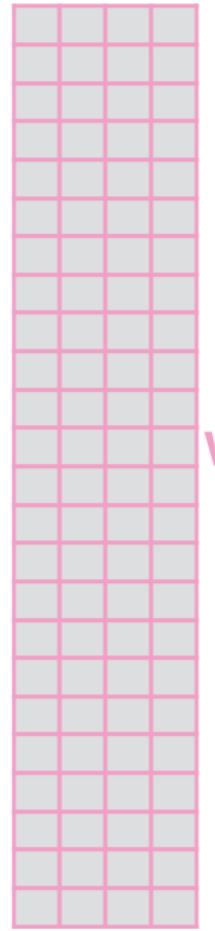
What to do?

1) Concatenate all the attention heads



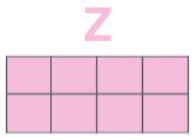
2) Multiply with a weight matrix W^O that was trained jointly with the model

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

=



Multi-head attention

1) This is our input sentence*

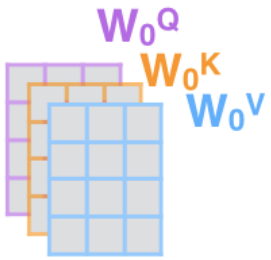
2) We embed each word*

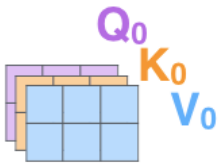
3) Split into 8 heads. We multiply X or R with weight matrices

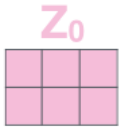
4) Calculate attention using the resulting Q/K/V matrices

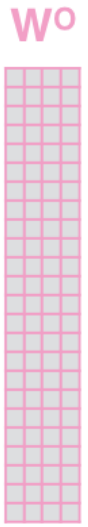
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

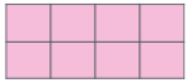
Thinking Machines



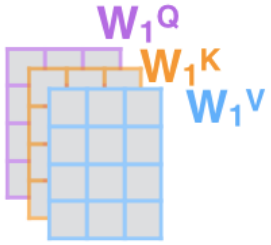


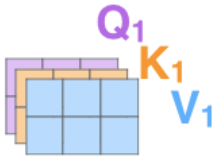


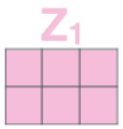


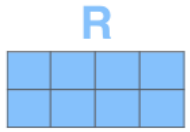


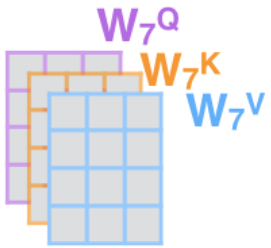
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

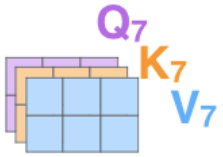


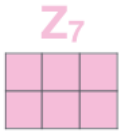












Attention in Transformer

- Encoder: In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- Encoder-decoder: the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence.
- Decoder: Similarly, self-attention layers in the decoder allow each position in the decoder to attend to all positions in the decoder up to and including that position (to preserve the auto-regressive property)

The Annotated Transformer

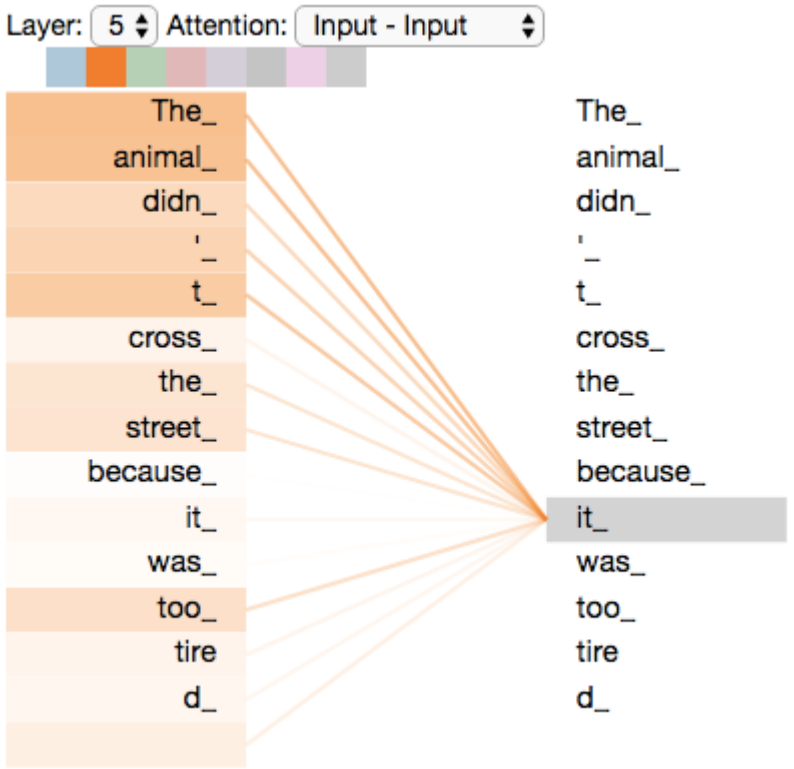
Apr 3, 2018

```
from IPython.display import Image
Image(filename='images/aiayn.png')
```

https://nlp.seas.harvard.edu/2018/04/03/attention.html

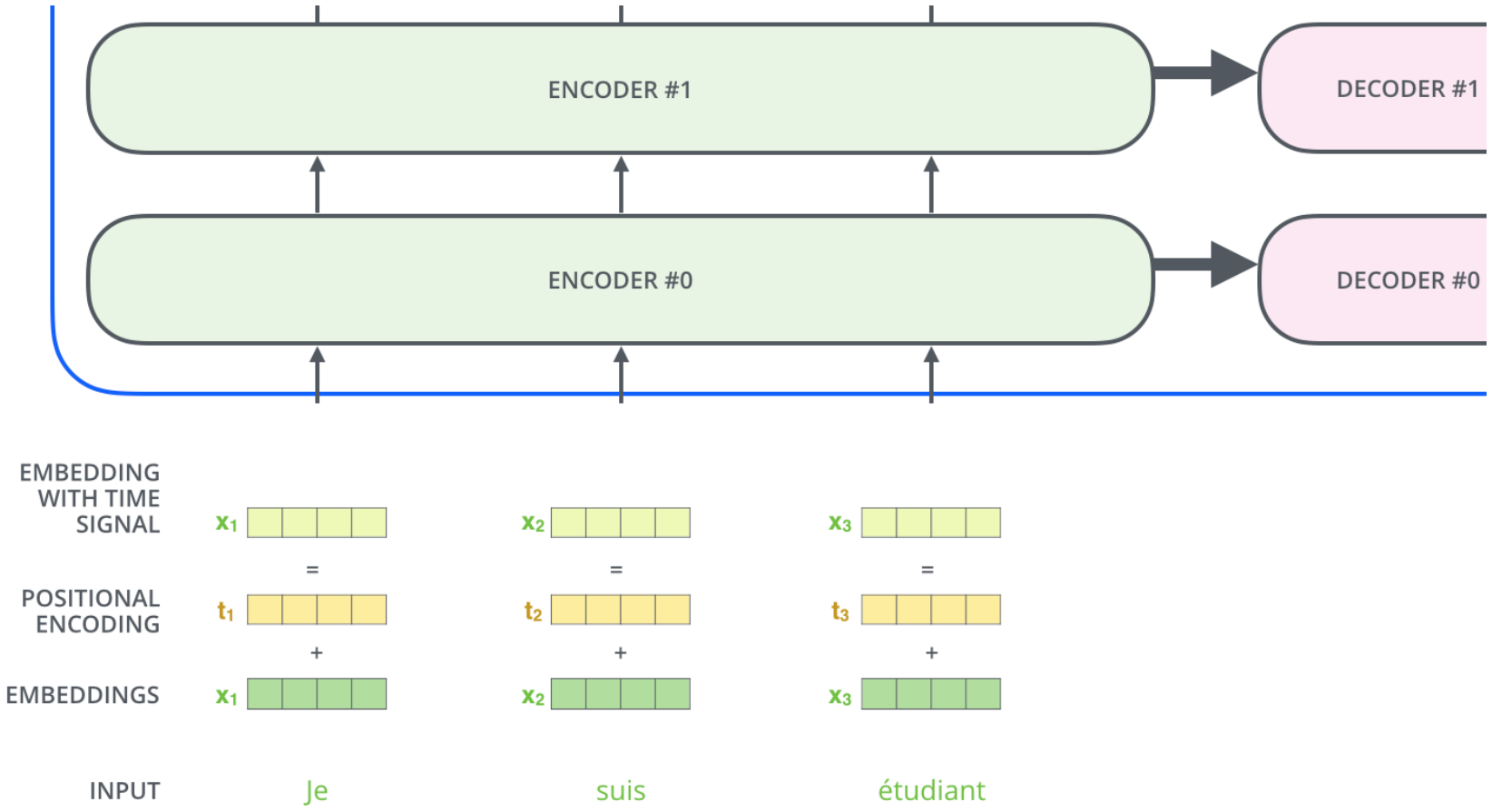
Self attention

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



What about word order?

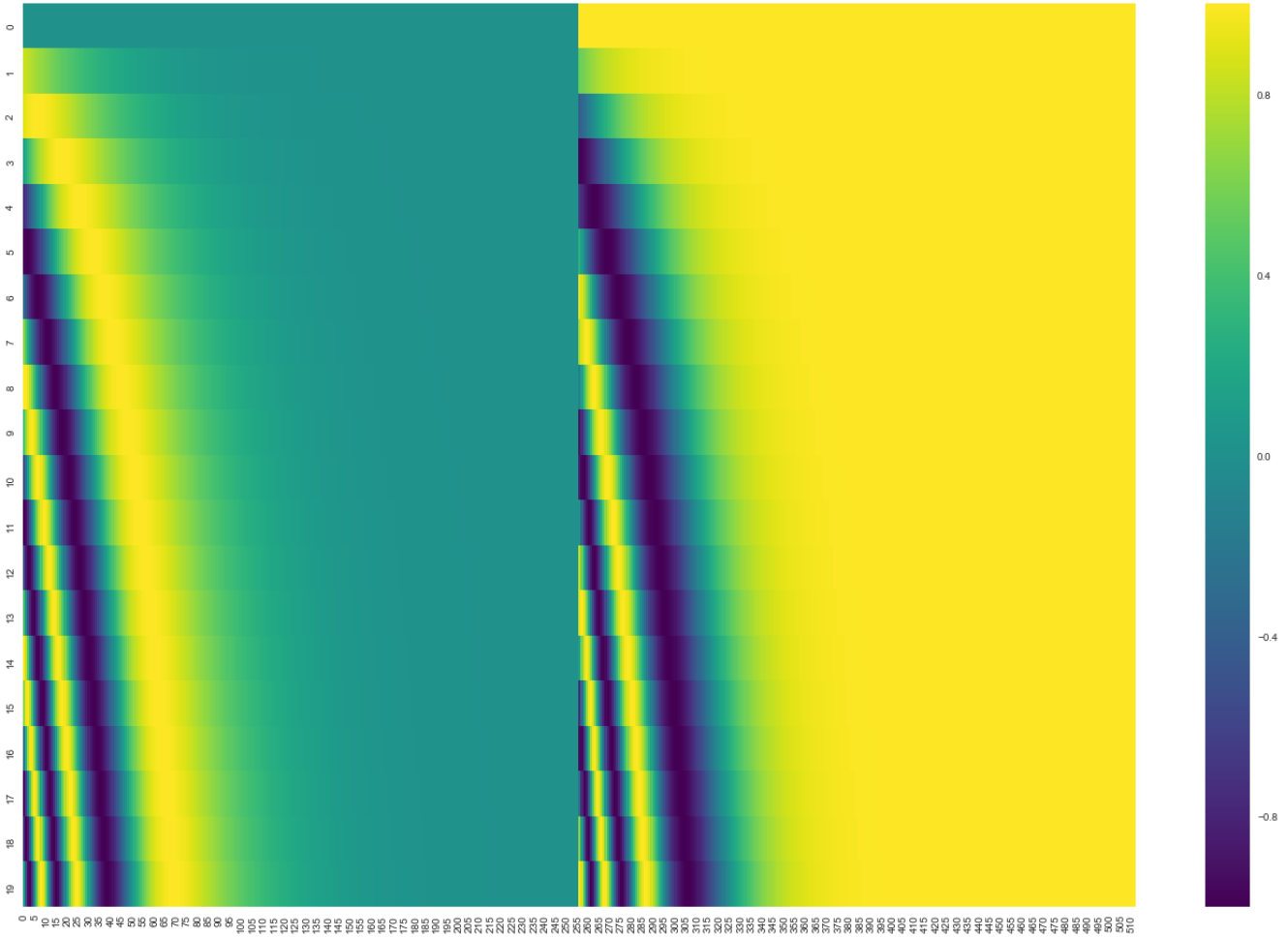
- Positional encoding
- To address this, the transformer adds a vector to each input embedding



Positional encoding

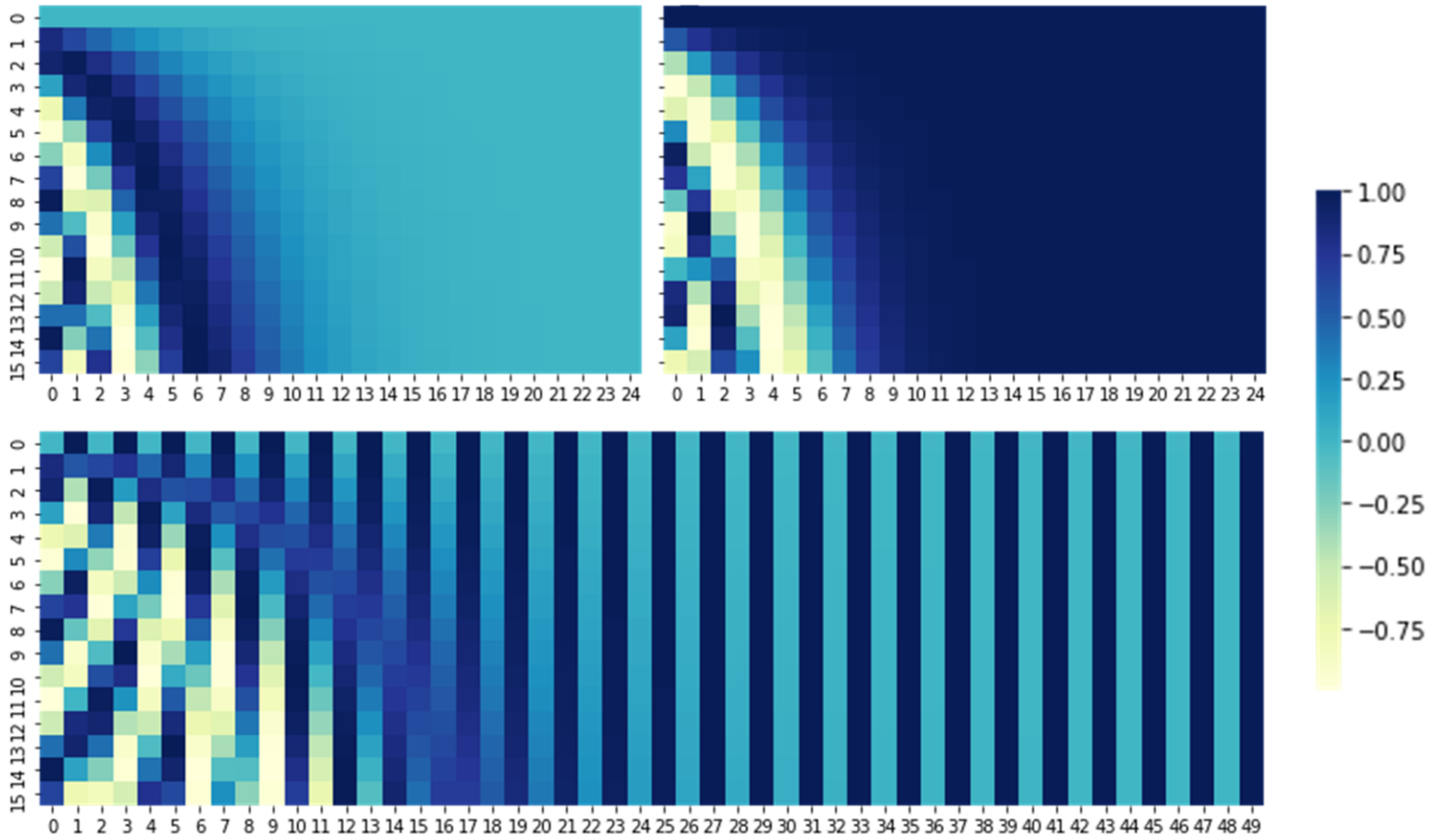
Each row corresponds to the positional encoding of a vector.

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)

Positional encoding



Positional encoding

Let t be the desired position in an input sentence, $\overrightarrow{p_t} \in \mathbb{R}^d$ be its corresponding encoding, and d be the encoding dimension (where $d \equiv_2 0$) Then $f : \mathbb{N} \to \mathbb{R}^d$ will be the function that produces the output vector $\overrightarrow{p_t}$ and it is defined as follows:

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases}$$

where

$$\omega_k = \frac{1}{10000^{2k/d}}$$

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Positional encoding not needed?

Transformer Language Models without Positional Encodings Still Learn Positional Information, 2022

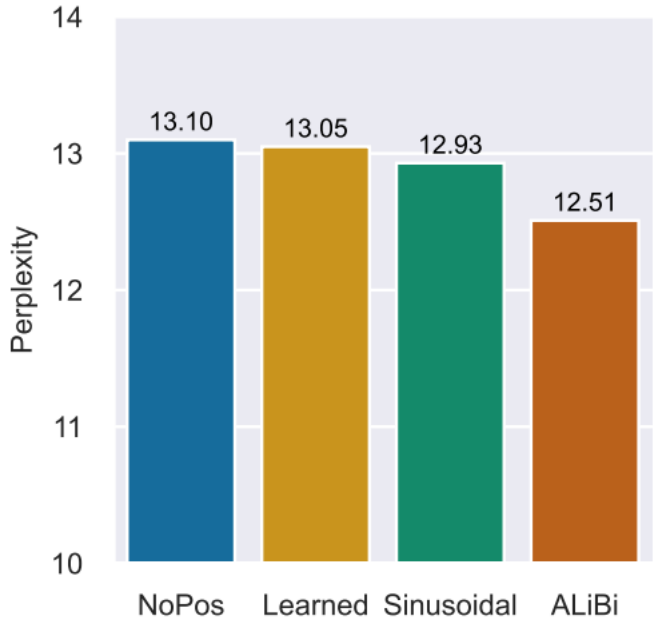
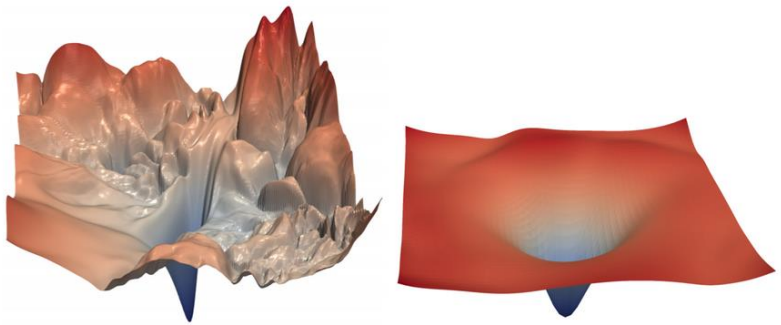


Figure 1: Transformer language models trained without explicitly encoding positional information (*NoPos*) approach the performance of models trained with various positional encoding methods. All models have 1.3B parameters, and are trained on an excerpt of the Pile.

The residuals

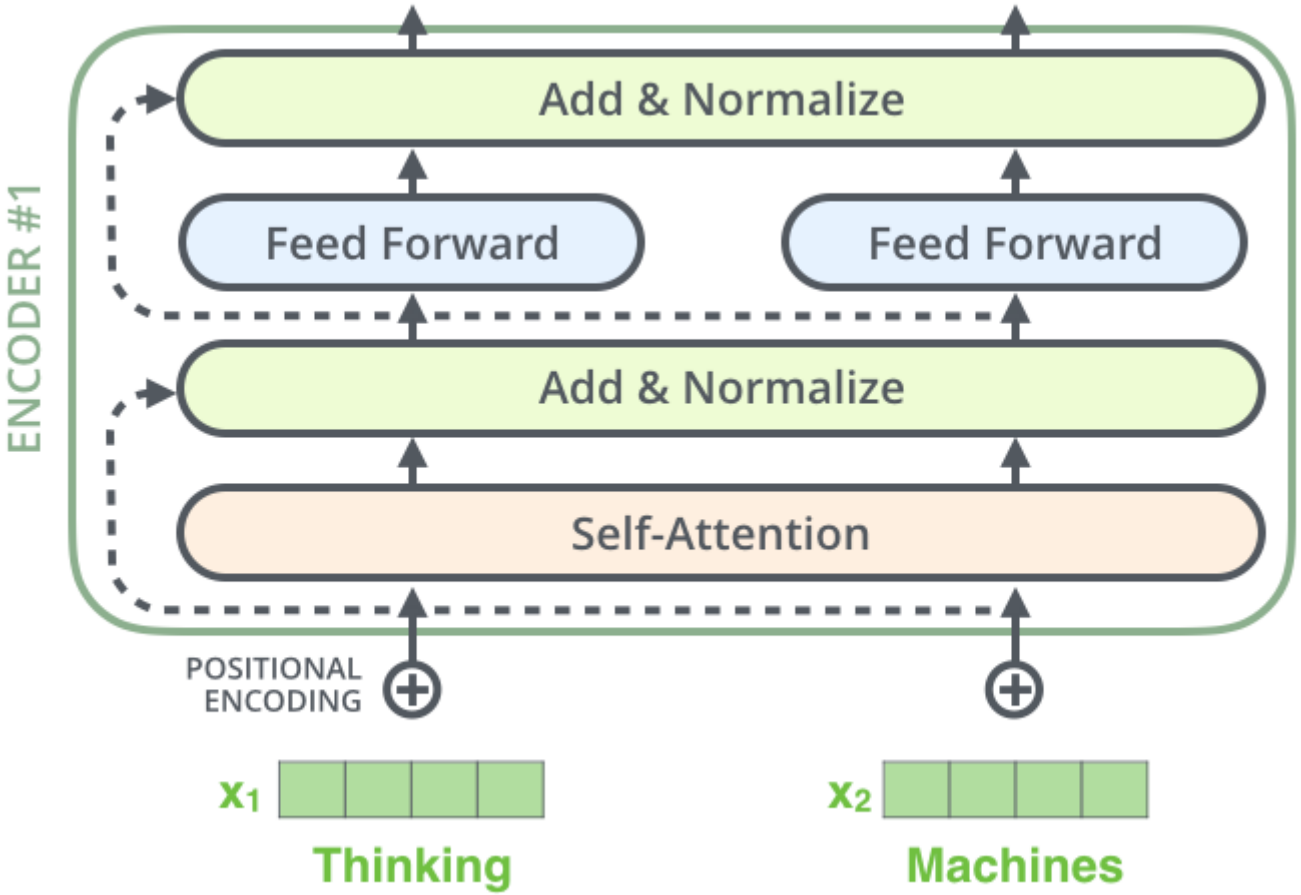
Visualizing the Loss Landscape of Neural Nets

Hao Li[1], Zheng Xu[1], Gavin Taylor[2], Christoph Studer[3], Tom Goldstein[1]

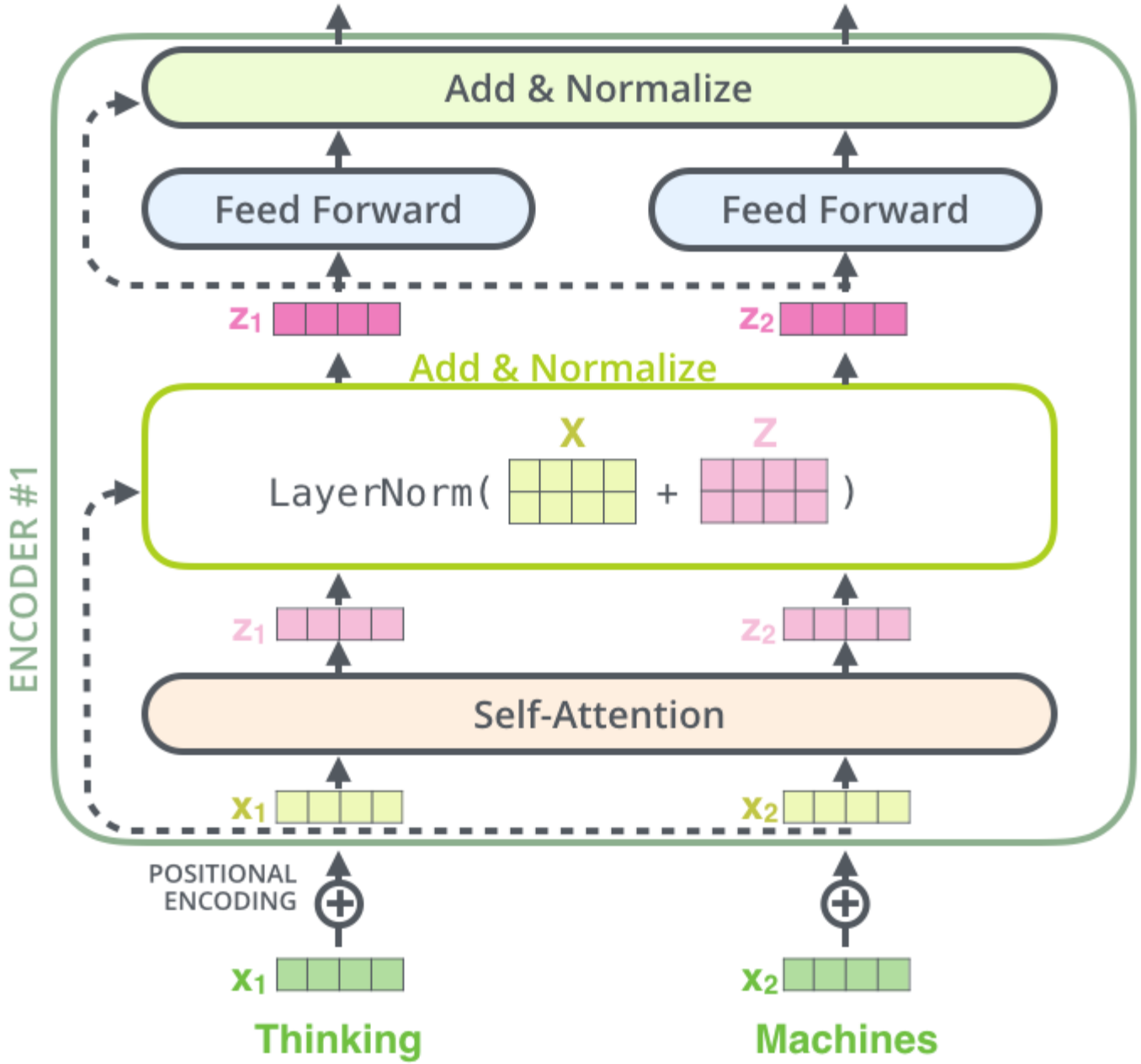


(a) without skip connections (b) with skip connections

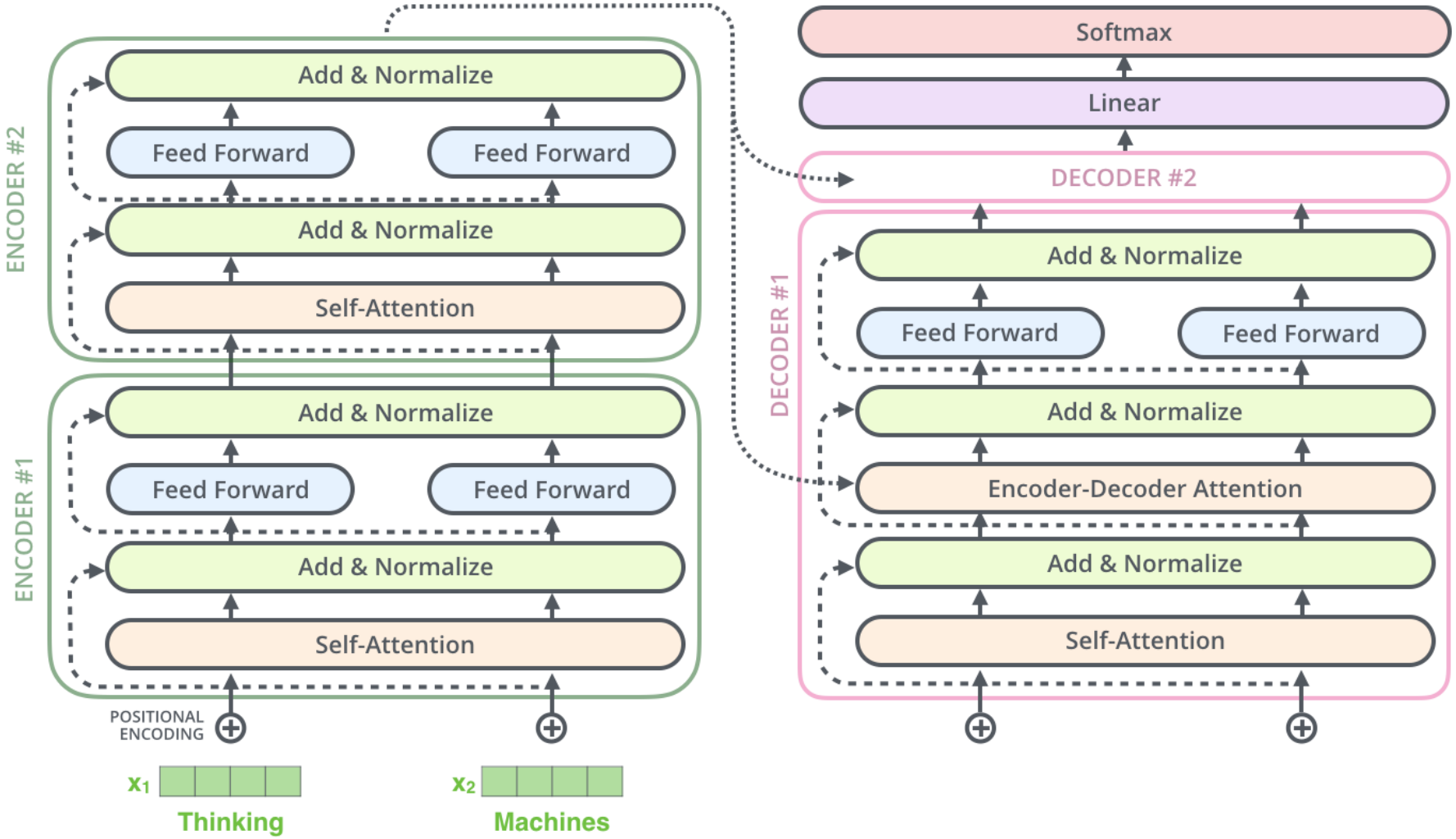
Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.



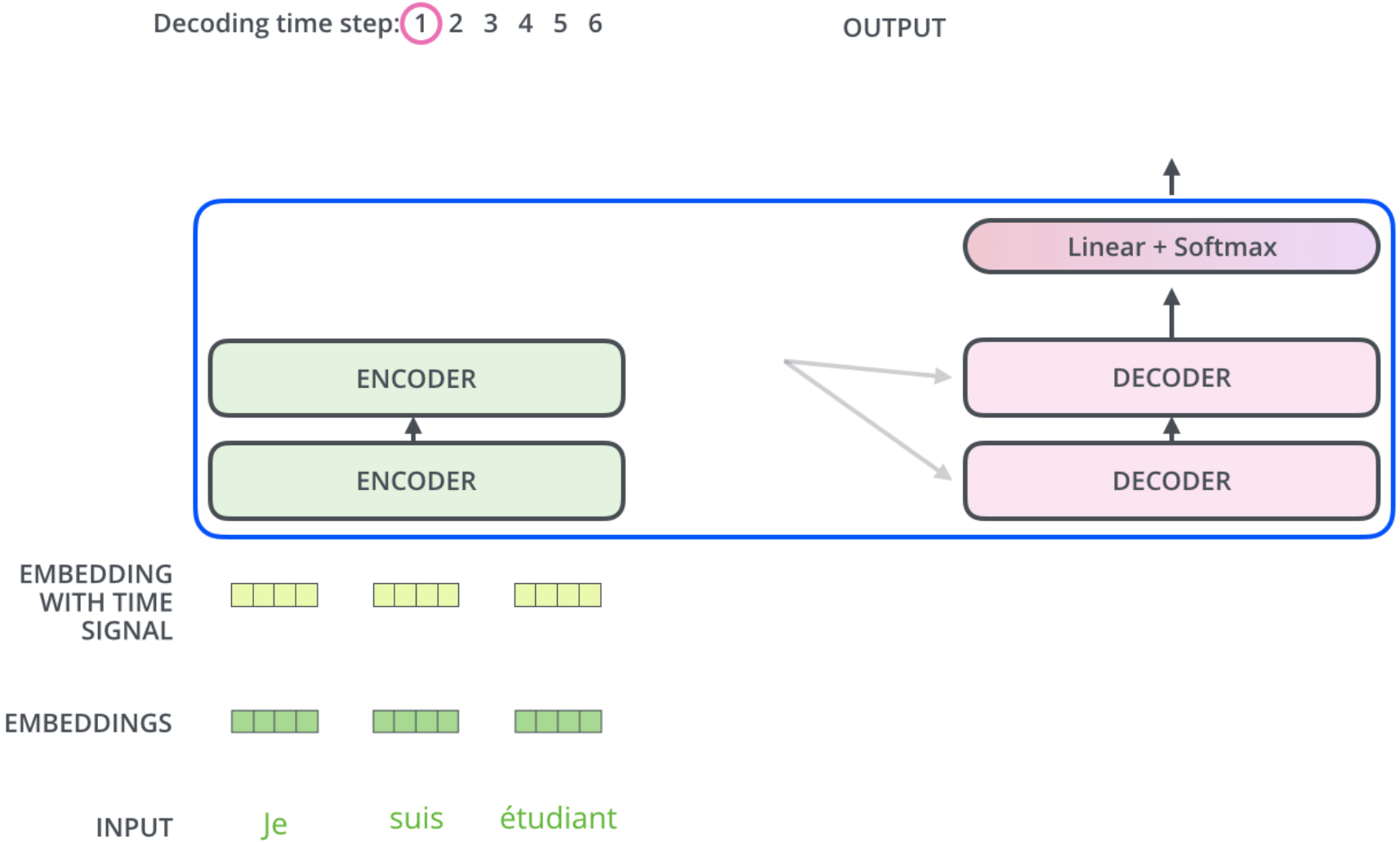
The encoder


Add & Normalize
Feed Forward
Feed Forward
z1
z2
Add & Normalize
X
Z
LayerNorm(+)
ENCODER #1
z1
z2
Self-Attention
x1
x2
POSITIONAL ENCODING
x1
x2
Thinking
Machines

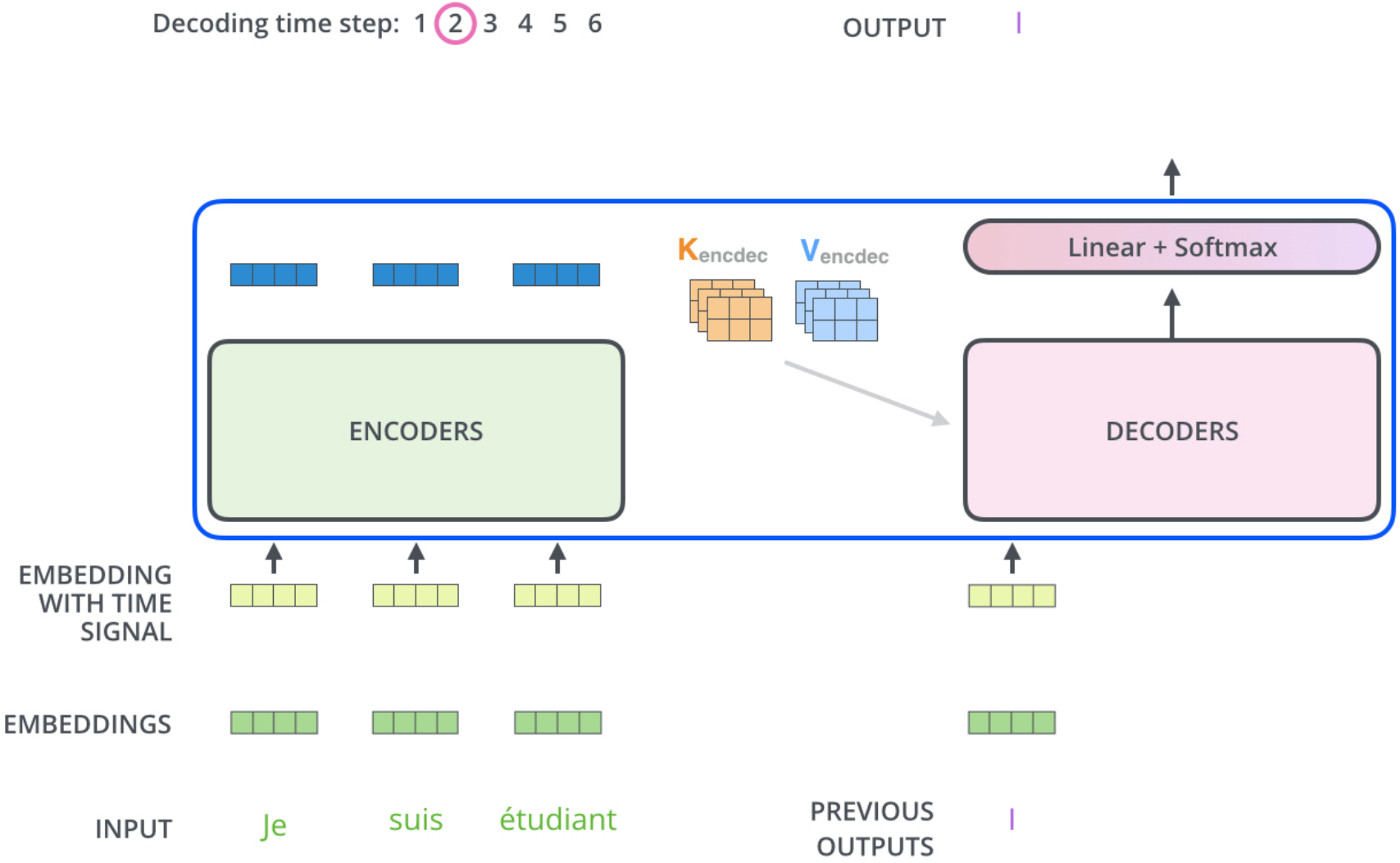
Transformer with 2 stacked encoders and decoders



Decoder side



Decoder side



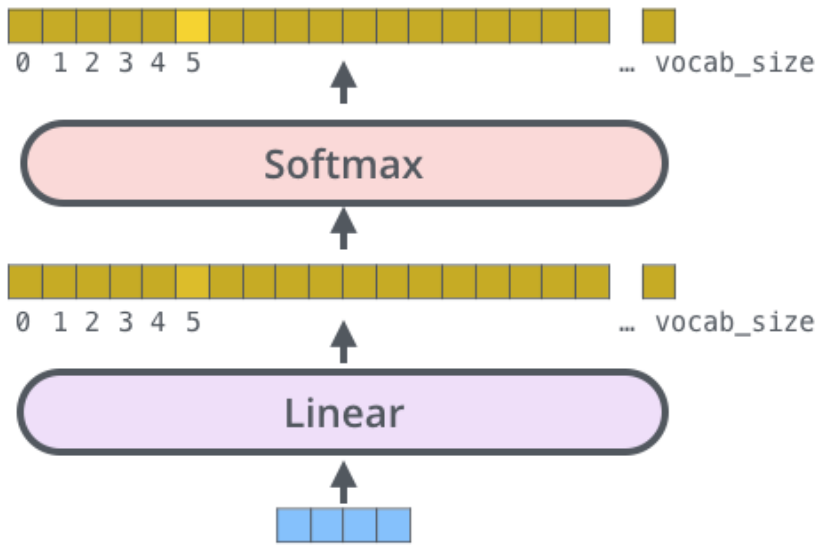
Final softmax layer

Which word in our vocabulary is associated with this index?

am

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

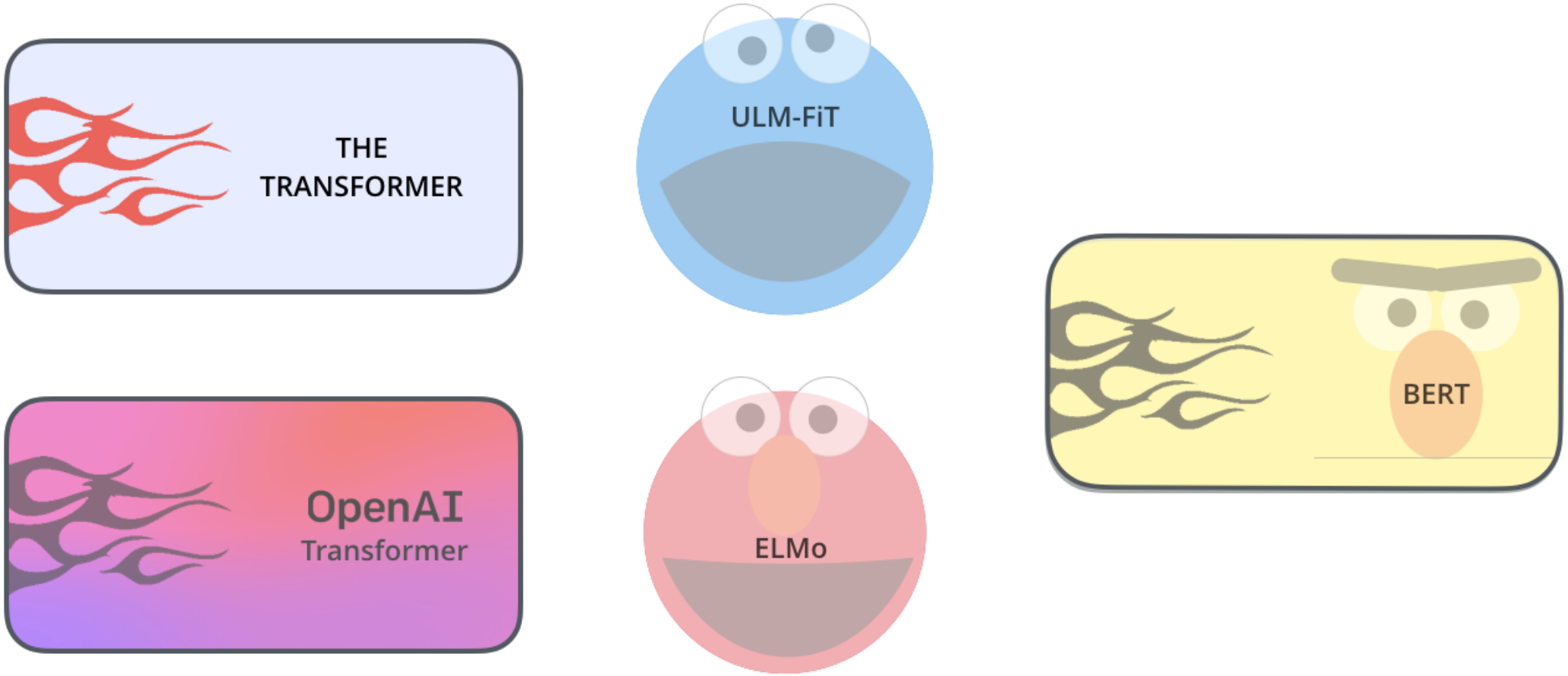
5



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

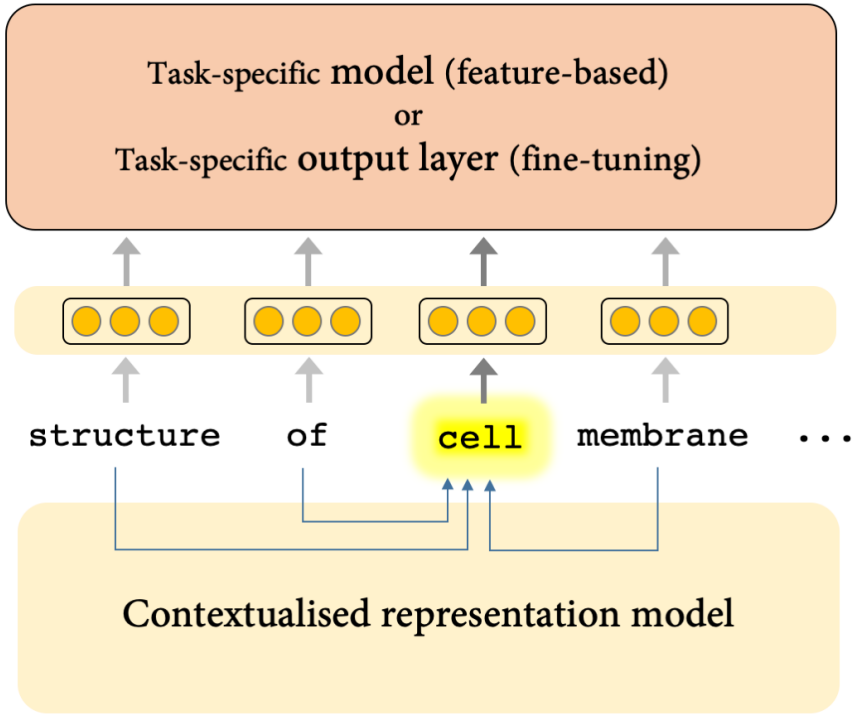
We need dynamic word embeddings!

Contextualised Embeddings

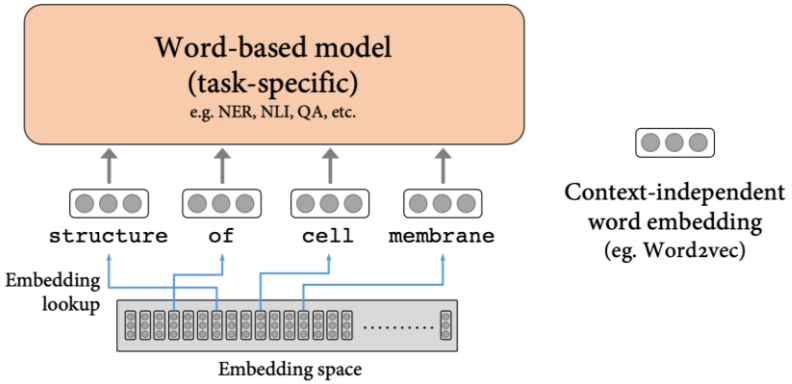
I ordered a wireless *mouse* from my laptop

Mouse has a high breeding rate

Contextualised Embeddings



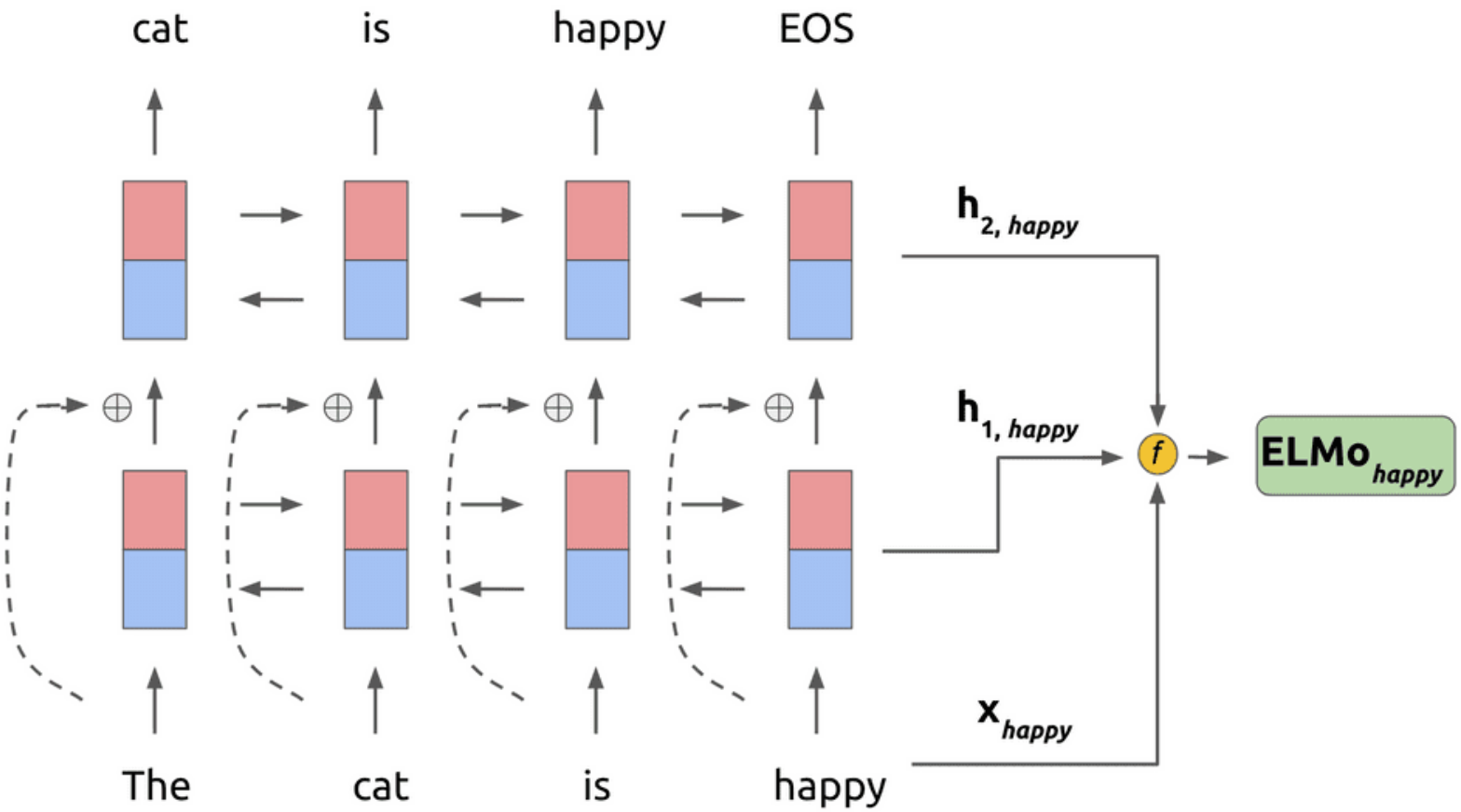
Context-dependent
word embedding
(eg. ELMo)



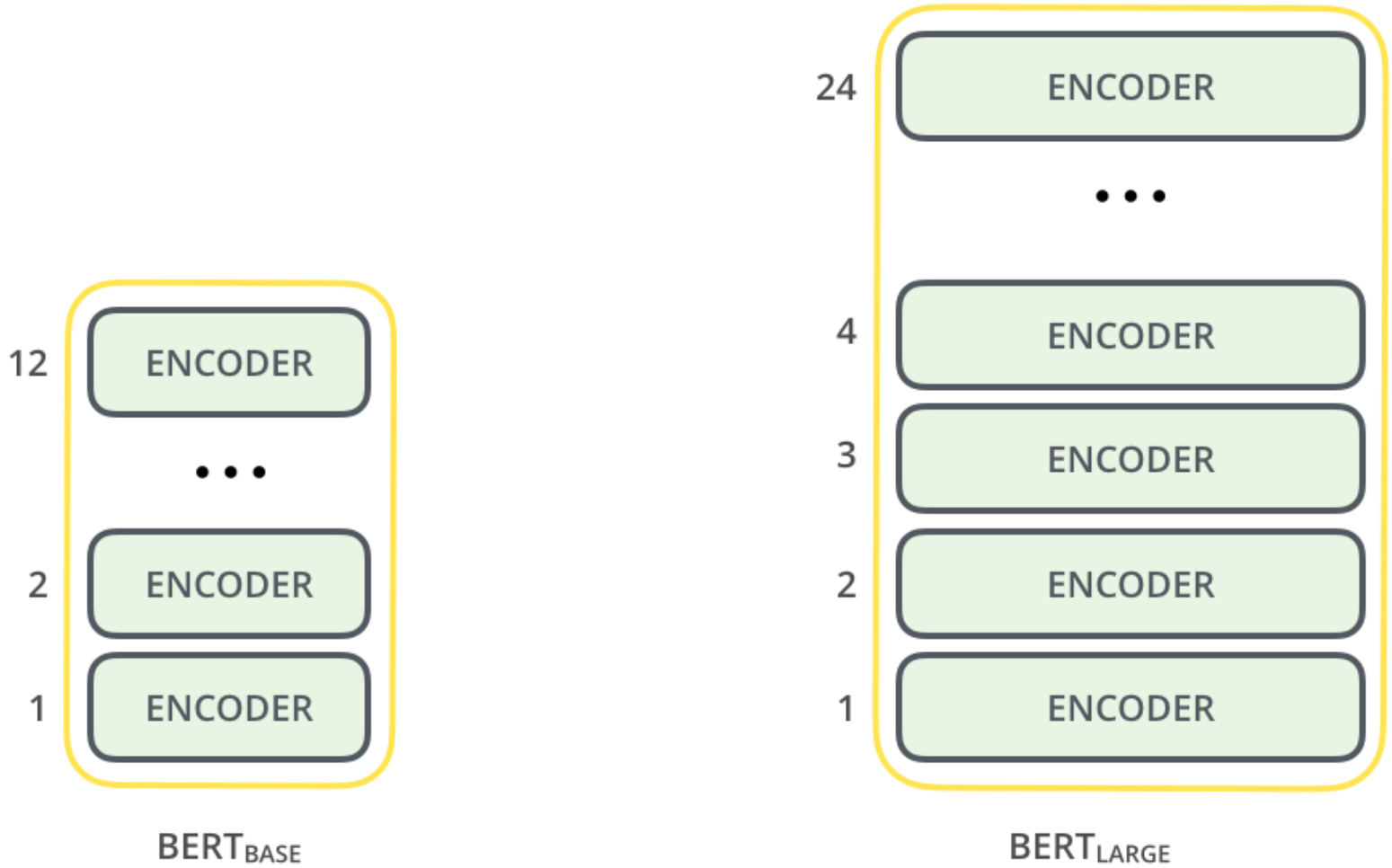
Context-independent
word embedding
(eg. Word2vec)

Contextualised Embeddings

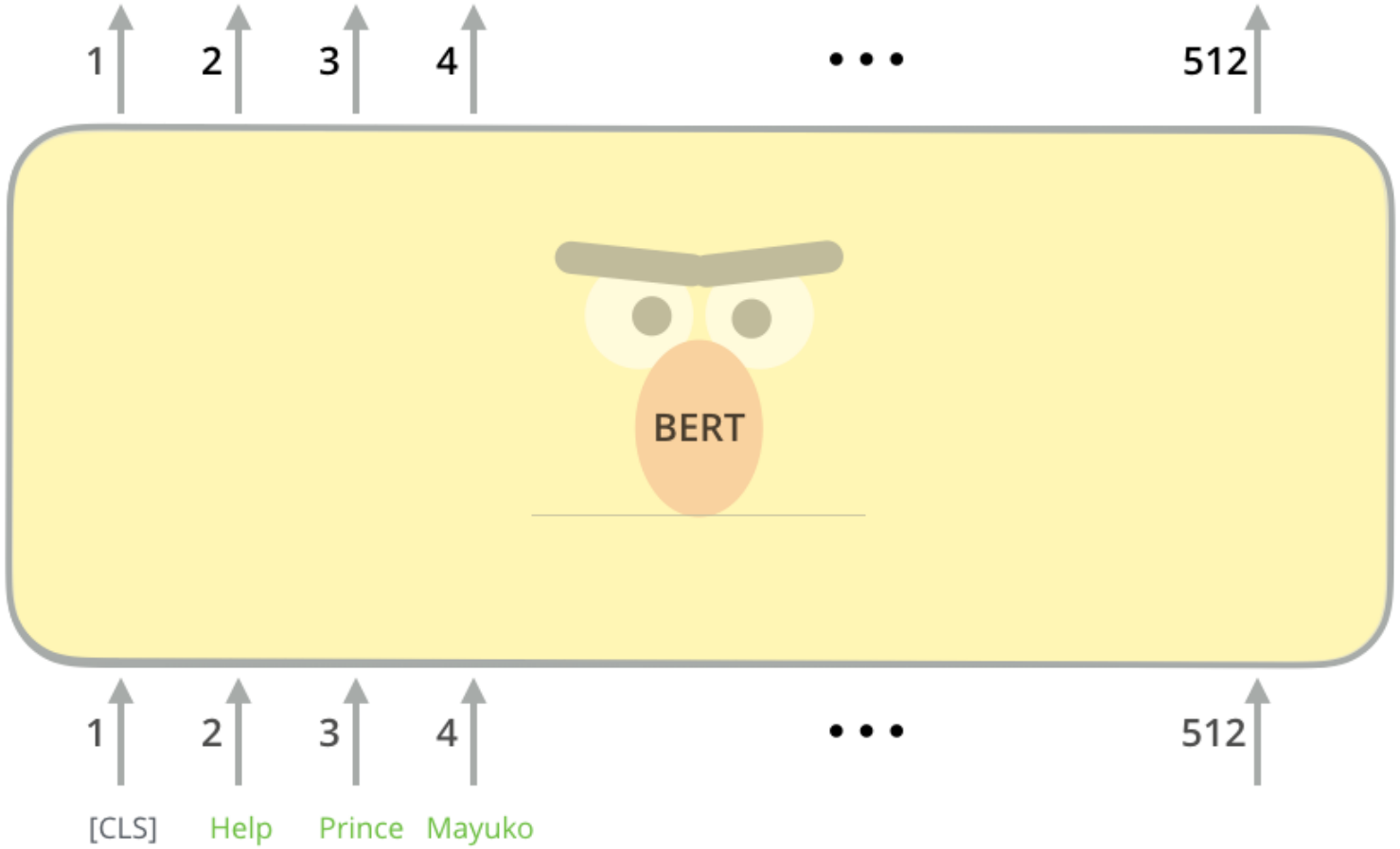
ELMo: Embeddings from Language Models



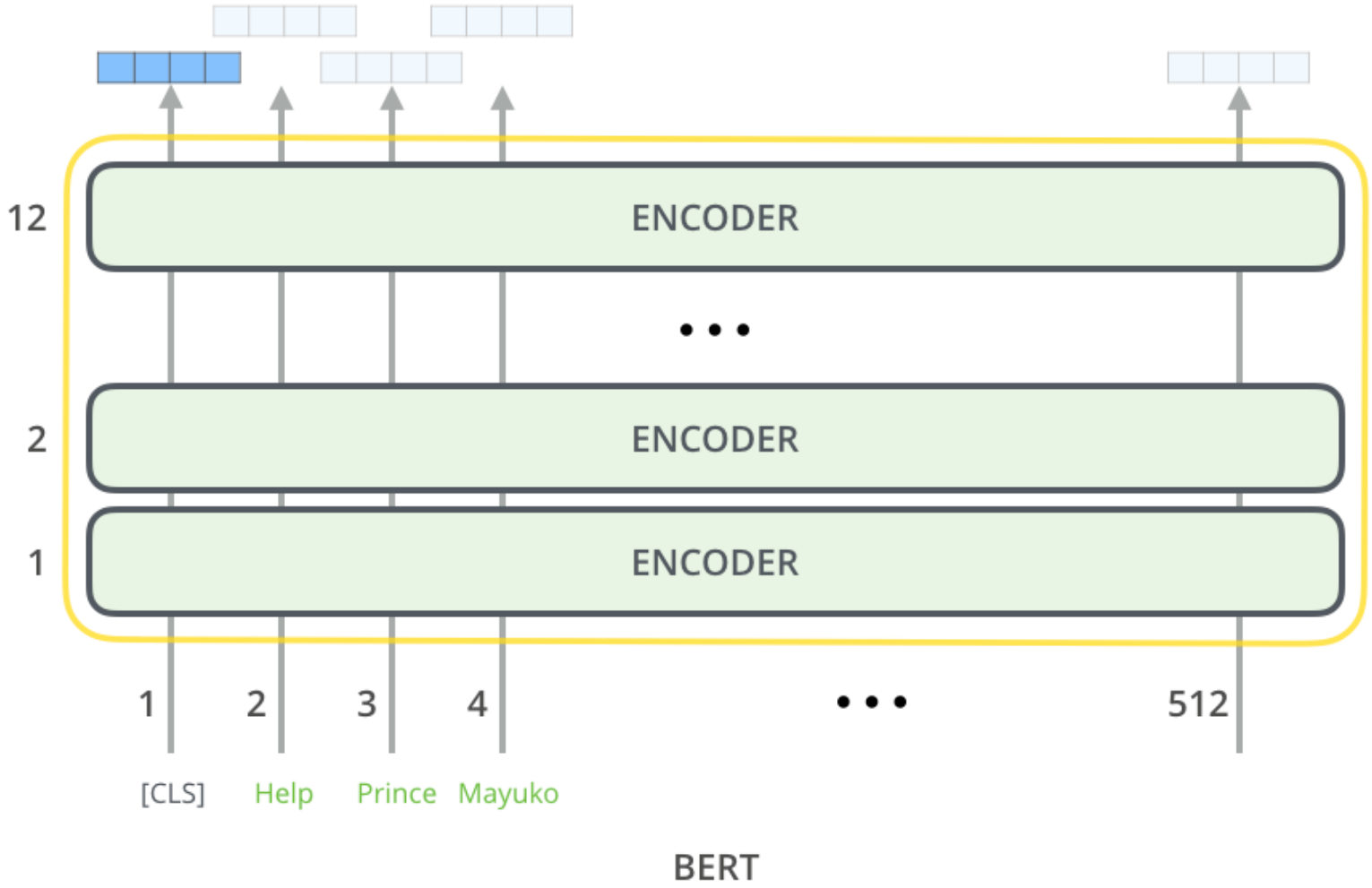
BERT


12
ENCODER
2
ENCODER
1
ENCODER
BERT BASE
24
ENCODER
4
ENCODER
3
ENCODER
2
ENCODER
1
ENCODER
BERT LARGE

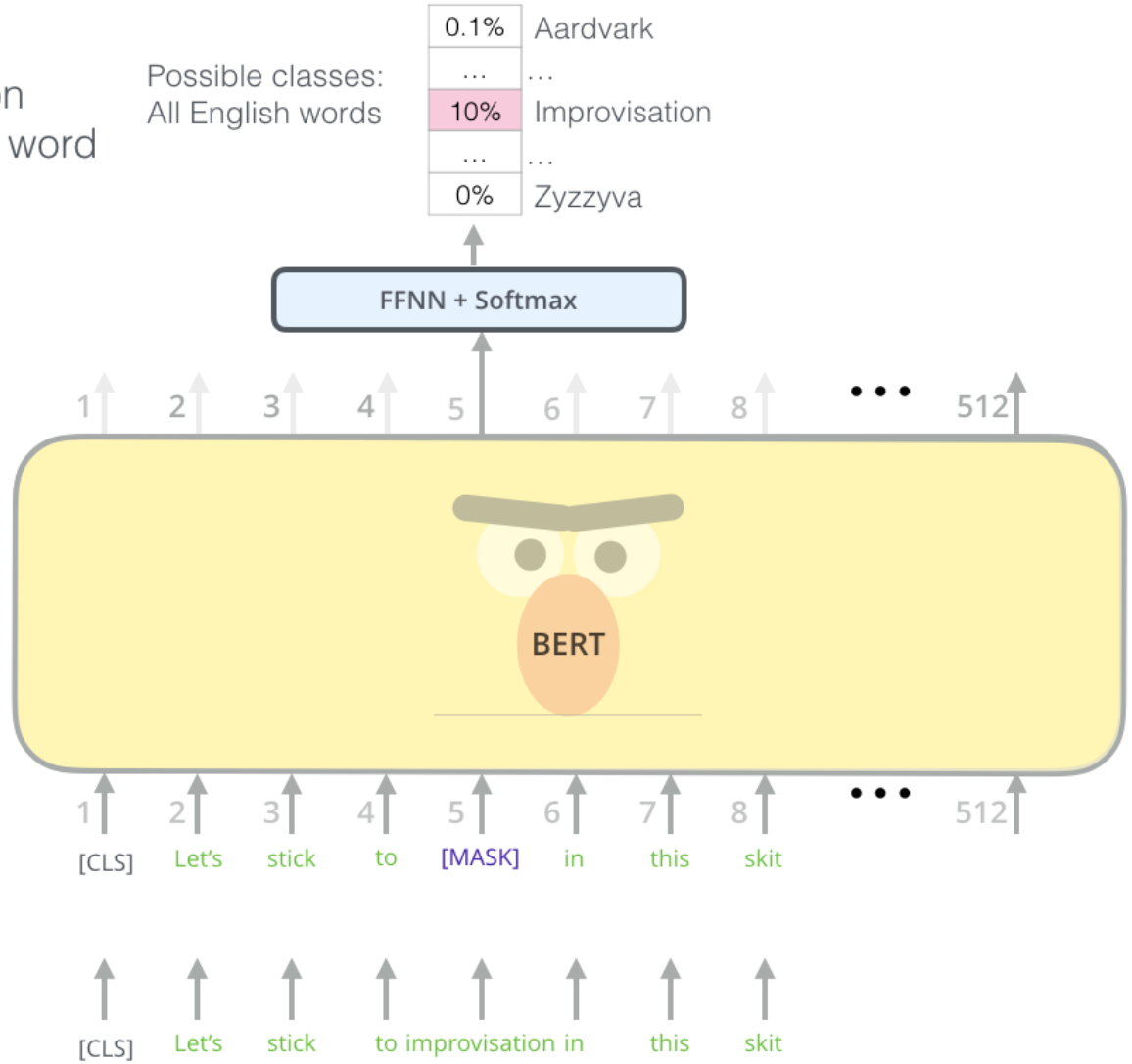
Input



Encoder (BERT-base)



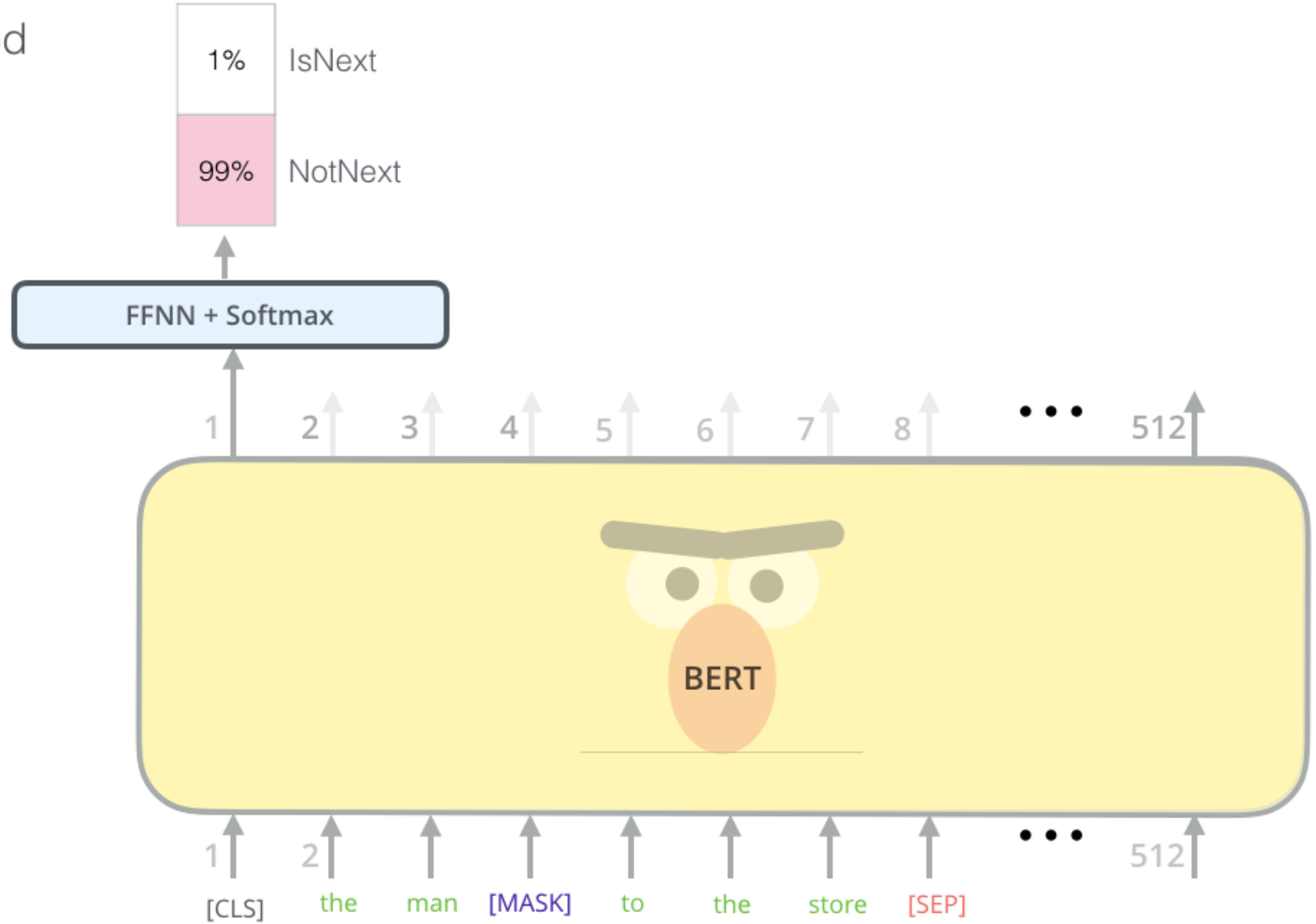
Use the output of the masked word's position to predict the masked word



Randomly mask 15% of tokens

Input

Predict likelihood that sentence B belongs after sentence A



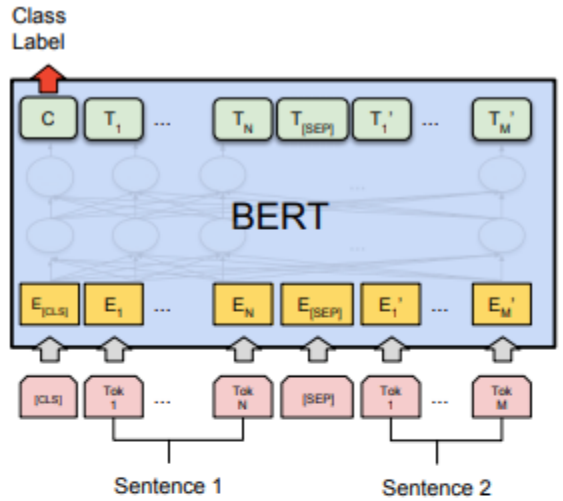
Tokenized Input

Input

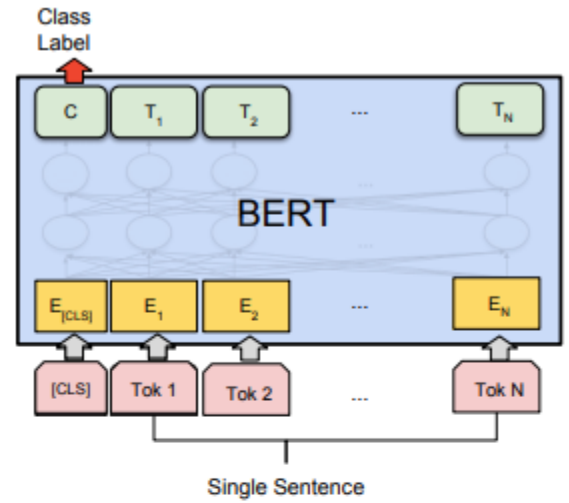
[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flightless birds [SEP]

Sentence A

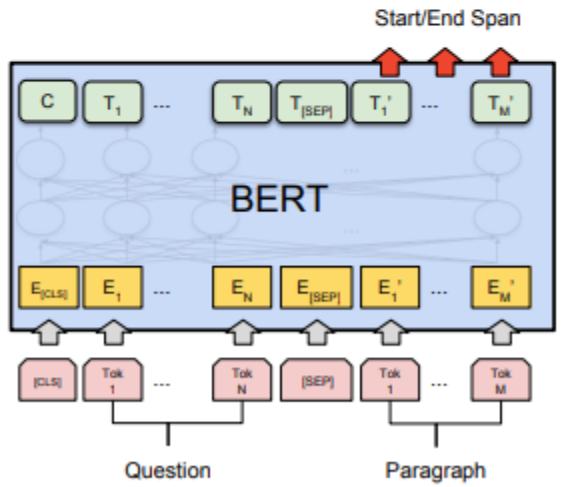
Sentence B



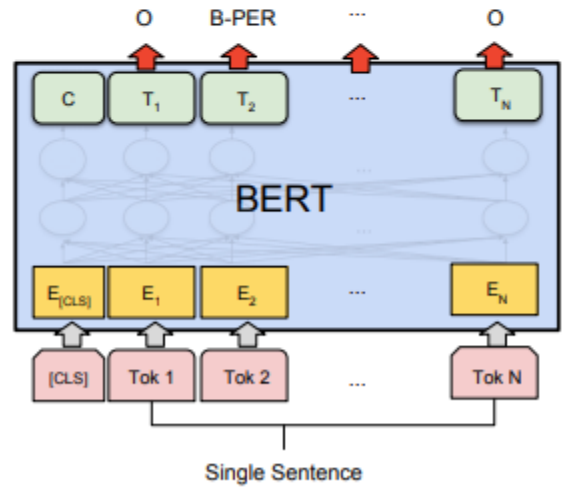
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



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

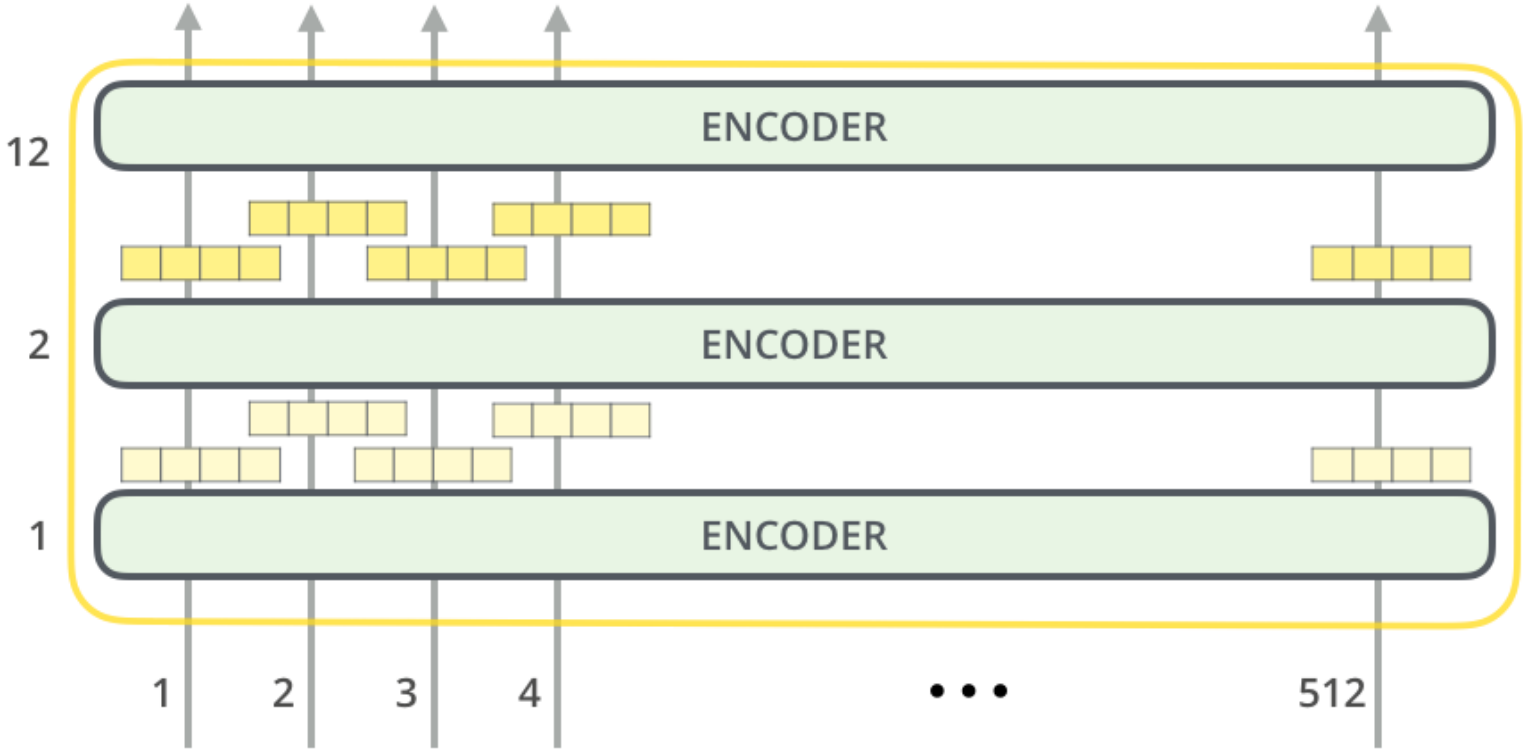


(c) Question Answering Tasks: SQuAD v1.1



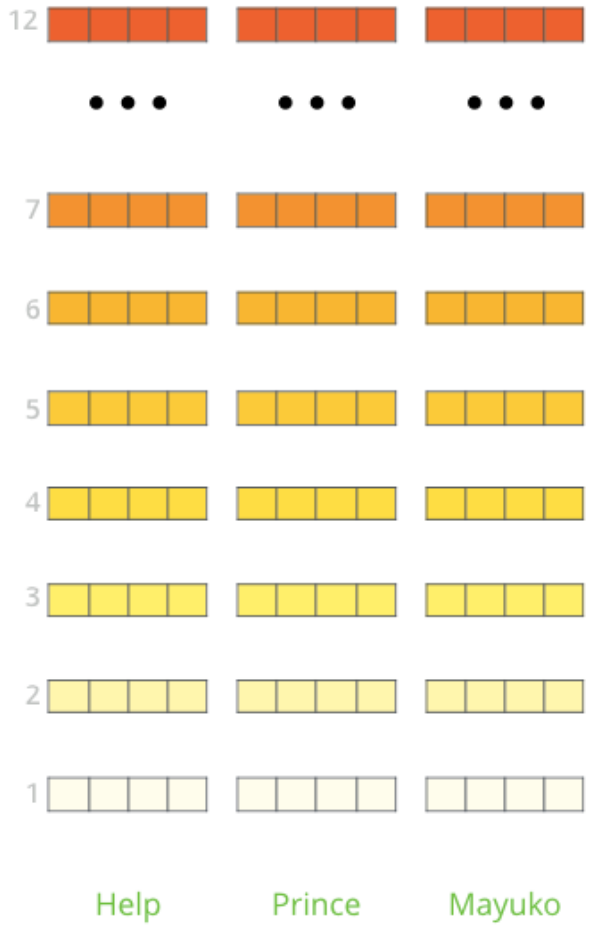
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Generate Contexualized Embeddings



BERT

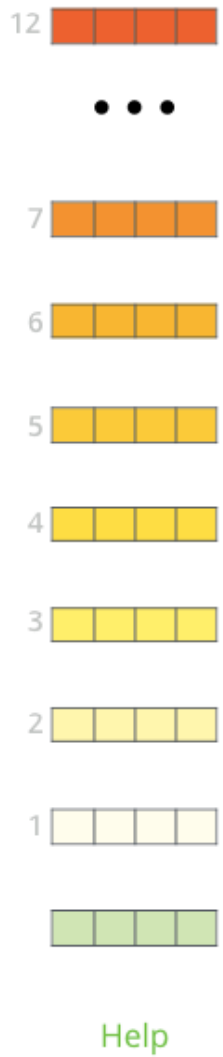
The output of each encoder layer along each token's path can be used as a feature representing that token.

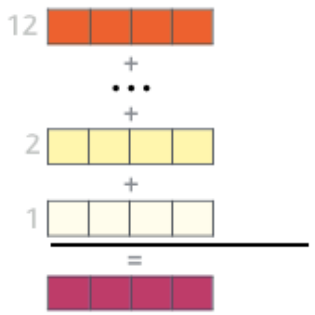
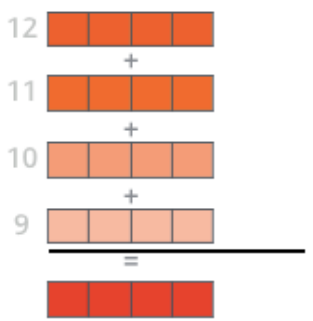
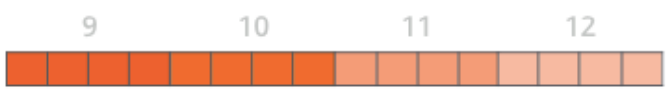


But which one should we use?

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

For named-entity recognition task CoNLL-2003 NER

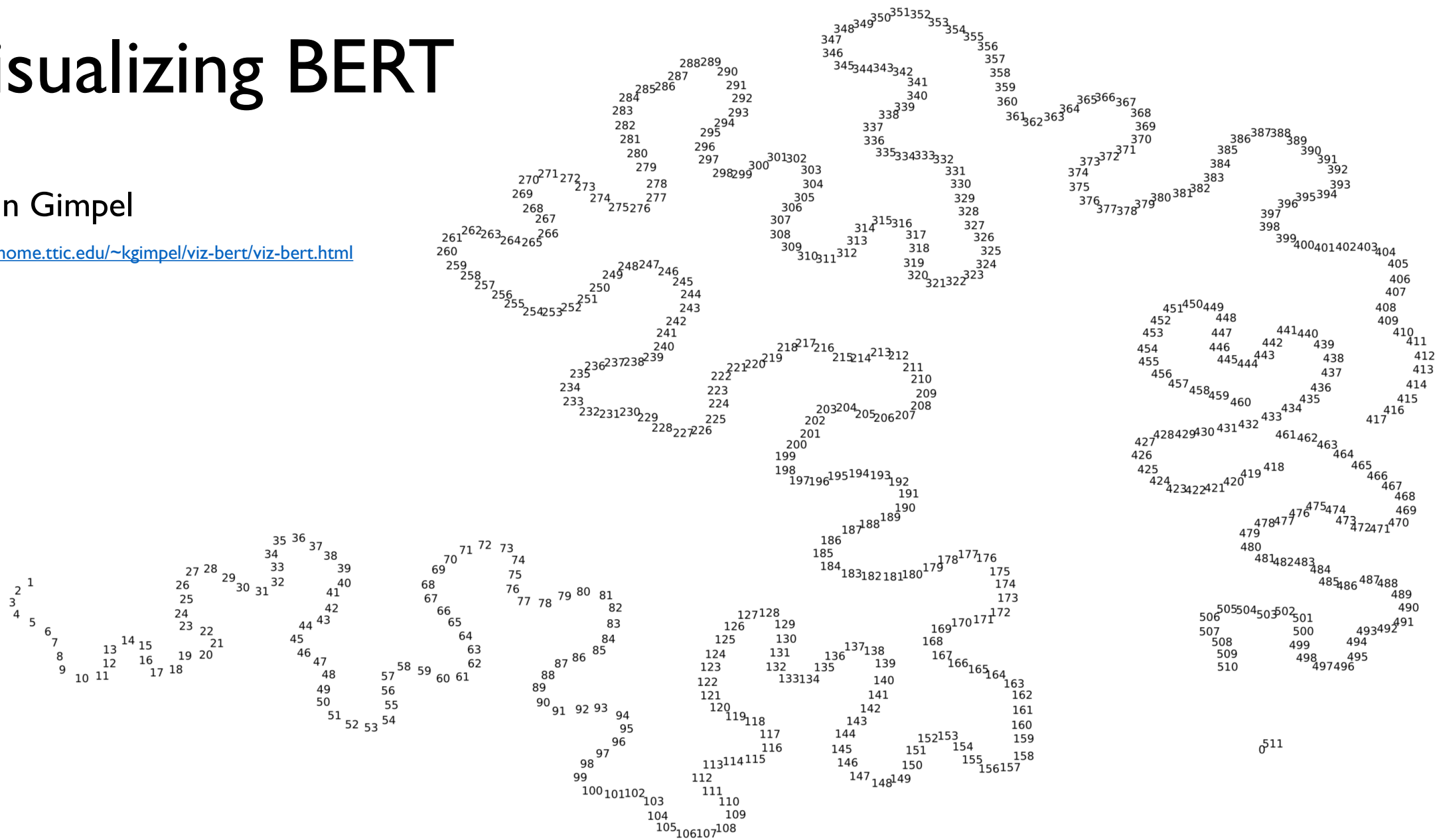


		Dev F1 Score
First Layer		91.0
Last Hidden Layer		94.9
Sum All 12 Layers		95.5
Second-to-Last Hidden Layer		95.6
Sum Last Four Hidden		95.9
Concat Last Four Hidden		96.1

Visualizing BERT

Kevin Gimpel

https://home.ttic.edu/~kgimpel/viz-bert/viz-bert.html

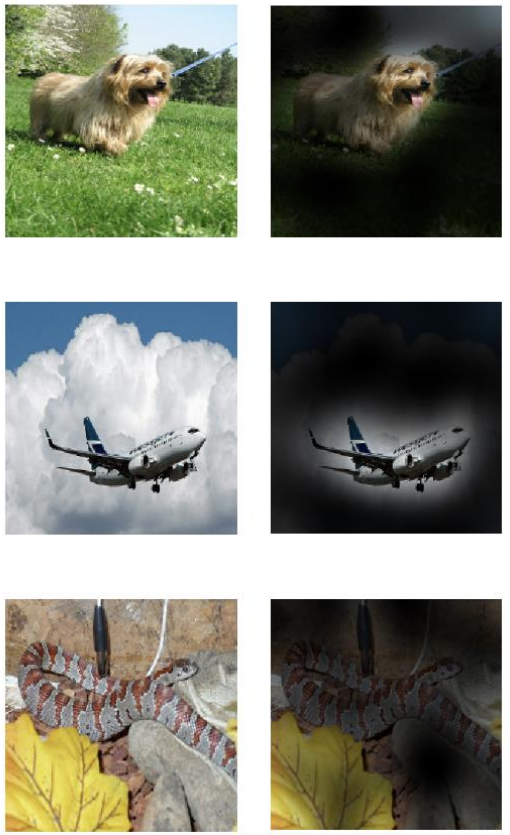


Transformers outside NLP

Vision Transformer (ViT) - Dosovitskiy et al. 2020

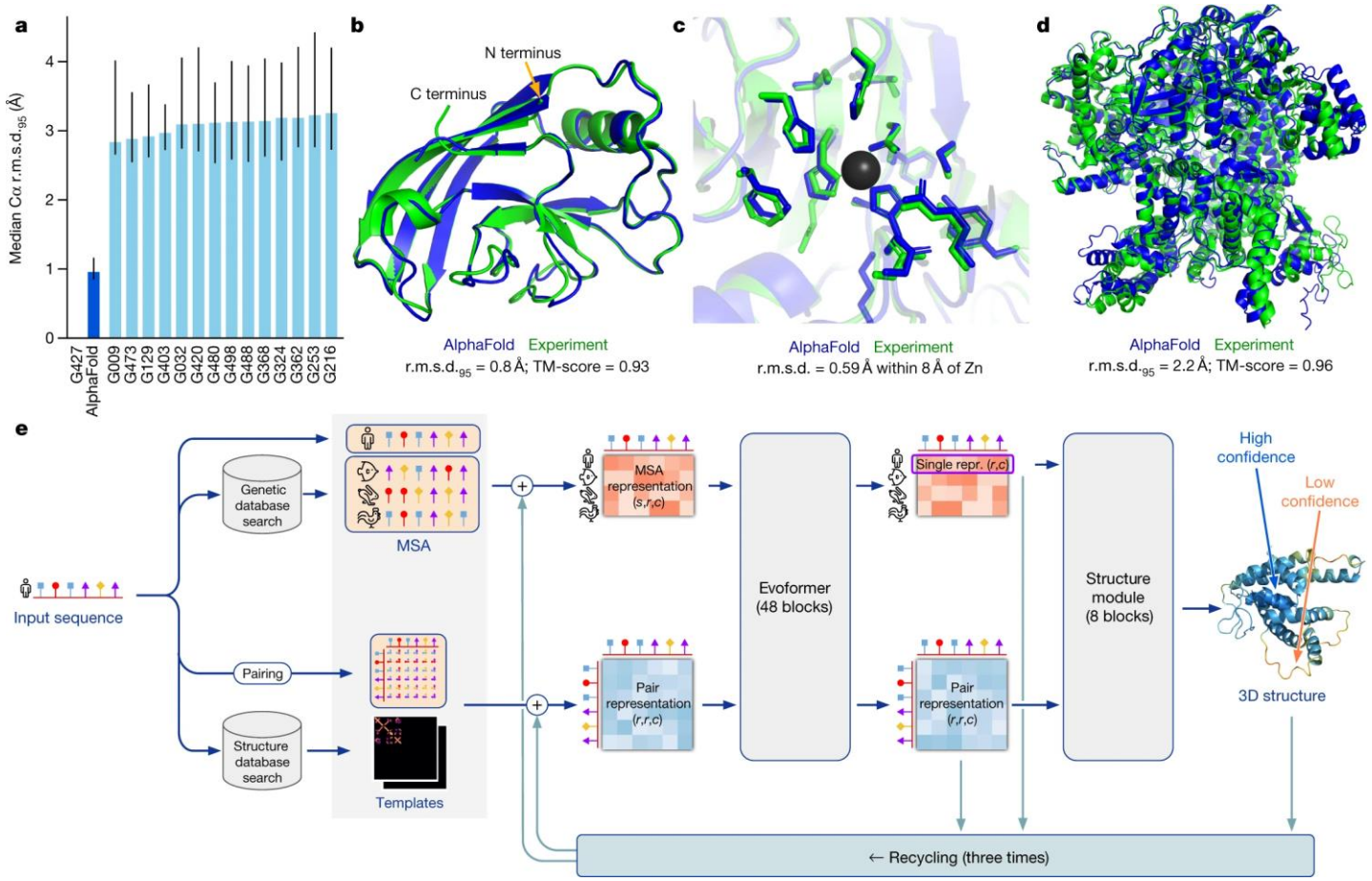
	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21K (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

Input Attention



Transformers outside NLP

AlphaFold 2



Transformers: scaling law

Kaplan et al (2020)

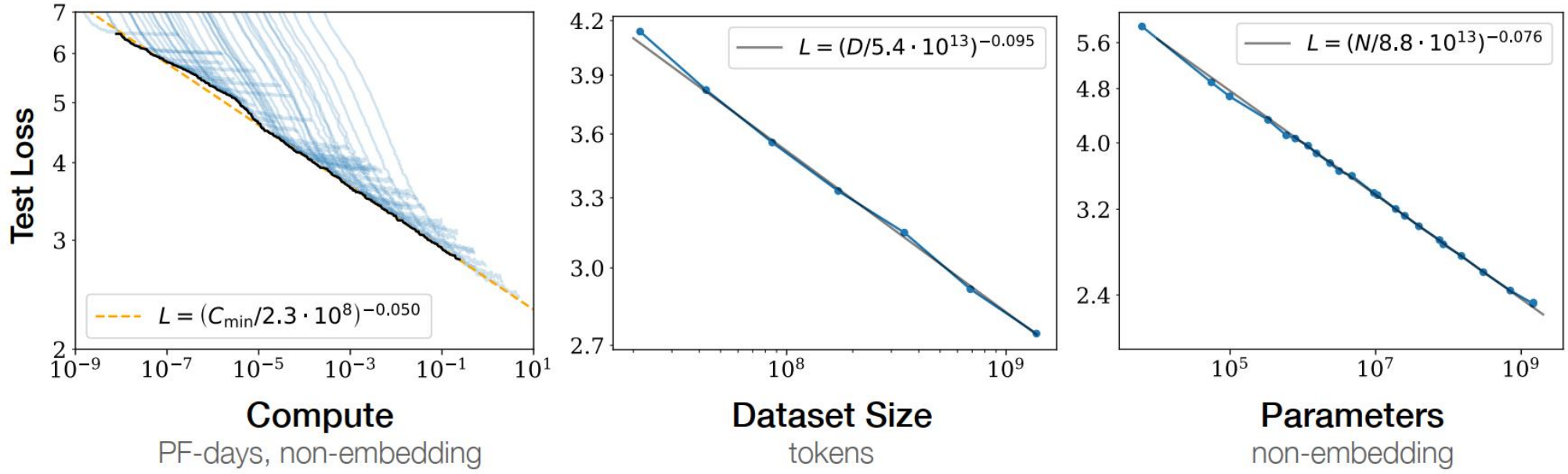


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

What to fix in Transformers?

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

- Quadratic self-attention (with sequence length)
  - Applies to all Transformers
  - For RNN it just grew linearly

Content from Keita Kurita

Improving quadratic self-attention

Linformer (Wang et al, 2020)

Key idea: map the sequence length dimension to a lower dimensional space for values, keys

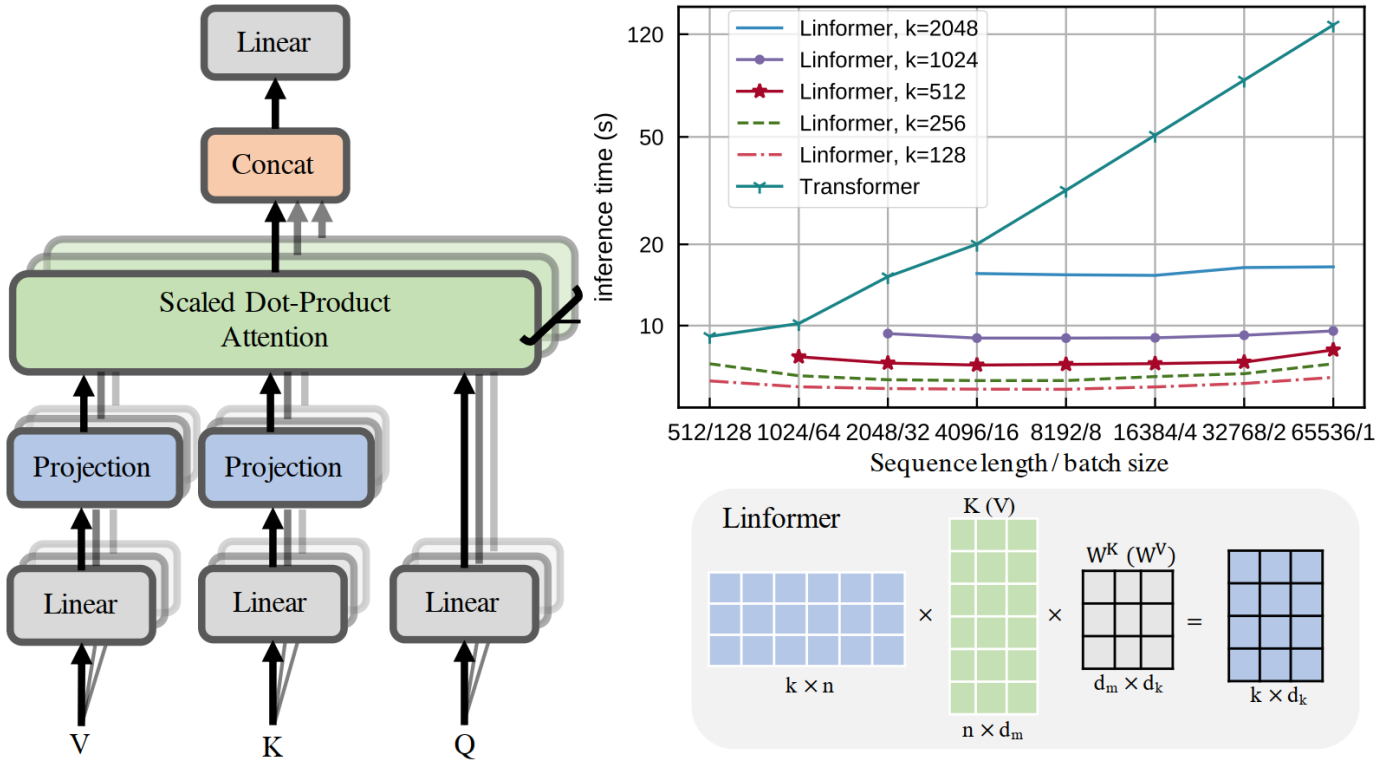
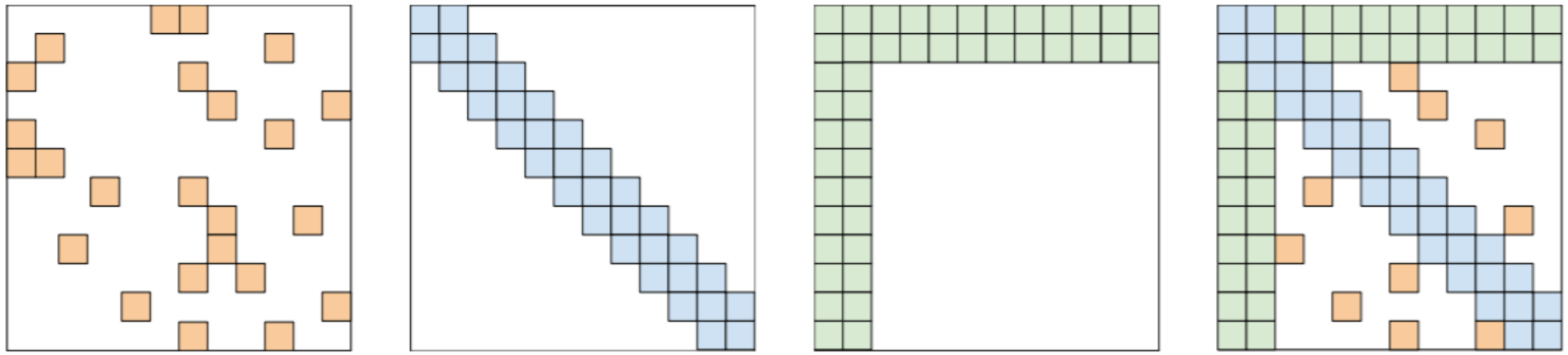


Figure 2: Left and bottom-right show architecture and example of our proposed multihead linear self-attention. Top right shows inference time vs. sequence length for various Linformer models.

Improving quadratic self-attention

BigBird (Zaheer et al, 2021)

Key idea: replace all-pairs interactions with a family of other interactions, **like local windows**, **looking at everything**, and **random interactions**.



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

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