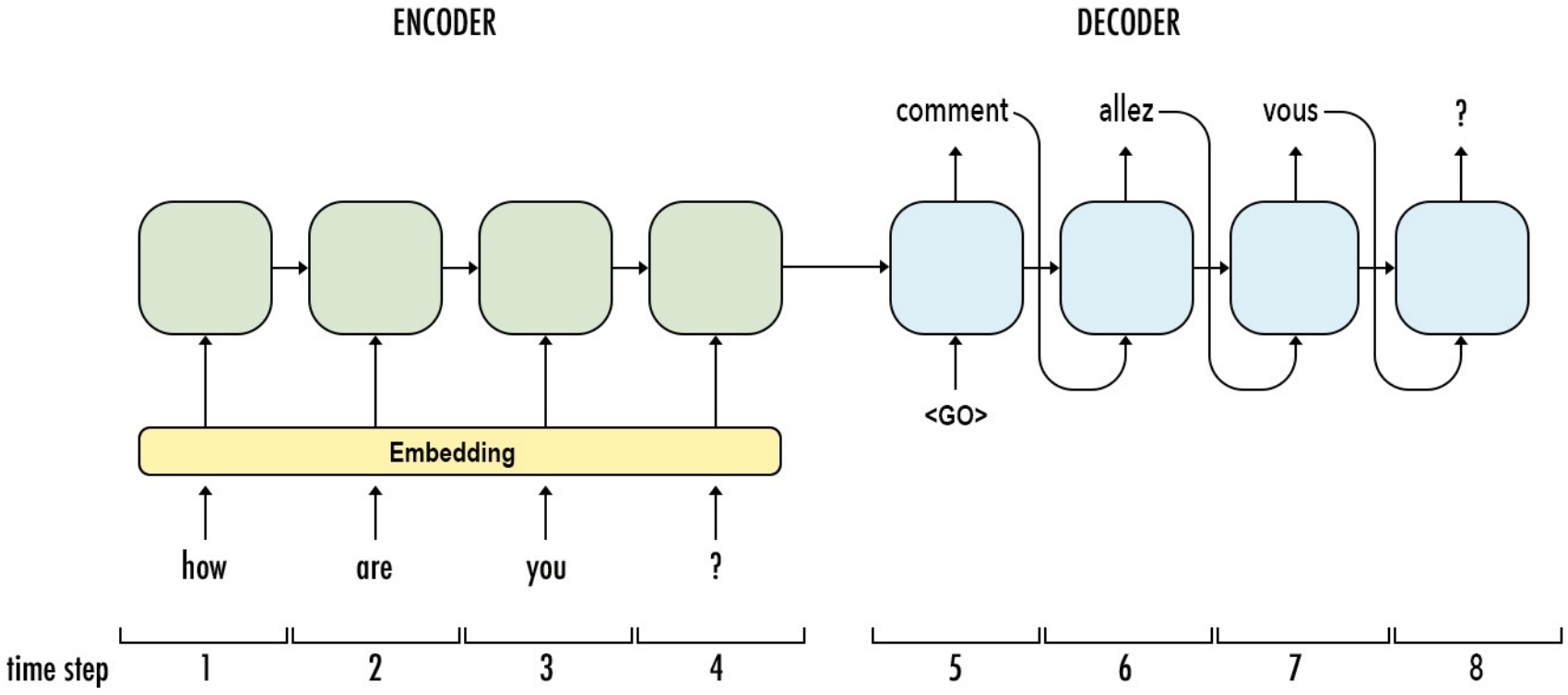
Attention & Transformer

Introduction

Learning sequential **input-output relations** is of great interest, e.g., Neural Machine Translation (NMT), Image captioning, etc.

Idea: compress a sequential input into a fixed size vector and this vector was fed to a recurrent neural network (RNN) to generate a sequence of outputs.

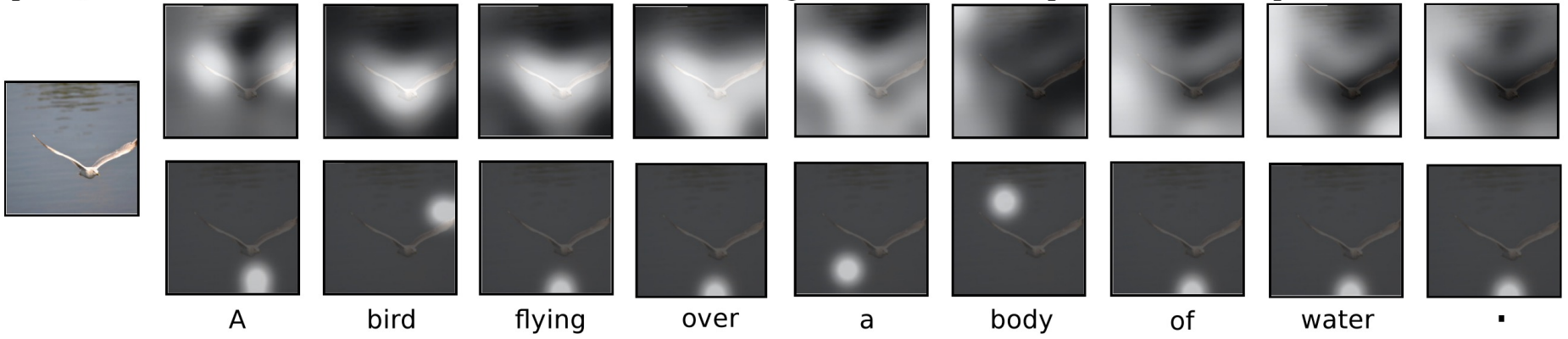


Motivation

Human beings rarely use all the available sensory inputs in order to accomplish specific tasks.

- Attention started out in the field of computer vision as an attempt to **mimic human perception**.
- Attention is a class of **sequence-to-sequence (seq2seq)** models

Figure 2. Attention over time. As the model generates each word, its attention changes to reflect the relevant parts of the image. "soft" (top row) vs "hard" (bottom row) attention. (Note that both models generated the same captions in this example.)



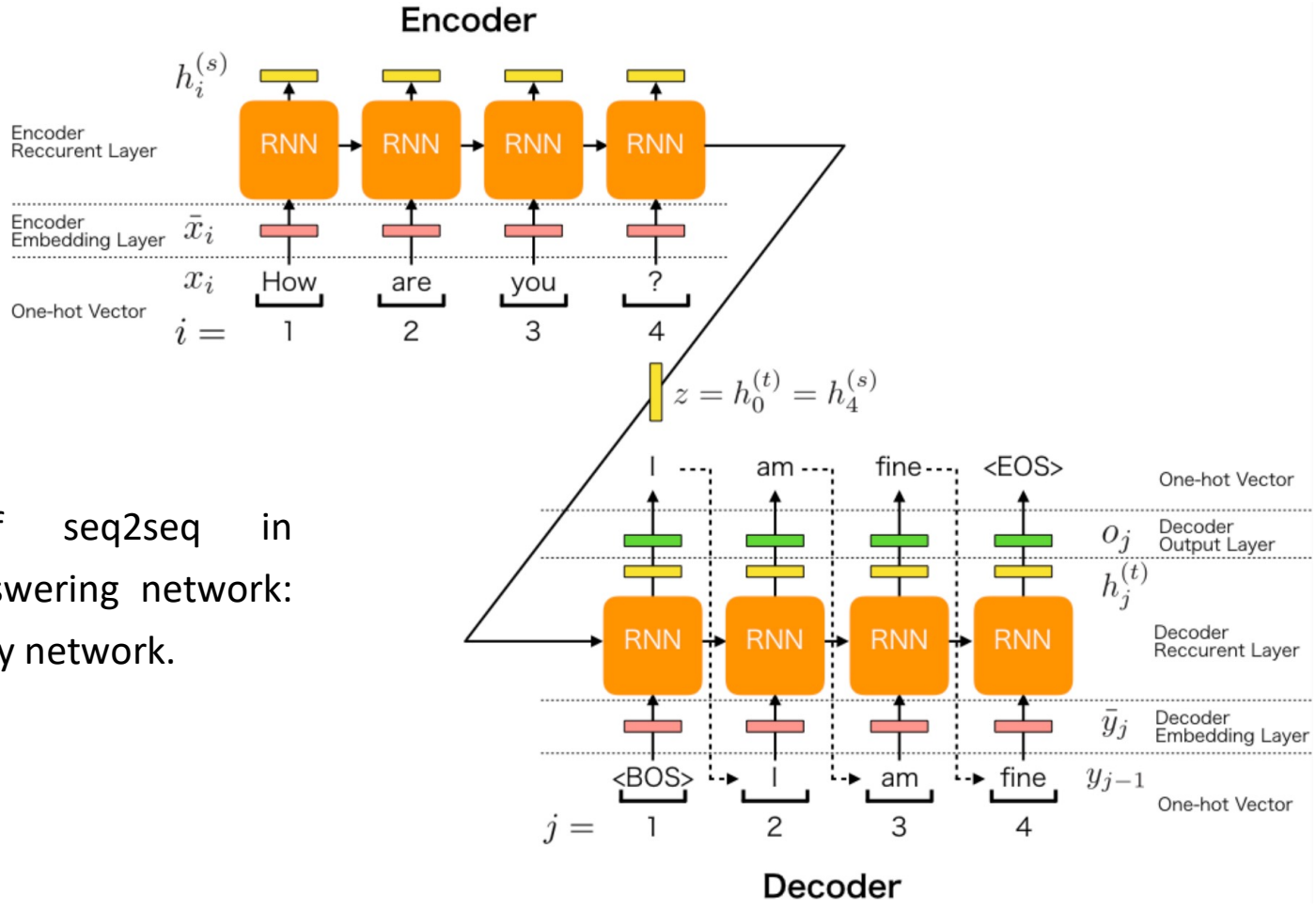
Kelvin et al., "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML'15

Sequence-to-Sequence model (seq2seq)

Two recurrent neural networks (RNNs) with an encoder-decoder architecture:

- **Encoder**: read the input words one by one to obtain an embedding vector of a fixed dimensionality.
- **Decoder**: conditioned on these inputs extract the output words one by one using another RNN.

Example of seq2seq in Question Answering network: many-to-many network.



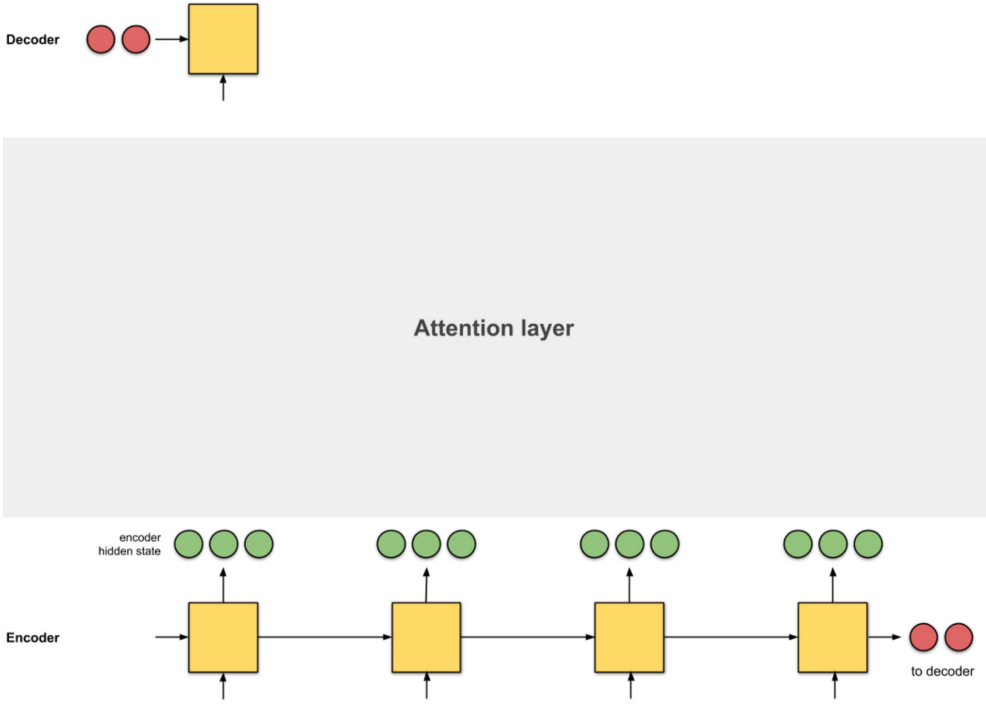
Sequence-to-Sequence model (seq2seq)

Issue of seq2seq:

- Only information that the decoder receives from the encoder is the **last encoder hidden state** (i.e., $h_4(s)$), a vector representation which is like a numerical summary of an input sequence.
- In case of long input text, the decoder uses just this one vector representation to output a translation → **severe loss** of information from the input.

→ Solution: instead of just one vector representation, let's give the decoder a vector representation from every encoder time step so that it can make well-informed translations

→ **Attention model.**



Sequence-to-Sequence model (seq2seq)

Issue of seq2seq:

- Only information that the decoder receives from the encoder is the **last encoder hidden state** (i.e., $h_4(s)$), a vector representation which is like a numerical summary of an input sequence.
- In case of long input text, the decoder uses just this one vector representation to output a translation → **severe loss** of information from the input.

→ Solution: instead of just one vector representation, let's give the decoder a vector representation from every encoder time step so that it can make well-informed translations

→ **Attention model.**

"The basic idea of Attention mechanism is to avoid attempting to learn a single vector representation for each sentence, instead, it pays attention to specific input vectors of the input sequence based on the attention weights."

Attention

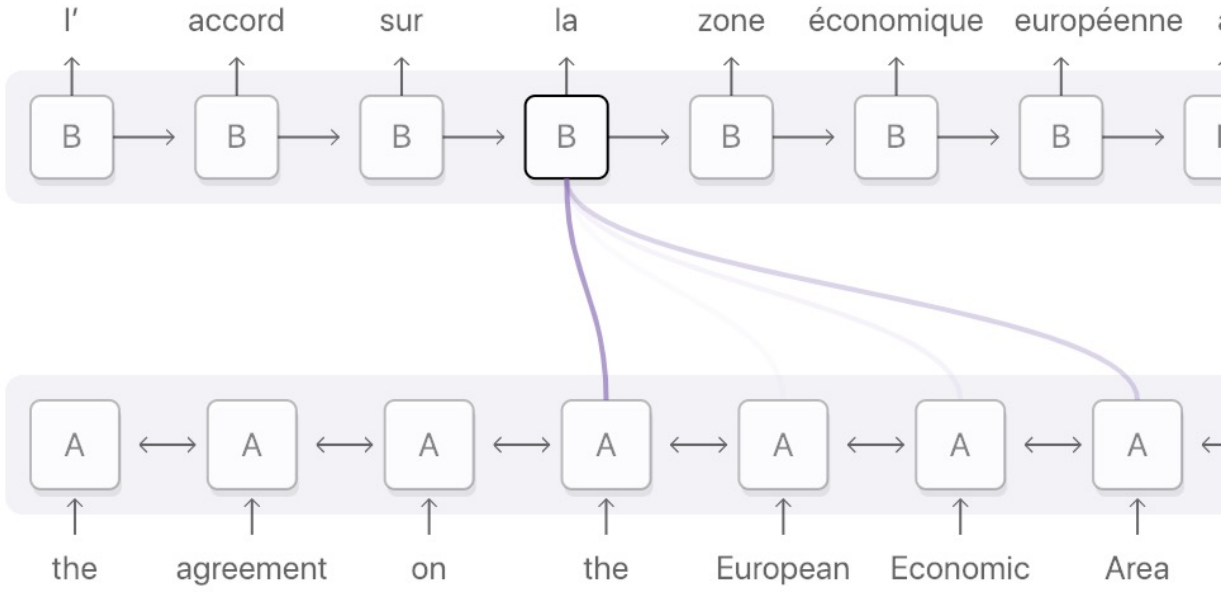
Introduction:

- Attention is an interface between the encoder and decoder that provides the decoder with information from **every encoder hidden state** (apart from the last encoder hidden state).
- Model is able to selectively focus on useful parts of the input sequence and hence, learn the **alignment** between the input elements and the output elements.

→ Effectively cope with long input sentences.

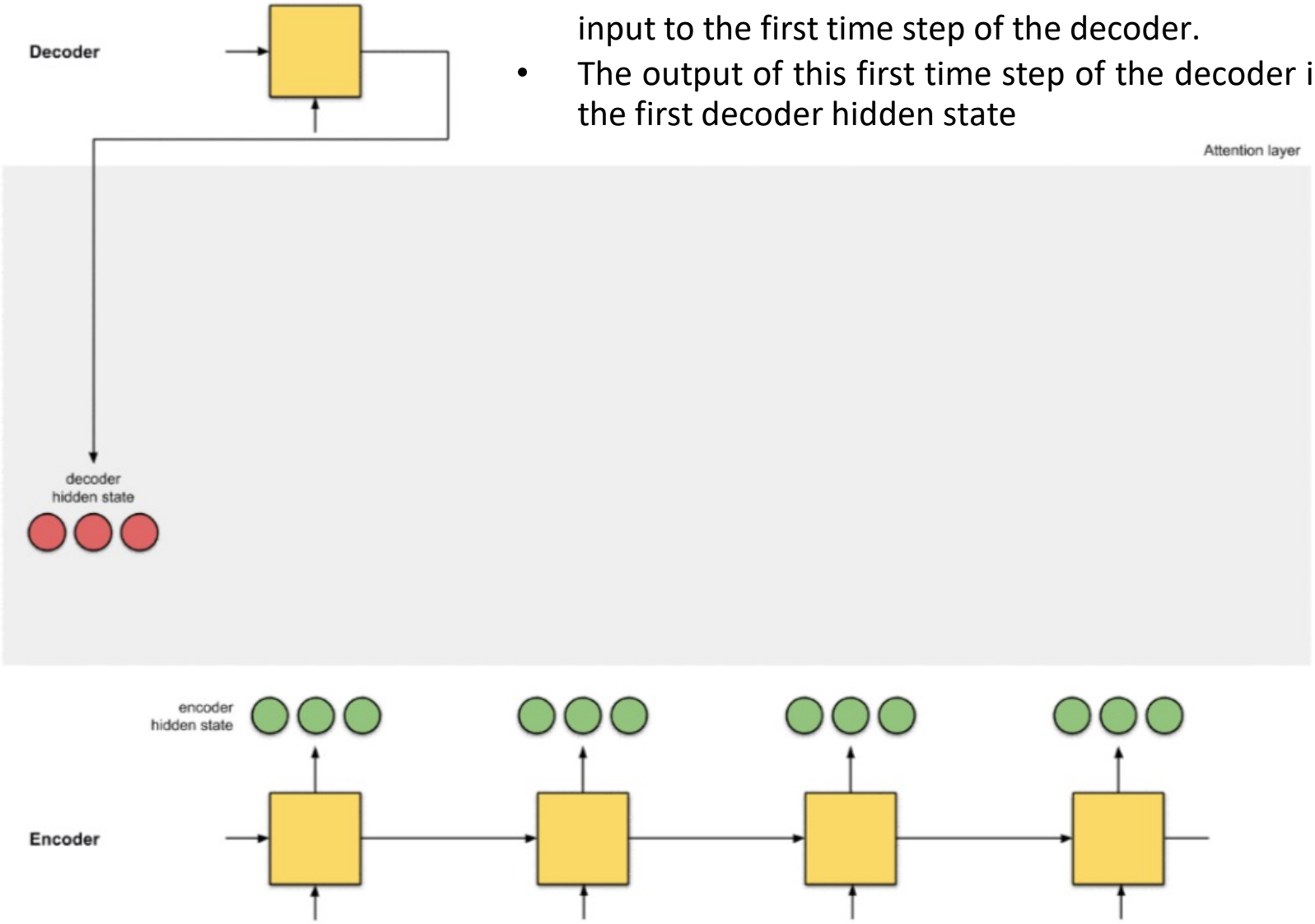
Alignment means matching segments of original text (input) with their corresponding segments of the translation (output).

Alignment for the French word ‘la’ is distributed across the input sequence but mainly on these 4 words: ‘the’, ‘European’, ‘Economic’ and ‘Area’. Darker purple indicates better attention scores



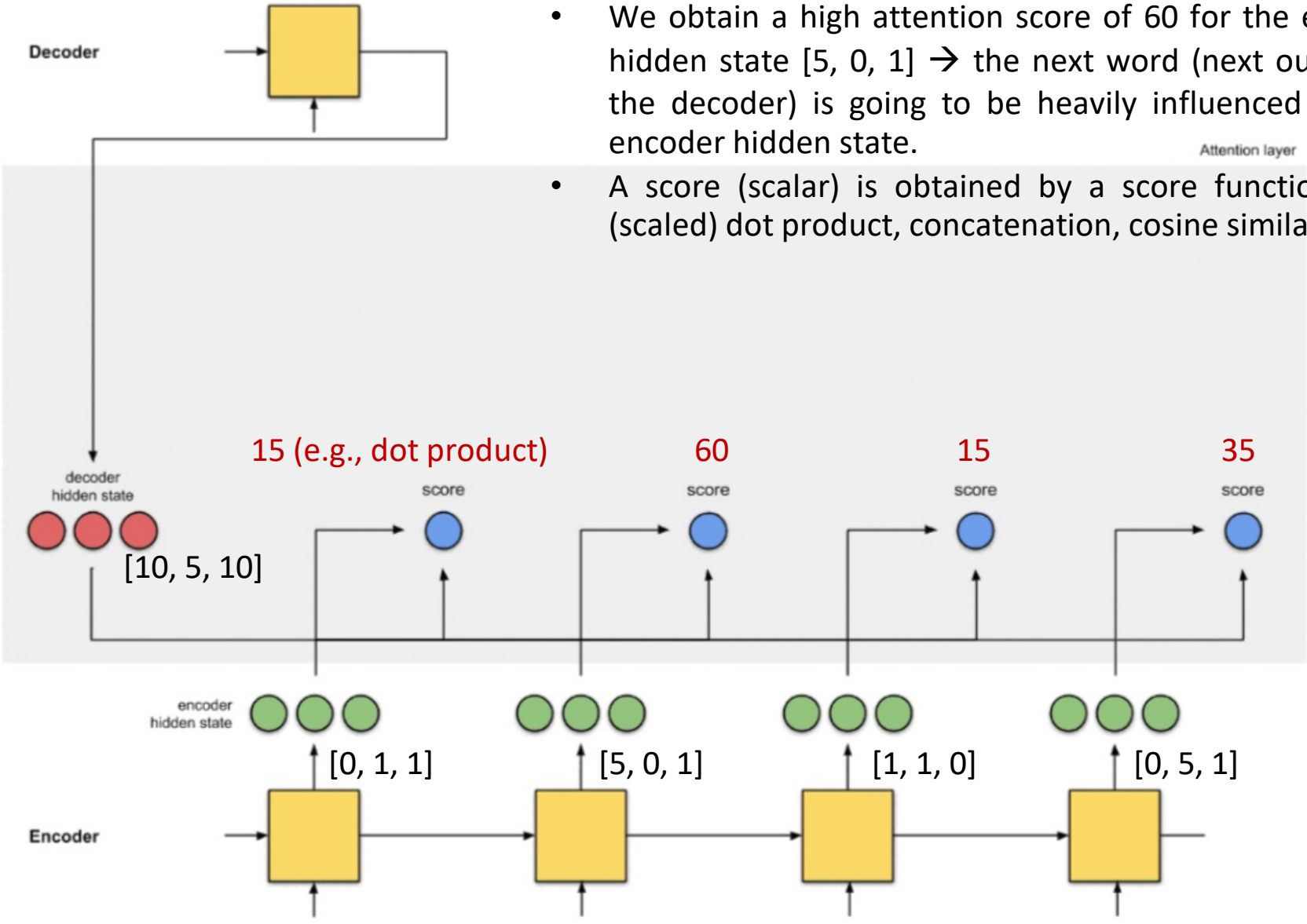
seq2seq with attention: (0) prepare hidden states

- The last consolidated encoder hidden state is fed as input to the first time step of the decoder.
- The output of this first time step of the decoder is called the first decoder hidden state



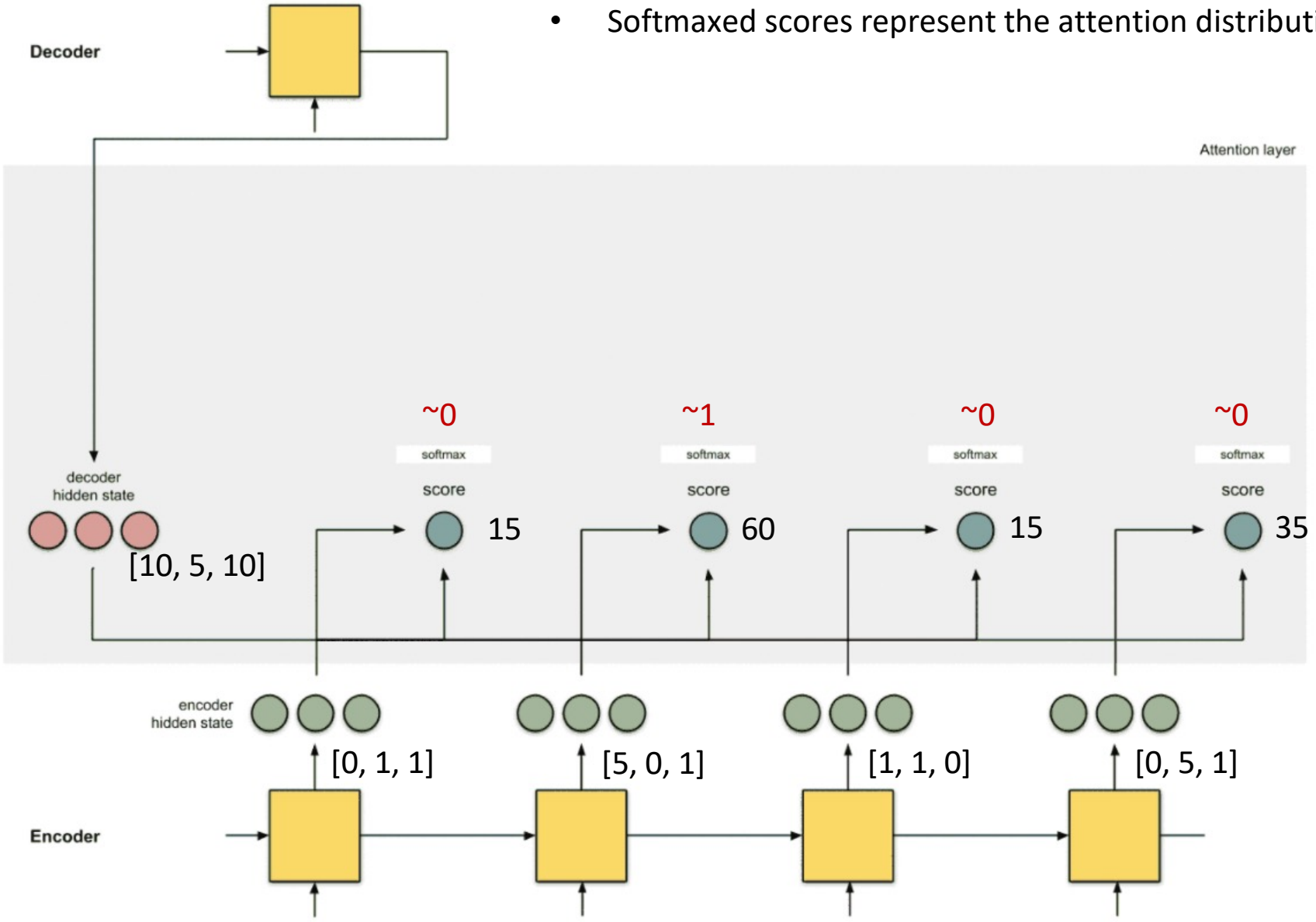
seq2seq with attention: (1) obtain a score for every hidden state

- We obtain a high attention score of 60 for the encoder hidden state [5, 0, 1] → the next word (next output by the decoder) is going to be heavily influenced by this encoder hidden state.
- A score (scalar) is obtained by a score function, e.g., (scaled) dot product, concatenation, cosine similarity…



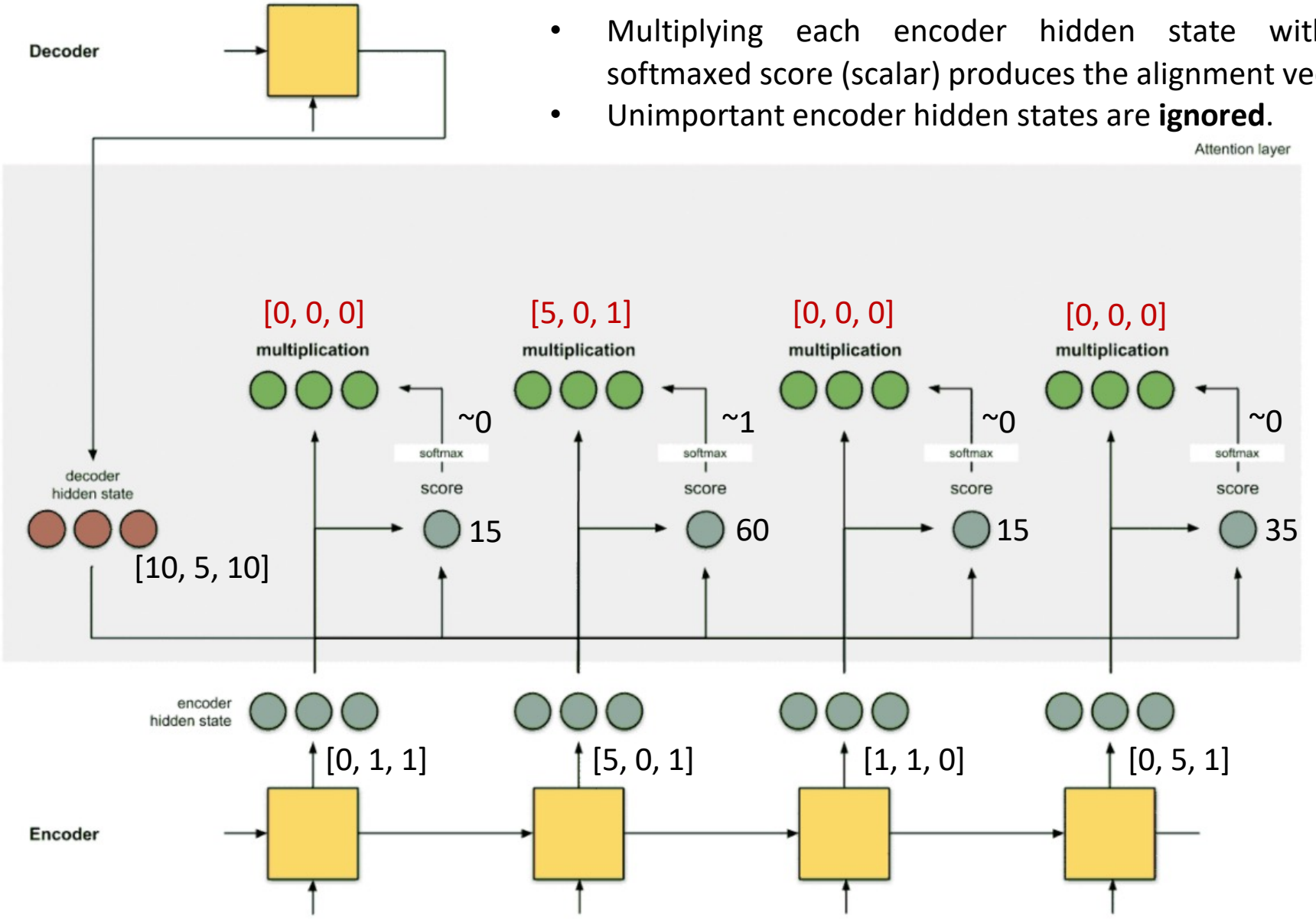
seq2seq with attention: (2) softmax the scores

- Softmaxed scores represent the attention distribution



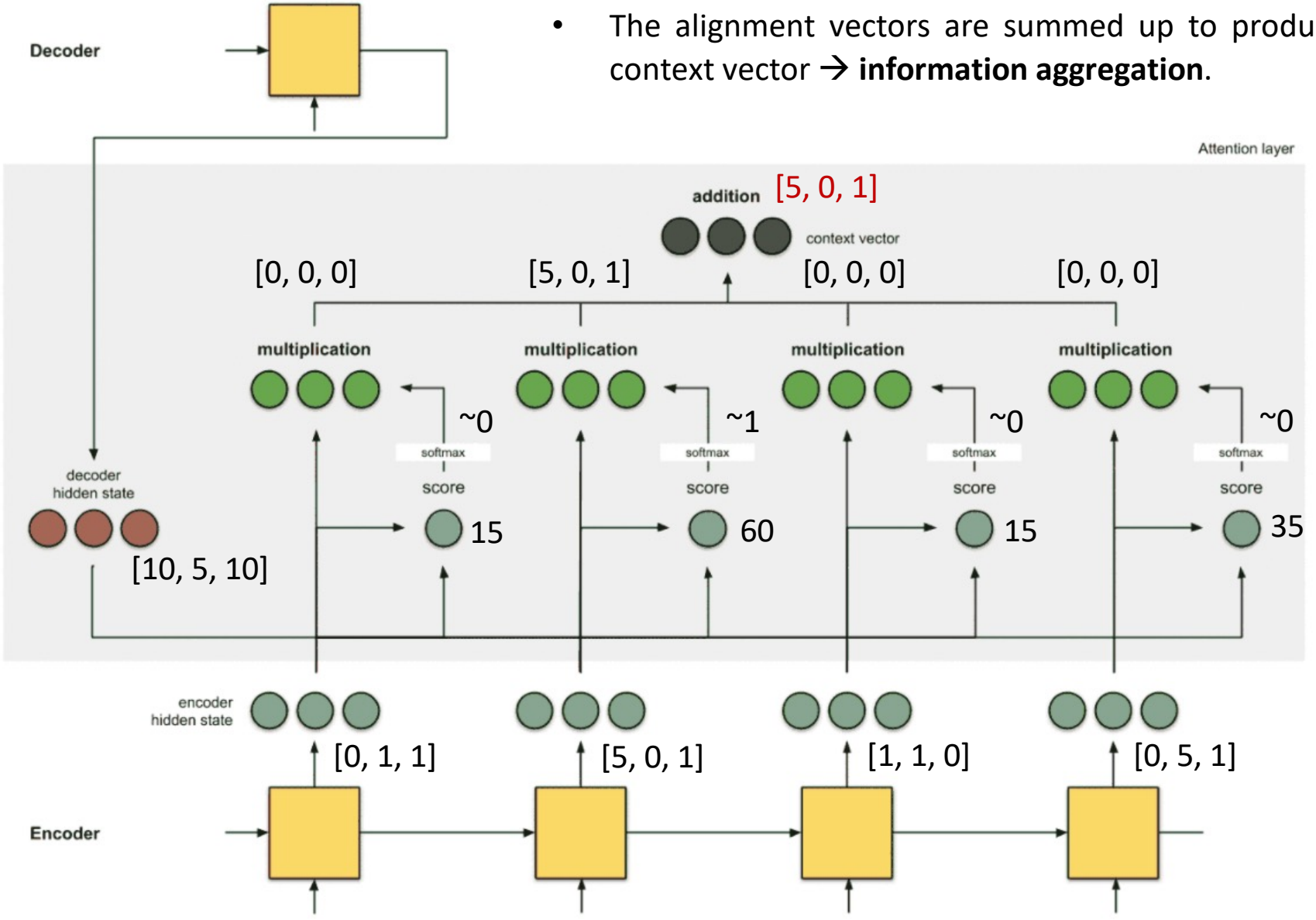
seq2seq with attention: (3) Multiply encoder hidden state and softmaxed score.

- Multiplying each encoder hidden state with its softmaxed score (scalar) produces the alignment vector.
- Unimportant encoder hidden states are **ignored**.



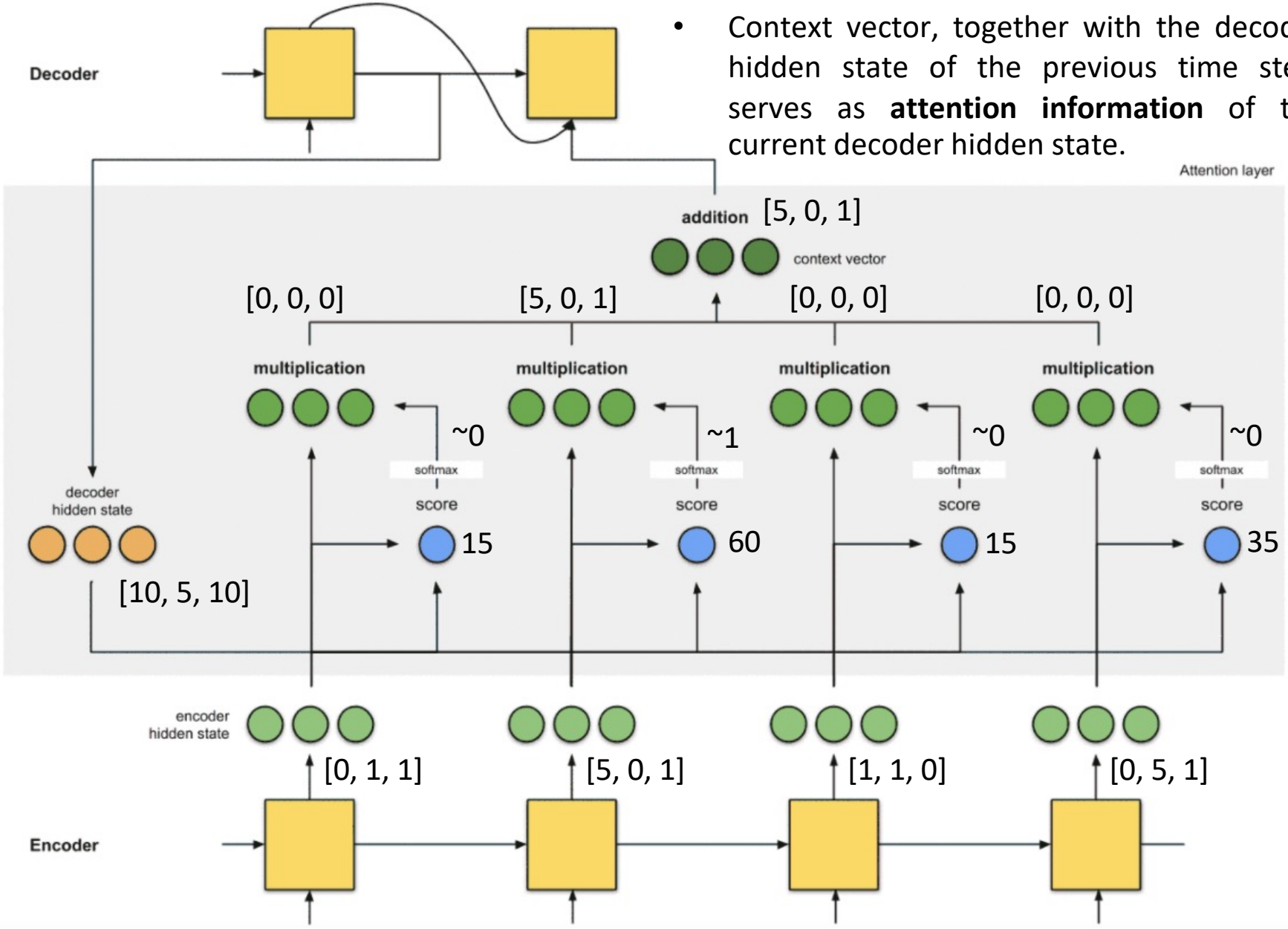
seq2seq with attention: (4) Sum up the alignment vector

- The alignment vectors are summed up to produce the context vector → **information aggregation**.



seq2seq with attention: (5) feed context vector to decoder

- Context vector, together with the decoder hidden state of the previous time step, serves as **attention information** of the current decoder hidden state.



seq2seq vs. *seq2seq with attention*

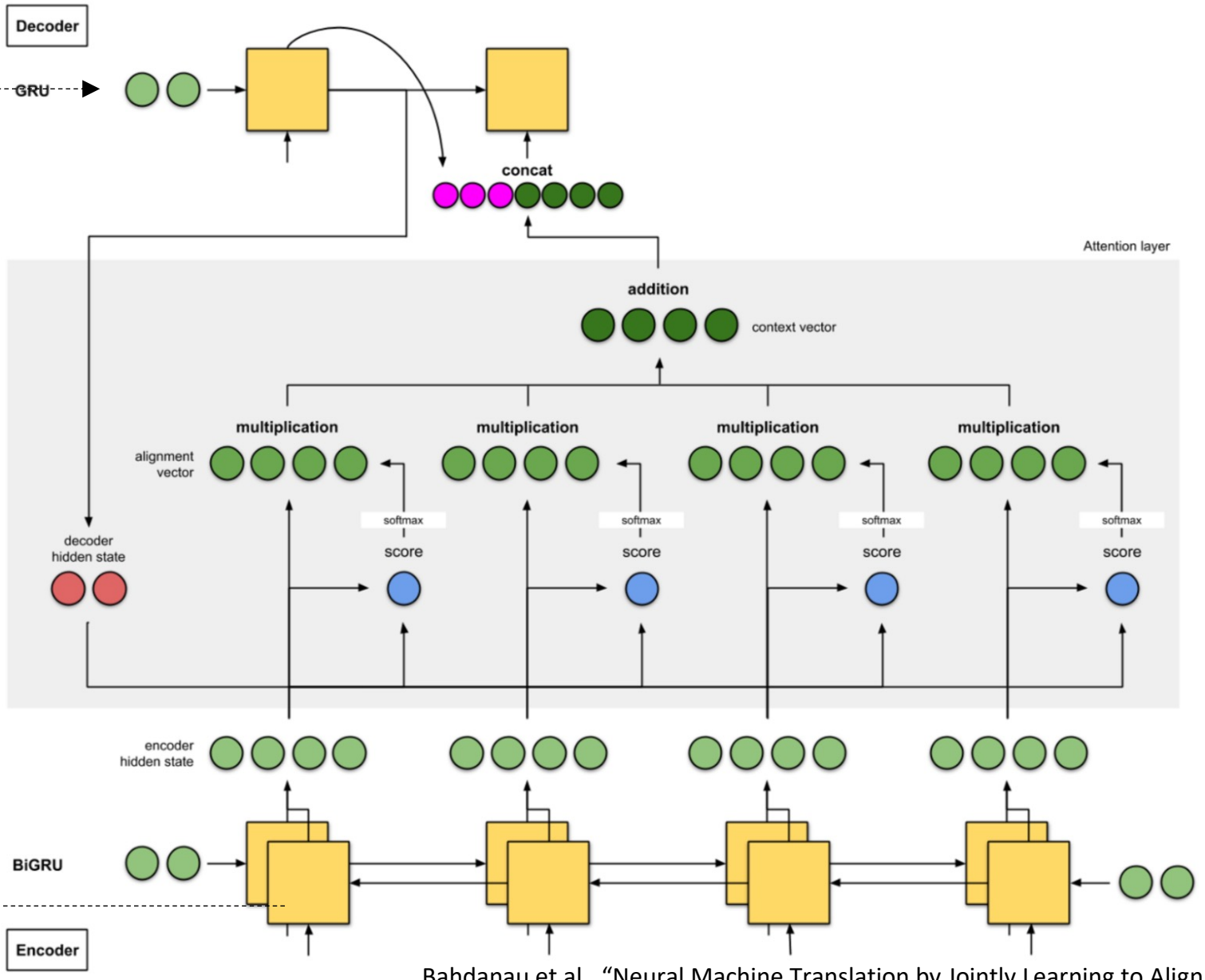
seq2seq:

- A translator reads the French text from start till the end. Once done, he starts translating to English word by word. It is possible that if the sentence is extremely long, he might have forgotten what he has read in the earlier parts of the text.

Seq2seq + attention:

- A translator reads the French text while **writing down the keywords** from the start till the end, after which he starts translating to English. While translating each French word, he makes use of the keywords he has written down.

seq2seq with bidirectional GRU encoder + attention



Bahdanau et al., “Neural Machine Translation by Jointly Learning to Align and Translate"

seq2seq with bidirectional GRU encoder + attention

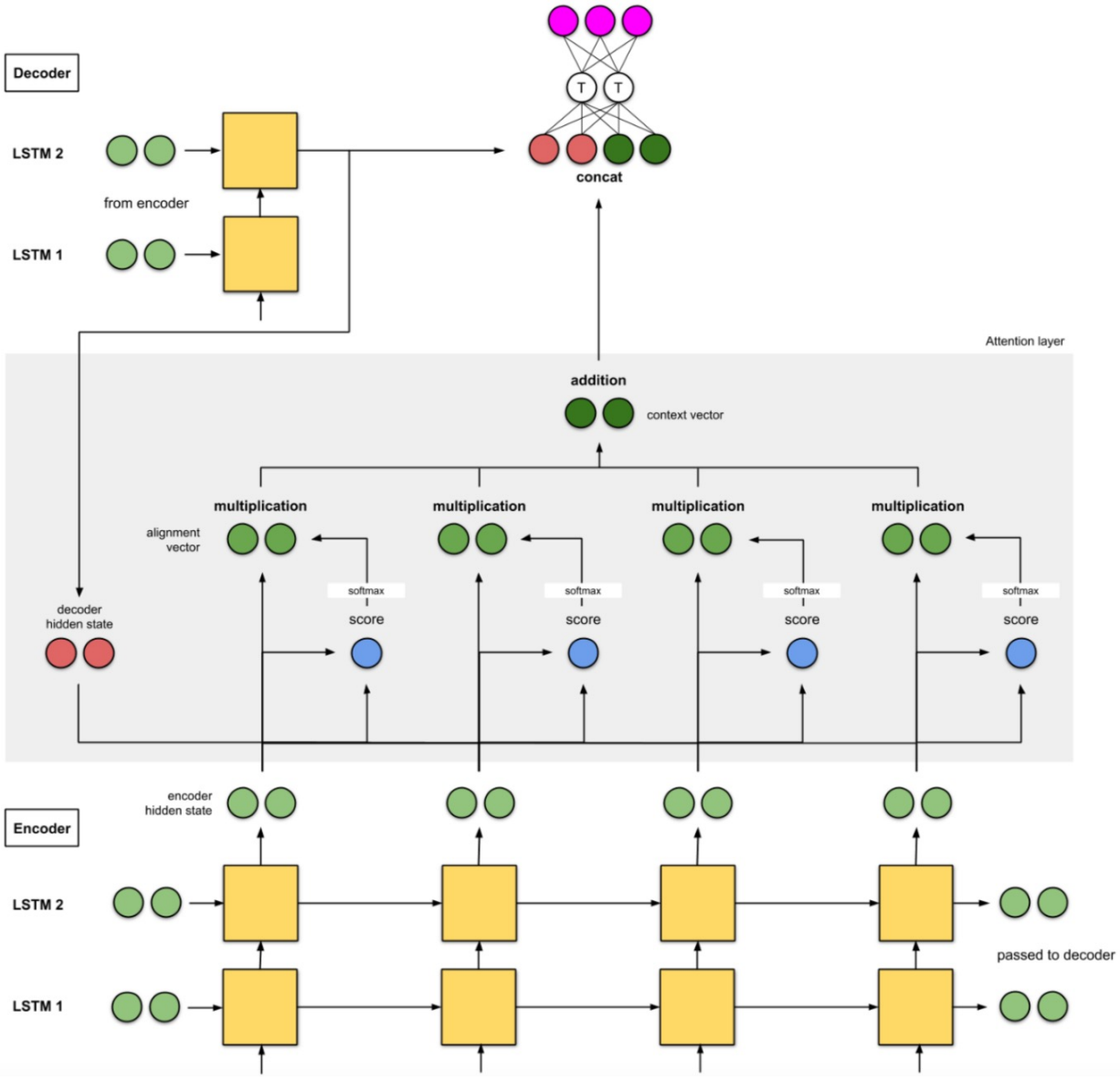
Architecture:

- The encoder is a bidirectional (forward and backward) gated recurrent unit (**BiGRU**). The decoder is a GRU whose initial hidden state is a vector modified from the last hidden state from the backward encoder GRU.
- The score function in the attention layer is the additive/concat. The input to the next decoder step is the concatenation between the generated word from the previous decoder time step (pink) and context vector from the current time step (dark green).

Intuition:

- Translator A reads the French text while **writing down the keywords**. Translator B (who takes on a senior role because he has an extra ability to translate a sentence from reading it **backwards**) reads the same French text from the last word to the first, while **writing down the keywords**.
- These two **regularly discuss** about every word they read thus far. Once done reading this French text, Translator B is then tasked to translate the French sentence to English word by word, based on the discussion and the consolidated keywords that the both of them have picked up.
- Translator A is the forward RNN, Translator B is the backward RNN of the encoder.

seq2seq with 2-layer stacked encoder + attention



Luong et al., “Effective Approaches to Attention-based Neural Machine Translation"

seq2seq with 2-layer stacked encoder + attention

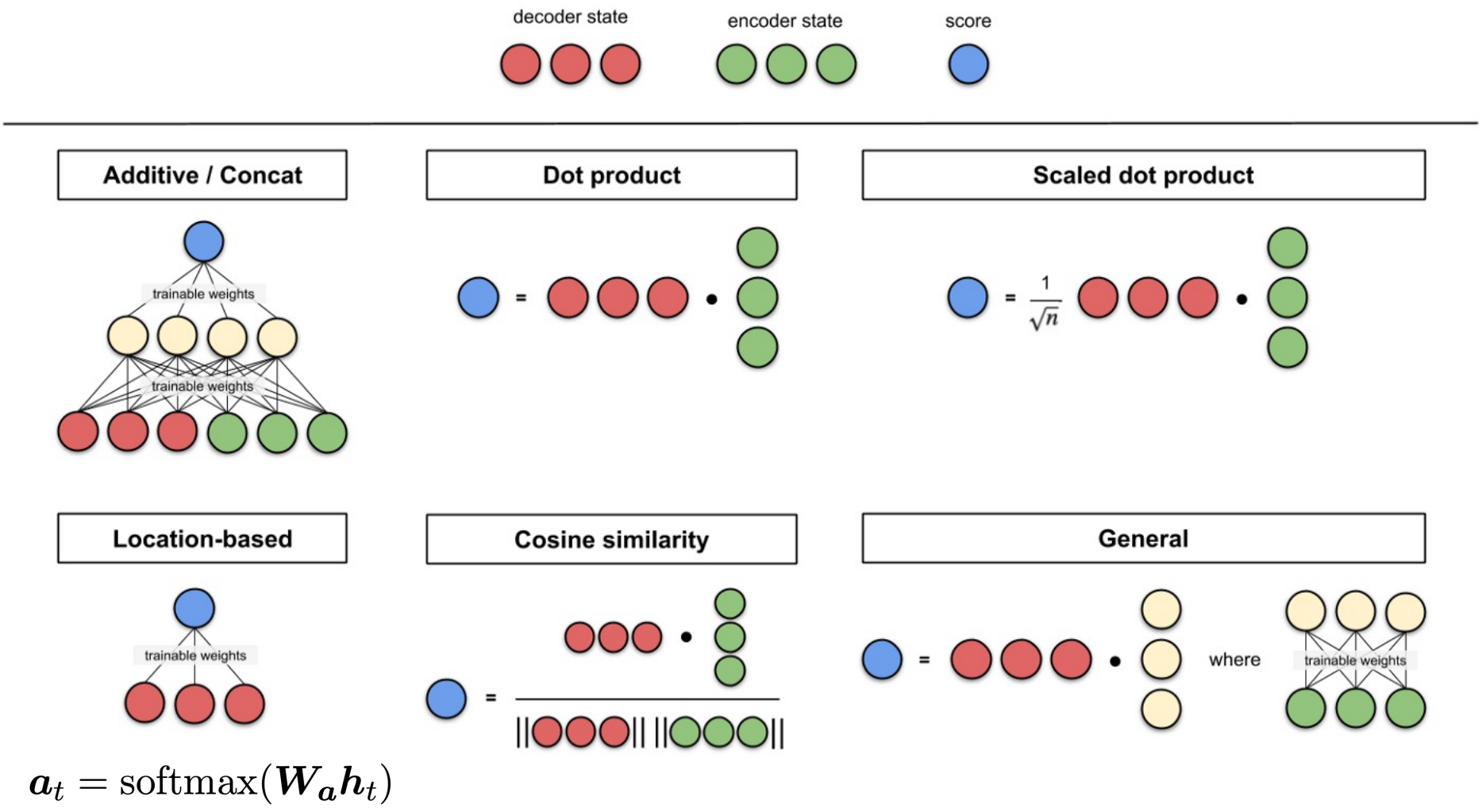
Architecture:

- The encoder is a **two-stacked long short-term memory** (LSTM) network. The decoder also has the same architecture, whose initial hidden states are the last encoder hidden states.
- The score functions they experimented were (i) additive/concat, (ii) dot product, (iii) location-based, and (iv) 'general'.
- The concatenation between output from current decoder time step, and context vector from the current time step are fed into a feed-forward neural network to give the *final output* (pink) of the current decoder time step.

Intuition:

- Translator A reads the French text while writing down the keywords. Likewise, Translator B (who is more senior than Translator A) also reads the same French text, while jotting down the keywords.
- The junior **Translator A has to report to Translator B** at every word they read. Once done reading, the both of them translate the sentence to English together word by word, based on the consolidated keywords that they have picked up.

Choice of score function

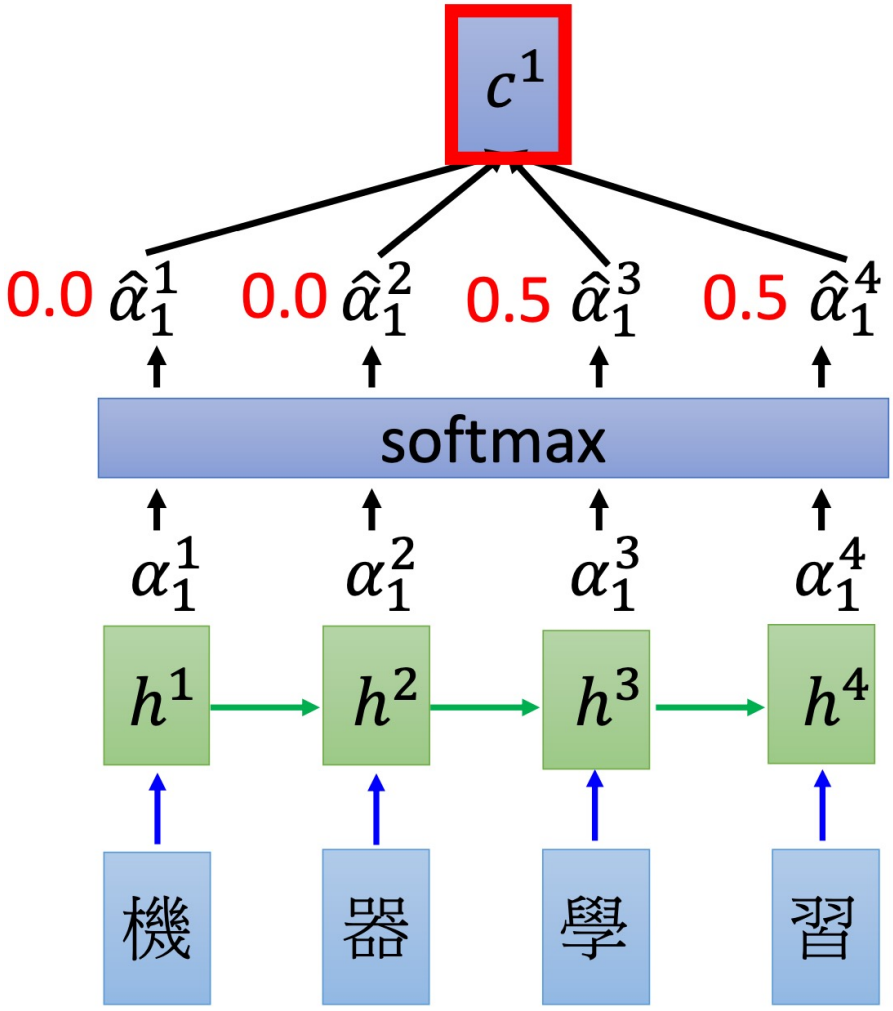


$$\boldsymbol{a}_t = \text{softmax}(\boldsymbol{W}_{\boldsymbol{a}} \boldsymbol{h}_t)$$

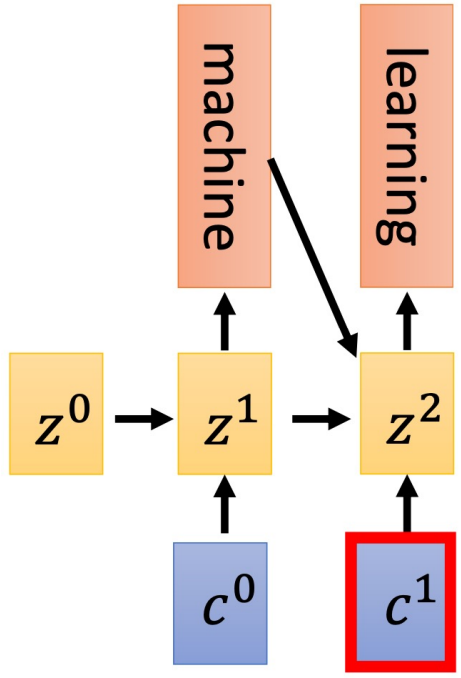
Location-based: Alignment scores are computed from solely the decoder (target) hidden state
→ using only the location (i.e., time step) of the decoder hidden state.

Summary of Attention mechanism

Encoder



Decoder



$$c^1 = \sum \hat{\alpha}_1^i h^i$$

$$= 0.5h^3 + 0.5h^4$$

Image caption generation

Introduction: input an **image**, but output a **sequence of words.**

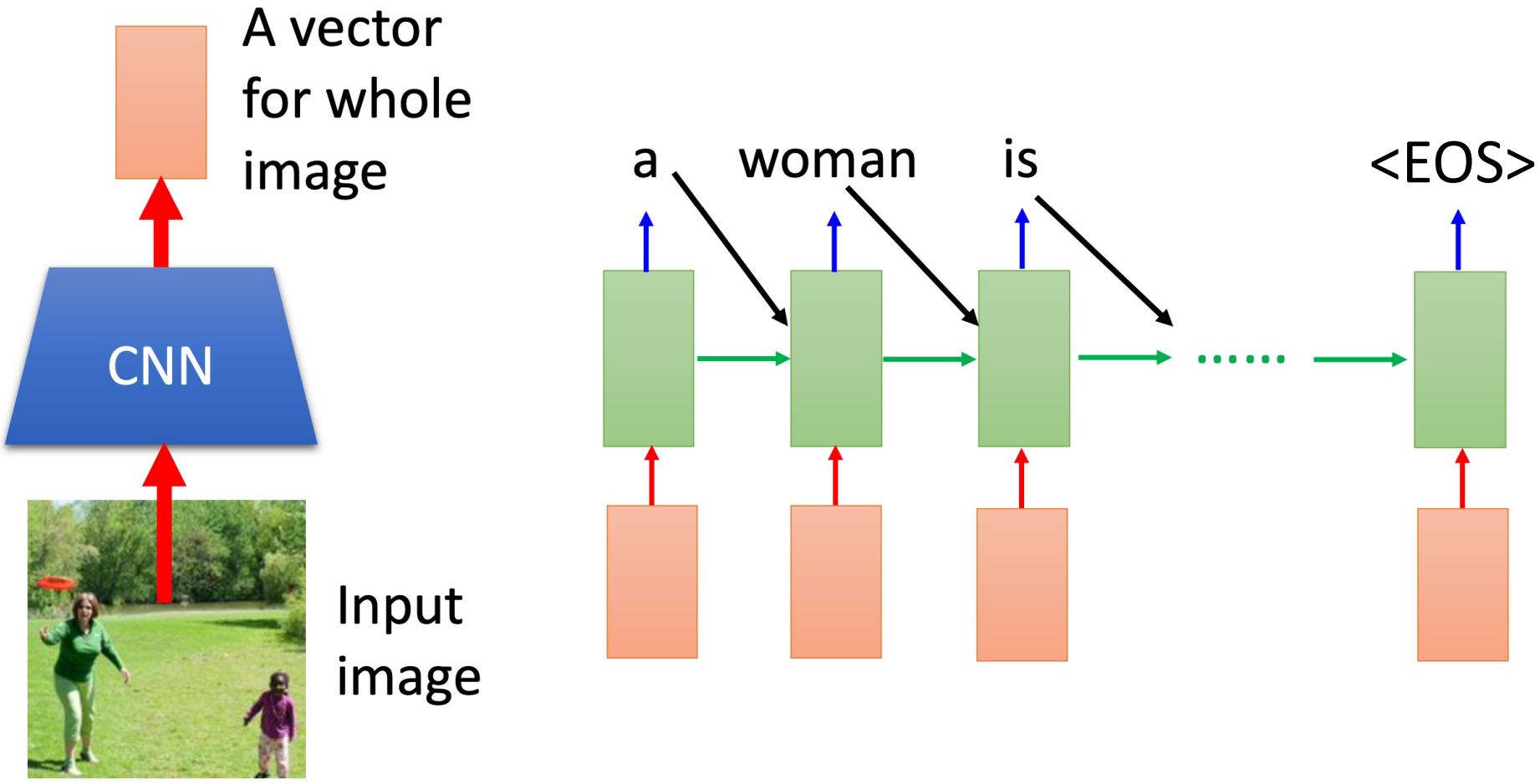


Image caption generation

Introduction: input an **image**, but output a **sequence of words.**

Idea: crop the image to pieces → attention on the **image pieces**.

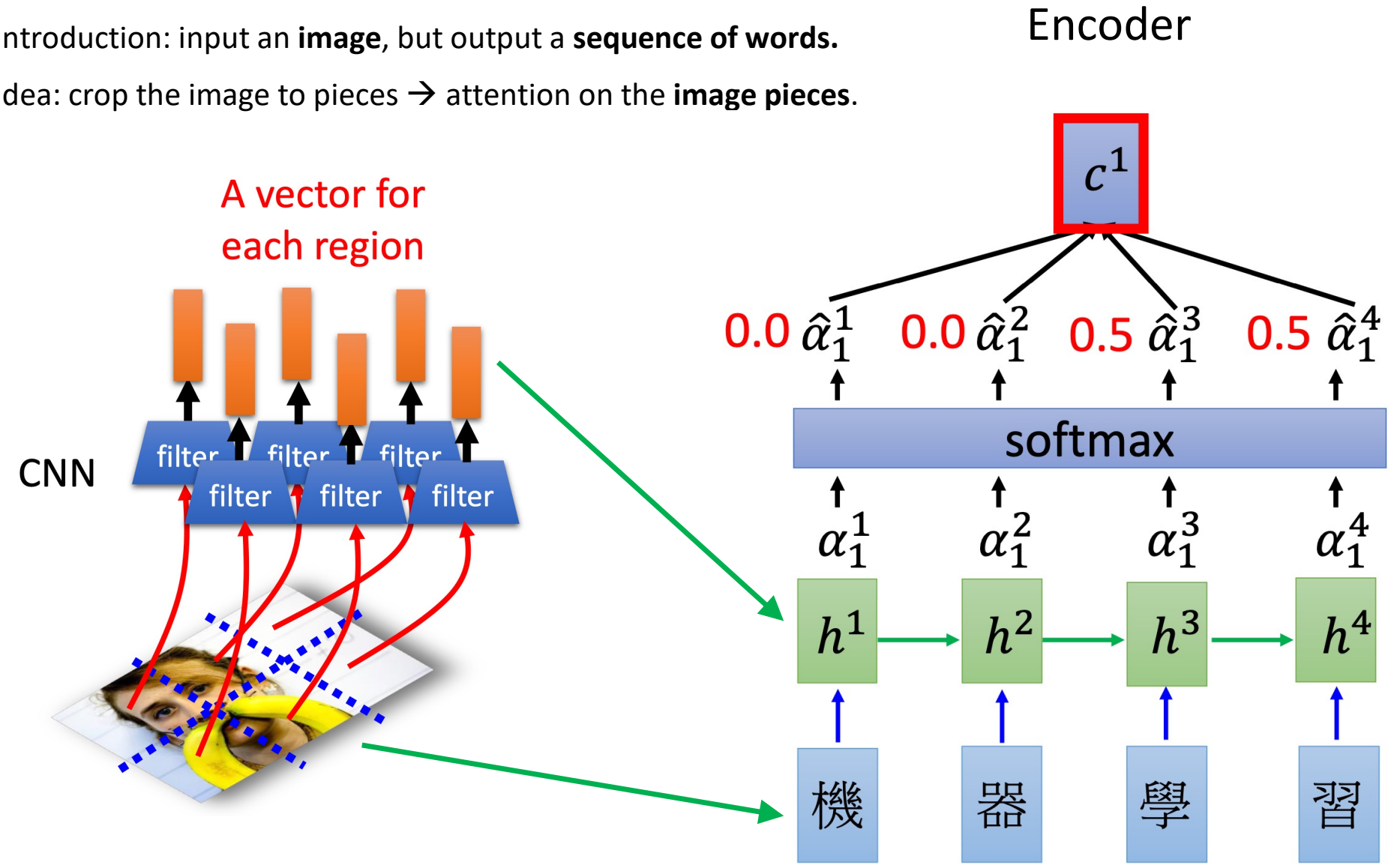
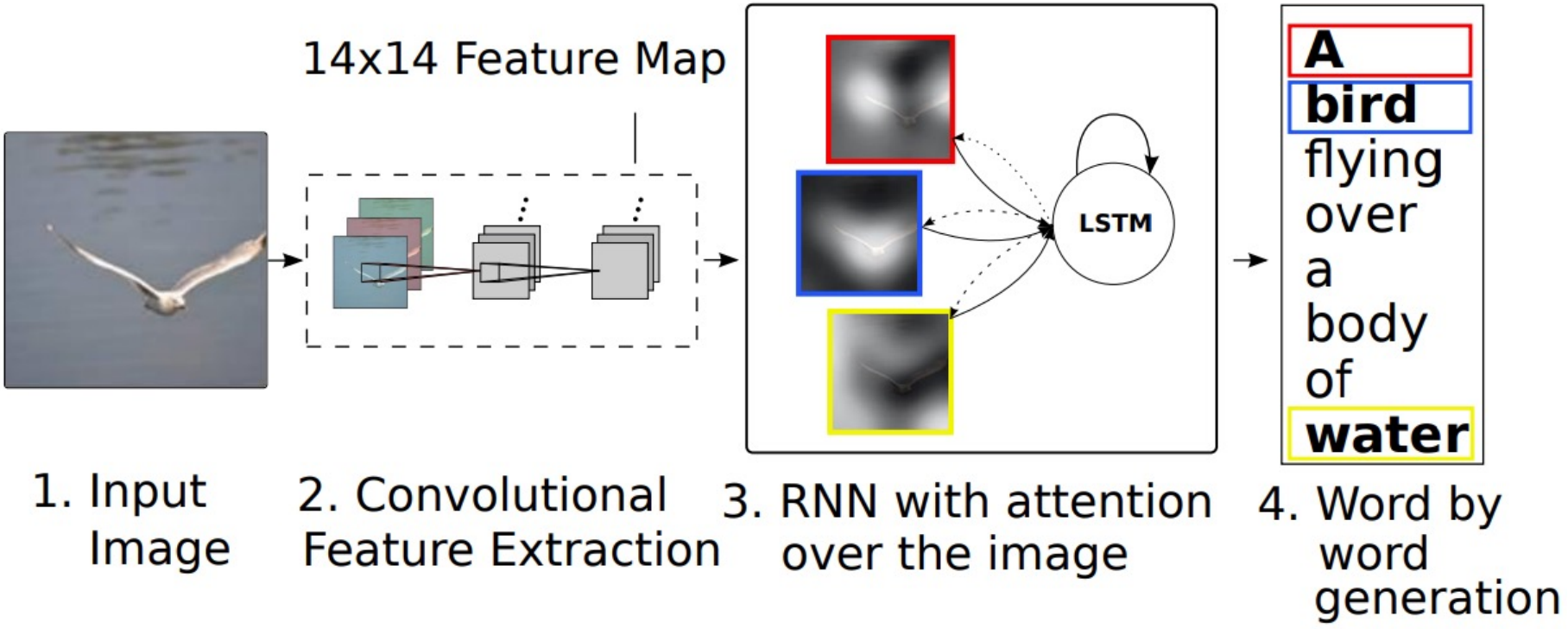


Image caption generation

Introduction: input an **image**, but output a **sequence of words.**

Idea: crop the image to pieces → attention on the **image pieces**.

→ Cross-domain adaptation of Computer Vision and NLP.



Kelvin et al., “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention"

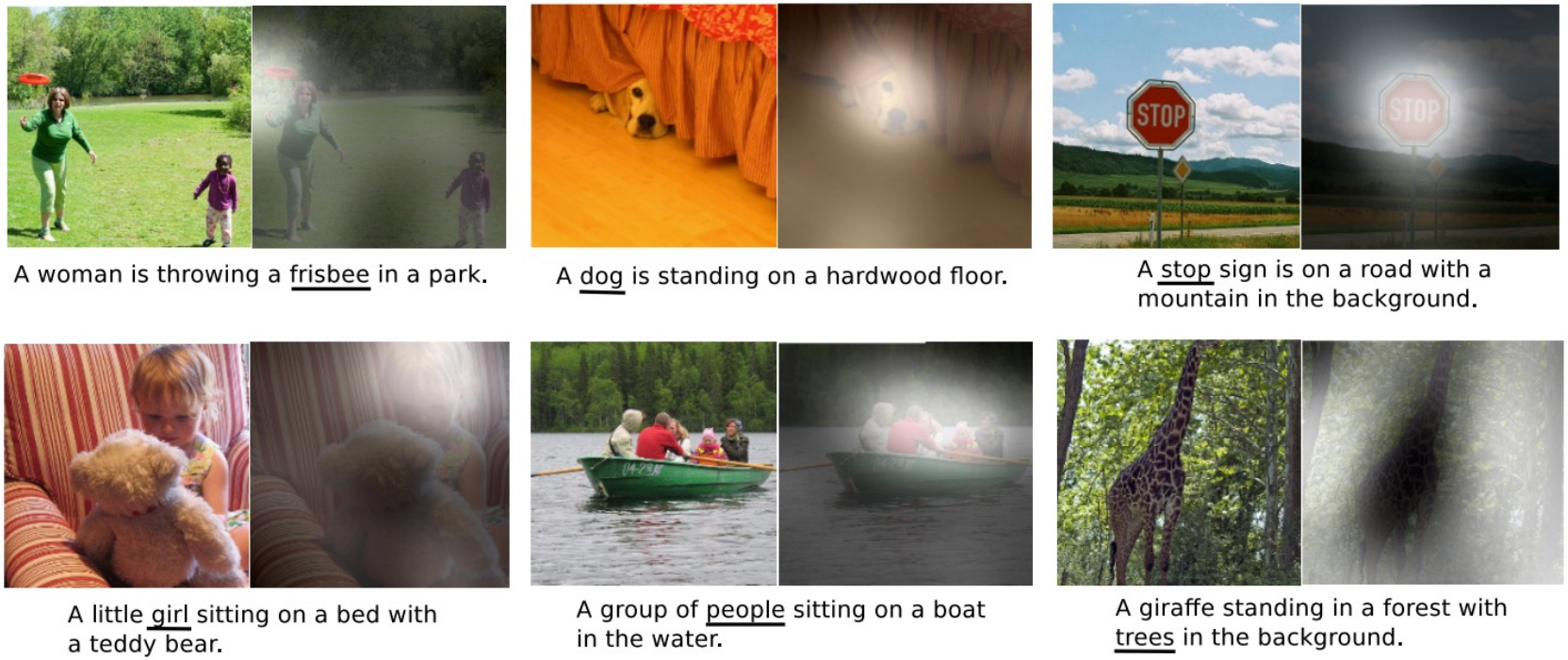
Image caption generation

Introduction: input an **image**, but output a **sequence of words.**

Idea: crop the image to pieces → attention on the **image pieces**.

→ Cross-domain adaptation of Computer Vision and NLP.

Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)

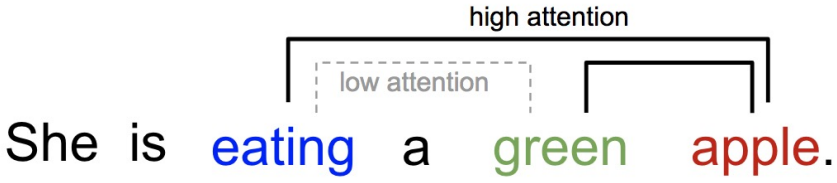


Kelvin et al., “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention"

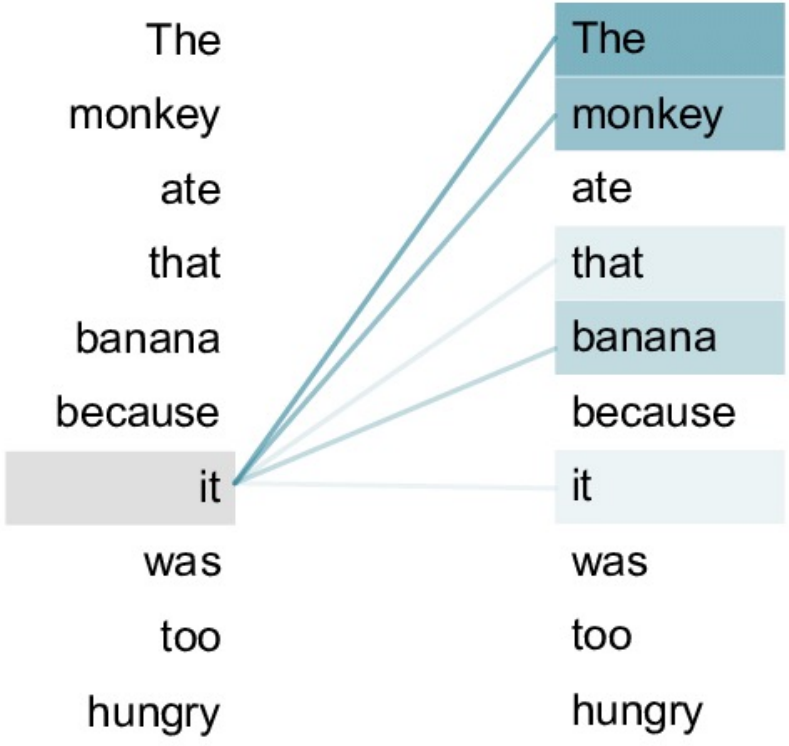
Self-attention

Remind: Attention is based on the **decoder** hidden states and **encoder** hidden states. Could attention happen within the scope of **only encoder** hidden states?→ Yes, **self-attention**.

Introduction: Self-attention, also known as intra-attention, is an attention mechanism relating **different positions of a single sequence** in order to compute a **representation of the same sequence**. It has been shown to be very useful in machine reading, abstractive summarization, or image description generation.



One word "attends" to other words in the same sentence differently.



Self-attention: (1) prepare input; initialize query, key and value

Self-attention

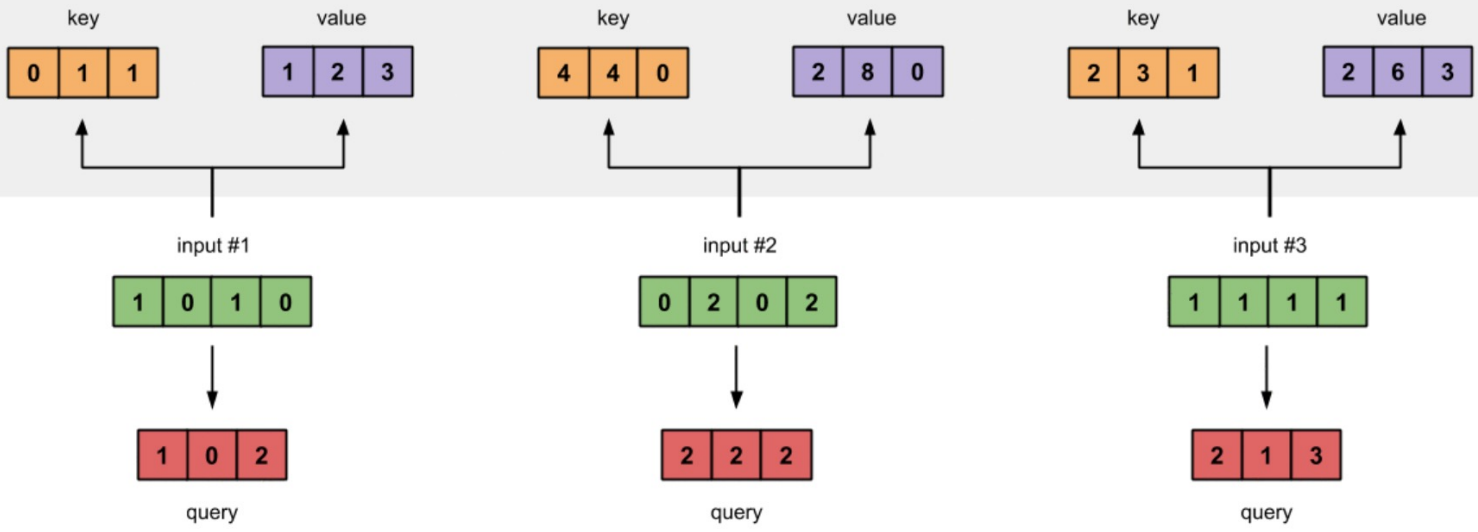
Inputs are embedding vectors.

Every input has three representation: **key**, **query** and **value**.

- query [3,1] = input [1,4] * matrix_query [4,3]
- key [3,1] = input [1,4] * matrix_key [4,3]
- value [3,1] = input [1,4] * matrix_value [4,3]

The matrices are **learnable** weights and **initialized** → help derive initial query, key and value.

Value is considered as an **updated embedding** of inputs adapted to the use of key and query.

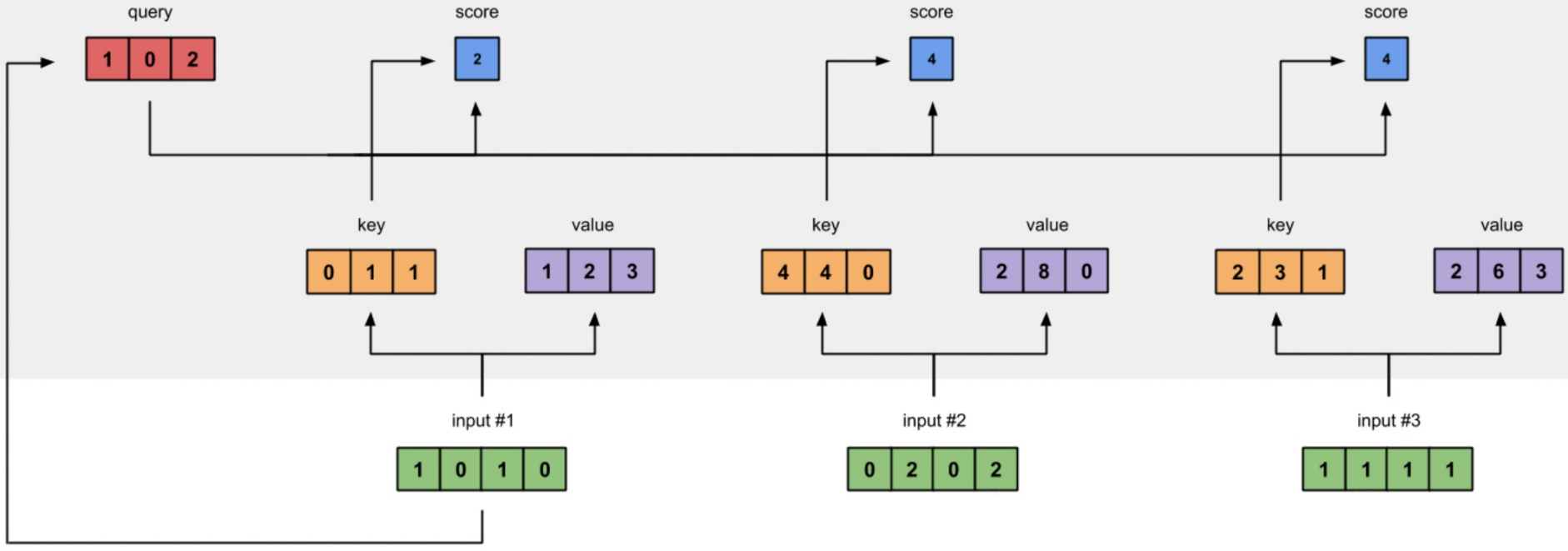


Self-attention: (2) - calculate attention scores if Input #1

Consider the case of **Input #1**:

- Score of Input #1: **dot product** between Input #1's **query with all keys**, including itself → motivation: how much attention the Input #1 pays to the other inputs → attention scores [[1,0,2]•[0,1,1], [1,0,2]•[4,4,0], [1,0,2]•[2,3,1]] = [2,4,4].

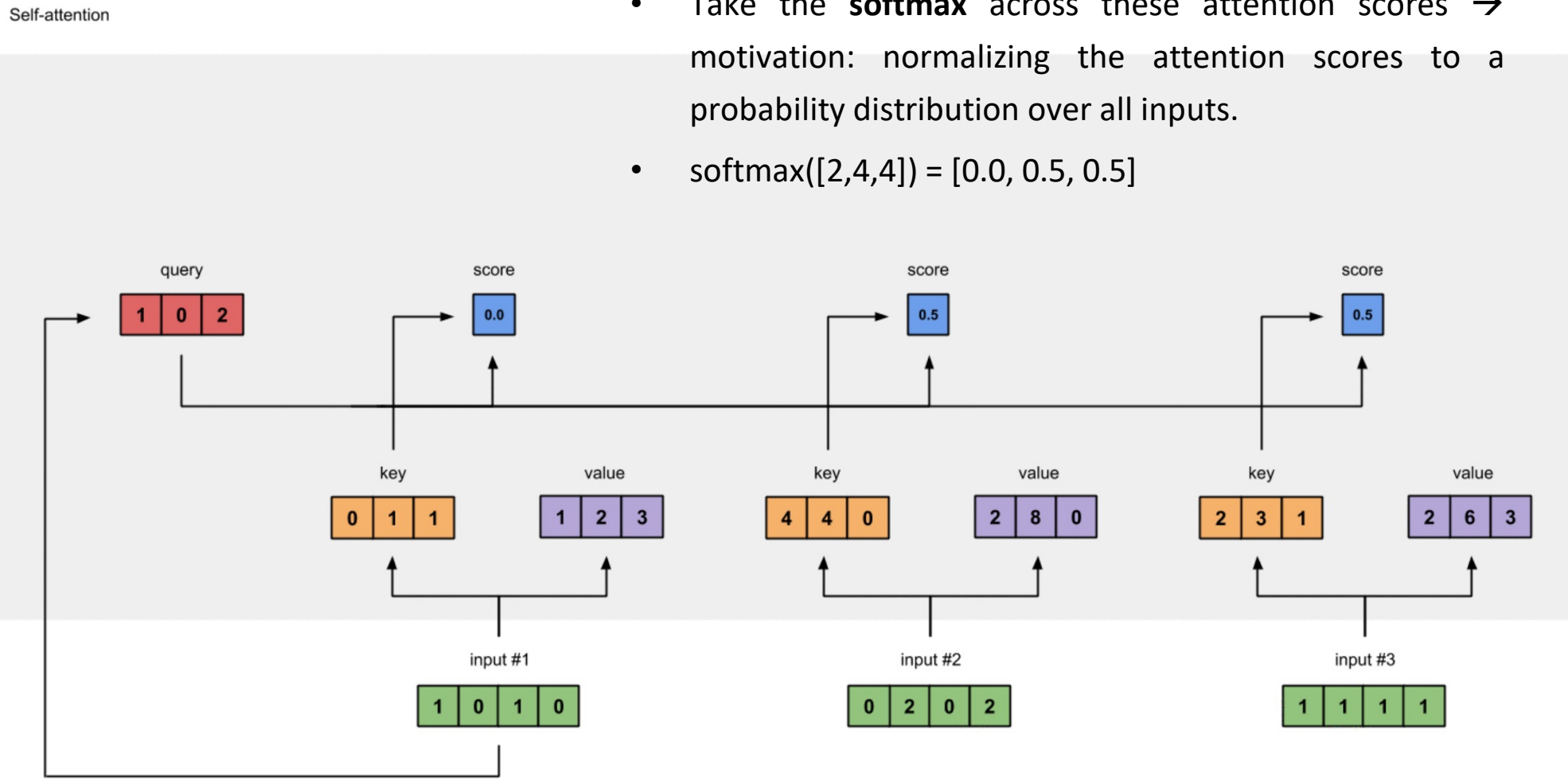
Self-attention



Self-attention: (3) - softmax attention scores

Consider the case of **Input #1:**

- Take the **softmax** across these attention scores → motivation: normalizing the attention scores to a probability distribution over all inputs.
- softmax([2,4,4]) = [0.0, 0.5, 0.5]

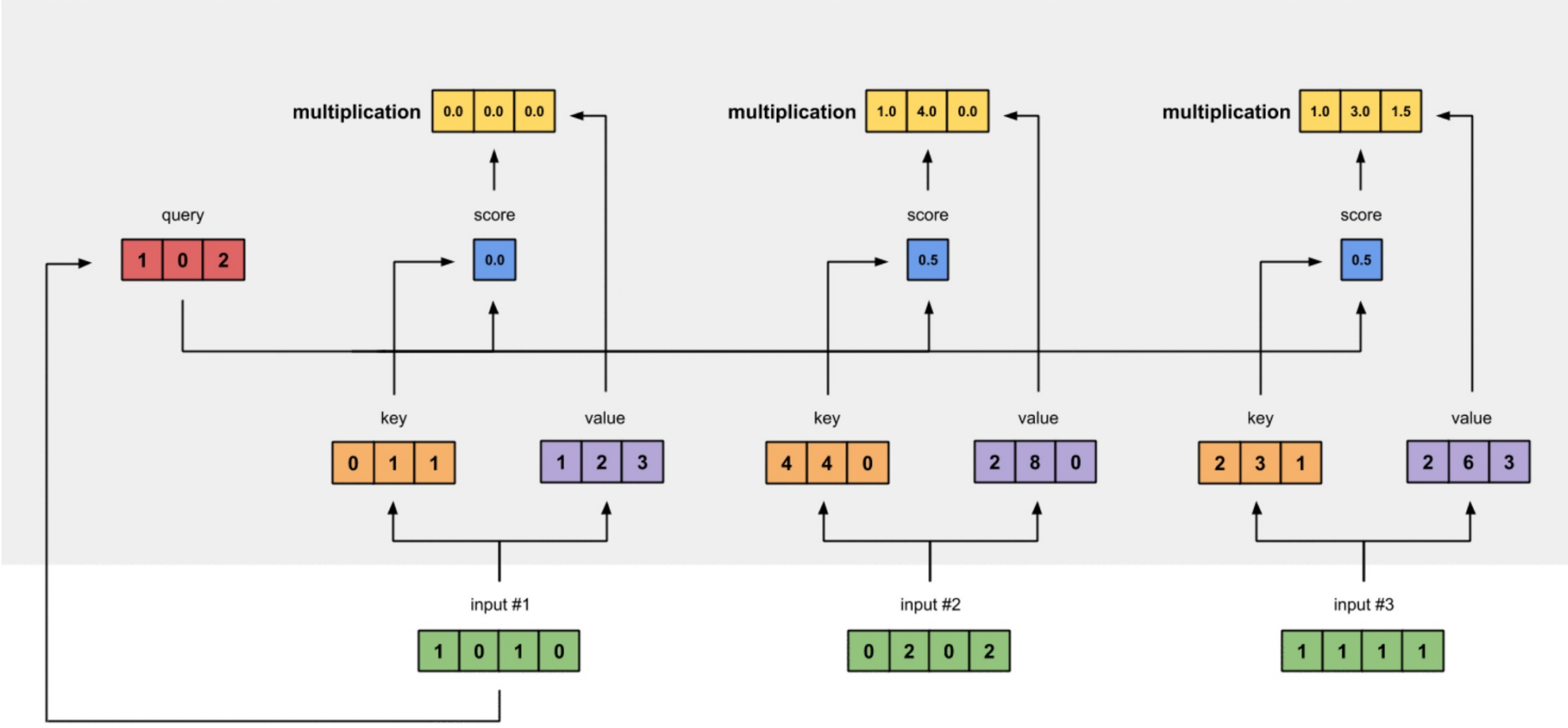


Self-attention: (4) - multiply scores with values

Consider the case of **Input #1**:

- Derive **weighted value** representation from multiplying the score with the value → This results in 3 alignment vectors (weighted values).

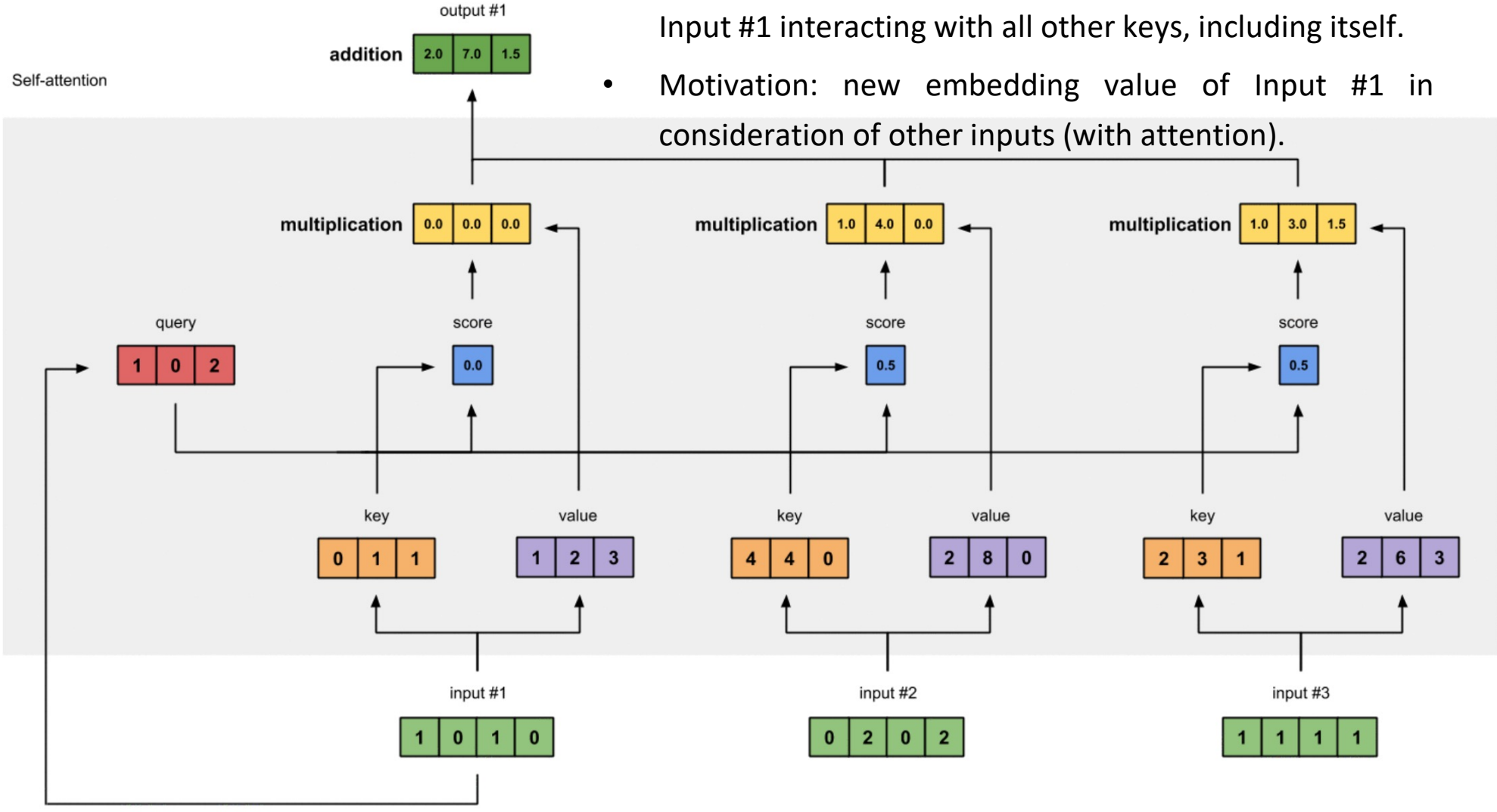
Self-attention



Self-attention: (5) - sum weighted values to get output

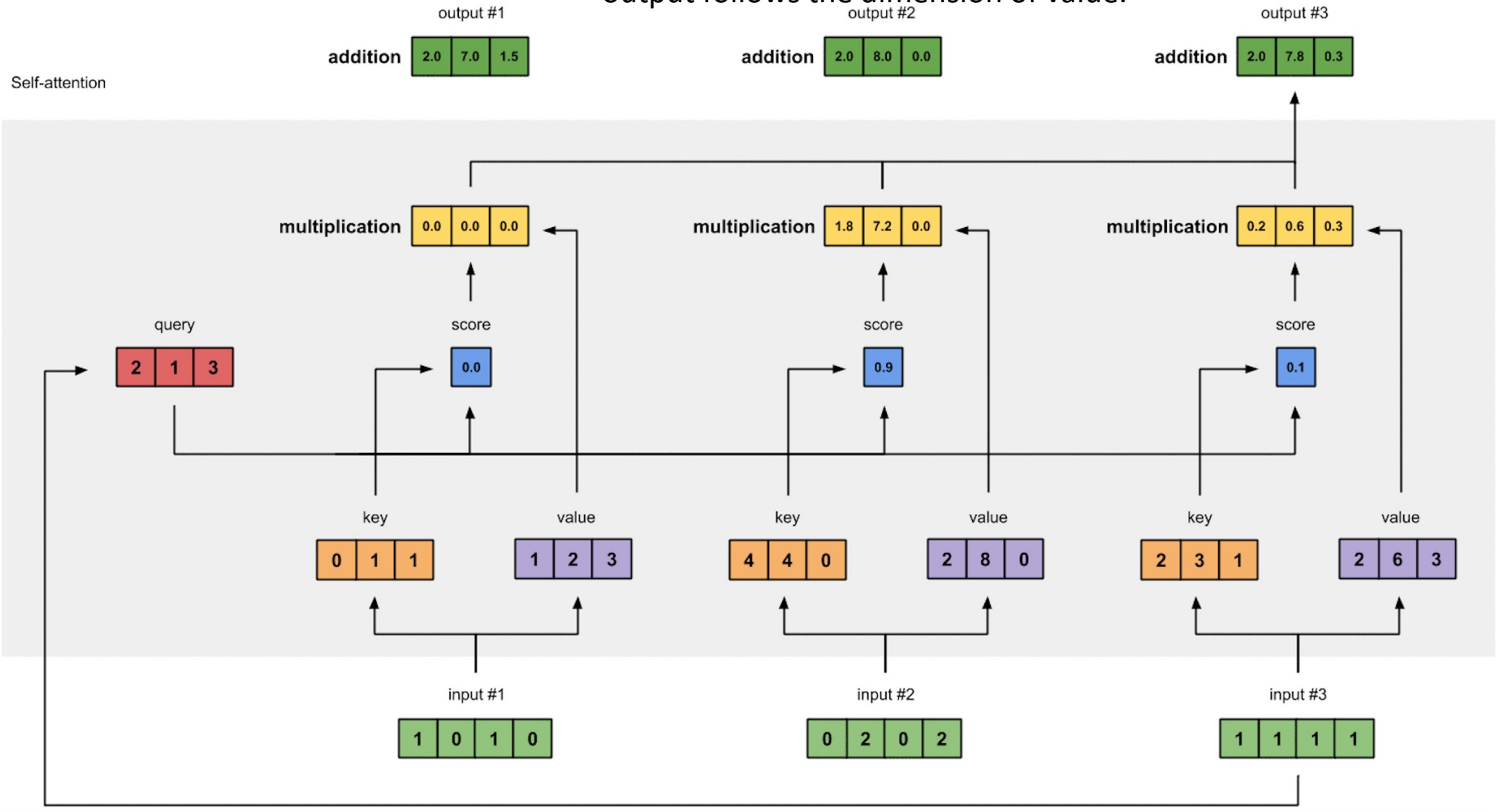
Consider the case of **Input #1**:

- Output #1 is based on the query representation from Input #1 interacting with all other keys, including itself.
- Motivation: new embedding value of Input #1 in consideration of other inputs (with attention).



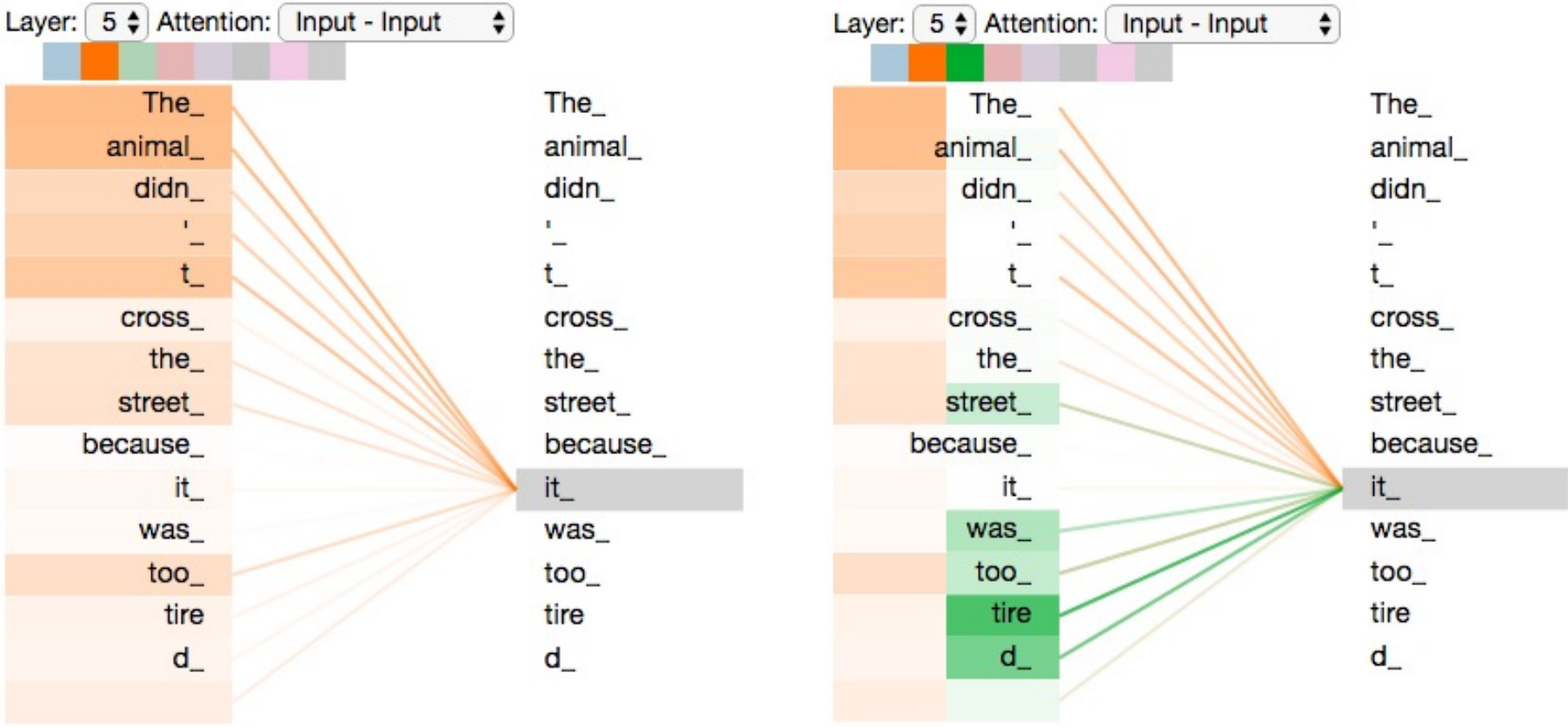
Self-attention: (6) - repeat for Input #2 and Input #3

Note: dimension of value may be different from query and key dimension (which are the same due dot product) → output follows the dimension of value.



Multihead self-attention

Idea: **repeating the self-attention** mechanism multiple times the model can learn to separate different kinds of useful **semantic information** onto different channels (or heads) → used in **Transformer Network**.



Transformer - Introduction

Weakness of LSTM in sequence-to-sequence:

- **Sequential processing**: sentences must be processed words by words → cannot be parallelized
- **No attention**: unable to deal with long sentence.

→ **LSTM is slow and inefficient.**

Transformer brings a great *improvement*:

- **Non sequential**: sentences are processed as a whole rather than word by word.
- **Self Attention**: this is the newly introduced 'unit' used to compute similarity scores between words in a sentence.
- **Positional embeddings**: another innovation introduced to replace recurrence. The idea is to use fixed or learned weights which encode information related to a specific position of a token in a sentence.

→ Transformer has **attention mechanism** and processes **sequential data as a whole**.

→ **Transformer is fast and efficient.**

Transformer - Architecture overview

Ashish et al., “Attention Is All You Need”, NIPS’17

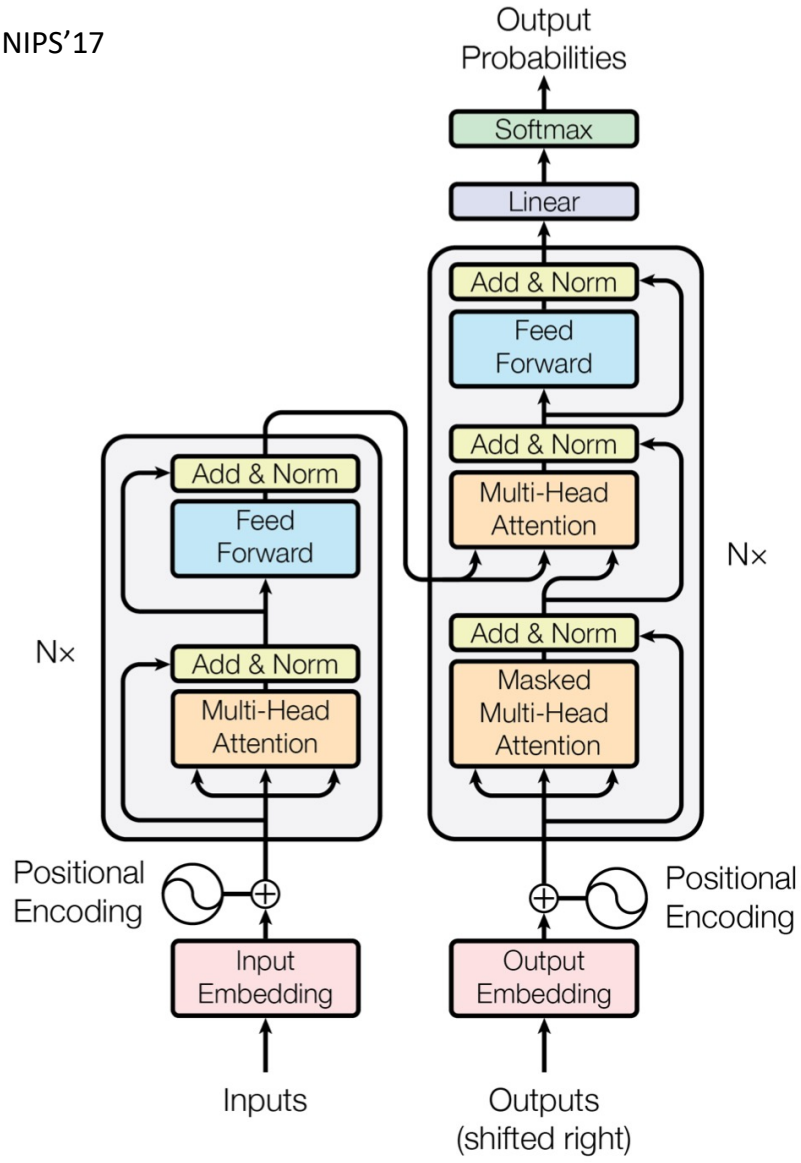
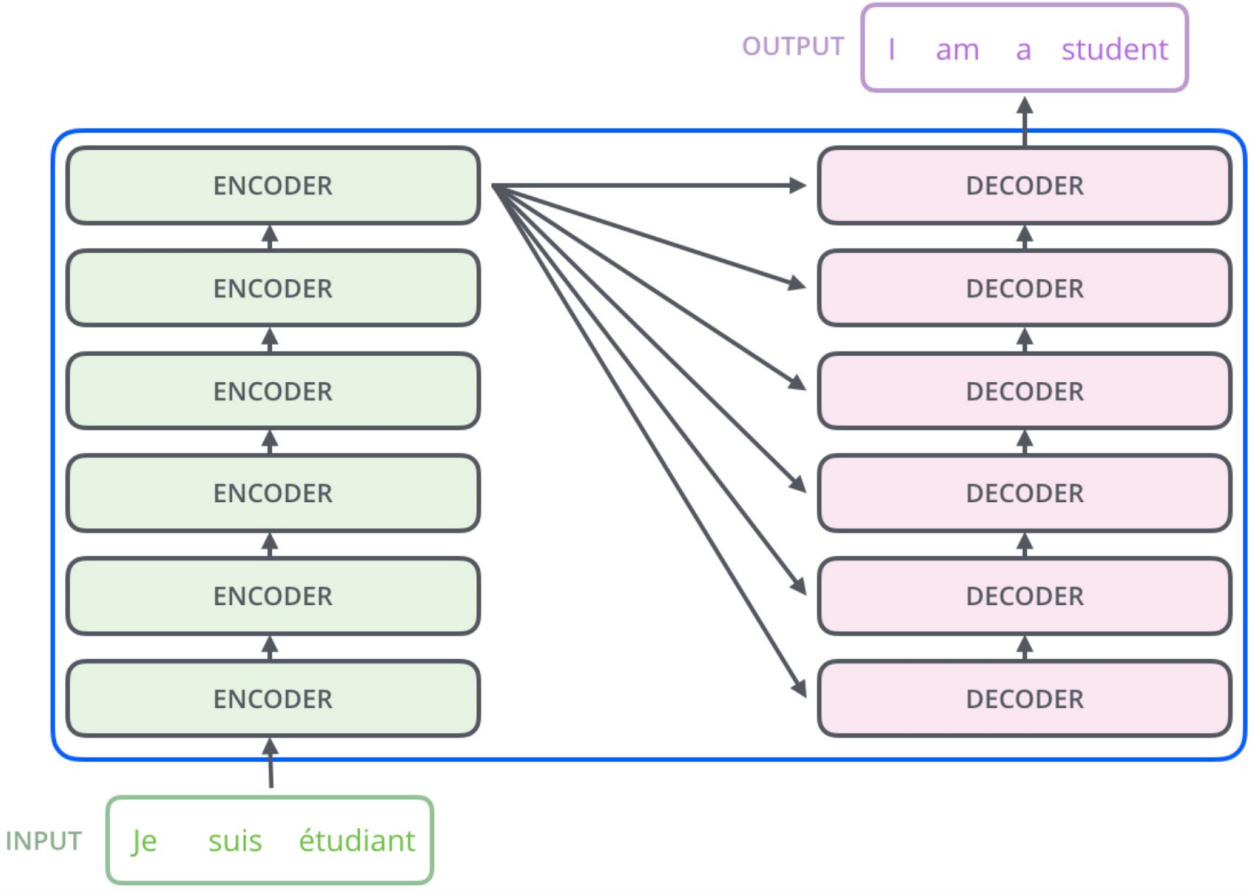


Figure 1: The Transformer - model architecture.

Transformer - Encoder and decoder

- Encoding component is a **stack of N encoders** (six in the paper). The encoders have identical structures (yet they do not share weights).
- Decoding component is a **stack of N decoders**. They are identical in structure and do not share weights.



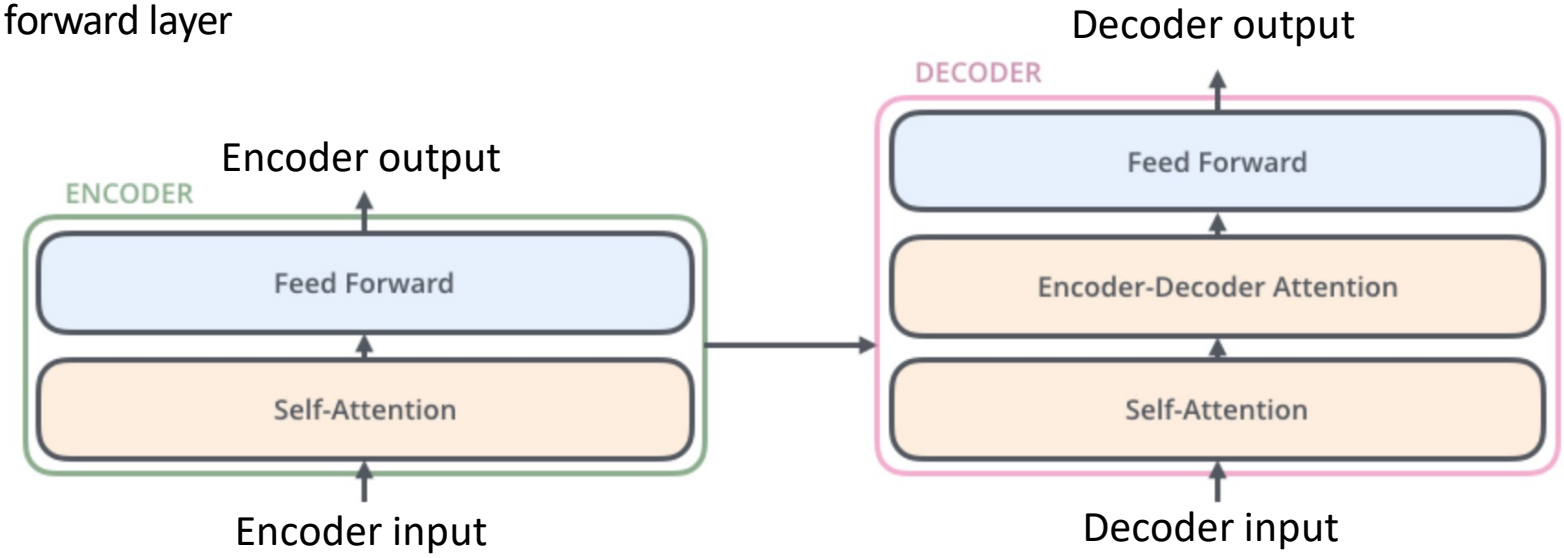
Transformer - Encoder and decoder

The encoder:

- Self-attention layer: helps the encoder look at other words in the input sentence as it encodes a specific word.
- Feed-forward neural network. The exact same feed-forward network (i.e., shared weights) is independently applied to each position (i.e., each output of the self-attention layer).

The decoder:

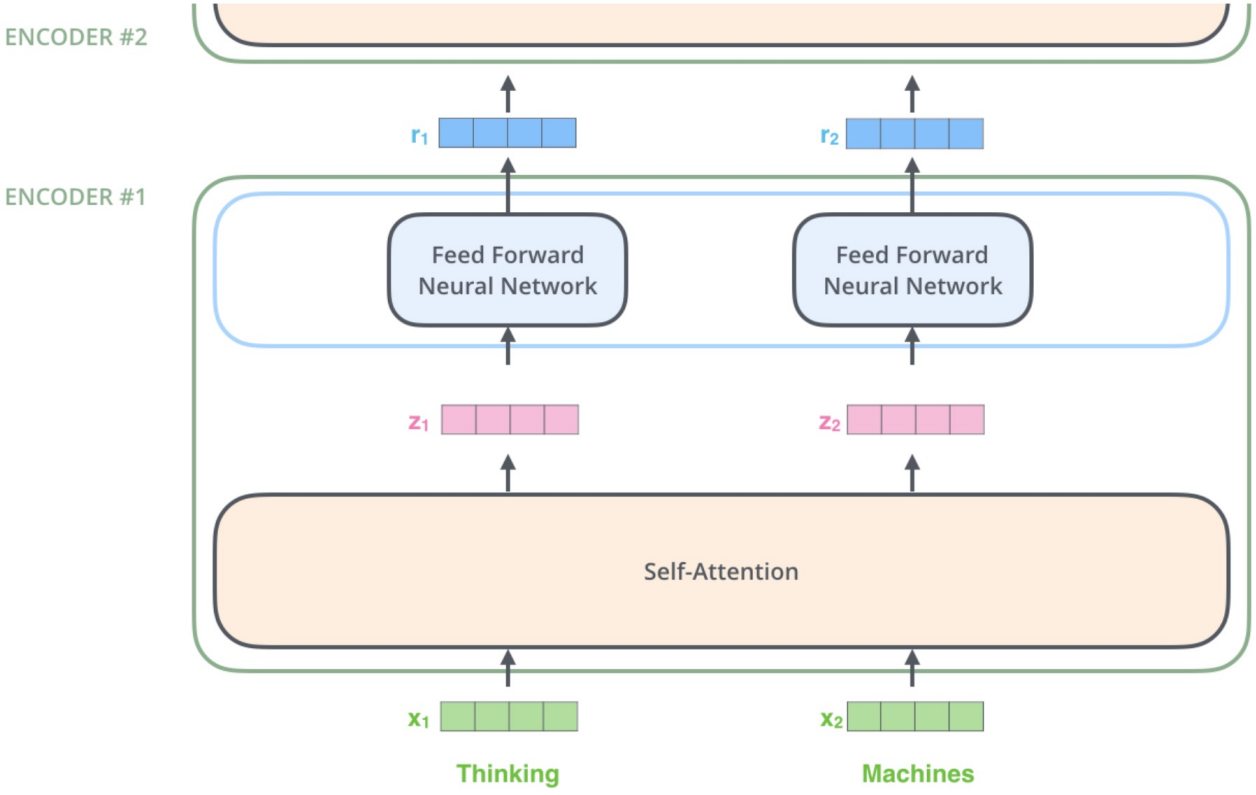
- Self-attention layer **(only take the decoder input itself as input)**
- Encoder-decoder attention layer **(take encoder output and the decoder input as input)**: helps the decoder focus on relevant parts of the input sentence (similar what attention does in seq2seq models).
- Feed forward layer



Transformer - Encoding

Encoder receives a list of **vectors as input**. It processes this list by passing these vectors into the **self-attention** layer, then into a **feed-forward** neural network, then sends out the output upwards to the **next encoder**.

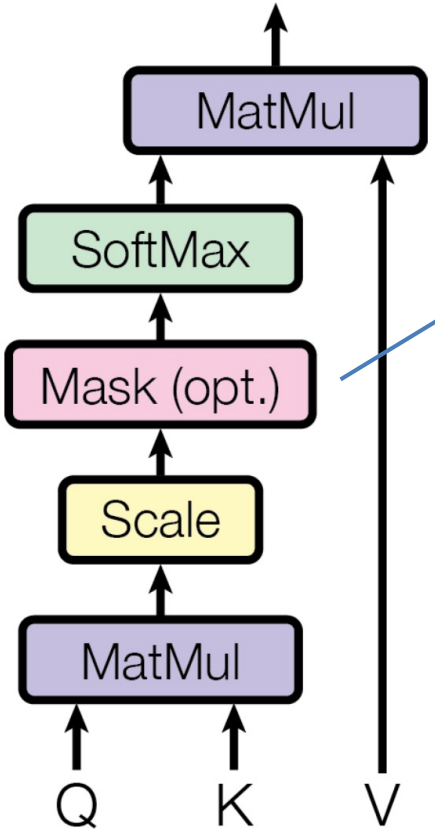
The word at each position passes through a self-attention process. Then, they each pass through a feed-forward neural network. The exact same network with each vector flowing through it separately → support **parallelization**.



Transformer - Encoding: Scaled Dot-Product Self-attention

Scaled Dot-Product Self-attention is a variant of Self-attention.

- The query, key and value vectors have dimensionality of 64.
- The embedding and encoder input/output vectors have dimensionality of 512 (d_{model}).



Mask is optional, i.e., only in decoder Masked Multi-head self-attention but not in encoder self-attention.

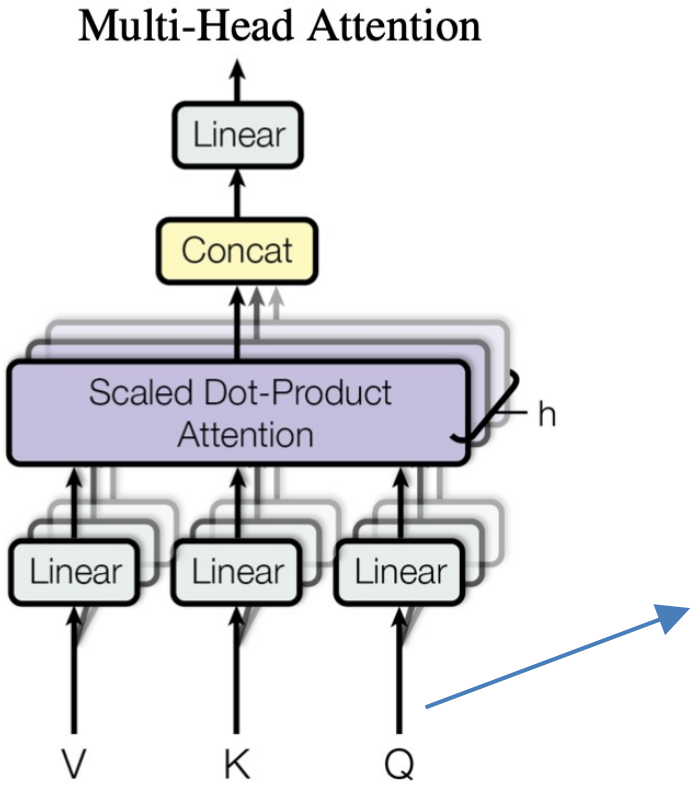
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

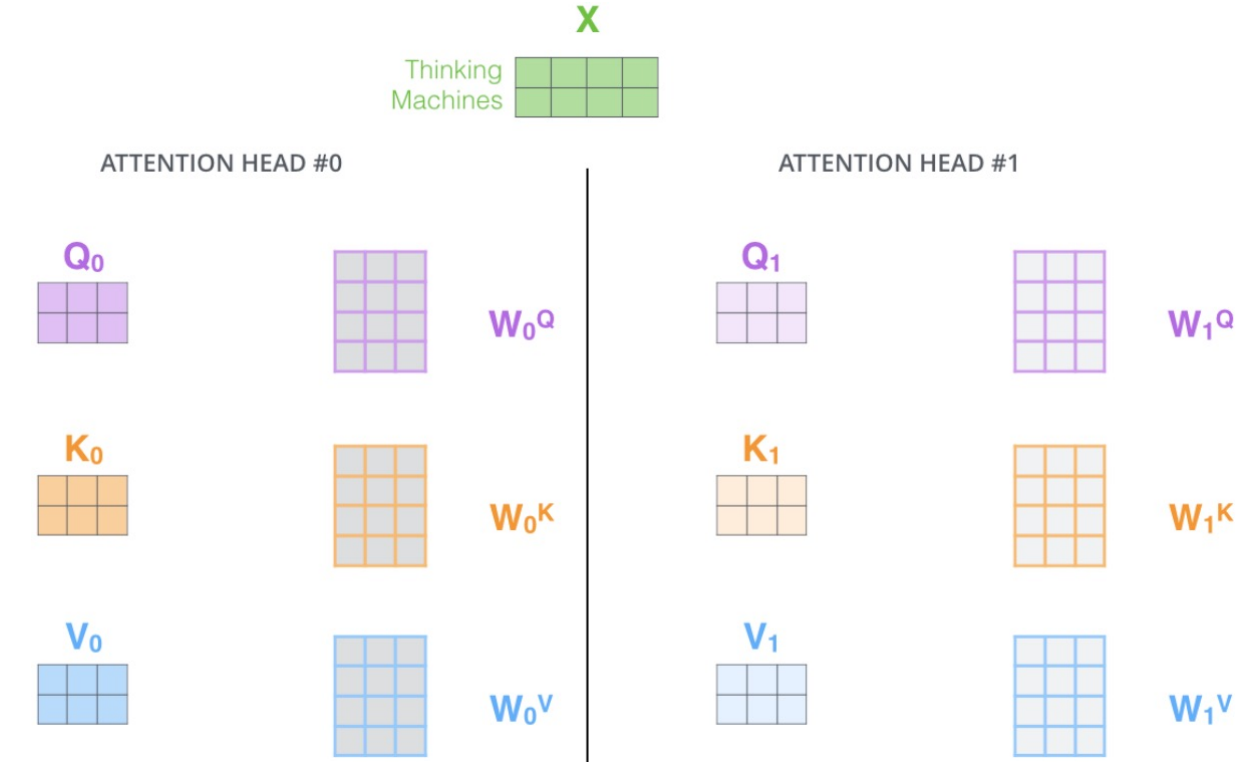
divided by sqrt(d_k), i.e., the square root (8) of the dimension of the key vectors used in the paper (64) to have a more stable gradient.

Transformer - Encoding: Multi-head attention

Multi-head attention is a variant of Self-attention, or specifically Scaled Dot-Product Self-attention.

- *Idea*: gives the attention layer multiple **representation subspaces** → rich diversity in **attention targets**.
- *How*: maintain **multiple sets** of Query/Key/Value weight matrices (the Transformer uses eight attention heads, eight sets for each encoder/decoder). Each of these sets is **randomly initialized**. After training, each set is used to project the input embeddings into a different representation subspace → **Run the attention N (eight) times with different random initial weights**.

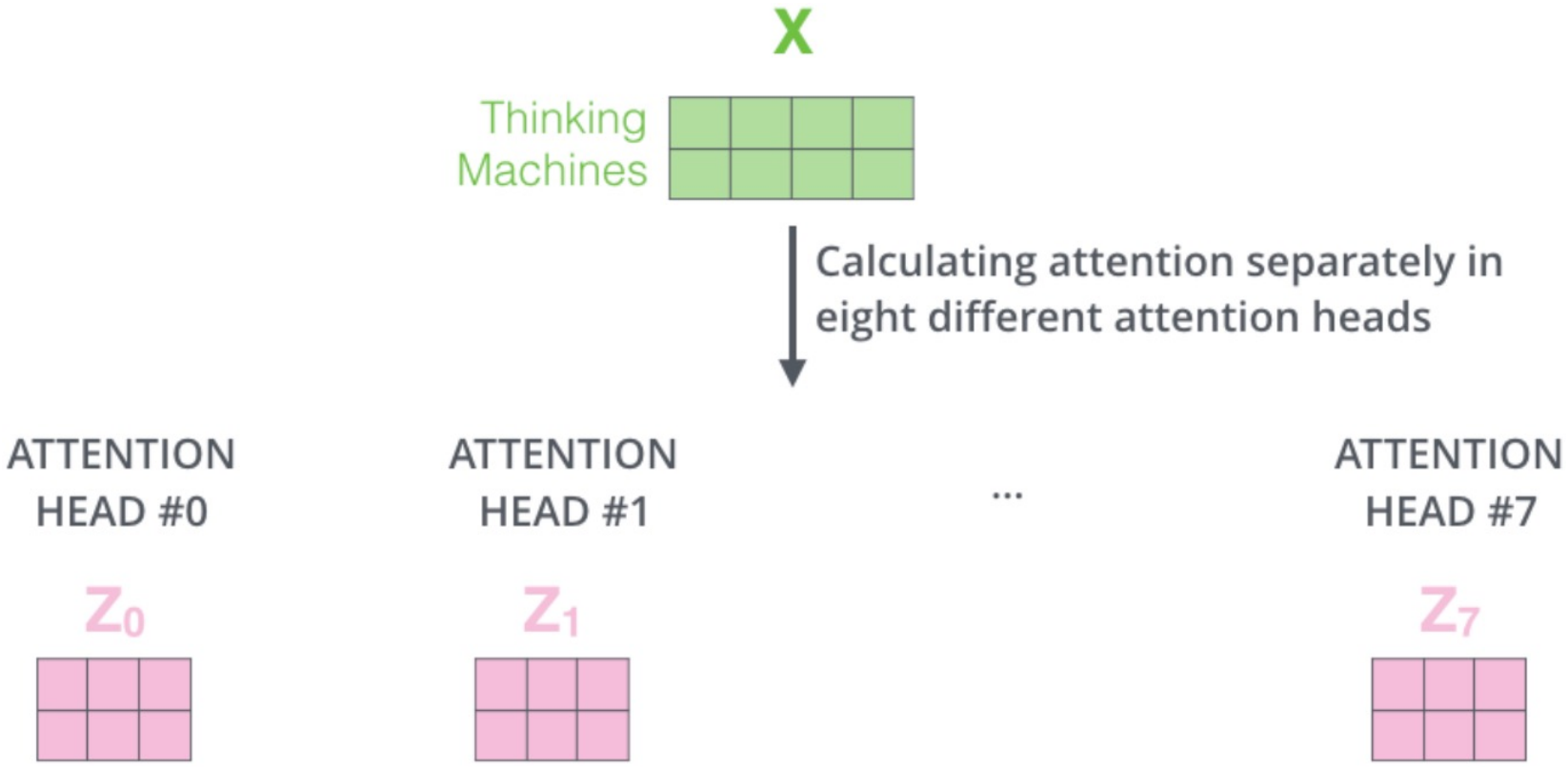




Transformer - Encoding: Multi-head attention

Multi-head attention is a variant of Self-attention, or specifically Scaled Dot-Product Self-attention.

- Do the same self-attention calculation, eight different times with different weight matrices → obtain eight different Z matrices.

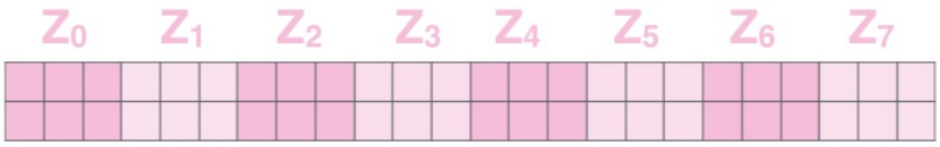


Transformer - Encoding: Multi-head attention

Multi-head attention is a variant of Self-attention, or specifically Scaled Dot-Product Self-attention.

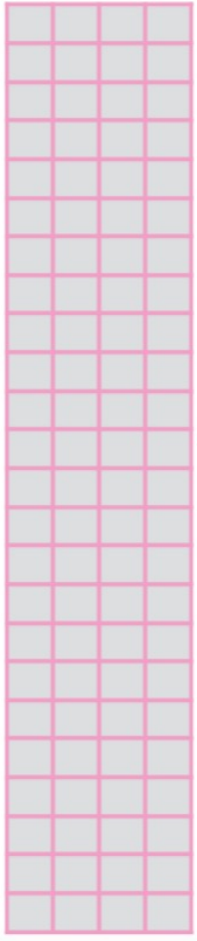
- The feed-forward layer is not expecting eight matrices – it's expecting a single matrix (a vector for each word). So we need a way to condense these eight down into a single matrix.

1) Concatenate all the attention heads



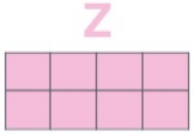
2) Multiply with a weight matrix W^O that was trained jointly with the model

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

=

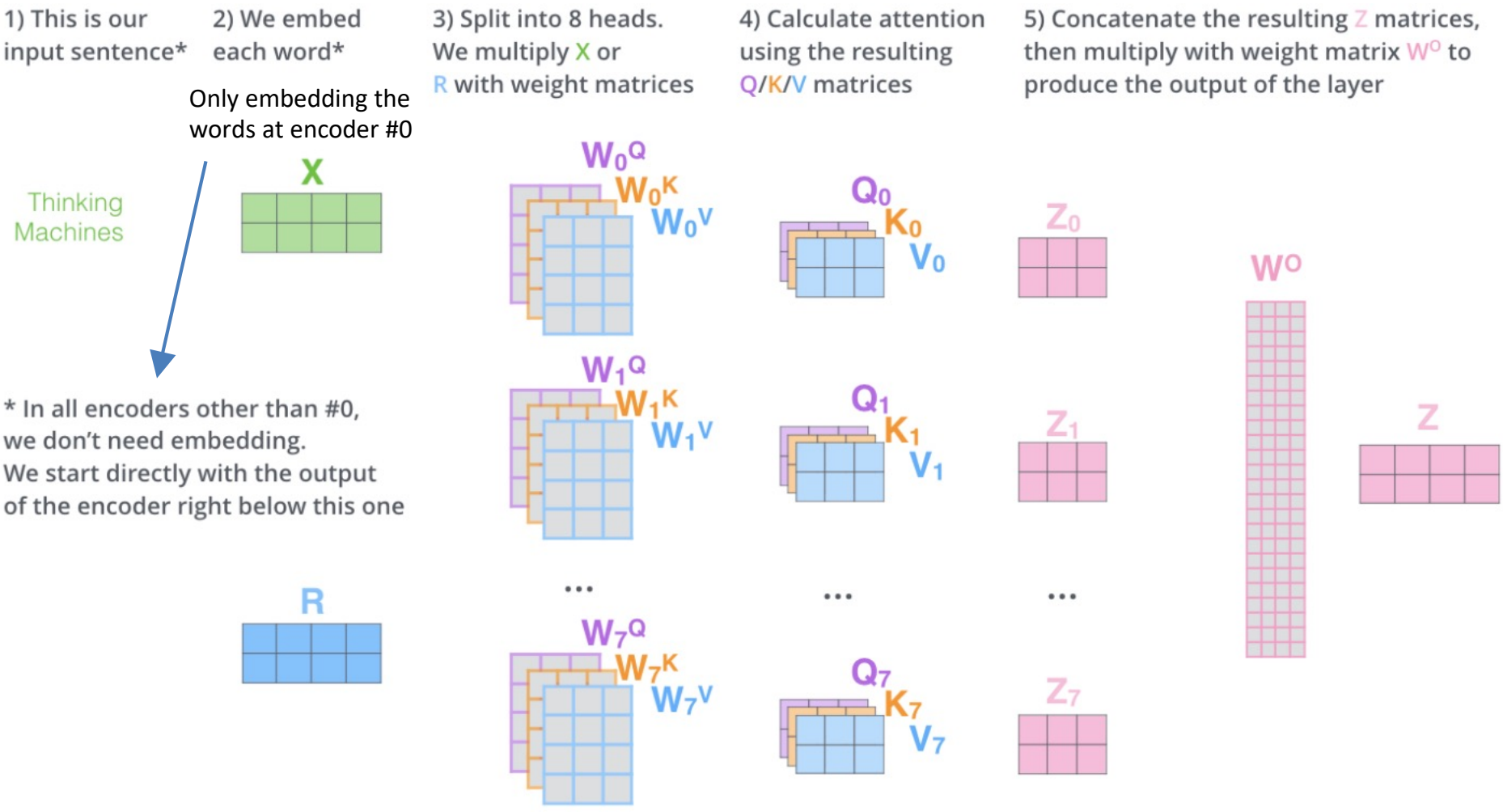


$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$

$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

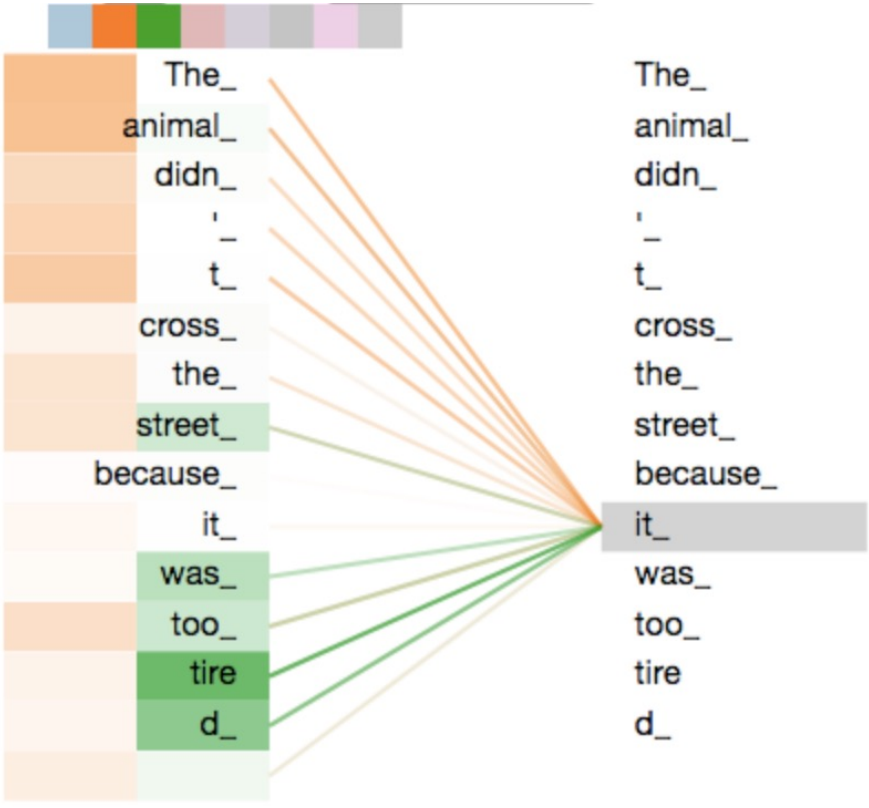
Transformer - Encoding: Multi-head attention summary

Multi-head attention is a variant of Self-attention, or specifically Scaled Dot-Product Self-attention.

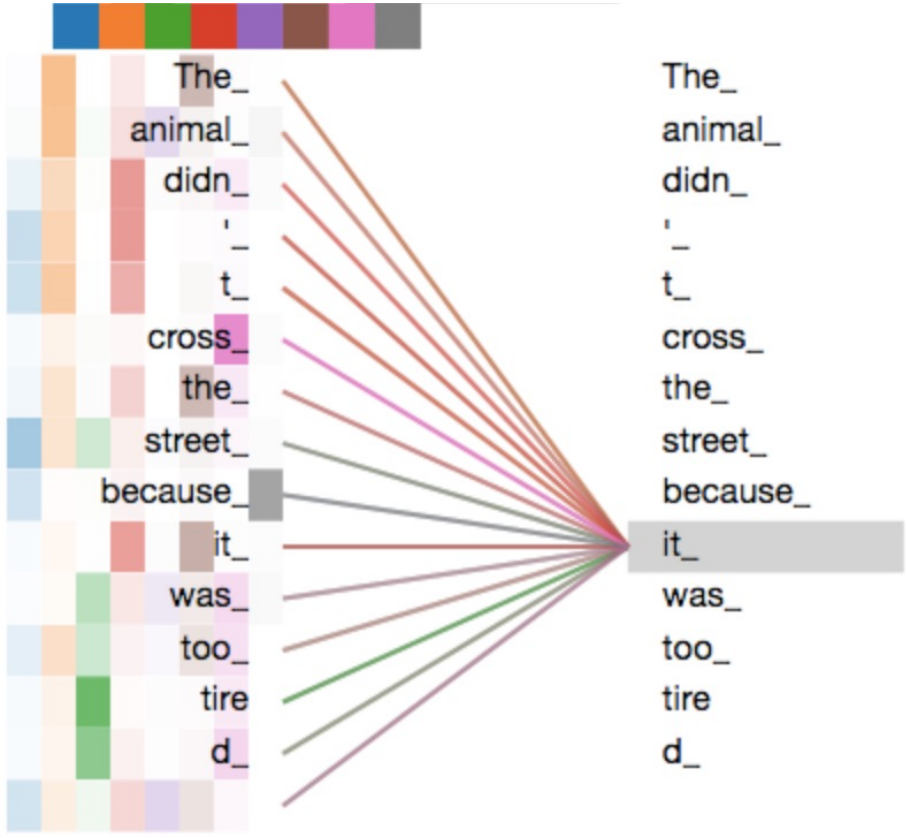


Transformer - Encoding: Multi-head attention visualization

Multi-head attention is a variant of Self-attention, or specifically Scaled Dot-Product Self-attention.



Two-head attention: 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".



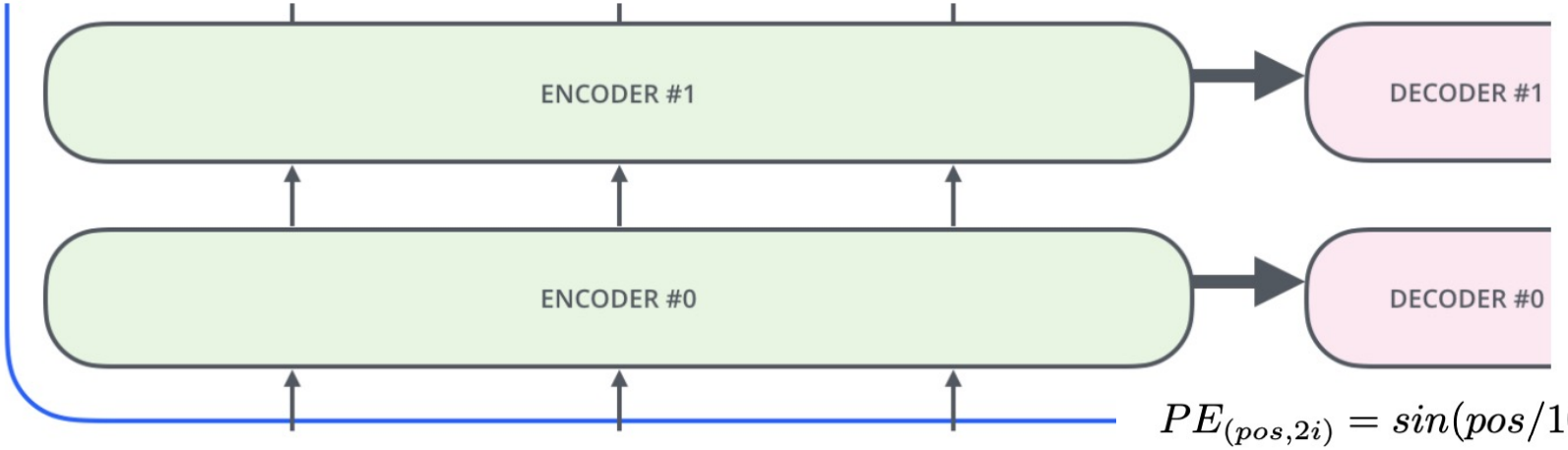
Eight-head attention

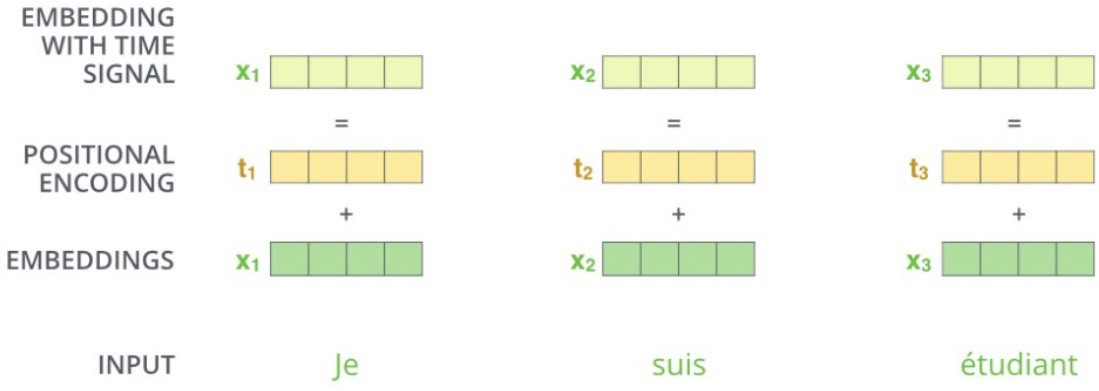
Transformer – Positional encoding

Issue: Transformer removes the sequence order of input words (which exists in RNN/LSTM), so we need a way to account for the order of the words in the input sequence.

Solution: adds a position vector to each input word embedding → **positional encoding**.

Transformer architecture is equipped with **residual connections** → position information **does not get vanished** once it reaches the upper layers





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

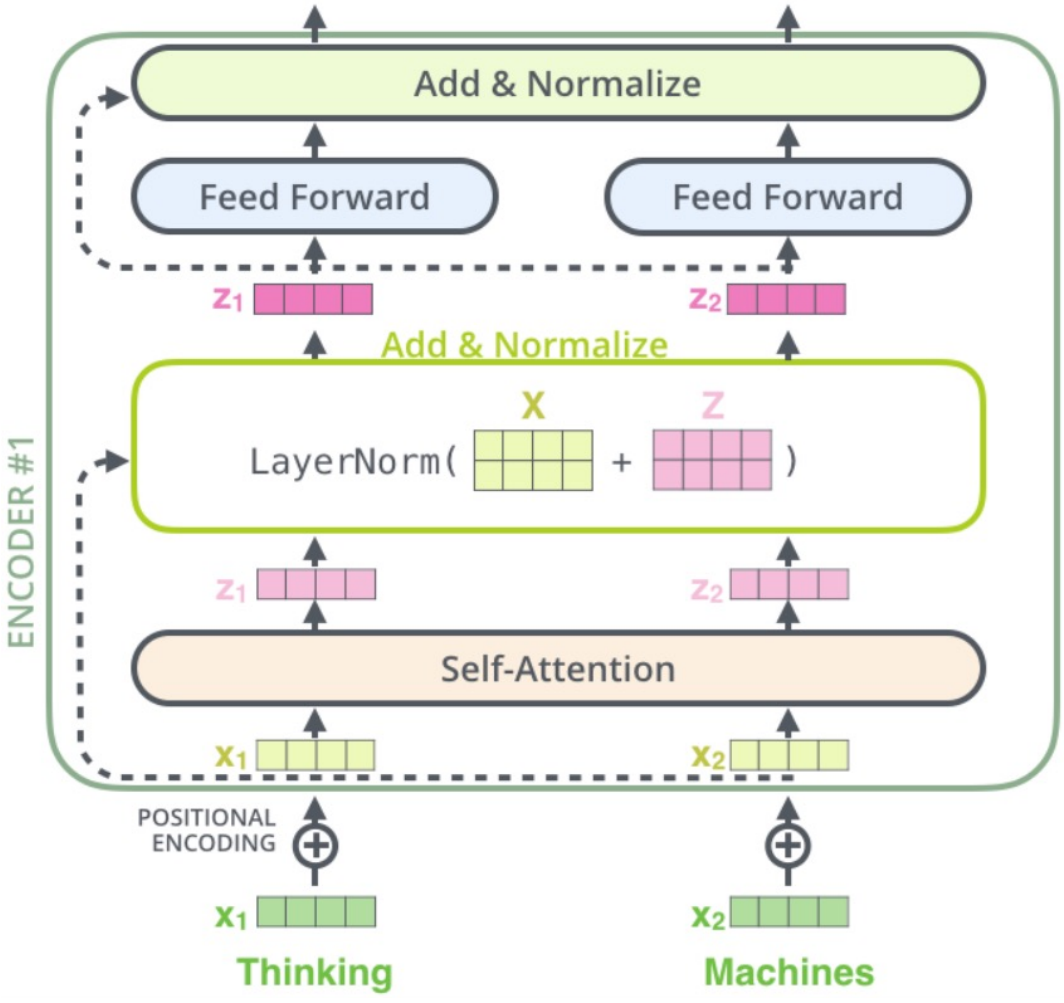
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

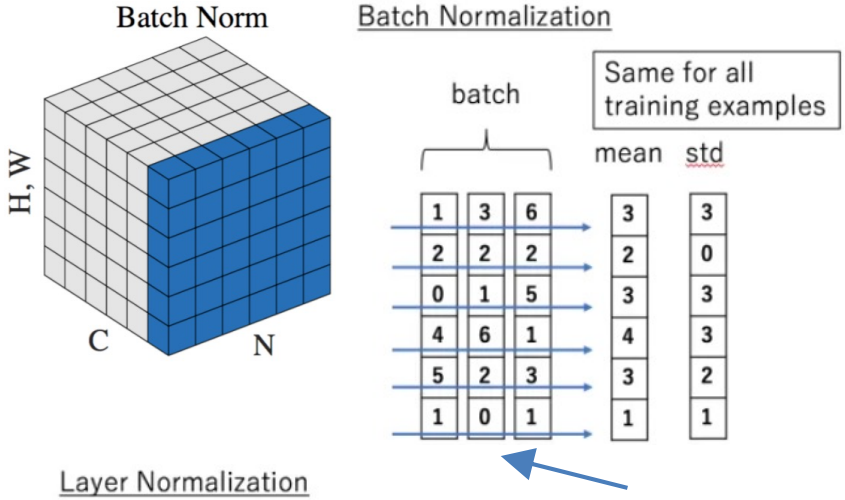
- **pos**: position in the sentence.
- **i**: feature i^{th} [0,255] inpositional encoding vector
- $\mathbf{d_{model}}$ = 512.

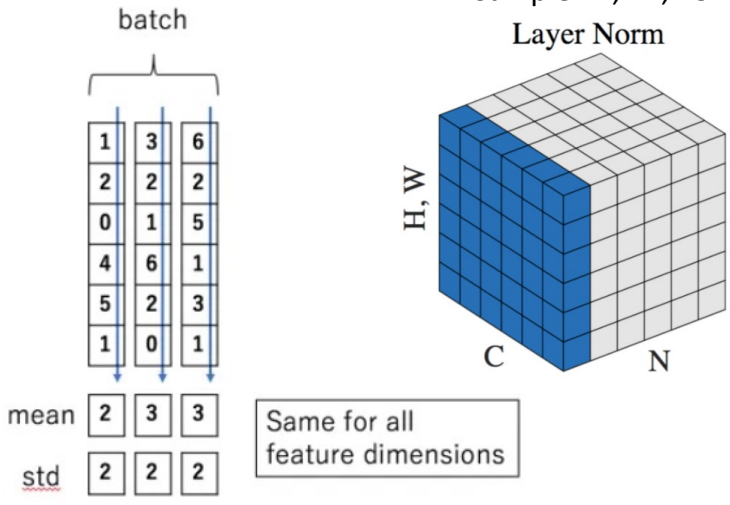
This function would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos} (regardless of pos).

Transformer – Encoding: Residual connection

- Residual connection is around each of the two sub-layers (i.e., attention layer and feed forward layer).
- The residual connection is followed by Layer Normalization. $\text{LayerNorm}(x + \text{Sublayer}(x))$





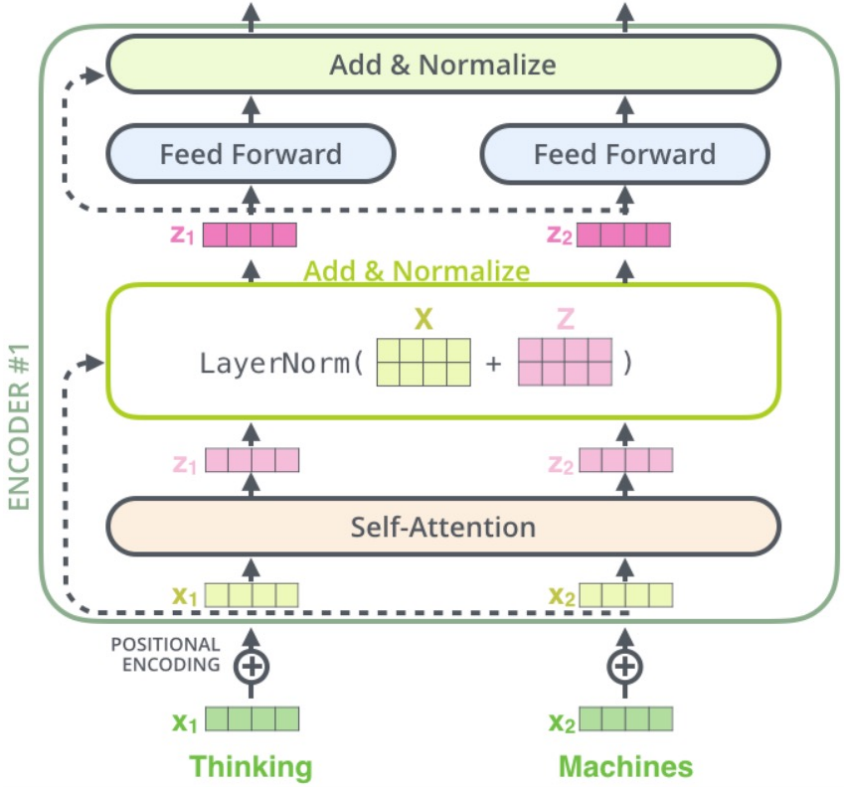


Transformer – Encoding: Feed forward network

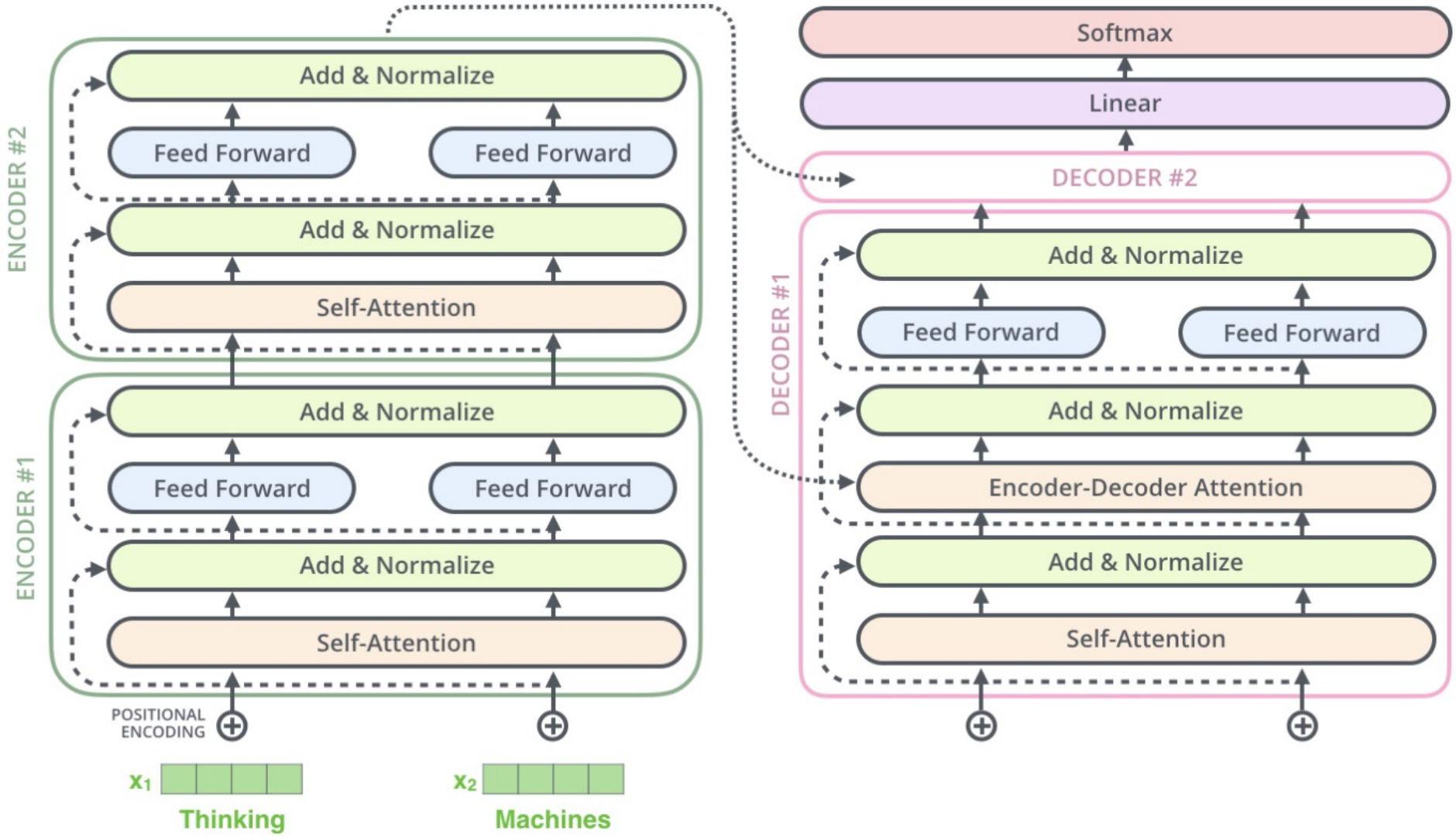
- Feed forward network: two fully connected layers, which are applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

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

- While the linear transformations are the same across different positions, they use different parameters from layer to layer.
- The dimensionality of input and output is d_{model} = 512, and the inner-layer has dimensionality dff =2048.



Transformer – Encoding and Decoding



Transformer of two-stacked encoder and decoder.

Transformer – Decoding

- The output of the top encoder **used by each decoder** in its "encoder-decoder attention" layer (a.k.a. decoder's **Multi-head attention**) to focus on appropriate places in the decoder's input sequence.

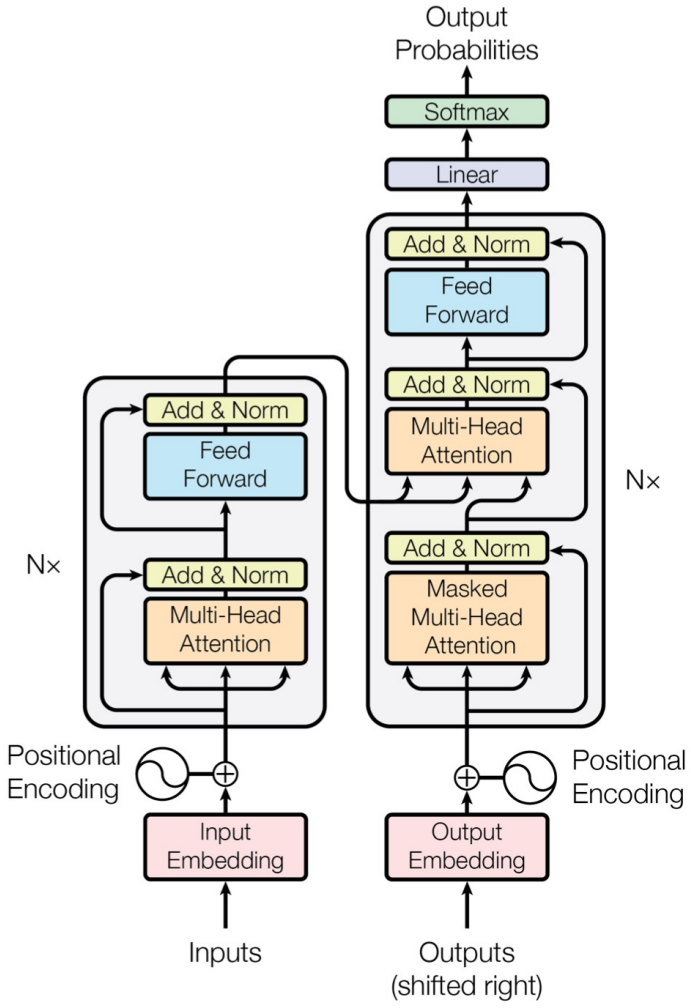
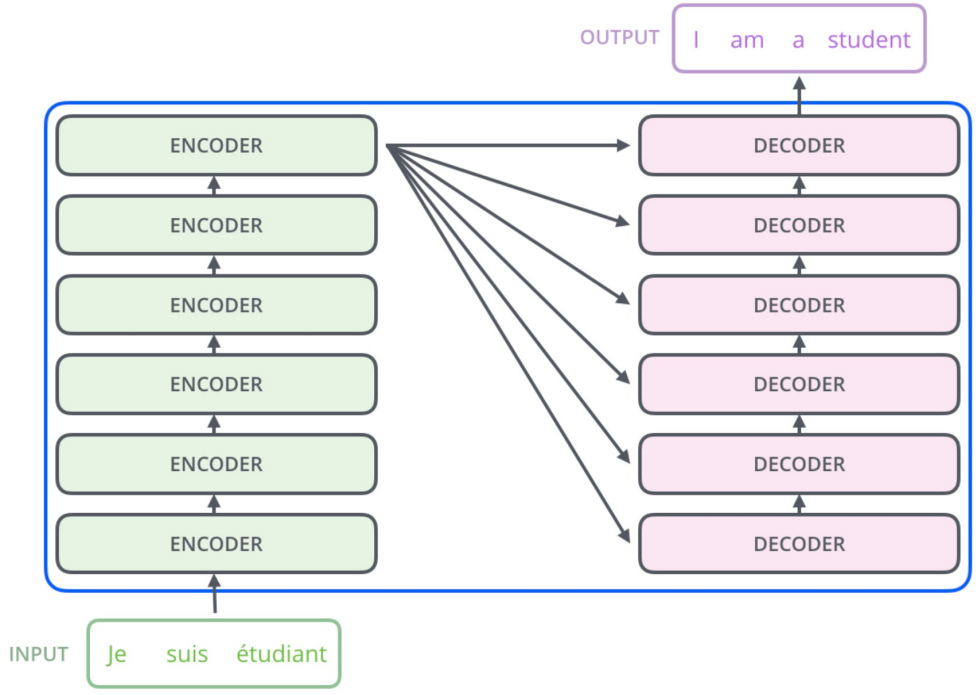
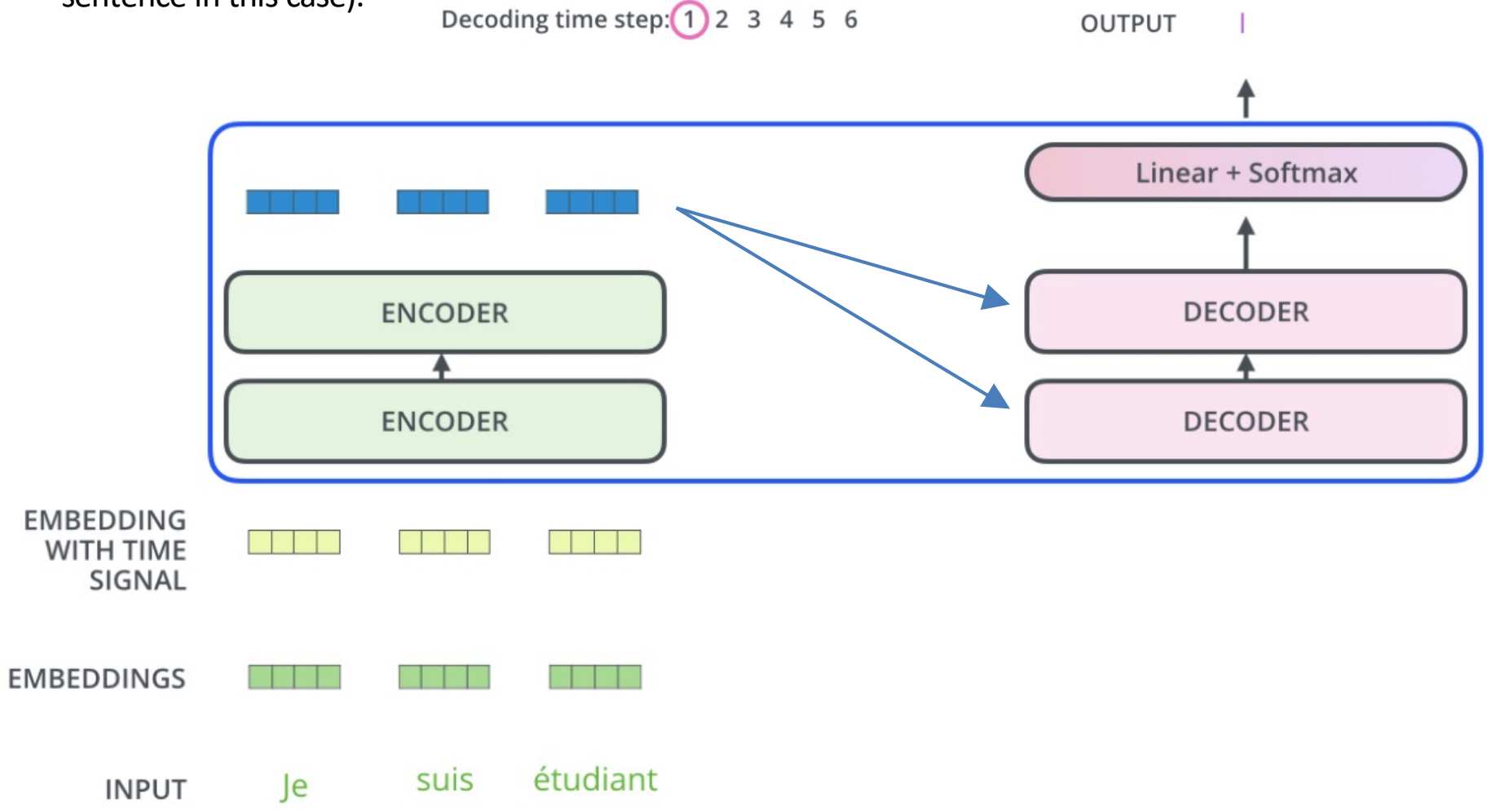


Figure 1: The Transformer - model architecture.



Transformer – Decoding

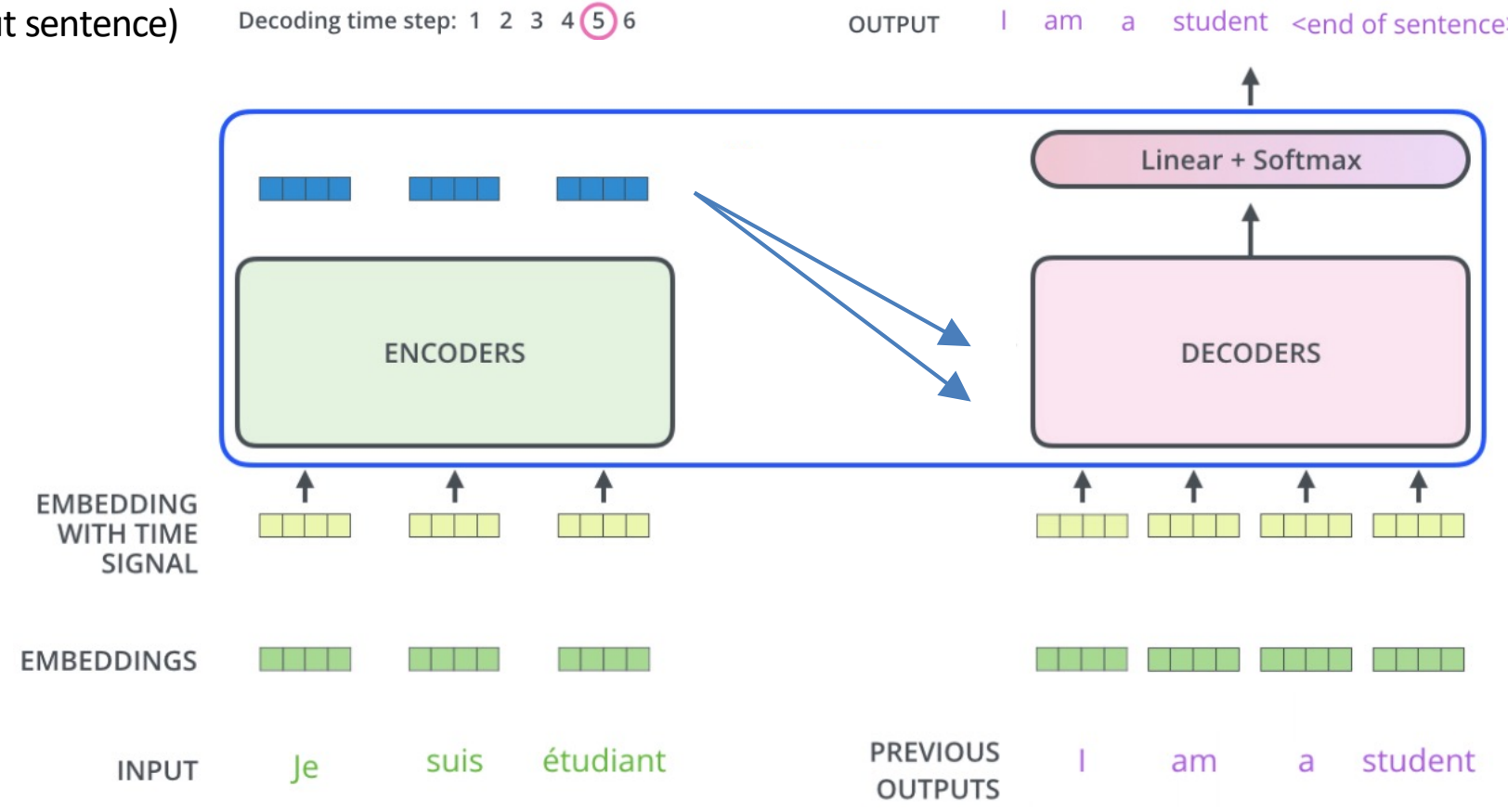
- The output of the top encoder **used by each decoder** in its "encoder-decoder attention" layer (a.k.a. decoder's **Multi-head attention**) to focus on appropriate places in the decoder's input sequence.
- Each step in the decoding phase outputs an element from the output sequence (the English translation sentence in this case).



Transformer – Decoding

The following steps repeat the process until a <EOS> symbol is reached :

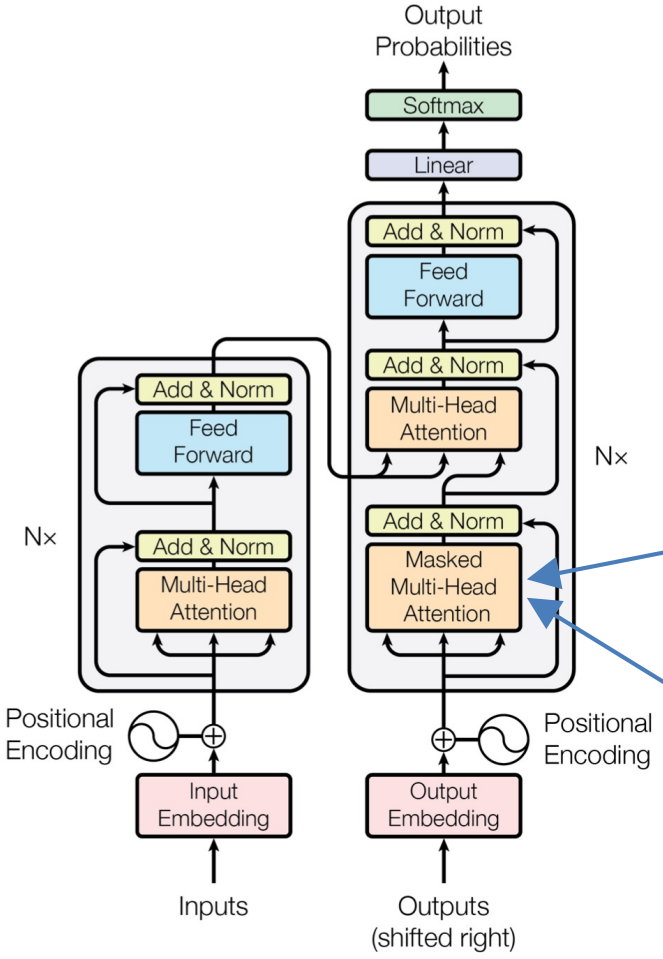
- Output of each step is **fed to the bottom decoder** in the next time step.
- Each decoder **bubbles up** their decoding results.
- Just like we did with the encoder inputs, we embed and add **decoder's positional encoding** to those decoder inputs to indicate the position of each word (to get the notion of context of the word in the output sentence)

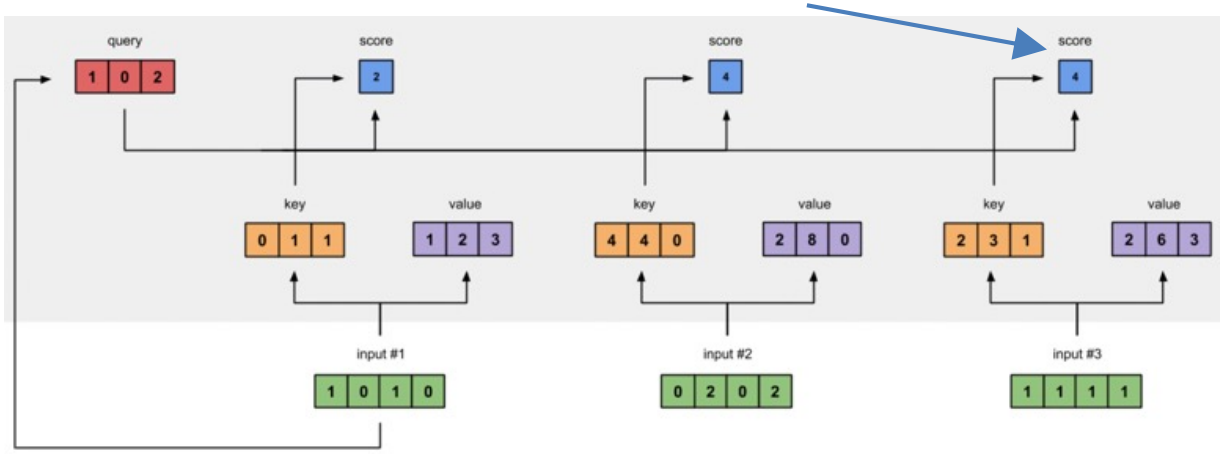


Transformer – Decoding: Masked Multi-head attention

In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence. This is done by masking future positions (setting them to **-inf**) before the softmax step in the self-attention calculation, so after softmax the **self-attention score becomes zero**.

Set these value to -inf if this time step does not happen yet.





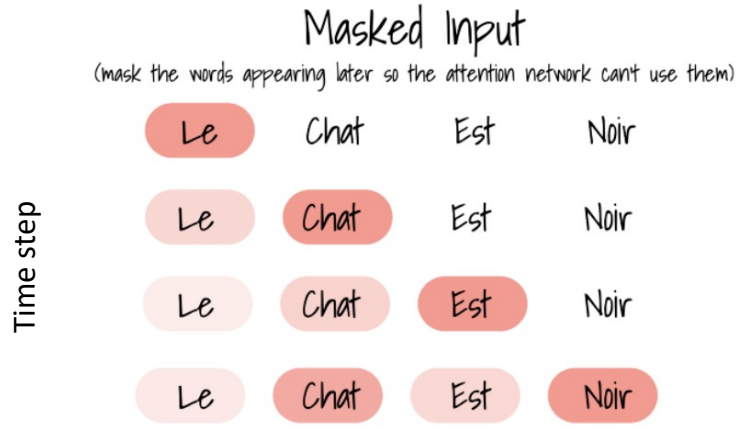
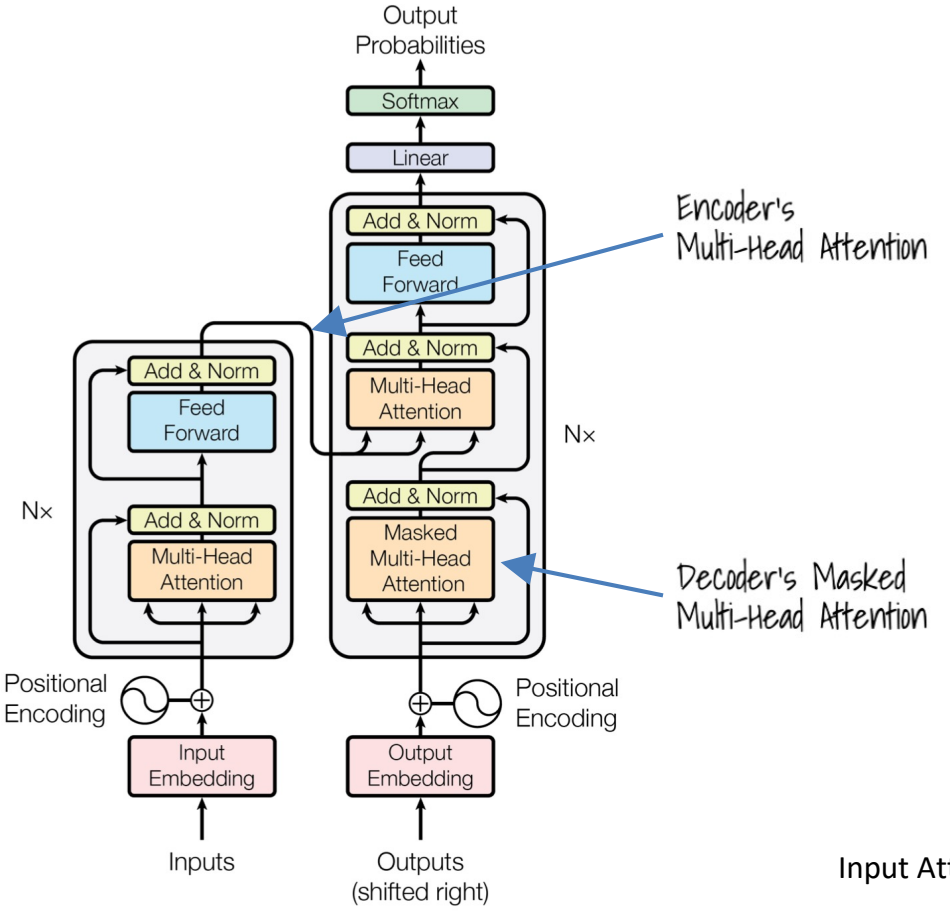
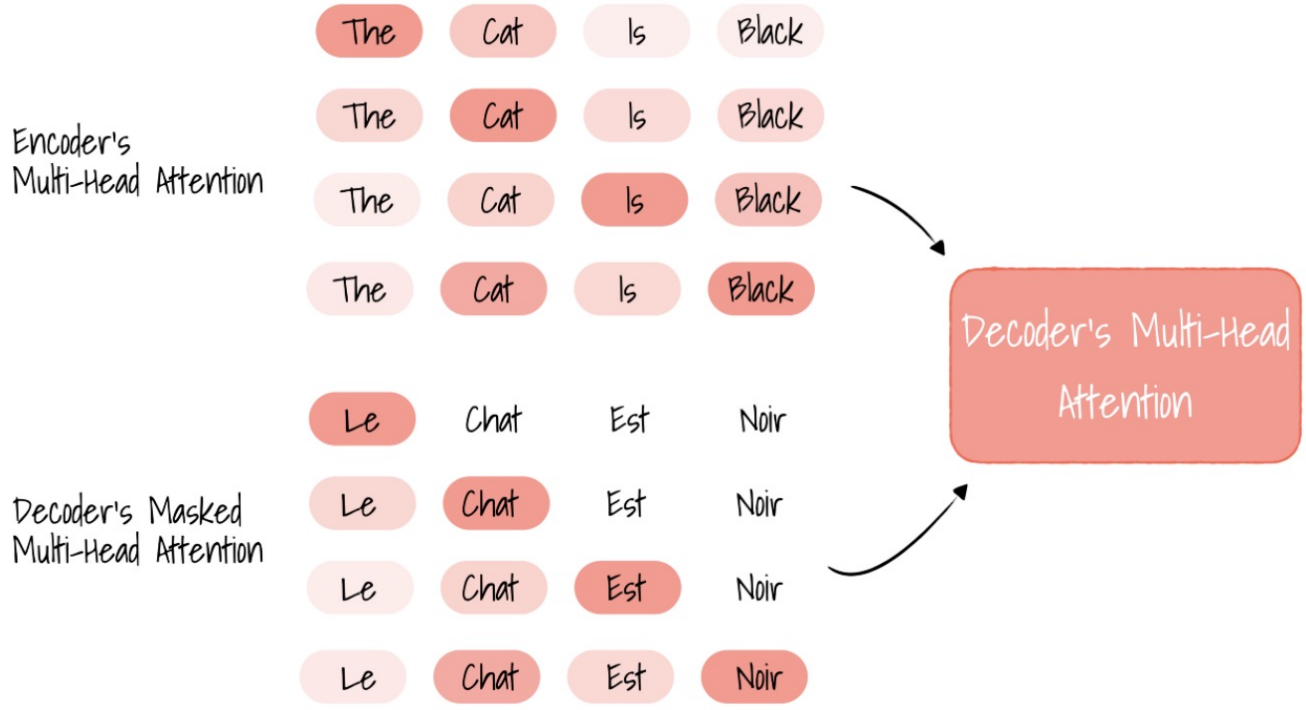


Figure 1: The Transformer - model architecture.

Transformer – Decoding: Multi-head attention

Decoder Multi-head attention (or Encoder-Decoder attention) receives vectors from the Encoder's multi-head attention and Decoder's Masked Multi-Head attention → determine how related each word vector is with respect to each other





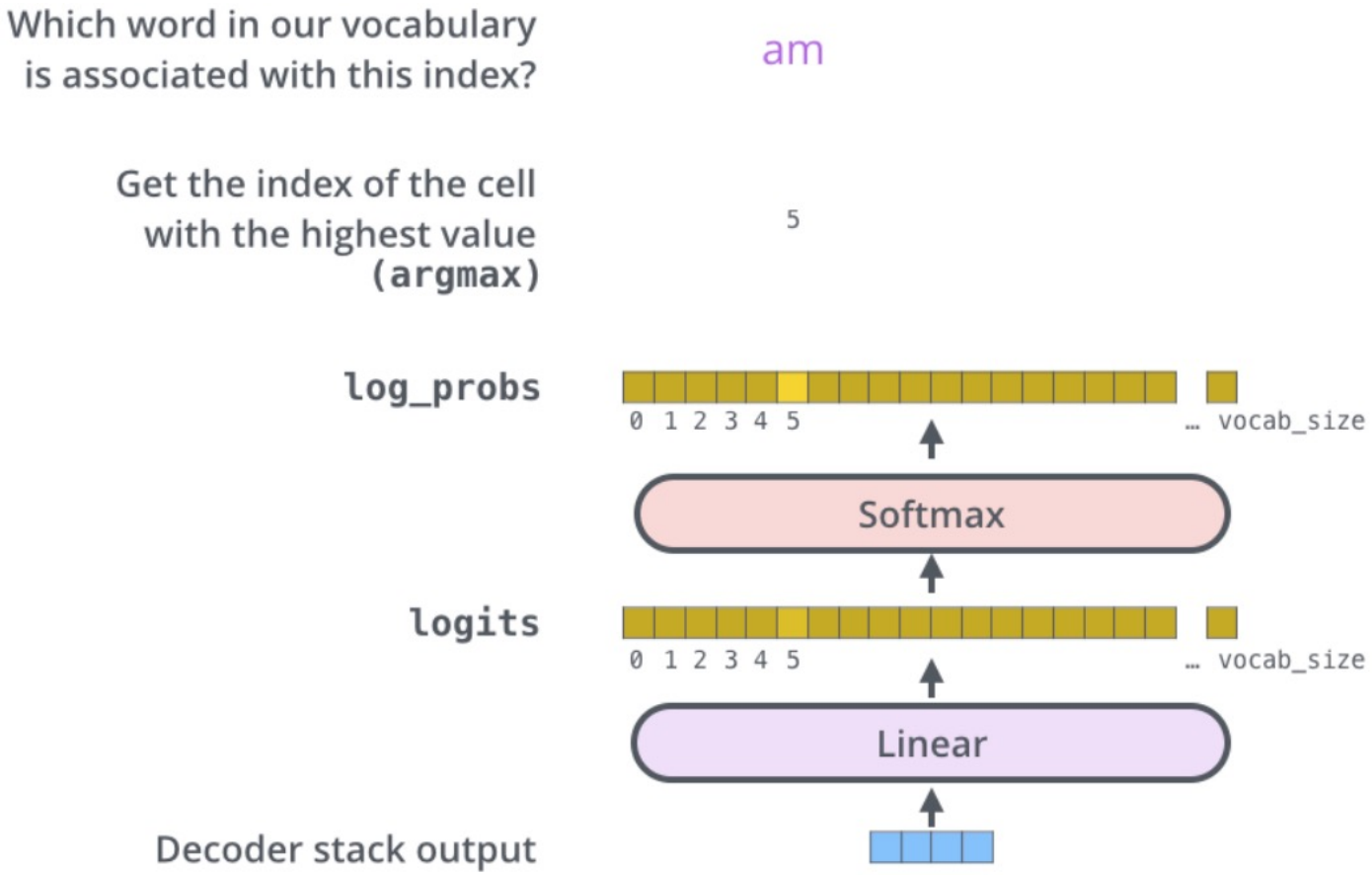
Input Attention vectors into the Decoder's Multi-Head Attention

Figure 1: The Transformer - model architecture.

Transformer – Decoding: Linear and Softmax

The decoder stack outputs a **vector of floats** → convert to a word using linear layer followed by a softmax:

- **Linear layer** is a simple fully connected neural network that projects the vector produced by the stack of decoders, into a much, much larger vector called a logits vector (size of vocabulary/dictionary).
- **Softmax layer** then turns those scores into probabilities.



Transformer – BLEU score evaluation

BLEU: **Bi**L**ingual** **E**valuation **U**nderstudy

- A **n-gram based string-matching** algorithm for evaluating Machine Translation.
- A higher match degree (score) indicates a higher **degree of similarity** with the reference translation.
- Intelligibility and grammatical correctness are not taken into account.

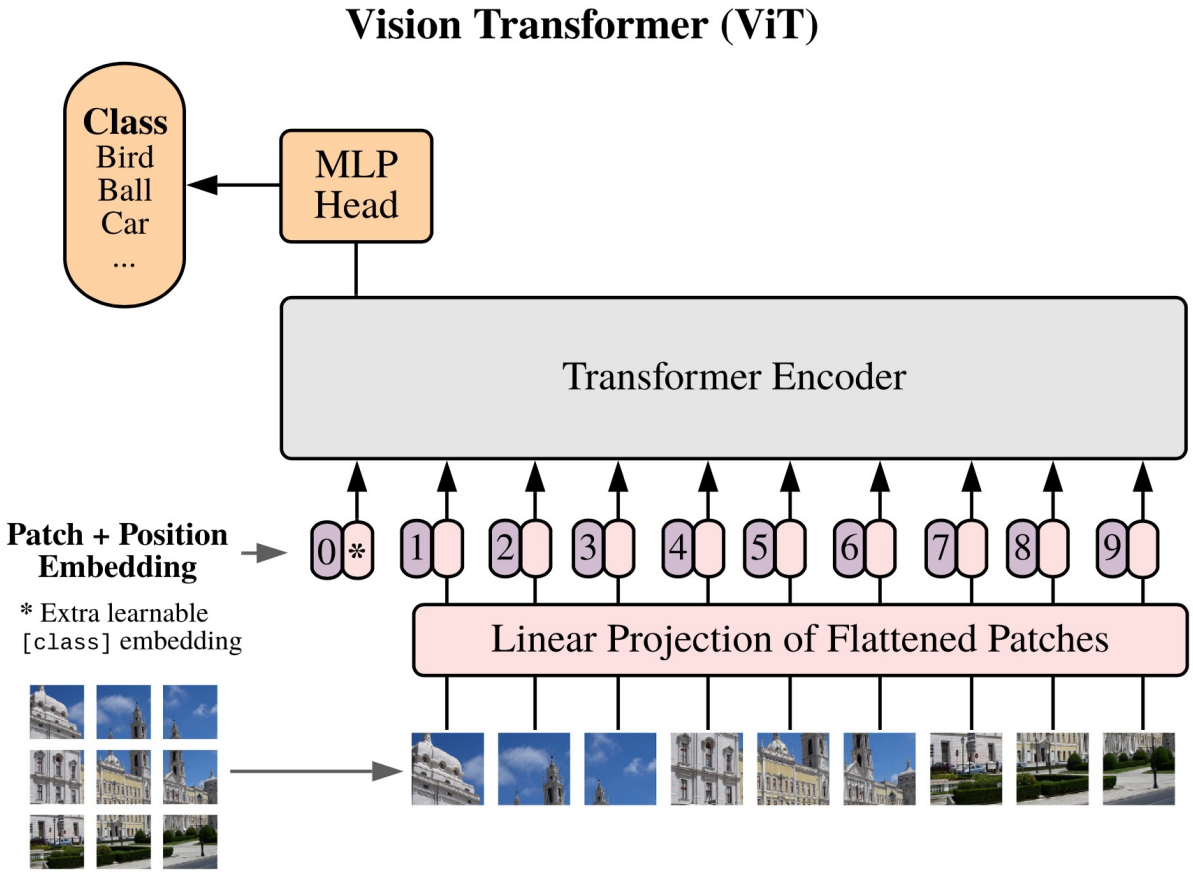
Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

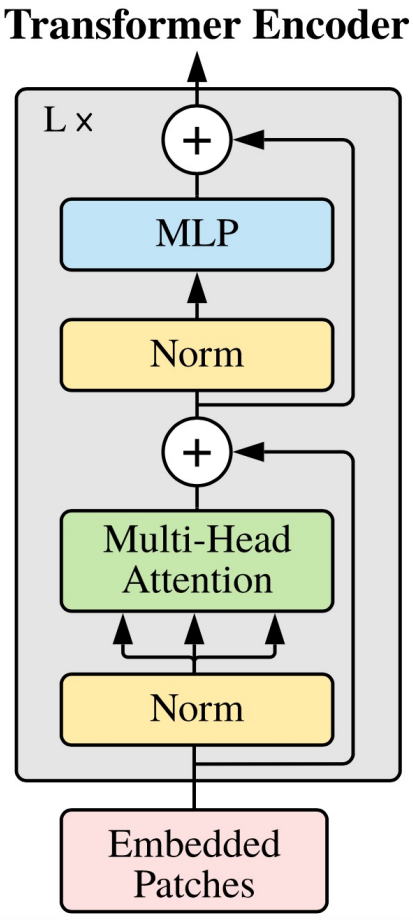
Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Vision Transformer

The Vision Transformer treats an input image as a **sequence of patches**, akin to a series of word embeddings generated by a natural language processing (NLP) Transformer.

- Applying Transformer network to solve visual tasks
- Attention on image patches; no use of CNN.





BERT: Bidirectional Encoder Representations from Transformers

BERT makes use of Transformer:

- Transformer consists of **an encoder** (reads the text input) and **a decoder** (produces a prediction).
- BERT only includes **an encoder** (generate a language model).

BERT proposes a new pre-training objective so that a deep bidirectional Transformer can be trained:

- **Masked Language Model** (MLM): predicts the original word of a masked word based only on its context
- **Next Sentence Prediction** (NSP): receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence

→ Train on **unlabeled data** (i.e., text corpus).

Merits of BERT:

- Just need to **fine-tune BERT** model (initialized with the pre-trained weights) for specific tasks to achieve state-of-the-art performance.
- BERT advances the state-of-the-art NLP tasks.

BERT: Pre-training and Fine-Tuning

- Pre-training on **unlabeled data** for general tasks (NSP: Next Sequence Prediction; Mask LM: Masked Language Model) and fine-tuning for every specific task.
- A specific task: **initializing weights** with pre-trained weights and fine-tuning the model.

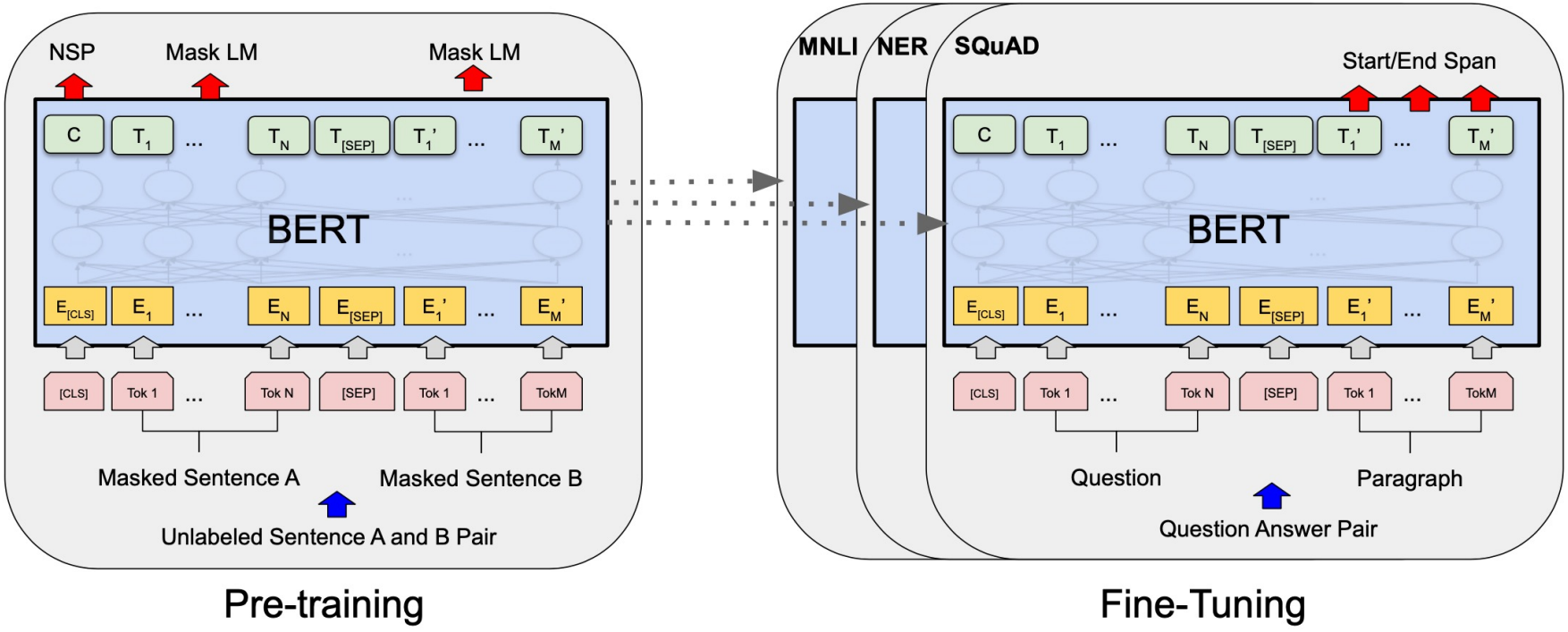


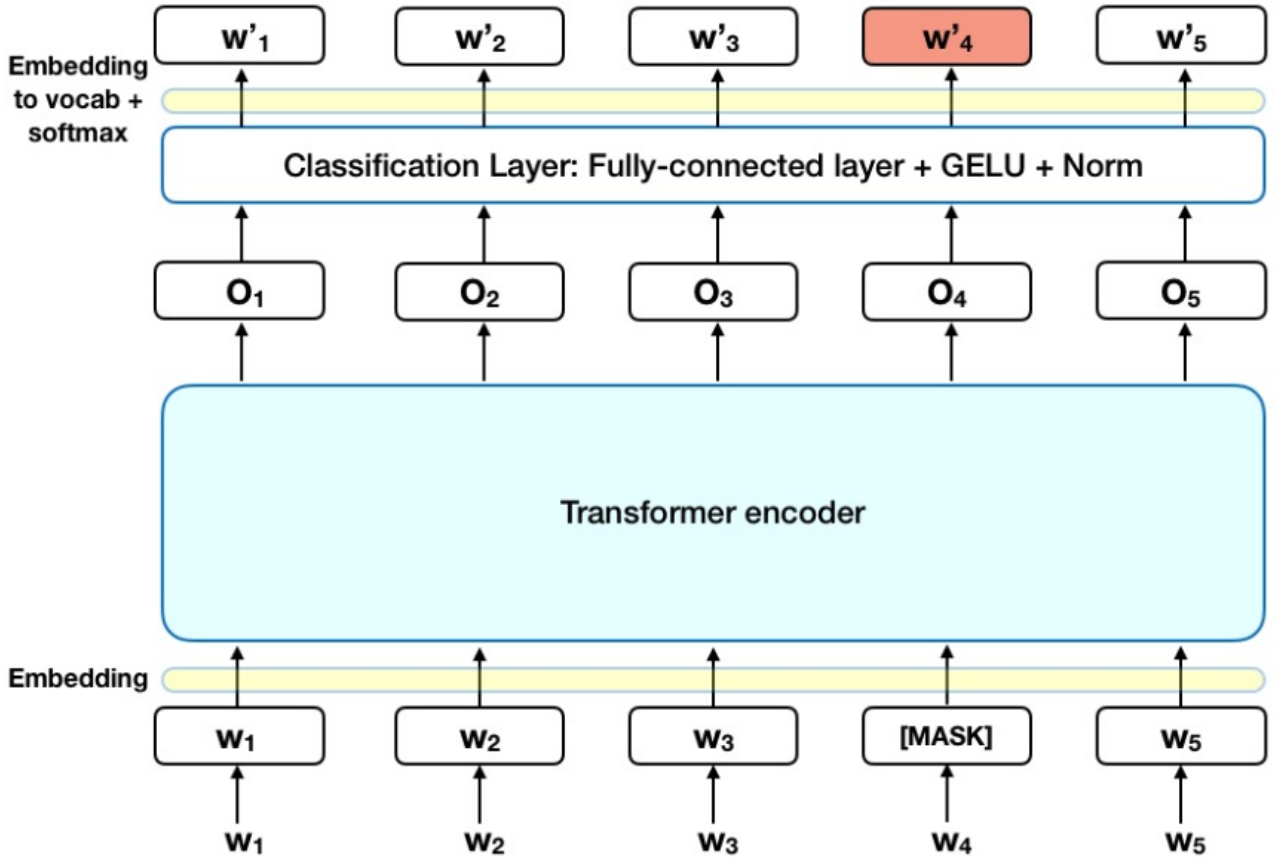
Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

BERT: Masked Language Model (MLM)

Before feeding word sequences into BERT, **15%** of the words in each sequence are **replaced with a [MASK] token**. The model then attempts to **predict the original value of the masked words**, based on the context provided by the other, non-masked, words in the sequence.

1. Add a **classification layer** on top of the encoder output.
2. **Multiply** the each output vector by the embedding matrix, transforming them into the vocabulary dimension.
3. Calculate the probability of each word in the vocabulary with **softmax**.

The loss is computed on the [MASK] word only.



Jacob et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"

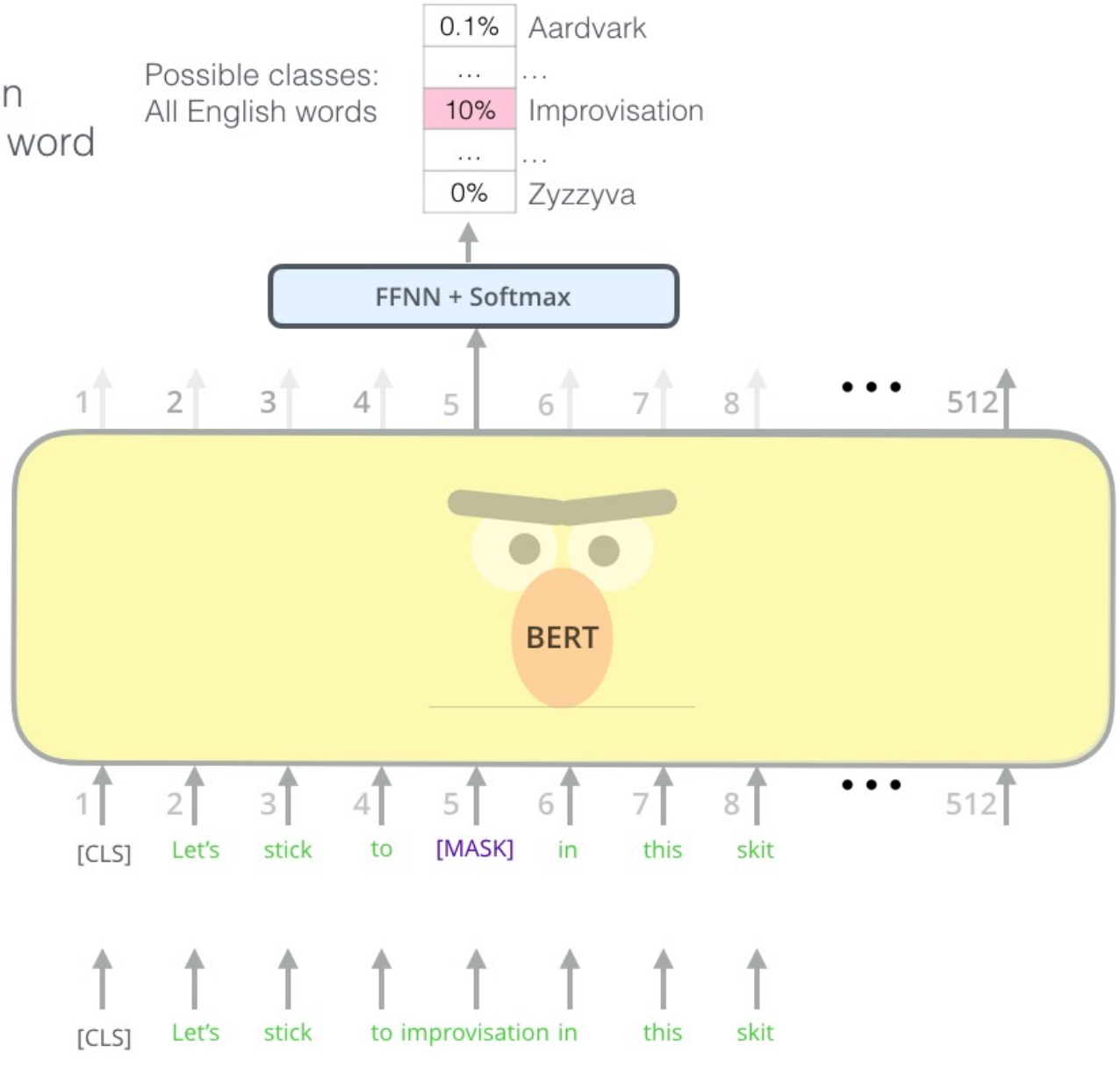
BERT: Masked Language Model (MLM)

15% of the words that were fed in as input were masked. But not all tokens were masked in the same way. Example: **My dog is hairy**

- 80% were replaced by the **<MASK>** token. **My dog is <MASK>**
  - The model learns the contextual bidirectional representation of all words in the sentence to predict the <MASK>.
  - If the model is trained on only predicting <MASK> tokens and then never see this token during fine-tuning → *decrease performance* during inference phase → need to have the other 20% of random and intact tokens.
- 10% were replaced by a random token. **My dog is apple**
  - The model tries to use the embedding of the random token to make prediction and it will eventually learn that it was actually not useful once it sees the target (correct token) → able to learn the *random masked word is useless* for contextual representation.
- 10% were left intact. **My dog is hairy**
  - The model learns that it was actually useful once it sees the target (correct token) → able to learn that the *intact masked word is useful* for contextual representation.

BERT: Masked Language Model (MLM)

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



Randomly mask 15% of tokens

Input

BERT: Next Sentence Prediction (NSP)

In order to understand relationship between two sentences, BERT training process also uses next sentence prediction → Understanding is relevant for tasks like **question answering**.

BERT separates sentences with a special [SEP] token. During training the model is fed with two input sentences at a time:

- 50% of the time the second sentence comes after the first one (**IsNext**).
- 50% of the time it is a a random sentence from the full corpus (**NotNext**).

→ predict whether the **second sentence is random or not**, with the assumption that the random sentence will be disconnected from the first sentence.

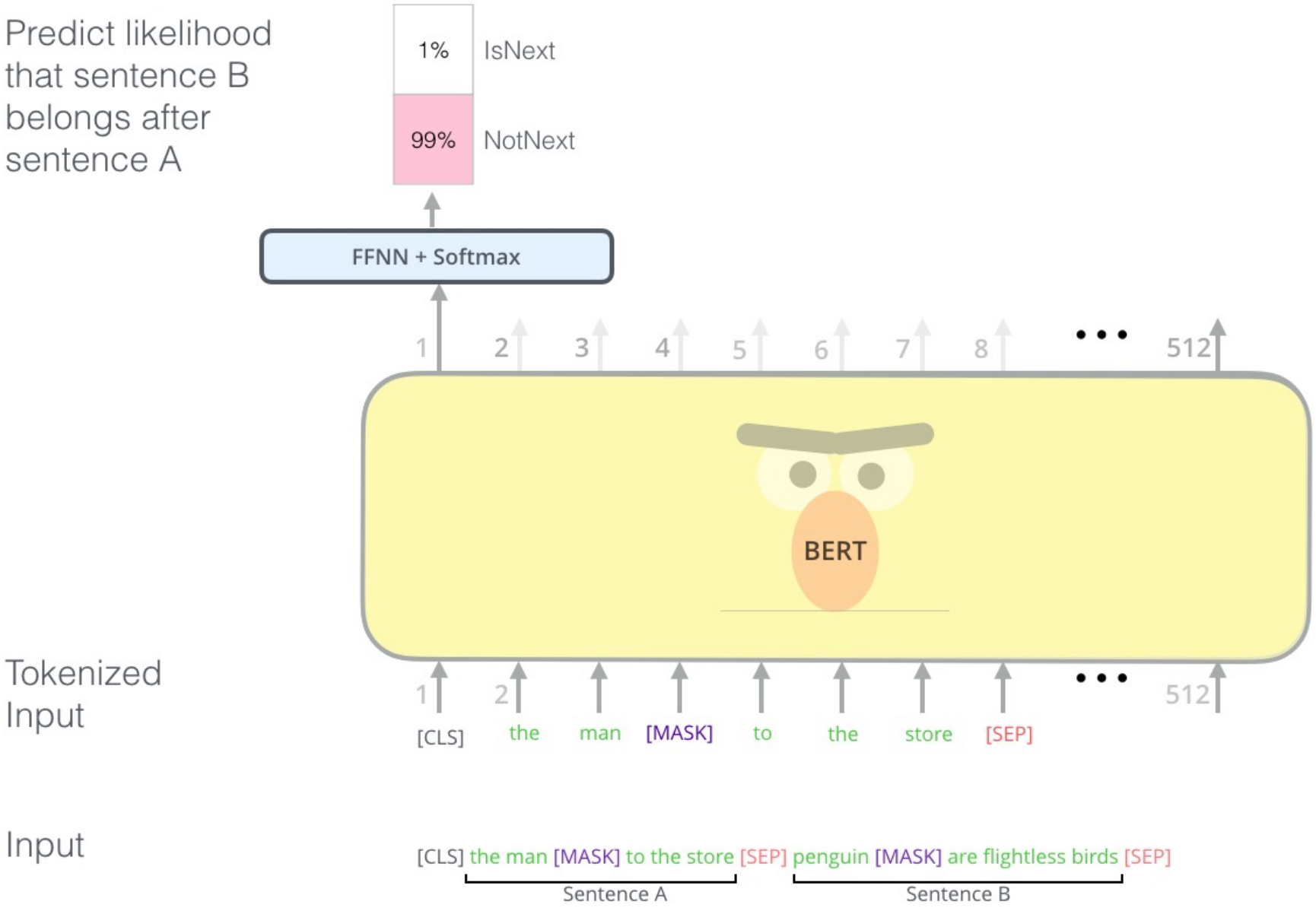
$\text{Input} =$ `[CLS] the man went to [MASK] store [SEP]`
`he bought a gallon [MASK] milk [SEP]`

$\text{Label} =$ `IsNext`

$\text{Input} =$ `[CLS] the man [MASK] to the store [SEP]`
`penguin [MASK] are flight ##less birds [SEP]`

$\text{Label} =$ `NotNext`

BERT: Next Sentence Prediction (NSP)



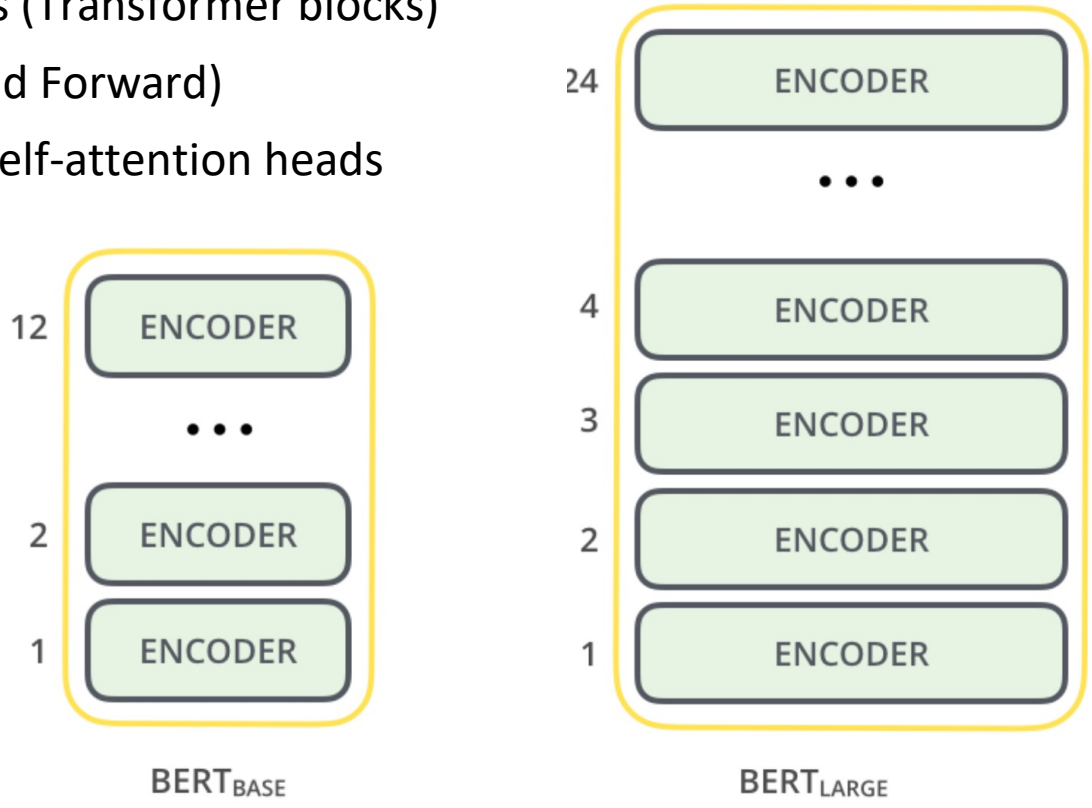
BERT: Architectures

Two BERT models were proposed:

- **$BERT_{BASE}$** : L = 12, H = 768, A = 12, Total Parameters = 110M.
- **$BERT_{LARGE}$**: L = 24, H = 1024, A = 16, Total Parameters = 340M.

Notations:

- **L**: number of layers (Transformer blocks)
- **H**: hidden size (Feed Forward)
- **A**: the number of self-attention heads



BERT vs. OpenAI GPT vs. ELMo

- OpenAI GPT: left-to-right Transformer.
- ELMo: concatenation of independently trained left-to-right and right-to- left LSTMs.
- BERT is jointly conditioned on both left and right context in all layers.

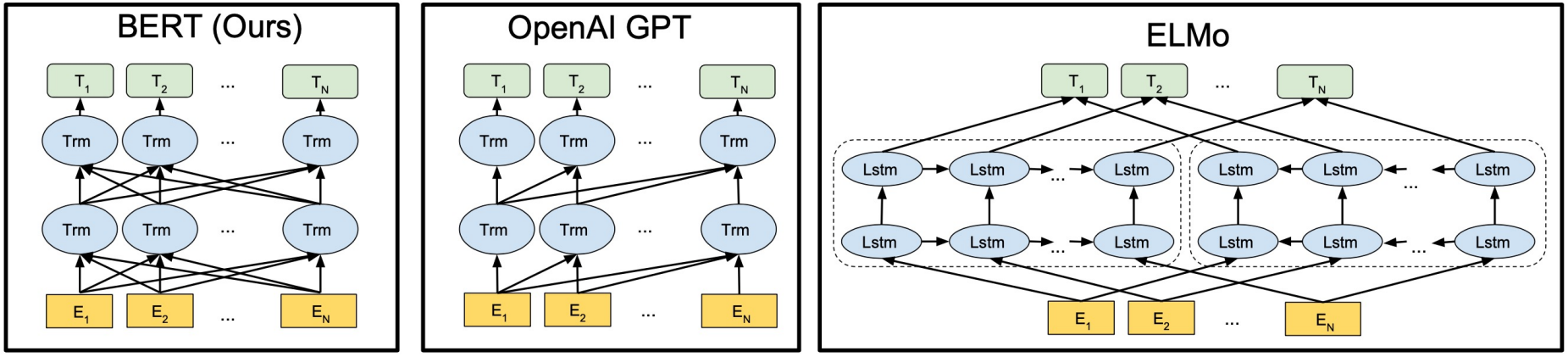
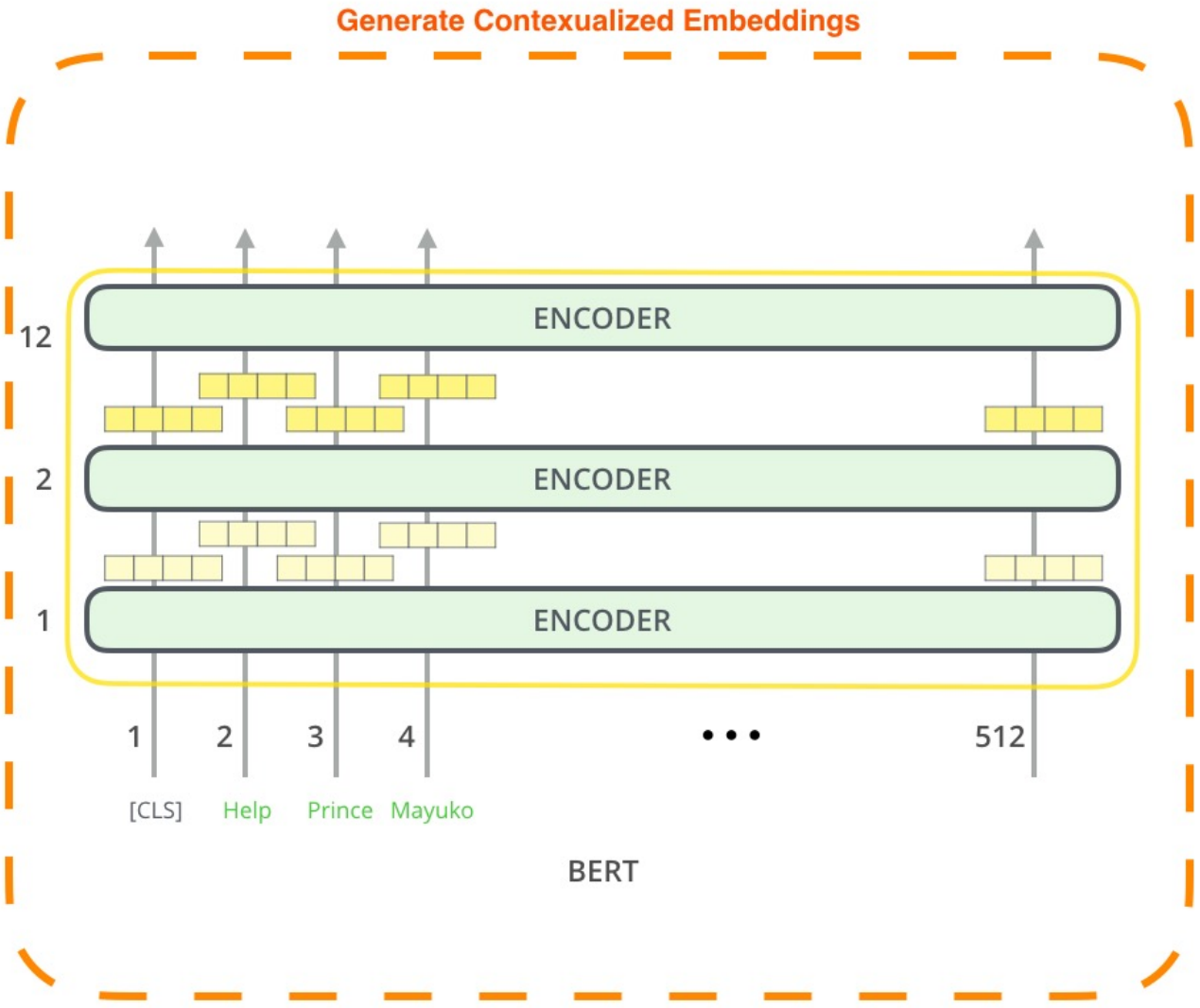


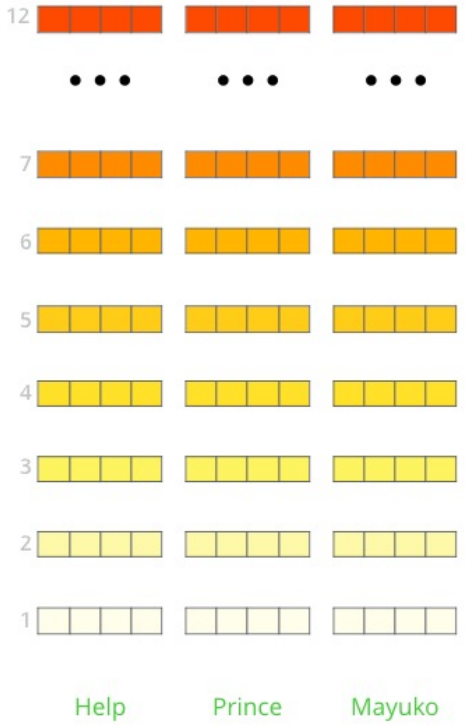
Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

Jacob et al., “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"

BERT for feature extraction



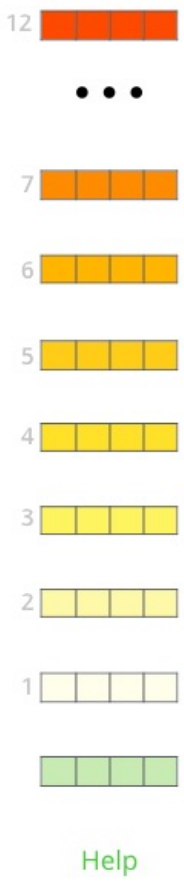
The output of each encoder layer along each token's path can be used as a feature representing that token.

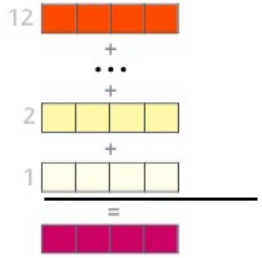
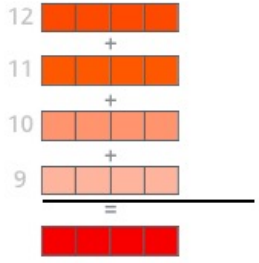
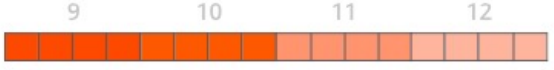


But which one should we use?

BERT for feature extraction

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

BERT for feature extraction

System	Dev F1	Test F1
ELMo (Peters et al., 2018a)	95.7	92.2
CVT (Clark et al., 2018)	-	92.6
CSE (Akbik et al., 2018)	-	**93.1**
Fine-tuning approach		
$\text{BERT}_{\text{LARGE}}$	96.6	92.8
$\text{BERT}_{\text{BASE}}$	96.4	92.4
Feature-based approach ($\text{BERT}_{\text{BASE}}$)		
Embeddings	91.0	-
Second-to-Last Hidden	95.6	-
Last Hidden	94.9	-
Weighted Sum Last Four Hidden	95.9	-
Concat Last Four Hidden	96.1	-
Weighted Sum All 12 Layers	95.5	-

Table 7: CoNLL-2003 Named Entity Recognition results. Hyperparameters were selected using the Dev set. The reported Dev and Test scores are averaged over 5 random restarts using those hyperparameters.

Jacob et al., “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"

BERT application: Question Answering

Question Answering is a **prediction** task:

- Receive a question as input.
- Identify the right answer from some corpus.

→ Given a question and a context paragraph, the model **predicts a start and an end token** from the paragraph that most likely answers the question.

→ Trained by learning two extra vectors that mark the beginning and the end of the answer.

- The question becomes the first sentence and the paragraph is the second sentence in the input sequence.
- Two new parameters learned during fine-tuning: a **start vector** and an **end vector** (i.e., two one-hot encoding vectors).

- Input Question:

  `Where do water droplets collide with ice crystals to form precipitation?`

- Input Paragraph:

  `... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. ...`

- Output Answer:

  `within a cloud`

Summary

- Sequence-to-sequence (seq2seq) model
- Attention model
    - seq2seq with attention
    - seq2seq with bidirectional GRU encoder
    - seq2seq with 2-layer stacked encoder + attention
- Image caption generation.
- Self-attention model
- Transformer network.
    - Encoder architecture
    - Decoder architecture
    - Multi-head attention
    - Masked attention
- Vision transformer.
- BERT

Q&A