Lecture 8-1: Seq2Seq Learning & Transformer

Pilsung Kang
School of Industrial Management Engineering
Korea University

AGENDA

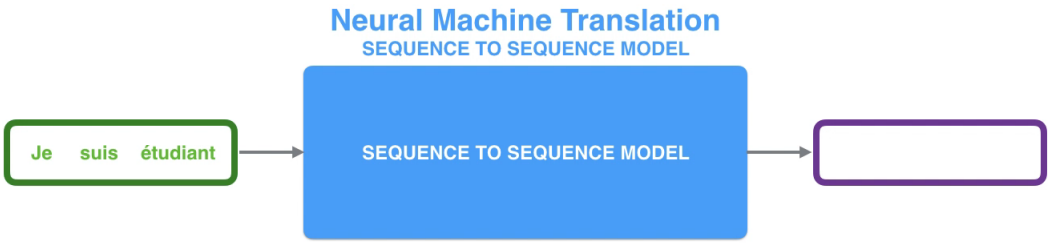
01 Sequence to Sequence (Seq2seq) Learning

02 Transformer

Sequence to Sequence Learning

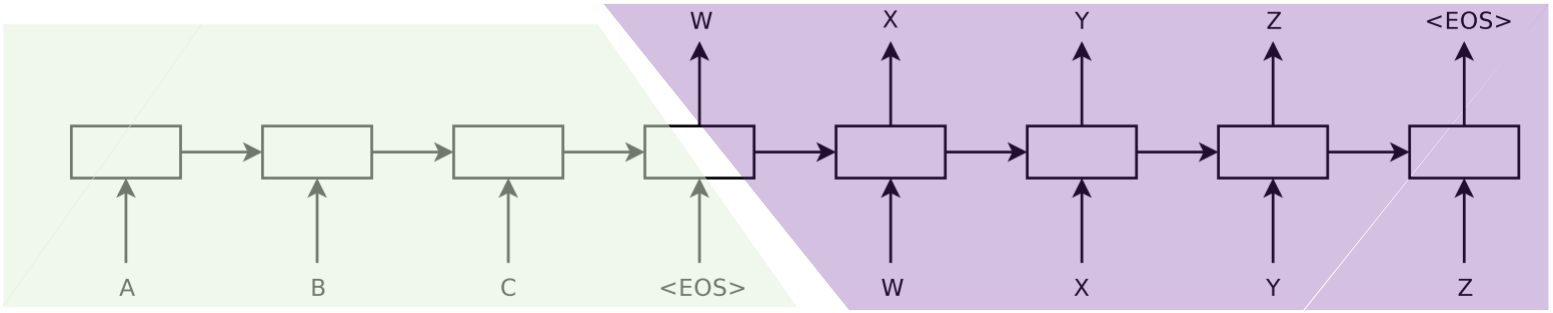
- Sequence-to-sequence model (Sutskever et al., 2014, Cho et al., 2014)
  - ✓ A model that takes a sequence of items (words, letters, features of an images, etc.)
  - ✓ Outputs another sequence of items
    - A trained model

    - Neural machine translation

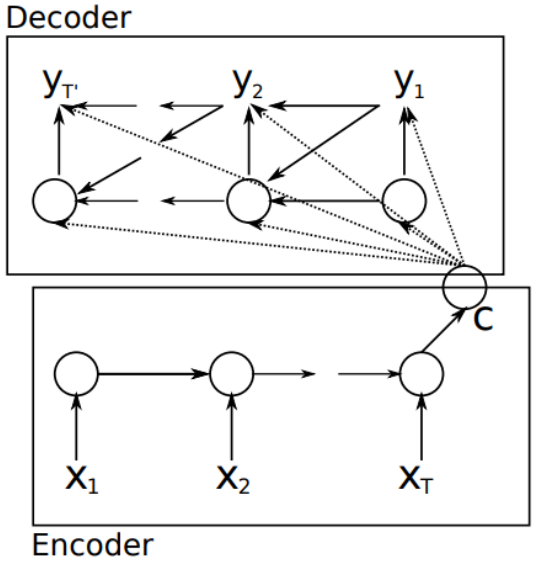


Sequence to Sequence Learning

- Main idea
  - ✓ Seq2Seq model consists of an encoder and a decoder



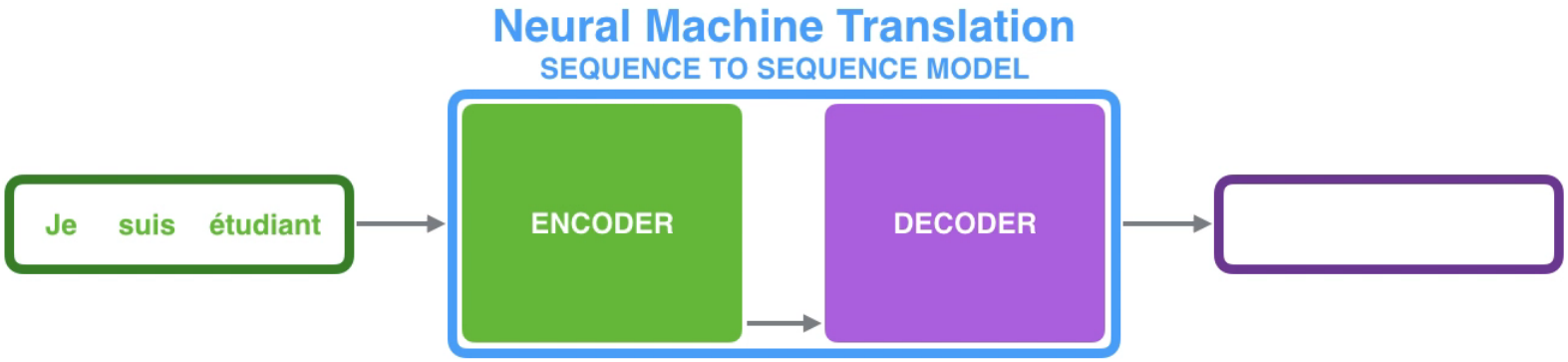
Sutskever et al., 2014



Cho et al., 2014

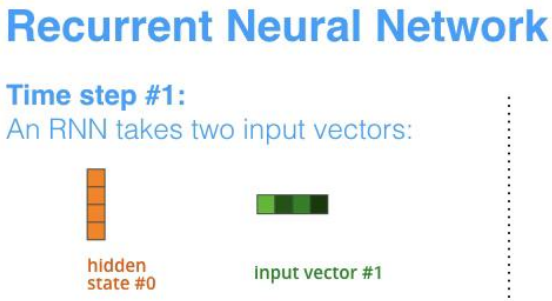
Sequence to Sequence Learning

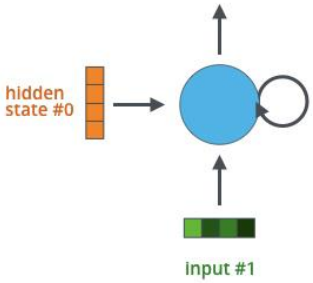
- Encoder-Decoder
  - ✓ The encoder processes each item in the input sequence and compiles the information it captures into a vector (context)
  - ✓ After processing the entire input sequence, the encoder send the context over to the decoder, which begins producing the output sequence item by item

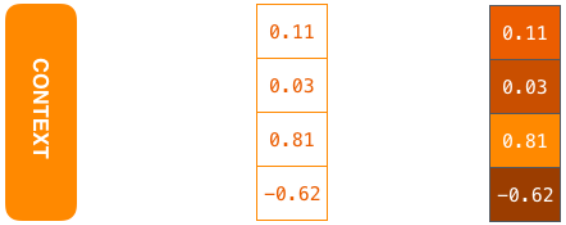


Sequence to Sequence Learning

- Encoder-Decoder
  - ✓ Recurrent neural network (RNN) is commonly used for the encoder and decoder structure
  - ✓ The context is a vector in the case of machine translation

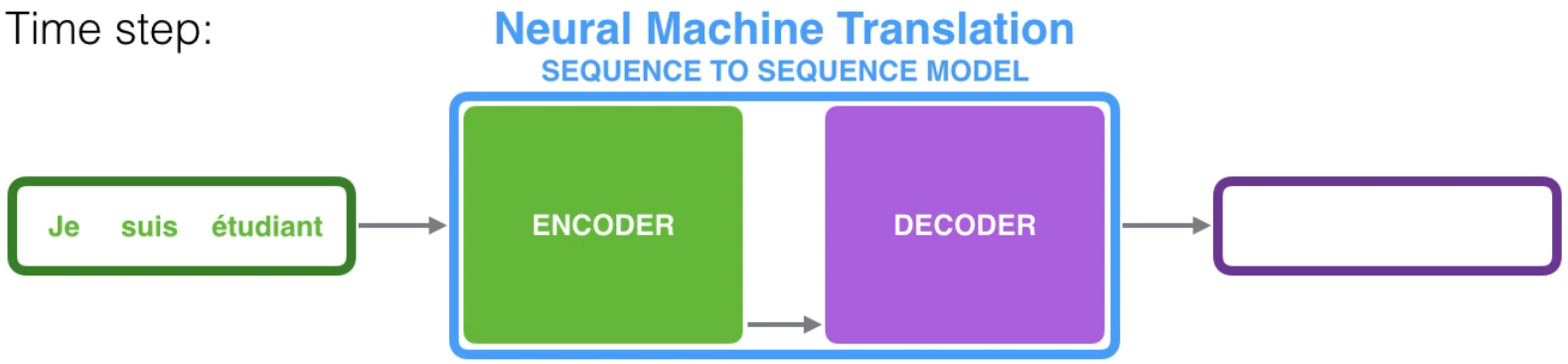






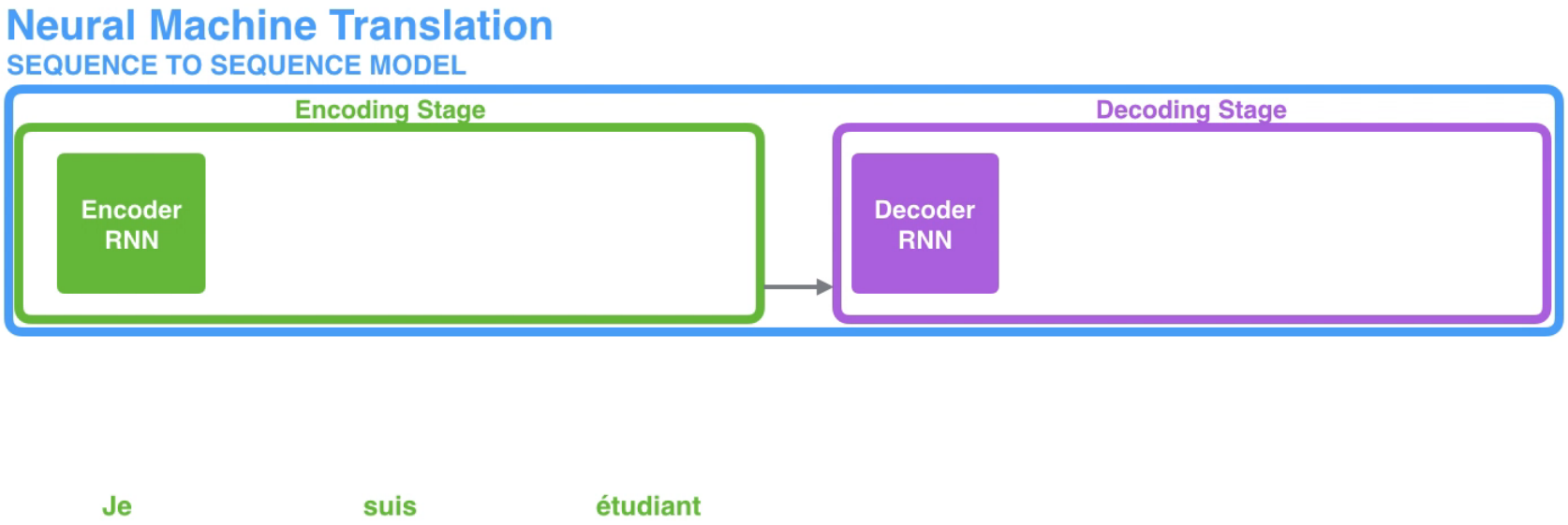
Sequence to Sequence Learning

- Encoder-Decoder
  - ✓ Each purse for the encoder or decoder is that RNN processes its inputs and generates an output for that time step



Sequence to Sequence Learning

- An unrolled view of Seq2Seq learning



Attention in Seq2Seq Learning

Alamar (Attention)

- Attention
  - ✓ Context vector is a bottleneck for these types of models, which makes it challenging for the models to deal with long sentences
  - ✓ Attention allows the model to focus on the relevant part of the input sequence as needed

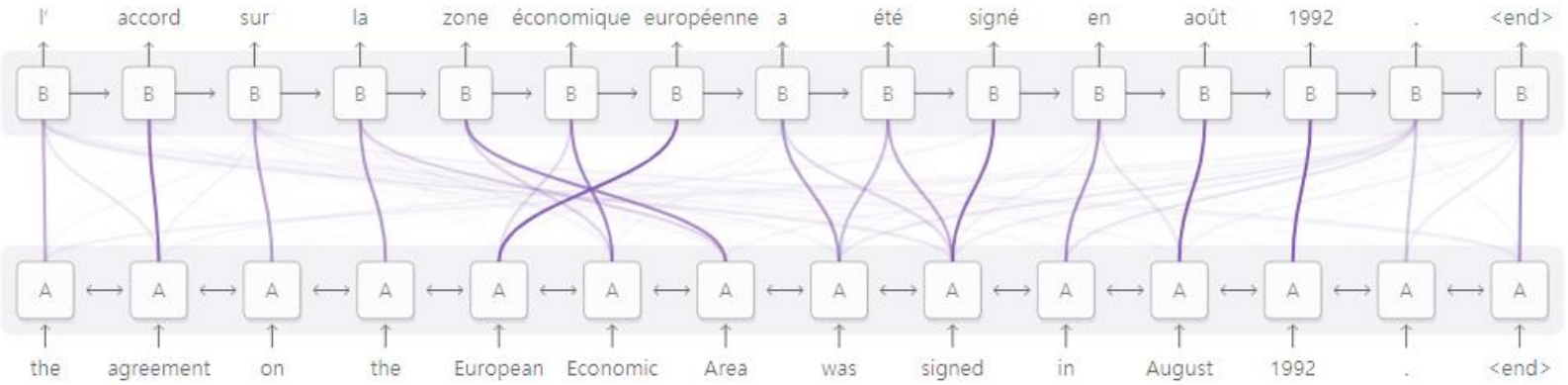


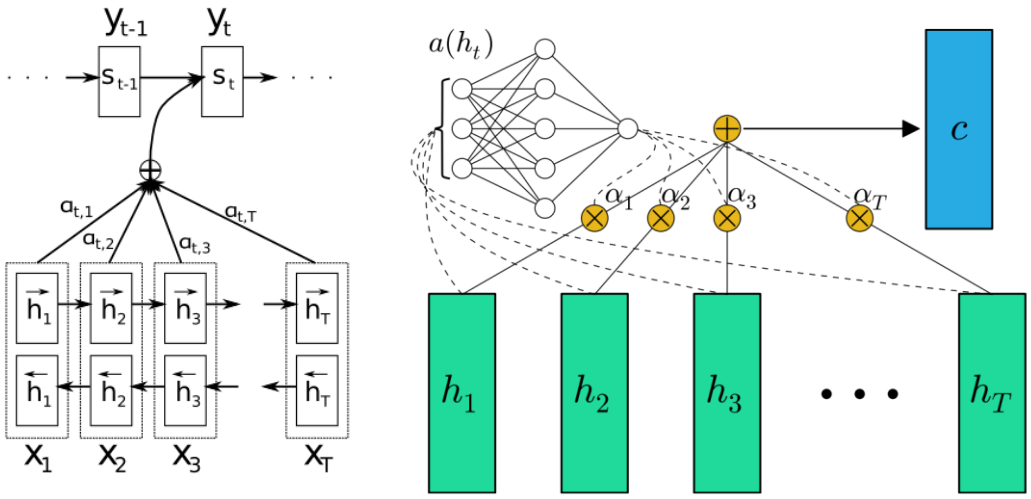
Diagram derived from Fig. 3 of Bahdanau, *et al.* 2014

Olah (2016)

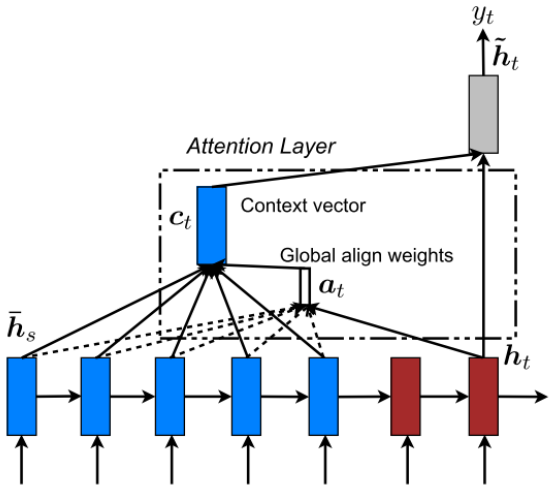
Attention in Seq2Seq Learning

- Attention
  - ✓ Bahadanau attention (Bahdanau et al., 2015)
    - Attention scores are separated trained, the current hidden state is a function of the context vector and the previous hidden state
  - ✓ Luong attention (Luong et al., 2015)
    - Attention scores are not trained, the new current hidden state is the simple tanh of the weigthed concatenation of the context vector and the current hidden state of the decoder

Bahdanau attention

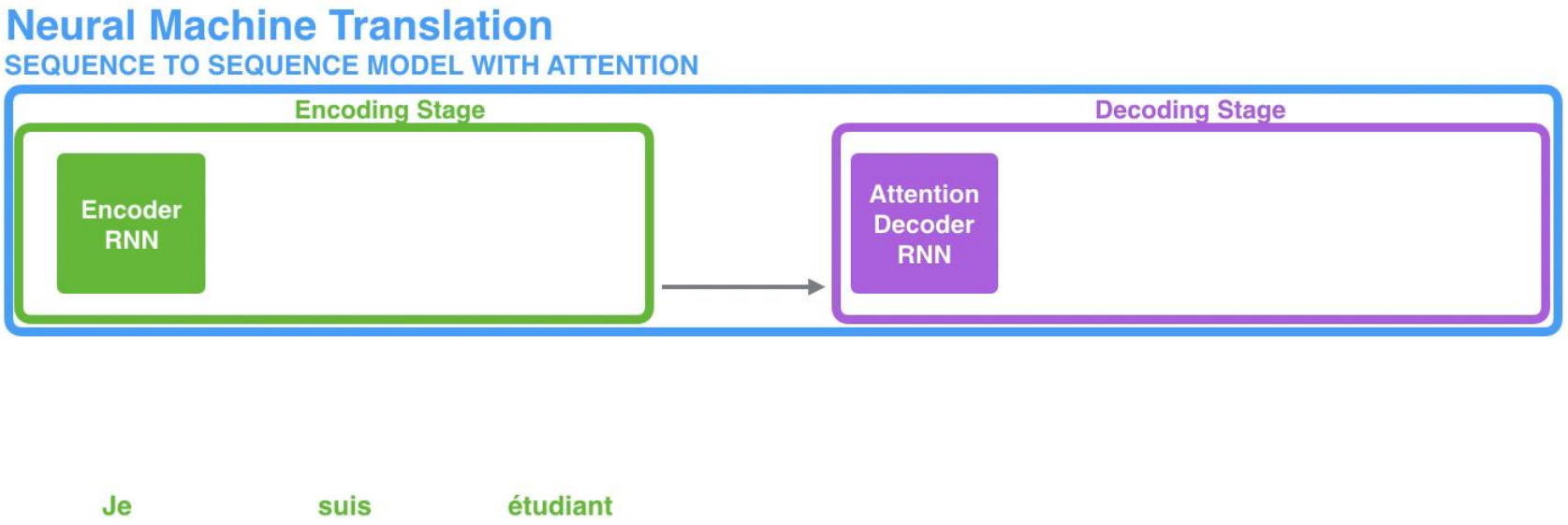


Luong attention



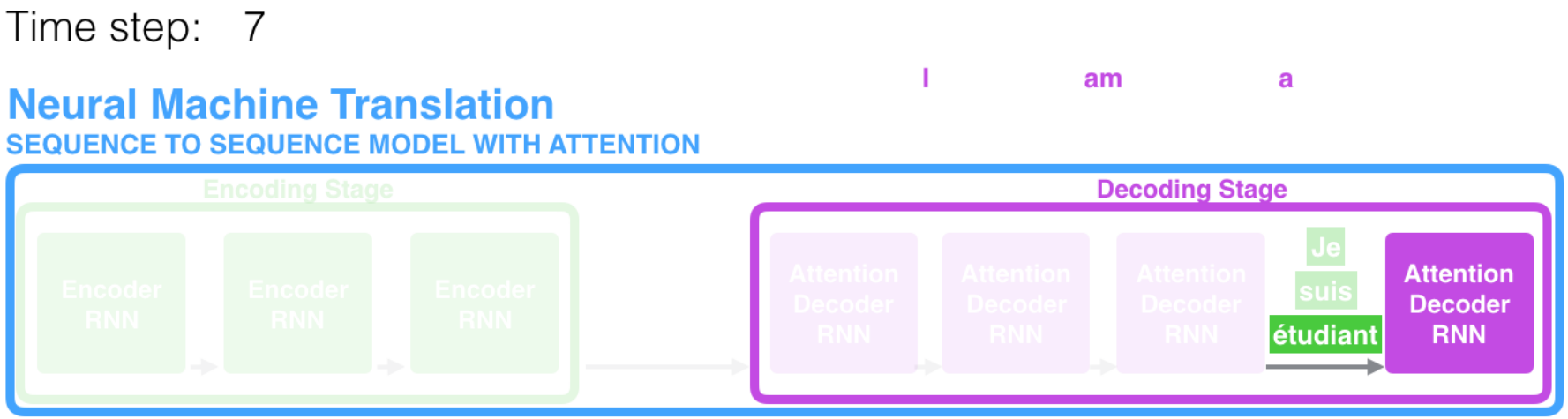
Attention in Seq2Seq Learning

- Attention model differs from a classic Seq2Seq model in two main ways:
  - ✓ The encoder passes a lot more data to the decoder
    - Instead of passing the last hidden state of the encoding stage, the encoder passes all the hidden states to the decoder



Attention in Seq2Seq Learning

- Attention model differs from a classic Seq2Seq model in two main ways:
  - ✓ The encoder passes a lot more data to the decoder
    - Instead of passing the last hidden state of the encoding stage, the encoder passes all the hidden states to the decoder



Attention in Seq2Seq Learning

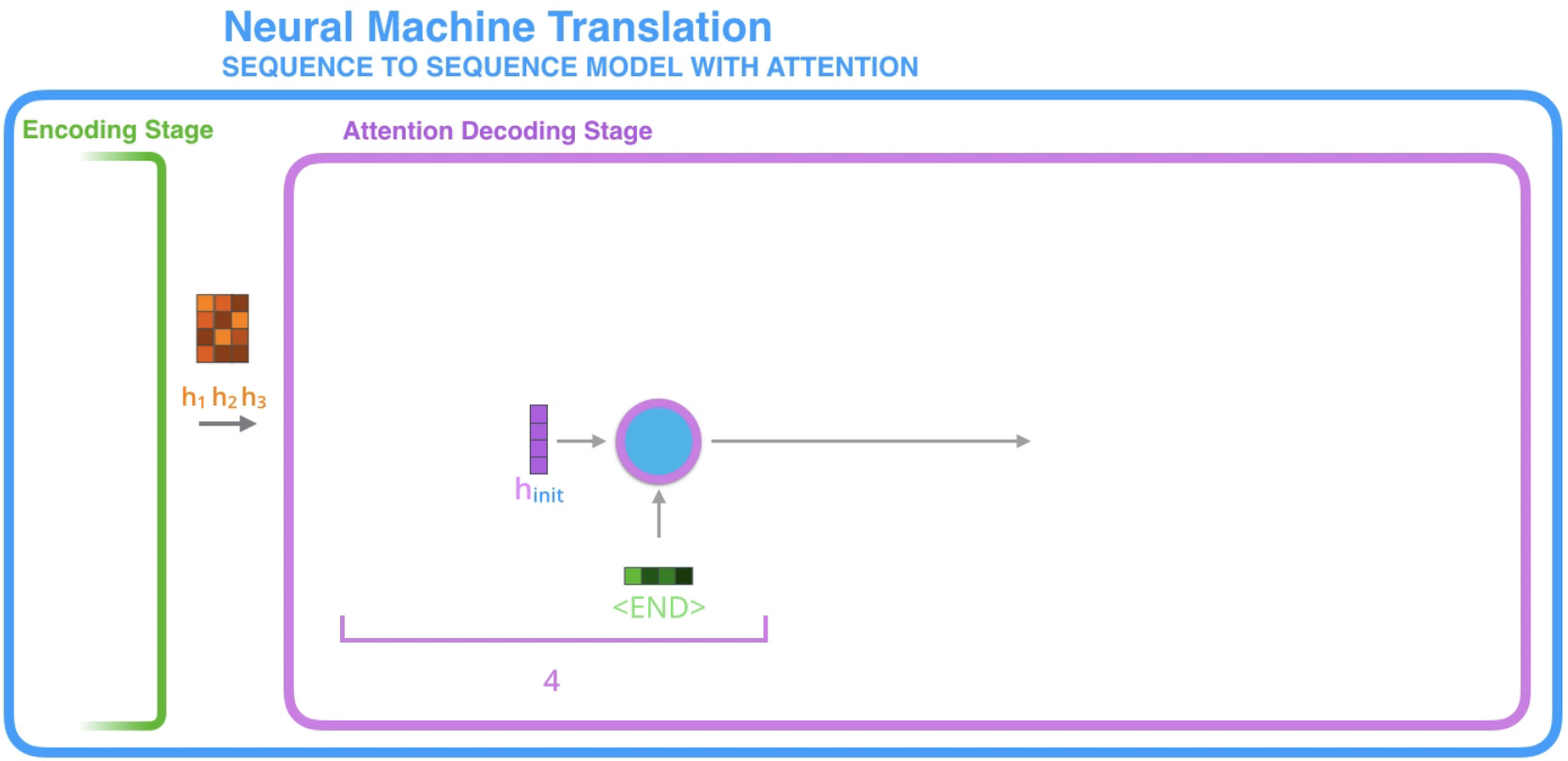
- Attention model differs from a classic Seq2Seq model in two main ways:
  - ✓ An attention decoder does an extra step before producing its output
    - Look at the set of encoder hidden states it received – each encoder hidden states is most associated with a certain word in the input sentence
    - Give each hidden states a score
    - Multiply each hidden state by its softmaxed score, this amplifying hidden state with high scores, an drowning out hidden state with low scores

Attention in Seq2Seq Learning

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

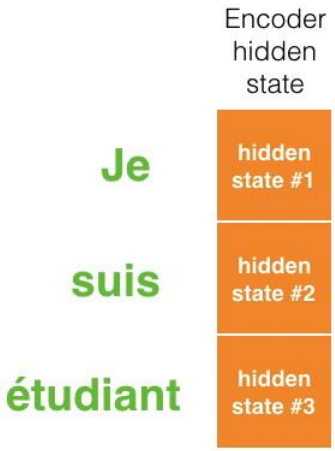
Attention in Seq2Seq Learning

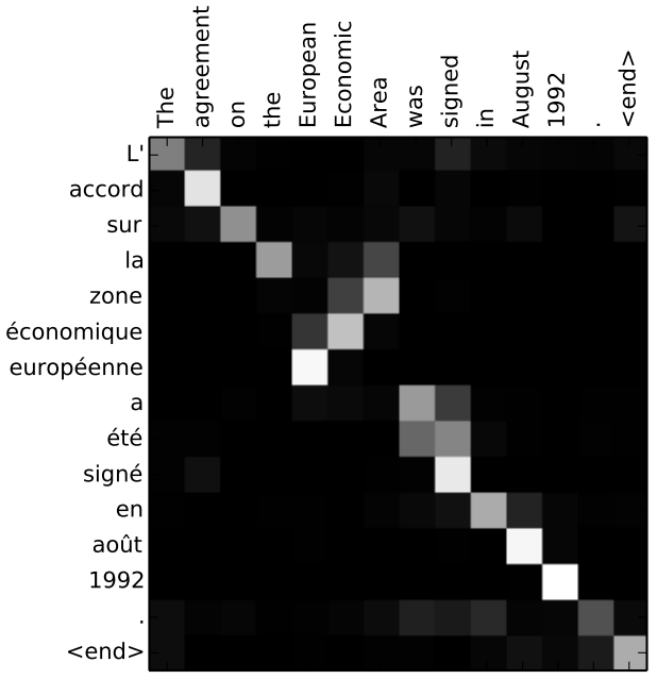
- Working mechanism of attention process



Attention in Seq2Seq Learning

- Working mechanism of attention process



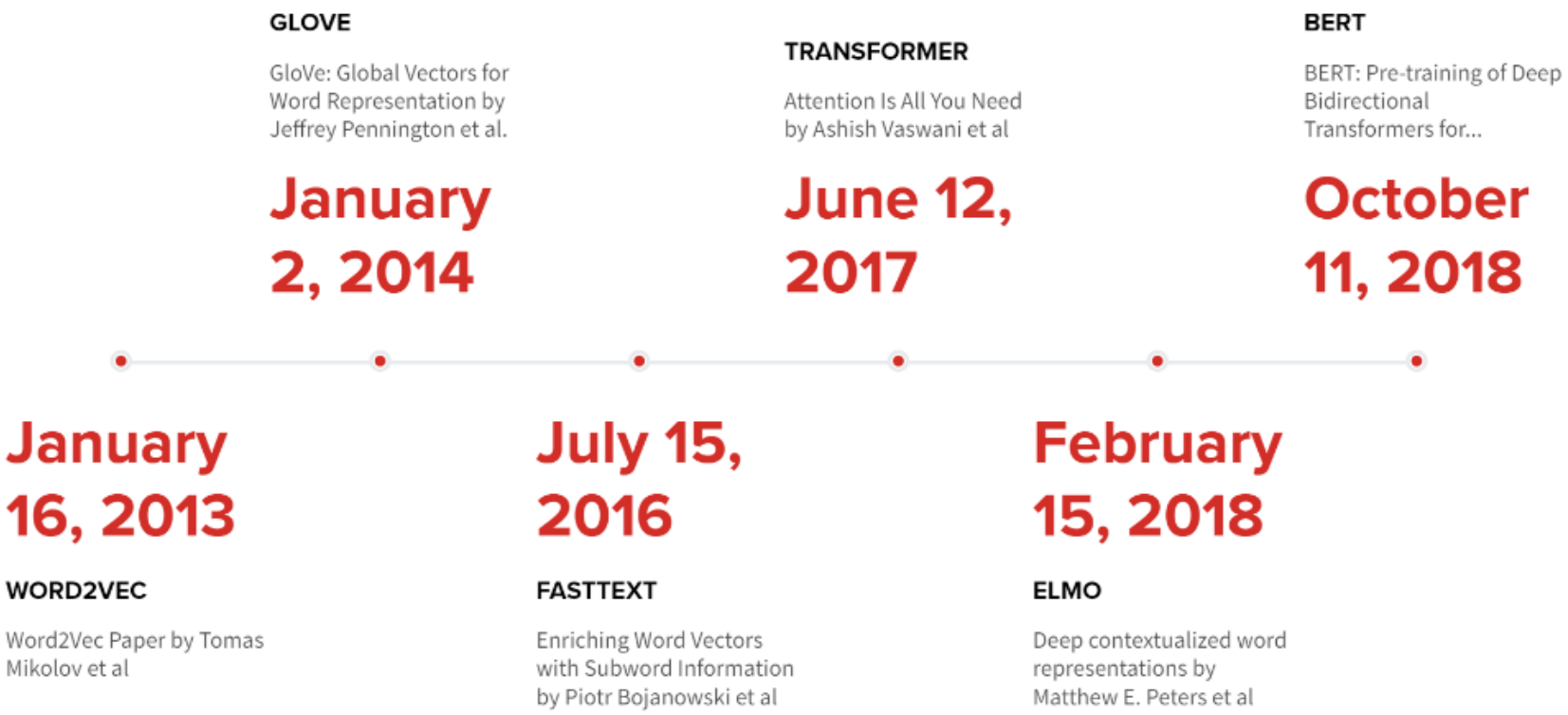


AGENDA

01 Sequence to Sequence (Seq2seq) Learning

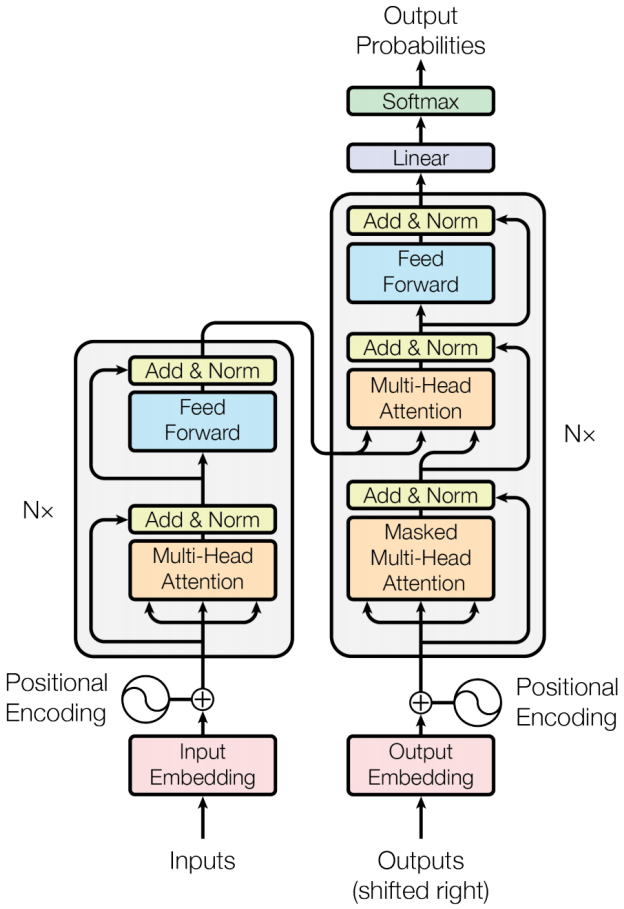
02 Transformer

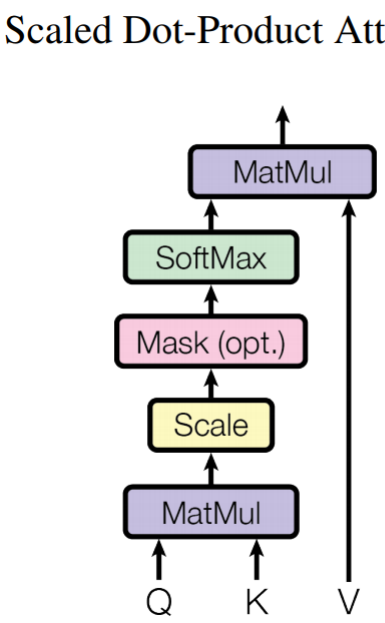
Transformer: Self-Attention

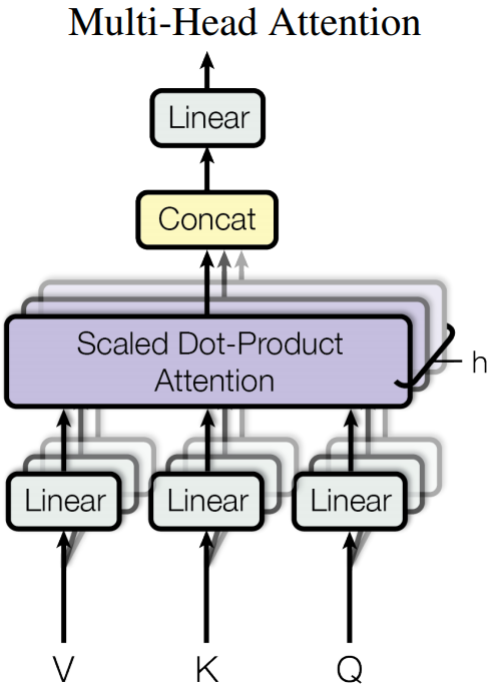


Transformer: Self-Attention

- Transformer (Vaswani et al., 2017)
  - ✓ A model that uses attention to boost the speed with which these models can be trained and easy to parallelize



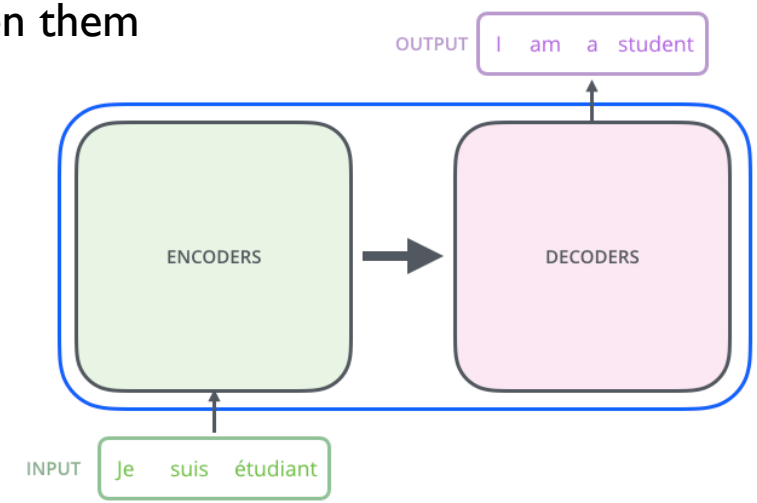




Transformer: Self-Attention

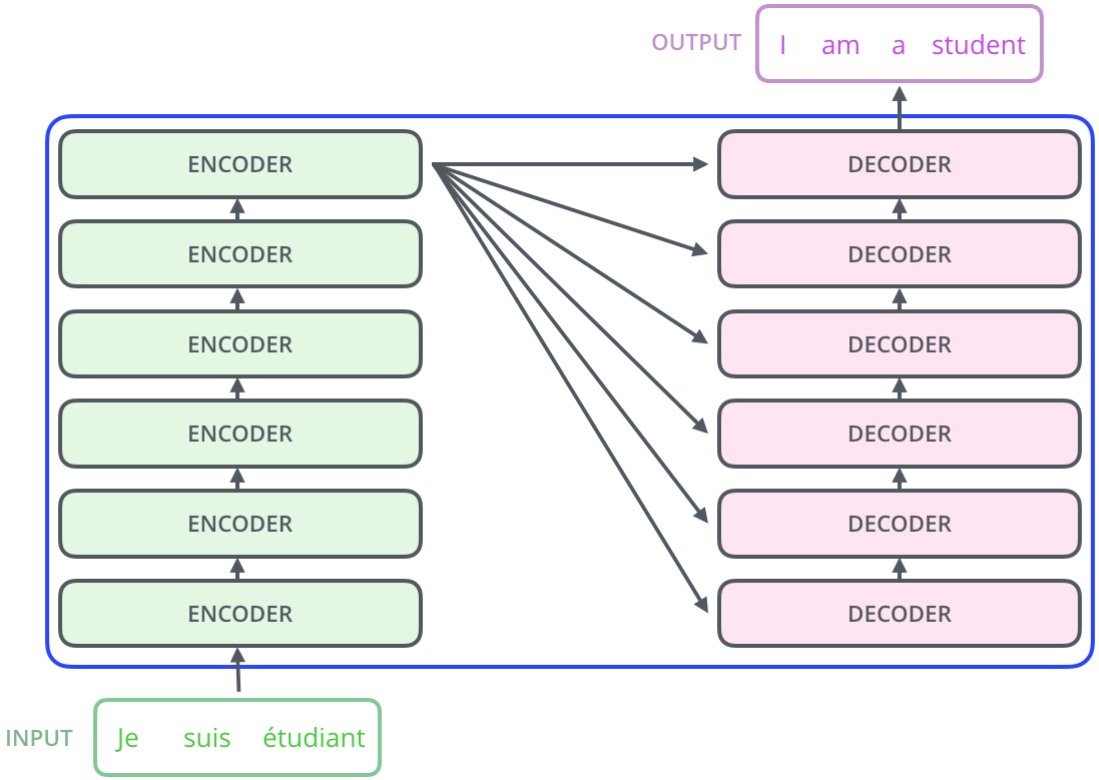
- Transformer (Vaswani et al., 2017)
  - ✓ A model that uses attention to boost the speed with which these models can be trained and easy to parallelize
  - ✓ A high level look

  - ✓ Inside the transformer, there are an encoding component and a decoding components and connections between them



Transformer: Self-Attention

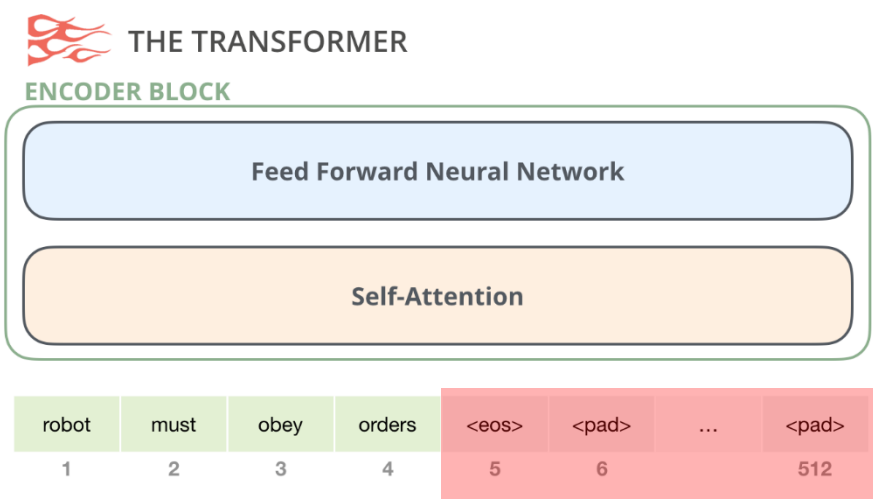
- The encoding component is a stack of encoders
  - ✓ The original paper stacks six of them on top of each other, but there is nothing magical about the number six
- The decoding component is a stack of decoders of the same number

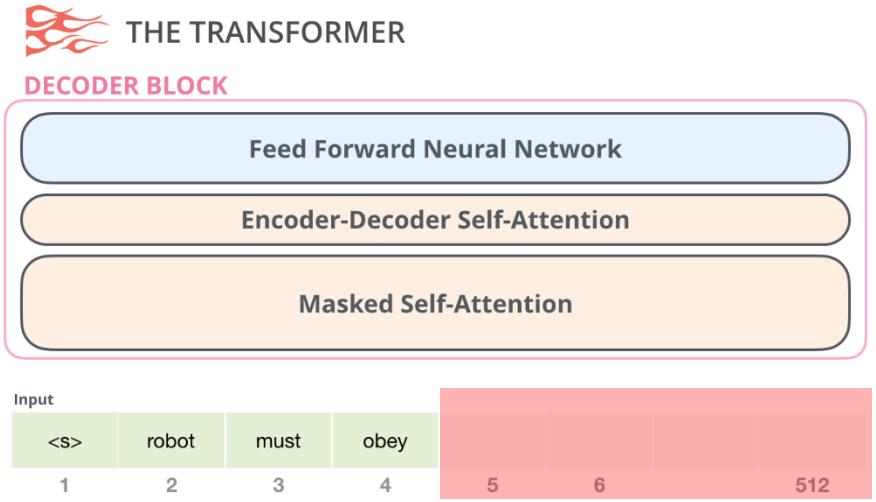


Transformer: Self-Attention

Alamar (Transformer)

- Encoding block vs. Decoding block = Unmasked vs. Masked

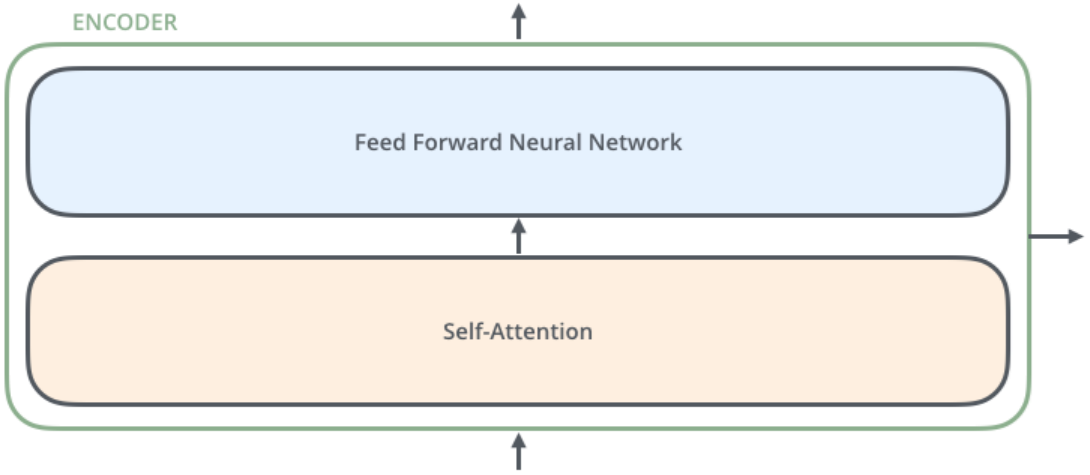




Transformer: Self-Attention

Alamar (Transformer)

- The encoder are all identical in structure (does not mean that they share the weights), each of which is broken down into two sub-layers



✓ The encoder's input first flow through a self-attention layer (a layer that helps the encoder look at other words in the input sentence as it encodes a specific word)

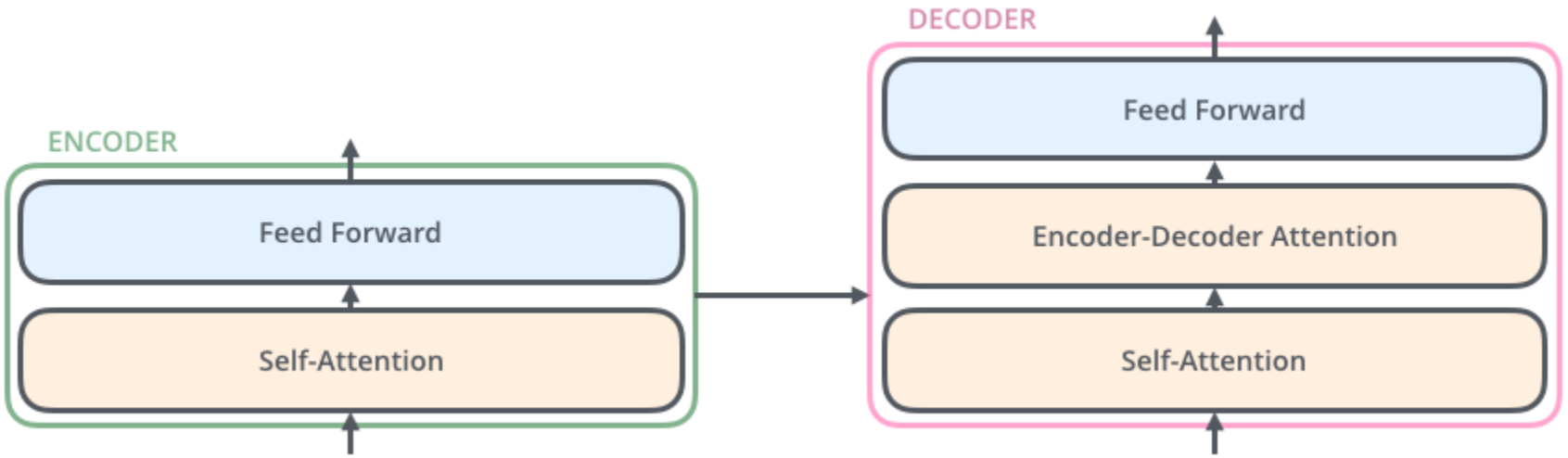
✓ The output of the self-attention layer are fed to a feed-forward neural network

- The exact same feed-forward network is idependently applied to each position

Transformer: Self-Attention

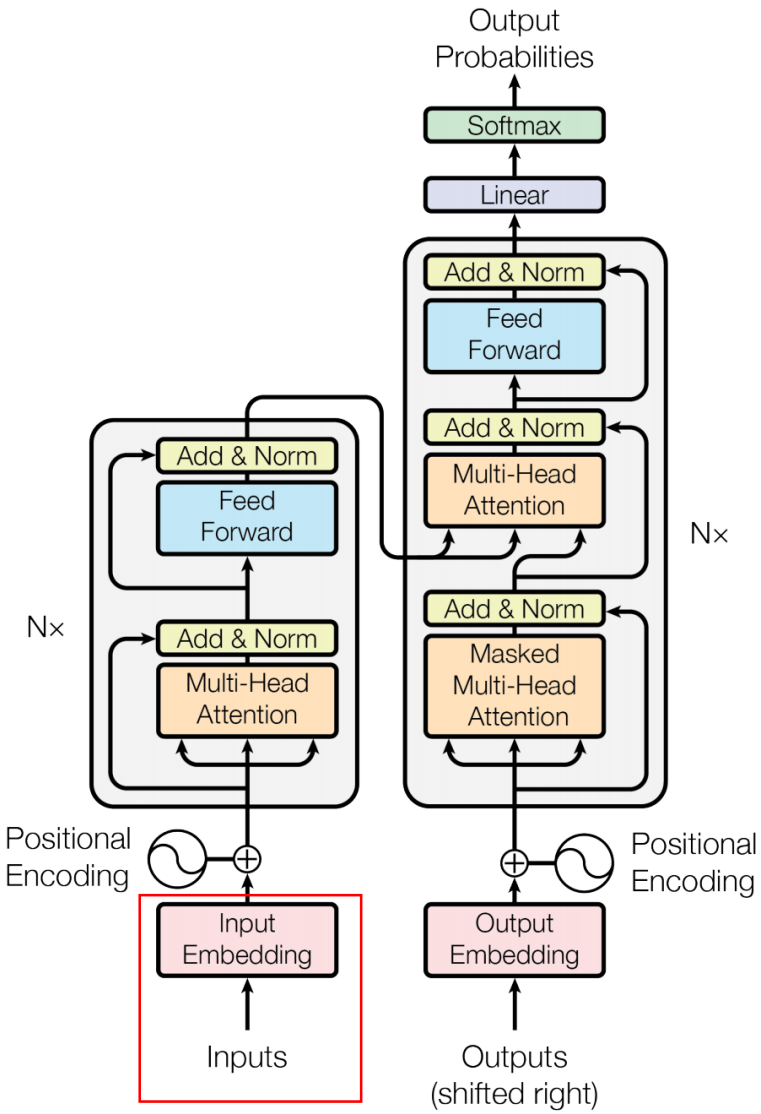
Alamar (Transformer)

- The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence



Transformer: Self-Attention

- Input Embeddings



Transformer: Self-Attention

Alamar (Transformer)

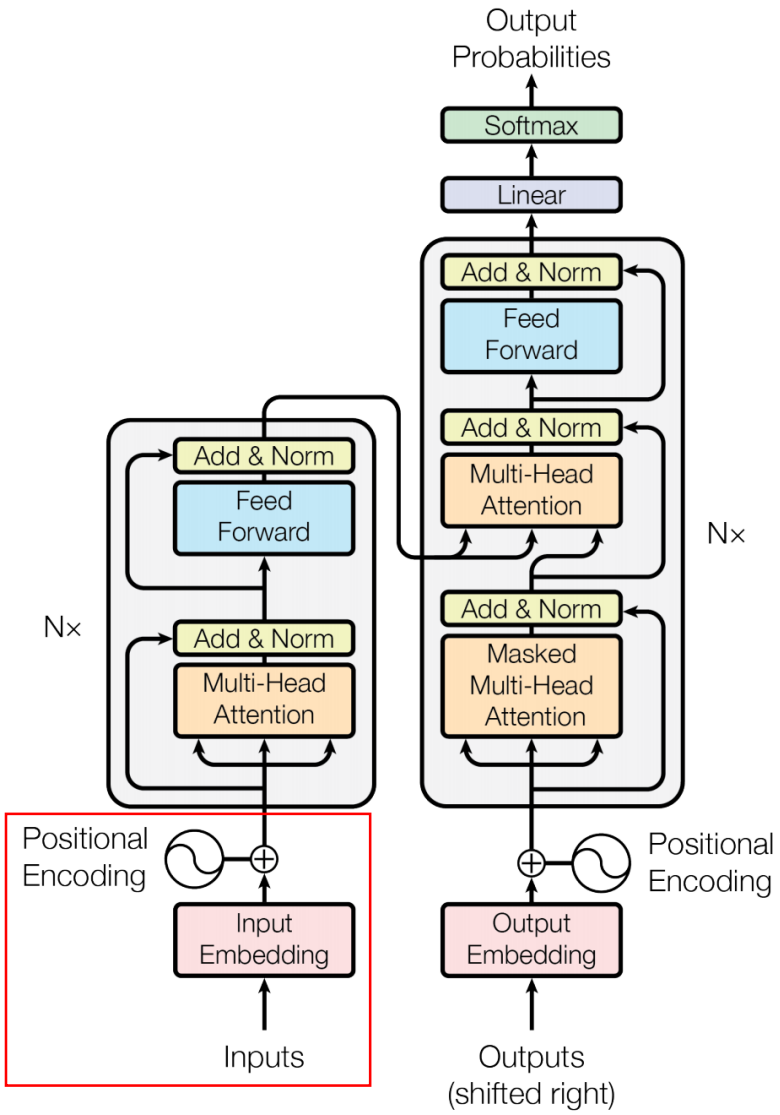
- More specific explanation
  - ✓ Let's begin by turning each input word into a vector using an embedding algorithm

Each word is embedded into a vector of size 512. We'll represent those vectors with these simple boxes.

- The embedding only happens in the bottom-most encoder
- The abstraction that is common to all the encoders is that they receive a list of vectors each of the size 512
- In the bottom encoder that would be the word embeddings, but in other encoders, it would be the output of the encoder that is directly below
- The size of this list is a hyperparameter we can set – basically it would be the length of the longest sentence in our training dataset

Transformer: Self-Attention

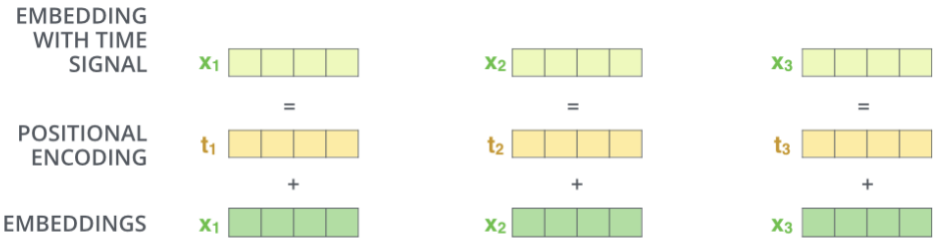
- Positional Encoding

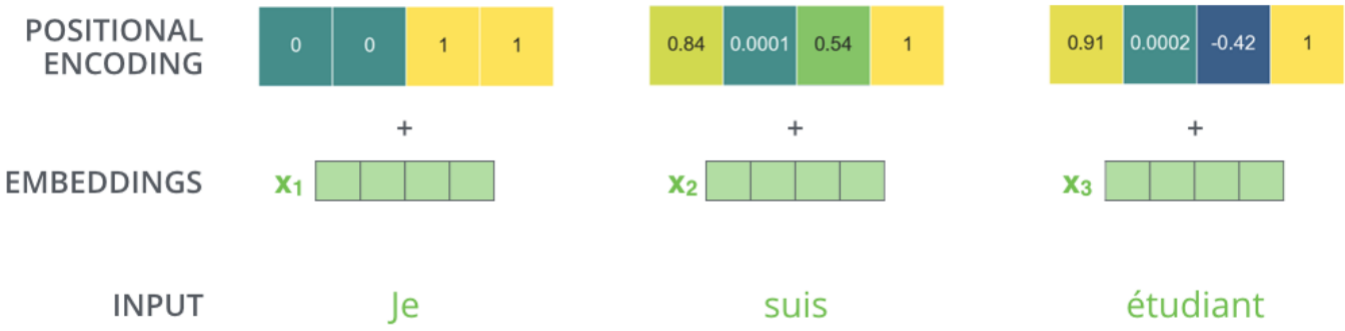


Transformer: Self-Attention

Alamar (Transformer)

- Positional encoding
  - ✓ A way to account for the order of the words in the input sequence
  - ✓ A vector added to each input embedding
    - Provides meaningful distances between the embedding vectors once they are projected into Q/K/V vectors and during dot-product attention



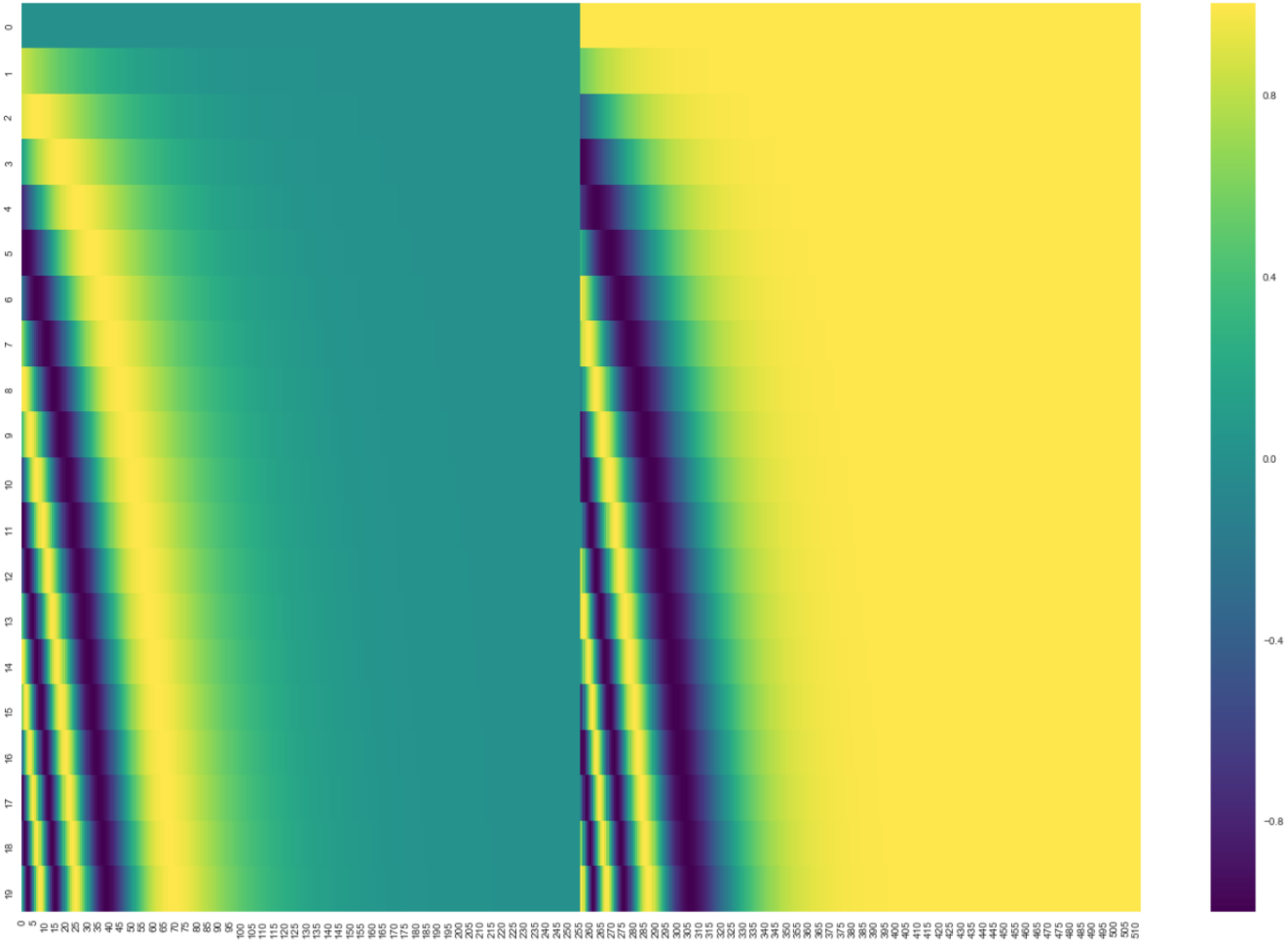


A real example of positional encoding with a toy embedding size of 4

Transformer: Self-Attention

Alamar (Transformer)

- Positional encoding



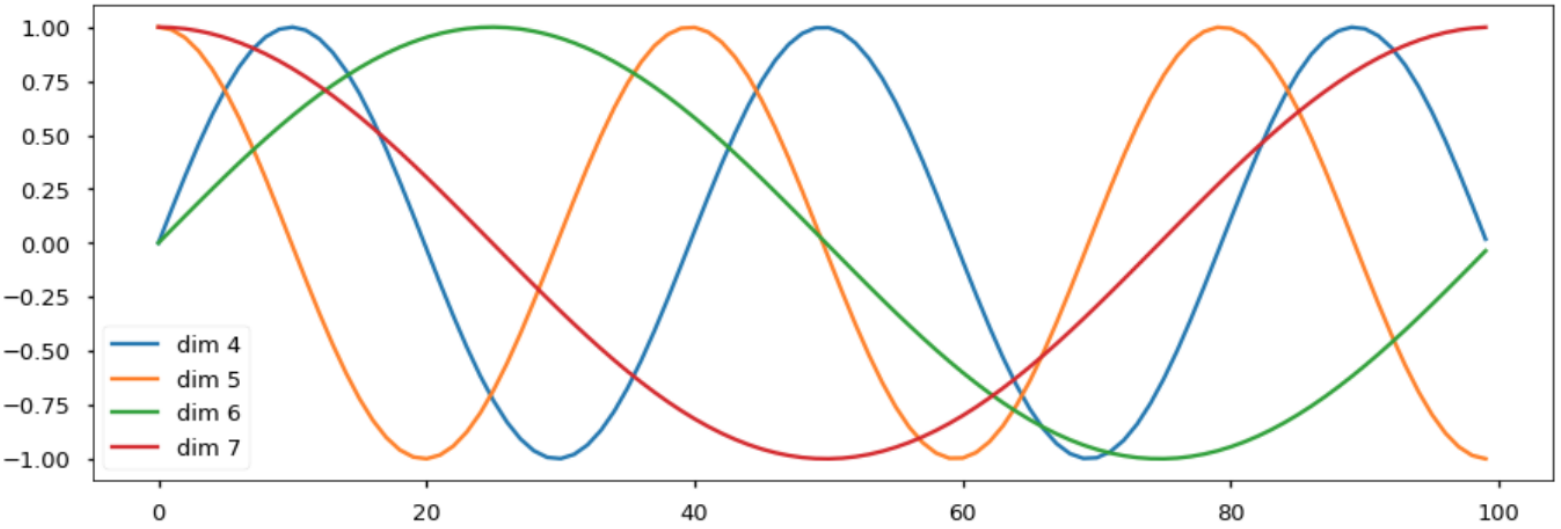
A real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns). You can see that it appears split in half down the center. That's because the values of the left half are generated by one function (which uses sine), and the right half is generated by another function (which uses cosine). They're then concatenated to form each of the positional encoding vectors.

Transformer: Self-Attention

- Positional encoding

> *Below the positional encoding will add in a sine wave based on position. The frequency and offset of the wave is different for each dimension.*

```
plt.figure(figsize=(15, 5))
pe = PositionalEncoding(20, 0)
y = pe.forward(Variable(torch.zeros(1, 100, 20)))
plt.plot(np.arange(100), y[0, :, 4:8].data.numpy())
plt.legend(["dim %d"%p for p in [4,5,6,7]])
None
```



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

Transformer: Self-Attention

- Two properties that a good positional encoding scheme should have
  - ✓ The norm of encoding vector is the same for all positions
  - ✓ The further the two positions, the larger the distance
    - A Simple Example (n = 10, dim = 10)

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

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

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
X1	0.000	1.000	0.000	1.000	0.000	1.000	0.000	1.000	0.000	1.000
X2	0.841	0.674	0.638	0.839	0.461	0.922	0.325	0.962	0.227	0.982
X3	0.909	-0.093	0.983	0.408	0.818	0.699	0.615	0.852	0.442	0.928
X4	0.141	-0.798	0.875	-0.155	0.991	0.368	0.838	0.678	0.634	0.841
X5	-0.757	-0.983	0.366	-0.668	0.942	-0.022	0.970	0.452	0.793	0.723
X6	-0.959	-0.526	-0.312	-0.965	0.680	-0.408	0.996	0.192	0.911	0.579
X7	-0.279	0.275	-0.847	-0.952	0.267	-0.730	0.915	-0.082	0.981	0.415
X8	0.657	0.896	-0.992	-0.632	-0.207	-0.938	0.734	-0.350	0.999	0.235
X9	0.989	0.932	-0.681	-0.109	-0.635	-0.999	0.473	-0.591	0.966	0.046
X10	0.412	0.360	-0.057	0.450	-0.919	-0.904	0.161	-0.788	0.882	-0.144

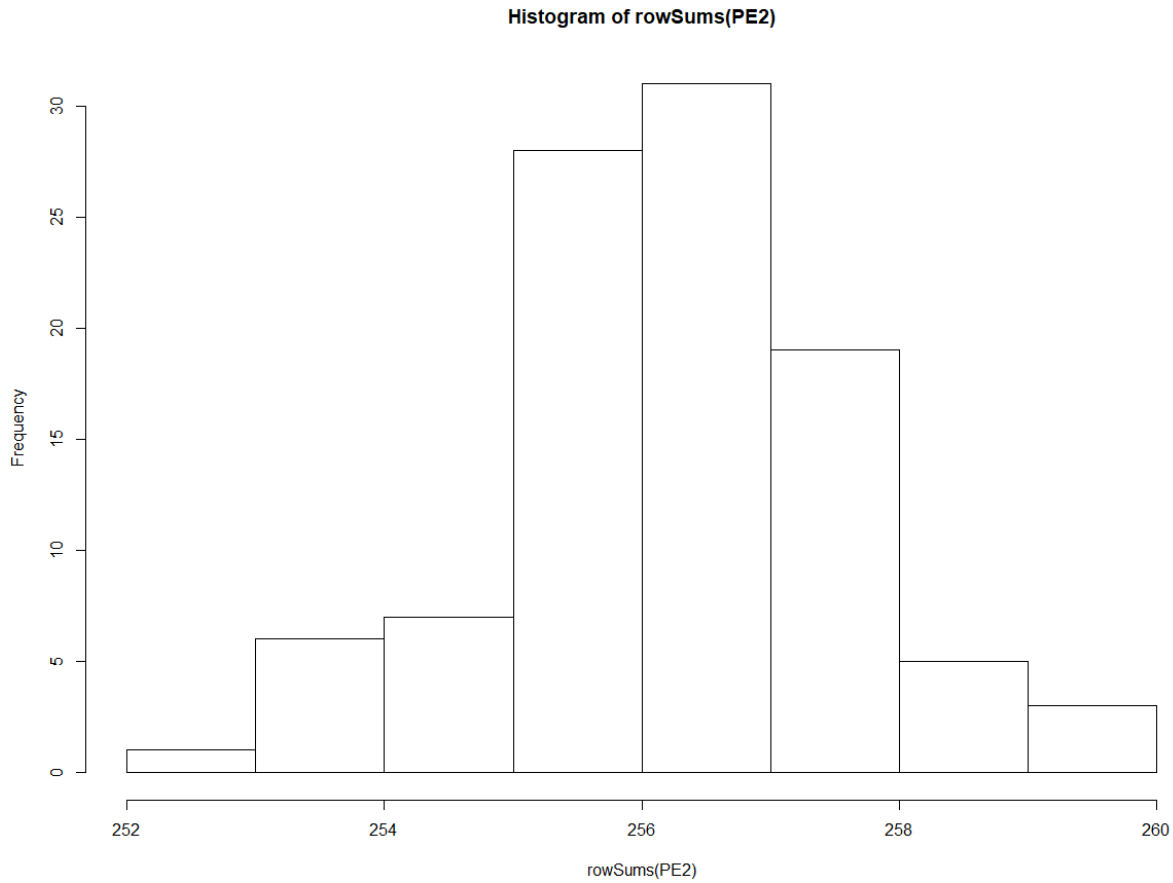
Transformer: Self-Attention

- A Simple Example (n = 10, dim = 10)
  - ✓ Distances between two positional encoding vectors

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
X1	0.000	1.275	2.167	2.823	3.361	3.508	3.392	3.440	3.417	3.266
X2	1.275	0.000	1.104	2.195	3.135	3.511	3.452	3.442	3.387	3.308
X3	2.167	1.104	0.000	1.296	2.468	3.067	3.256	3.464	3.498	3.371
X4	2.823	2.195	1.296	0.000	1.275	2.110	2.746	3.399	3.624	3.399
X5	3.361	3.135	2.468	1.275	0.000	1.057	2.176	3.242	3.659	3.434
X6	3.508	3.511	3.067	2.110	1.057	0.000	1.333	2.601	3.169	3.118
X7	3.392	3.452	3.256	2.746	2.176	1.333	0.000	1.338	2.063	2.429
X8	3.440	3.442	3.464	3.399	3.242	2.601	1.338	0.000	0.912	1.891
X9	3.417	3.387	3.498	3.624	3.659	3.169	2.063	0.912	0.000	1.277
X10	3.266	3.308	3.371	3.399	3.434	3.118	2.429	1.891	1.277	0.000

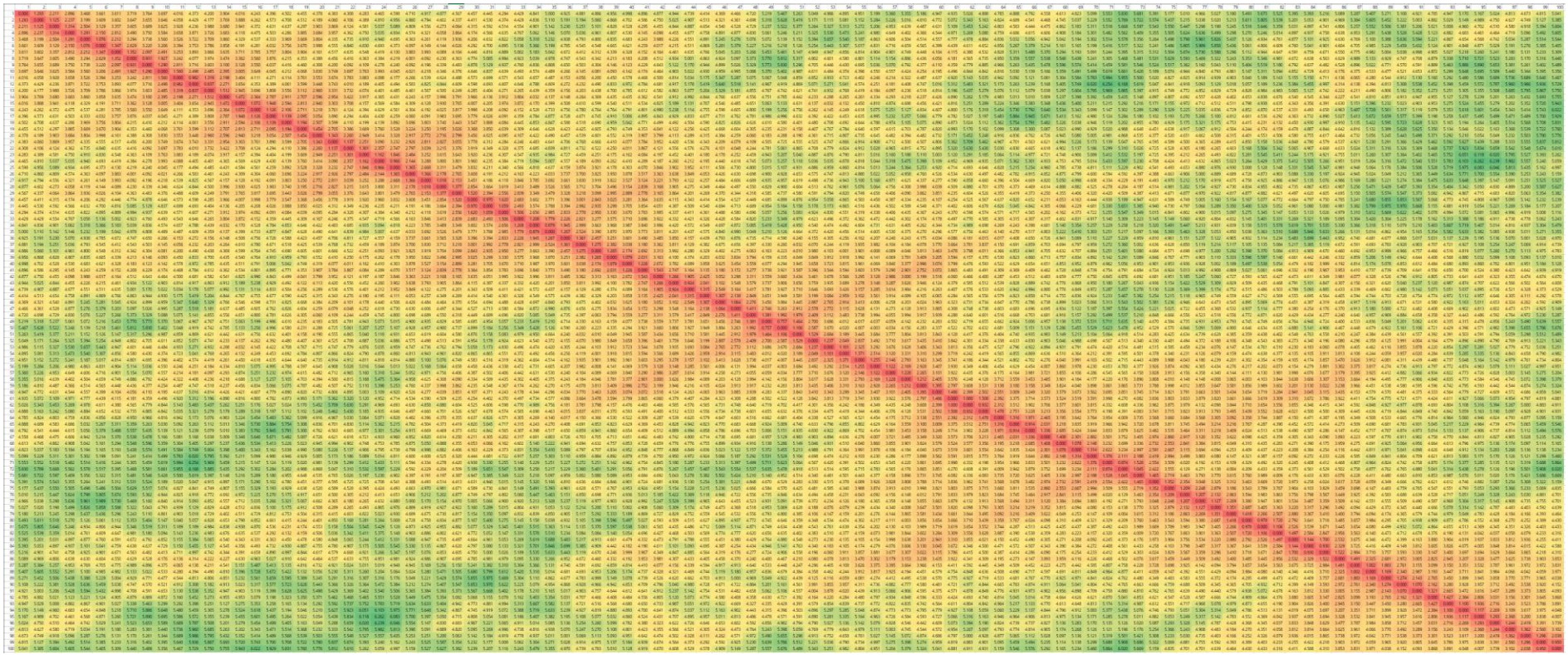
Transformer: Self-Attention

- Two properties that a good positional encoding scheme should have
  - ✓ The norm of encoding vector is the same for all positions
    - What if n = 100 and dim = 512?
    - L_2-norm distribution
      - Mean: 256.25
      - Std: 1.35



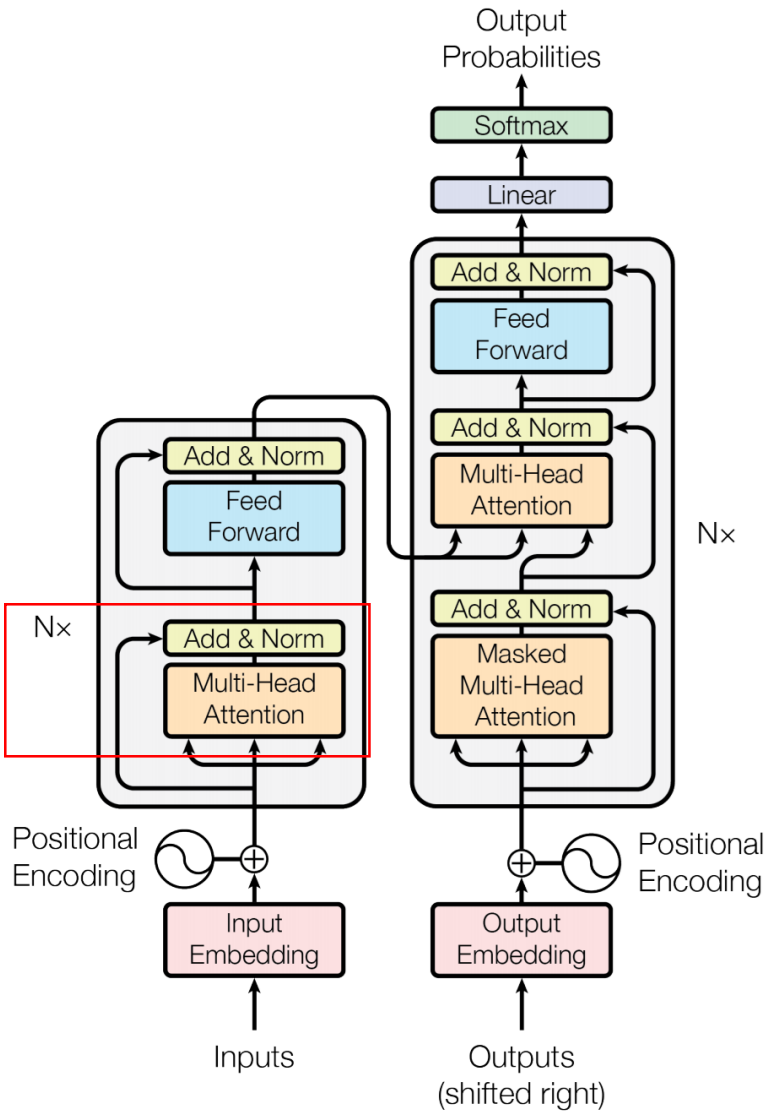
Transformer: Self-Attention

- Two properties that a good positional encoding scheme should have
  - ✓ The further the two positions, the larger the distance



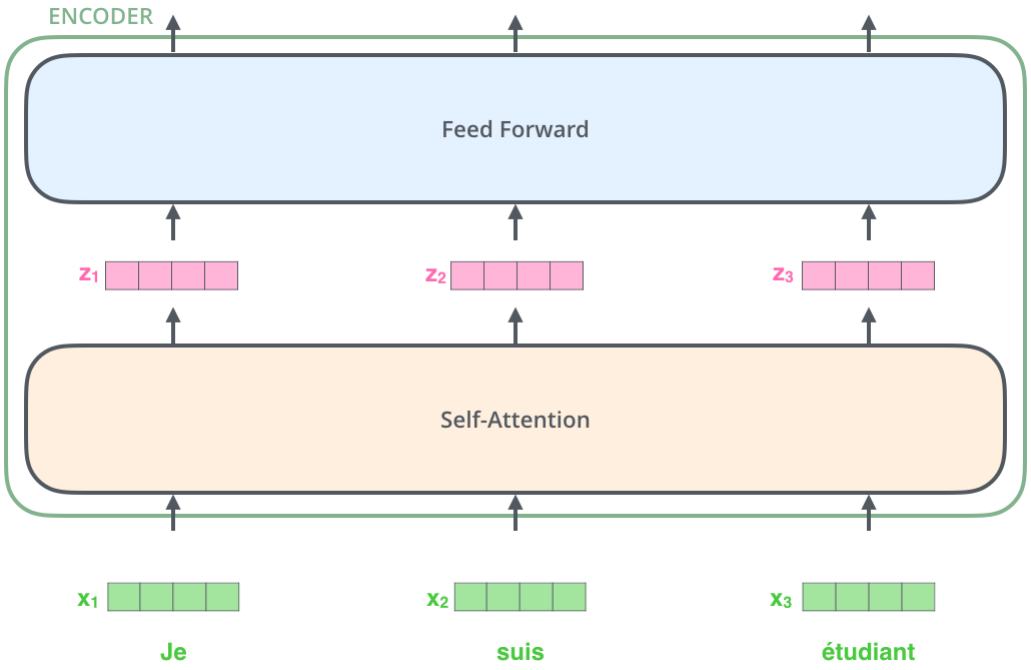
Transformer: Self-Attention

- Multi-Head Attention
- Residual connection & Normalization



Transformer: Self-Attention

- After embedding the words, each of them flows through each of the two layers of the encoder



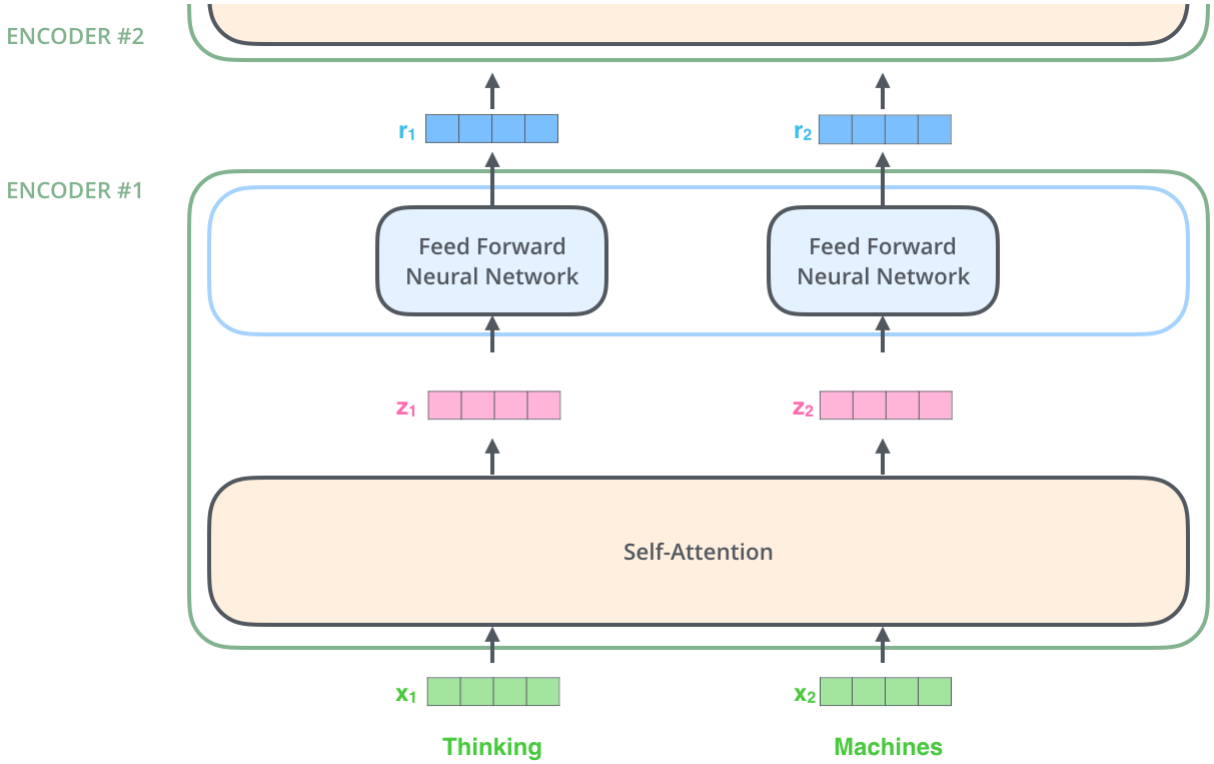
✓ Word in each position flows through its own path in the encoder

- There are dependencies between these paths in the self-attention layer
- The feed-forward layer does not have those dependencies (parallelization becomes possible)

Transformer: Self-Attention

Alamar (Transformer)

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

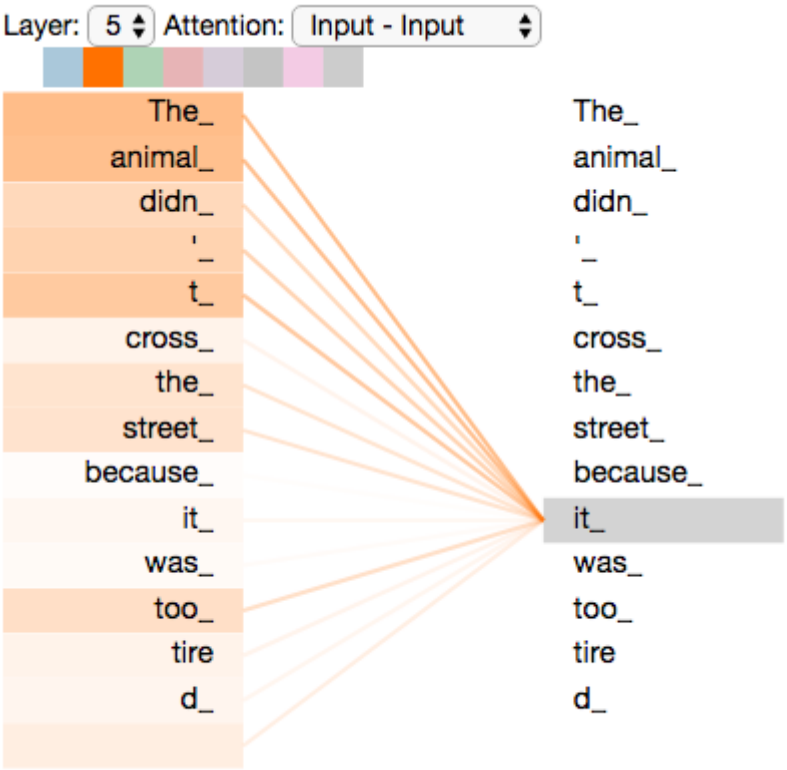


Transformer: Self-Attention

- Self-Attention at a High Level
  - ✓ Input sentence to translate:
    - `The animal did't cross the street because it was too tired`
  - ✓ What does "`it`" refer to? `street` or `animal`?
    - Simple question to a human but not as simple to an algorithm
  - ✓ 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
  - ✓ Self-attention is the method the Transformer uses to bake the "understanding" of other relevant words into the one we're currently processing

Transformer: Self-Attention

- Self-Attention example

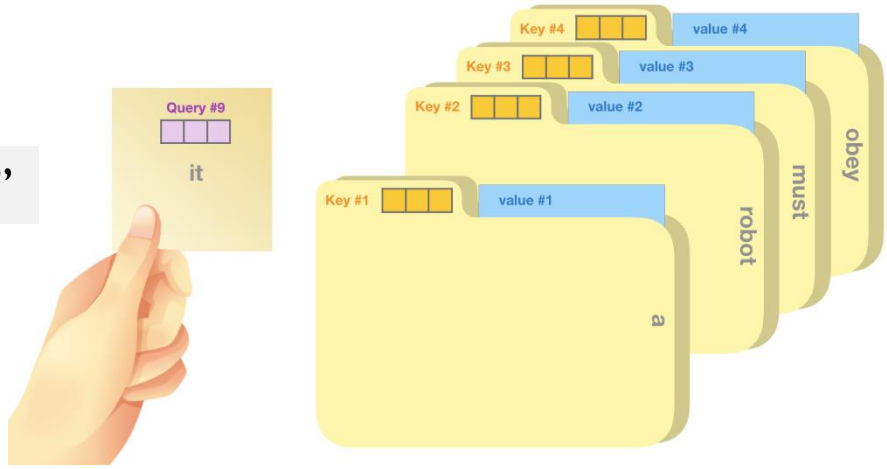


https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb#scrollTo=OJKU36QAfqOC

Transformer: Self-Attention

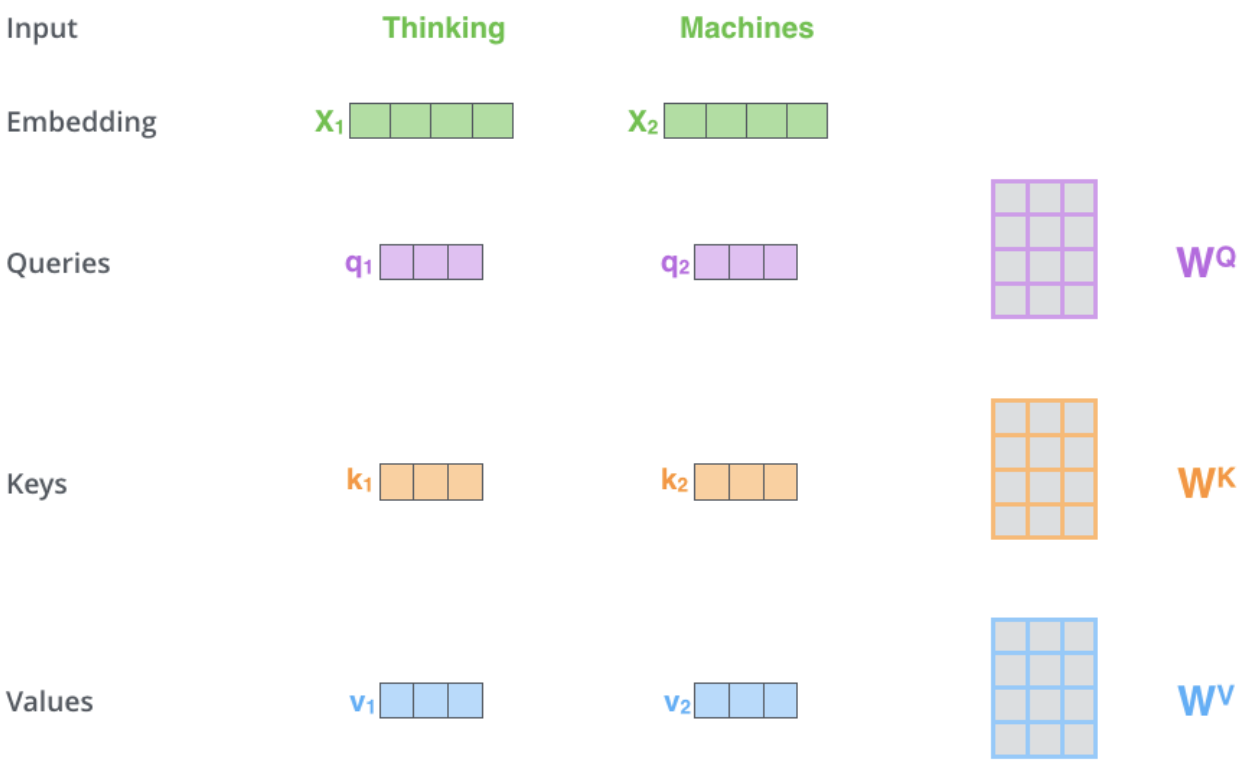
- Self-Attention in Detail
  - ✓ Step 1: Create three vectors from each of the encoder's input vectors
    - Query: The query is a representation of the current word used to score against all the other words (using their keys). We only care about the query of the token we're currently processing.
    - Key: Key vectors are like labels for all the words in the segment. They're what we match against in our search for relevant words.
    - Value: Value vectors are actual word representations, once we've scored how relevant each word is, these are the values we add up to represent the current word.

"A robot must obey the orders given it"



Transformer: Self-Attention

- Self-Attention in Detail
  - ✓ Step 1: Create three vectors from each of the encoder's input vectors
    - These vectors are created by multiplying the embedding by three matrices that we trained during the training process

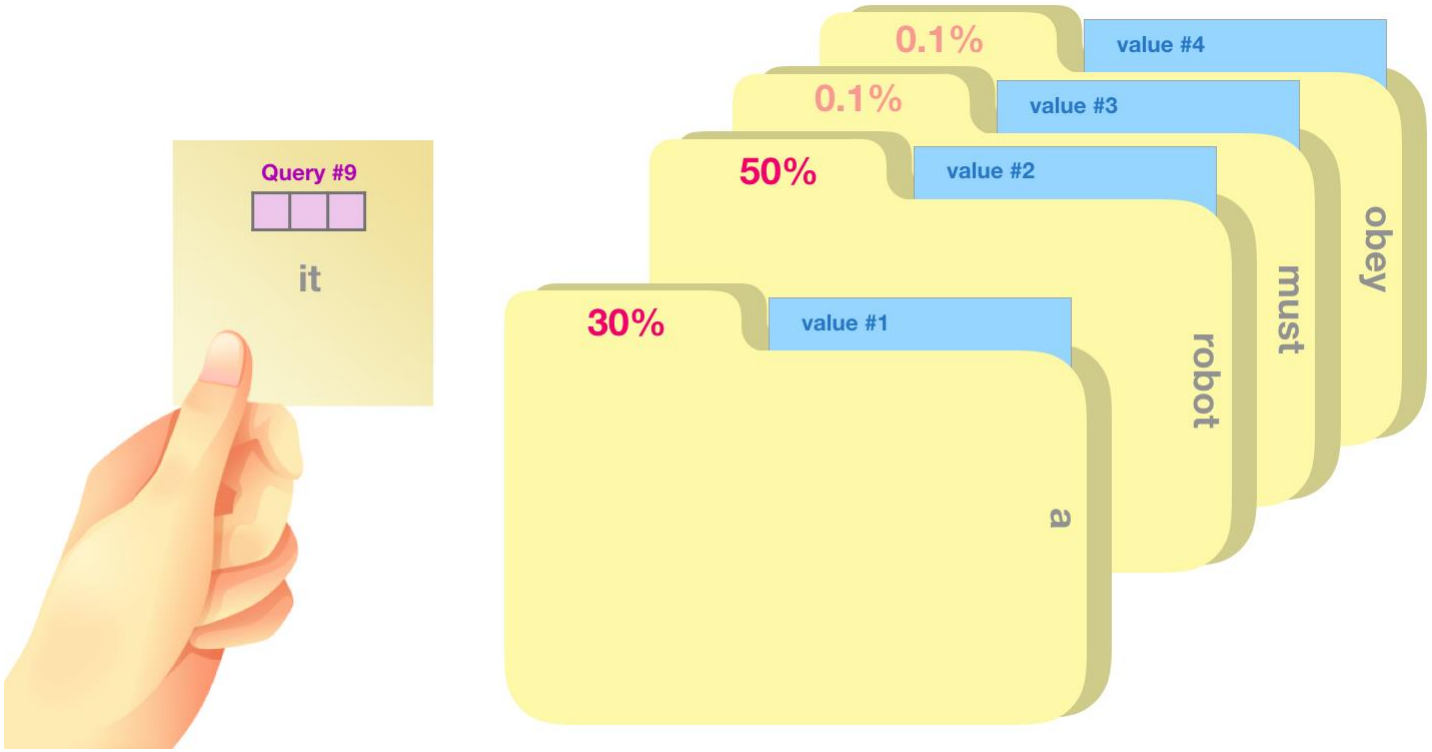


Transformer: Self-Attention

- Self-Attention in Detail
  - ✓ Step 1: Create three vectors from each of the encoder's input vectors
    - Note) These new vectors are smaller in dimension that the embedding vector
      - Q, K, and V are 64-dim. while embedding and encoder input/output vectors are 512-dim.
      - They do not have to be smaller but it is an architecture choice to make the computation of multi-headed attention (mostly) constant

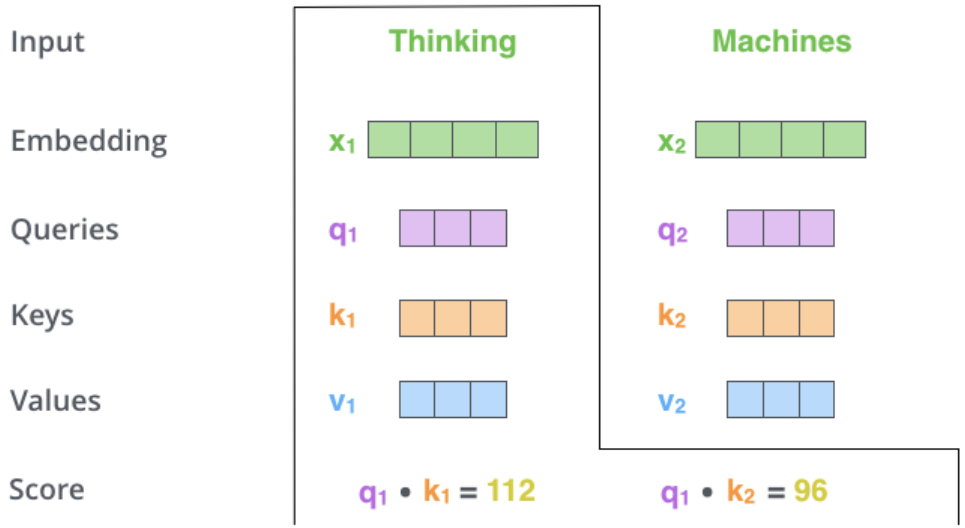
Transformer: Self-Attention

- Self-Attention in Detail
  - ✓ Step 2: Calculate a score, i.e., how much focus to place on other parts of the input sentence as we encode a word at a certain position
    - Multiplying the query vector by each key vector produces a score for each folder (technically: dot product followed by softmax)



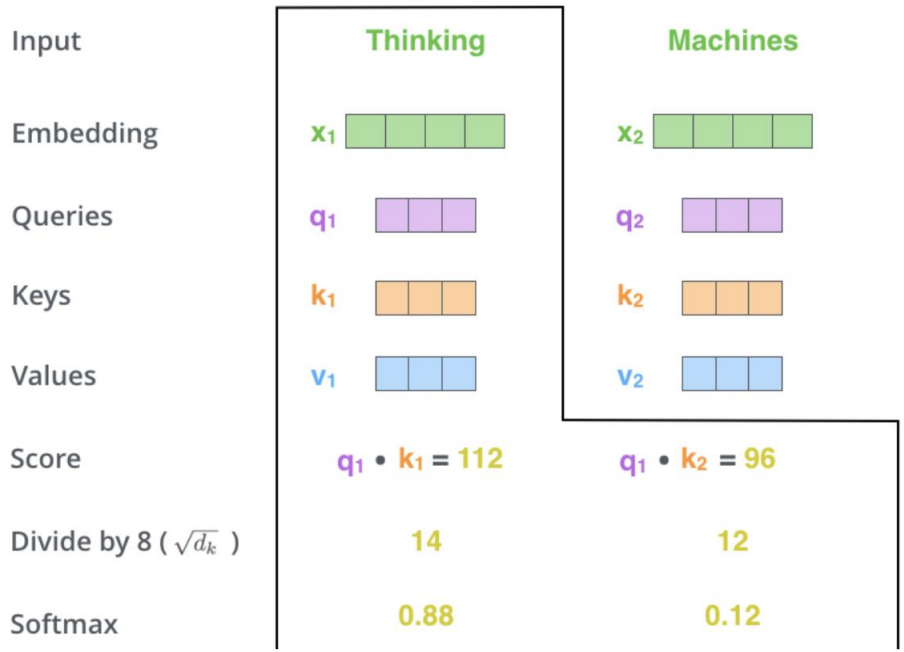
Transformer: Self-Attention

- Self-Attention in Detail
  - ✓ Step 2: Calculate a score, i.e., 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 with the key vector of the respective word we are scoring



Transformer: Self-Attention

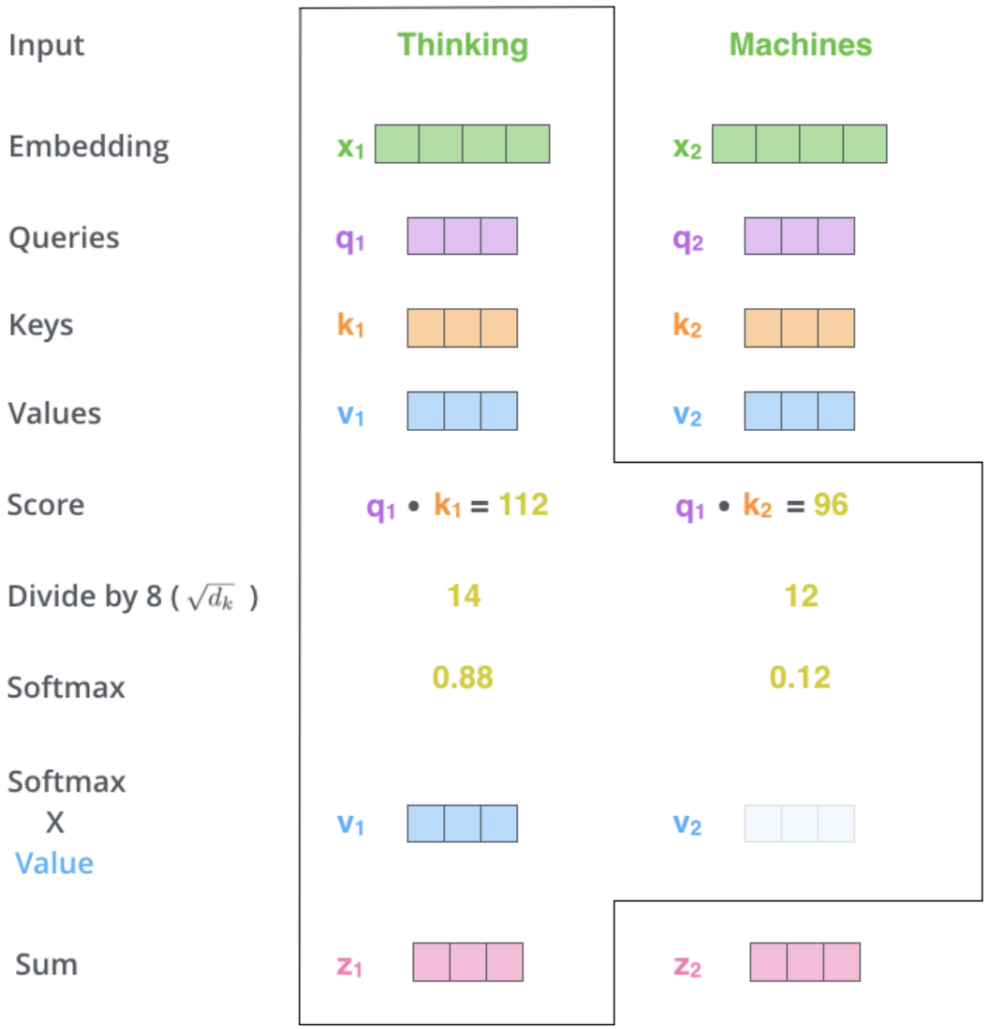
- Self-Attention in Detail
  - ✓ Step 3: Divide the score by $\sqrt{d_k}$ (= 8 in the original paper since $d_k = 64$)
    - This lead to having more stable gradients
  - ✓ Step 4: Pass the result through a softmax operation
    - The softmax score determines how much each word will be expressed at this position



Transformer: Self-Attention

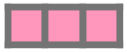
Alamar (Transformer)

- Self-Attention in Detail
  - ✓ Step 5: Multiply each value vector by the softmax score
    - to keep intact the values of the words we want to focus on
    - drown-out irrelevant words
  - ✓ Step 6: Sum up the weighted value vector which produces the output of the self-attention layer at this position



Transformer: Self-Attention

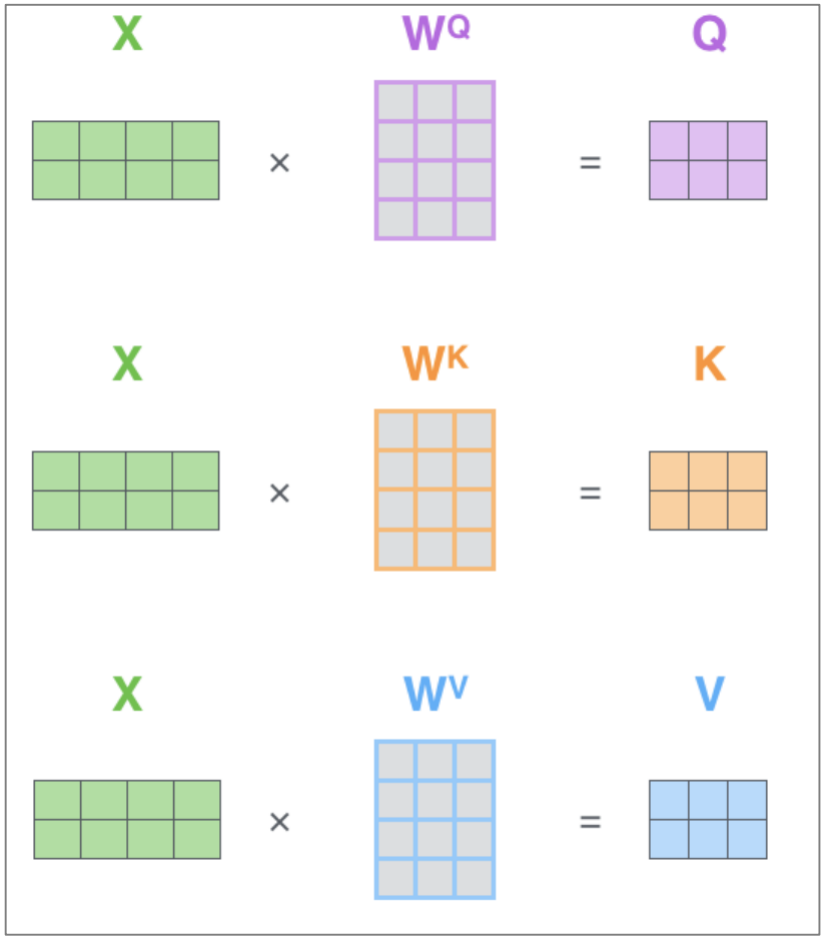
- Self-Attention in Detail
  - ✓ Step 5: Multiply each value vector by the softmax score
  - ✓ Step 6: Sum up the weighted value vector which produces the output of the self-attention layer at this position

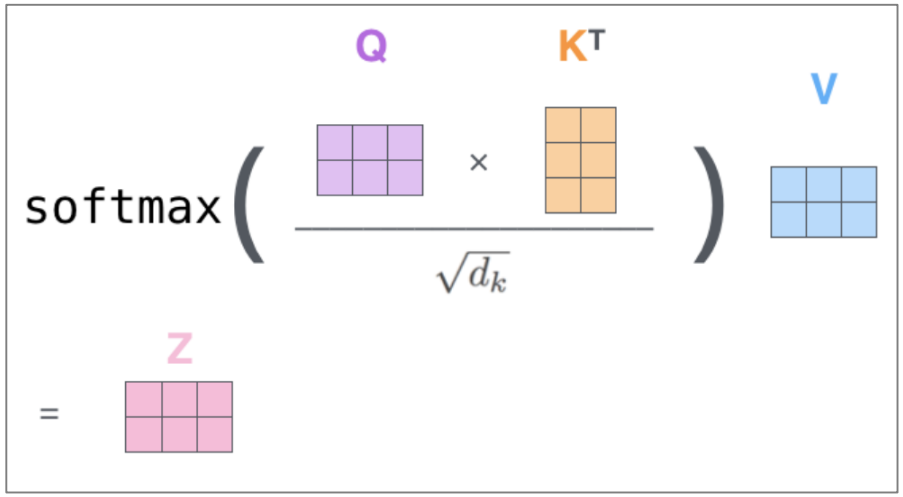
Word	Value vector	Score	Value X Score
<s>		0.001	
a		0.3	
robot		0.5	
must		0.002	
obey		0.001	
the		0.0003	
orders		0.005	
given		0.002	
it		0.19	
		Sum:	

Transformer: Self-Attention

Alamar (Transformer)

- Matrix calculation of self-attention



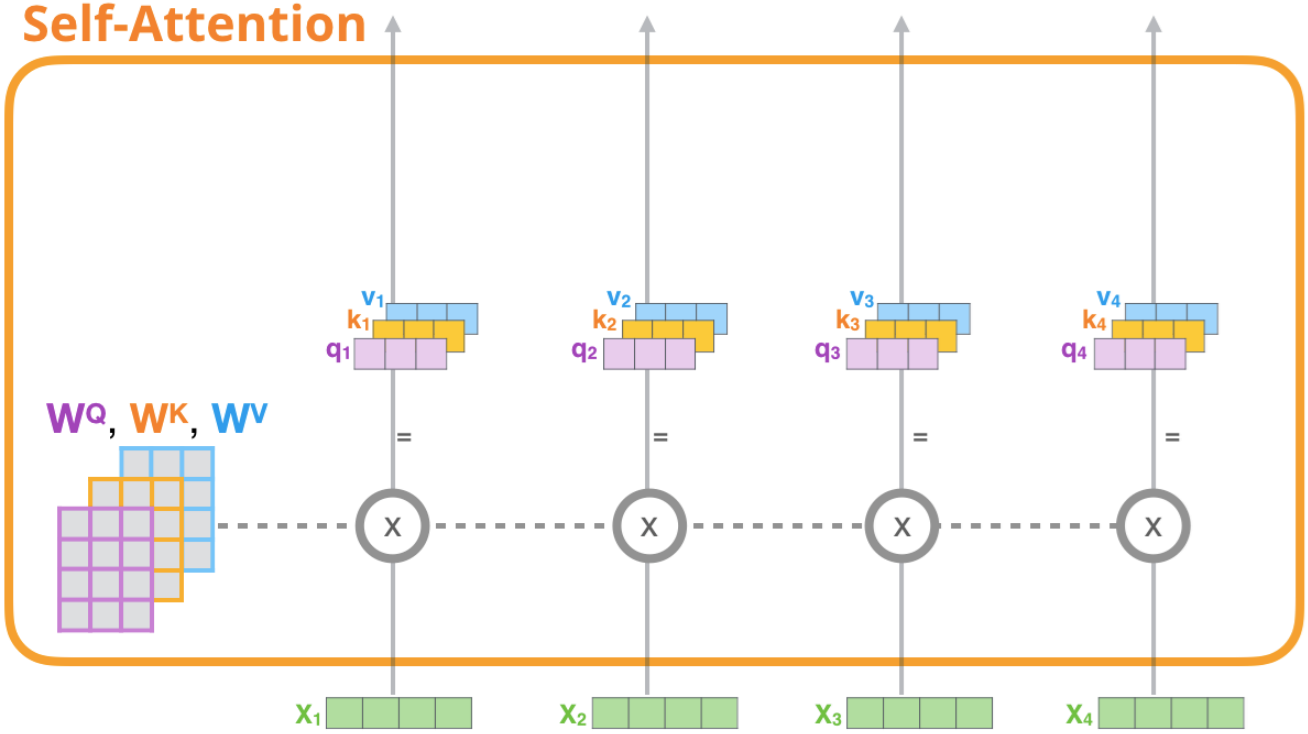


Transformer: Self-Attention

Alamar (GPT-2)

- Another illustration of Self-Attention
  - ✓ Create Query, Key, and Value Vectors

1) For each input token, create a query vector, a key vector, and a value vector by multiplying by weight Matrices W^Q, W^K, W^V

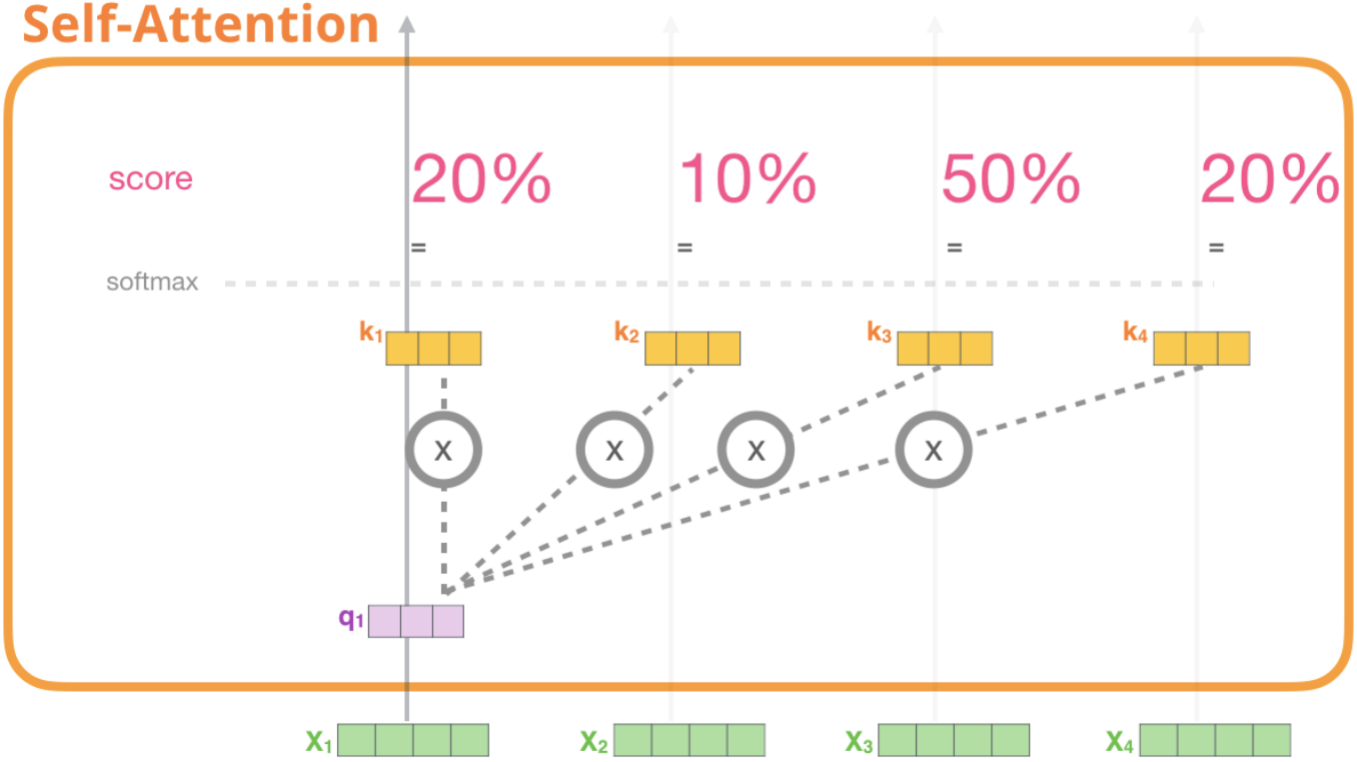


Transformer: Self-Attention

Alamar (GPT-2)

- Another illustration of Self-Attention
  - ✓ Score

2) Multiply (dot product) the current query vector, by all the key vectors, to get a score of how well they match

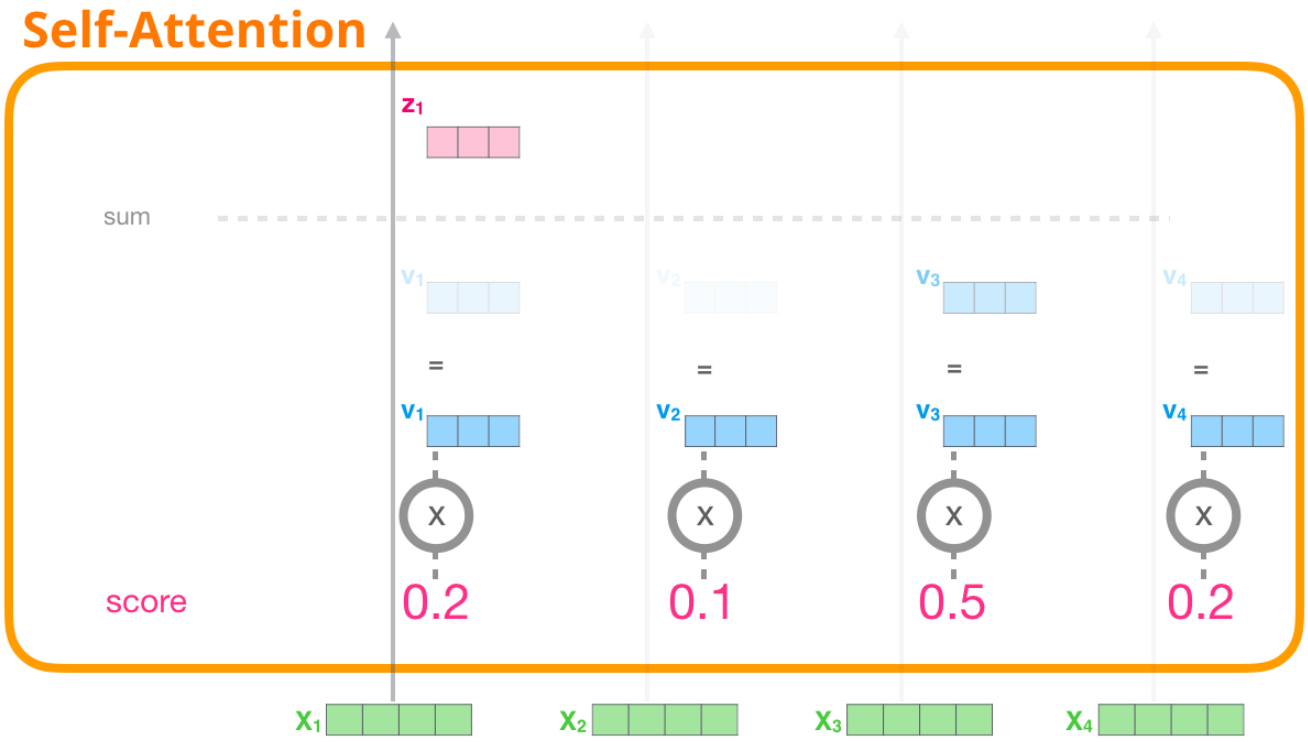


Transformer: Self-Attention

Alamar (GPT-2)

- Another illustration of Self-Attention
  - ✓ Sum

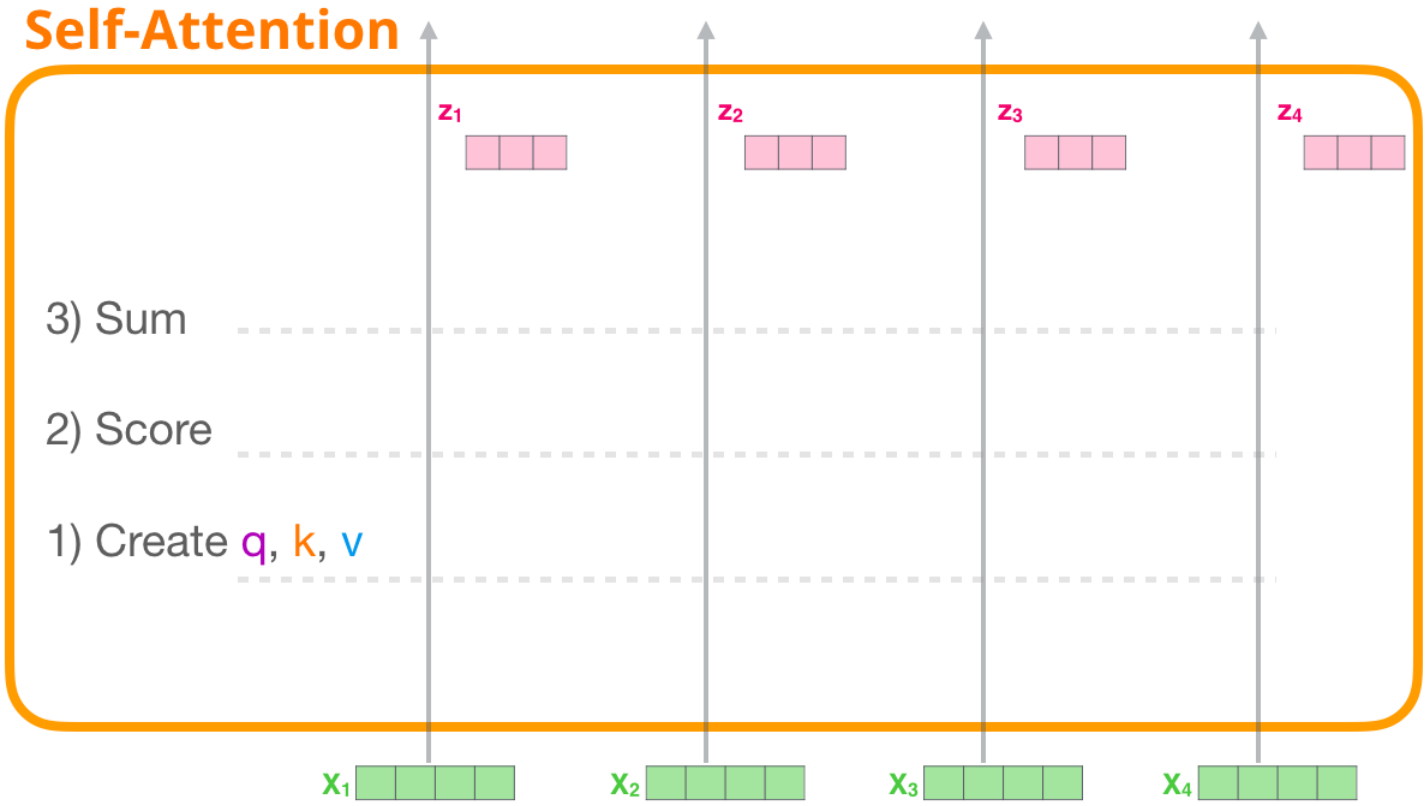
3) Multiply the value vectors by the scores, then sum up



Transformer: Self-Attention

Alamar (GPT-2)

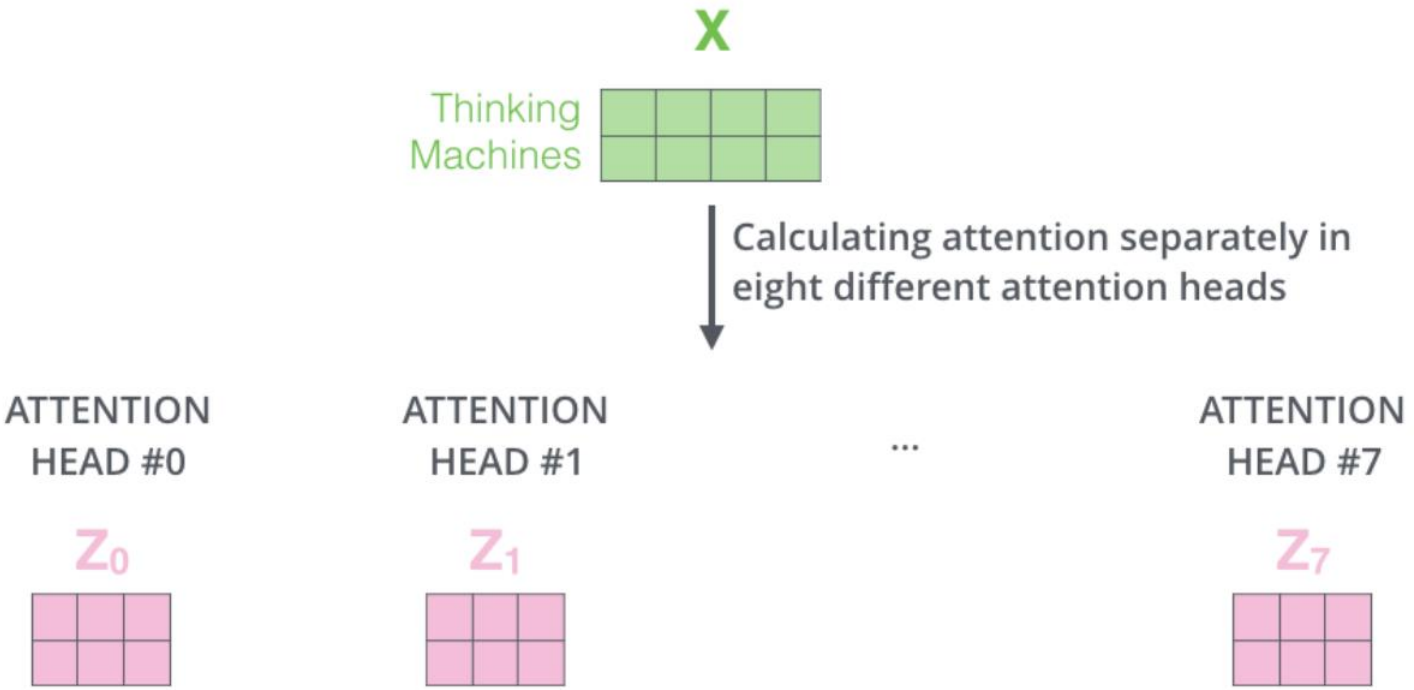
- Another illustration of Self-Attention
  - ✓ Do the same operation for each path to end up with a vector representing each token containing appropriate context of that token



Transformer: Self-Attention

Alamar (Transformer)

- Multi-headed attention
  - ✓ Expand the model's ability to focus on different positions

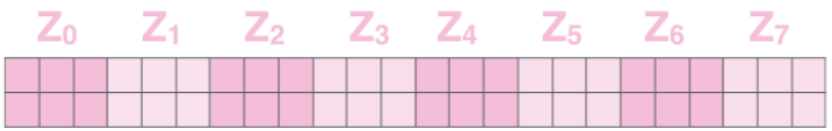


Transformer: Self-Attention

Alamar (Transformer)

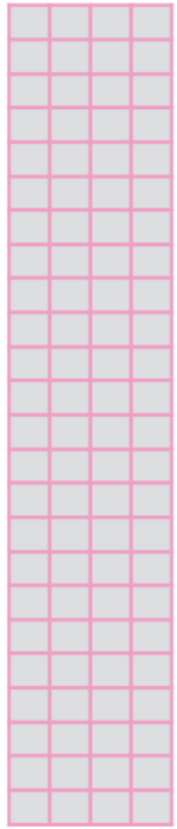
- Multi-headed attention
  - ✓ Attention heads are concatenated and multiplied by an additional weight matrix to be used as an input of feed-forward neural network

1) Concatenate all the attention heads



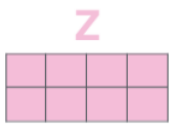
2) Multiply with a weight matrix W^O that was trained jointly with the model

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

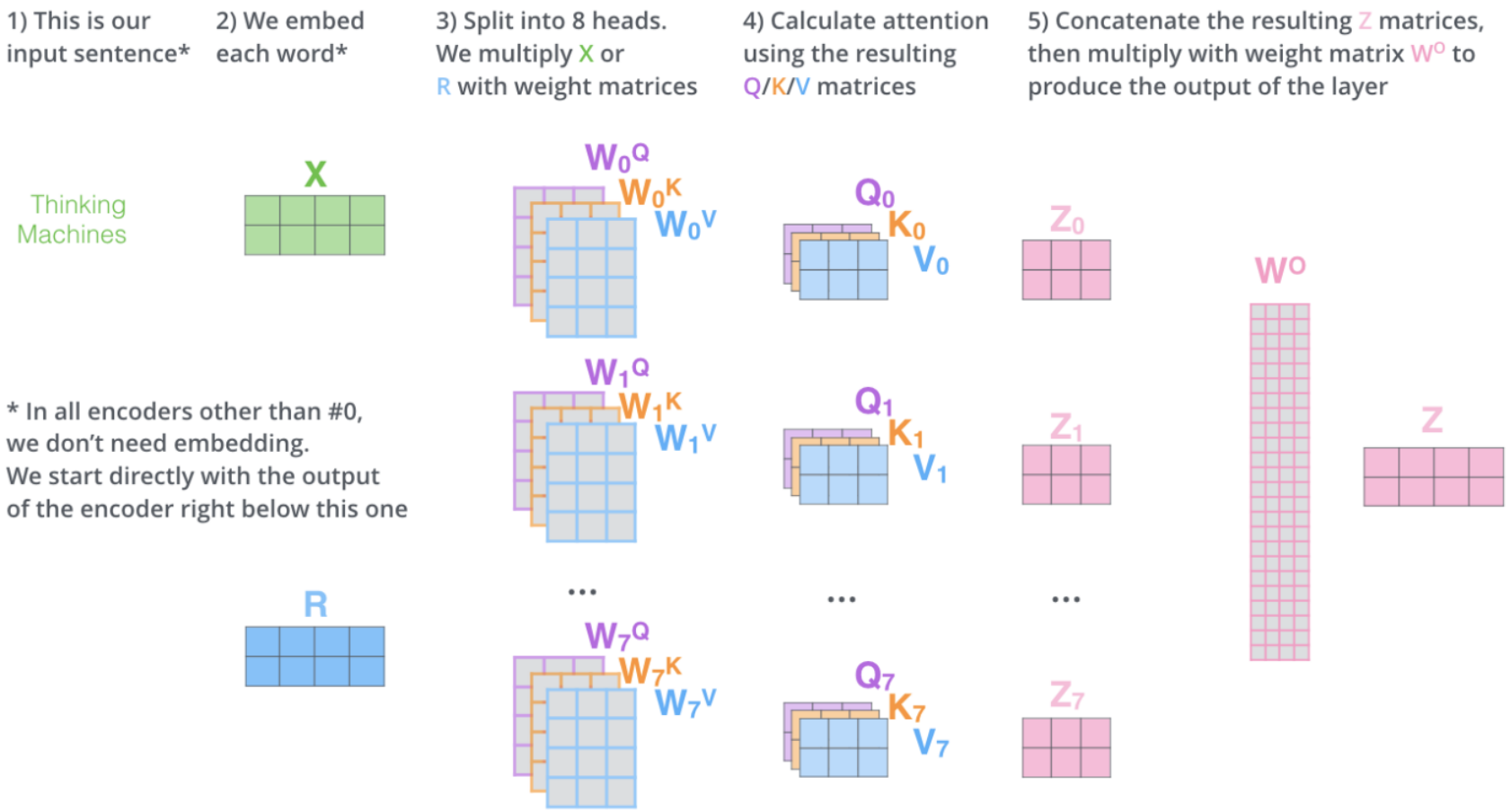
=



Transformer: Self-Attention

Alamar (Transformer)

- Multi-headed attention

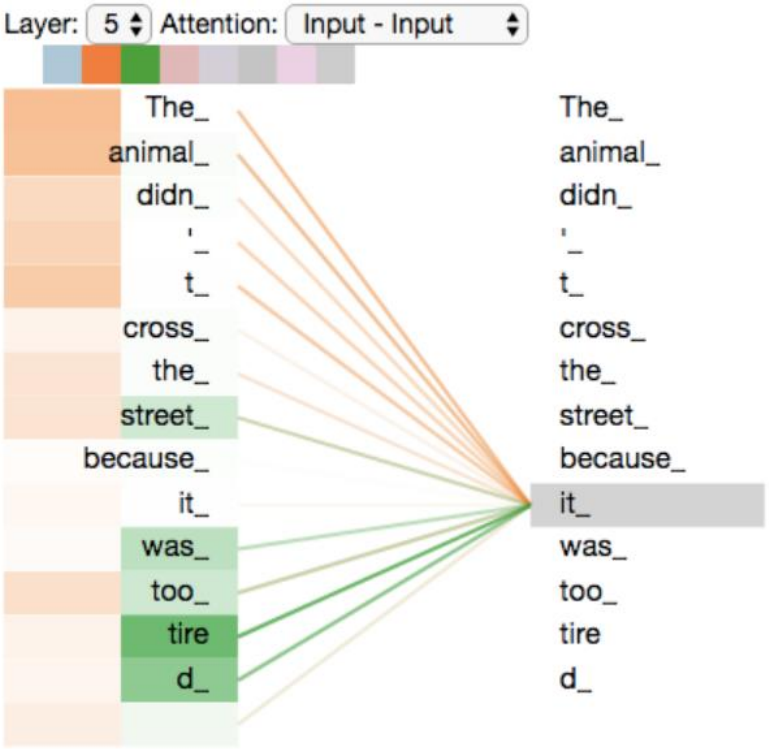


Transformer: Self-Attention

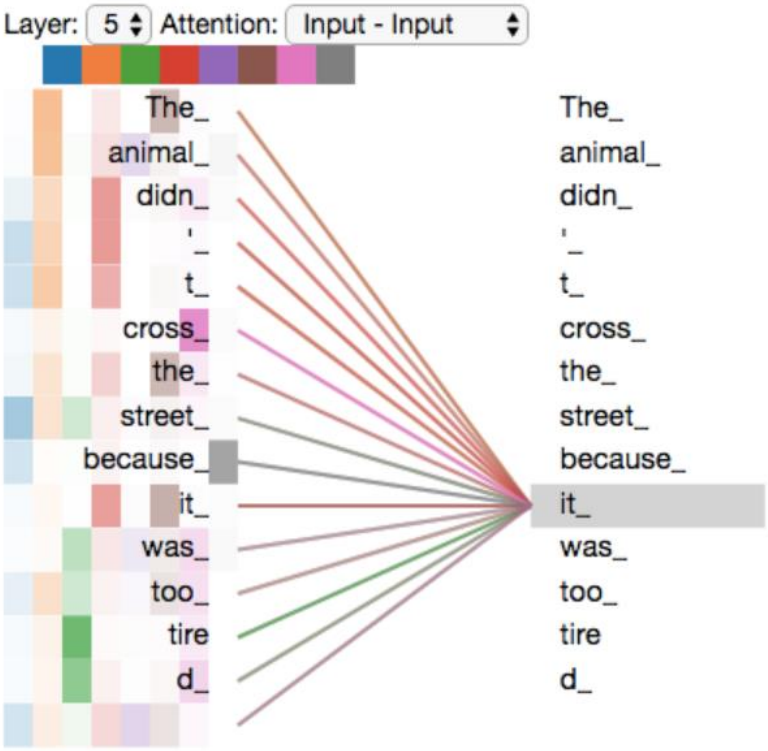
Alamar (Transformer)

- Multi-headed attention

Attention with two heads



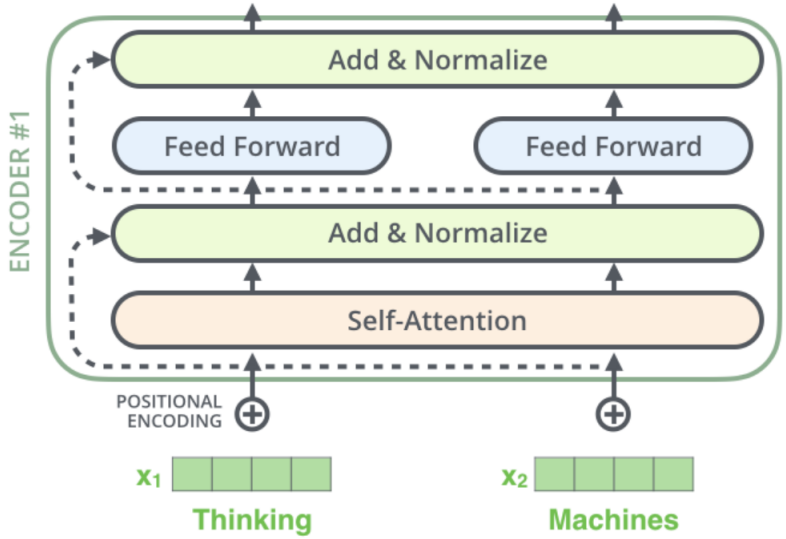
Attention with eight heads

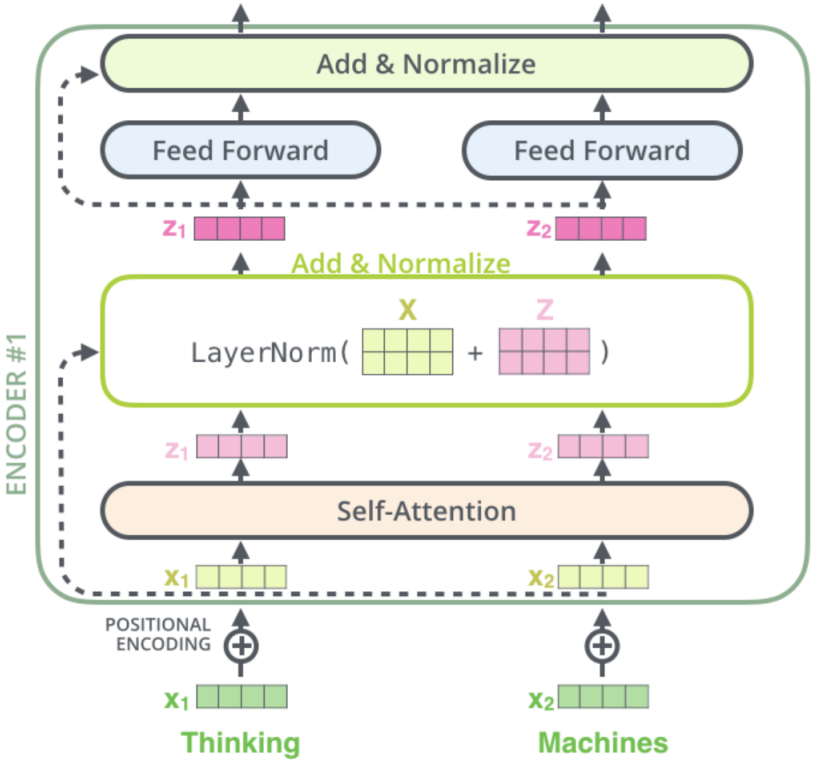


Transformer: Self-Attention

Alamar (Transformer)

- Residual
  - ✓ Each sub-layer (self-attention, FFNN) in each encoder has a residual connection around it followed by a layer-normalization step

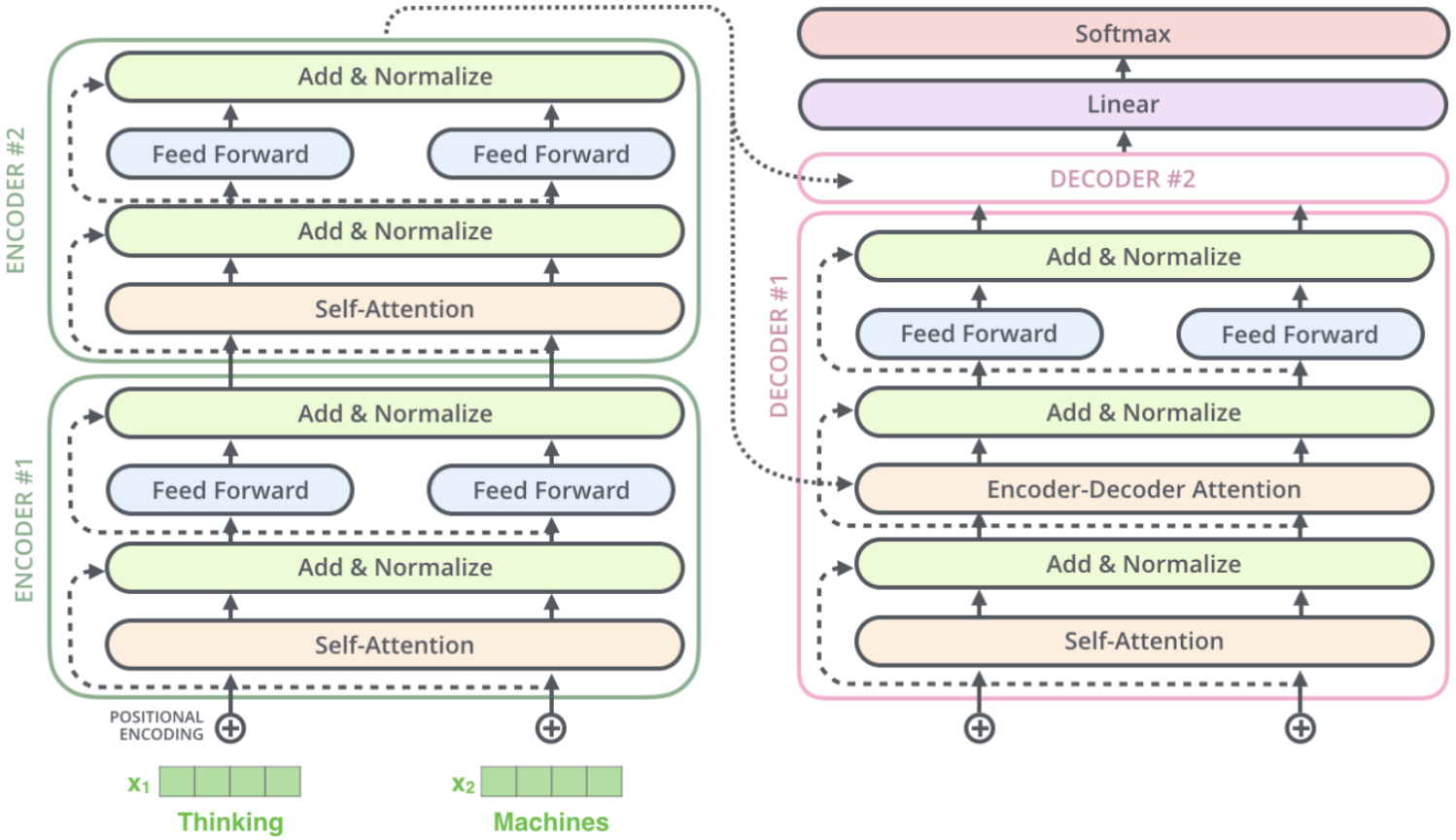




Transformer: Self-Attention

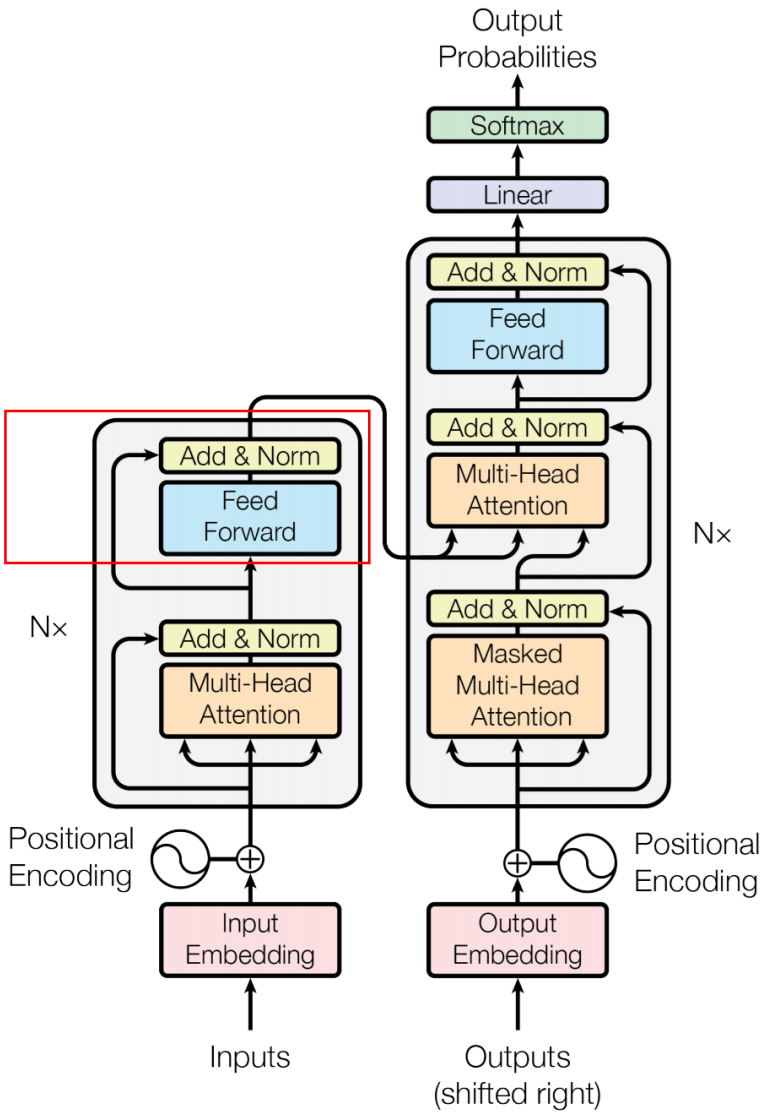
Alamar (Transformer)

- Residual
  - ✓ This goes for the sub-layers of the decoder as well
    - Ex: 2 stacked encoders and decoders



Transformer: Self-Attention

- Position-wise Feed-Forward Networks



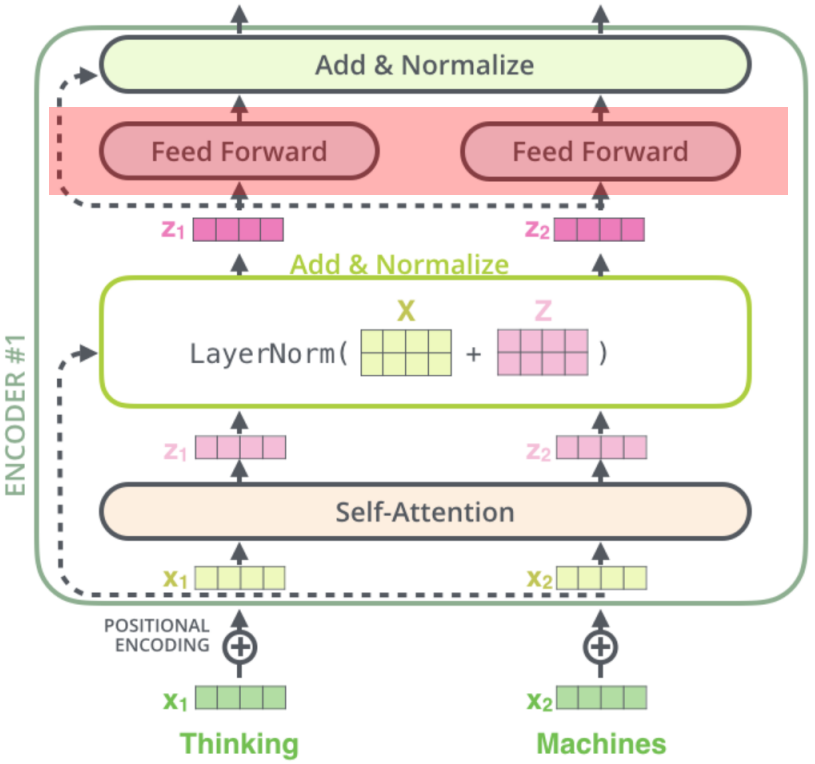
Transformer: Self-Attention

Vaswani et. al (2017), Kim (2019)

- Position-wise Feed-Forward Networks
  - ✓ Fully connected feed-forward network
  - ✓ Applied to each position separately and identically

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

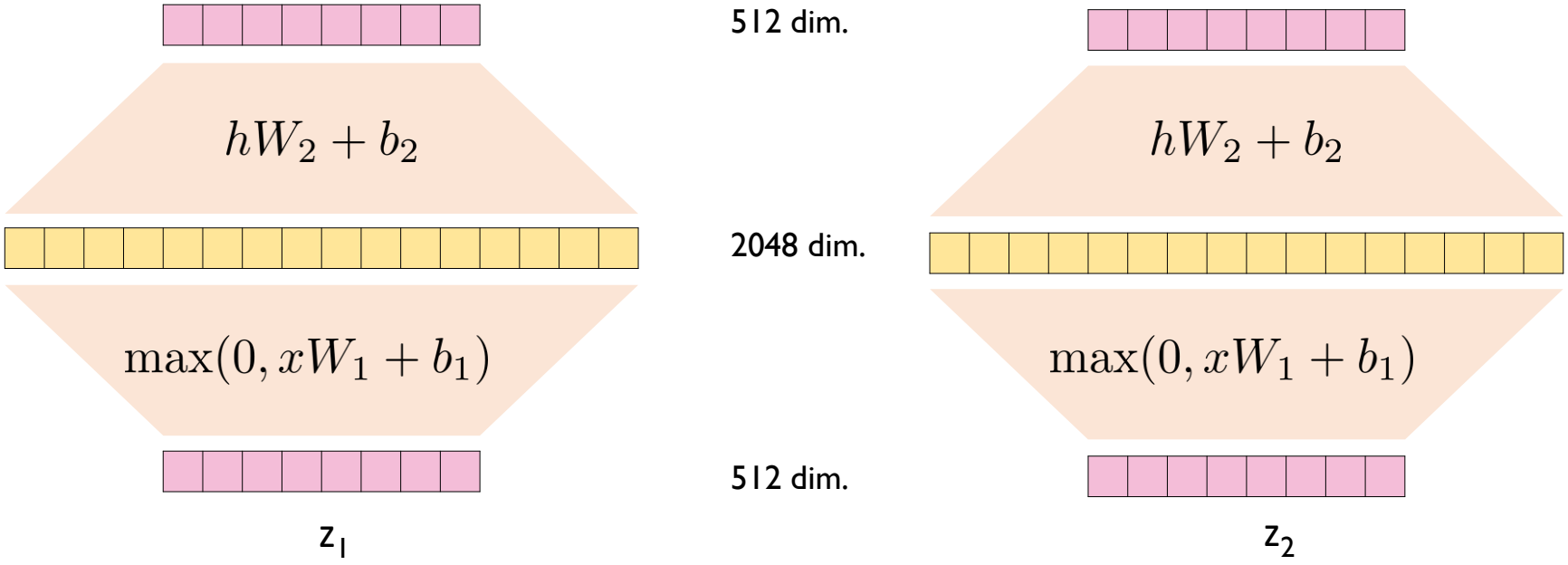
  - ✓ The linear transformations are the same across different positions
  - ✓ They use different parameters from layer to layer



Transformer: Self-Attention

Vaswani et. al (2017), Kim (2019)

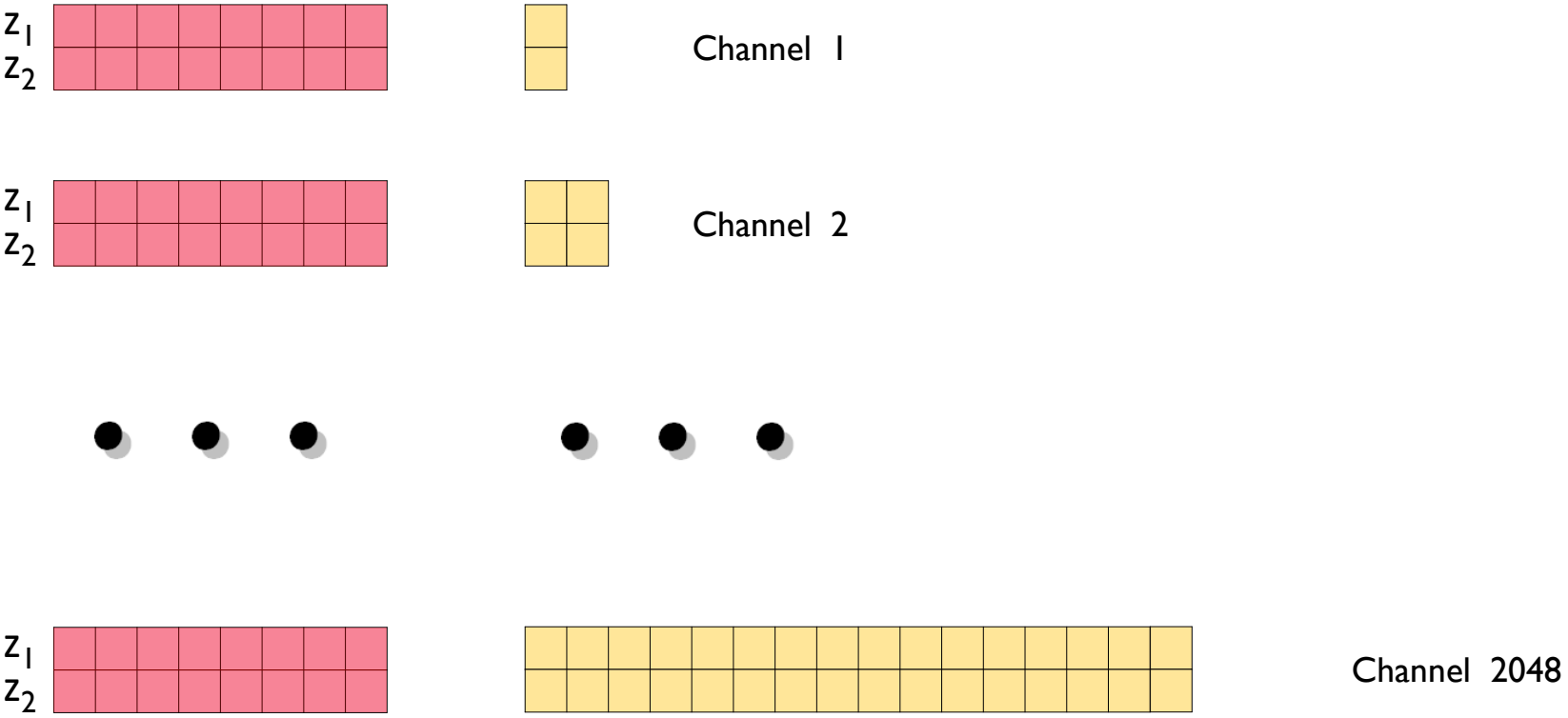
- Position-wise Feed-Forward Networks



Transformer: Self-Attention

Vaswani et. al (2017), Kim (2019)

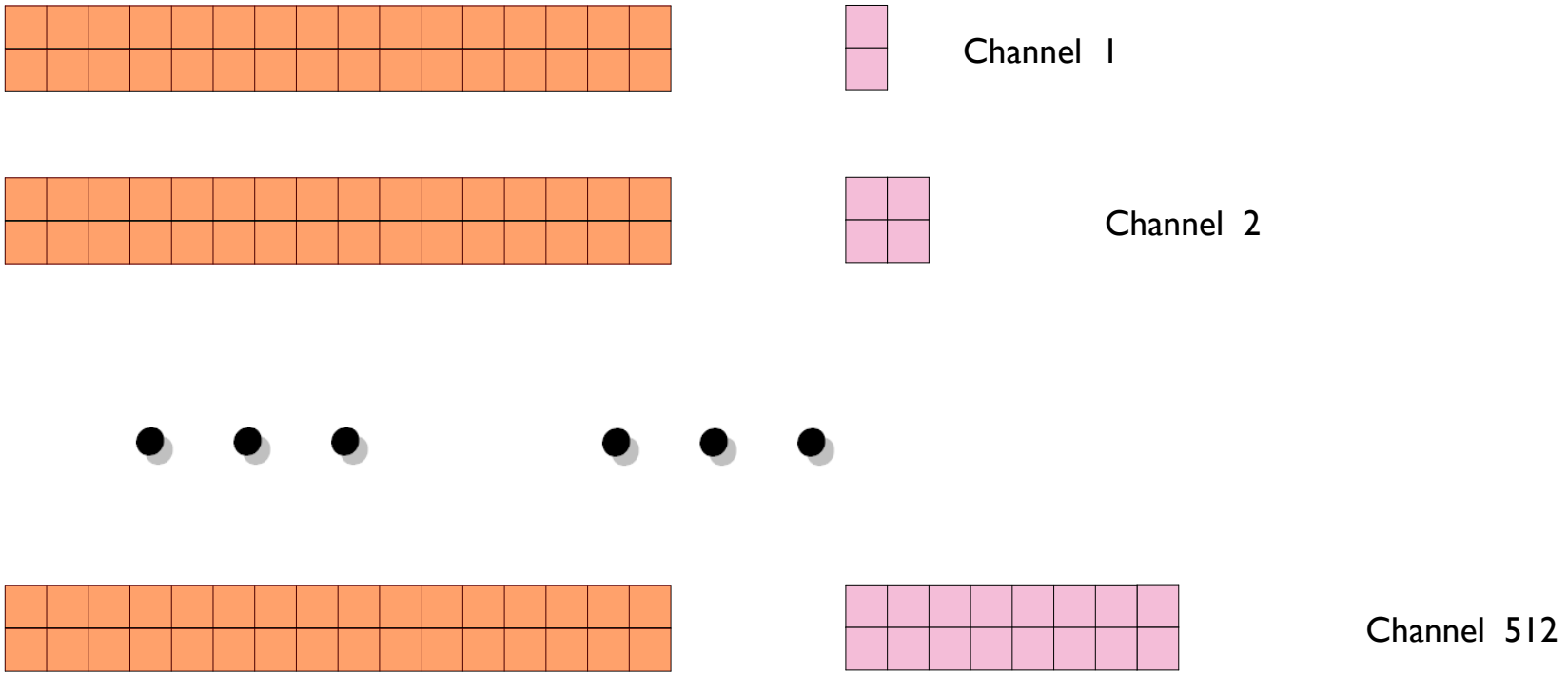
- Position-wise Feed-Forward Networks
  - ✓ Another way of describing this is as two convolutions with kernel size 1
    - Convolution layer 1



Transformer: Self-Attention

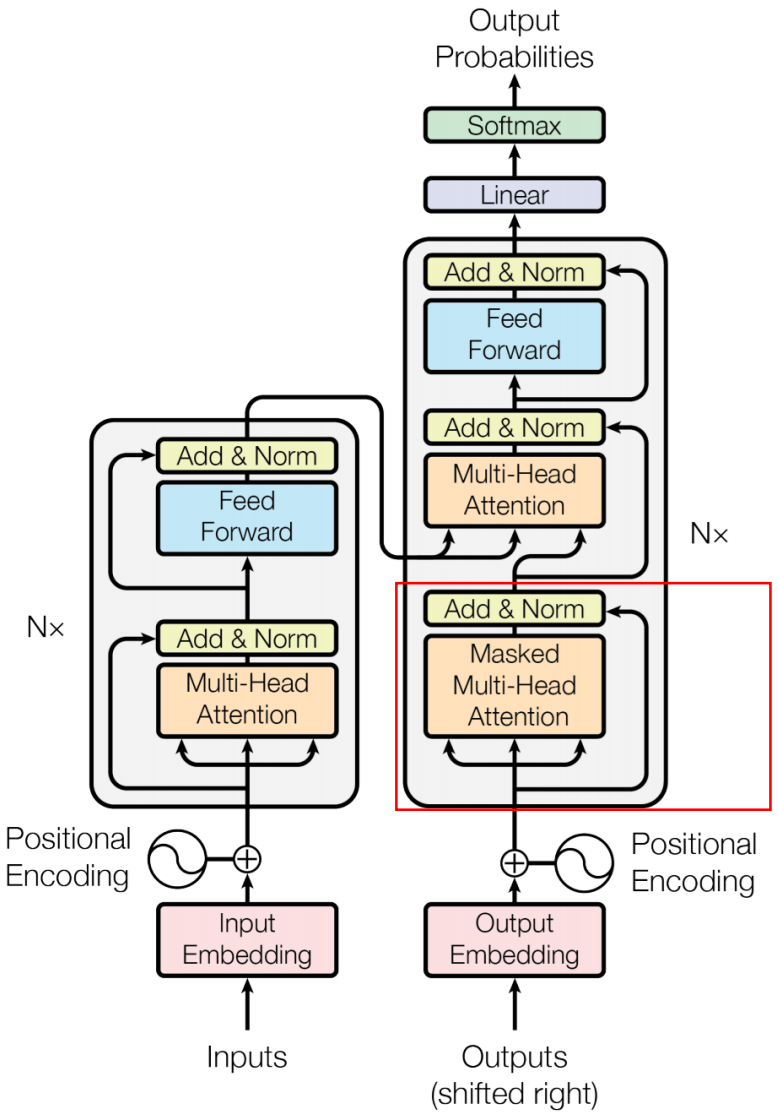
Vaswani et. al (2017), Kim (2019)

- Position-wise Feed-Forward Networks
  - ✓ Another way of describing this is as two convolutions with kernel size 1
    - Convolution layer 2



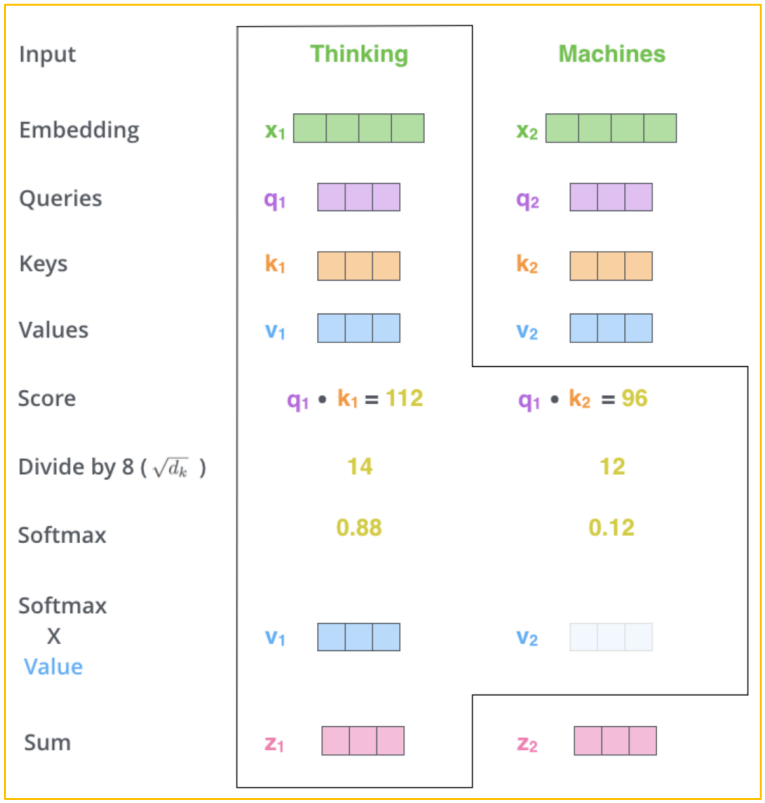
Transformer: Self-Attention

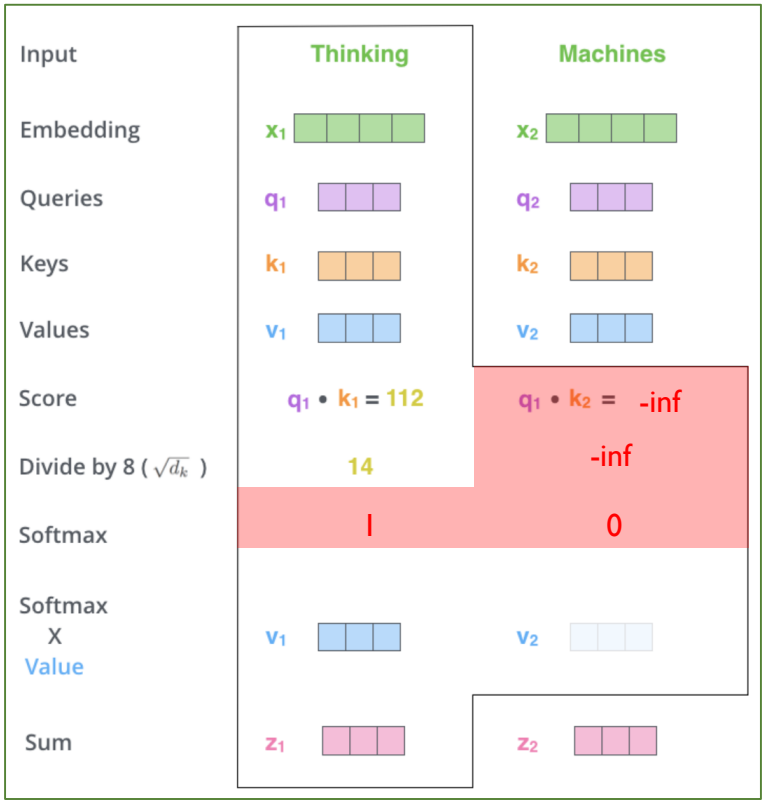
- Masked Multi-Head Attention



Transformer: Self-Attention

- Masked Multi-head Attention
  - ✓ Self attention layers in the decoder is only allowed to attend to earlier positions in the output sequence, which is done by masking future positions (setting them to –inf) before the softmax step in the self attention calculation.

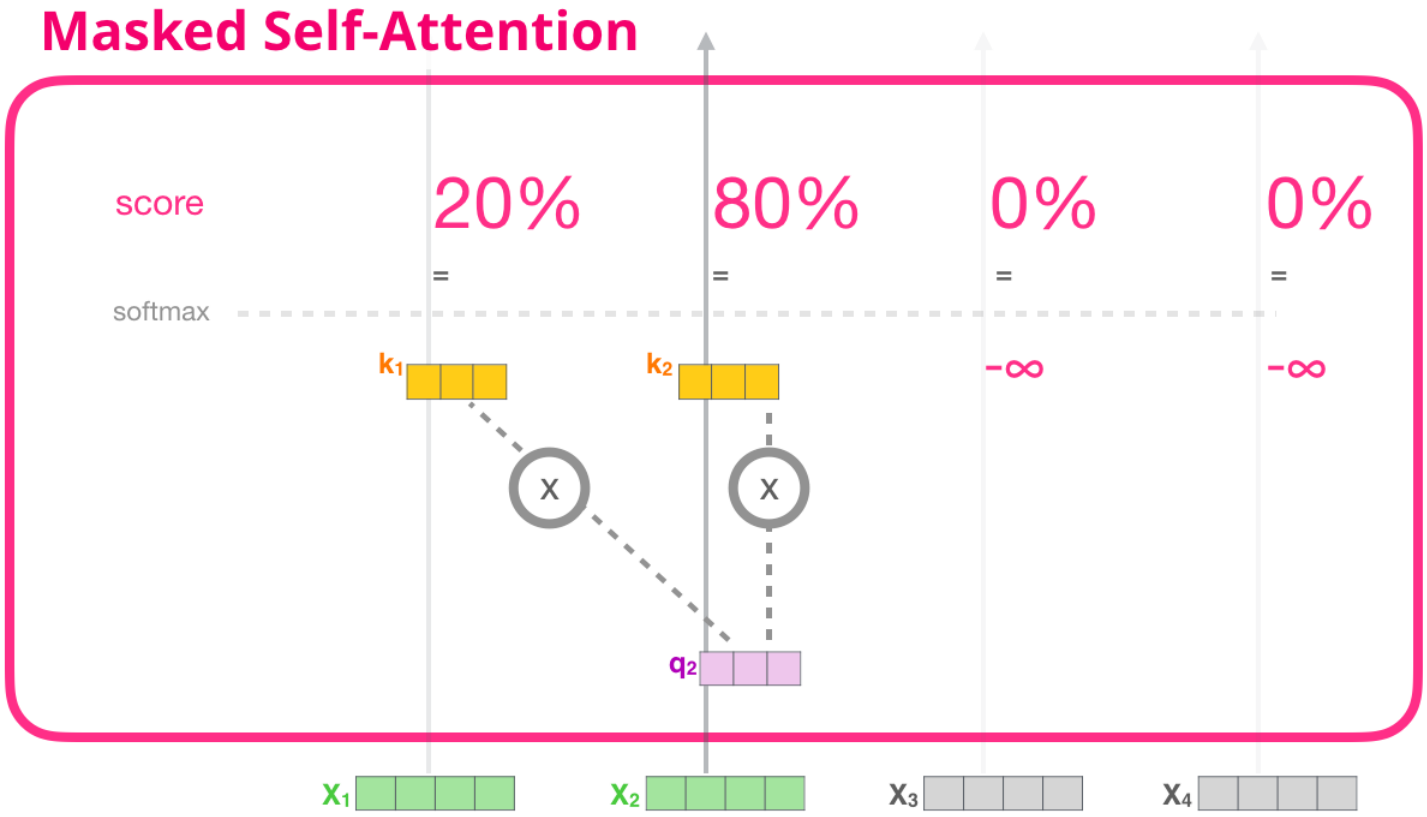




Transformer: Self-Attention

Alamar (GPT-2)

- Masked Multi-head Attention



Transformer: Self-Attention

Alamar (GPT-2)

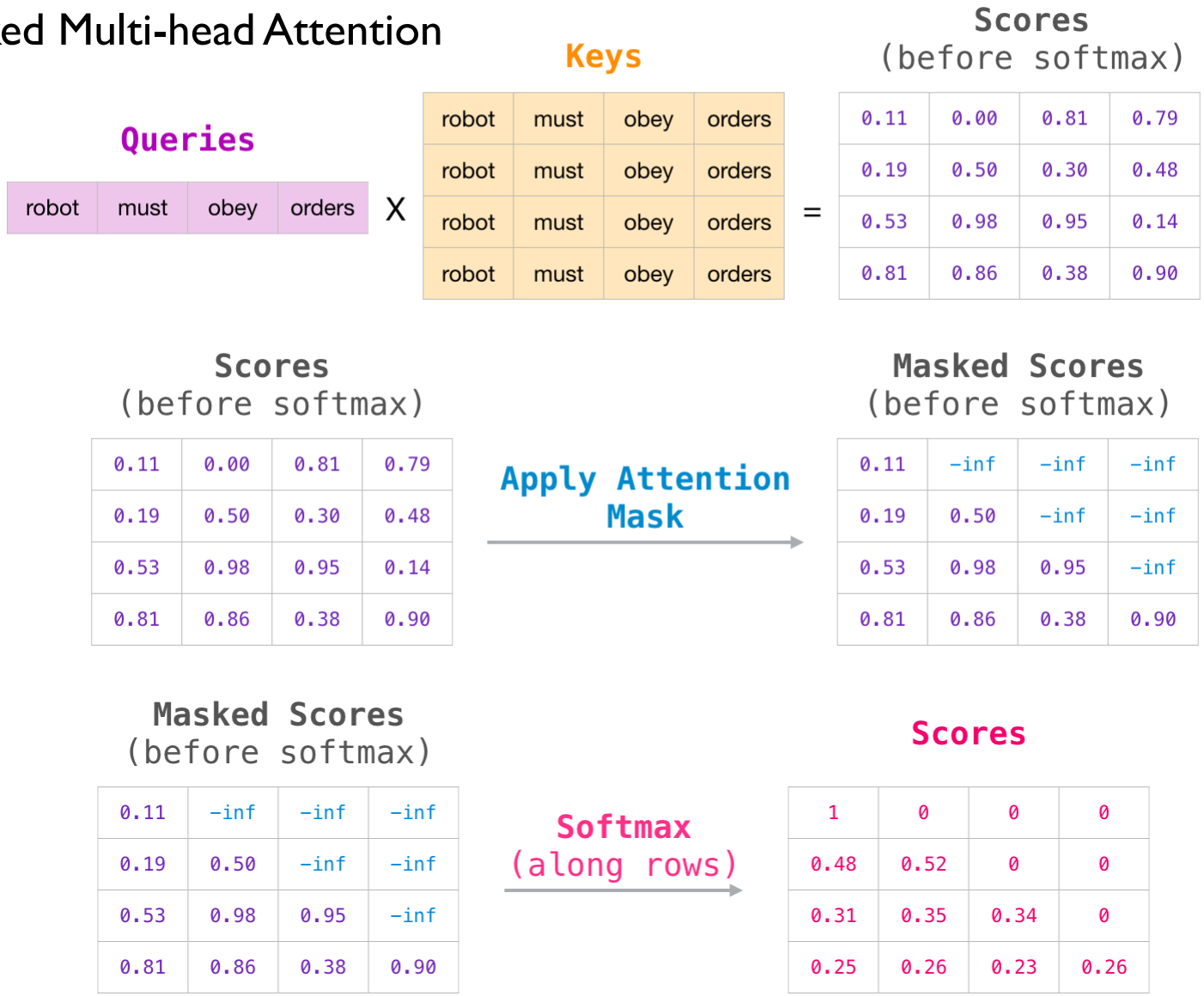
- Masked Multi-head Attention
  - ✓ Do not need to be done sequentially, but can be done at one batch

	Features				Labels
	position: 1	2	3	4	
Example: 1	robot	must	obey	orders	must
2	robot	must	obey	orders	obey
3	robot	must	obey	orders	orders
4	robot	must	obey	orders	<eos>

Transformer: Self-Attention

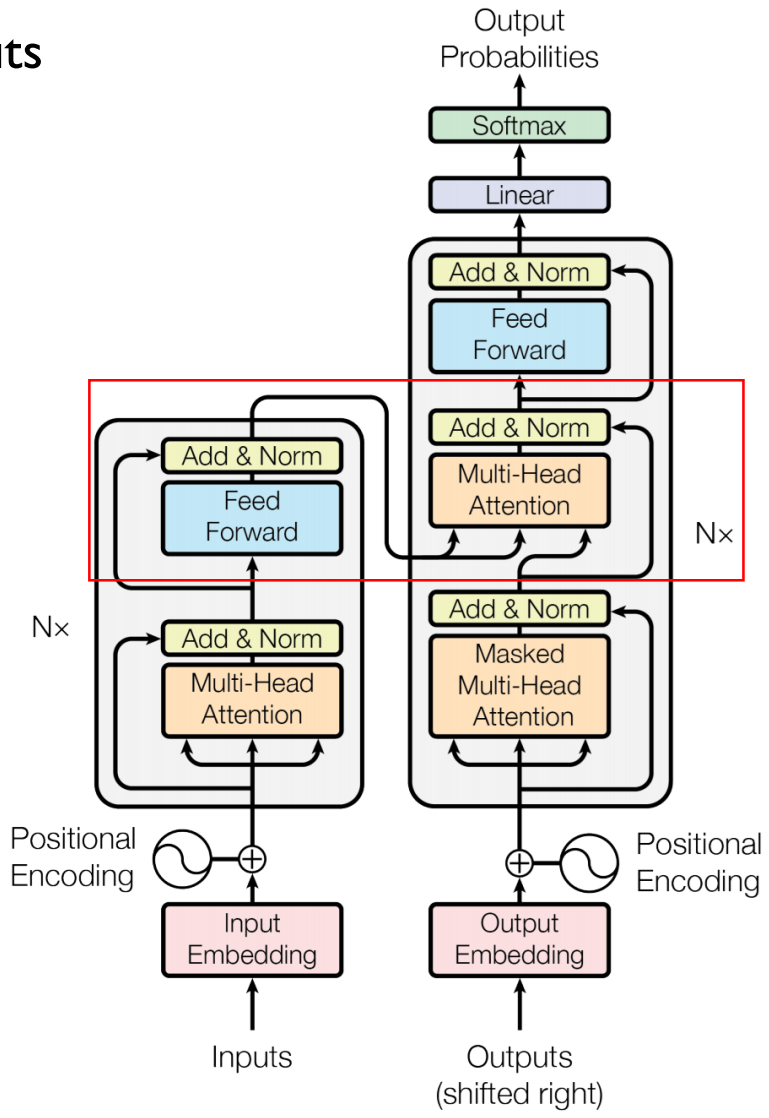
Alamar (GPT-2)

- Masked Multi-head Attention



Transformer: Self-Attention

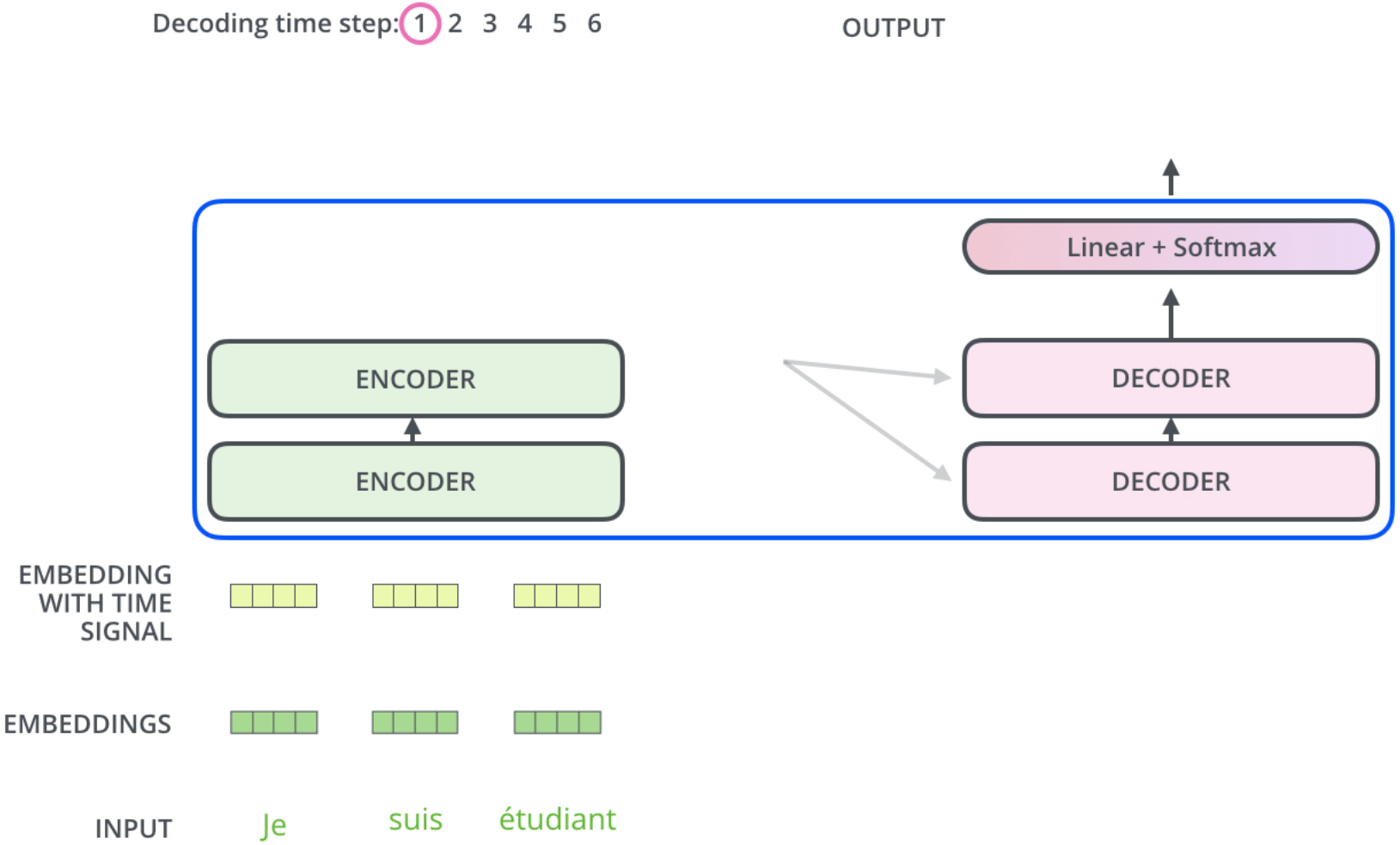
- Multi-Head Attention with Encoder Outputs



Transformer: Self-Attention

Alamar (Transformer)

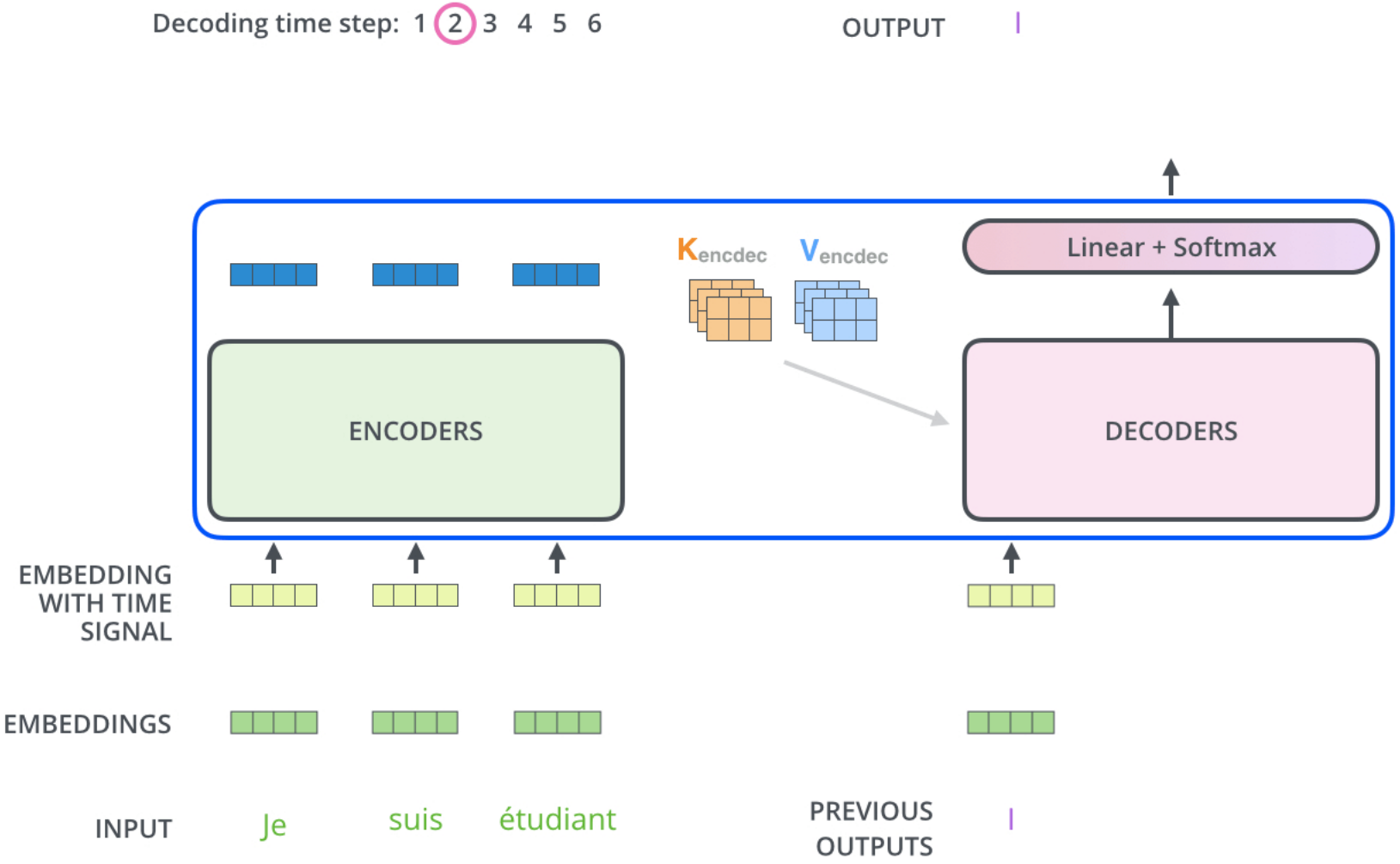
- Decoder Side



Transformer: Self-Attention

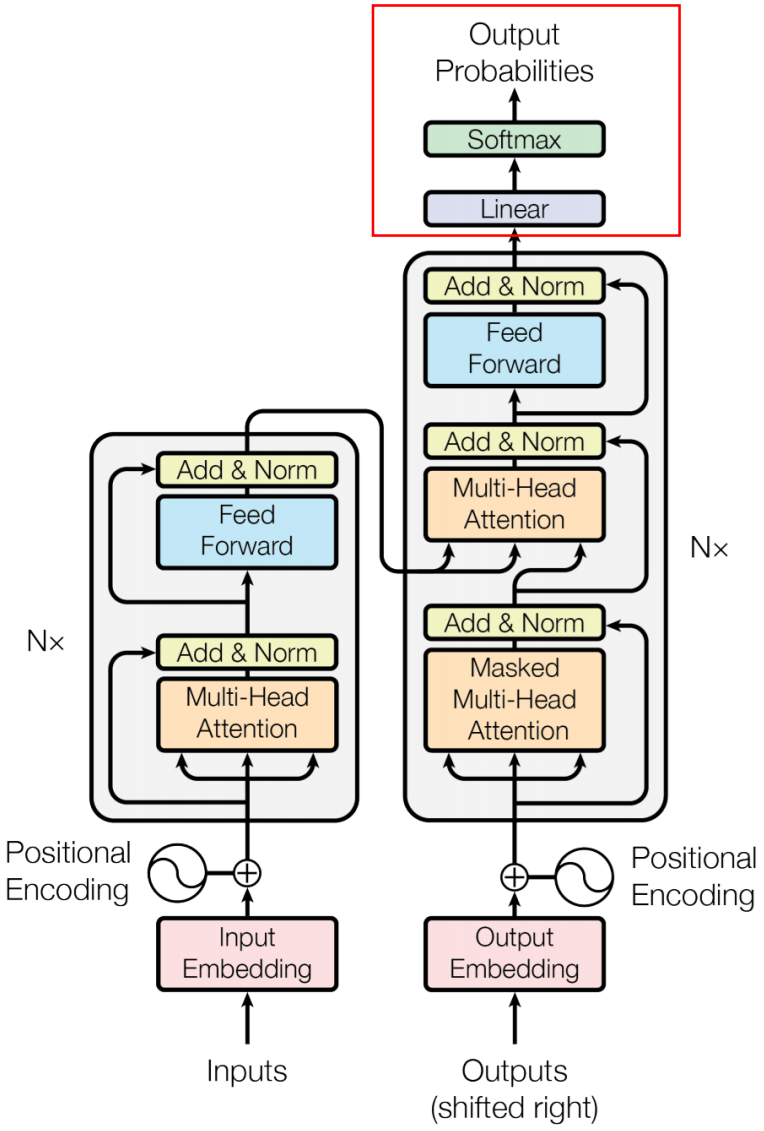
Alamar (Transformer)

- Decoder Side



Transformer: Self-Attention

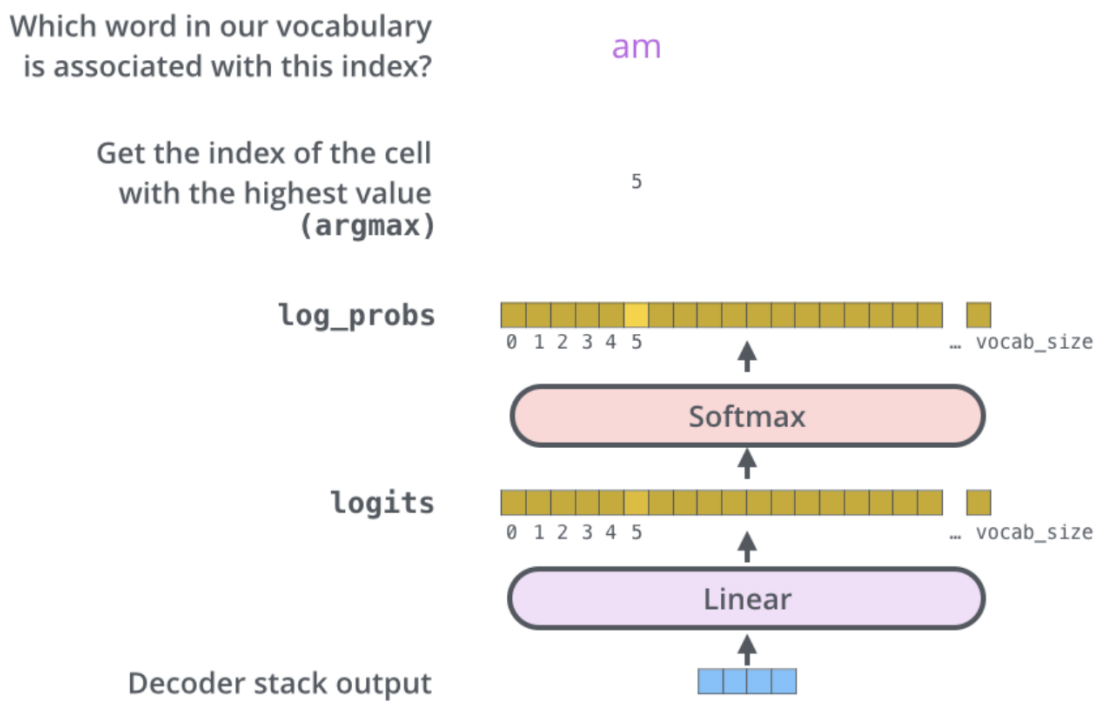
- The Final Linear and Softmax Layer



Transformer: Self-Attention

Alamar (Transformer)

- The Final Linear and Softmax Layer
  - ✓ Linear layer: 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
  - ✓ Softmax layer: turns those scores into probability
    - The cell with the highest probability is chosen, the word associated with it is produced as the output of this time step



Transformer: Self-Attention

Alamar (Transformer)

- Complexity

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Transformer: Self-Attention

- Performance (in terms of BLEU score)

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	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**	**41.8**	$2.3 \cdot 10^{19}$	

Transformer: Self-Attention

- (Note) BLEU (Bilingual Evaluation Understudy) score

BLEU

BLEU(Bilingual Evaluation Understudy)score란 성과지표로 데이터의 X가 순서정보를 가진 단어들(문장)로 이루어져 있고, y 또한 단어들의 시리즈(문장)로 이루어진 경우에 사용되며, 번역을 하는 모델에 주로 사용된다. 3가지 요소를 살펴보자.

- n-gram을 통한 순서쌍들이 얼마나 겹치는지 측정(precision)
- 문장길이에 대한 과적합 보정 (Brevity Penalty)
- 같은 단어가 연속적으로 나올때 과적합 되는 것을 보정(Clipping)

$$BLEU = min(1, \frac{output\ length(\text{예측 문장})}{reference\ length(\text{실제 문장})})(\prod_{i=1}^{4} precision_i)^{\frac{1}{4}}$$

Transformer: Self-Attention

- (Note) BLEU (Bilingual Evaluation Understudy) score

1. n-gram(1~4)을 통한 순서쌍들이 얼마나 겹치는지 측정(precision)

   - `예측된 sentence` : **빛이 쐬는** 노인은 **완벽한** 어두운곳에서 **잠든 사람과 비교할 때** 강박증이 **심해질** 기회가 **훨씬 높았다**
   - `true sentence` : **빛이 쐬는** 사람은 **완벽한** 어둠에서 **잠든 사람과 비교할 때** 우울증이 **심해질** 가능성이 **훨씬 높았다**

- 1-gram precision: $\frac{\text{일치하는} 1-gram\text{의 수(예측된 } sentence\text{중에서)}}{\text{모든} 1-gram\text{쌍 (예측된 } sentence\text{중에서)}} = \frac{10}{14}$
- 2-gram precision: $\frac{\text{일치하는} 2-gram\text{의 수(예측된 } sentence\text{중에서)}}{\text{모든} 2-gram\text{쌍 (예측된 } sentence\text{중에서)}} = \frac{5}{13}$
- 3-gram precision: $\frac{\text{일치하는} 3-gram\text{의 수(예측된 } sentence\text{중에서)}}{\text{모든} 3-gram\text{쌍 (예측된 } sentence\text{중에서)}} = \frac{2}{12}$
- 4-gram precision: $\frac{\text{일치하는} 4-gram\text{의 수(예측된 } sentence\text{중에서)}}{\text{모든} 4-gram\text{쌍 (예측된 } sentence\text{중에서)}} = \frac{1}{11}$

$$(\prod_{i=1}^{4} precision_i)^{\frac{1}{4}} = (\frac{10}{14} \times \frac{5}{13} \times \frac{2}{12} \times \frac{1}{11})^{\frac{1}{4}}$$

https://donghwa-kim.github.io/BLEU.html

Transformer: Self-Attention

- Transformer variations

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213


ANY
questions?

References

Research Papers

- Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.
- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.
- Luong, M. T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025.
- Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. Advances in NIPS.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

Other Materials

- Alammar, J. Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models with Attention), https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/
- Alammar, J. The Illustrated Transformer, http://jalammar.github.io/illustrated-transformer/
- Olah, C. (2016). Attention and Augmented Recurrent Neural Networks, https://distill.pub/2016/augmented-rnns/
- Kim, D. (2019). Transfomer & BERT, https://www.youtube.com/watch?v=xhY7m8QVKjo&list=PLetSlH8YjIfUuwVM3j9XQ3UQTrY2KhdO1&index=18
- BLEU Score: https://donghwa-kim.github.io/BLEU.html
- https://nlp.seas.harvard.edu/2018/04/03/attention.html