University of Ljubljana, Faculty of Computer and Information Science

Attention mechanism, Transformers and BERT

Prof Dr Marko Robnik-Šikonja

Natural Language Processing, Edition 2022

Contents

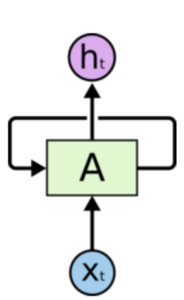
- encoder decoder networks
- attention mechanism
- transformer networks
- BERT

- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).
- Jay Alammar: The Illustrated Transformer. Blog, 2019.
- Sasha Rush: Annotated Transformer. Jupyter Notebook using PyTorch.
- Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Volume 1, pp. 4171-4186.

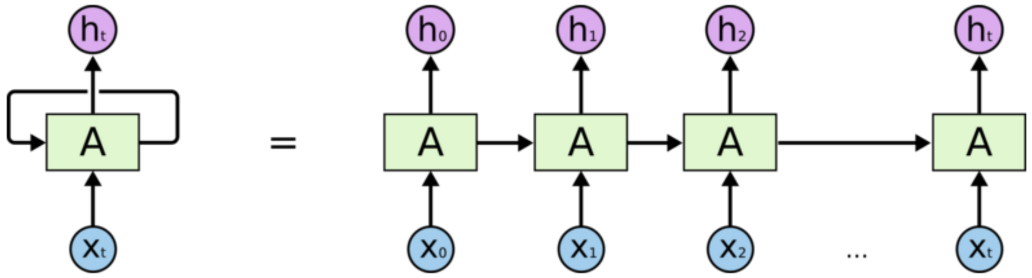
- some slides by Jay Alammar, Jacob Devlin and Andrej Miščič

Recurrent Neural Networks

- Recurrent Neural Networks are networks with loops in them, allowing information to persist.



Recurrent Neural Networks have loops.

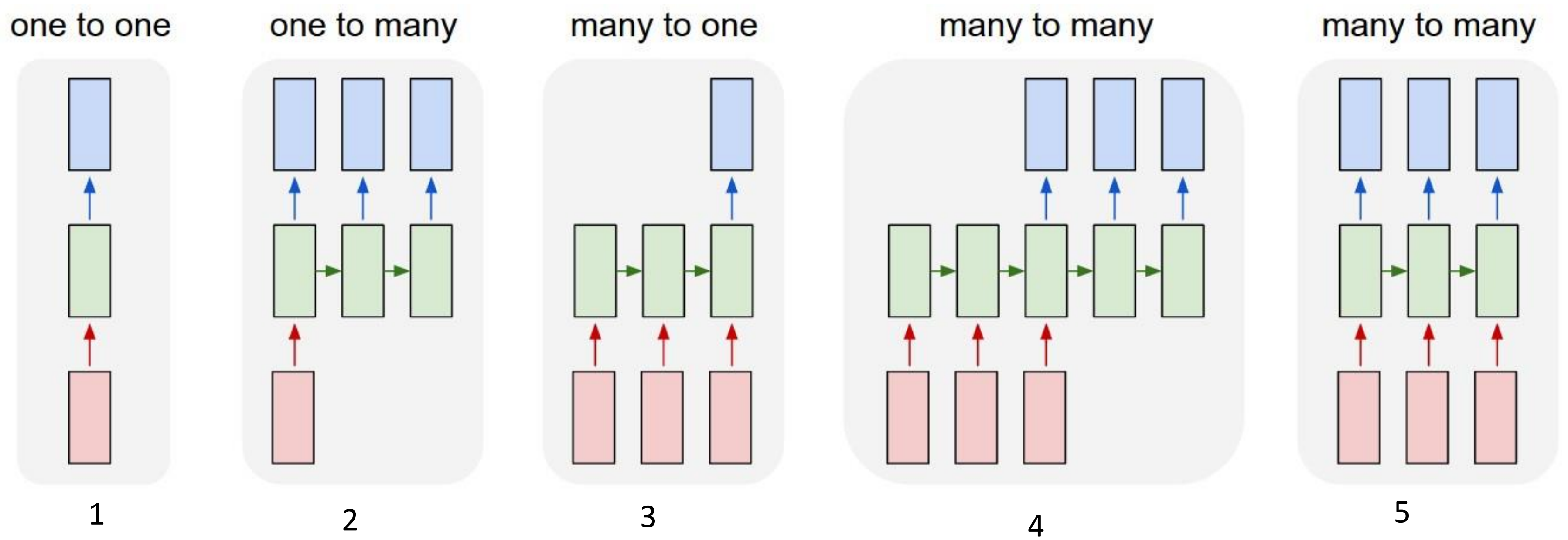


An unrolled recurrent neural network.

In the above diagram, a chunk of neural network, ***A***, looks at some input ***x_t*** and outputs a value ***h_t***.
A loop allows information to be passed from one step of the network to the next.

A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.
The diagram above shows what happens if we unroll the loop.

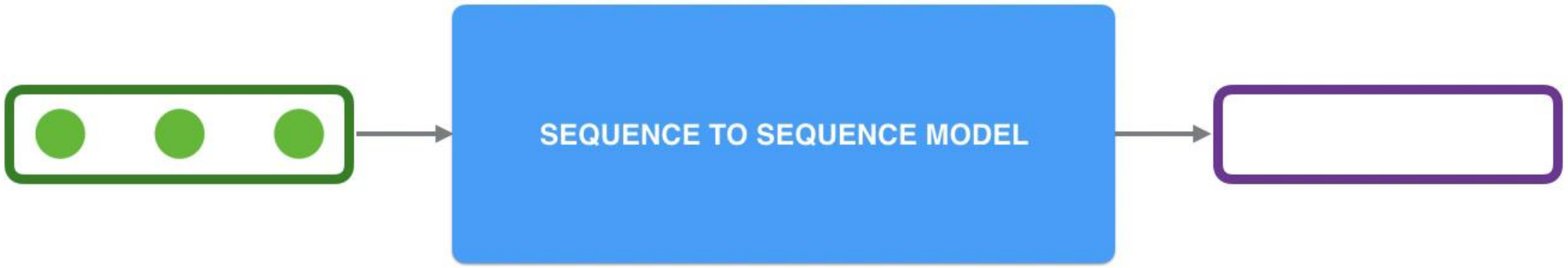
Examples of Recurrent Neural Networks



- Each rectangle is a vector and arrows represent functions (e.g. matrix multiply).
- Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state

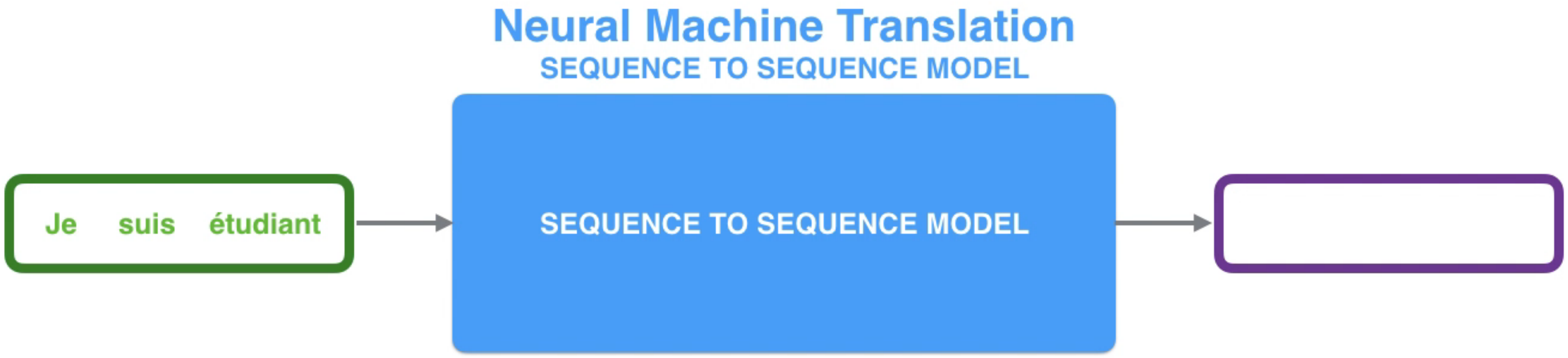
1. Standard mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification).
2. Sequence output (e.g. image captioning takes an image and outputs a sentence of words).
3. Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).
4. Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).
5. Synced sequence input and output (e.g. video classification where we wish to label each frame of the video).

Seq2Seq model



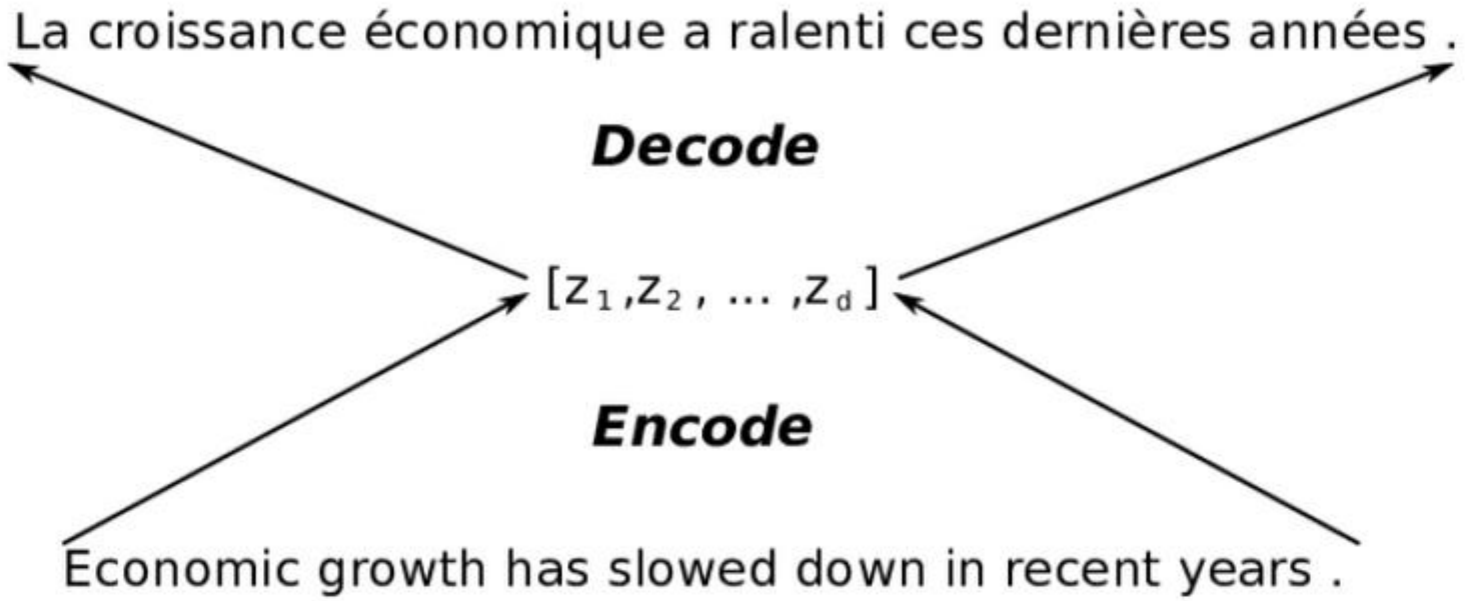
Videos by Jay Alammar: Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention), 2018

Seq2Seq for NMT


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

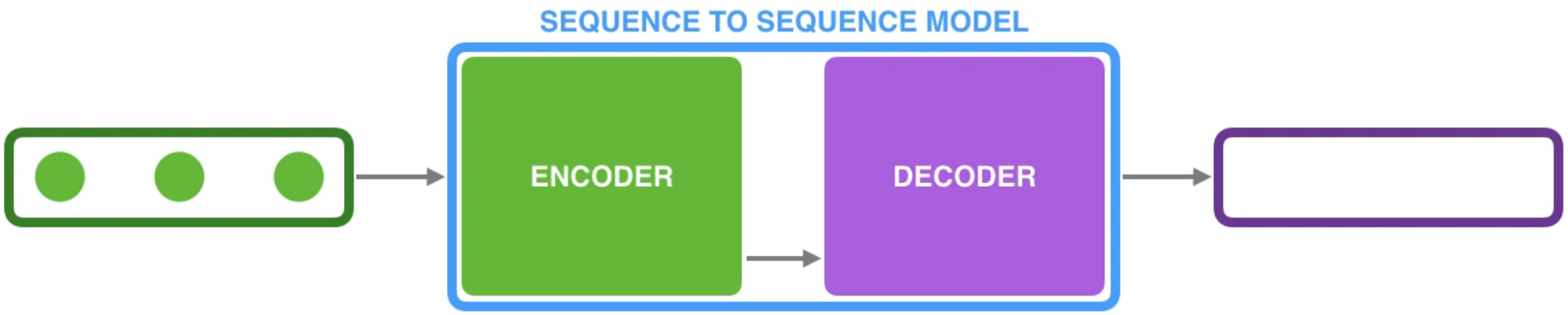
Encoder-Decoder model

- encode into a latent space

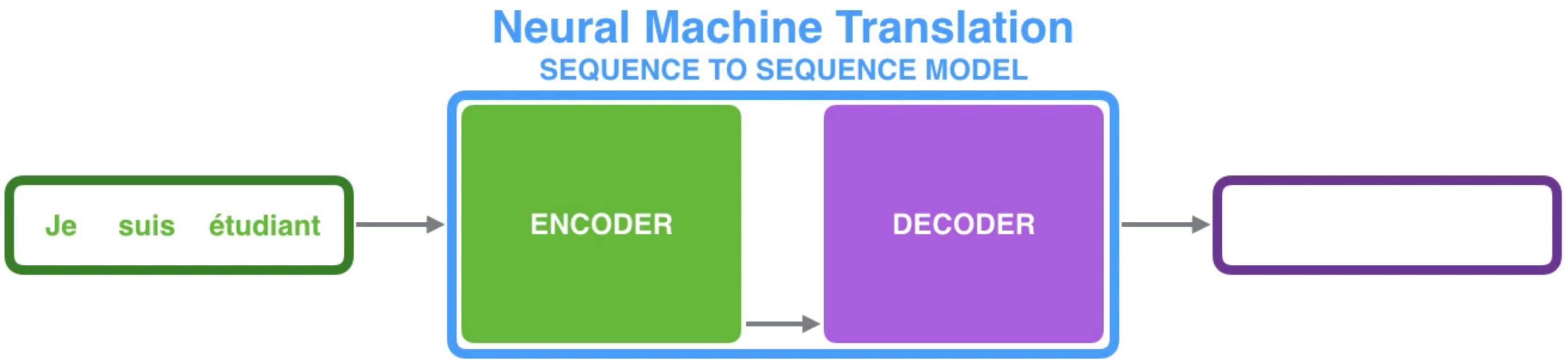


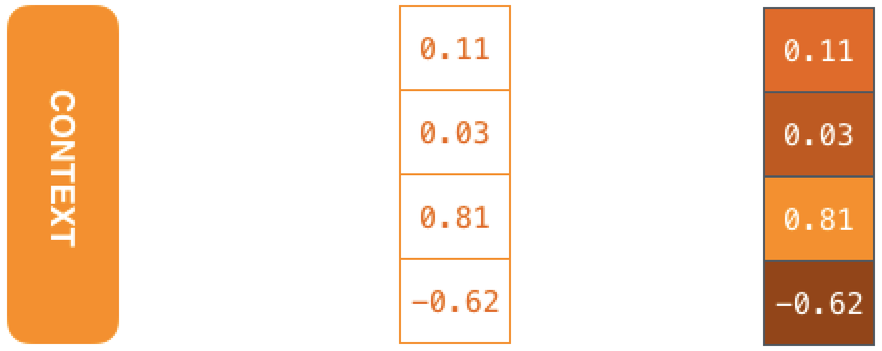
La croissance économique a ralenti ces dernières années .
Decode
$[z_1, z_2, \ldots, z_d]$
Encode
Economic growth has slowed down in recent years .

Encoder-decoder for sequences

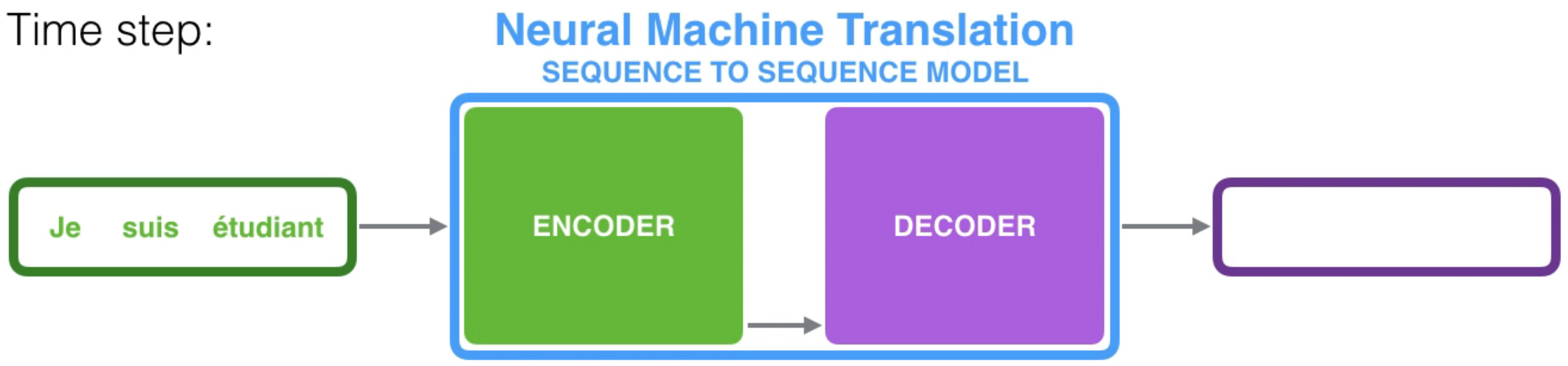


Encoder-decoder for NMT

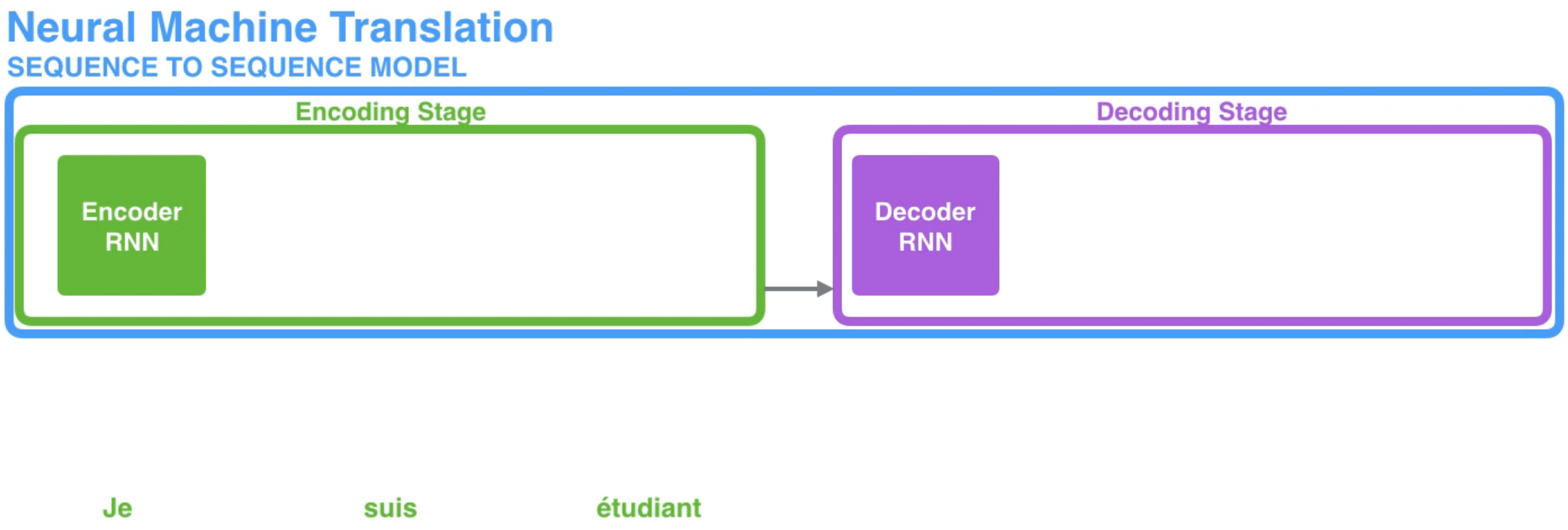




Encoder-decoder hidden states



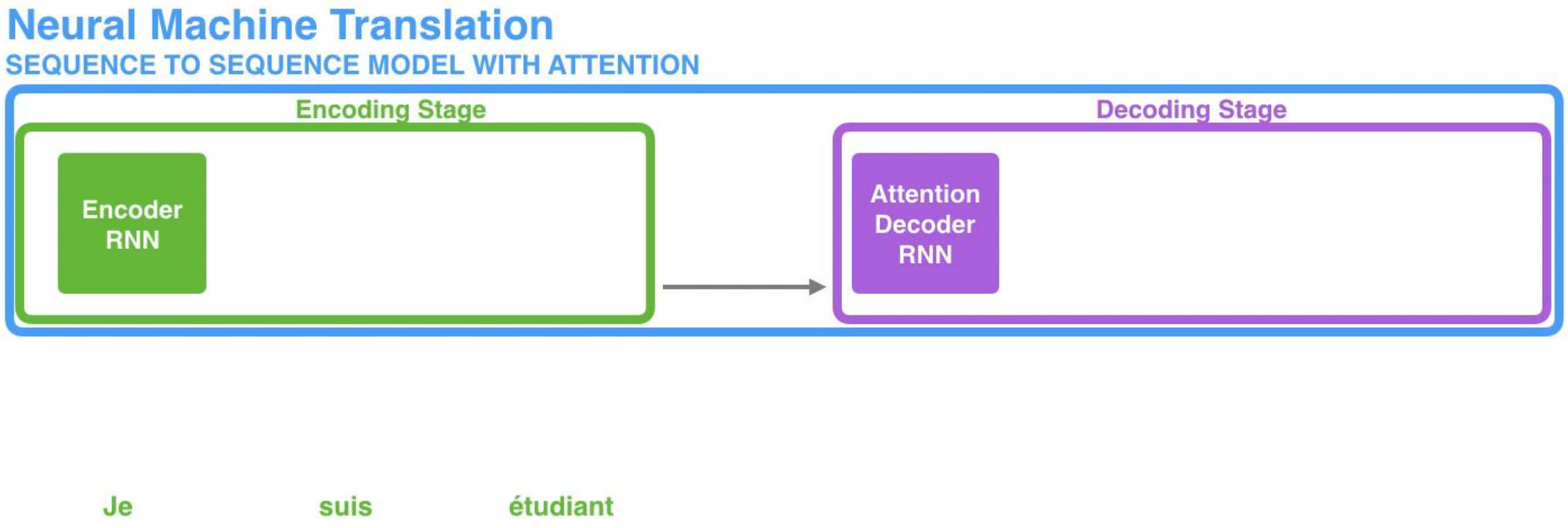
Unrolled encoder-decoder



Problems of encoder-decoder models

- long dependencies that would require larger networks and many more training data
- the information of different length sentences is stored in the fixed length hidden layer (migh be too long or to short)
- solution: attention mechanism

NMT with attention



Attention mechanism implementation for RNNs 1/2

- for all input words, we store their hidden layer weights
- during decoding, we add these vectors to the decoder input
- we use bidirectional encoding (forward and backward LM) and concatenate both weight vectors
- vectors are stored into a matrix

$$\overrightarrow{\boldsymbol{h}}_j^{(f)} = \mathrm{RNN}(\mathrm{embed}(f_j), \overrightarrow{\boldsymbol{h}}_{j-1}^{(f)})$$
$$\overleftarrow{\boldsymbol{h}}_j^{(f)} = \mathrm{RNN}(\mathrm{embed}(f_j), \overleftarrow{\boldsymbol{h}}_{j+1}^{(f)}).$$

$$\boldsymbol{h}_j^{(f)} = [\overleftarrow{\boldsymbol{h}}_j^{(f)}; \overrightarrow{\boldsymbol{h}}_j^{(f)}].$$

$$H^{(f)} = \mathrm{concat\_col}(\boldsymbol{h}_1^{(f)}, \ldots, \boldsymbol{h}_{|F|}^{(f)}).$$

Attention mechanism implementation for RNNs 2/2

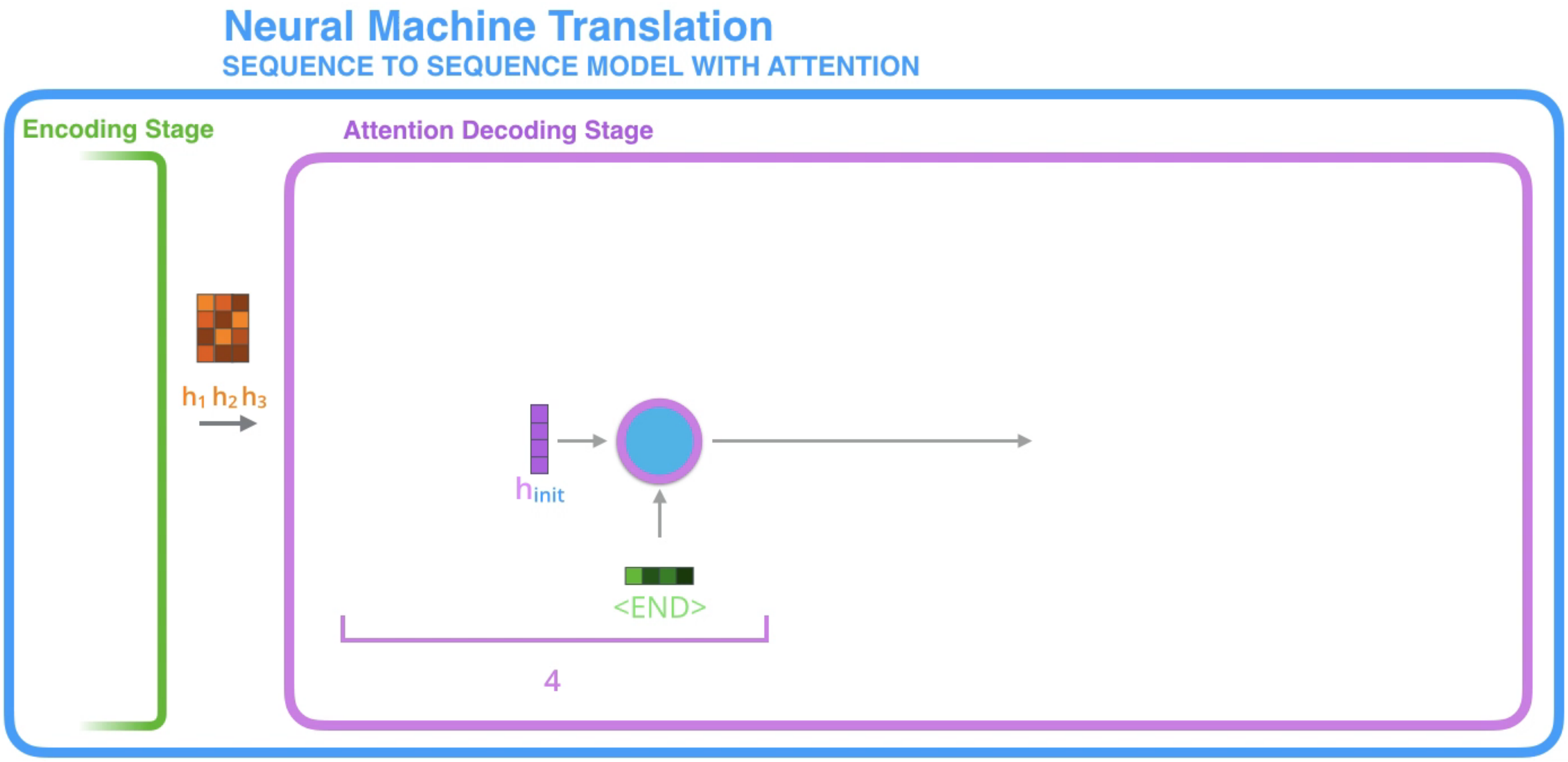
- we train the attention – which stored vectors are more or less important for decoding certain words
- the importance is determined with the attention vector α_t (between 0 and 1, sums to 1), applied to stored hidden weights and given as additional input to the decoder

$$\boldsymbol{c}_t = H^{(f)}\boldsymbol{\alpha}_t$$

Illustration of attention

Attention at time step 4

Decoder with attention



Attention produces alignments

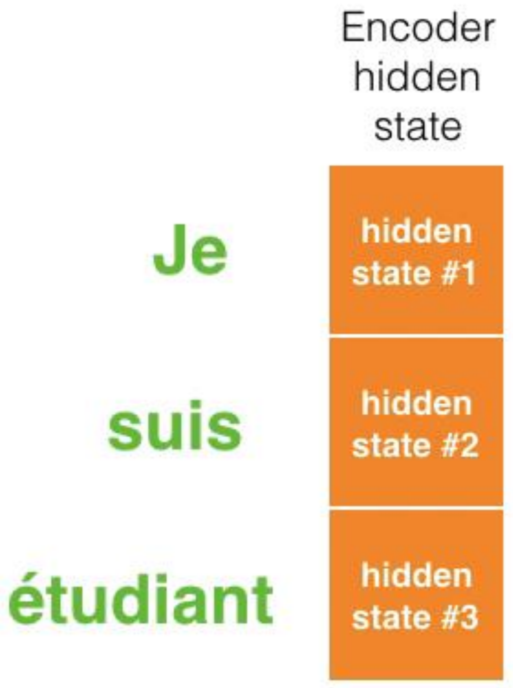
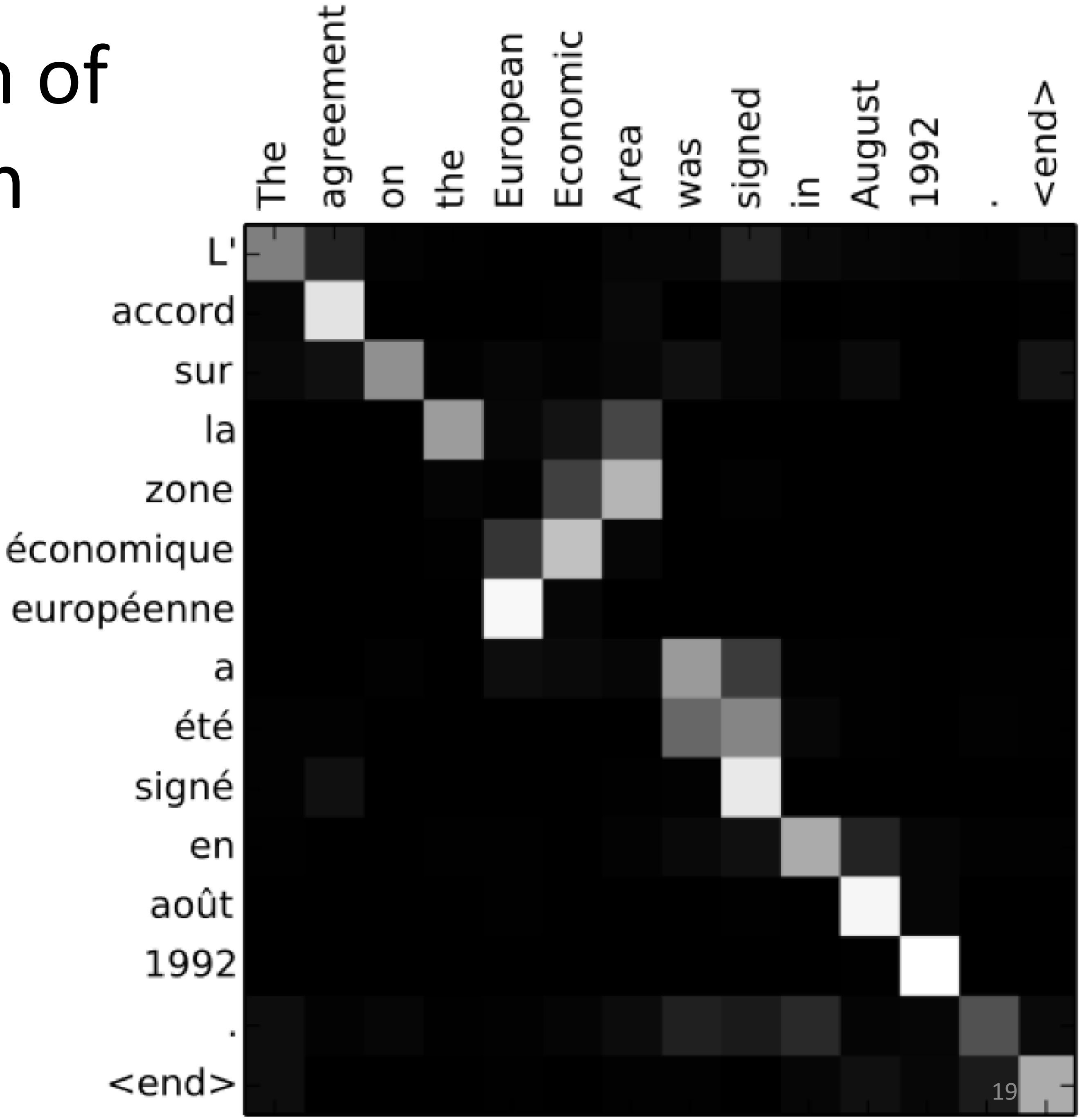
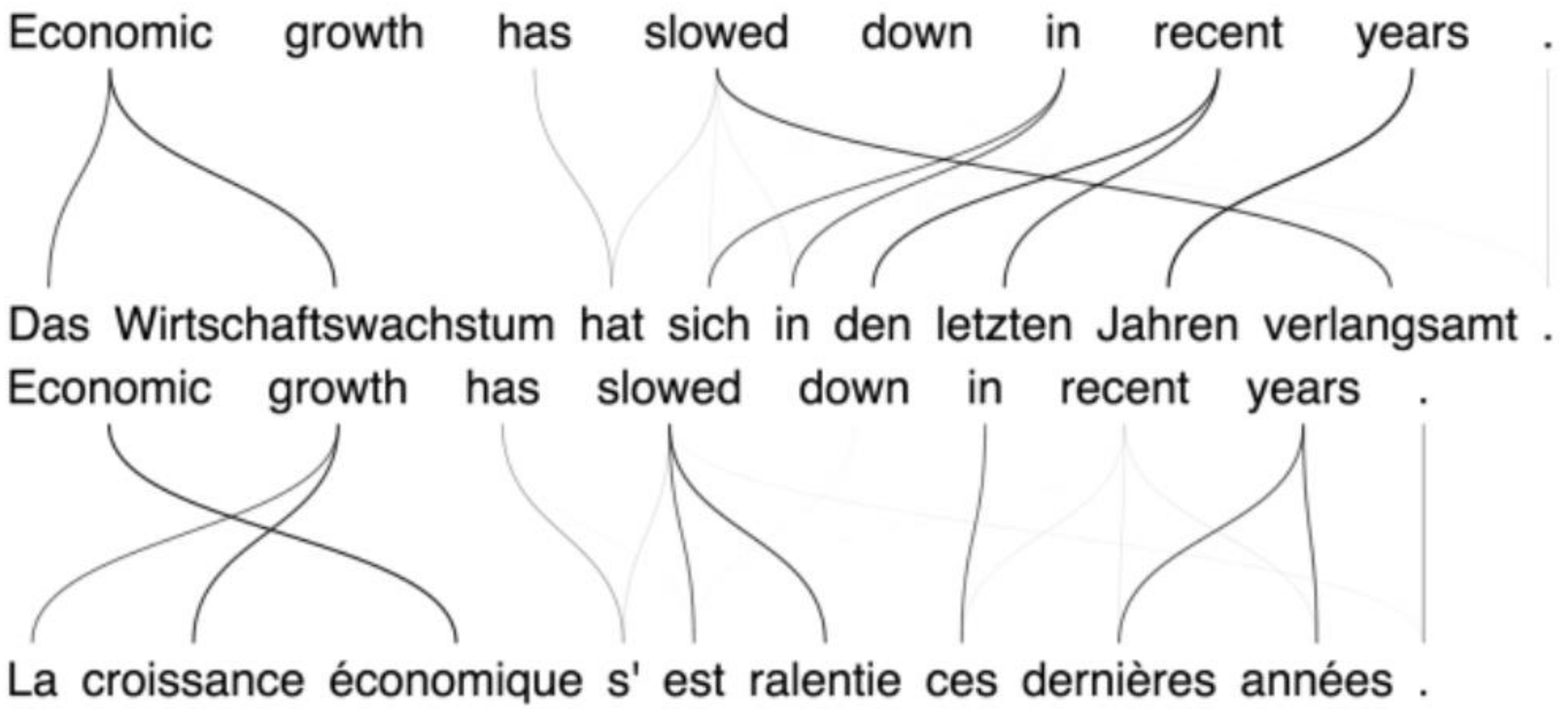


Illustration of attention



Attention illustration


Economic growth has slowed down in recent years .
Das Wirtschaftswachstum hat sich in den letzten Jahren verlangsamt .
Economic growth has slowed down in recent years .
La croissance économique s' est ralentie ces dernières années .

Problems with RNNs

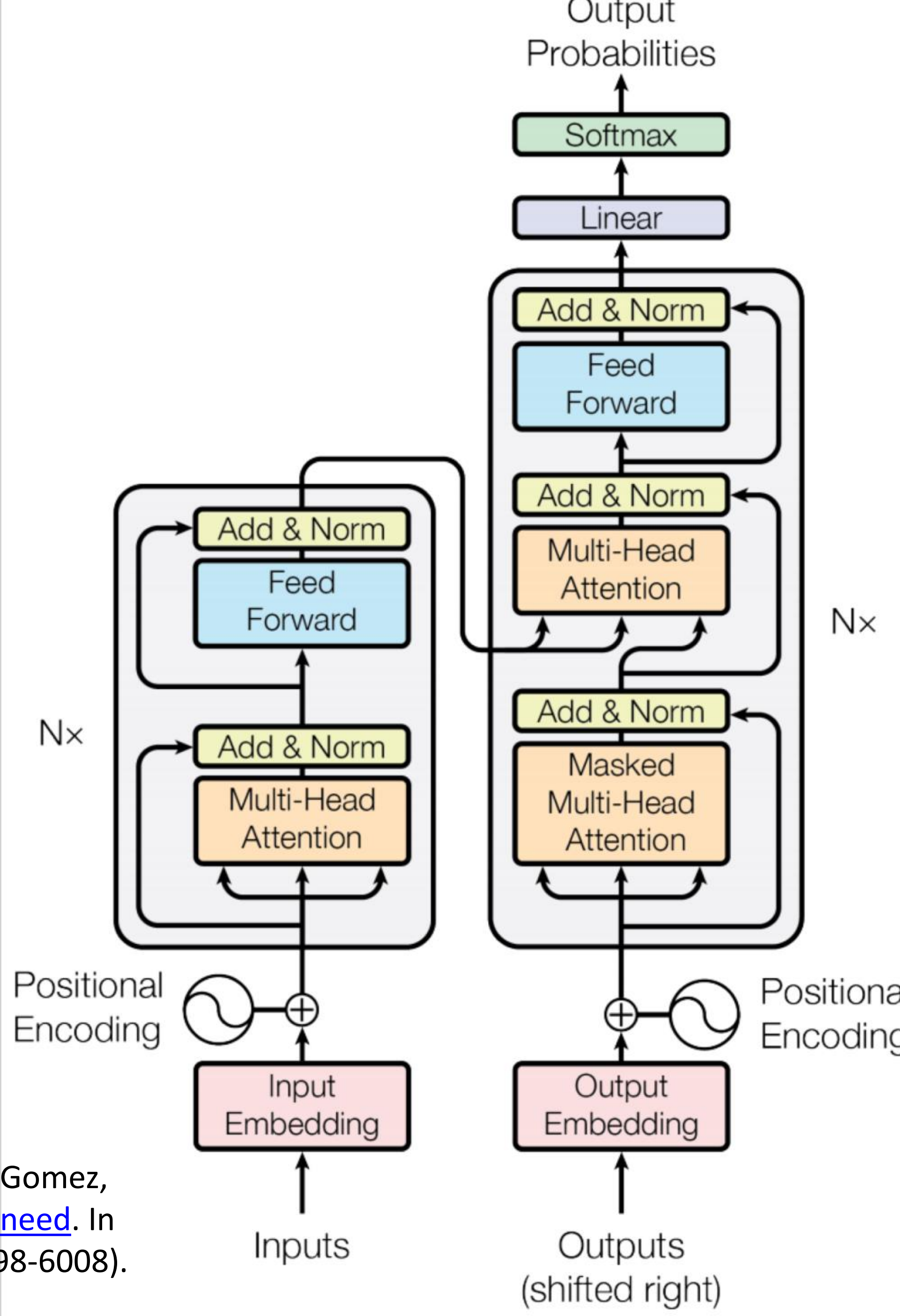
- We want parallelization but RNNs are inherently sequential
- Despite GRUs and LSTMs, RNNs still need attention mechanism to deal with long range dependencies – path length between states grows with sequence
- If attention gives us access to any state… maybe we can just use attention and don't need the RNN?

Transformer model

- currently the most successful DNN
- non-recurrent
- architecturally it is an encoder-decoder model
- fixed input length (relatively long)
- can be parallelized
- adapted for GPU (TPU) processing
- based on extreme use of attention

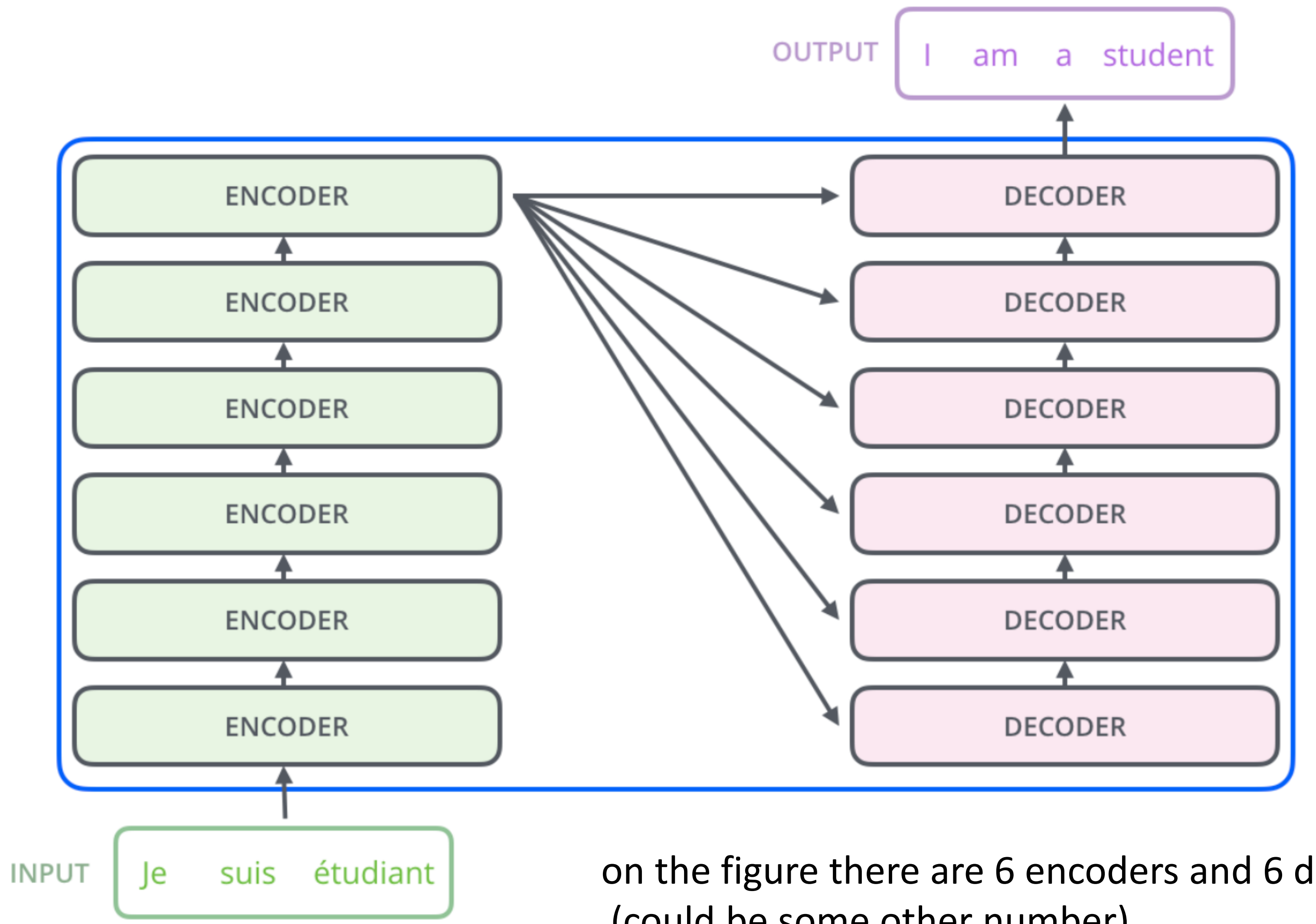
Transformer overview

- Initial task: machine translation with parallel corpus
- Predict each translated word
- Final cost/loss/error function is standard cross-entropy error on top of a softmax classifier



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

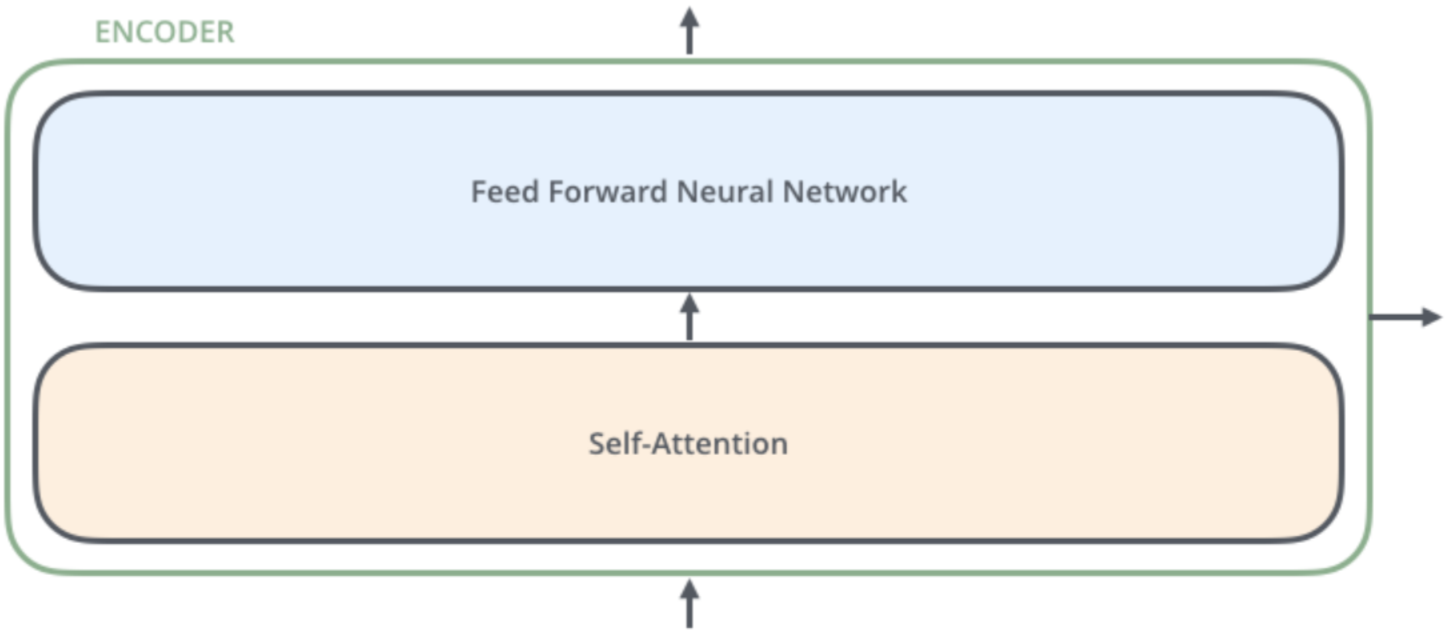
Transformer is encoder-decoder model



on the figure there are 6 encoders and 6 decoders
(could be some other number)

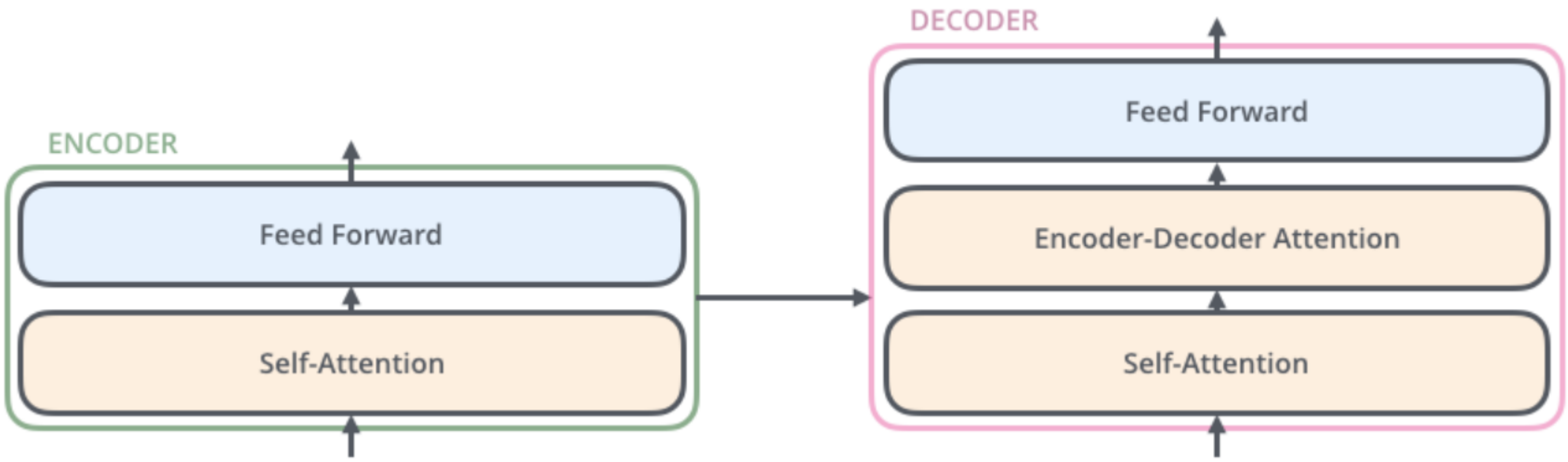
Transformer: encoder

- two layers
- no weight sharing between different encoders
- self-attention helps to focus on relevant part of input

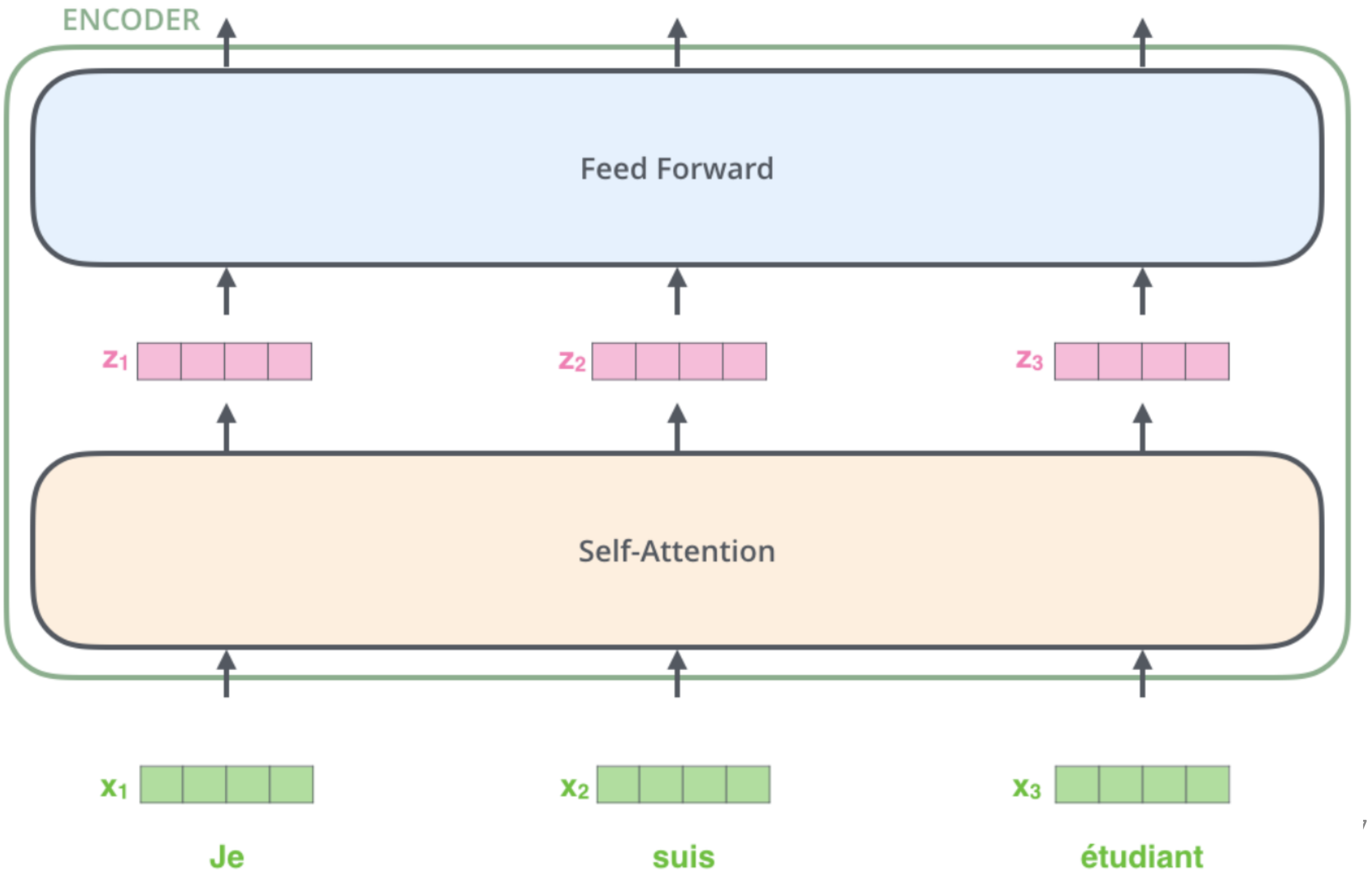


Transformer: decoder

- the same as encoder but with an additional attention layer in between, receiving input fom encoder



Start with embeddings

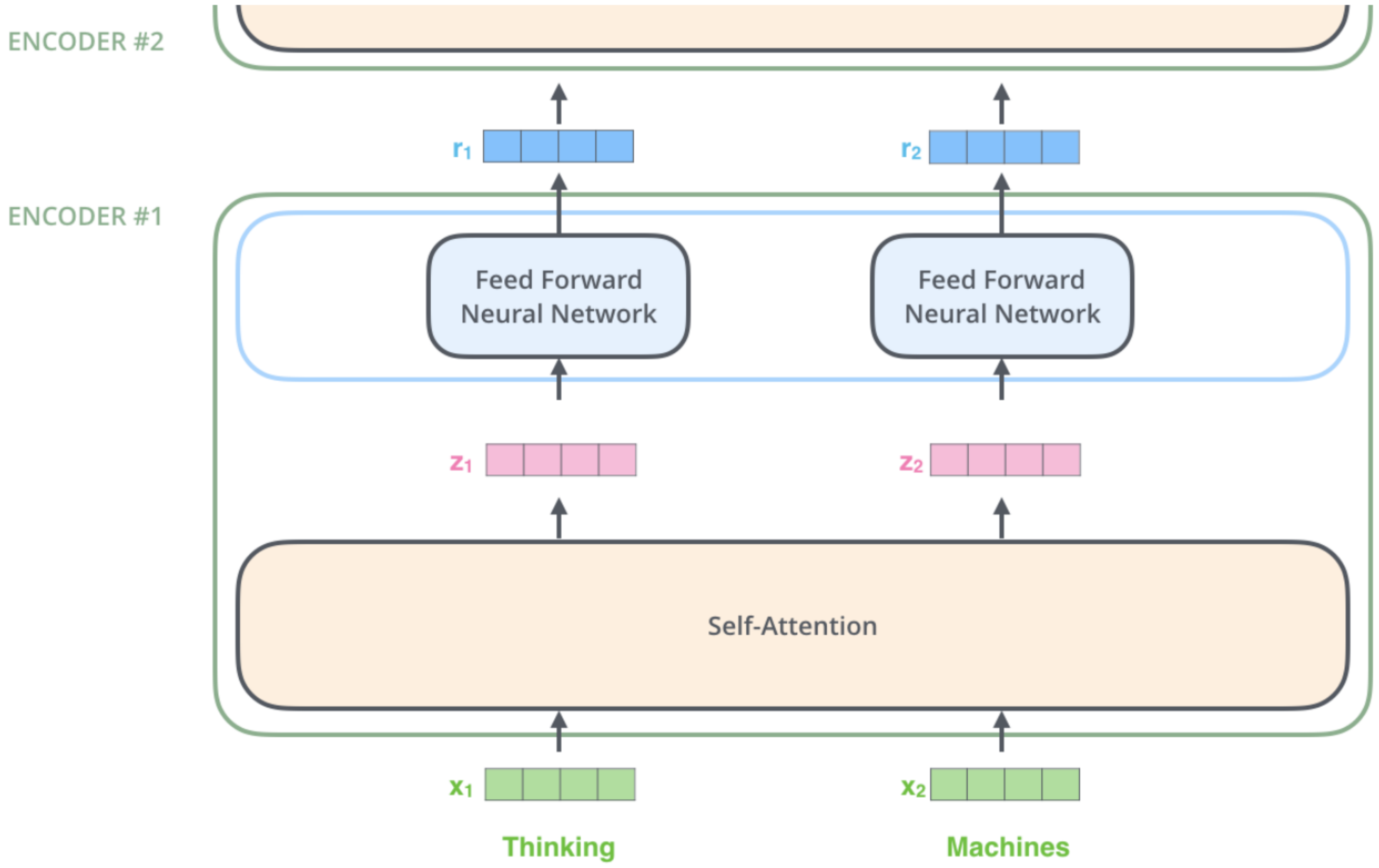


Input to transformer

- embeddings, e.g., 512 dimensional vectors (special, we will discuss that later)
- fixed length, e.g., max 128 tokens
- dependencies between inputs are only in the self-attention layer, no dependencies in feed forward layer – good for parallelization

- **Let us first present the working of the transformer with illustration of the prediction, later we will cover also training.**

Encoding


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

Self-attention

- As the model processes each word (each position in the input sequence), self-attention allows it to look at other positions in the input sequence for clues that can help lead to a better encoding for this word

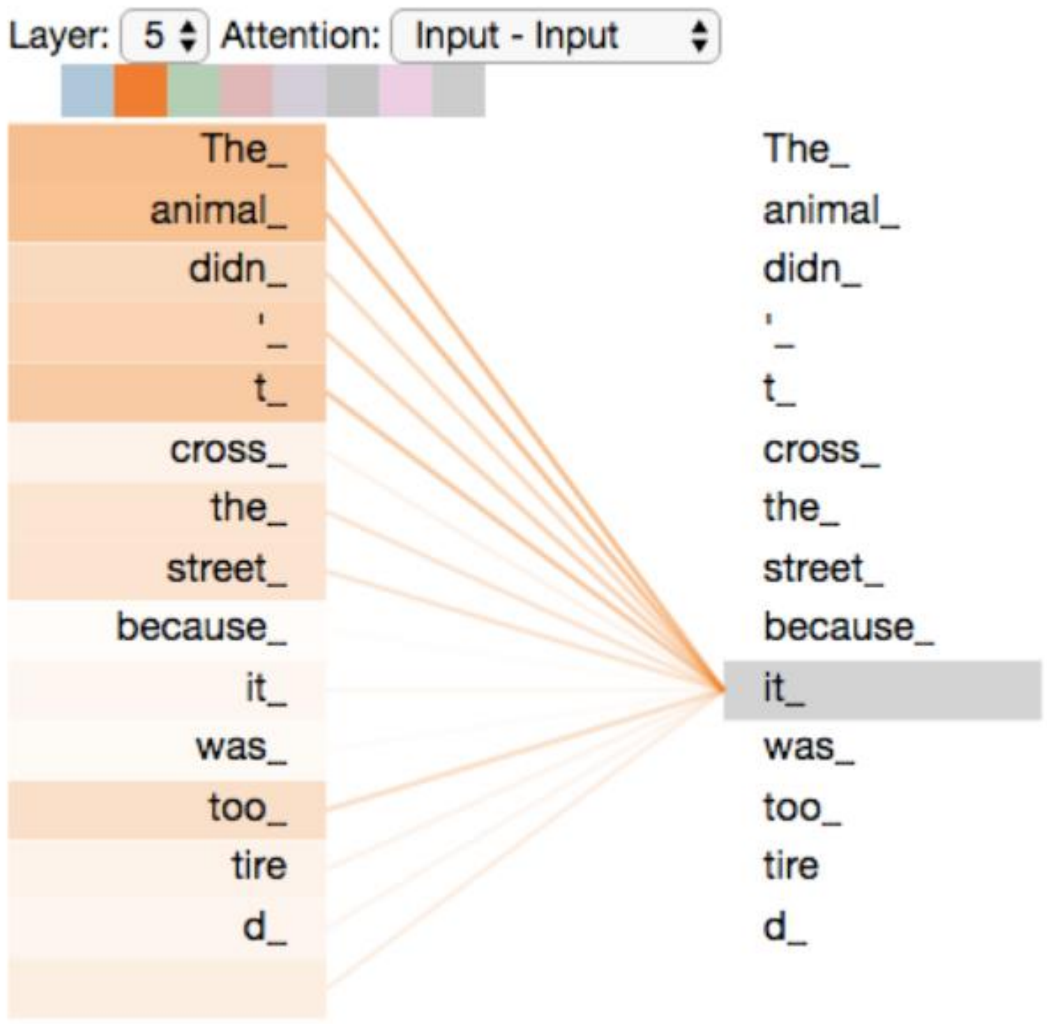
  ”The animal didn't cross the street because it was too tired”

- What does “it” in this sentence refer to? Is it referring to the street or to the animal? It’s a simple question to a human, but not as simple to an algorithm.
- *”The animal didn't cross the street because it was too wide”*

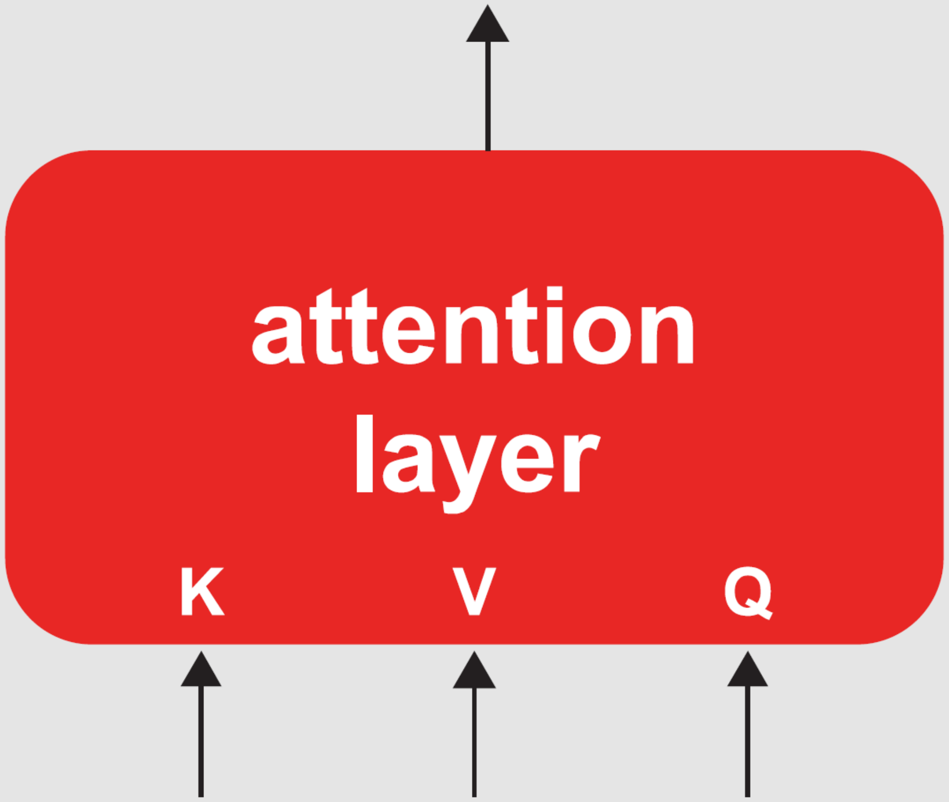
- When the model is processing the word “it”, self-attention allows it to associate “it” with “animal”.

Illustrating self-attention

- As we are encoding the word "it" in encoder #5 (the top encoder in the stack), part of the attention mechanism was focusing on "The Animal", and baked a part of its representation into the encoding of "it".



$$Y_i = \sum_{j=1}^{n} \alpha_{ij} V_j$$



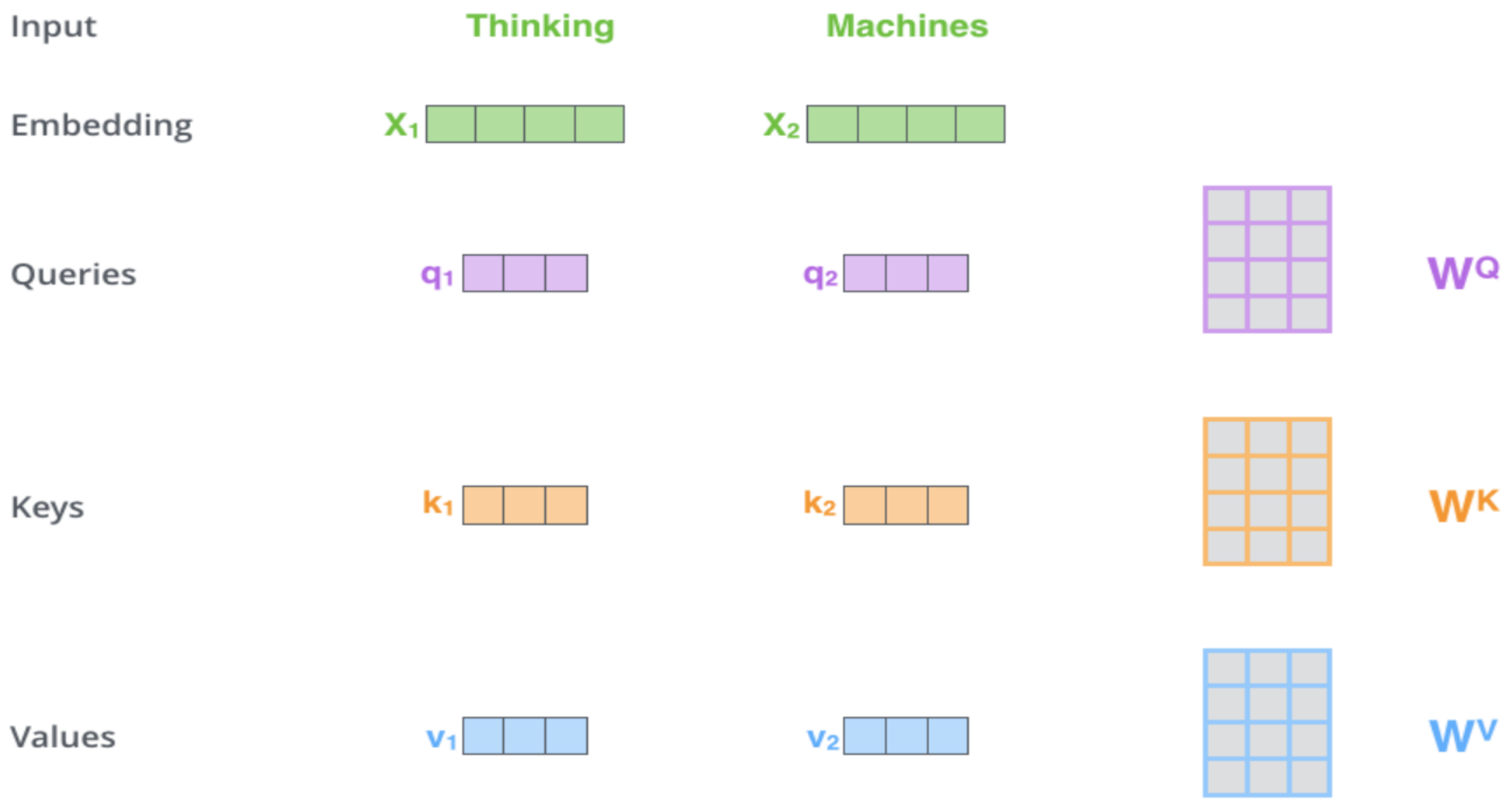
$$sim_{ij} = \phi(Q_i, K_j)$$

$$\alpha_{ij} = \frac{exp(sim_{ij})}{\sum_{k=1}^{n} exp(sim_{ik})}$$

Self-attention details 1/4

- create three vectors from each of the encoder’s input vectors (in the first layer these are the embedding of each word).
- Query vector Q, Key vector K, Value vector V are created by multiplying the embedding by three matrices that are trained during the training process.
- Q, K, and V are smaller than the embedding vector, typically 64, while the embedding and encoder input/output vectors have dimensionality of 512.
- They are smaller to make the computation of multiheaded attention (mostly) constant.

Self-attention details 1/4

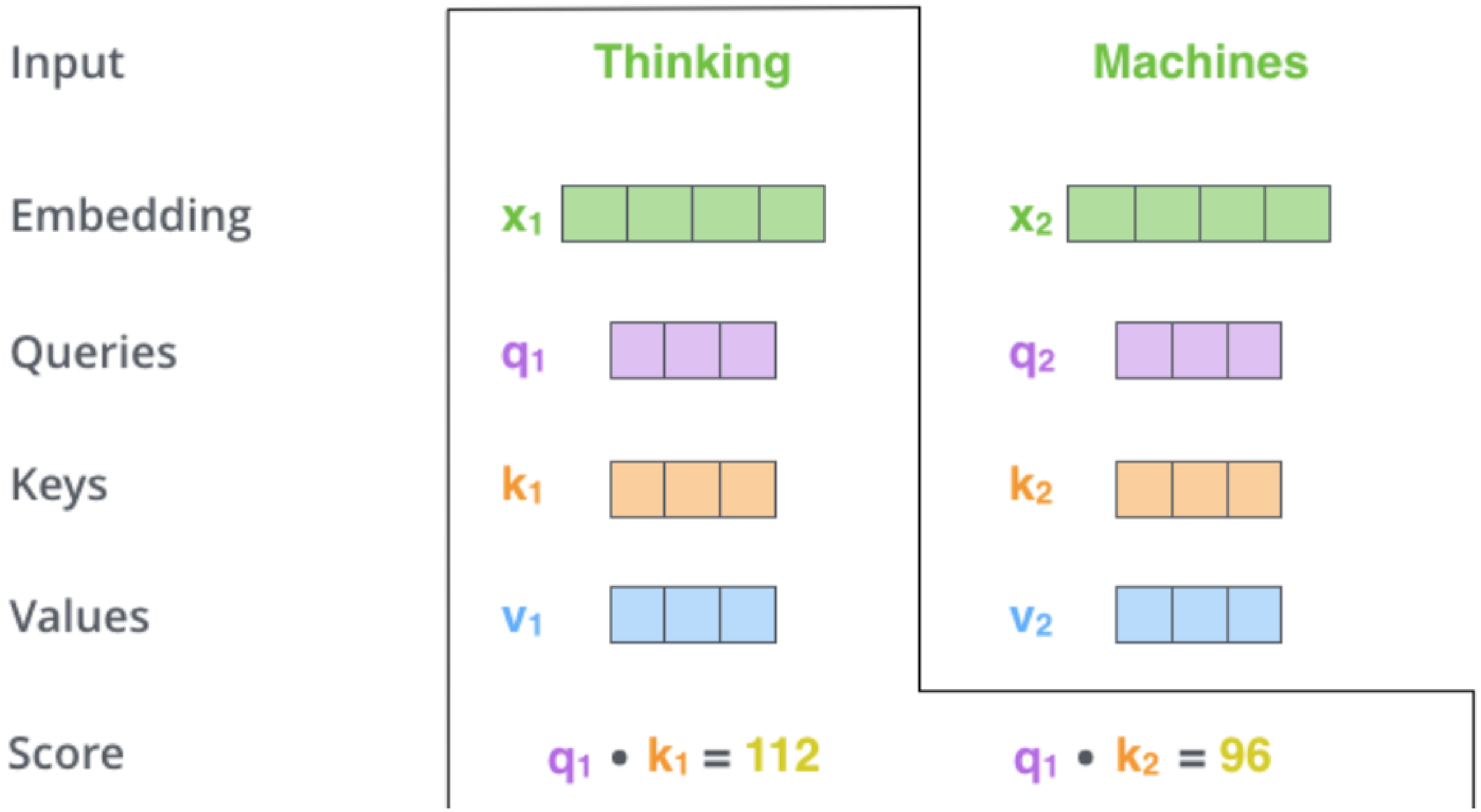


Multiplying x_1 by the W^Q weight matrix produces q_1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

Details 2/4: attention vectors Q, K, V

- The “query” Q, “key” K, and “value” V vectors are abstractions that are useful for calculating and thinking about attention.
- To calculate self-attention for a given word (e.g., “Thinking”), we score each word of the input sentence against this word. The score determines how much focus to place on other parts of the input sentence as we encode a word at a certain position.
- The score is calculated by taking the dot product of the query vector Q with the key vector K of the respective word.
- E.g., on computing the self-attention for the word in position #1, we would compute dot product of q_1 and $k_{1,}$ and q_1 and k_2.

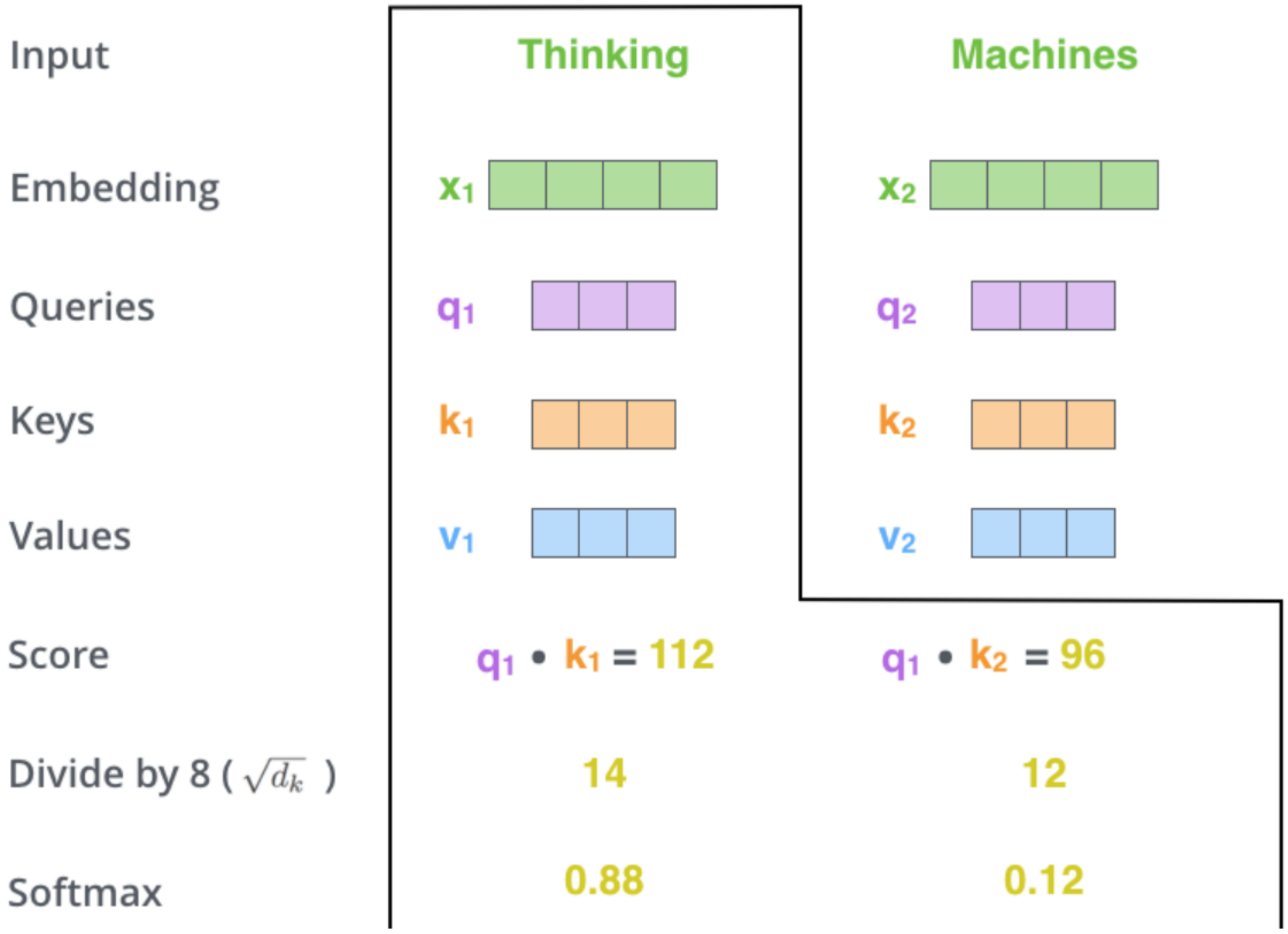
Details 2/4: scoring



Details 3/4: normalize scores

- divide the scores by the square root of the dimension of the key vectors used (in example, the vectors are of dimension 64, therefore divide by 8)
- This leads to more stable gradients.
- Then pass the result through a softmax operation. Softmax normalizes the scores so they’re all positive and add up to 1.

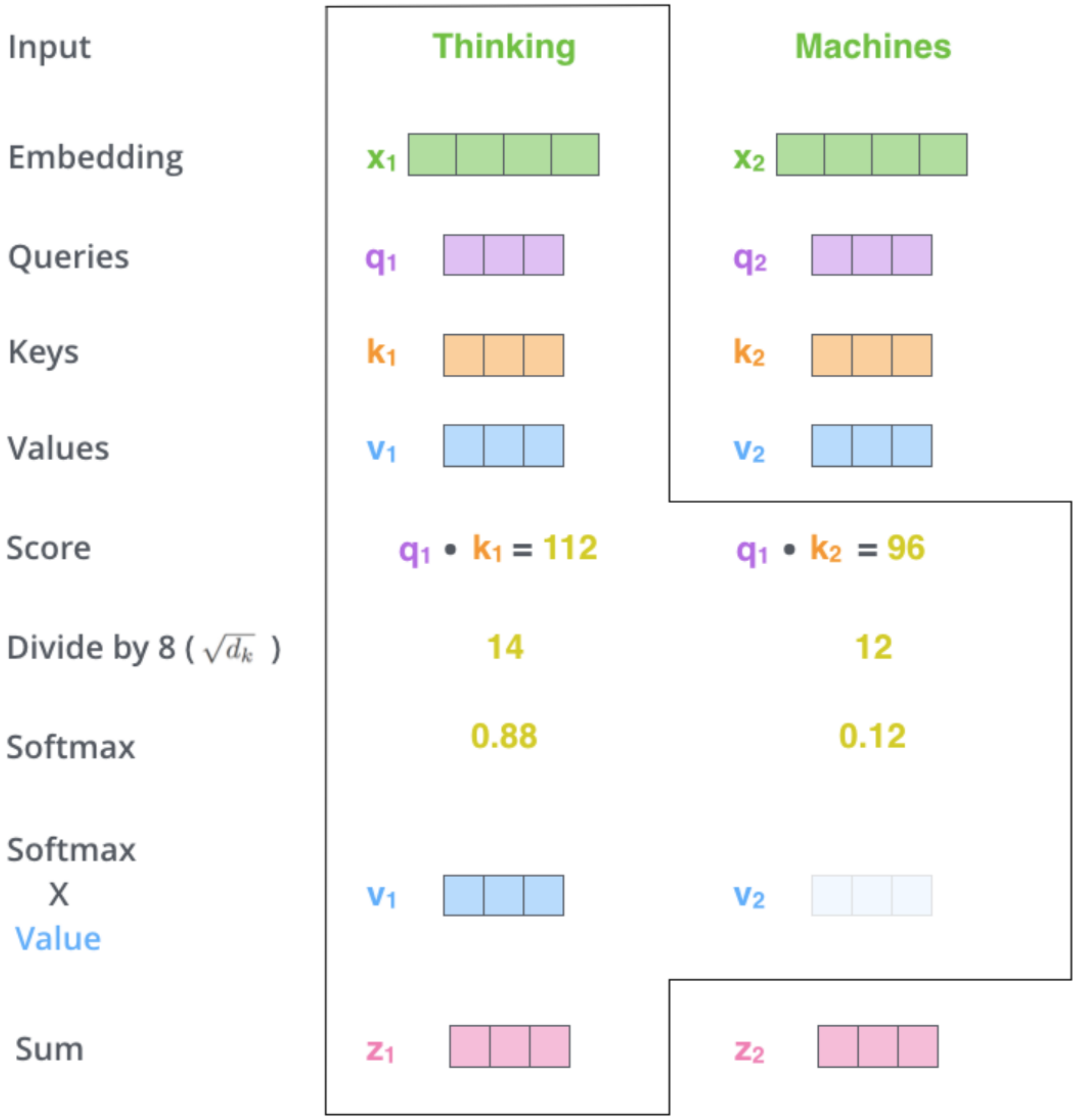
Details 3/4: normalization of scores

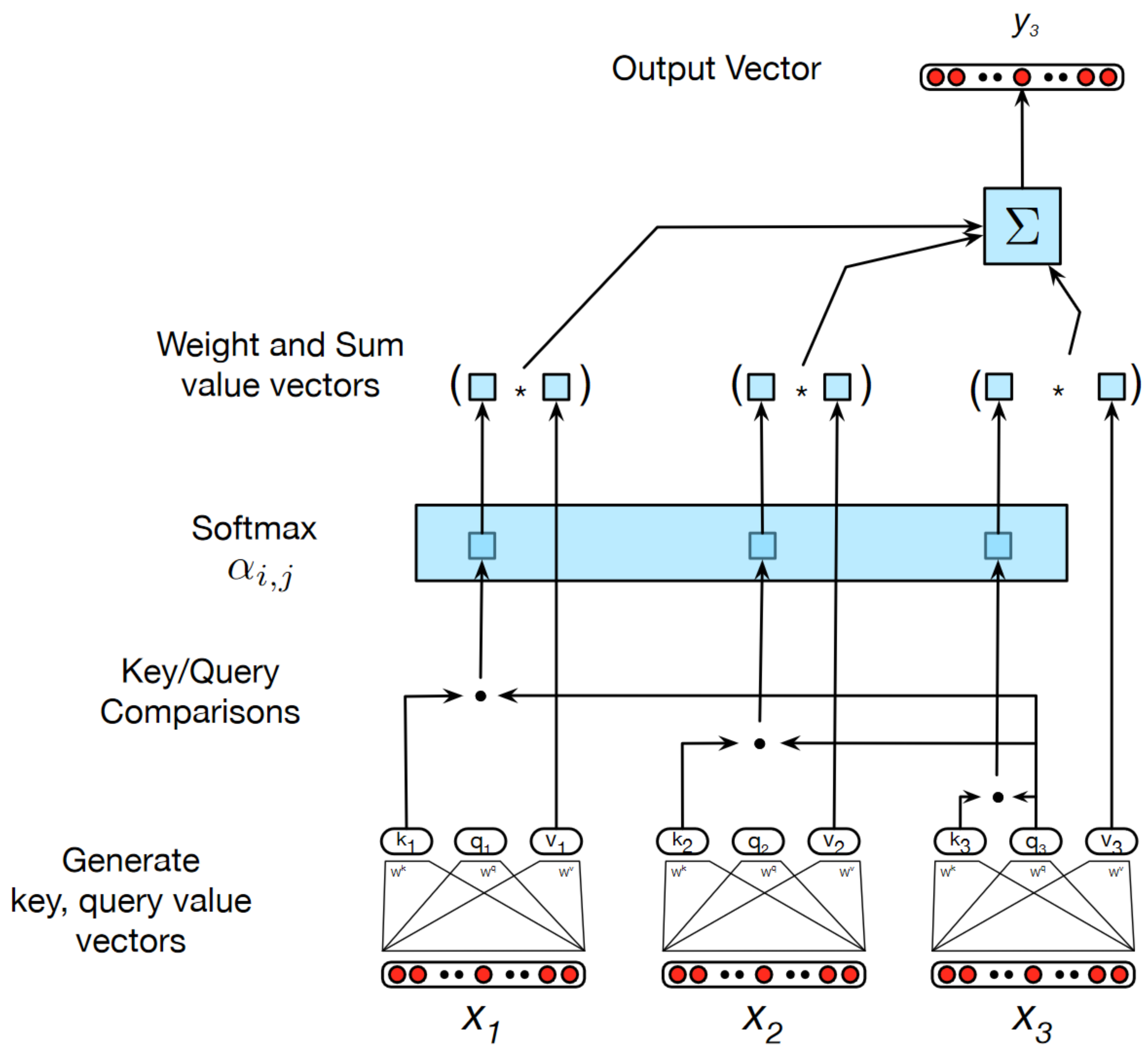


Details 4/4: apply attention scores

- The softmax score determines how much each word will be expressed at this position. Usually the word at this position will have the highest softmax score, but sometimes it’s useful to attend to another word that is relevant to the current word.
- Next multiply each value vector by the softmax score (in preparation to sum them up). The intuition is to keep intact the values of the word(s) we want to focus on, and drown-out irrelevant words (by multiplying them with small scores, e.g., 0.001).
- End computation by summing up the weighted value vectors. This produces the output of the self-attention layer at this position (for the given word – the first one in the example).
- The resulting vector is send to the feed-forward neural network.
- In the actual implementation, however, the calculation is done in matrix form for faster processing.

Details 4/4: self-attention output

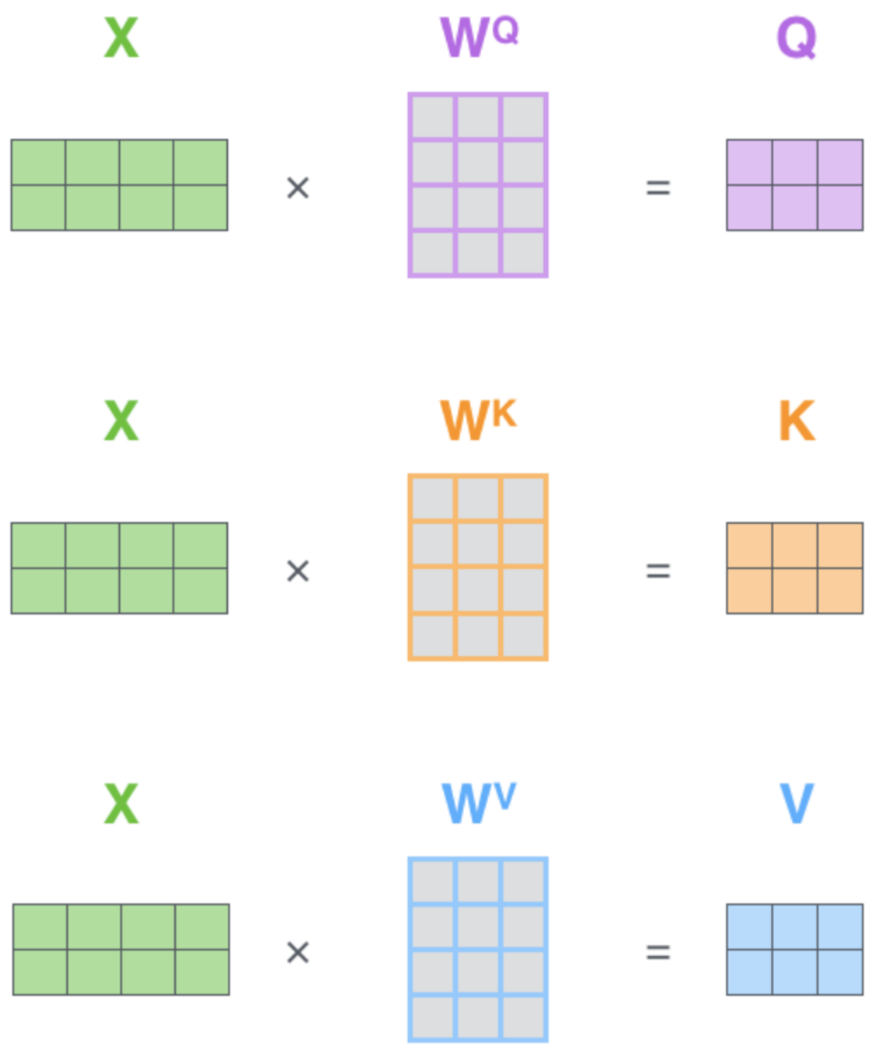




y_3
Output Vector
Σ
Weight and Sum value vectors
Softmax $\alpha_{i,j}$
Key/Query Comparisons
Generate key, query value vectors
k_1 q_1 v_1
k_2 q_2 v_2
k_3 q_3 v_3
W^k W^q W^v
x_1
x_2
x_3

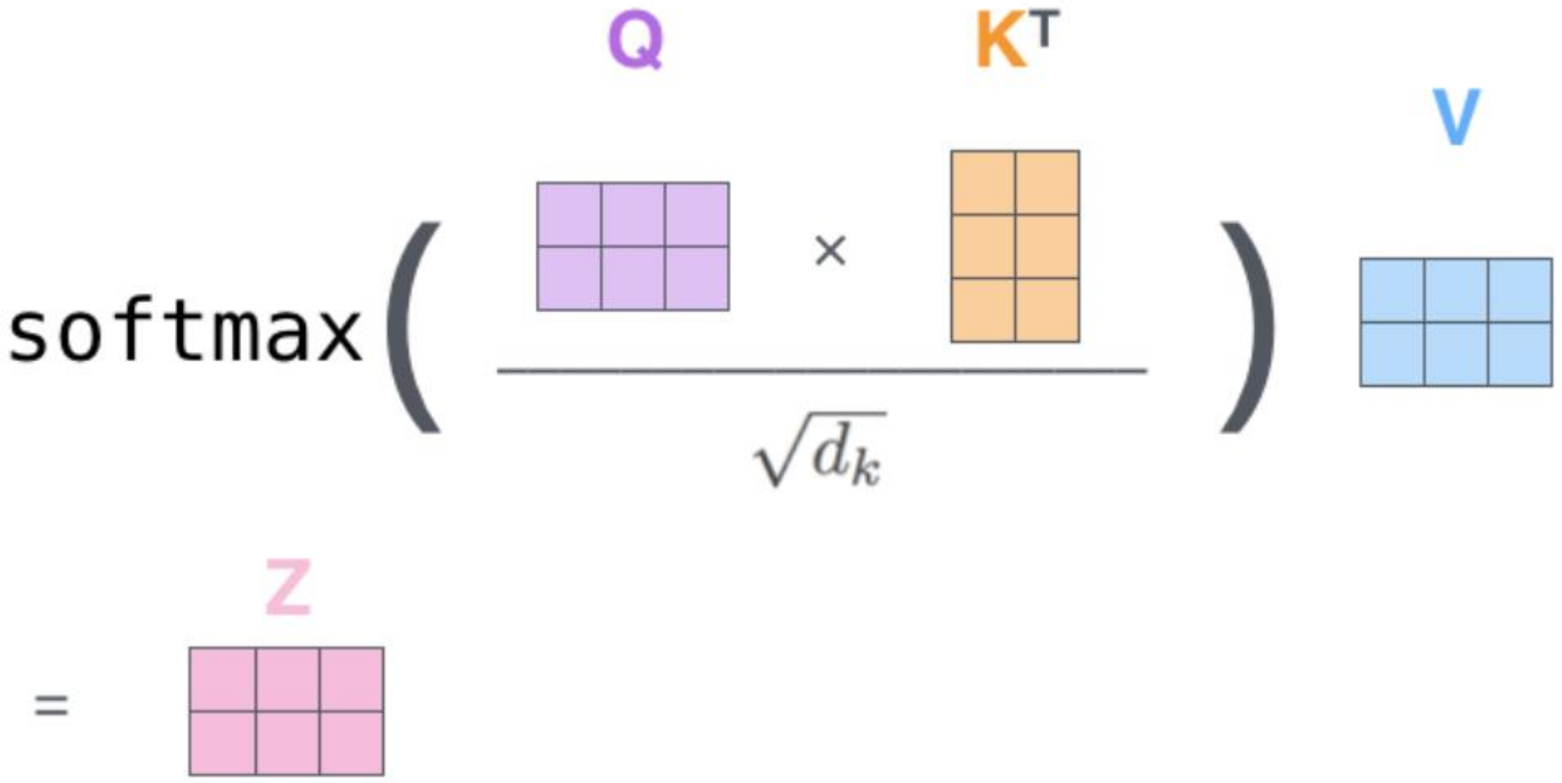
Matrix calculation of self-attention 1/2

Every row in the X matrix corresponds to a word in the input sentence.
The embedding vector (512) is larger then the q/k/v vectors (64)



Matrix calculation of self-attention 2/2

- final calculation



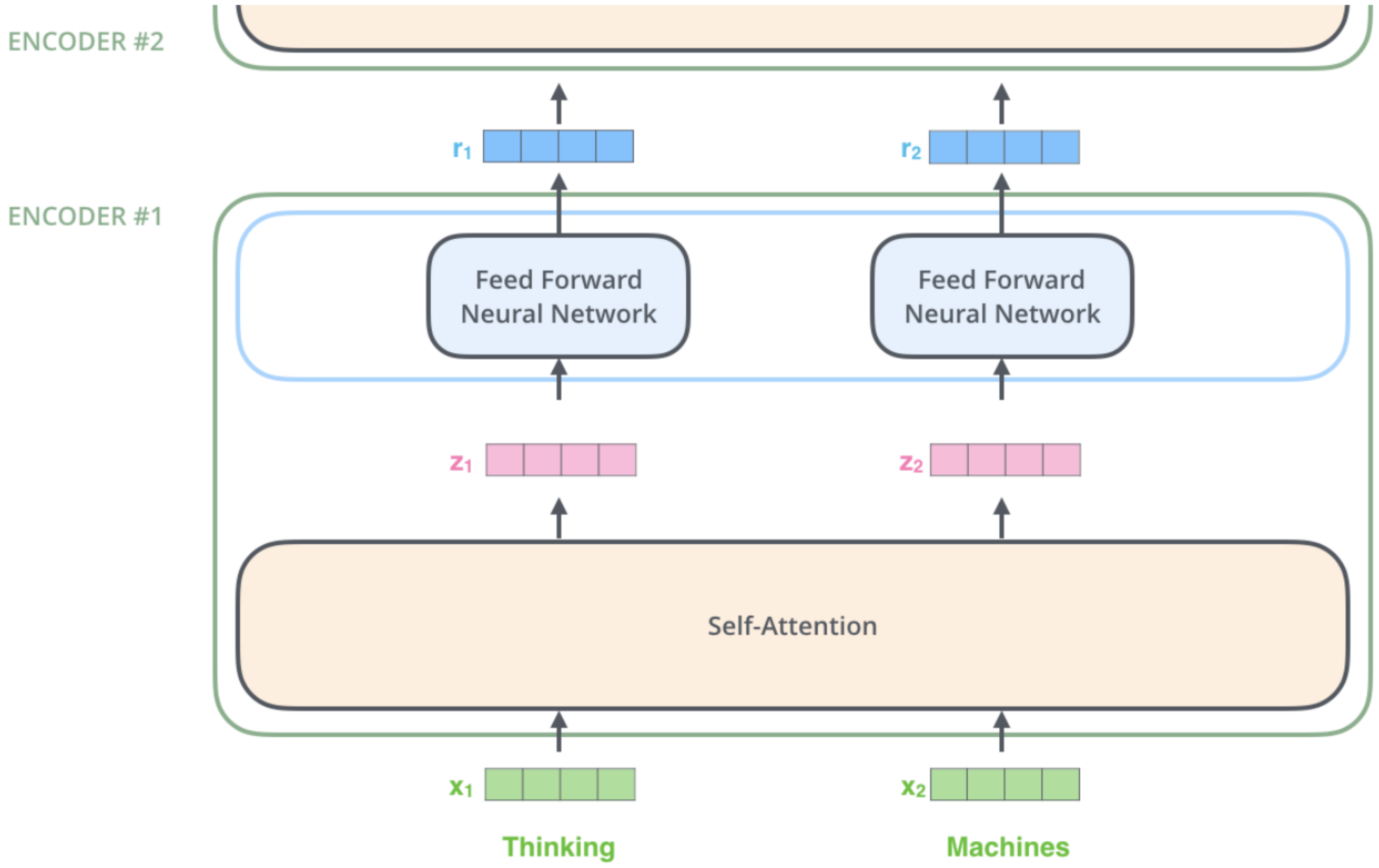
$$Z = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Computing attention head

$$A(q, K, V) = \sum_i \frac{e^{q \cdot k_i}}{\sum_j e^{q \cdot k_j}} v_i$$

$$A(Q, K, V) = softmax(QK^T)V$$

Encoding


ENCODER #2
r1
r2
ENCODER #1
Feed Forward Neural Network
Feed Forward Neural Network
z1
z2
Self-Attention
x1
x2
Thinking
Machines

In summary 1/3

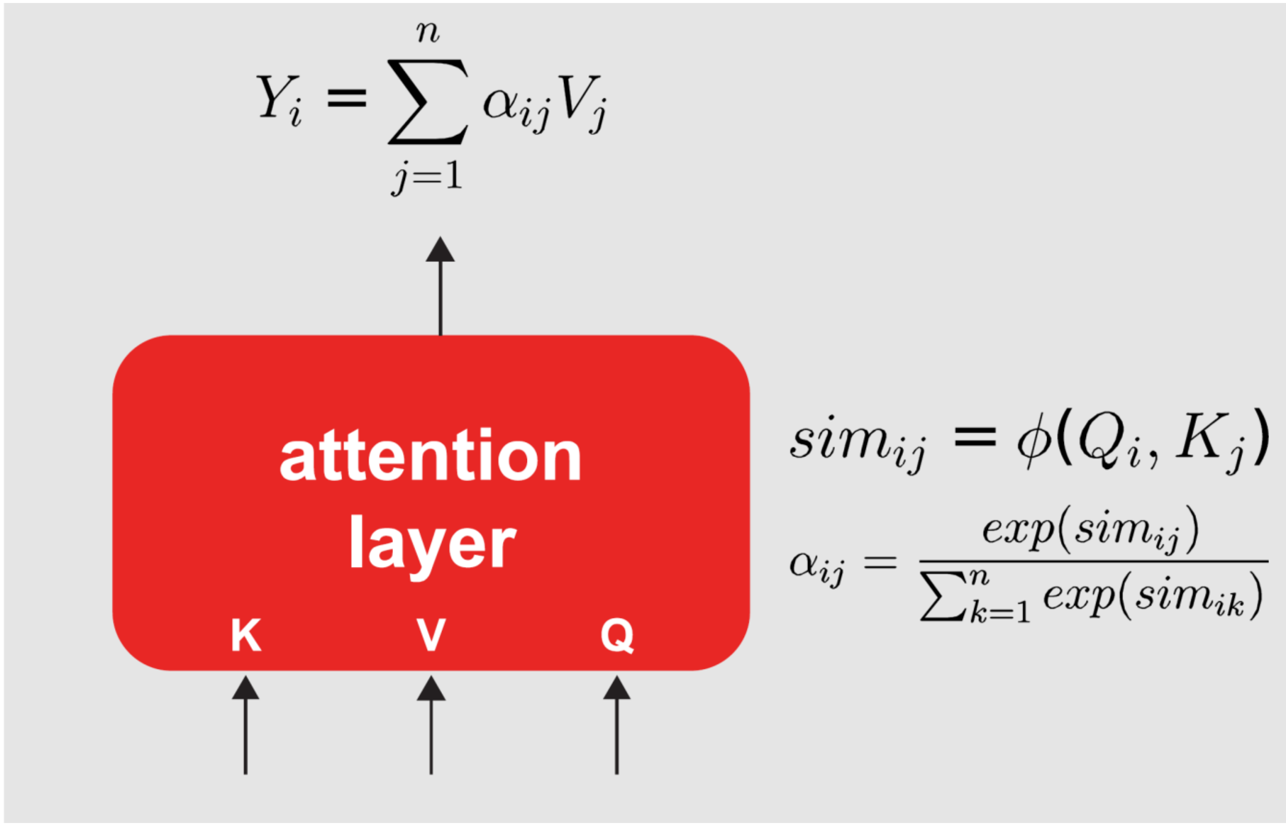
Self-attention

- recall attention:

summary of encoder states based on similarity between a particular decoder state and encoder states

- Transformer generalizes:

summary of values V based on similarity between a particular query Qi and keys K



$$Y_i = \sum_{j=1}^{n} \alpha_{ij} V_j$$

$$sim_{ij} = \phi(Q_i, K_j)$$

$$\alpha_{ij} = \frac{exp(sim_{ij})}{\sum_{k=1}^{n} exp(sim_{ik})}$$

In summary 2/3

Self-attention

Notation:

n - input length

***d*m** - representation dim.

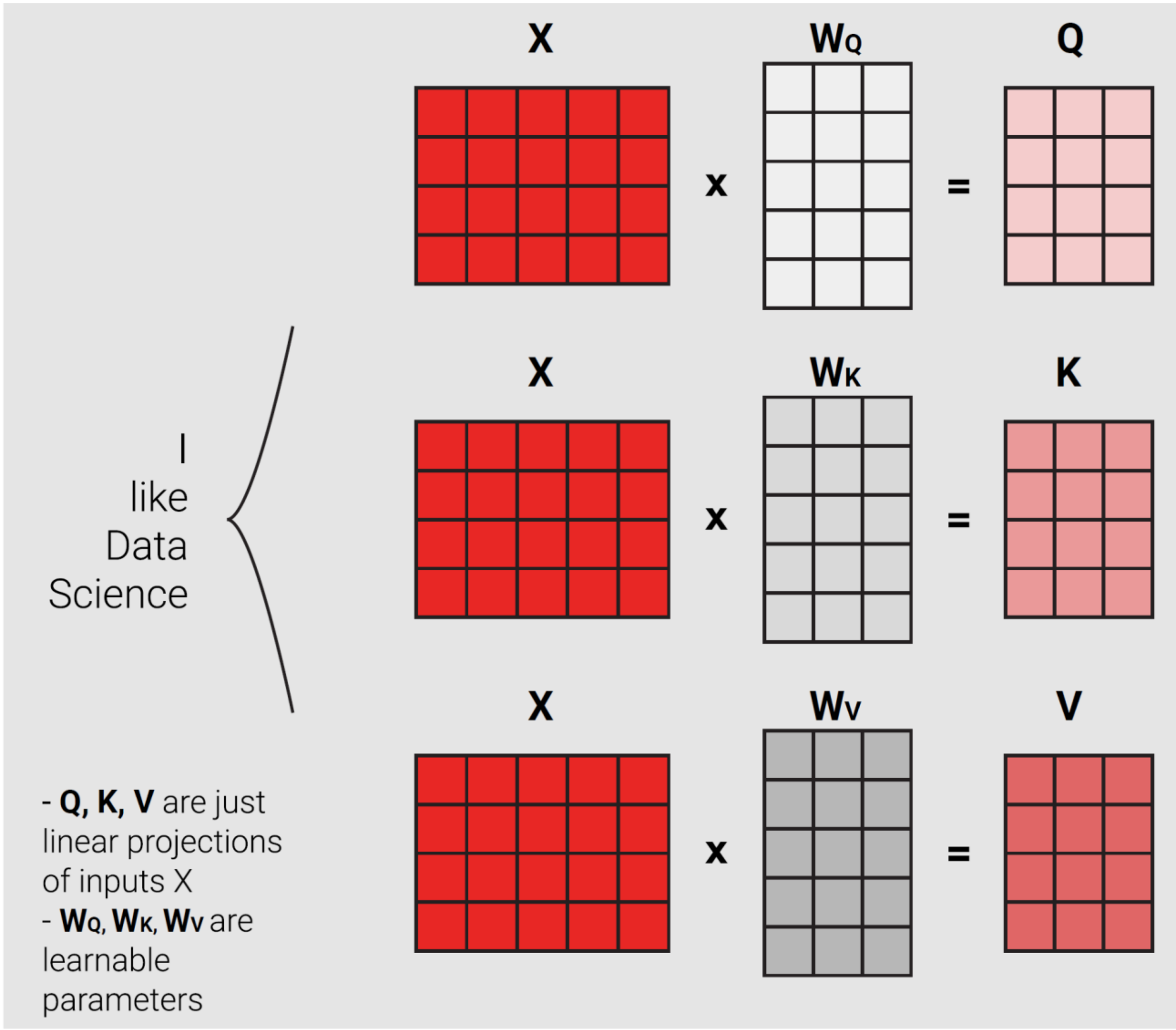
d - dimension of queries, keys, values

Example:

n = 4

***d*m** = 5

d = 3



- **Q, K, V** are just linear projections of inputs X
- **W**Q, **W**K, **W**V are learnable parameters

In summary 3/3

Self-attention

- scaled dot-product attention;

- with larger d the variance of dot products increases, softmax gets more peaked, its gradient gets smaller;

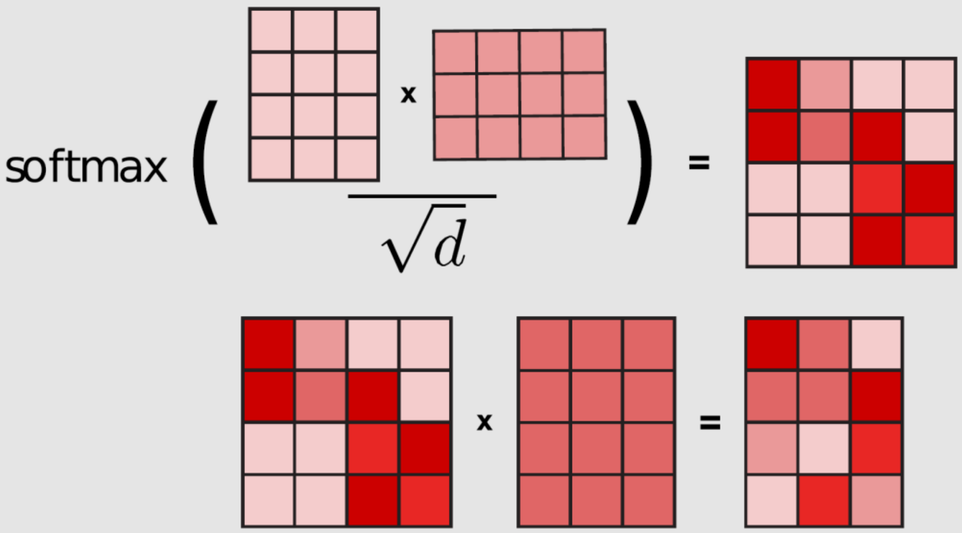
- dividing by $\sqrt{d}$ mitigates this effect.

$$sim_{ij} = \frac{Q_i^T K_j}{\sqrt{d}} \qquad \alpha_{ij} = \frac{exp(sim_{ij})}{\sum_{k=1}^{n} exp(sim_{ik})} \qquad Y_i = \sum_{j=1}^{n} \alpha_{ij} V_j$$

Transformer attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

matrix formulation



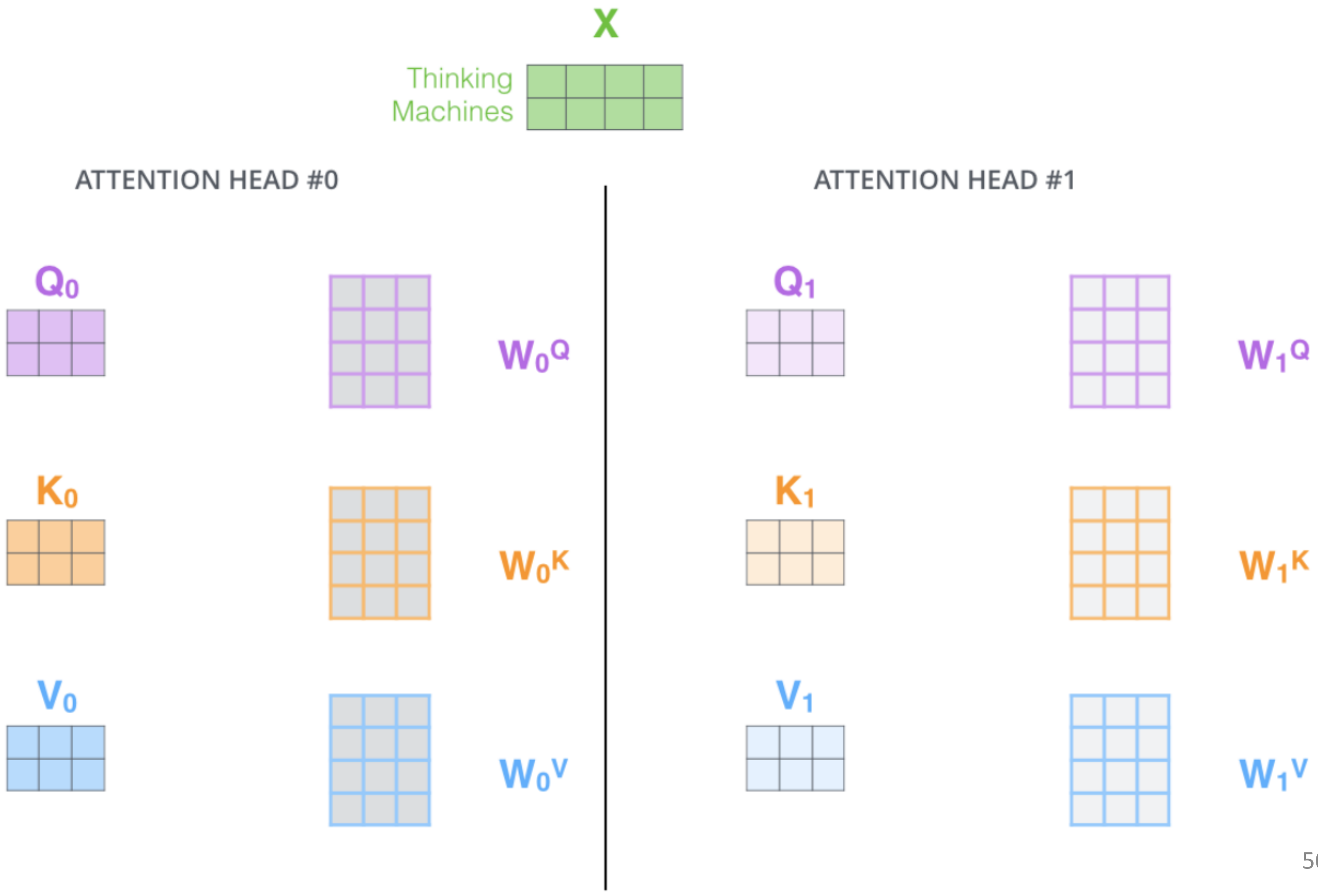
attention weights

output

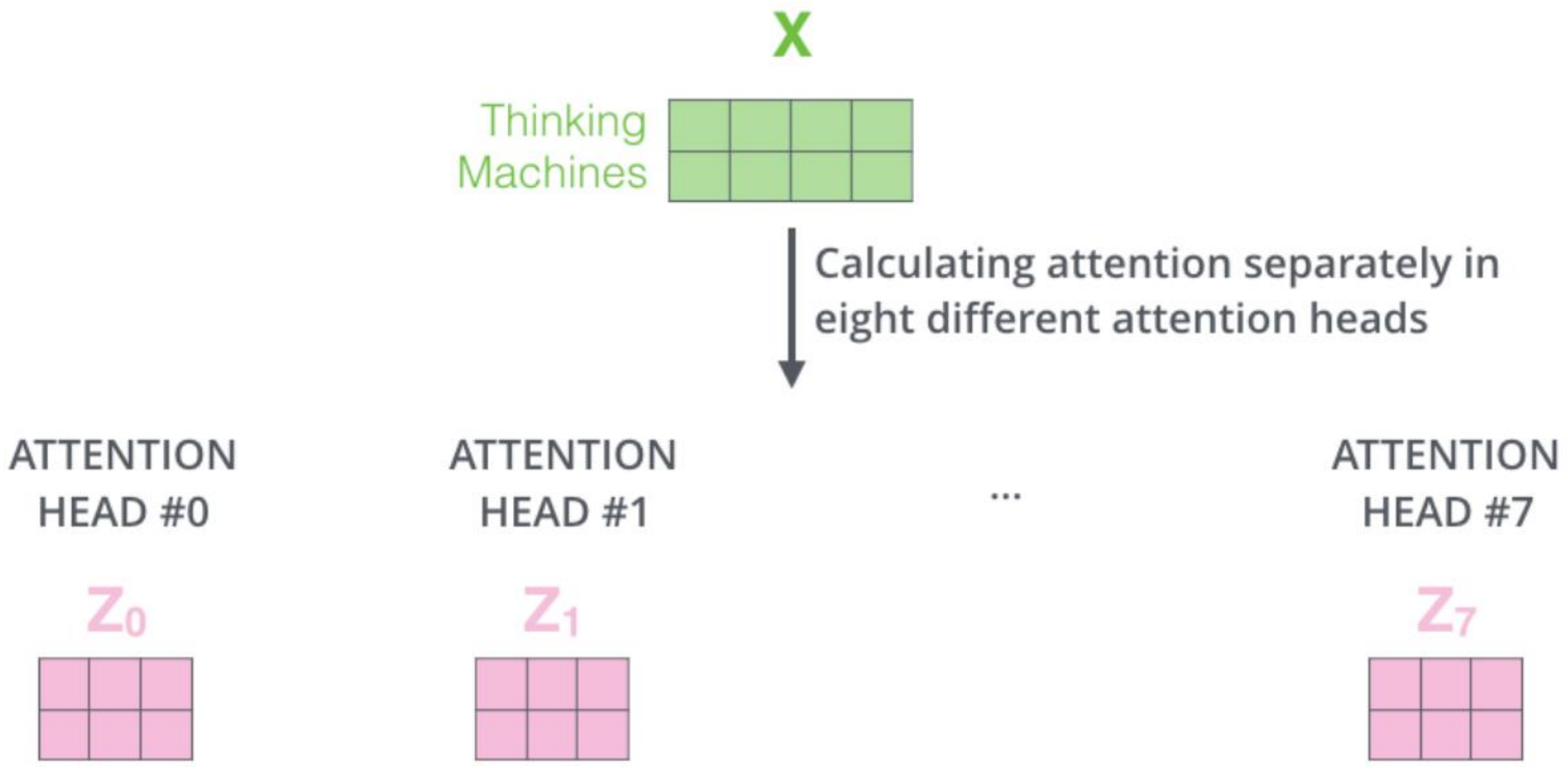
Multi-headed attention

- Self-attention layer is replicated several times, called “multi-headed” attention.
- This expands the model’s ability to focus on different positions. E.g., one attention head might be dominated by the the actual word, but other heads might reveal other important information
- E.g., in translating a sentence like “The animal didn’t cross the street because it was too tired”, we would want to know which word “it” refers to.
- Multi-head attention layer can cover multiple “representation subspaces”
- I.e., we have multiple sets of Query/Key/Value weight matrices (original Transformer uses 8 attention heads)
- Each attention head is randomly initialized. After training, each set is used to project the input embeddings (or vectors from lower encoders/decoders) into a different representation subspace.

Example: two attention heads

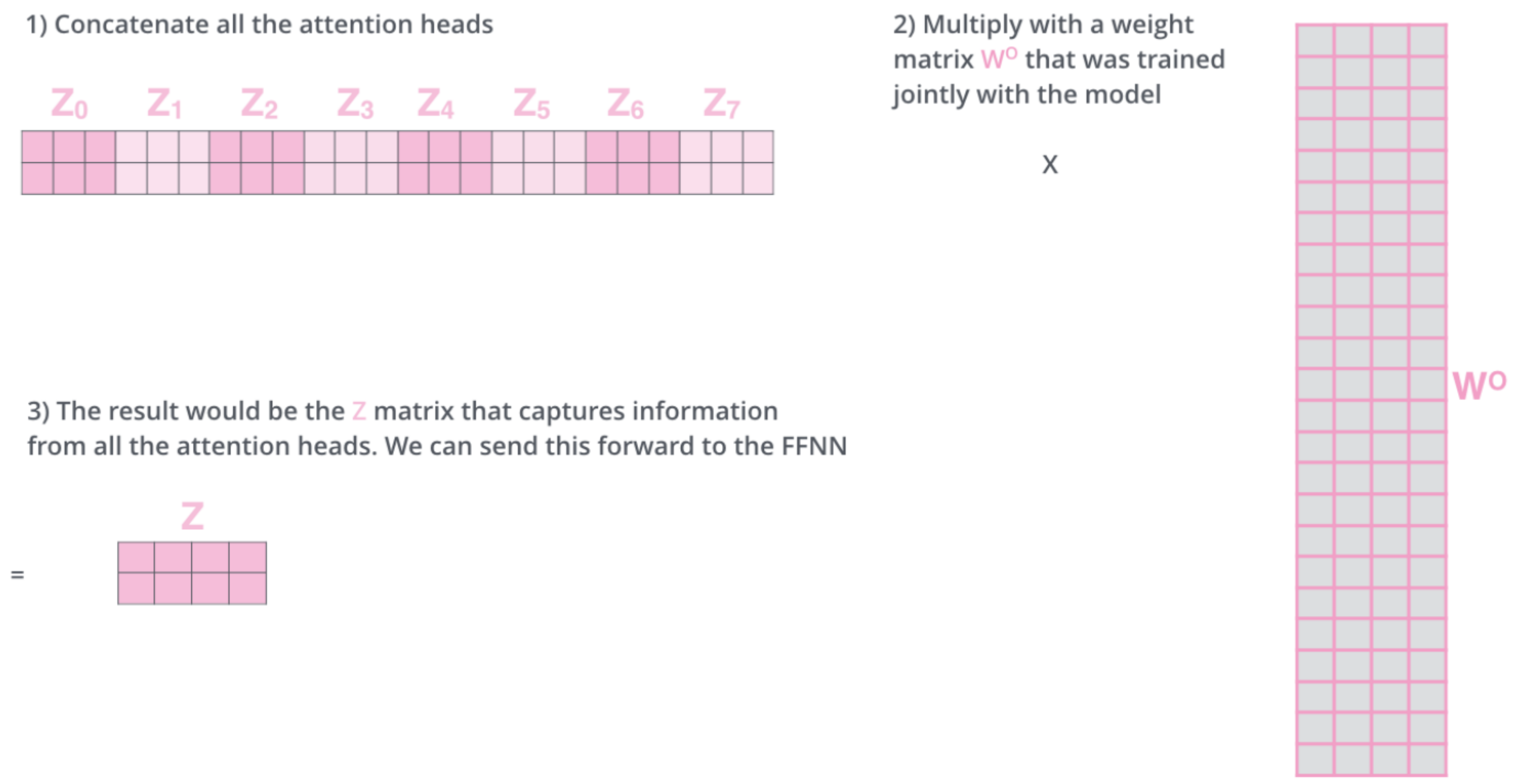


Example: 8 att. heads



- What to do with 8 Z matrices, the feed-forward layer is expecting a single matrix (one vector for each word). We need to condense all attention heads into one matrix.

Condensation of attention heads

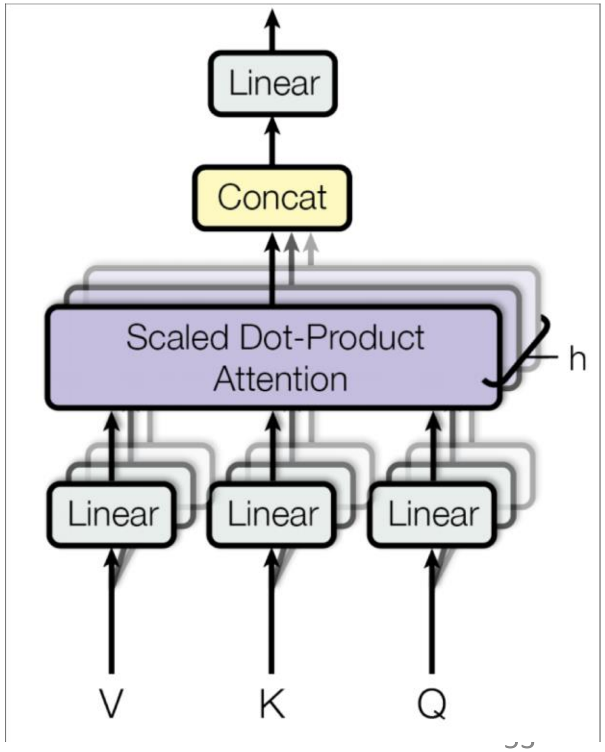

1) Concatenate all the attention heads

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

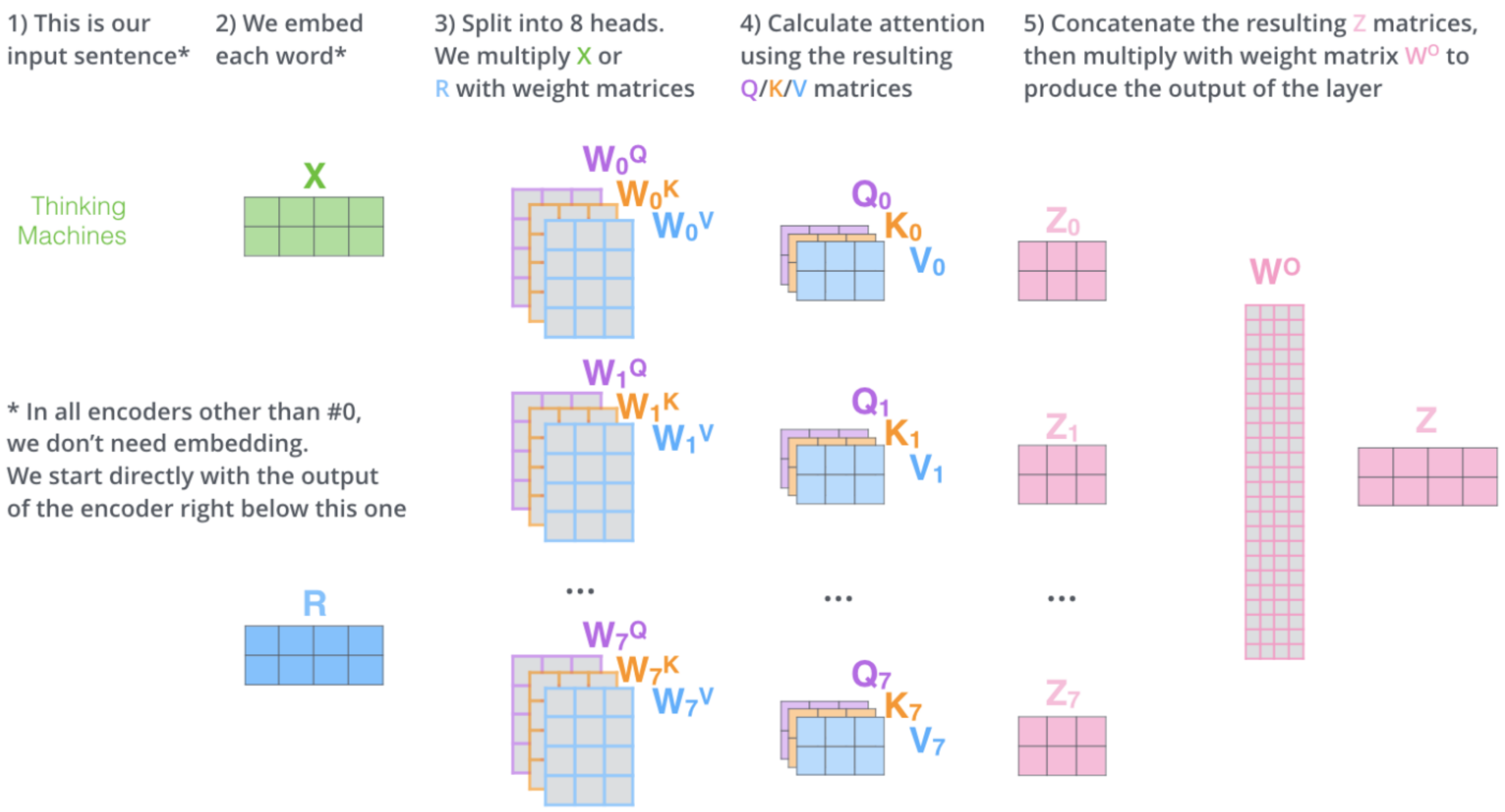
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

Computing multi-head attention

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_\mathrm{h})W^O$$
$$\text{where } \mathrm{head}_\mathrm{i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



Summary of self-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
X
R
W_0^Q W_0^K W_0^V
W_1^Q W_1^K W_1^V
W_7^Q W_7^K W_7^V
Q_0 K_0 V_0
Q_1 K_1 V_1
Q_7 K_7 V_7
Z_0
Z_1
Z_7
W^O
Z
...
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Illustration of self-attention: 1 head

- encoder #5 (the top encoder in the stack)
- As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

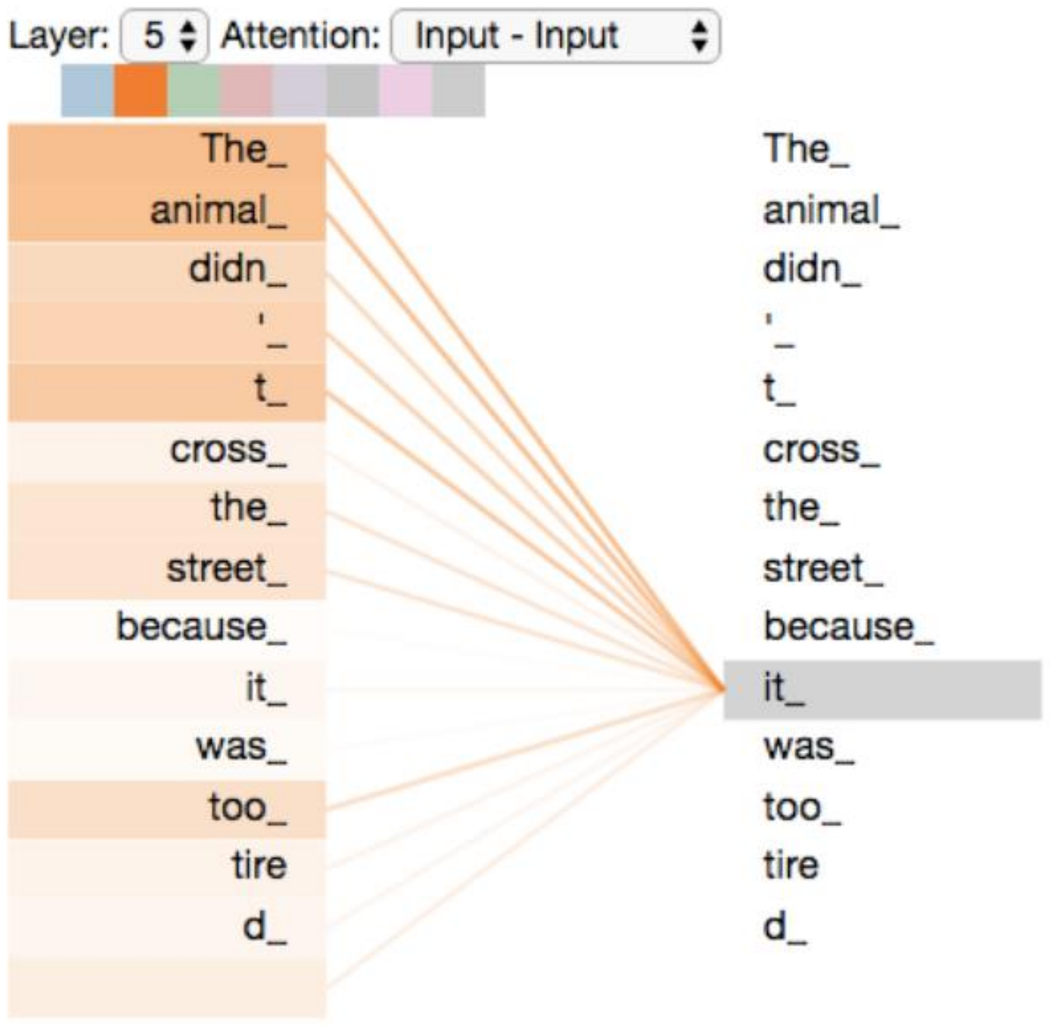
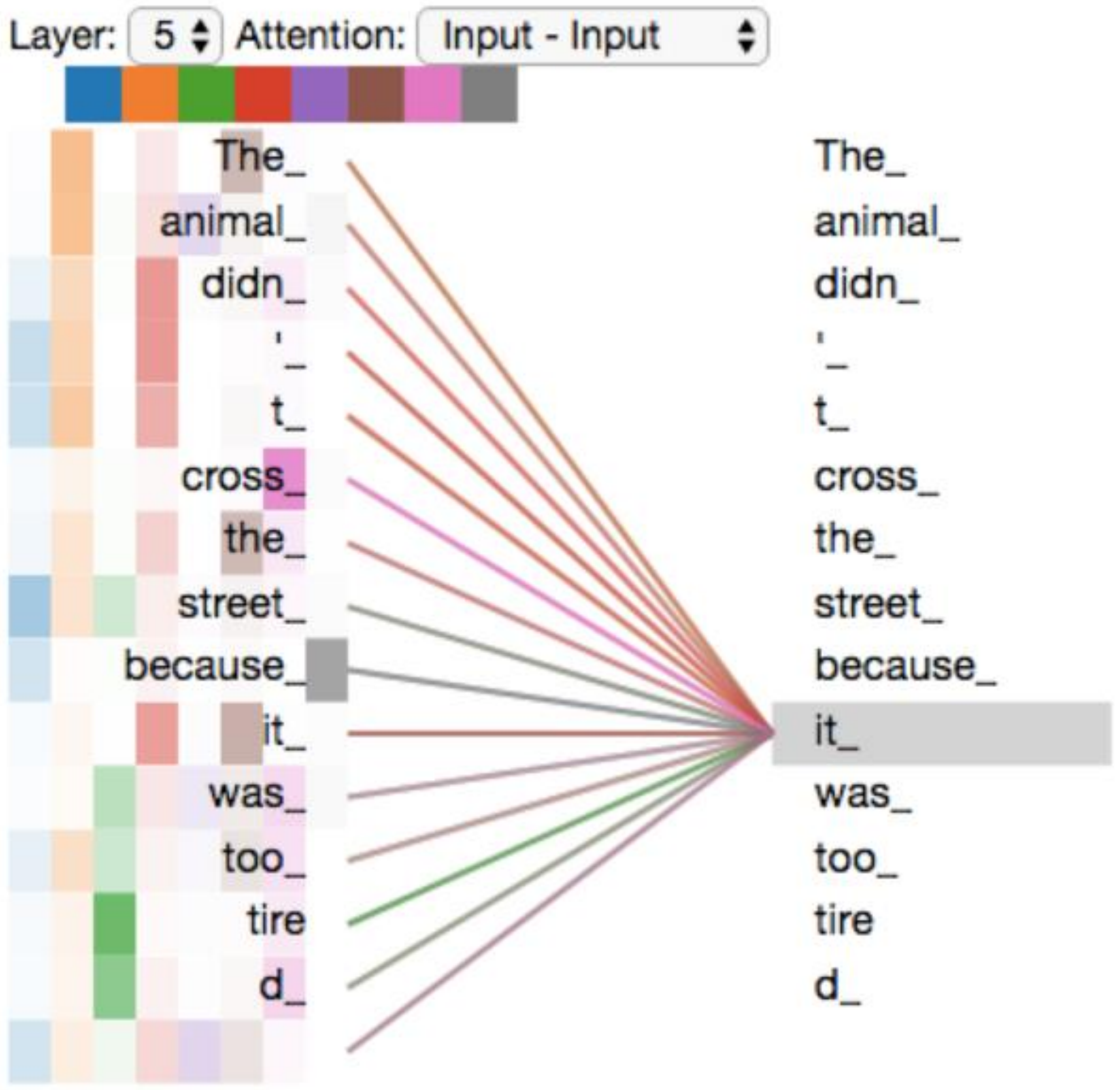


Illustration of self-attention: all heads

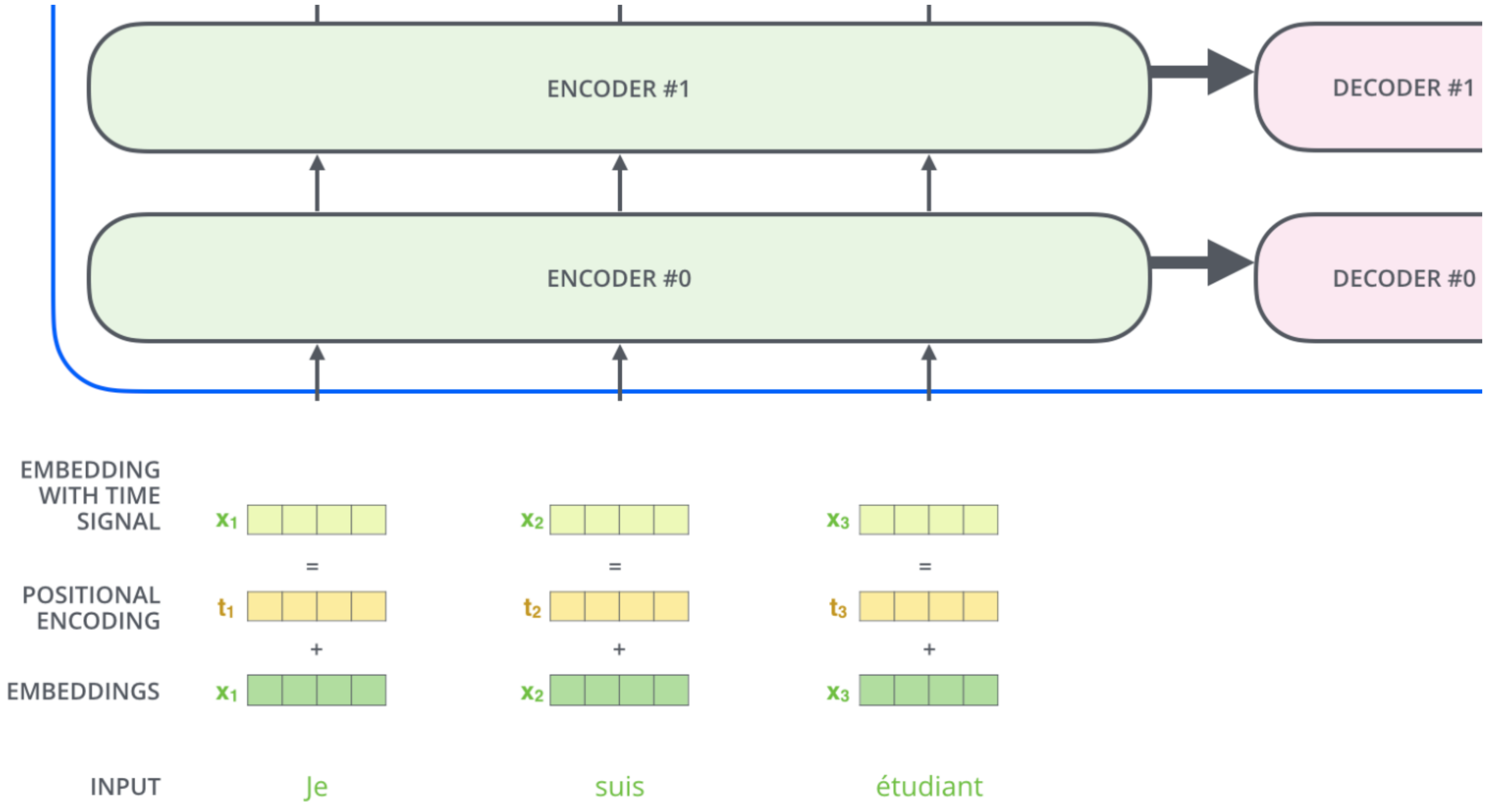
- all the attention heads in one picture are harder to interpret



Representing the order of the sequence using positional encoding

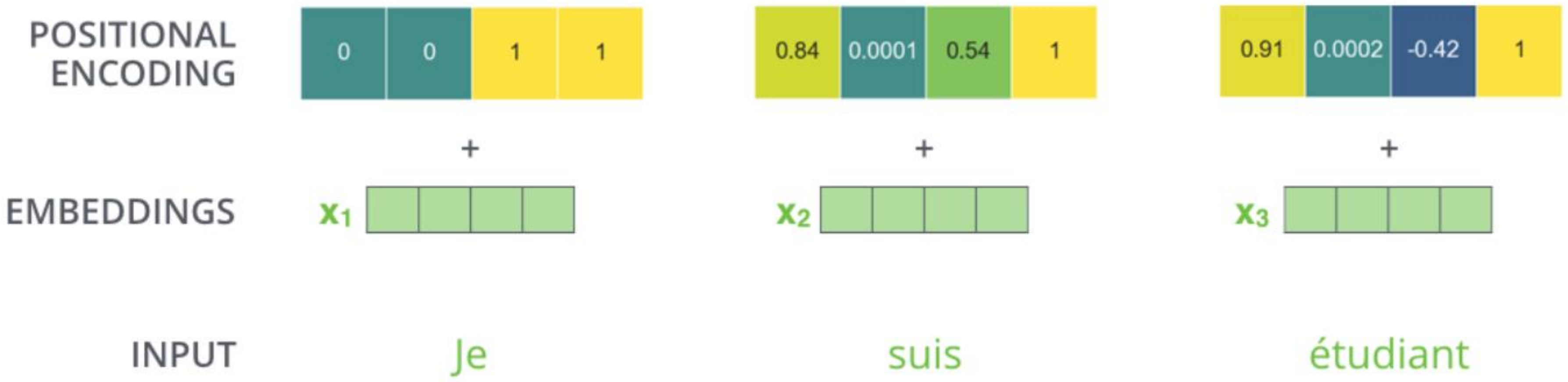
- Order of the sequence is important but it is lost with the described transformation, therefore
- The transformer adds a position vector to each input embedding.
- These vectors follow a specific pattern that the model learns, which helps it determine the position of each word, or the distance between different words in the sequence.
- Adding these values to the embeddings provides meaningful distances between the embedding vectors once they’re projected into Q/K/V vectors and during the dot-product attention.

Adding position encoding



Example: encoding position

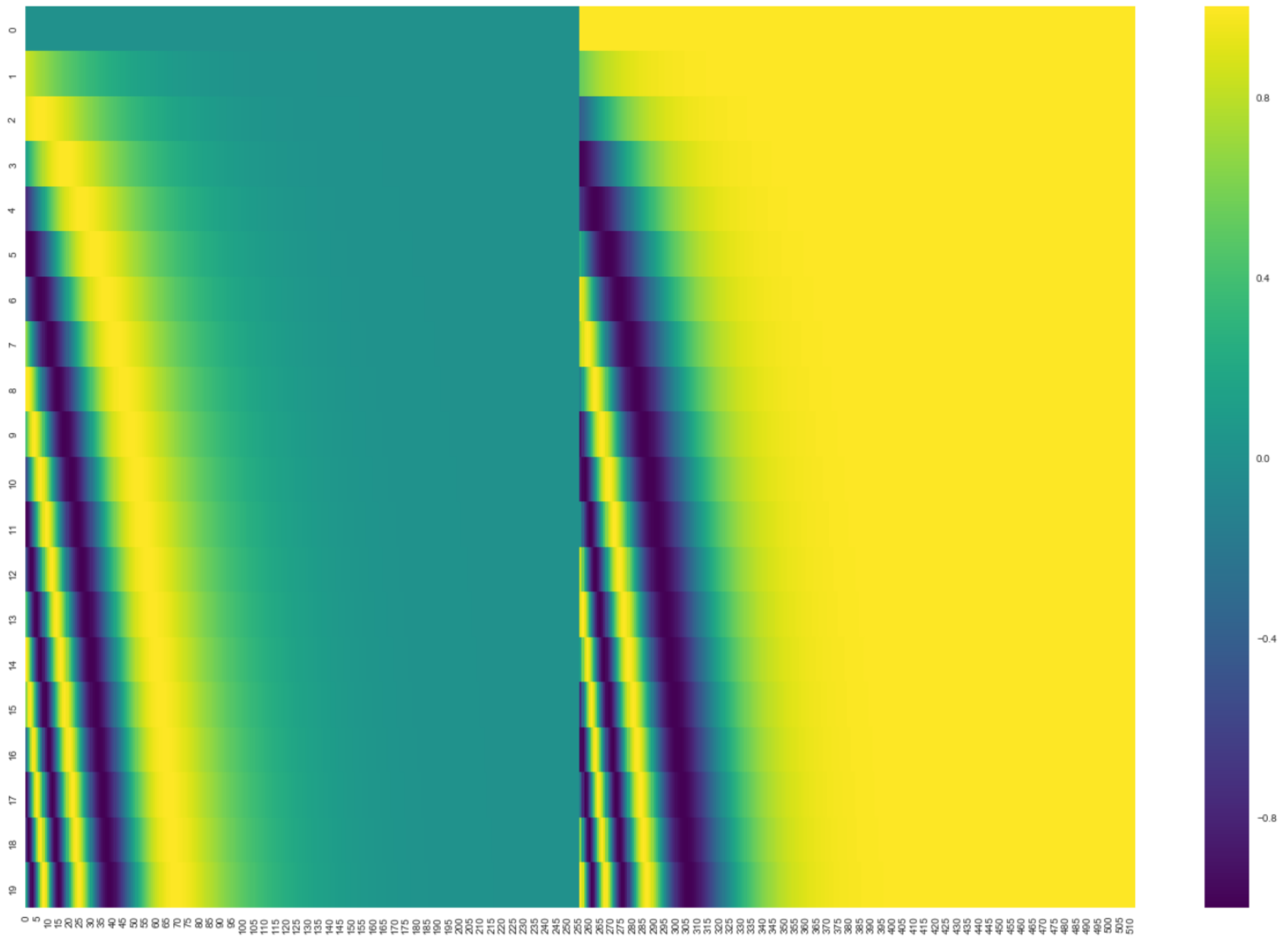
- the values of positional encoding vectors follow a specific pattern.



Patterns of possitional encodings

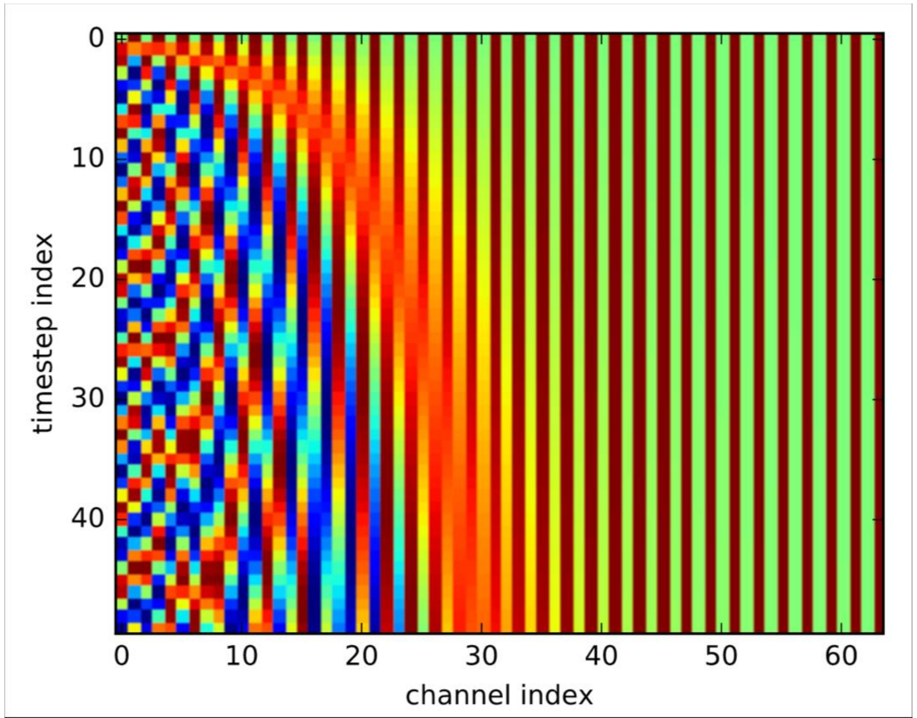
- many different possibilities how to generate a pattern
- next slide contains an example of positional encoding for 20 words (rows) with an embedding size of 512 (columns).
- the values of the left half are generated by one function (which uses sine), and the right half is generated by another function (which uses cosine). They're then concatenated to form each of the positional encoding vectors

Example of positional encoding



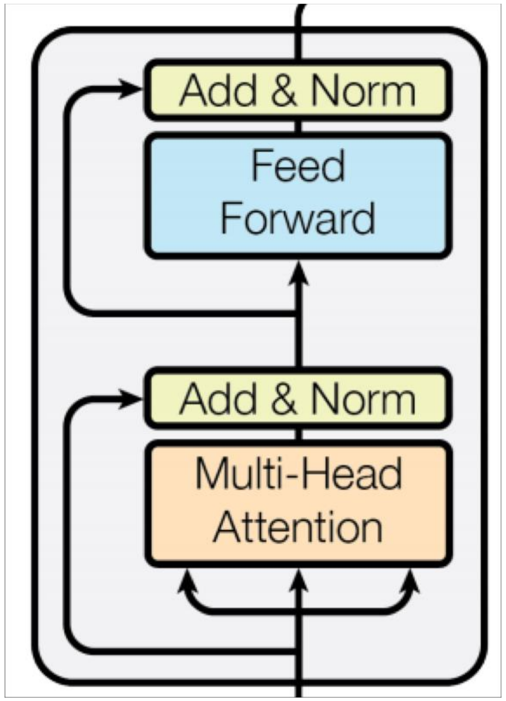
Another positional encoding

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



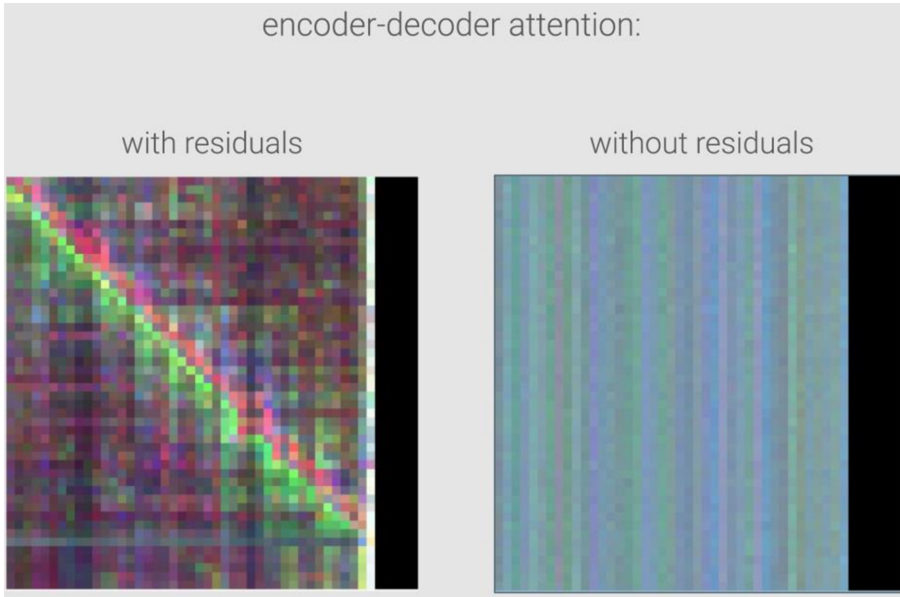
Encoder blocks

- Each block has two "sublayers"
  - Multihead attention
  - 2-layer feed-forward NNet (with ReLU)
- Each of these two steps also has residual (short-circuit) connection and LayerNorm
  - LayerNorm(x + Sublayer(x))

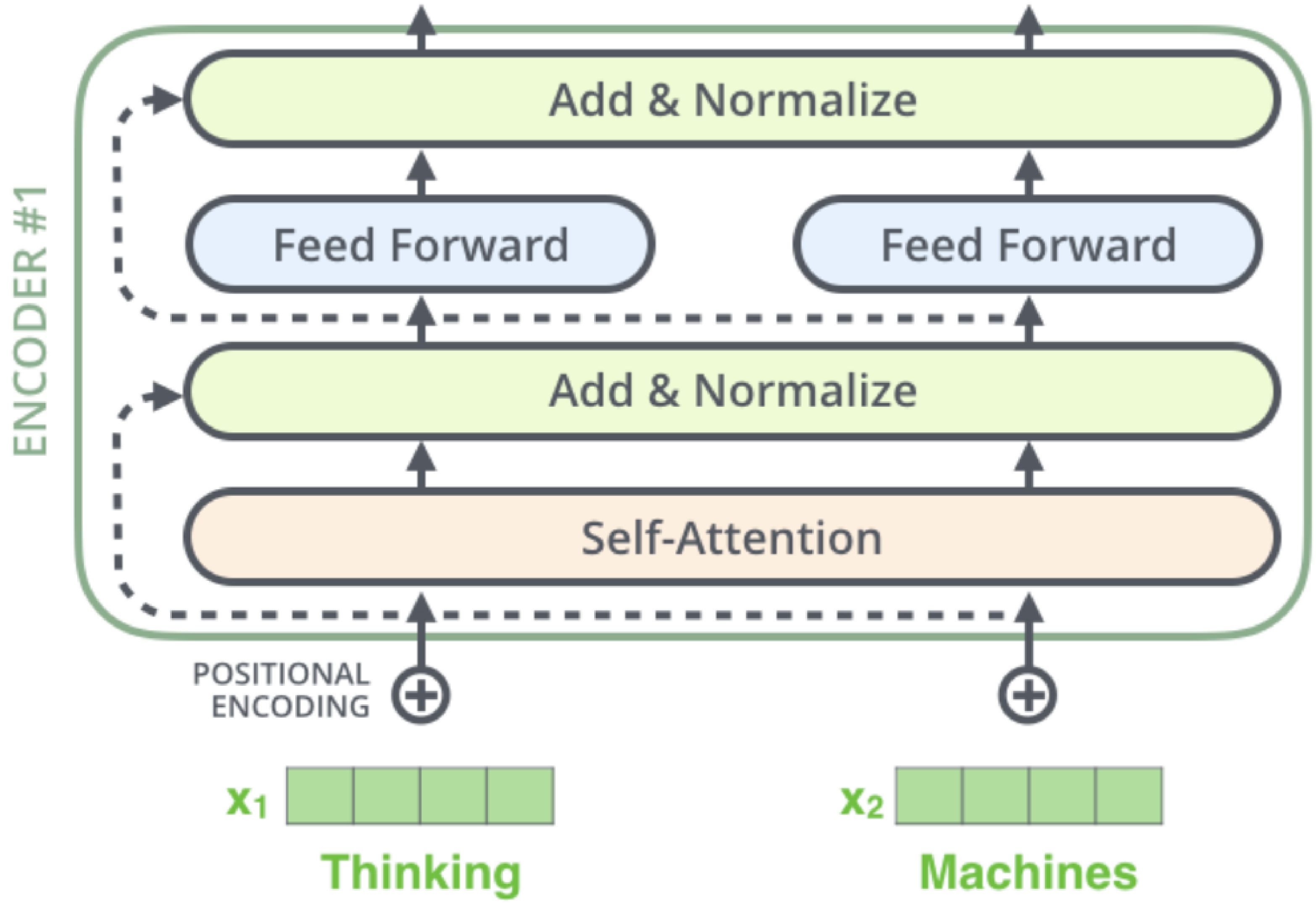


The Residual connections

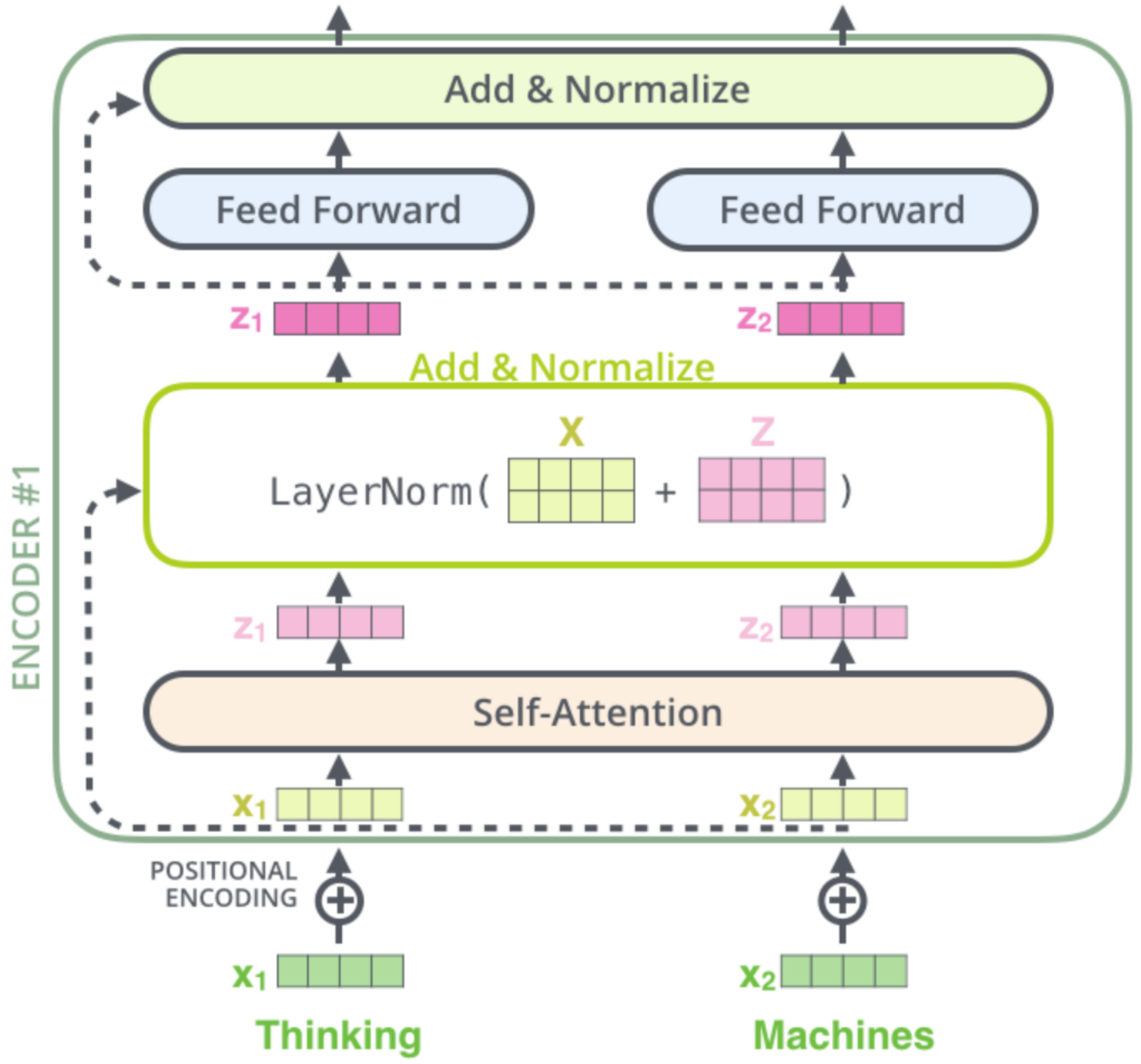
- each sub-layer of transformer (self-attention and feed-forward NN) in each encoder has a residual connection around it, and is followed by a layer-normalization step.
- the same for decoder sub-layers
- enable learning of deeper networks by improving a gradient flow
- in transformers they also maintain positional information in higher layers



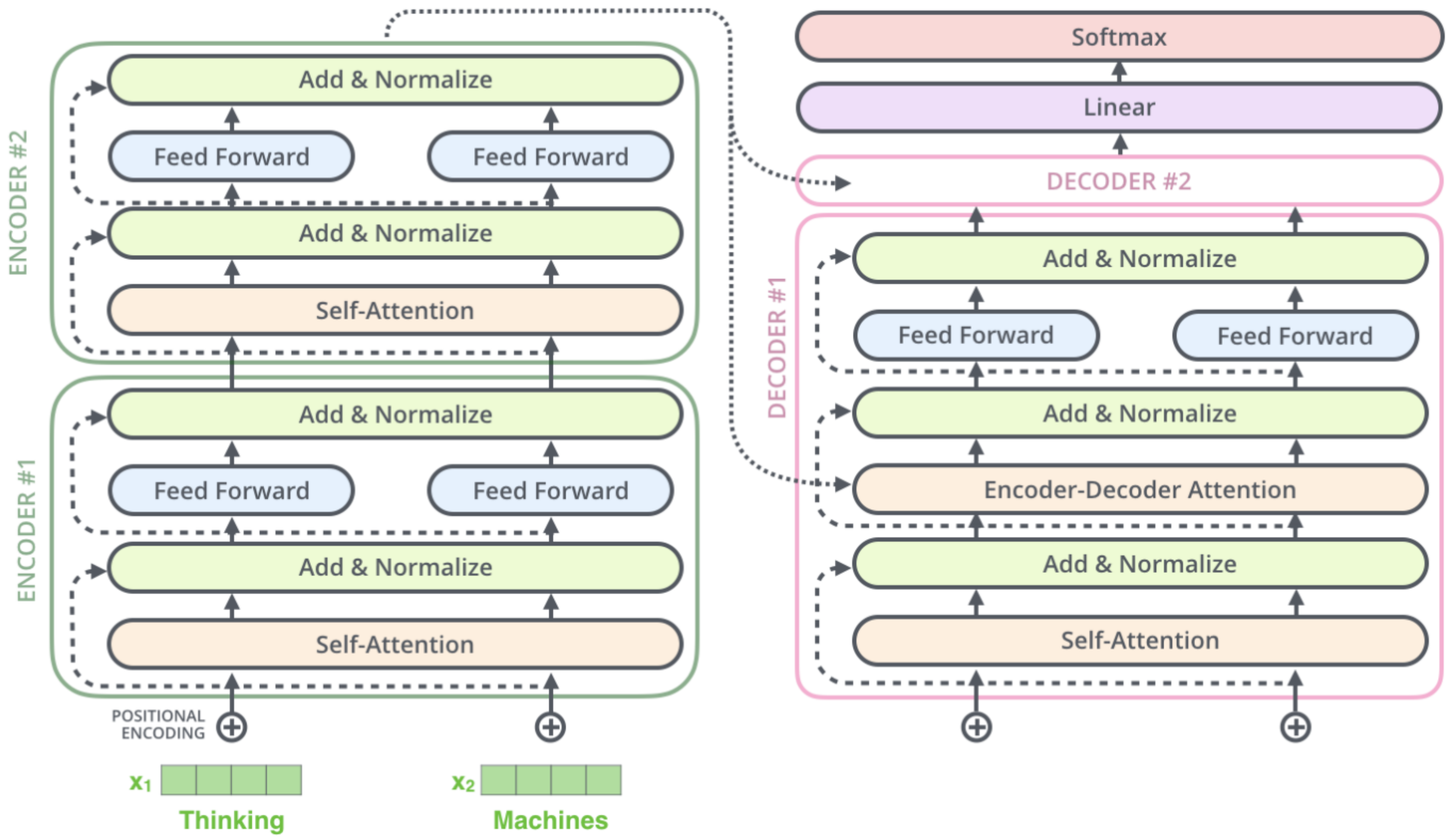
Architecture with residual connection – top level view



Architecture with residual connection – example

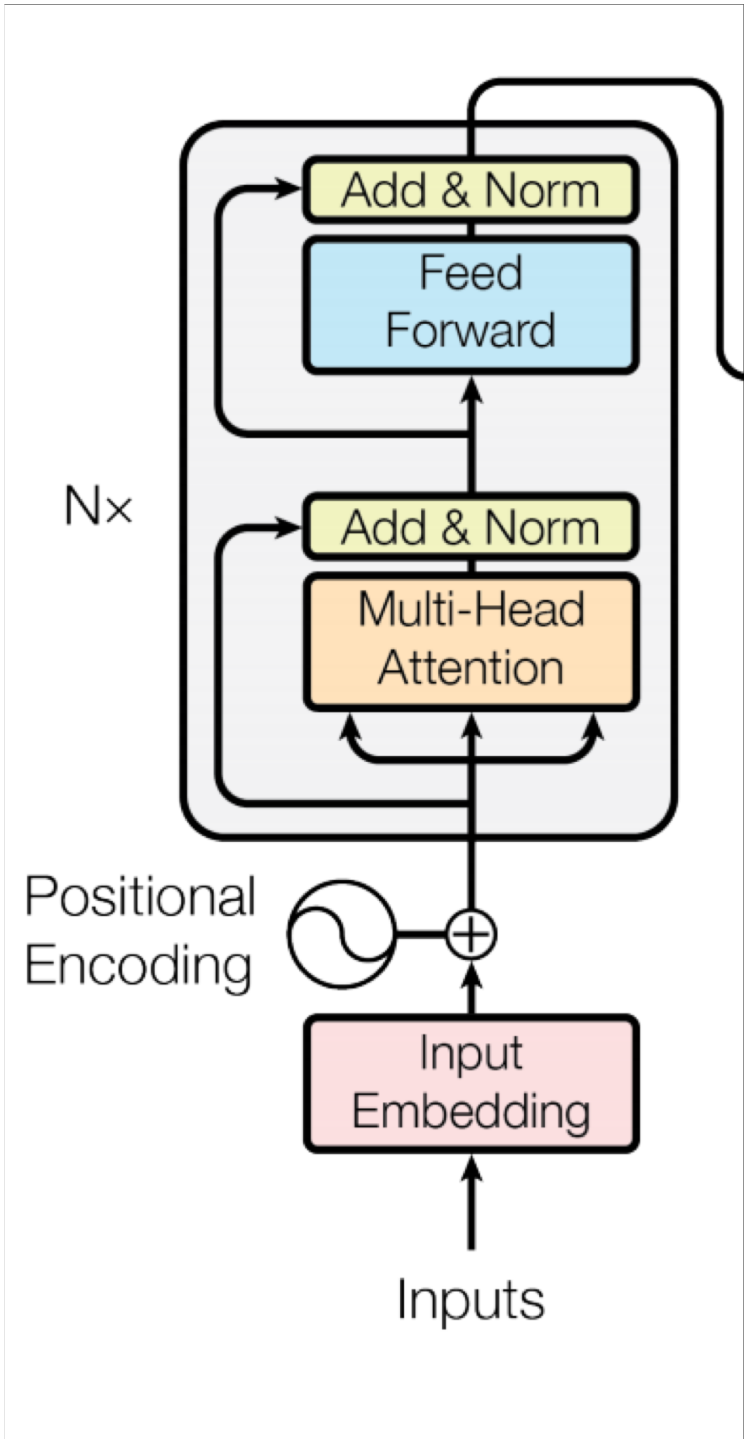


Example: 2 stacked transformer



Complete encoder

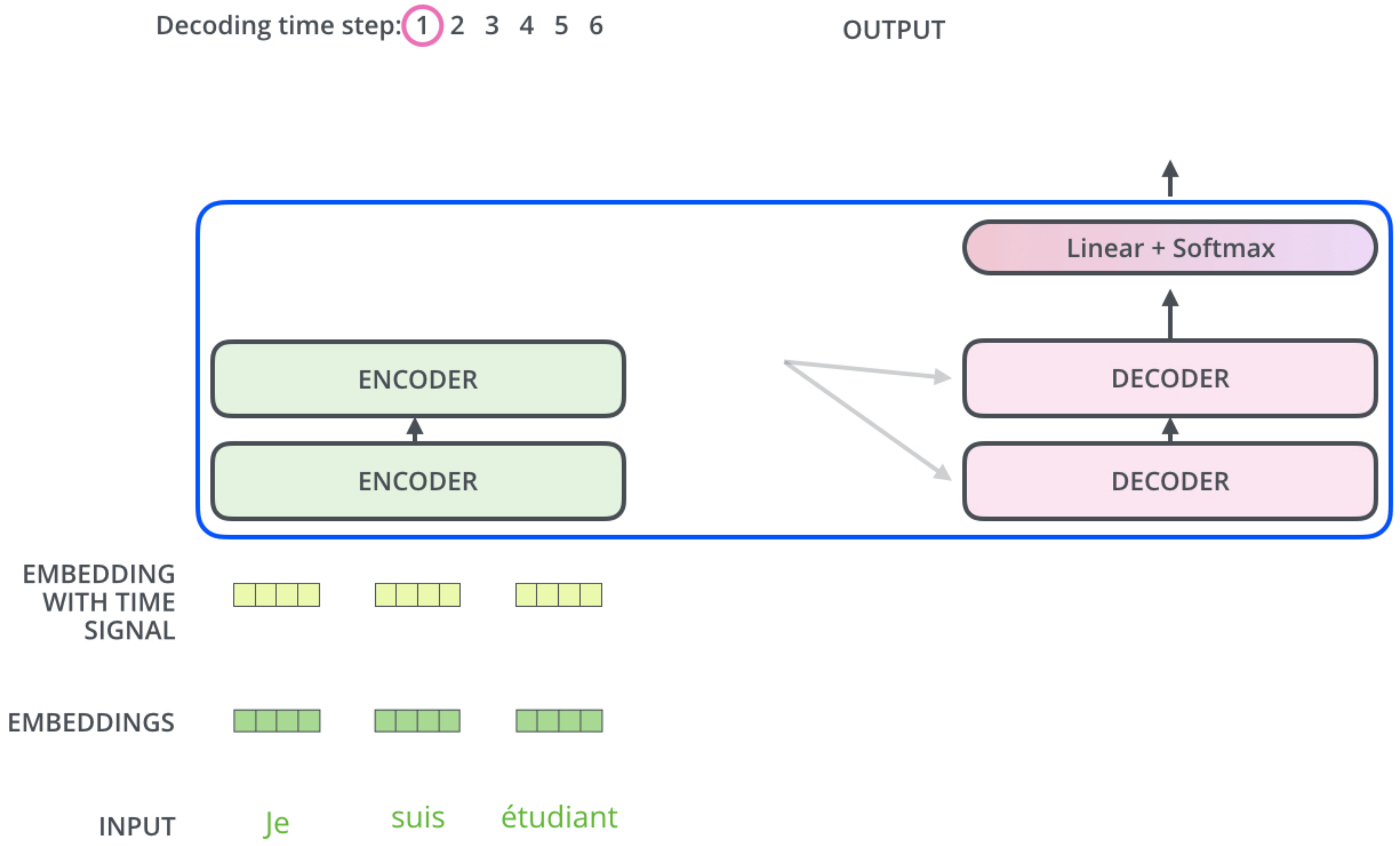
- each block is repeated several times, e.g., 6 times



Decoder

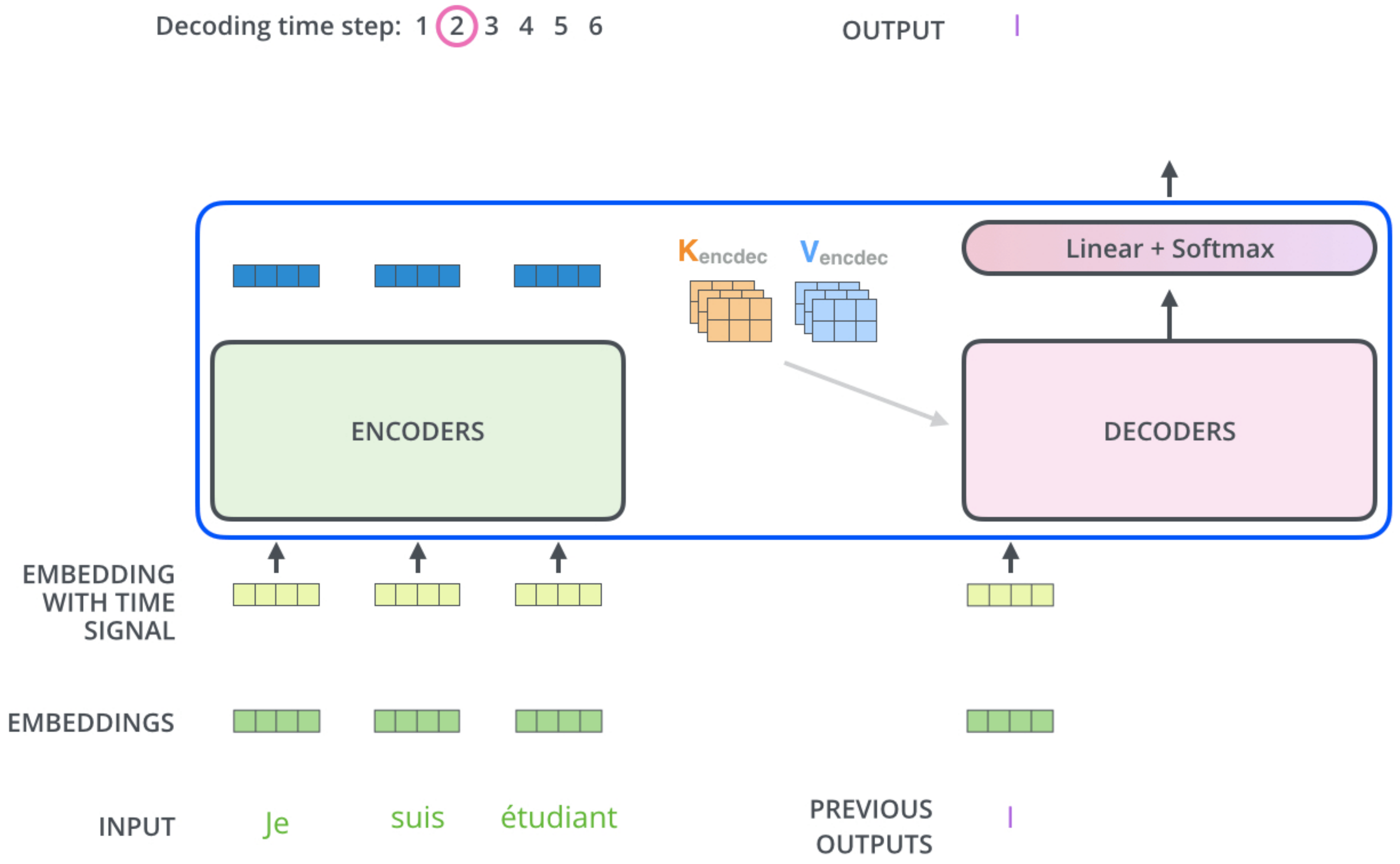
- Decoders have the same components as encoders
- An encoder start by processing the input sequence.
- The output of the top encoder is transformed into a set of attention vectors K and V.
- These are used by each decoder in its “encoder-decoder attention” layer which helps the decoder to focus on appropriate places in the input sequence.

Encoder-decoder in action 1/2



After finishing the encoding phase, we begin the decoding phase. Each step in the decoding phase outputs an element from the output sequence (the English translation sentence in this case).

Encoder-decoder in action 2/2



The steps repeat until a special symbol indicating the end of output. The output of each step is fed to the bottom decoder in the next time step. We add positional encoding to decoder inputs to indicate the position of each word.

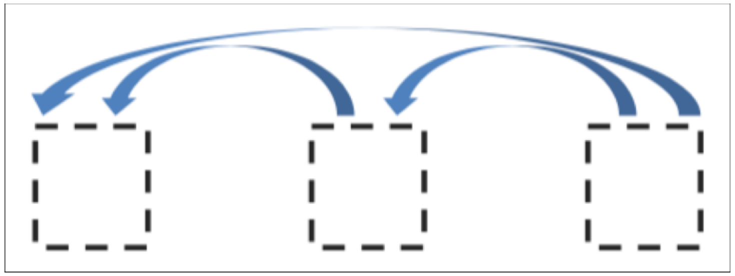
Self-attention and encoder-decoder attention in the decoder

- In the decoder, the self-attention layer is only allowed to attend to itself and earlier positions in the output sequence (to maintain the autoregressive property).
- This is done by masking future positions (setting them to -inf) before the softmax step in the self-attention calculation.

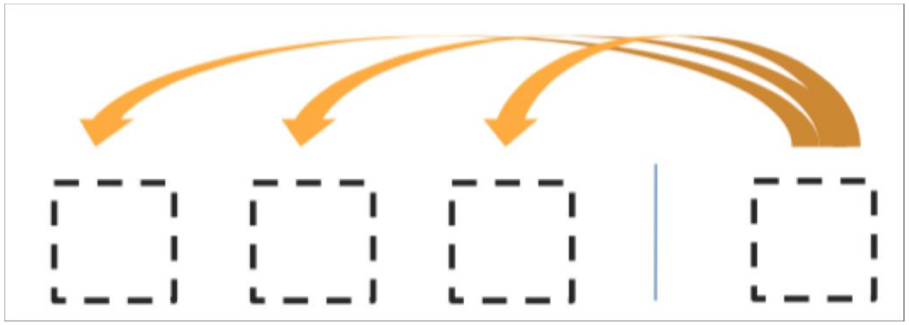
- The “Encoder-Decoder Attention” layer works just like multiheaded self-attention, except it creates its Q (queries) matrix from the layer below it, and takes the K (keys) and V (values) matrix from the output of the encoder stack.

Attentions in the decoder

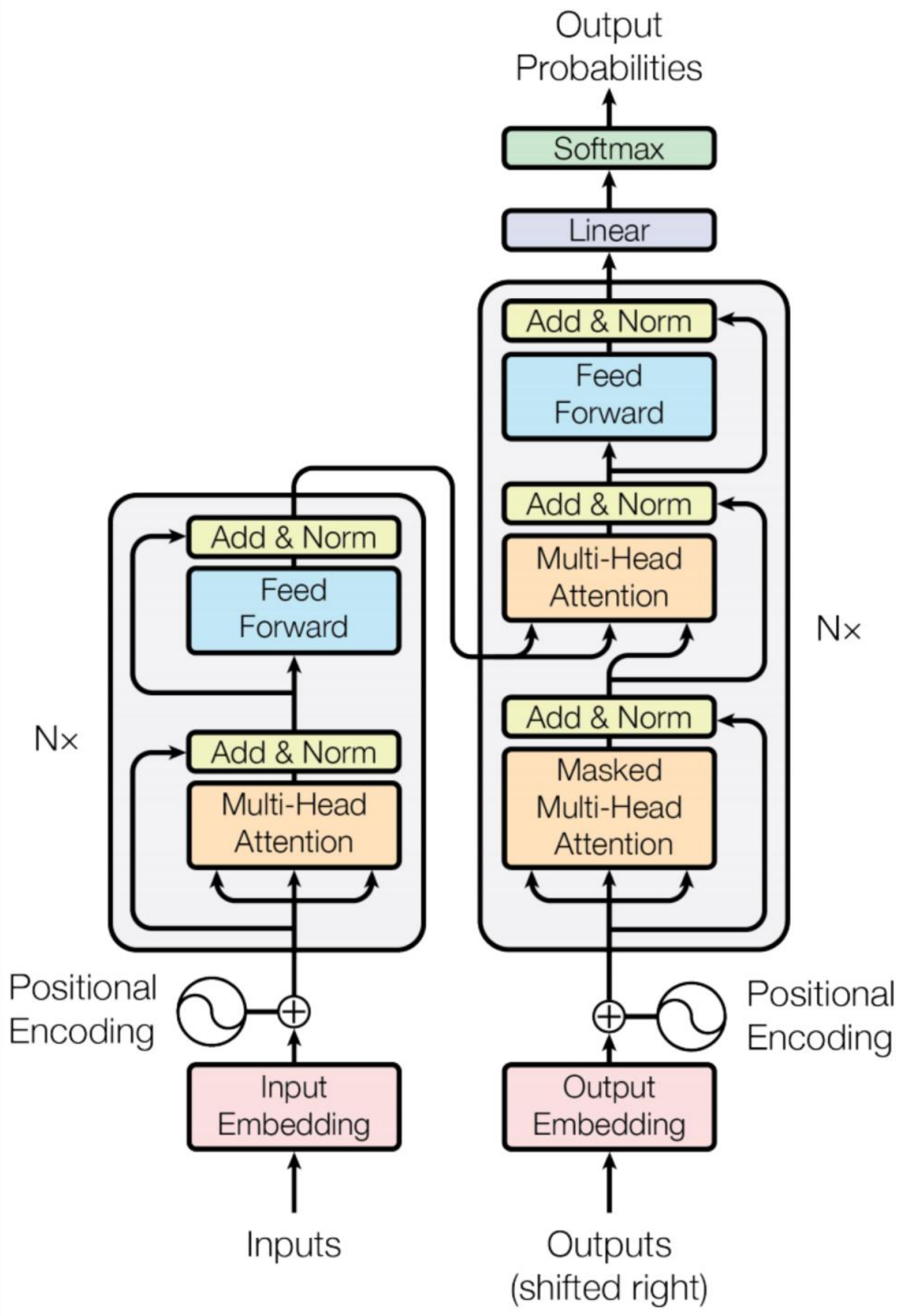
1. Masked decoder self-attention on previously generated outputs



2. Encoder-Decoder Attention, where queries come from previous decoder layer and keys and values come from output of the encoder



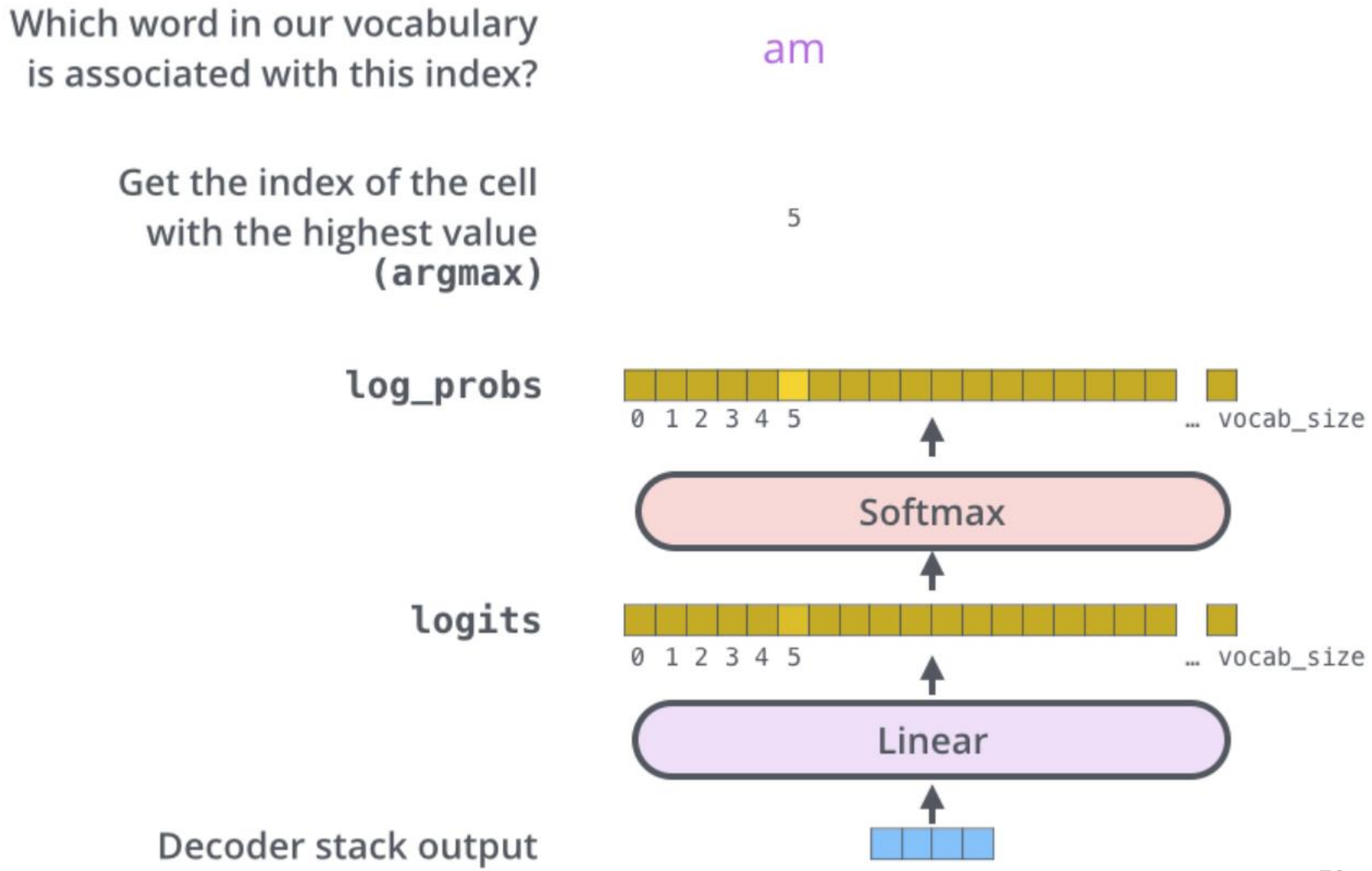
One encoder-decoder block



Final Linear and Softmax Layer

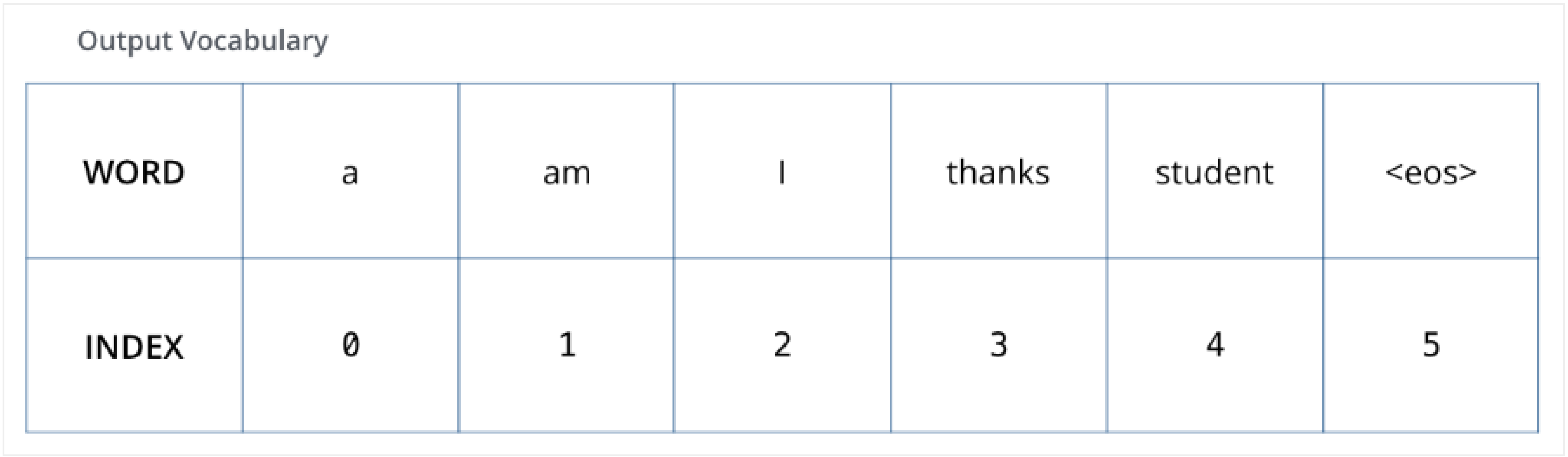
- The decoder stack outputs a vector of floats.
- The final linear layer which is followed by a softmax layer turns them into words.
- The Linear layer is a simple fully connected neural network that projects the vector produced by the stack of decoders, into a much larger vector called a logits vector (probability scores for each word).
- Example: the model knows 10,000 unique English words ("output vocabulary") that it's learned from its training dataset. Therefore, the logits vector is 10,000 cells wide – each cell corresponding to the score of a unique word.
- The softmax layer turns those scores into probabilities (all positive, between 0 and 1, sum to 1.0).
  The cell with the highest probability is chosen, and the word associated with it is produced as the output for this time step.

Producing the output words


Which word in our vocabulary is associated with this index?
am
Get the index of the cell with the highest value (argmax)
5
log_probs
0 1 2 3 4 5 … vocab_size
Softmax
logits
0 1 2 3 4 5 … vocab_size
Linear
Decoder stack output

Training the transformer

- During training, an untrained model would go through the exact same forward pass. But since we are training it on a labeled training dataset, we can compare its output with the actual correct output.
- For illustration, let's assume that our output vocabulary only contains six words(a, am, i, thanks, student, <eos>)
- The input is typically in the order of 10^4 (e.g., 30 000)

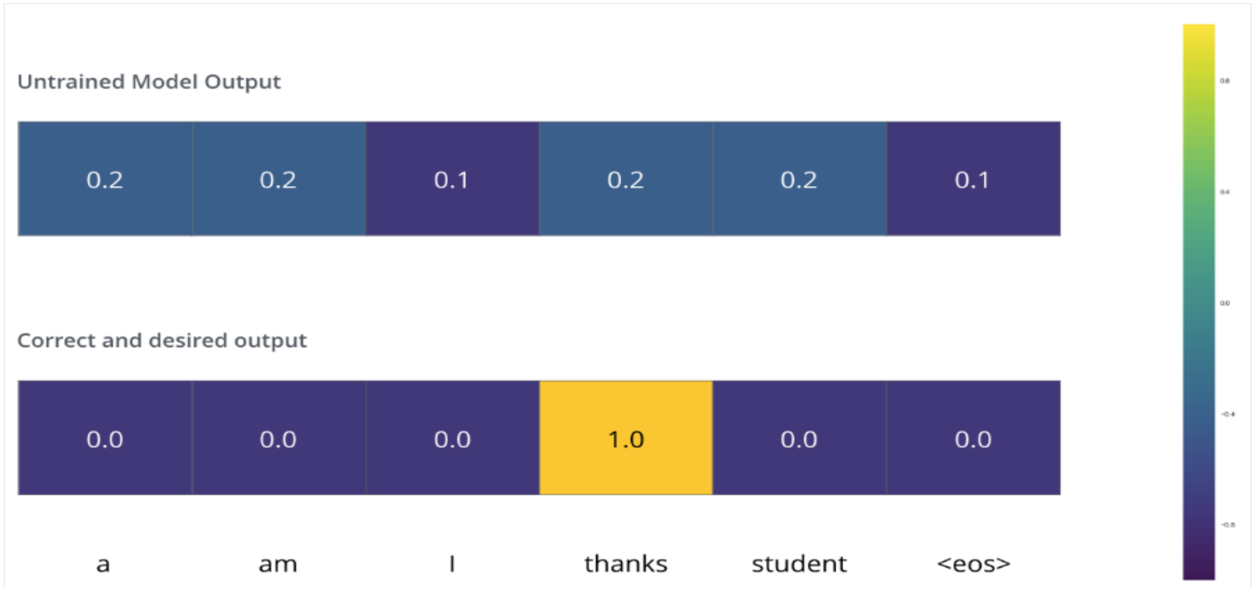


Output Vocabulary

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

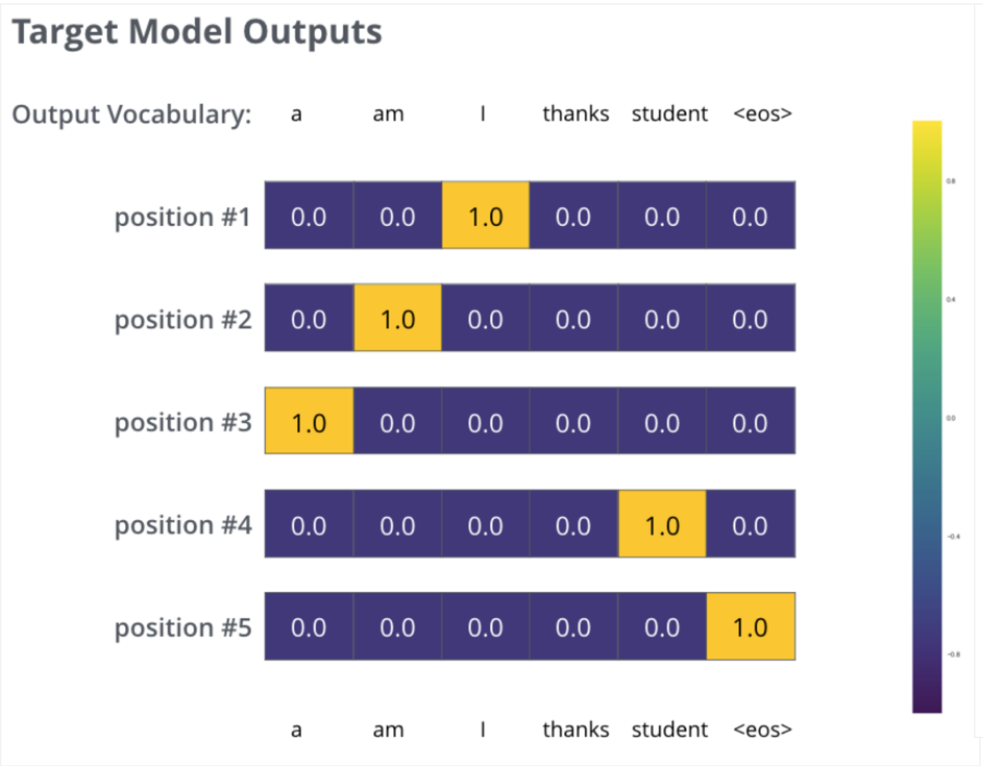
The Loss Function

- evaluates the difference between the true output and the returned output
- transformer typically uses cross-entropy or Kullback–Leibler divergence.
- both true and returned output are 1-hot encoded, e.g.,



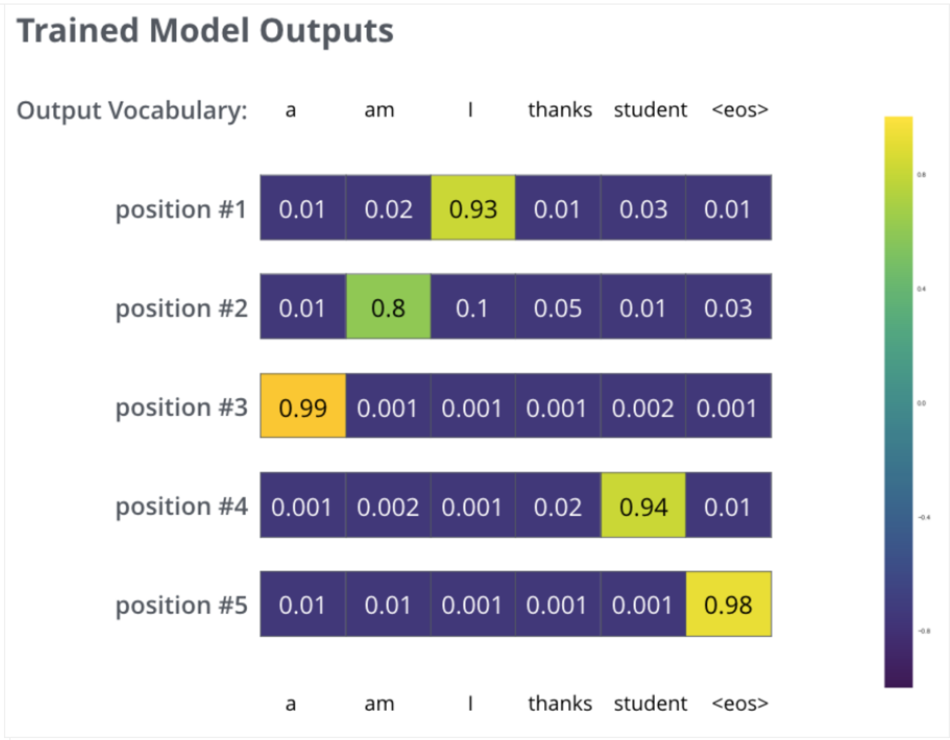
Loss evaluation for sequences

- loss function has to be evaluated for the whole sentence, not just a single word
- transformers use greedy decoding or beam search



Target Model Outputs

Output Vocabulary:	a	am	I	thanks	student	<eos>
position #1	0.0	0.0	1.0	0.0	0.0	0.0
position #2	0.0	1.0	0.0	0.0	0.0	0.0
position #3	1.0	0.0	0.0	0.0	0.0	0.0
position #4	0.0	0.0	0.0	0.0	1.0	0.0
position #5	0.0	0.0	0.0	0.0	0.0	1.0
	a	am	I	thanks	student	<eos>



Trained Model Outputs

Output Vocabulary:	a	am	I	thanks	student	<eos>
position #1	0.01	0.02	0.93	0.01	0.03	0.01
position #2	0.01	0.8	0.1	0.05	0.01	0.03
position #3	0.99	0.001	0.001	0.001	0.002	0.001
position #4	0.001	0.002	0.001	0.02	0.94	0.01
position #5	0.01	0.01	0.001	0.001	0.001	0.98
	a	am	I	thanks	student	<eos>

Animated workings of transformer

Transformer tricks

- Byte-pair encodings for input tokens
- Checkpoint averaging
- ADAM optimizer with learning rate changes
- Dropout during training at every layer just before adding residual
- Label smoothing
- Auto-regressive decoding with beam search and length penalties
- Use of transformers is spreading but they are hard to optimize and unlike LSTMs don't usually just work out of the box and they don't play well yet with other building blocks on tasks.

BERT

- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
- State-of-the-art pretained LM based on transformer architecture (only the encoder part)
- Idea:
- use large unlabeled corpora and an auxiliary task to pretrain a model for general language representation
- fine-tune the model on a (possibly small) dataset for a specific downstream task
- presentation based on slides from Jacob Devlin and Jay Alammar

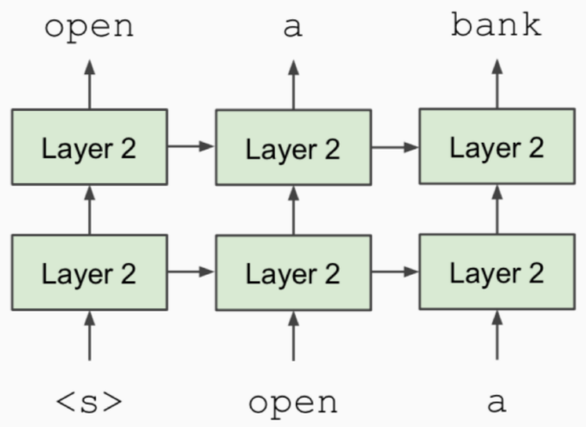
Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Volume 1, pp. 4171-4186.

BERT: motivation 1/3

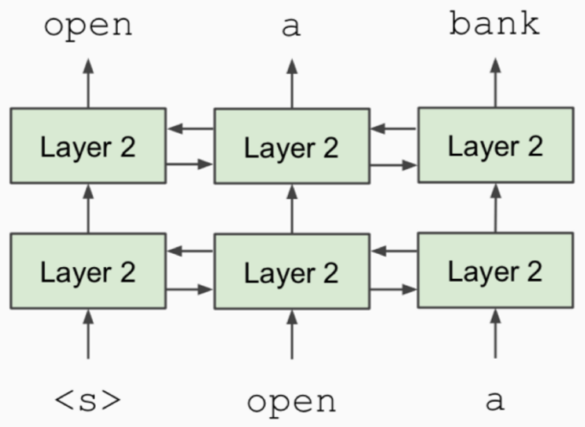
- **Problem**: Language models only use the left or right context, but language understanding is bidirectional.
- Why are LMs unidirectional?
  - Reason 1: Directionality is needed to generate a well-formed probability distribution.
    - We don’t care about this.
  - Reason 2: Words can “see themselves” in a bidirectional encoder.

BERT: motivation 2/3

Unidirectional context
Build representation incrementally



Bidirectional context
Words can "see themselves"



BERT: motivation 3/3

- **Solution**: Mask out *k*% of the input words, and then predict the masked words
- BERT uses *k* = 15%

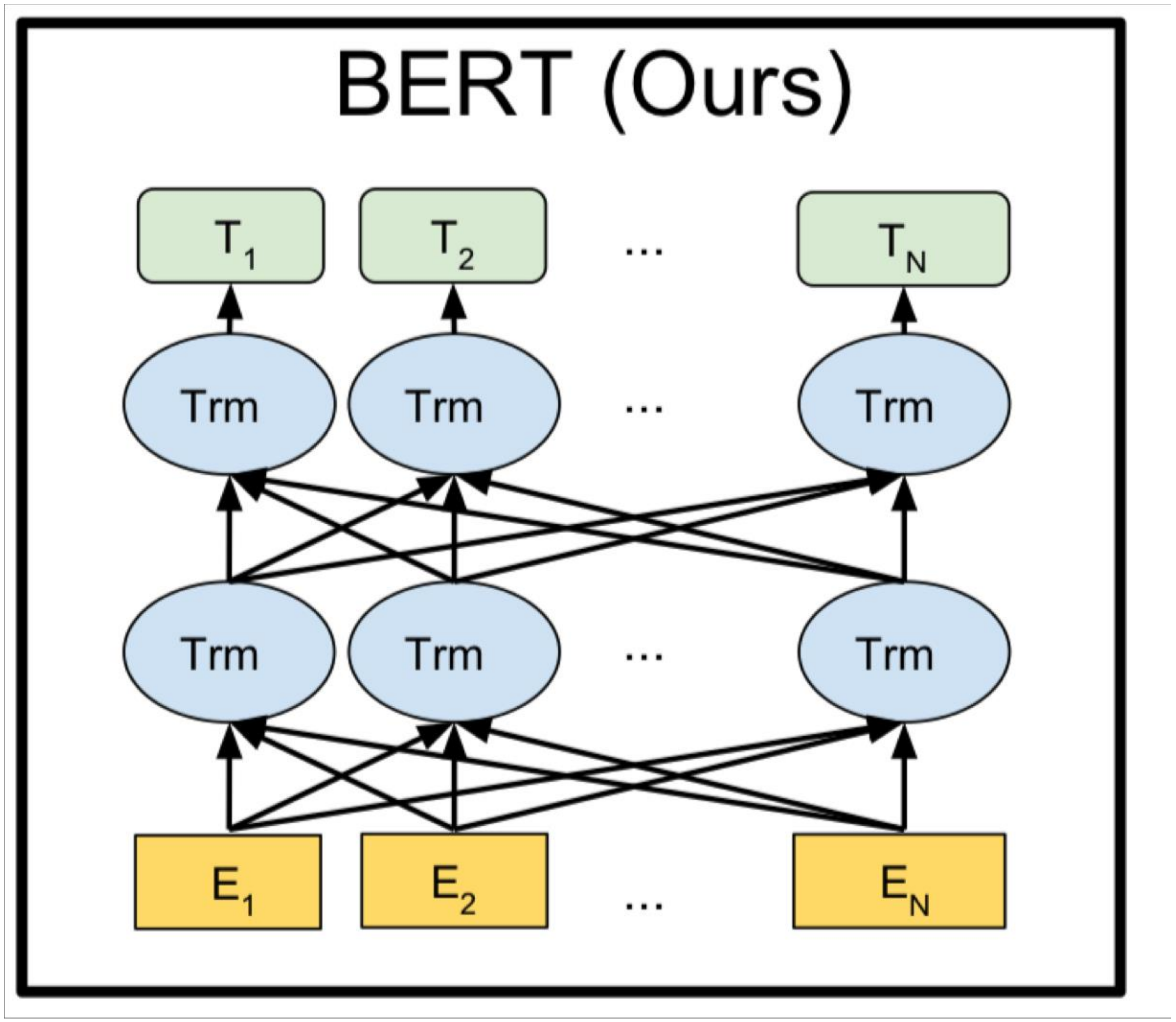
store gallon

↑ ↑

the man went to the [MASK] to buy a [MASK] of milk

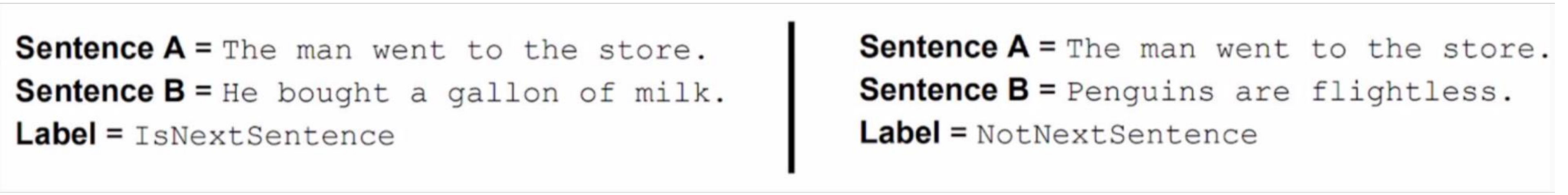
- Too little masking: Too expensive to train (not enough masks)
- Too much masking: Not enough context

BERT architecture



BERT uses several tasks

- besides masked LM, BERT learns relationships between sentences
- predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence



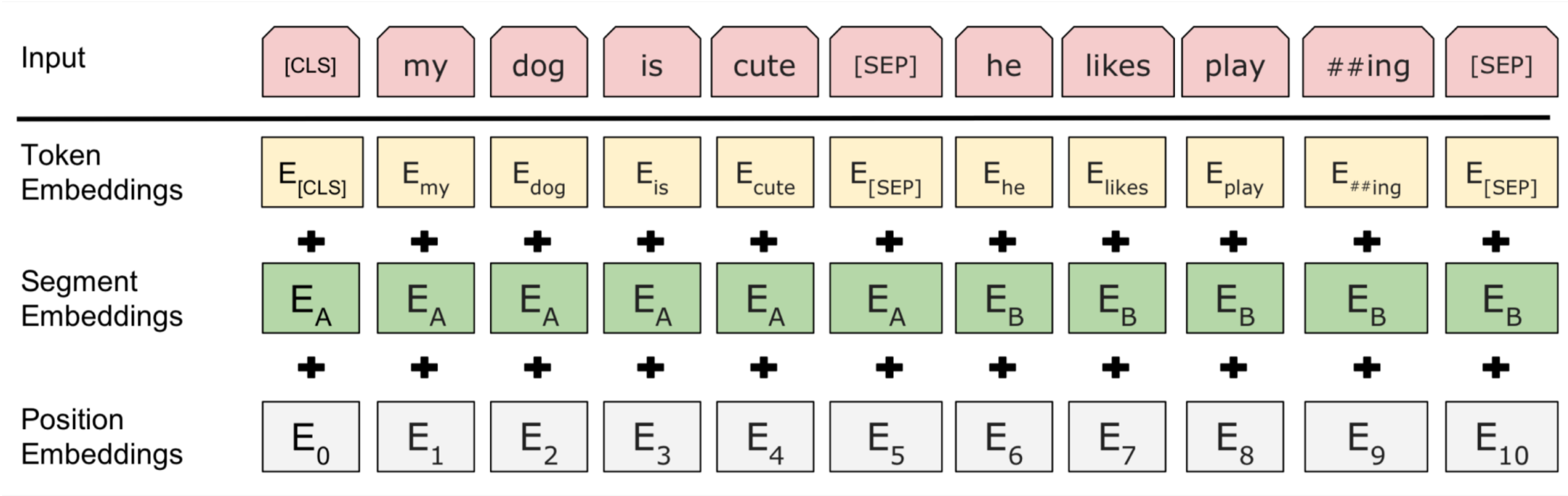
Sentence A = `The man went to the store.`
Sentence B = `He bought a gallon of milk.`
Label = `IsNextSentence`

Sentence A = `The man went to the store.`
Sentence B = `Penguins are flightless.`
Label = `NotNextSentence`

- some follow-up BERT-like models, e.g., RoBERTa, drop this task and claim better performance on downstream tasks

Sentence-pair encoding for BERT

- Token embeddings are word pieces (sub-word encoding)
- (Relatively) common words are in the vocabulary: *at, fairfax, 1910s*
- Other words are built from wordpieces: *hypatia = h ##yp ##ati ##a*
- Learned segmented embedding represents each sentence
- Positional embedding is the same as for other transformer architectures

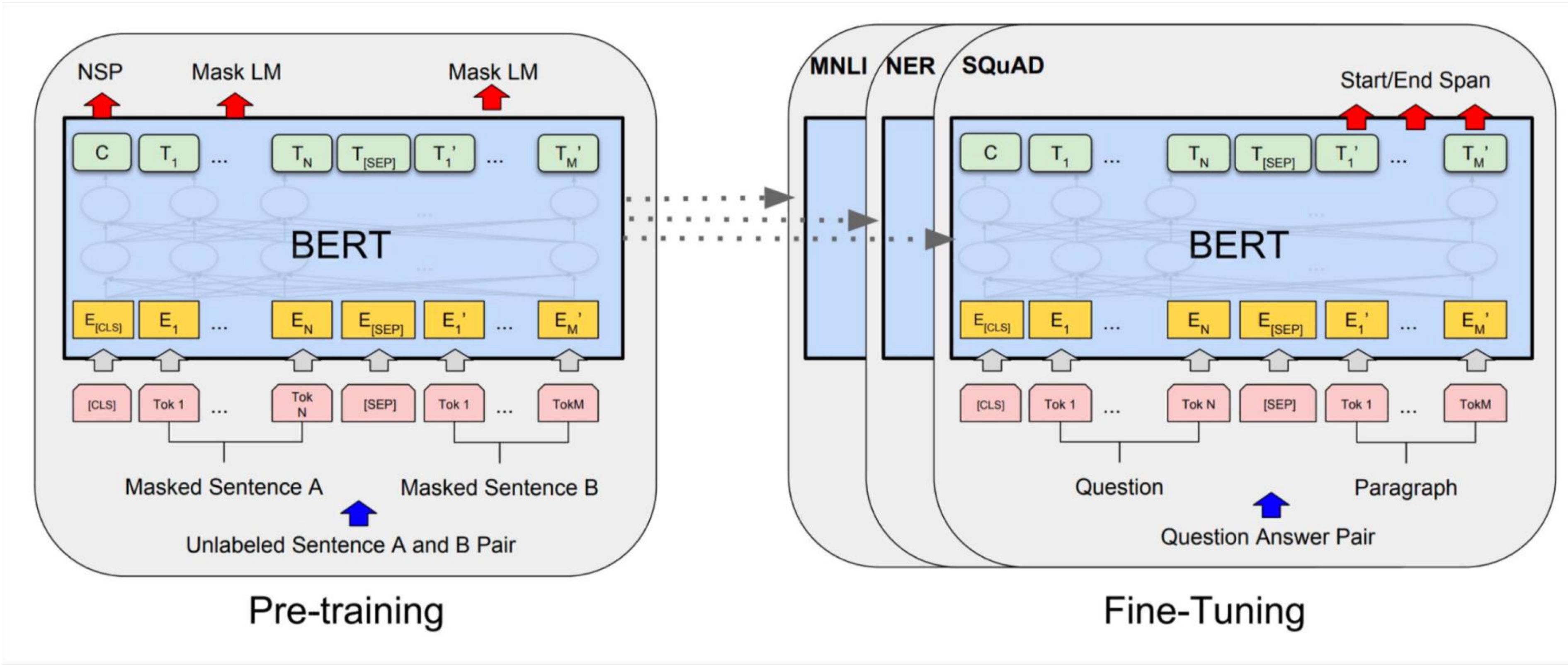


BERT training

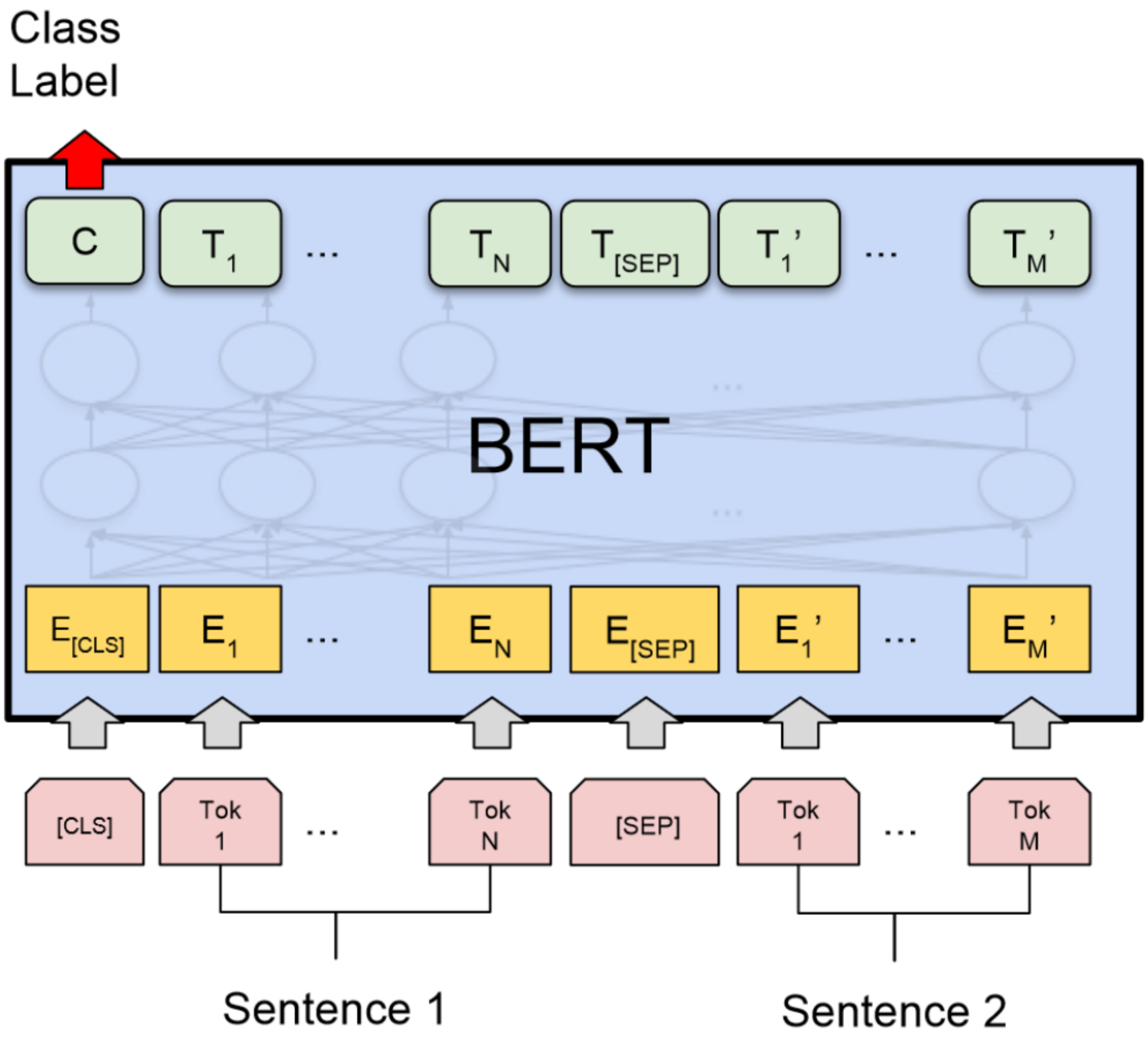
- Transformer encoder
- Self-attention ⇒ no locality bias
- Long-distance context has “equal opportunity”
- Single multiplication per layer ⇒ efficiency on GPU/TPU
- Trained on Wikipedia + BookCorpus
- English BERT was trained on 2 model sizes:
  - BERT-Base: 12-layer, 768-hidden neurons, 12-head, 110M parameters
  - BERT-Large: 24-layer, 1024-hidden neurons, 16-head, 340M parameters
- Trained on 4x4 or 8x8 TPU slice for 4 days

Use of BERT

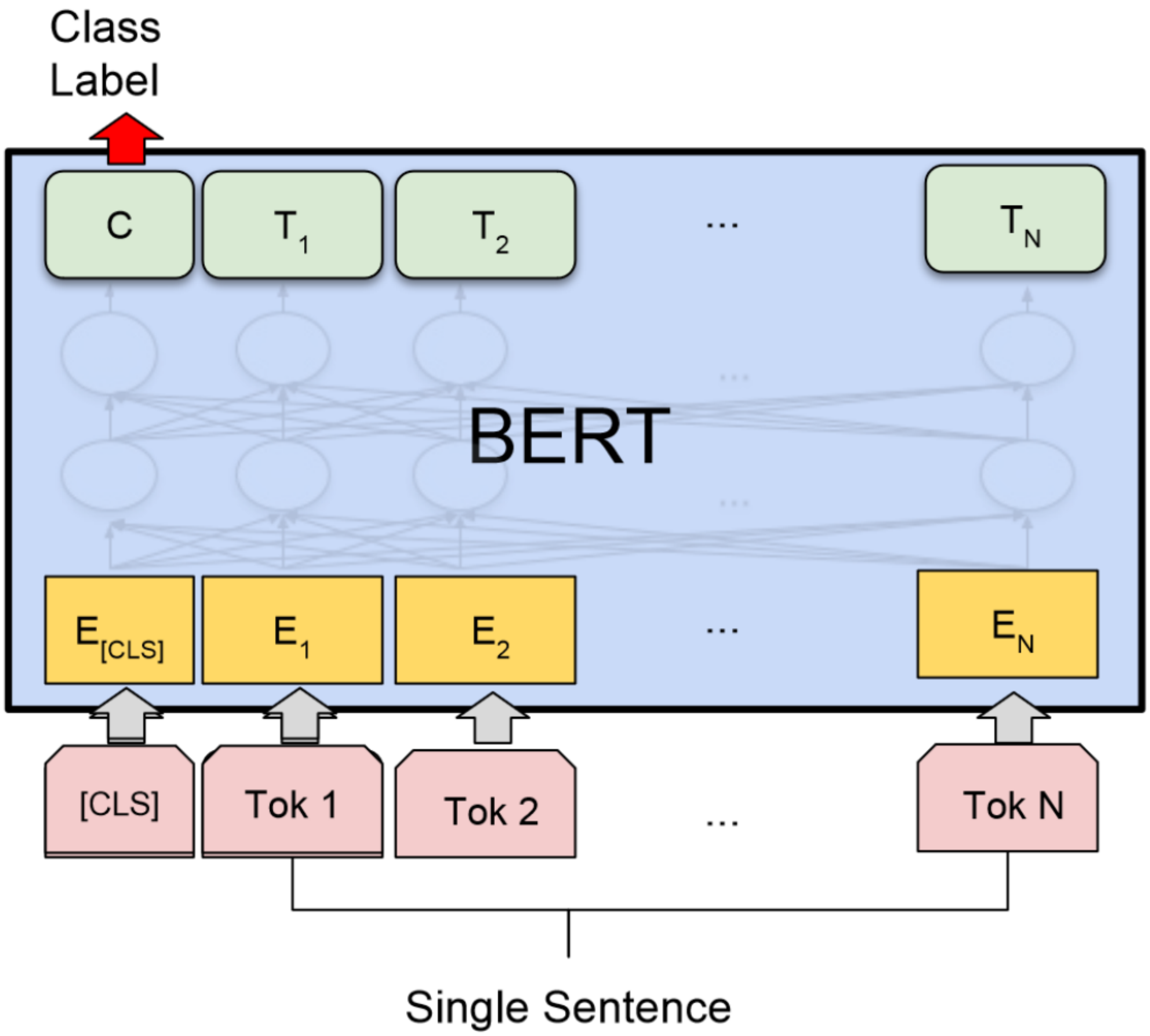
- train a classifier built on the top layer for each task that you fine-tune for, e.g., Q&A, NER, inference
- achieves state-of-the-art results for many tasks
- GLUE and SuperGLUE tasks for NLI



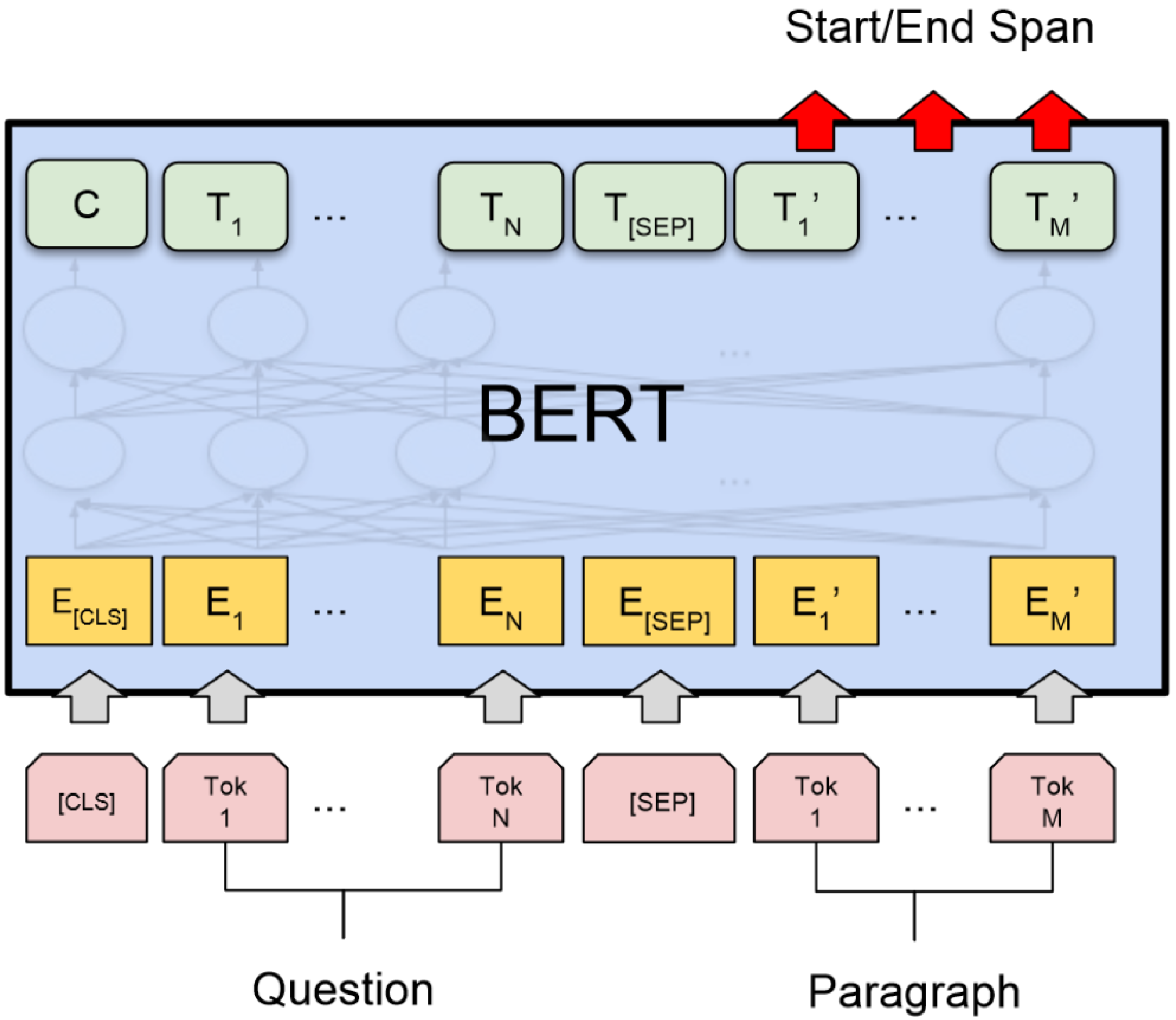
Two sentence classification using BERT-inference



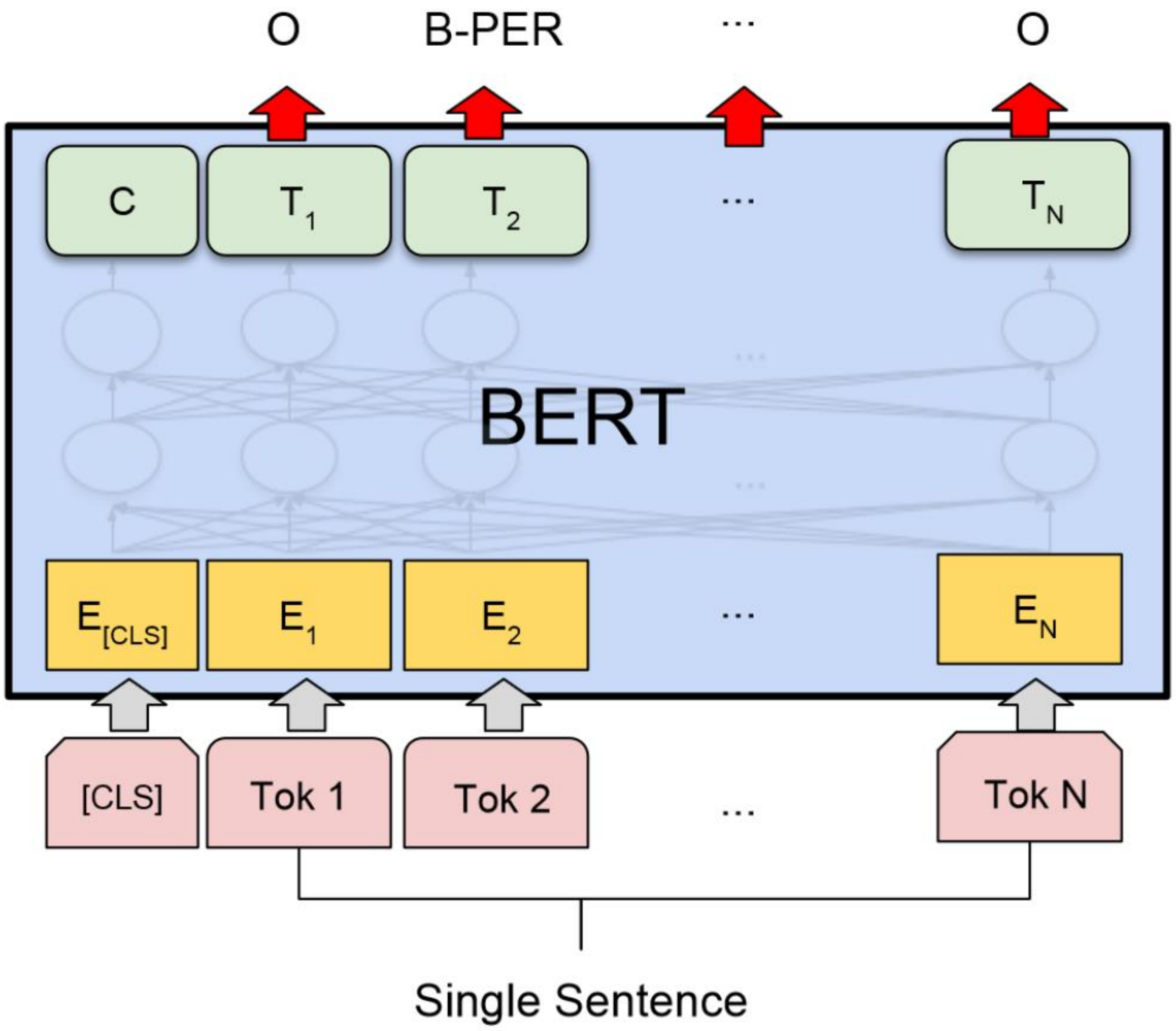
Sentence classification using BERT – sentiment, grammatical correctness



Questions and answers with BERT



Sentence tagging with BERT- NER, POS tagging, SRL



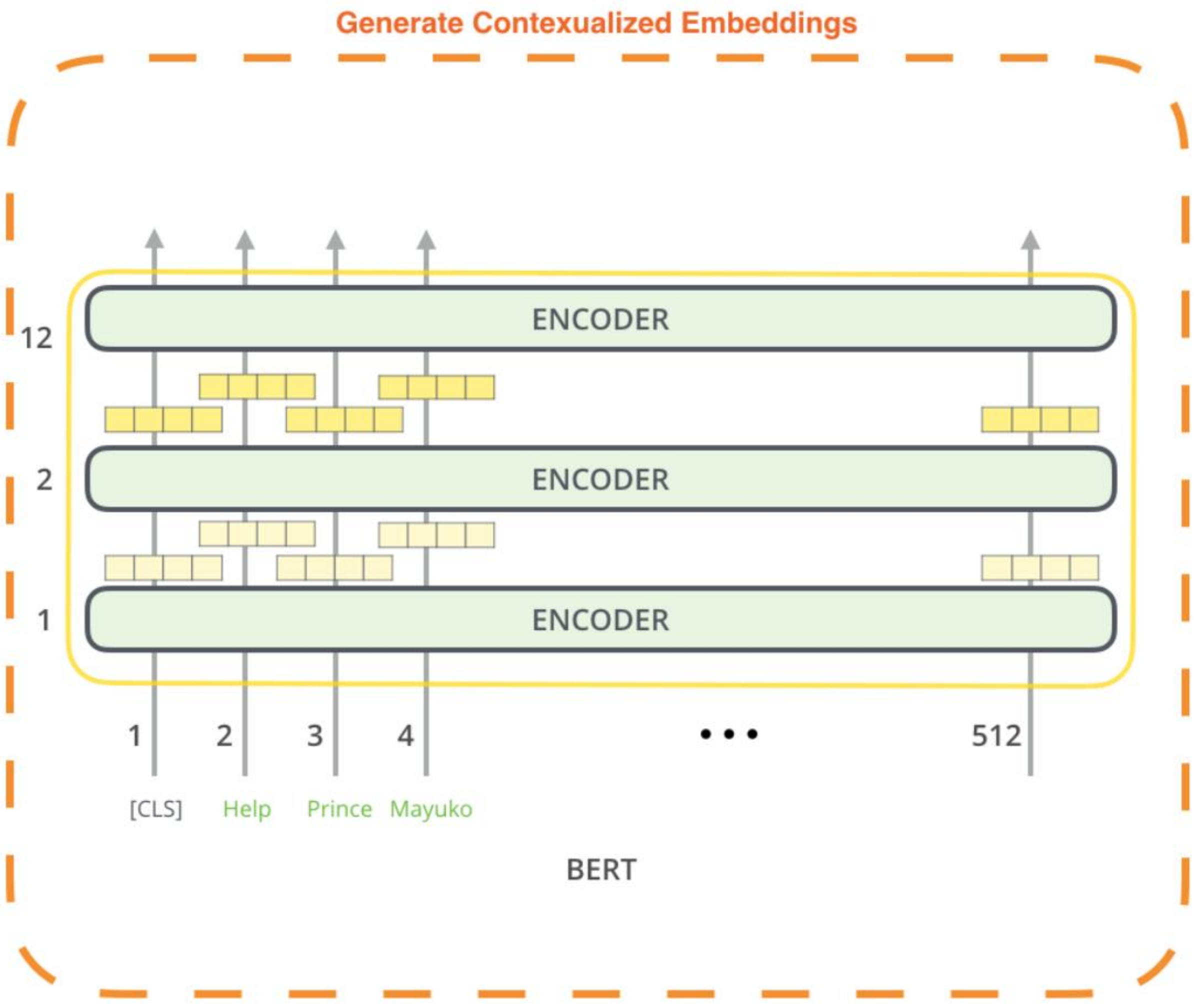
Multilingual BERT

- BERT base architecture was trained on 104 languages with the largest Wikipedia (at least 1 million words at the time)
- achieves good results for many tasks
- good cross-lingual transfer

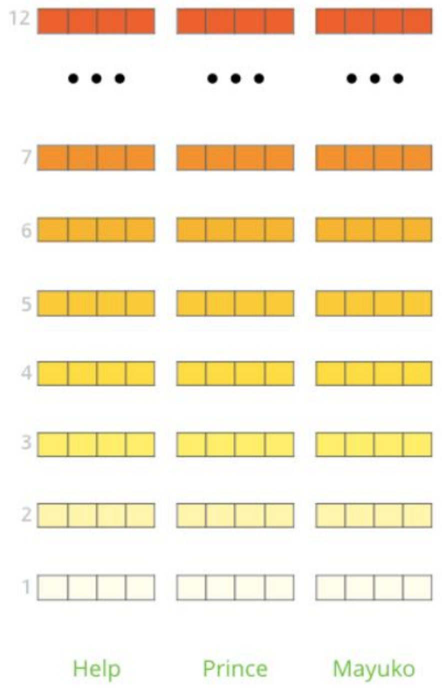
BERT can produce embeddings

- one can extract fixed size contextual vectors from BERT, achieving slightly lower accuracy than using the whole BERT as the first stage model

Layer-wise embeddings

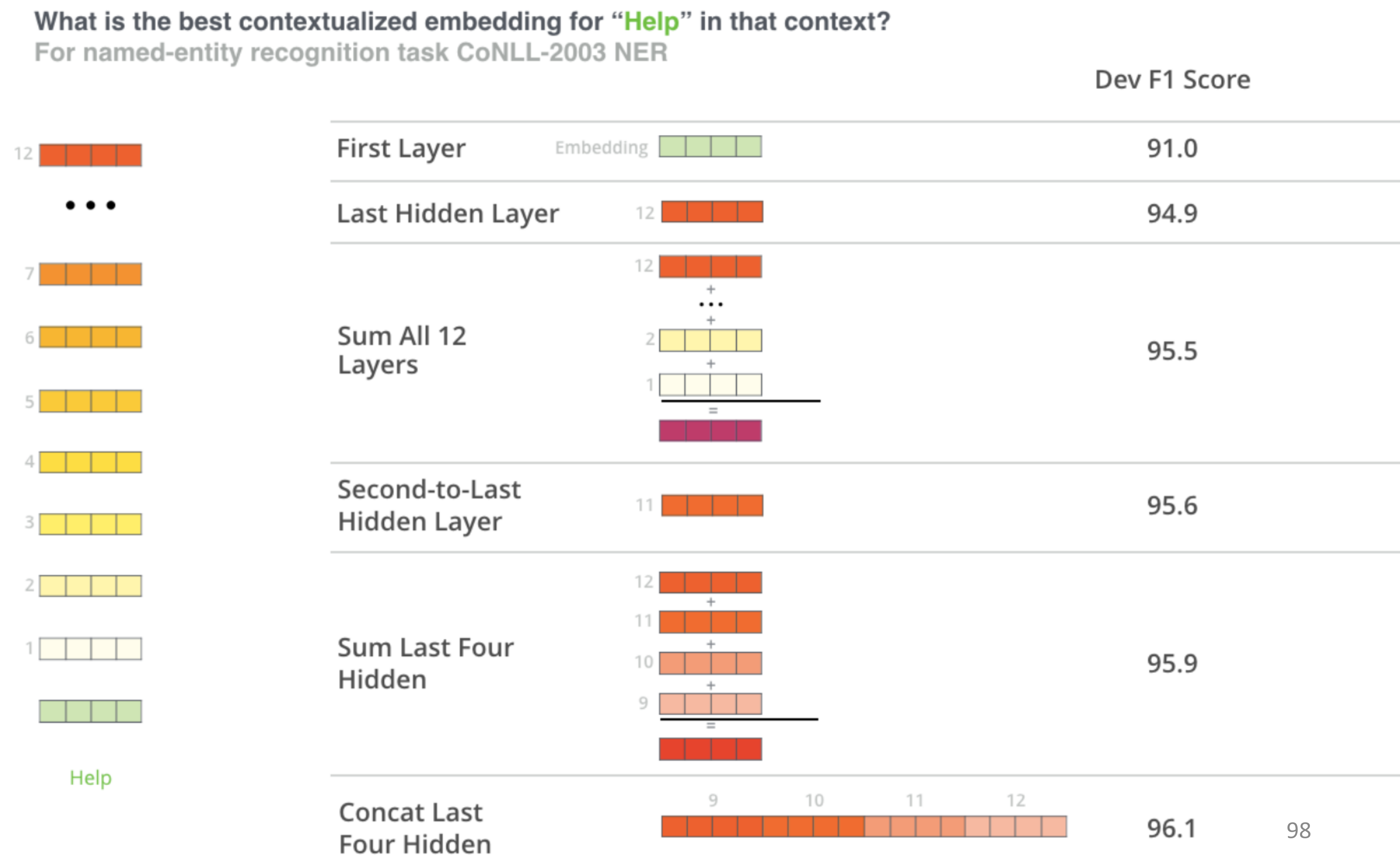


The output of each encoder layer along each token's path can be used as a feature representing that token.



But which one should we use?

Which layer of BERT to use as embeddings?


12
7
6
5
4
3
2
1
Help
Embedding
12
2
1
11
10
9

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

Examples of GLUE tasks

- GLUE benchmark is dominated by natural language inference tasks, but also has sentence similarity and sentiment

MultiNLI

Premise: Hills and mountains are especially sanctified in Jainism.

Hypothesis: Jainism hates nature.

Label: Contradiction

CoLA (Corpus of Linguistic Acceptability)

Sentence: The wagon rumbled down the road. Label: Acceptable

Sentence: The car honked down the road. Label: Unacceptable

SuperGLUE tasks

BoolQ - Boolean Questions
CB – Commitment Bank
COPA - Choice of Plausible Alternatives

MultiRC - Multi-Sentence Reading Comprehension
ReCoRD - Reading Comprehension with Commonsense Reasoning Dataset
RTE - Recognizing Textual Entailment
WiC - Word-in-Context
WSC - Winograd Schema Challeng

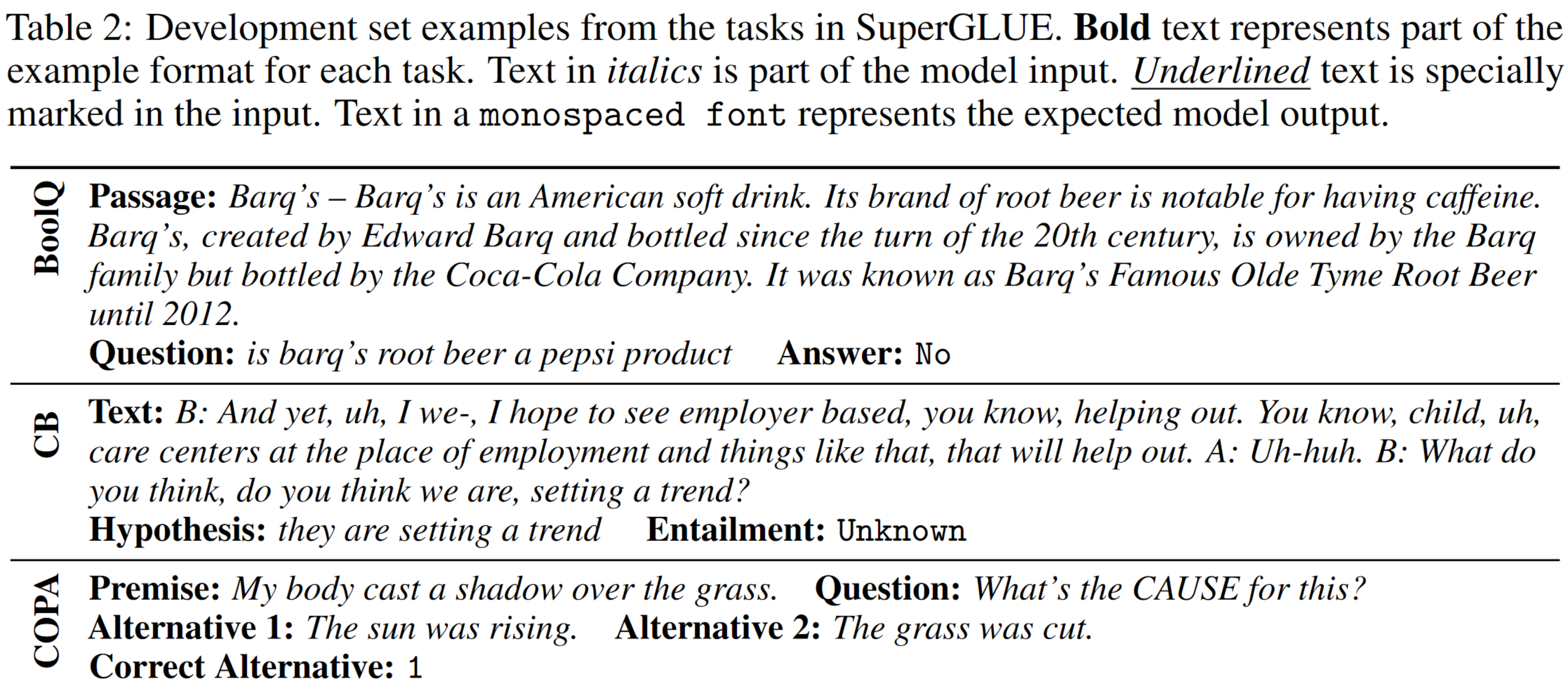


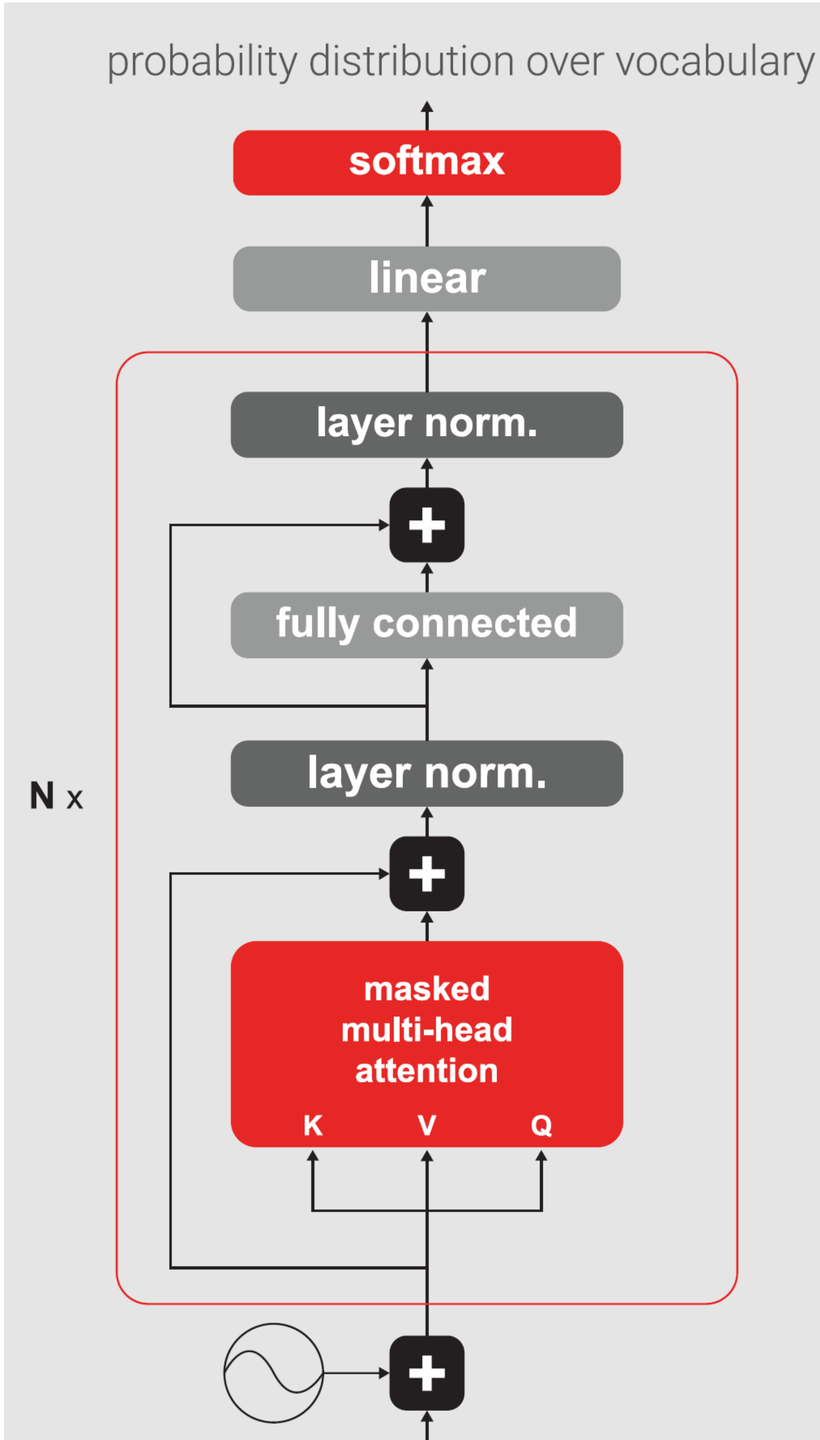
Table 2: Development set examples from the tasks in SuperGLUE. **Bold** text represents part of the example format for each task. Text in *italics* is part of the model input. <u>*Underlined*</u> text is specially marked in the input. Text in a `monospaced font` represents the expected model output.

BoolQ	**Passage:** *Barq's – Barq's is an American soft drink. Its brand of root beer is notable for having caffeine. Barq's, created by Edward Barq and bottled since the turn of the 20th century, is owned by the Barq family but bottled by the Coca-Cola Company. It was known as Barq's Famous Olde Tyme Root Beer until 2012.* **Question:** *is barq's root beer a pepsi product* **Answer:** `No`
CB	**Text:** *B: And yet, uh, I we-, I hope to see employer based, you know, helping out. You know, child, uh, care centers at the place of employment and things like that, that will help out. A: Uh-huh. B: What do you think, do you think we are, setting a trend?* **Hypothesis:** *they are setting a trend* **Entailment:** `Unknown`
COPA	**Premise:** *My body cast a shadow over the grass.* **Question:** *What's the CAUSE for this?* **Alternative 1:** *The sun was rising.* **Alternative 2:** *The grass was cut.* **Correct Alternative:** `1`

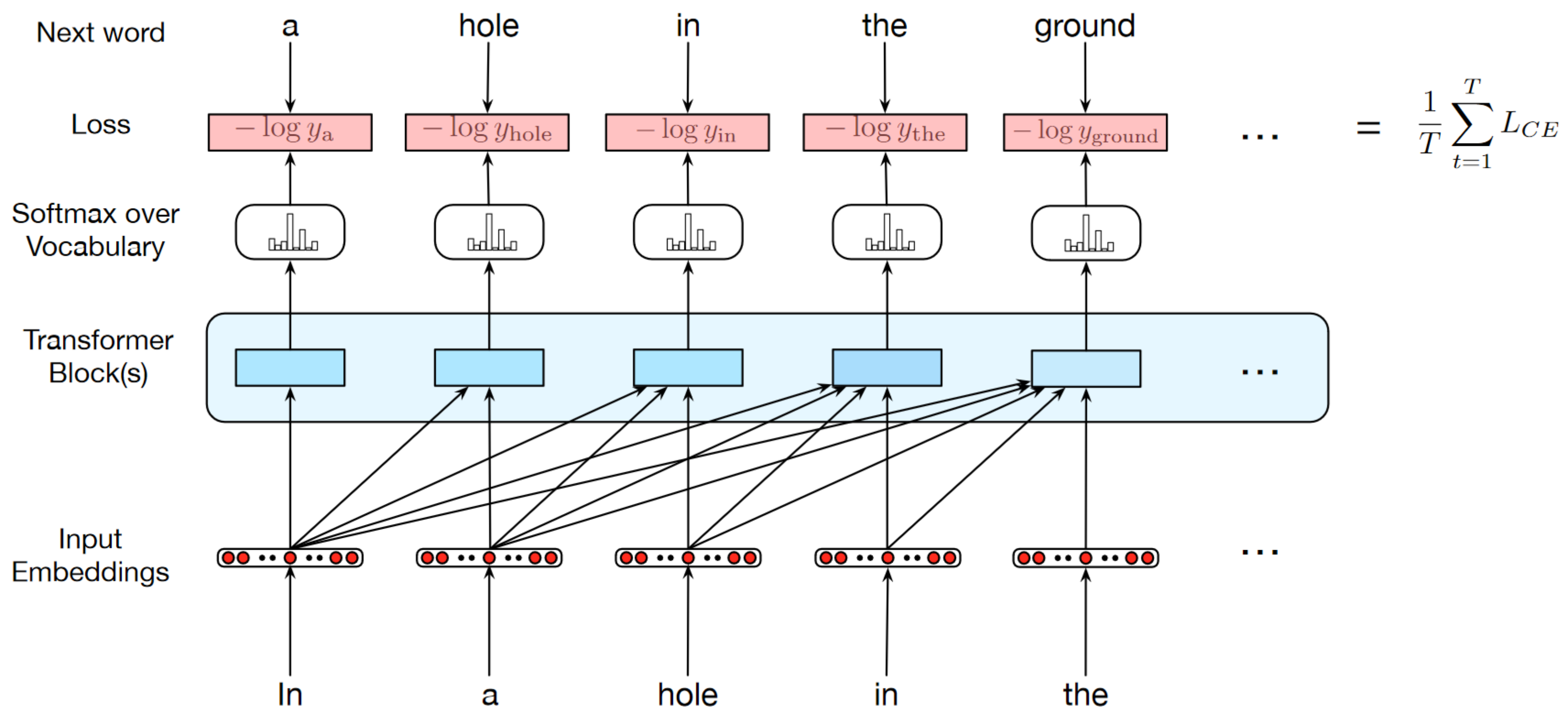
MultiRC	**Paragraph:** *Susan wanted to have a birthday party. She called all of her friends. She has five friends. Her mom said that Susan can invite them all to the party. Her first friend could not go to the party because she was sick. Her second friend was going out of town. Her third friend was not so sure if her parents would let her. The fourth friend said maybe. The fifth friend could go to the party for sure. Susan was a little sad. On the day of the party, all five friends showed up. Each friend had a present for Susan. Susan was happy and sent each friend a thank you card the next week* **Question:** *Did Susan's sick friend recover?* **Candidate answers:** *Yes, she recovered* (`T`), *No* (`F`), *Yes* (`T`), *No, she didn't recover* (`F`), *Yes, she was at Susan's party* (`T`)
ReCoRD	**Paragraph:** *(CNN) Puerto Rico on Sunday overwhelmingly voted for statehood. But Congress, the only body that can approve new states, will ultimately decide whether the status of the US commonwealth changes. Ninety-seven percent of the votes in the nonbinding referendum favored statehood, an increase over the results of a 2012 referendum, official results from the State Electorcal Commission show. It was the fifth such vote on statehood. "Today, we the people of Puerto Rico are sending a strong and clear message to the US Congress ... and to the world ... claiming our equal rights as American citizens, Puerto Rico Gov. Ricardo Rossello said in a news release. @highlight Puerto Rico voted Sunday in favor of US statehood* **Query** For one, they can truthfully say, "Don't blame me, I didn't vote for them, " when discussing the <placeholder> presidency **Correct Entities:** `US`
RTE	**Text:** *Dana Reeve, the widow of the actor Christopher Reeve, has died of lung cancer at age 44, according to the Christopher Reeve Foundation.* **Hypothesis:** *Christopher Reeve had an accident.* **Entailment:** `False`
WiC	**Context 1:** *Room and board.* **Context 2:** *He nailed boards across the windows.* **Sense match:** `False`
WSC	**Text:** *Mark told Pete many lies about himself, which Pete included in his book. He should have been more truthful.* **Coreference:** `False`

GPT family

- GPT: Generative Pre-trained Transformers
- use only the decoder part of transformer
- pretrained for language modeling (predicting the next word given the context)
- Shortcoming: unidirectional, does not incorporate bidirectionality
- "What are <u>those</u>?" he said while looking at my <u>crocs</u>.


probability distribution over vocabulary
softmax
linear
layer norm.
+
fully connected
N x
layer norm.
+
masked multi-head attention
K
V
Q
+

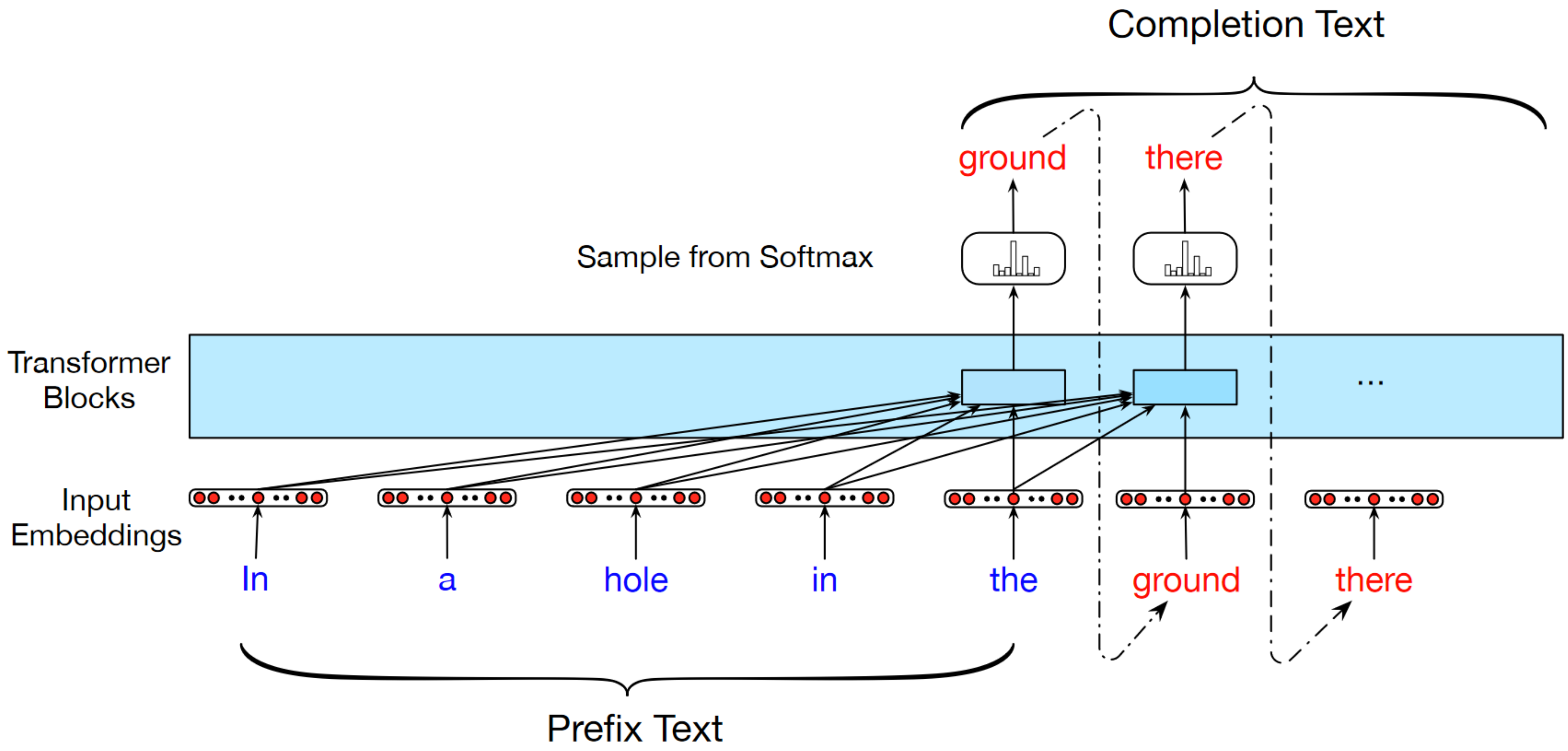
Transformer as language model



- Can be computed in parallel

Autoregressive generators

- priming the generator with the context



- can be used also in summarization

GPT-2 and GPT-3

- few architectural changes, layer norm now applied to input of each subblock
- GPT-3 also uses some sparse attention layers
- more data, larger batch sizes (GPT-3 uses batch size of 3.2M)
- the models are scaled:

GPT-2:	**GPT-3:**
48 layers, 25 heads	96 layers, 96 heads
$d_m = 1600$, $d = 64$	$d_m = 12288$, $d = 128$
context size = 1024	context size = 2048
~ 1.5B parameters	**~ 175B parameters**

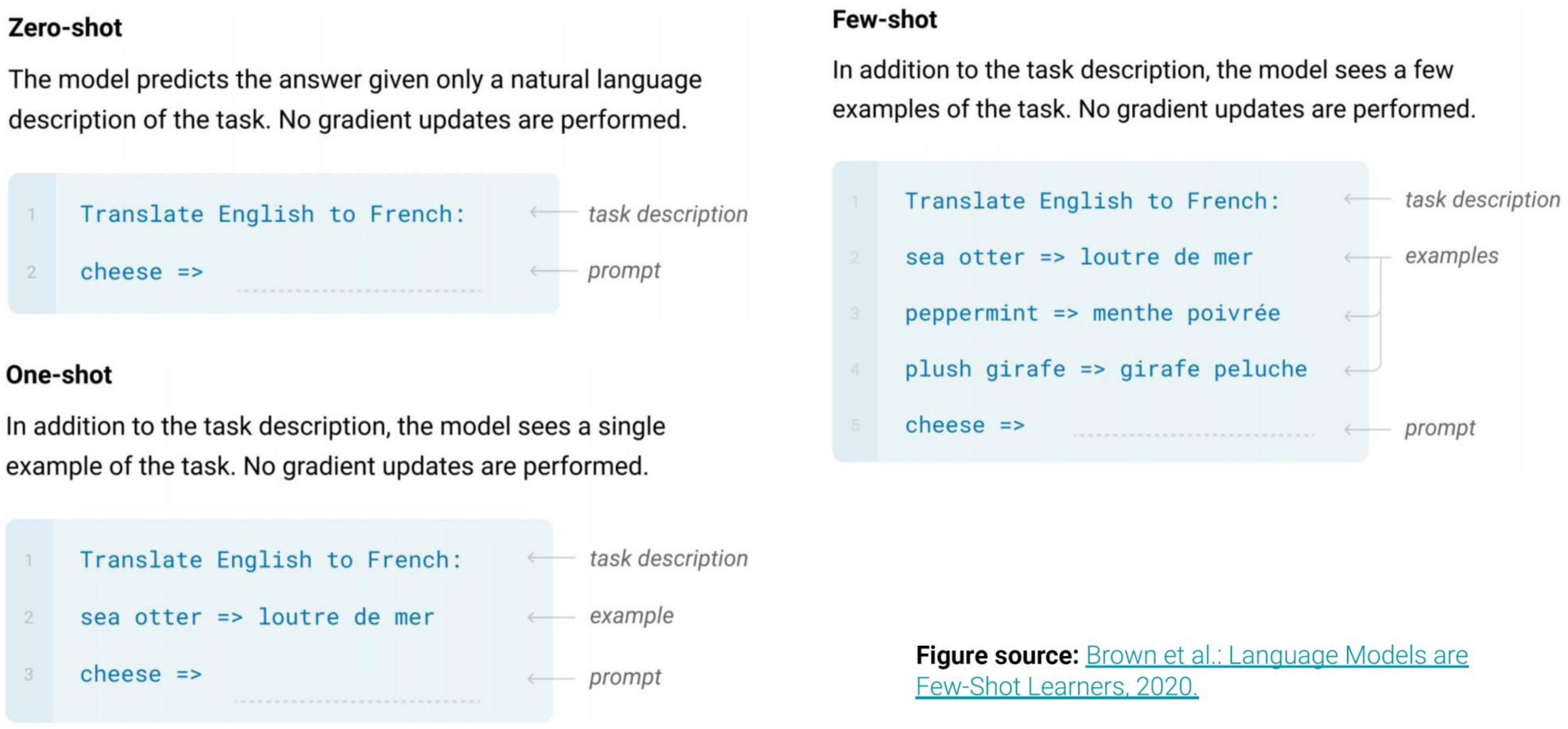
[1] Radford et al.: Language Models are Unsupervised Multitask Learners, 2019.
[2] Brown et al.: Language Models are Few-Shot Learners, 2020.

- see demos at https://transformer.huggingface.co/

In-context learning in GPT-2 and GPT-3

- GPT-2 and GPT-3 ditch the “pre-train and fine-tune” training paradigm of GPT;
- GPT-2 explores unsupervised zero-shot learning, whereas in GPT-3 the authors expand the idea into in-context learning;
- use text input to condition the model on task description and some examples with ground truth.
- Uses zero-shot learning, one-shot learning, few-shot learning (as many examples as they can fit into the context, usually 10-100)
- no gradients updates are performed.

In-context learning



Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:
cheese =>
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:
sea otter => loutre de mer
cheese =>
```

Few-shot

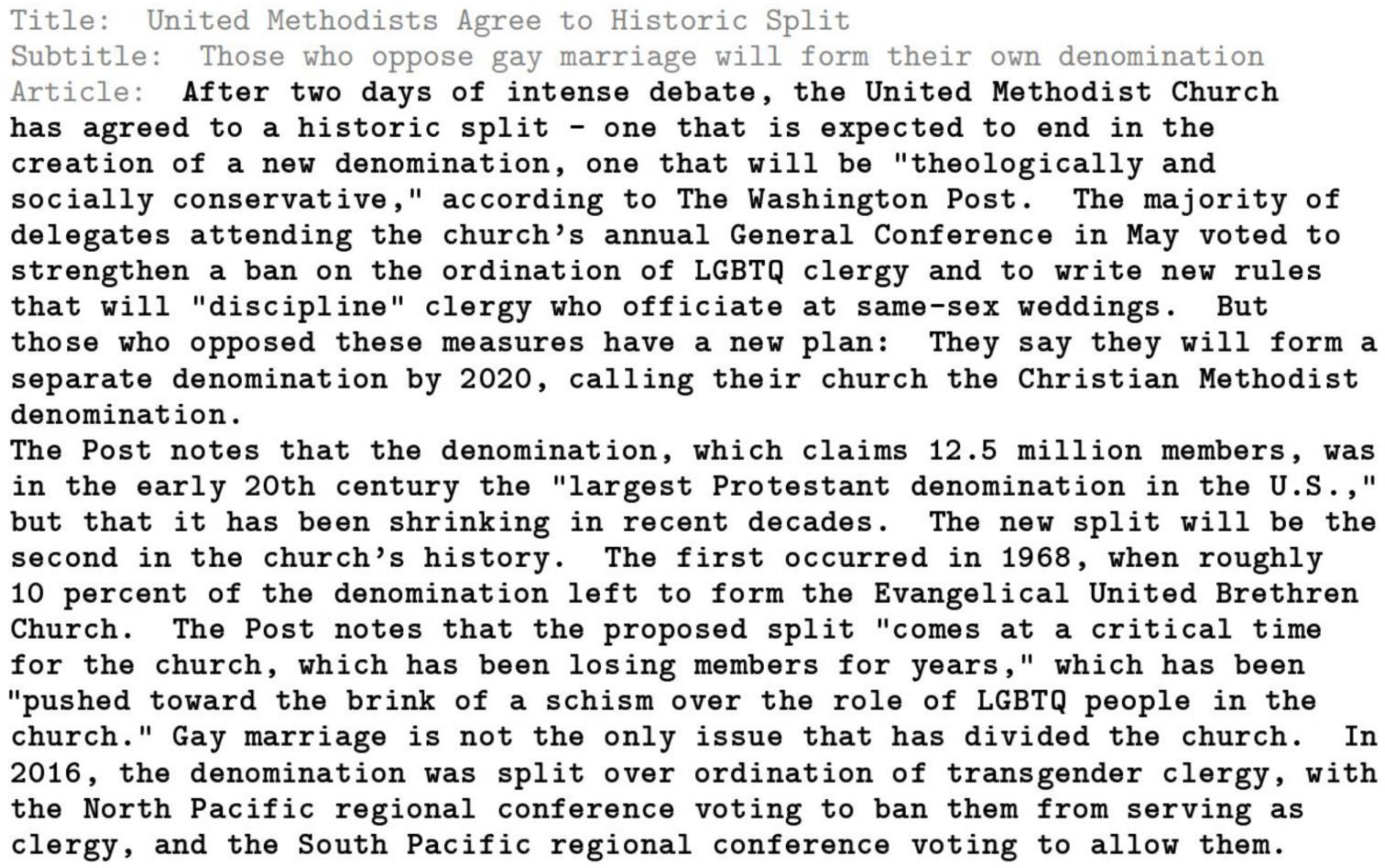
In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>
```

Figure source: Brown et al.: Language Models are Few-Shot Learners, 2020.

- GPT-3 is still a language model and can be used for text generation

- only 12% of respondents correctly classified this as not written by a human



```
Title:  United Methodists Agree to Historic Split
Subtitle:  Those who oppose gay marriage will form their own denomination
Article:  After two days of intense debate, the United Methodist Church
has agreed to a historic split - one that is expected to end in the
creation of a new denomination, one that will be "theologically and
socially conservative," according to The Washington Post.  The majority of
delegates attending the church's annual General Conference in May voted to
strengthen a ban on the ordination of LGBTQ clergy and to write new rules
that will "discipline" clergy who officiate at same-sex weddings.  But
those who opposed these measures have a new plan:  They say they will form a
separate denomination by 2020, calling their church the Christian Methodist
denomination.
The Post notes that the denomination, which claims 12.5 million members, was
in the early 20th century the "largest Protestant denomination in the U.S.,"
but that it has been shrinking in recent decades.  The new split will be the
second in the church's history.  The first occurred in 1968, when roughly
10 percent of the denomination left to form the Evangelical United Brethren
Church.  The Post notes that the proposed split "comes at a critical time
for the church, which has been losing members for years," which has been
"pushed toward the brink of a schism over the role of LGBTQ people in the
church." Gay marriage is not the only issue that has divided the church.  In
2016, the denomination was split over ordination of transgender clergy, with
the North Pacific regional conference voting to ban them from serving as
clergy, and the South Pacific regional conference voting to allow them.
```

Attention efficiency

- time and space complexity of self-attention grows quadratically with n (size of input)

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \qquad K, V, Q \in \mathbb{R}^{n \times d}$$

- not suitable for very long sequences like
  - documents
  - character-level language models
  - images (as sequences of pixels);
  - protein sequences.

Many approaches to reducing the complexity

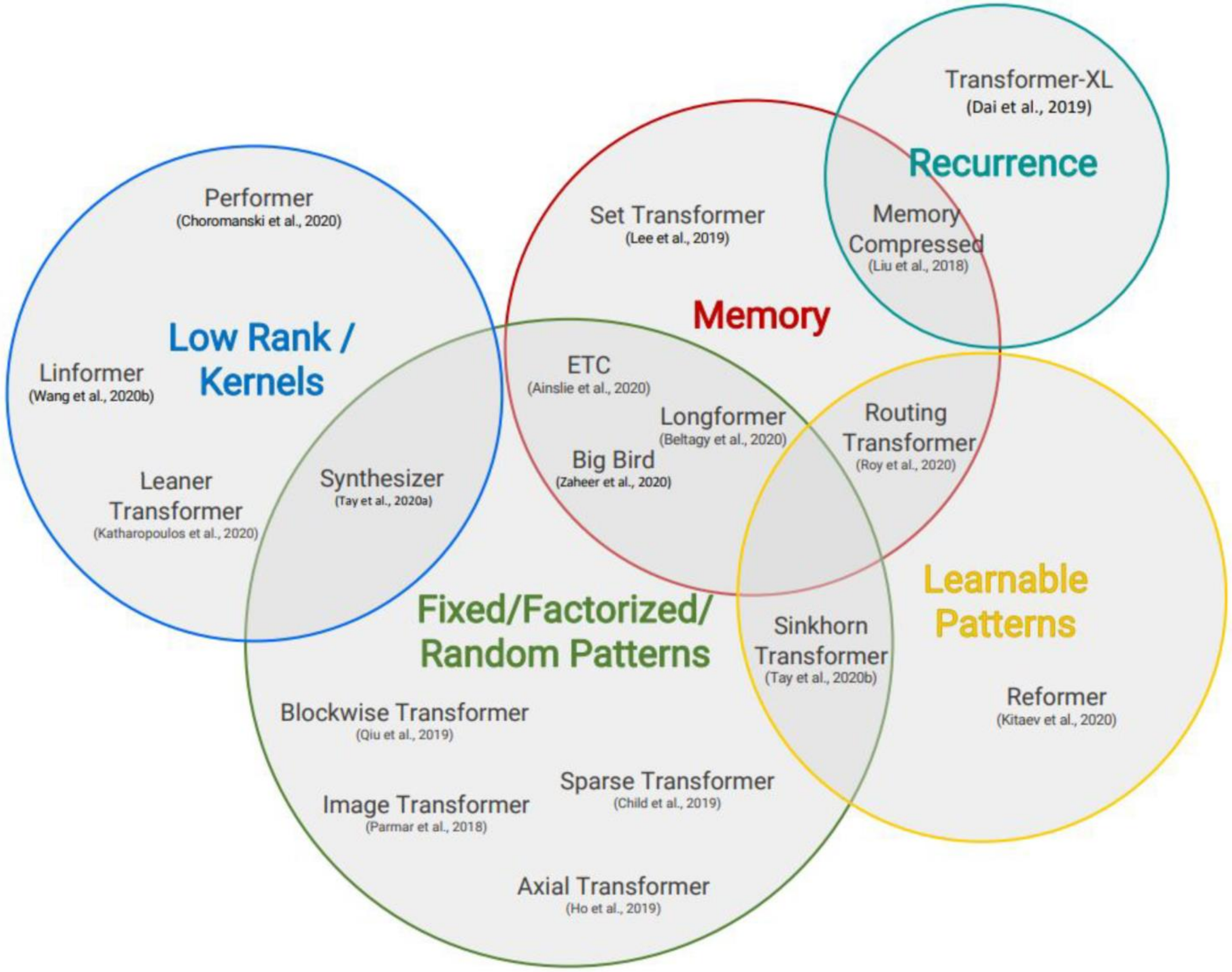


Figure source: Tay et al.: Efficient Transformers: A Survey, 2020.

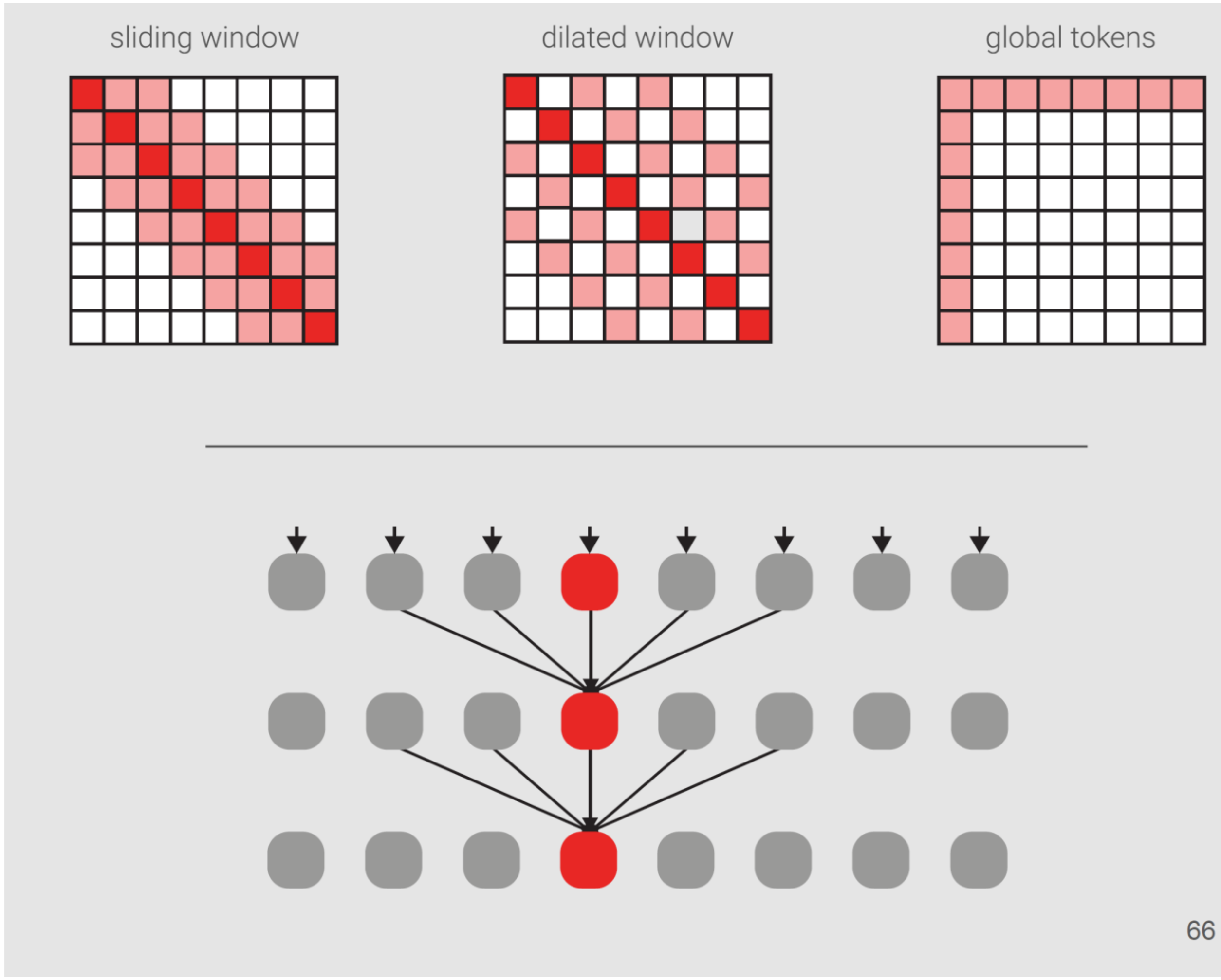
Longformer [1]

- sliding window attention:
each position can attend to 1/2w tokens on each side - **O(w x n)**

- dilated window attention:
increases the receptive field of the attention layer - **O(w x n)**

- global attention:
k special tokens that aggregate information from whole sequence (e.g. [CLS] as in BERT) - **O(k x n)**

[1] Beltagy et al.: Longformer: The Long-Document Transformer, 2020.



Transformers are everywhere

- music: vocabulary consists of MIDI pitches, pauses, velocity
- object detection (attention to objects)