Advanced Topics in Machine Learning

Alejo Nevado-Holgado

Lecture 16 (NLP 8) - Transformers

V 0.1 (4 Mar 2020 - final version)

Course structure

➢ **Introduction:** What is NLP. Why it is hard. Why NNs work well ← Lecture 9 (NLP 1)

➢ **Word representation:** How to represent the meaning of individual words
- Old technology: One-hot representations, synsets ← Lecture 9 (NLP 1)
- Embeddings: First trick that boosted the performance of NNs in NLP ← Lecture 9 (NLP 1)
  - Word2vec: Single layer NN. CBOW and skip-gram ← Lecture 10 (NLP 2)
  - Co-occurrence matrices: Basic counts and SVD improvement ← Lecture 10 (NLP 2)
  - Glove: Combining word2vec and co-occurrence matrices idea ← Lecture 10 (NLP 2)
  - Evaluating performance of embeddings ← Lecture 10 (NLP 2)

➢ **Named Entity Recognition (NER):** How to find words of specific meaning within text
- Multilayer NNs: Margin loss. Forward- and back-propagation ← Lecture 11 (NLP 3)
- Better loss functions: margin loss, regularisation ← Lecture 11 (NLP 3)
- Better initializations: uniform, xavier ← Lecture 11 (NLP 3)
- Better optimizers: Adagrad, RMSprop, Adam... ← Lecture 11 (NLP 3)

Course structure

➢ **Language modelling:** How to represent the meaning of full pieces of text
- Old technology: N-grams ← Lecture 12 (NLP 4)
- Recursive NNs language models (RNNs) ← Lecture 12 (NLP 4)
- Evaluating performance of language models ← Lecture 12 (NLP 4)
- Vanishing gradients: Problem. Gradient clipping ← Lecture 13 (NLP 5)
- Improved RNNs: LSTM, GRU, Bidirectional... ← Lecture 13 (NLP 5)

➢ **Machine translation:** How to translate text
- Old technology: Georgetown–IBM experiment and ALPAC report ← Lecture 14 (NLP 6)
- Seq2seq: Greedy decoding, encoder-decoder, beam search ← Lecture 14 (NLP 6)
- Attention: Simple attention, transformers, reformers ← Lecture 14 (NLP 6)
- Evaluating performance: BLEU ← Lecture 14 (NLP 6)

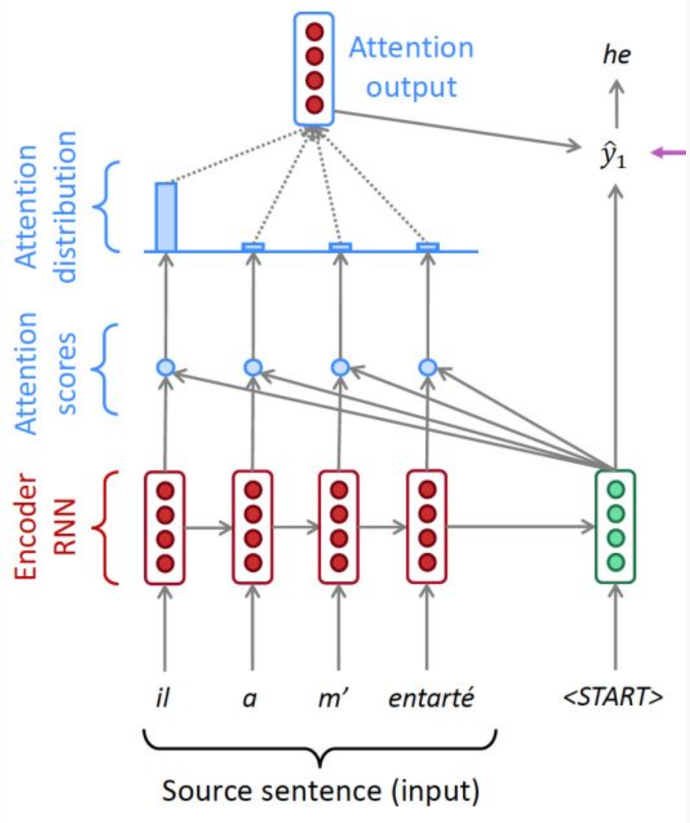
Course structure

- **Question Answering:** X
  - Task definition, datasets, cloze-style tasks, Attentive Reader ⟵ Lecture 15 (NLP 7)
- **Conference Resolution:** X
  - Task definition, pairs method, clustering method, language models ⟵ Lecture 15 (NLP 7)
- **Convolutional Neural Networks:** X
  - CNNs in vision, CNNs in language, example ⟵ Lecture 15 (NLP 7)
- **Transformers:** X
  - Architecture: encoder, self-attention, encoding position, decoder ⟵ Lecture 16 (NLP 8)
  - Existing systems. Ranking ⟵ Lecture 16 (NLP 8)

Transformers: Why a new architecture?

The problem: Recurrences are very slow to train, and their computations cannot be parallelized. Although LSTMs and GRUs capture long terms relationships much better than vanilla RNNs, they still don’t do it well enough.

The solution: We saw in previous lectures that attention can give any time step access to any other time step, no matter the length of the input. The whole purpose of recurrence in RNN architectures was accessing previous time steps no matter the length of the input. Why don’t we simply use pure attention to access all time steps? It is parallelizable, and maybe it captures long term relationships better than recurrence.

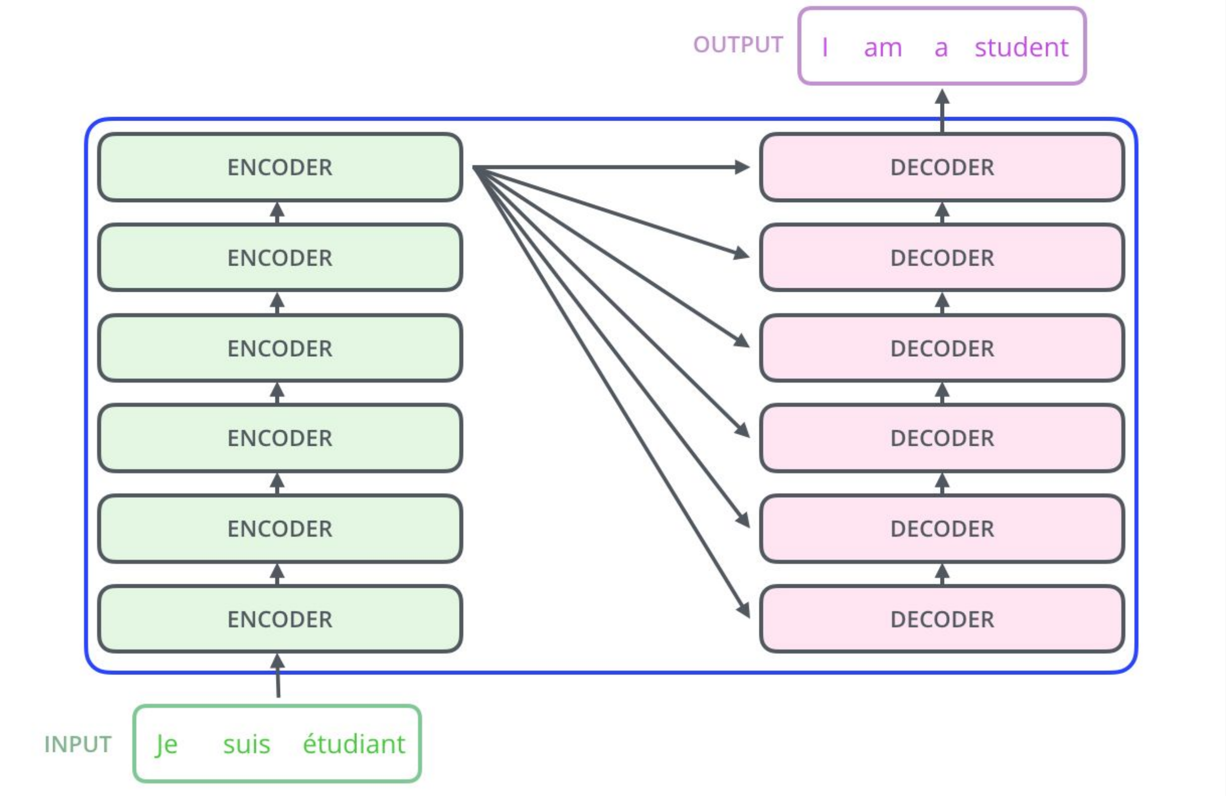


Transformers: Architecture

The transformer follows an encoder-decoder architecture, with all decoders attending to the last state of the encoder. This is the same as we studied a few lectures ago for translation (Lecture NLP 6)

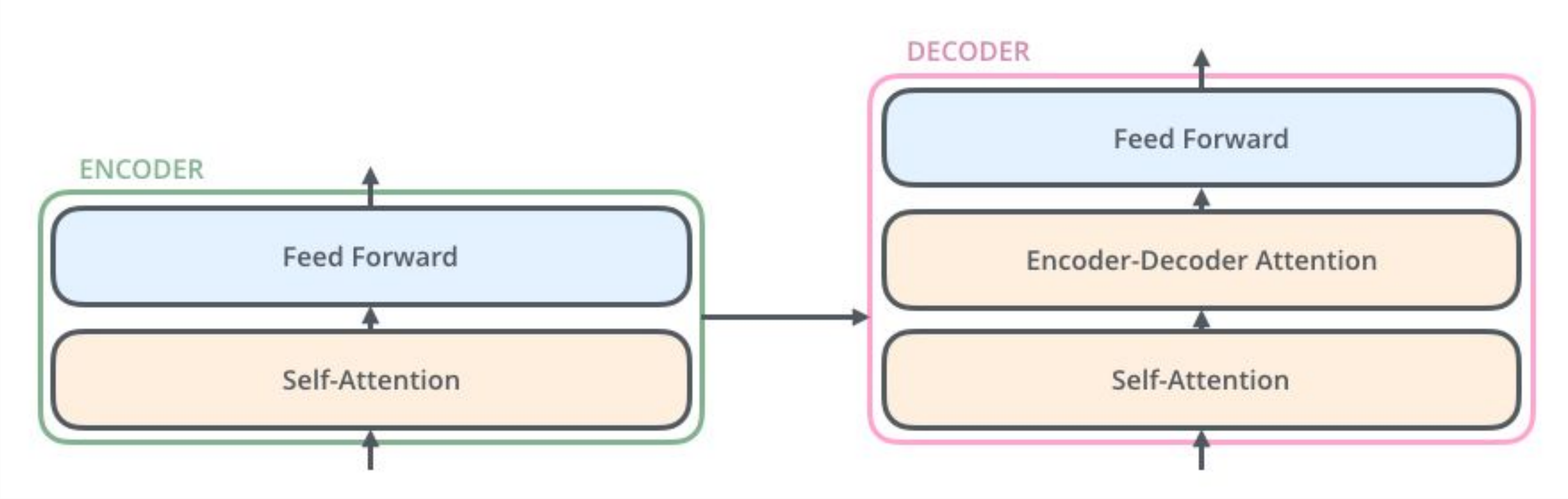
Original paper: https://arxiv.org/abs/1706.03762

Best description out there: http://jalammar.github.io/illustrated-transformer/



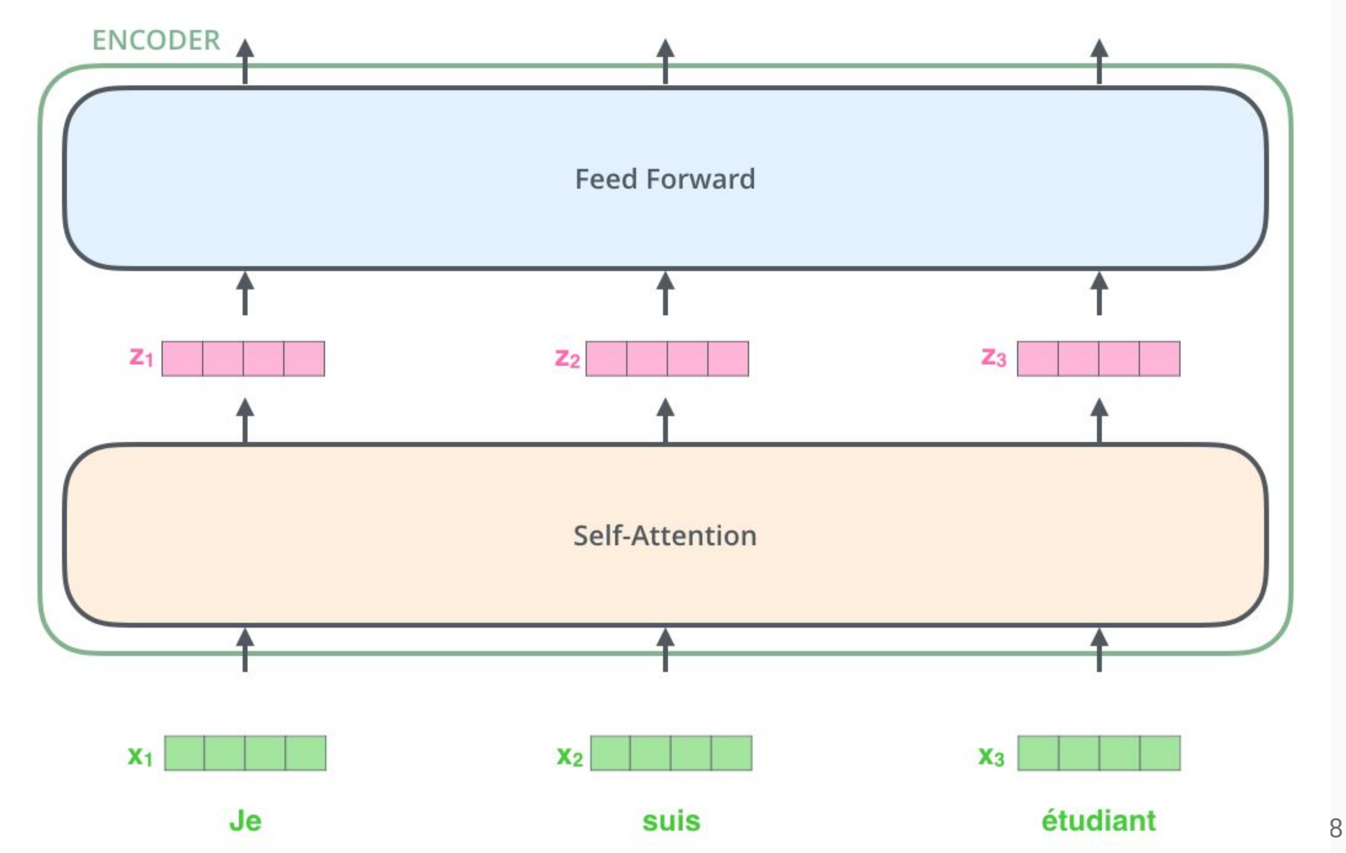
Transformers: Architecture

The difference between the transformer and the encoder-decoder of Lecture NLP 6 is on the internal architecture of each encoder and each decoder. Rather than simple hidden states, each encoder and decoder is a mini-NN of its own. This is sometimes called a 'module', 'block' or even 'layer', and it is very common in modern NNs (e.g. VGG16, ResNet, ByteNet…)



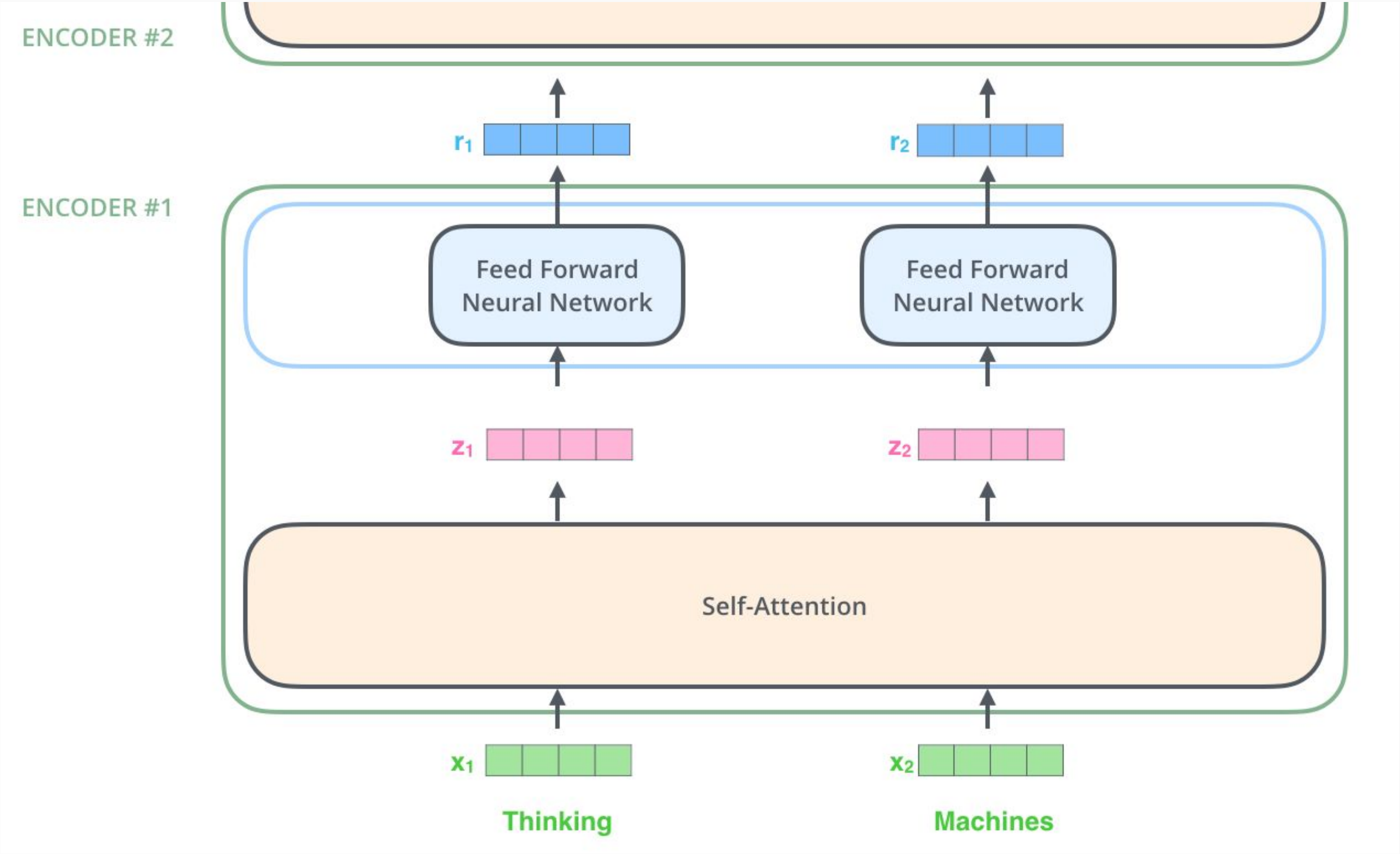
Transformers: Encoder

The first encoder receives an embedding $[\mathbf{x}^{(t)}]_X$ per word (X = num embedding dimensions). The self-attention layer mixes information across words, producing a new presentation per word $[\mathbf{z}^{(t)}]_Z$, like the attention mechanism of a seq2seq.



Transformers: Encoder

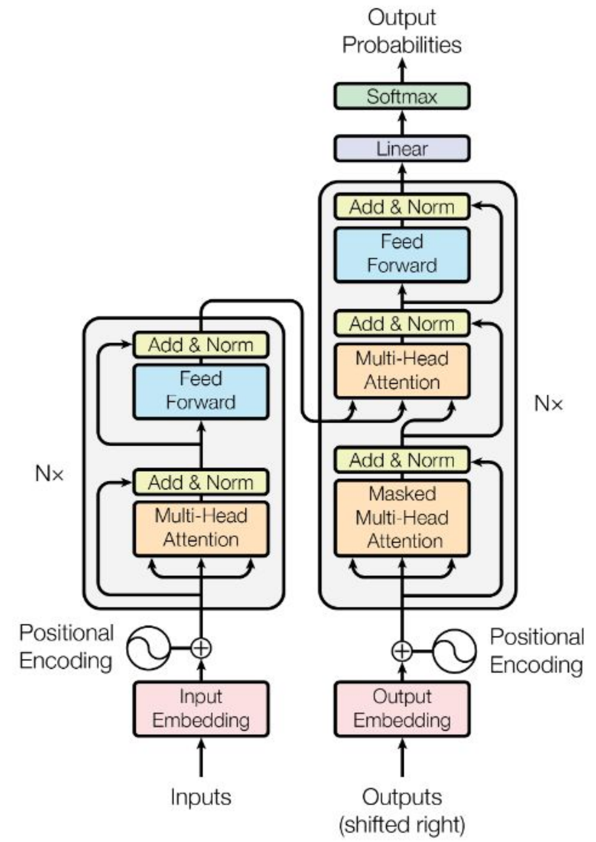
The new presentation per word $[\mathbf{z}^{(t)}]_Z$ that emerges from the self-attention layer, is then transformed with a feed forward fully connected NN into $[\mathbf{r}^{(t)}]_R$.



Transformers: Encoder

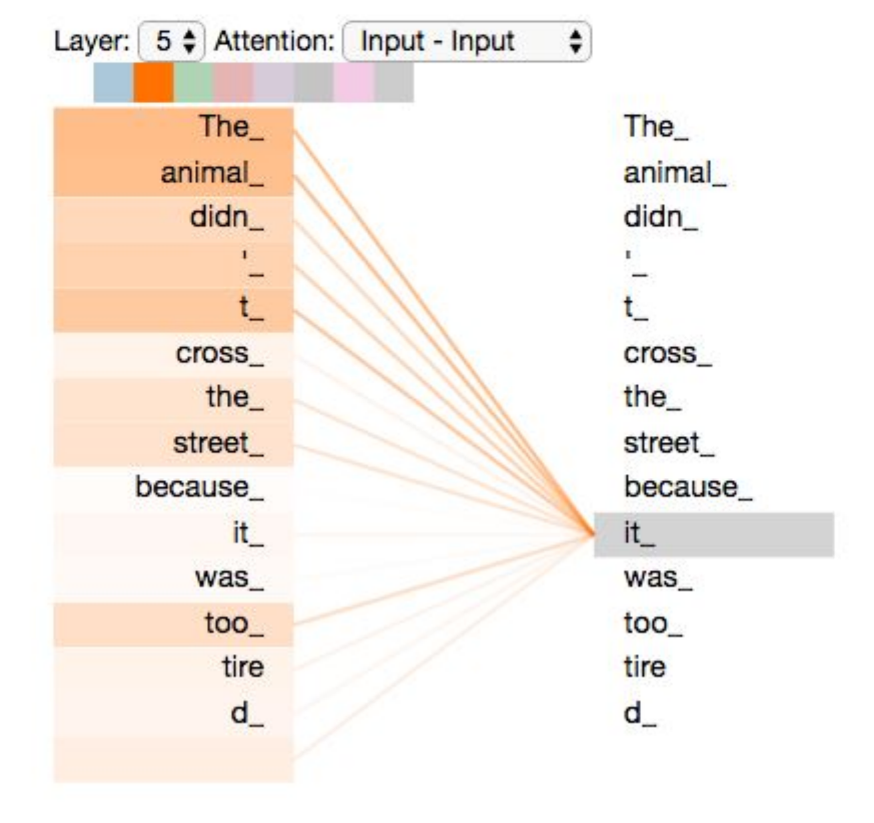
Be careful! In the original paper they use a residual connection in the self-attention and feed-forwards layers. The description by Jalammar does not emphasize this

Encoder: The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection [11] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$, where $\mathrm{Sublayer}(x)$ is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{\text{model}} = 512$.



Transformers: Self-attention

The effect of attention is that $[z^{(t)}]_Z$ (for a given time step 't') becomes a mixture of the original embeddings $[x^{(t)}]_X$. The idea is that each time step 'borrows' information from other time steps that it is related to. For instance, a pronoun may borrow information from the complement of the name that it refers to. Layer, the feed-forwards NN further transforms $[z^{(t)}]_Z$ into $[r^{(t)}]_R$, but this time without mixing information across time steps.



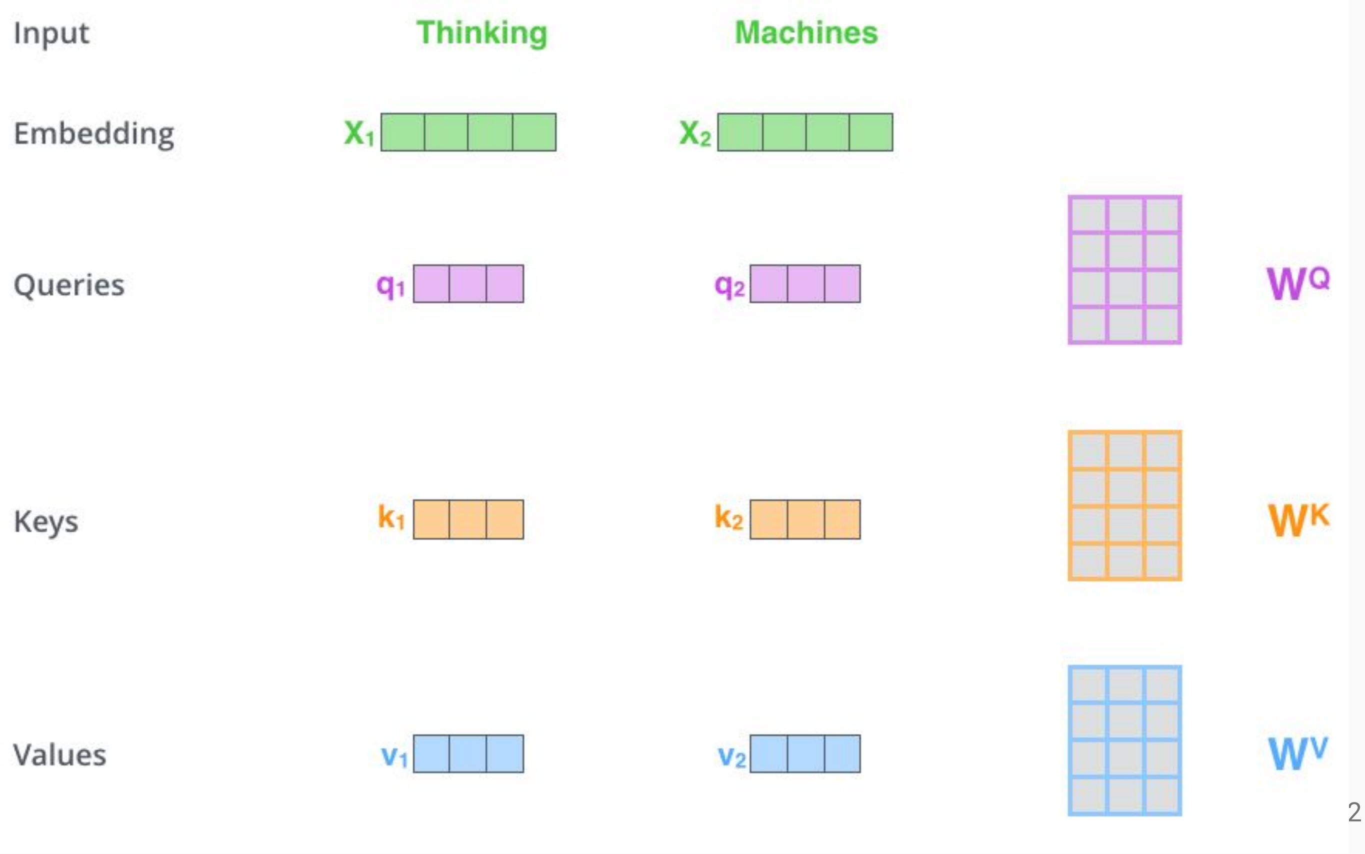
Transformers: Self-attention

The attention mechanism is multiplicative. We first create a query, a key and a value vector per time step by linearly transforming each embedding:

$[q^{(t)}]_Q = [W^Q]_{QX}\ [x^{(t)}]_X$

$[k^{(t)}]_Q = [W^K]_{KQ}\ [x^{(t)}]_X$

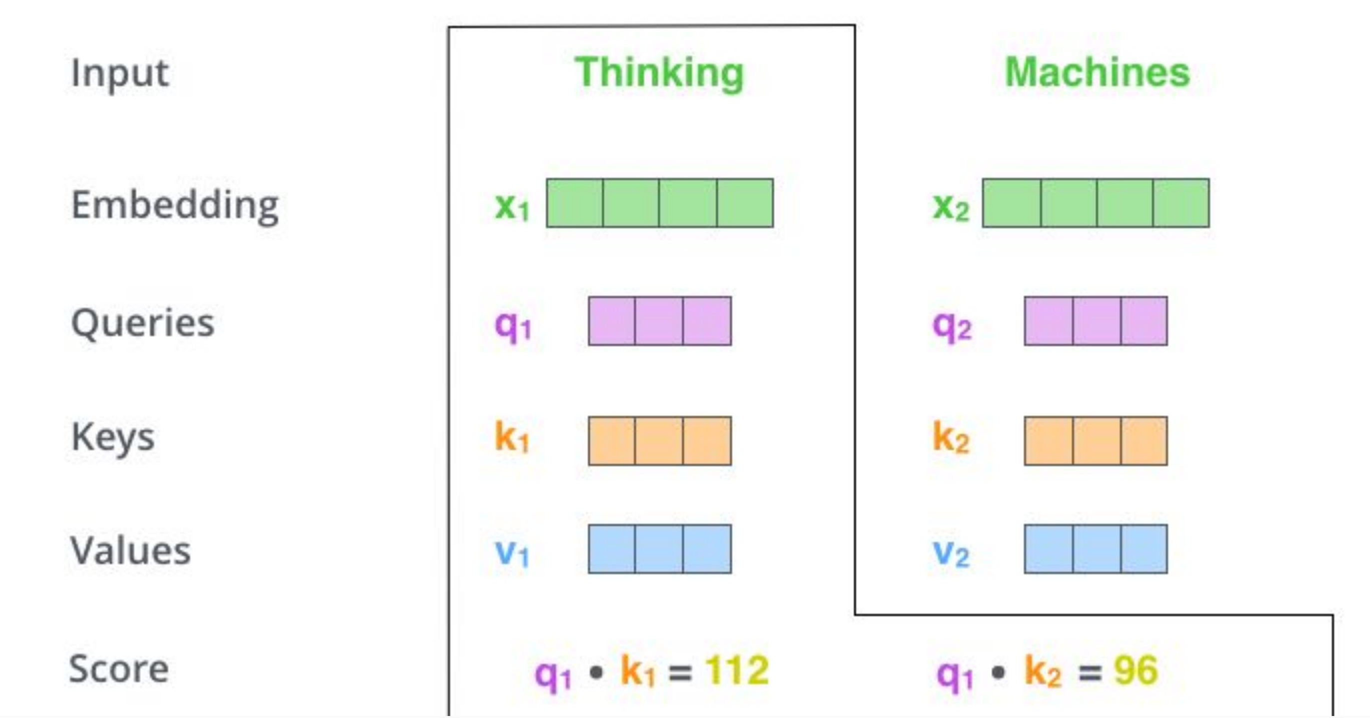
$[v^{(t)}]_V = [W^V]_{VX}\ [x^{(t)}]_X$



Transformers: Self-attention

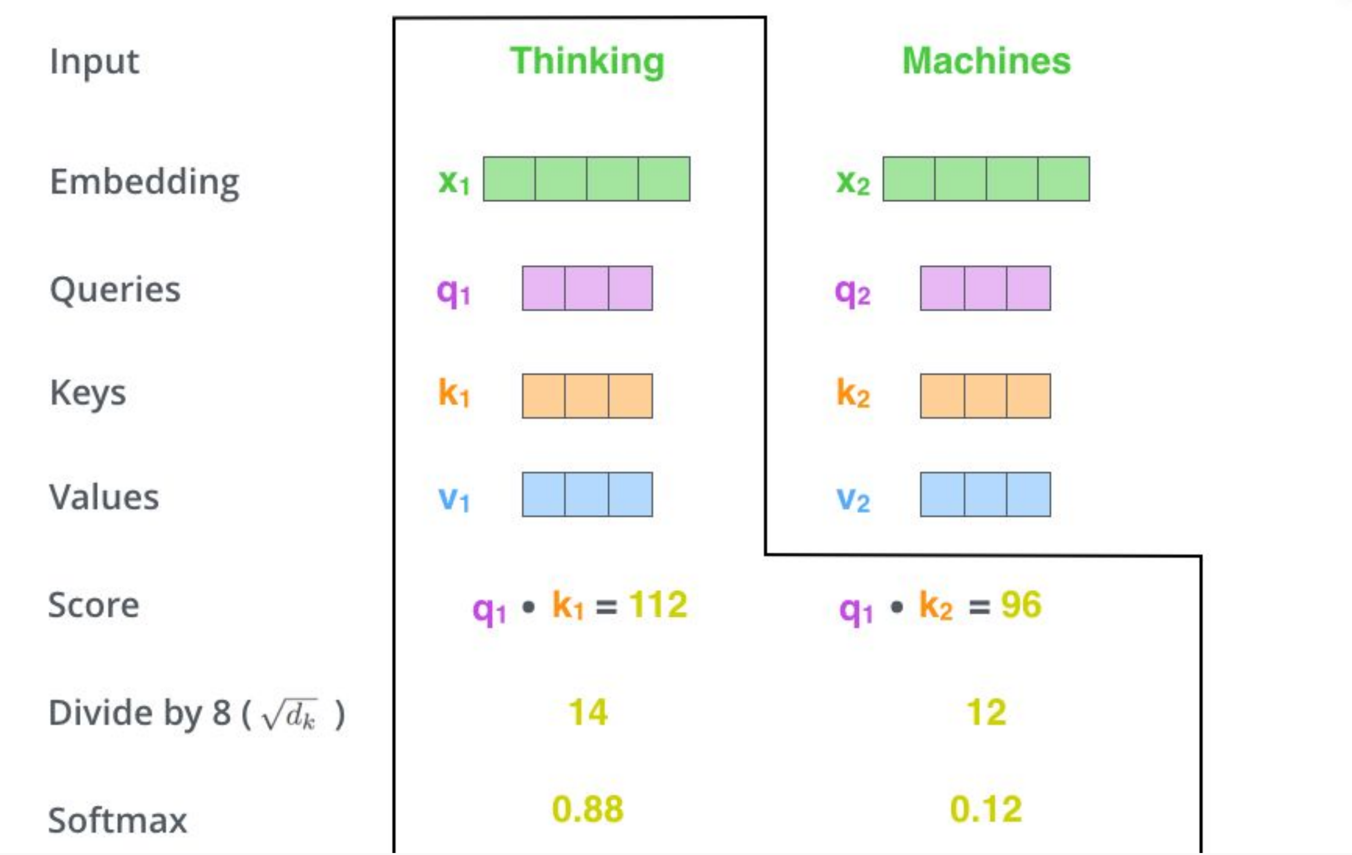
Then we dot-multiplicate each query with each key:

$[\mathbf{q}^{(t)}]_Q \cdot [\mathbf{k}^{(\tau)}]_Q$



Transformers: Self-attention

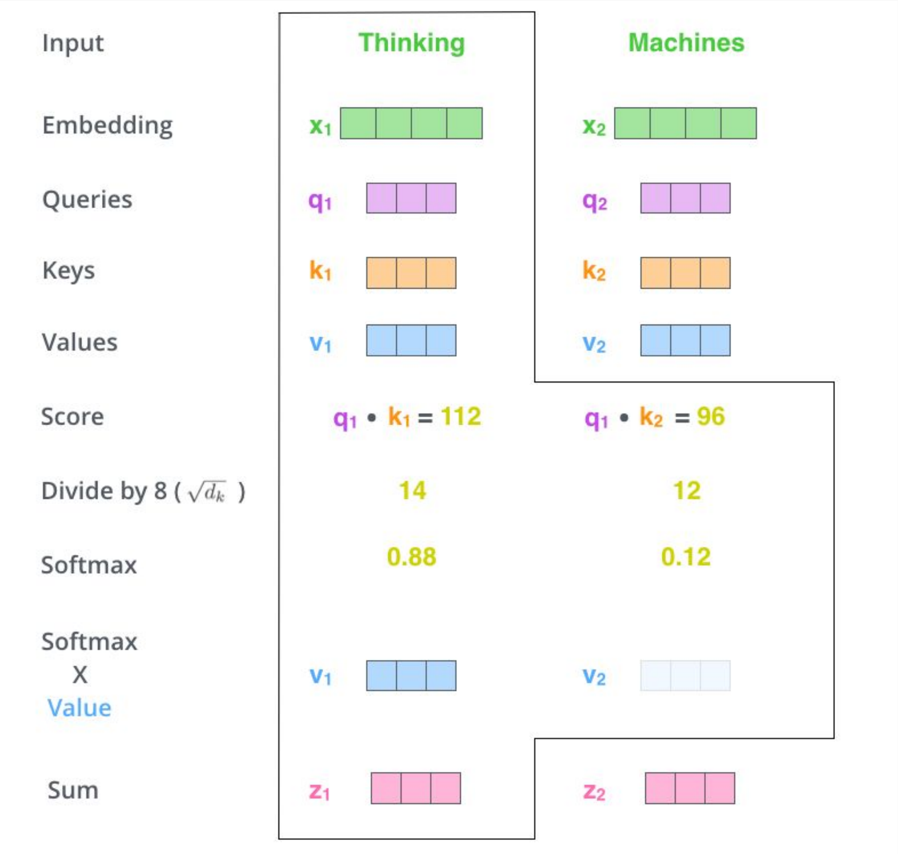
Then we rescale dividing by sqrt(Q) and apply softmax. Q is the number of dimensions of the query $[\mathbf{q}^{(t)}]_Q$ and the key $[\mathbf{k}^{(\tau)}]_Q$ (both vectors need to have the same size to allow for the dot product)



Transformers: Self-attention

The result is a weight given to each value to form the new hidden state:

$$[\mathbf{z}^{(t)}]_Z = \sum_{\tau} [\, sm_{\tau}([\mathbf{q}^{(t)}]_Q \cdot [\mathbf{k}^{(\tau)}]_Q) / \sqrt{Q} \,]_1 \, [\mathbf{v}^{(\tau)}]_Z$$

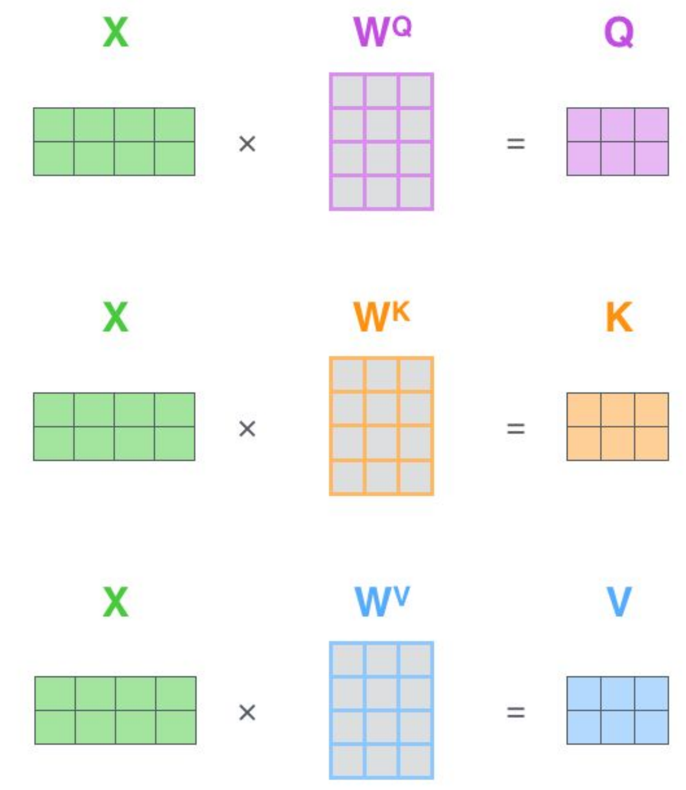


Transformers: Self-attention

Computationally, we do all these calculations in parallel by using matrices rather than vectors. A matrix represents all the time steps in one go:

$[Z]_{TZ} = [\ sm(\ [Q]_{TQ}\ [K^*]_{QT}\ )\ /\sqrt{Q}\ ]_{TT}\ [V]_{TZ}$

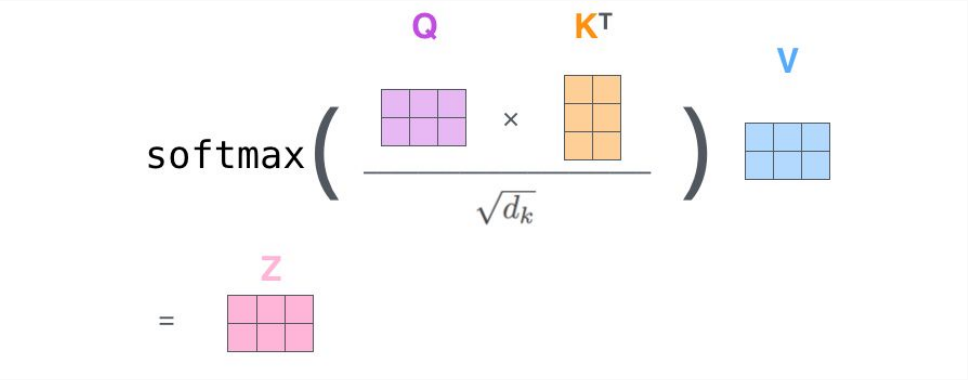
This is extremely efficient, because modern computers (and specially GPUs) have hardware optimized to perform these operations very fast.



Transformers: Self-attention

Computationally, we do all these calculations in parallel by using matrices rather than vectors. A matrix represents all the time steps in one go:

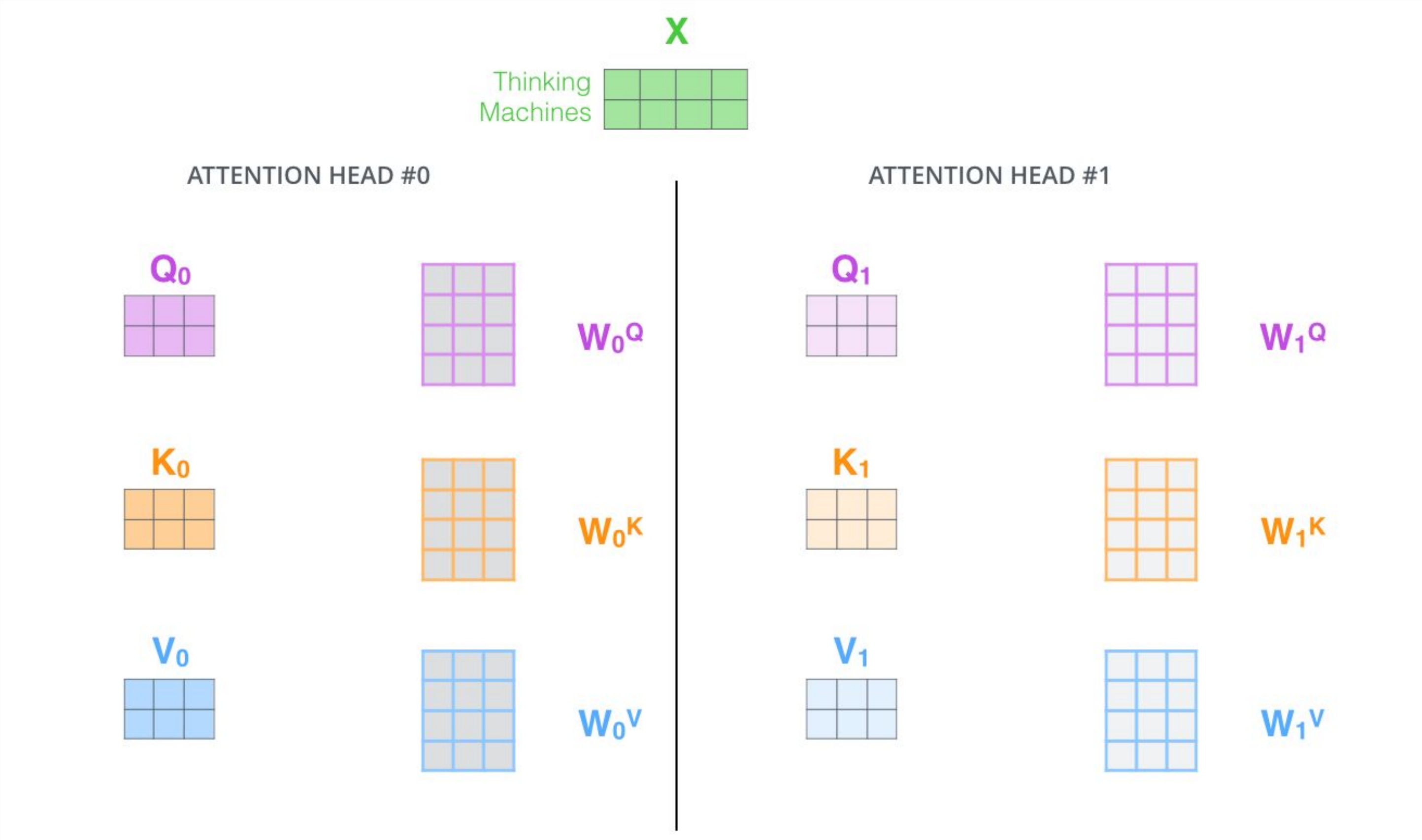
$$[Z]_{TZ} = [\ sm(\ [Q]_{TQ}\ [K^{*}]_{QT}\ )\ /\sqrt{Q}\ ]_{TT}\ [V]_{TZ}$$



This is extremely efficient, because modern computers (and specially GPUs) have hardware optimized to perform these operations very fast.

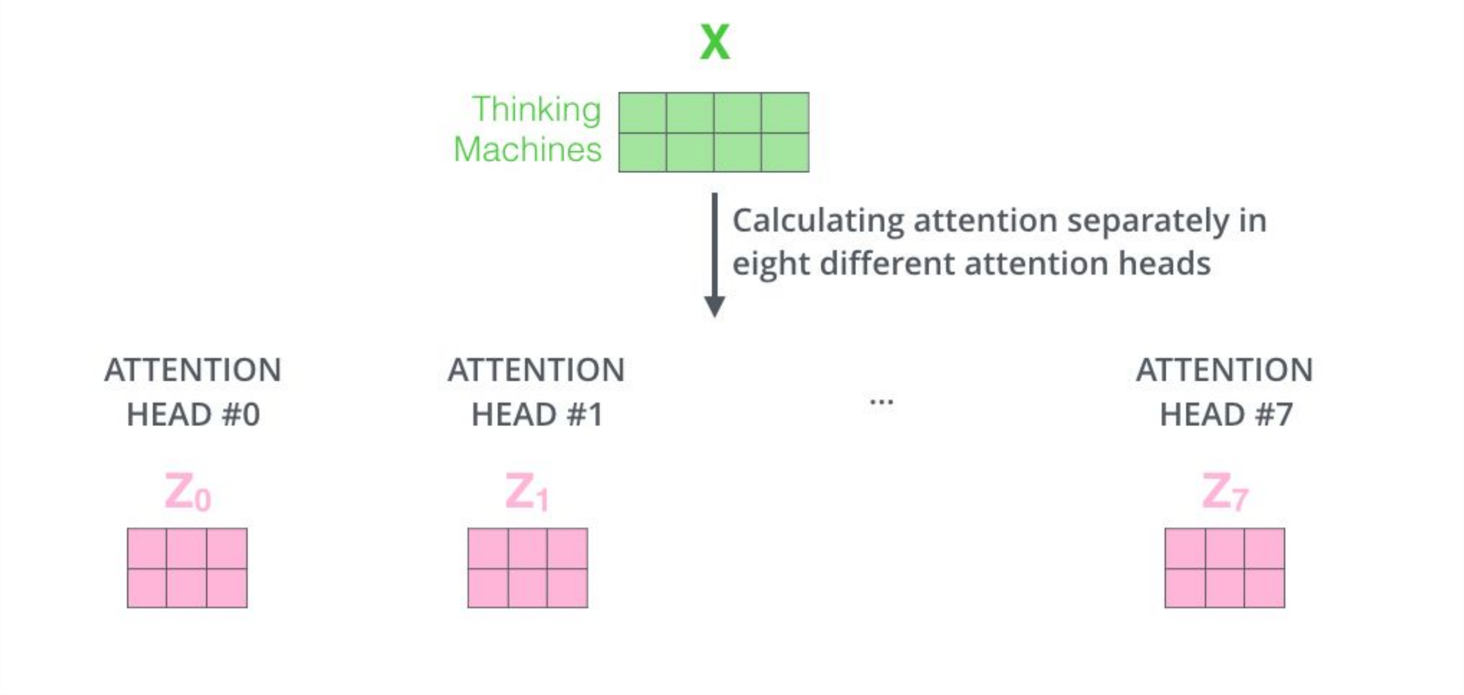
Transformers: Self-attention

Another novelty of the transformer, is that it uses several attention channels in parallel. They are called ‘heads’:



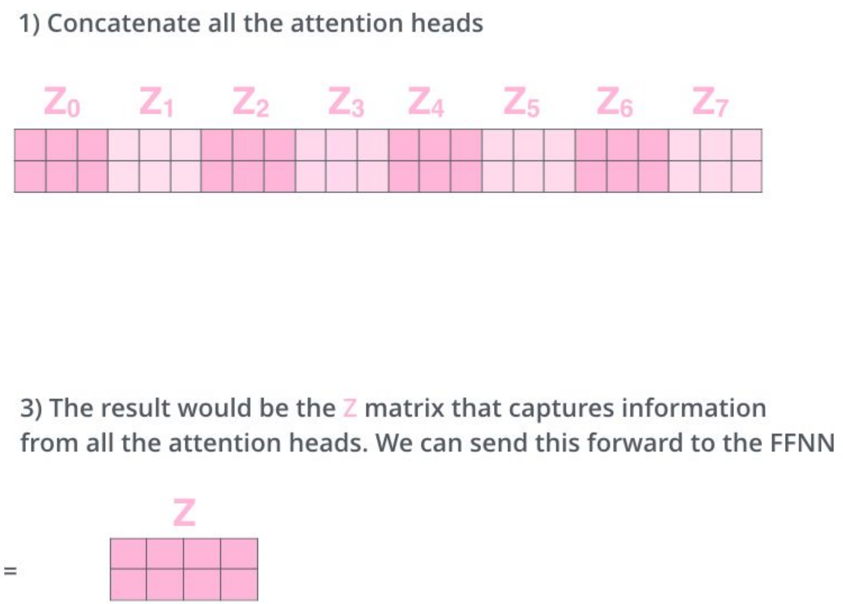
Transformers: Self-attention

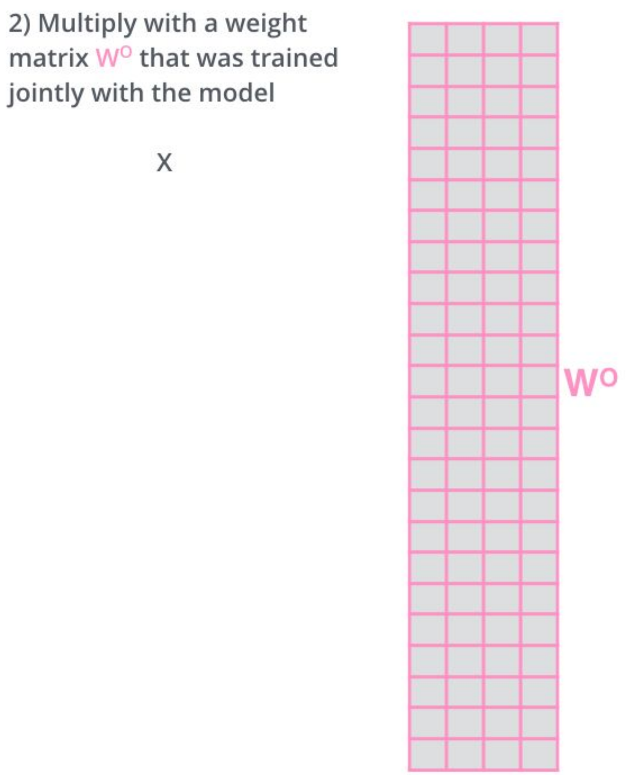
Another novelty of the transformer, is that it uses several attention channels in parallel. They are called ‘heads’, and their results are $[Z^1]_{TZ}$, $[Z^2]_{TZ}$, $[Z^3]_{TZ}$, …, $[Z^H]_{TZ}$ (H = number of heads):



Transformers: Self-attention

But we so many heads we end up with too many matrixes. To prevent the hidden states to group exponentially in size, we pool all heads with a linear transformation.

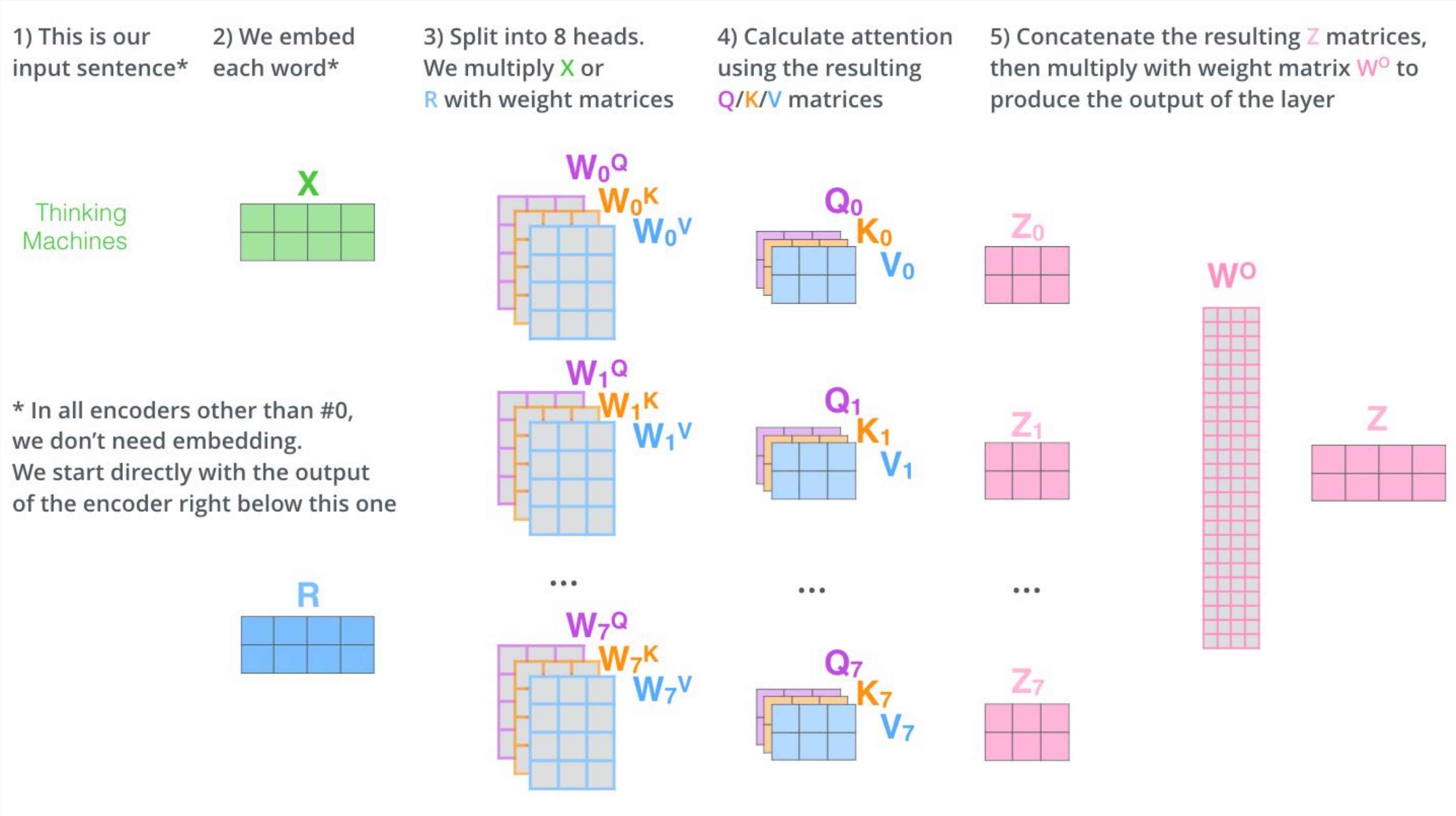




$$\text{concat}(\,[Z^1]_{TZ}, [Z^2]_{TZ}, [Z^3]_{TZ}, \ldots, [Z^H]_{TZ}\,) = [Z]_{T(Z\times H)}$$

$$[Z]_{T(Z\times H)} \times [W^O]_{(Z\times H)Z} = [Z^{\text{next layer}}]_{TZ}$$

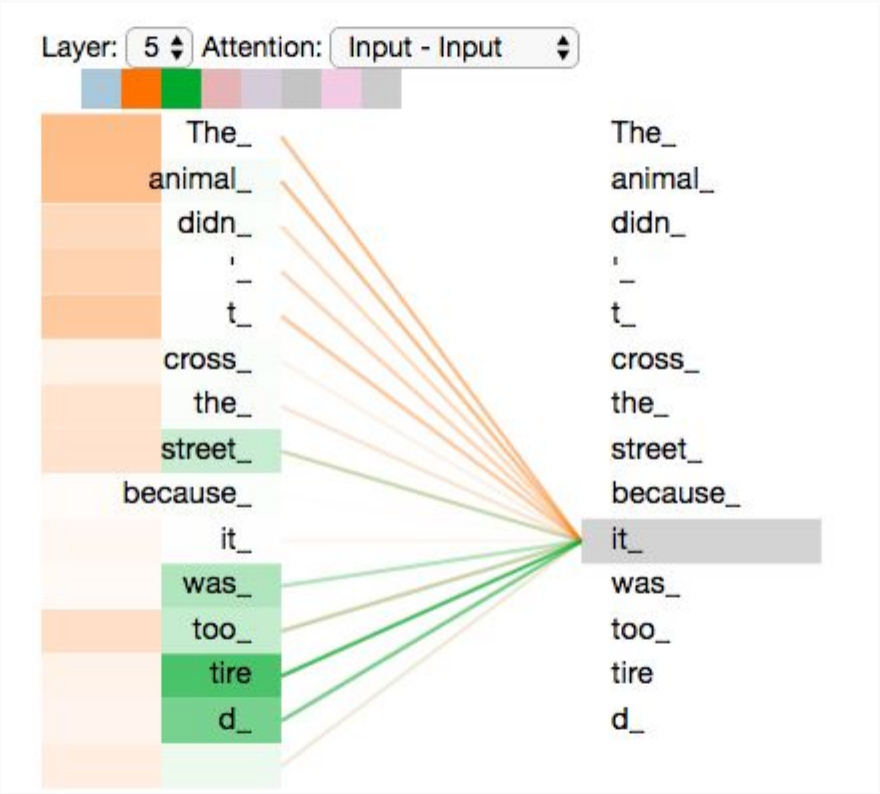
Transformers: Self-attention



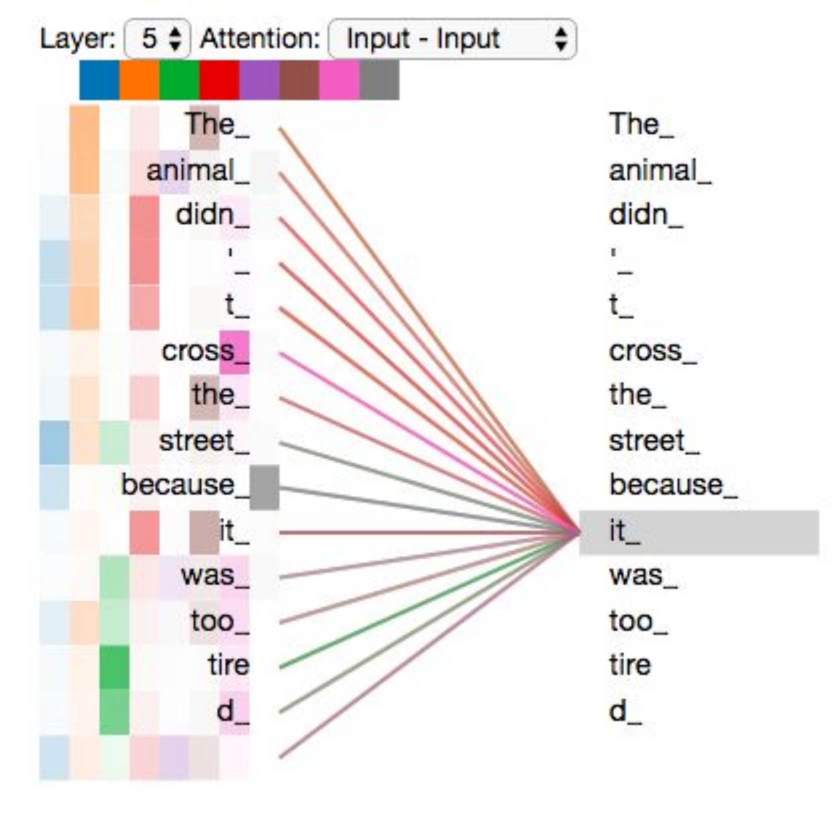
Transformers: Self-attention

The effect of attention is that $[z^{(t)}]_Z$ (for a given time step 't') becomes a mixture of the original embeddings $[x^{(t)}]_X$. The idea is that each time step 'borrows' information from other time steps that it is related to. For instance, a pronoun may borrow information from the complement of the name that it refers to. Layer, the feed-forwards NN further transforms $[z^{(t)}]_Z$ into $[r^{(t)}]_R$, but this time without mixing information across time steps.

Multi-heads = It does all of this several times, with different with a different $[W^Q]_{QX}$, $[W^K]_{KQ}$, $[W^V]_{VX}$ per head

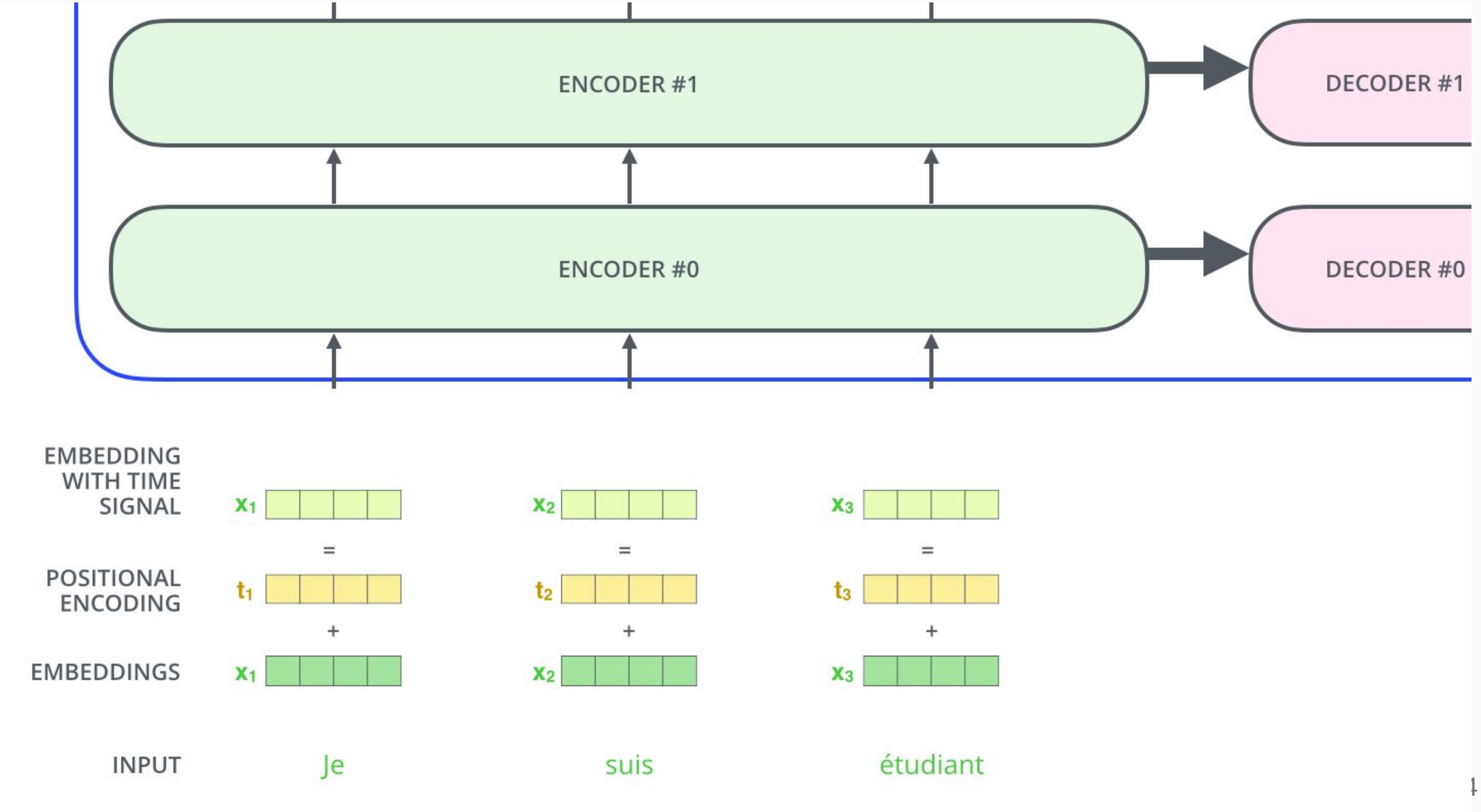


Transformers: Self-attention



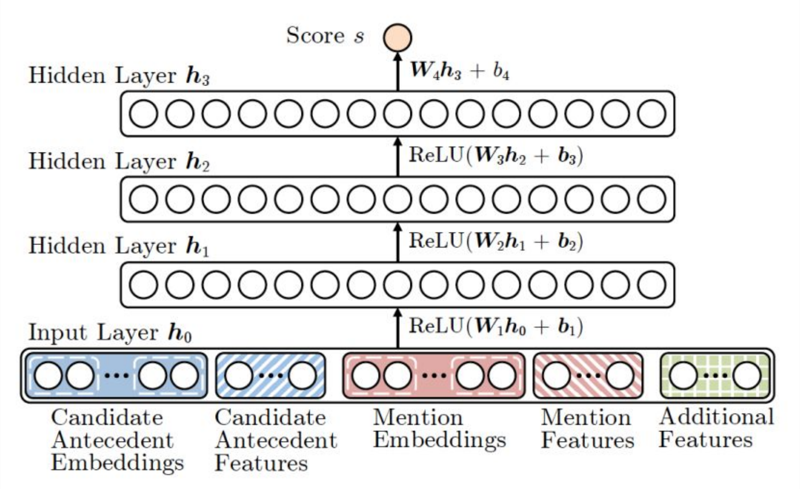
Transformers: Encoding position

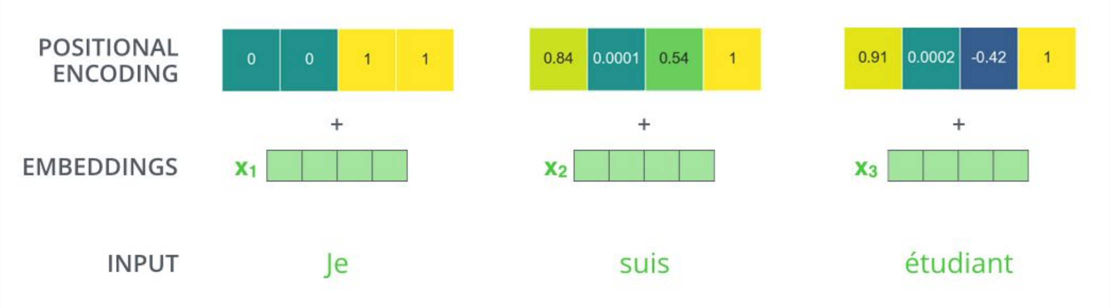
We also add information about the position of each word. We do this with a positional encoding vector $[\mathbf{t}^{(t)}]_X$ per possible position.



Transformers: Encoding position

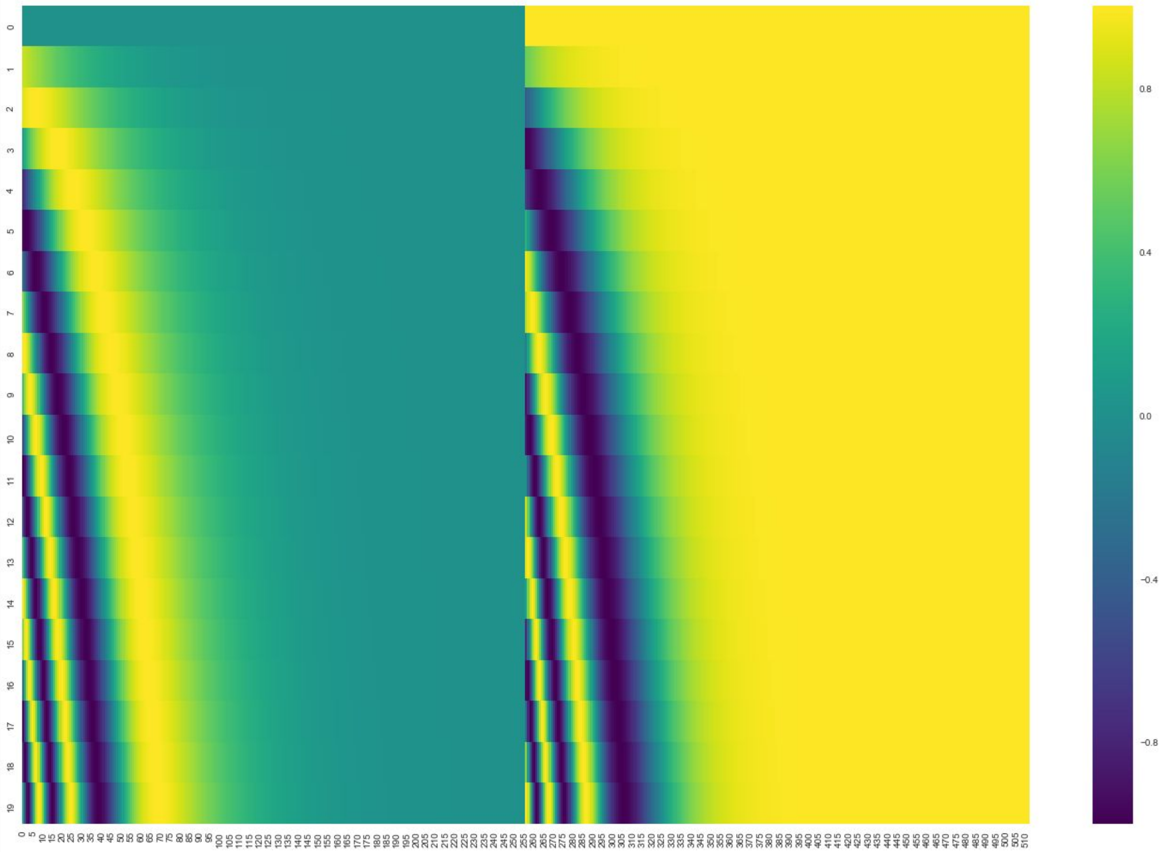
- This is a similar idea that we have used before for conference resolution, where we concatenated extra features to the word embeddings.
- This is a common trick in NN NLP
- However the transformer e-wise multiplies $[\mathbf{t}^{(t)}]_X$ rather than concatenating it to $[\mathbf{x}^{(t)}]_X$.



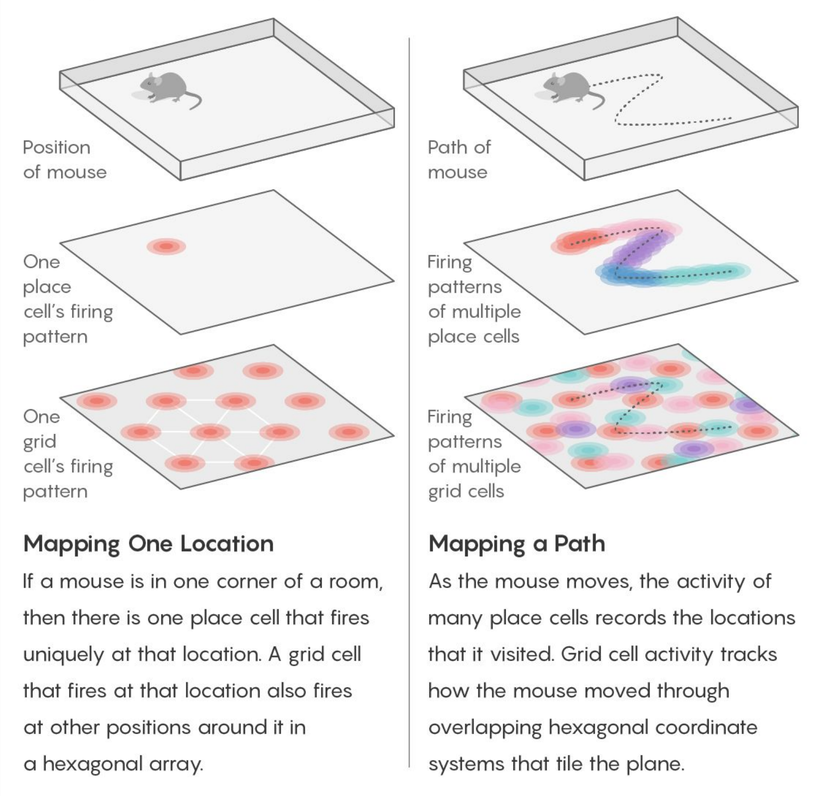


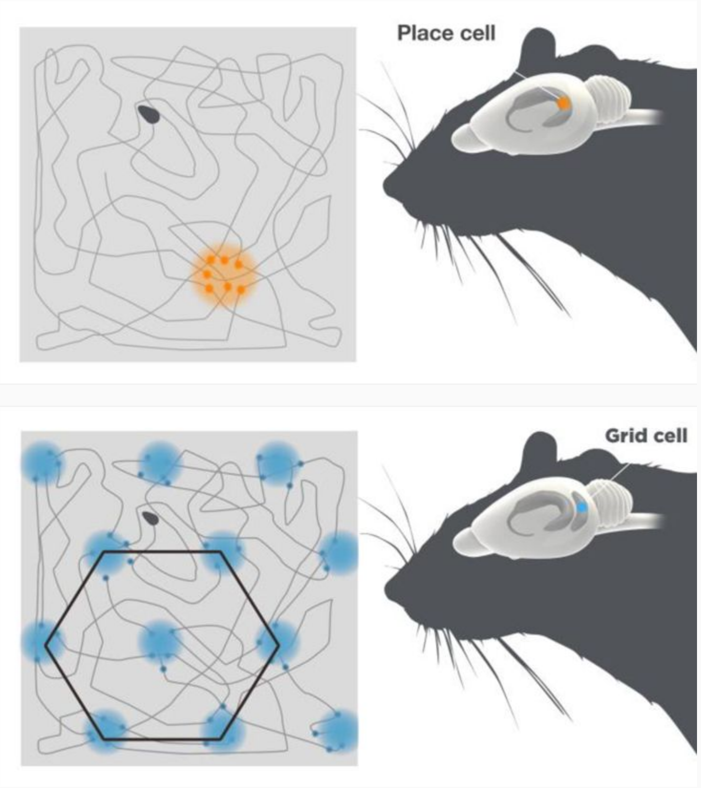
Transformers: Encoding position

- ➢ The positional encoding vectors have pre-specified values
- ➢ These values follow some sort of wavelet function
- ➢ This is quite similar to how the hippocampus in the human brain encodes position!



The real brain: Encoding position

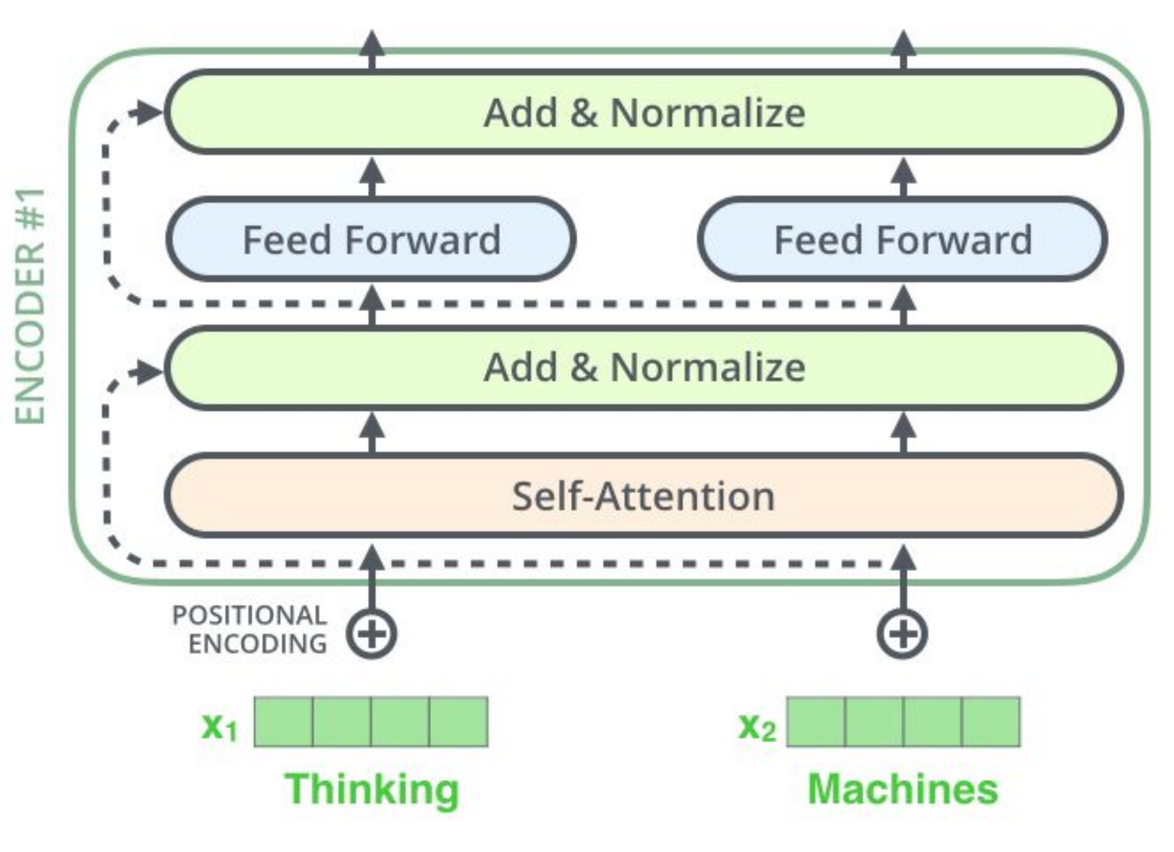




Transformers: Encoder

To simplify the explanation, we have so far ignored two smaller details of the architecture.

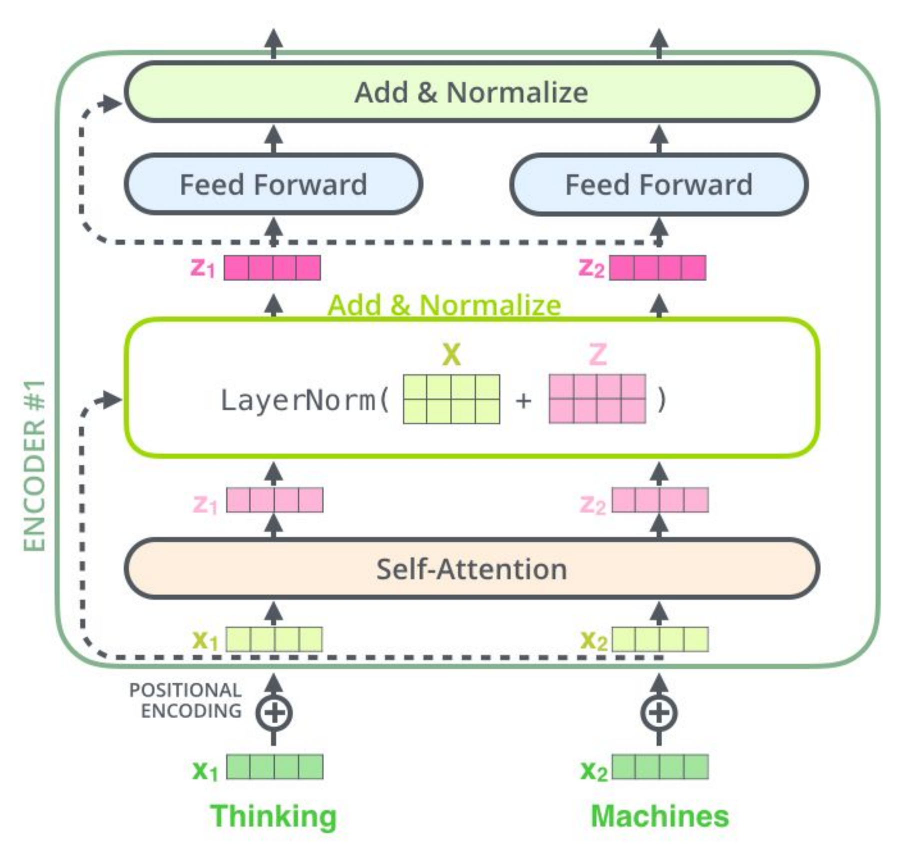
- 1) There is a residual connection bypassing each layer
- 2) There is a normalization step after each layer.



Transformers: Encoder

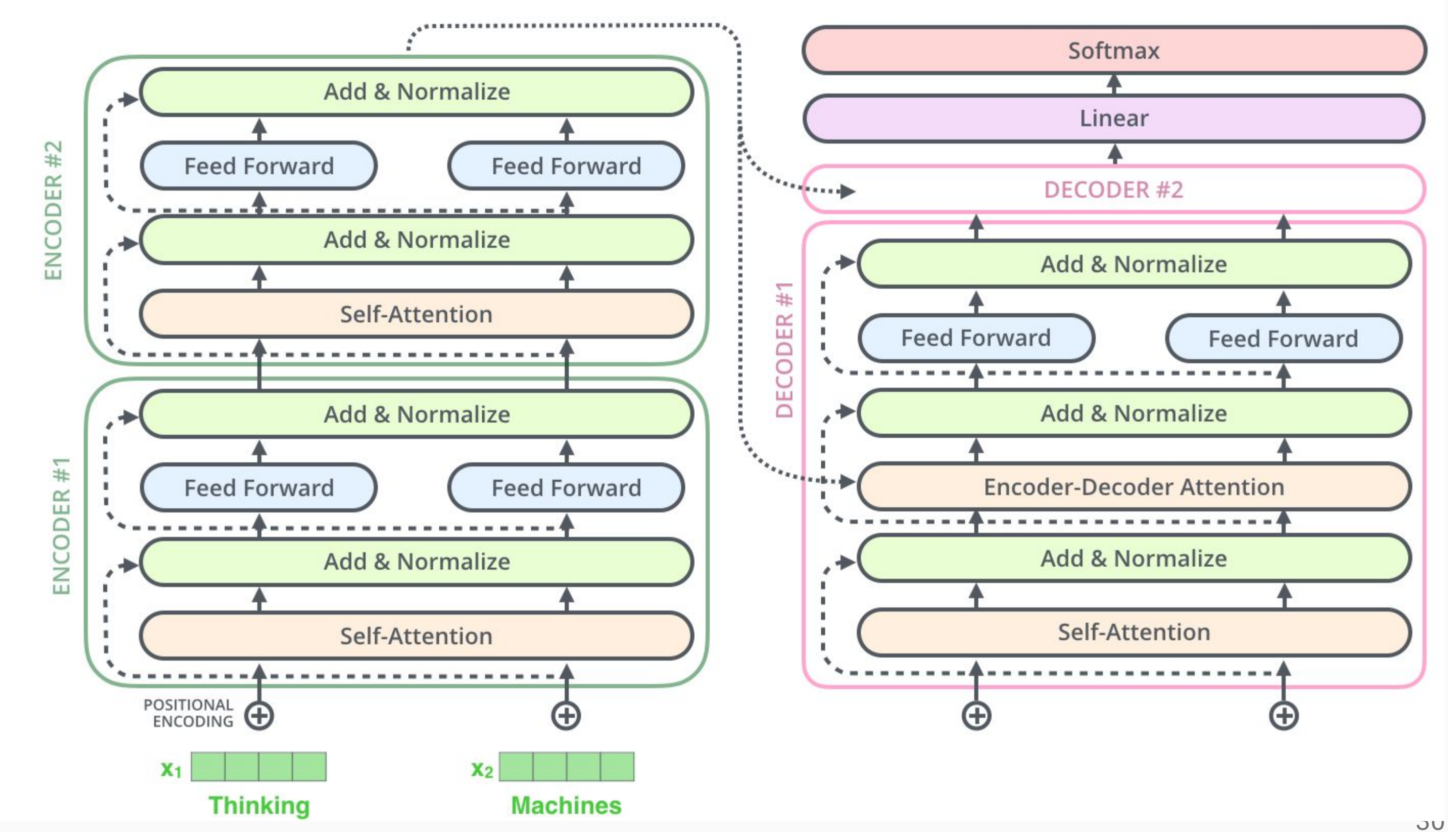
To simplify the explanation, we have so far ignored two smaller details of the architecture.

- 1) There is a residual connection bypassing each layer
- 2) There is a normalization step after each layer.

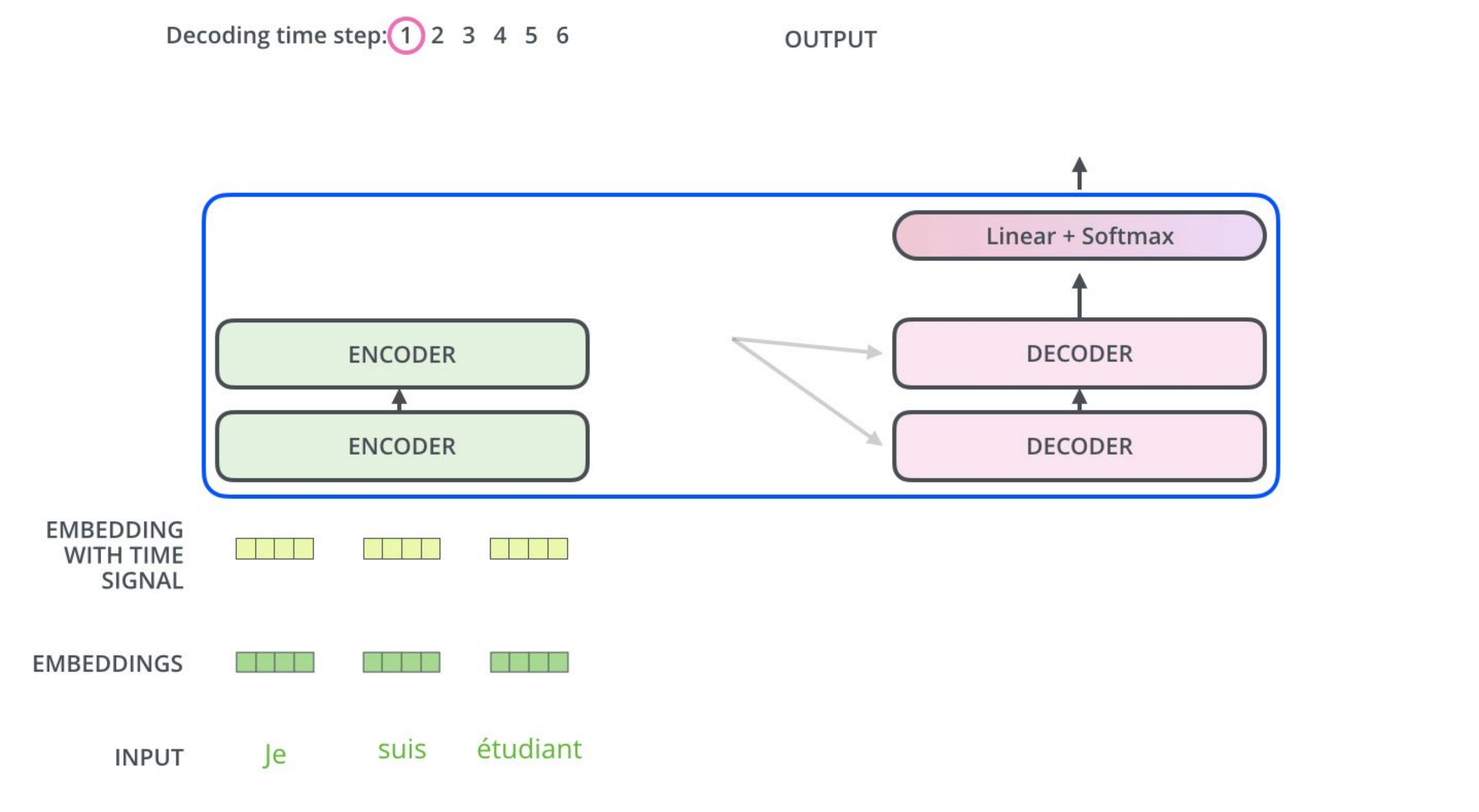


Transformers: Encoders → decoders

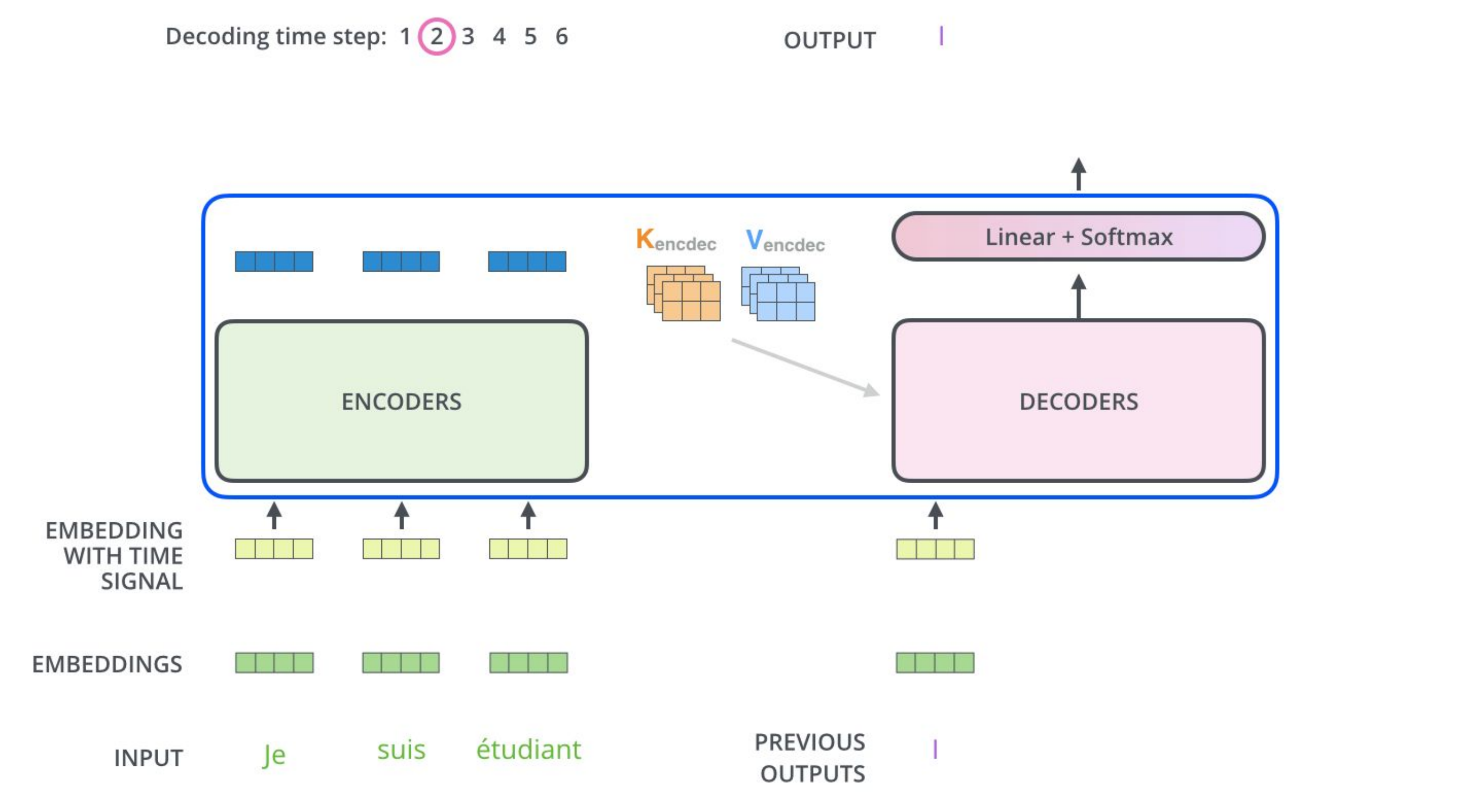
Besides self-attention, the decoder also uses encoder-decoder attention. This attention is the same as simple self-attention, but it also uses the outputs of the last layer of the encoder



Transformers: Encoders → decoders



Transformers: Encoders → decoders

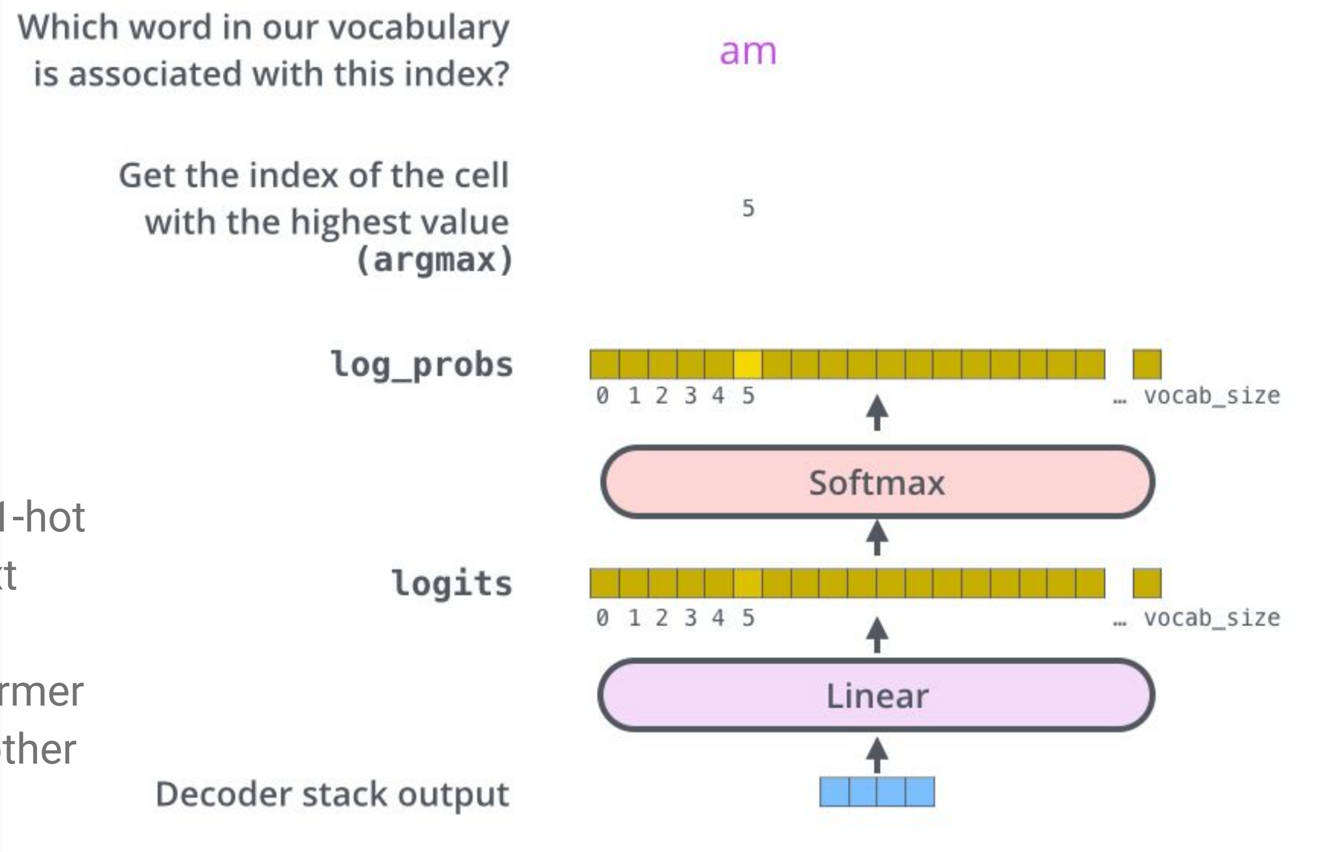


Transformers: Output

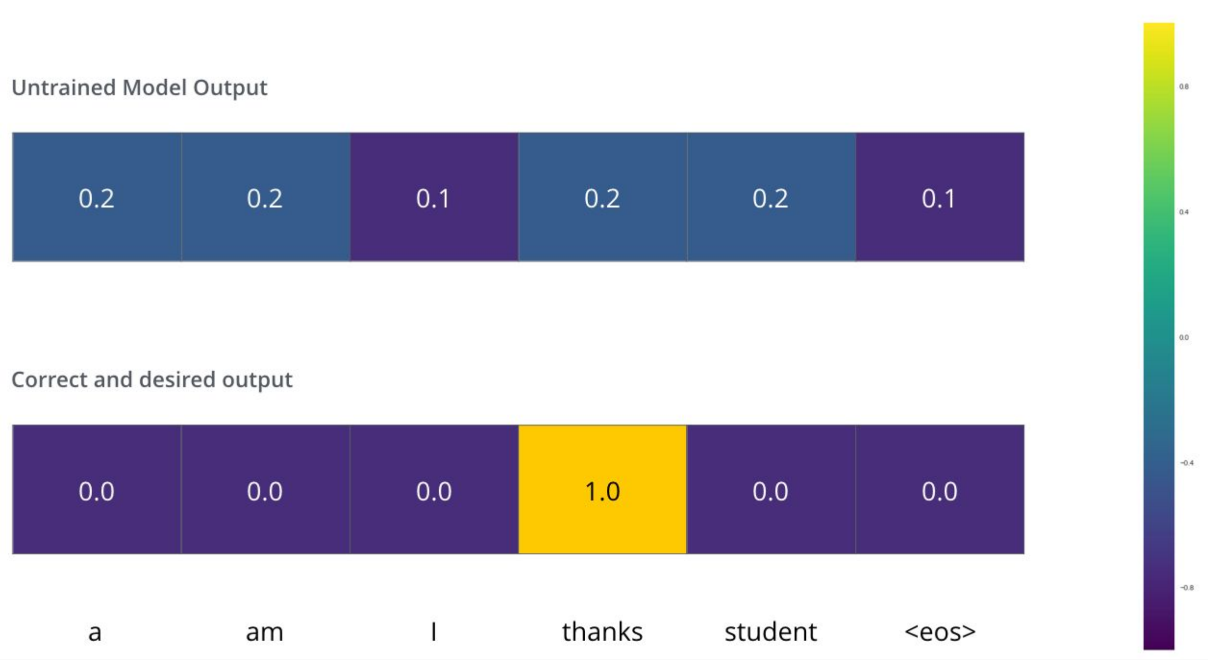
The NN is trained in a language model task. Remember from lecture NLP 4, this consists on predicting the next word.

The output of the neural network tries to find the 1-hot representation of the next word

Once trained, the transformer can be re-used in many other NLP tasks



Transformers: Output



Transformers: Output

After training the model, its outputs will approximate the desired 1-hot representations of words in the vocabulary

Trained Model Outputs

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

Target Model Outputs

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

Transformers: Existing systems

All of these models are Transformer architecture models

ULMfit	GPT	BERT	GPT-2
Jan 2018	June 2018	Oct 2018	Feb 2019
Training:	Training	Training	Training
1 GPU day	240 GPU days	256 TPU days ~320–560 GPU days	~2048 TPU v3 days according to a reddit thread
fast.ai	OpenAI	Google AI	OpenAI

Transformers: Existing systems

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Jan 15, 2019	BERT + MMFT + ADA (ensemble) *Microsoft Research Asia*	**85.082**	**87.615**
2 Jan 10, 2019	BERT + Synthetic Self-Training (ensemble) *Google AI Language* https://github.com/google-research/bert	84.292	86.967
3 Dec 13, 2018	BERT finetune baseline (ensemble) *Anonymous*	83.536	86.096
4 Dec 16, 2018	Lunet + Verifier + BERT (ensemble) *Layer 6 AI NLP Team*	83.469	86.043
4 Dec 21, 2018	PAML+BERT (ensemble model) *PINGAN GammaLab*	83.457	86.122
5 Dec 15, 2018	Lunet + Verifier + BERT (single model) *Layer 6 AI NLP Team*	82.995	86.035

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Jan 10, 2020	Retro-Reader on ALBERT (ensemble) *Shanghai Jiao Tong University* http://arxiv.org/abs/2001.09694	**90.115**	**92.580**
2 Nov 06, 2019	ALBERT + DAAF + Verifier (ensemble) *PINGAN Omni-Sinitic*	90.002	92.425
3 Sep 18, 2019	ALBERT (ensemble model) *Google Research & TTIC* https://arxiv.org/abs/1909.11942	89.731	92.215
4 Jan 23, 2020	albert+transform+verify (ensemble) *qianxin*	89.528	92.059
5 Dec 08, 2019	ALBERT+Entailment DA (ensemble) *CloudWalk*	88.761	91.745
6 Feb 20, 2020	Tuned ALBERT (ensemble model) *Group Data & Analytics Cell | Aditya Birla Group)* https://www.adityabirla.com/About/group-data-and-analytics	88.637	91.230

Course structure

- **Question Answering:** X
  - Task definition, datasets, cloze-style tasks, Attentive Reader ← Lecture 15 (NLP 7)
- **Conference Resolution:** X
  - Task definition, pairs method, clustering method, language models ← Lecture 15 (NLP 7)
- **Convolutional Neural Networks:** X
  - CNNs in vision, CNNs in language, example ← Lecture 15 (NLP 7)
- **Transformers:** X
  - Architecture: encoder, self-attention, encoding position, decoder ← Lecture 16 (NLP 8)
  - Existing systems. Ranking ← Lecture 16 (NLP 8)

Literature

➢ **Papers** =

- Attention is all you need. https://arxiv.org/abs/1706.03762
- The illustrated transformer. http://jalammar.github.io/illustrated-transformer/
- Language Models are Unsupervised Multitask Learners. https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf
- Reformer: The Efficient Transformer. https://arxiv.org/abs/2001.04451
- Illustrating the reformer. https://towardsdatascience.com/illustrating-the-reformer-393575ac6ba0