Recent Advances of Language Transformer Machine

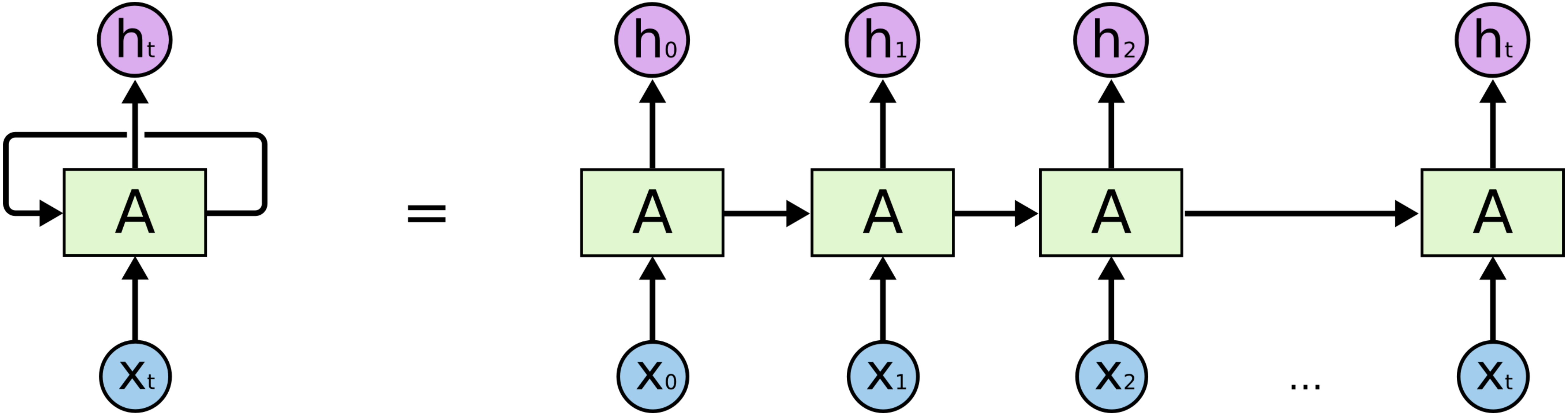
Presenters: Lecheng Zheng, Dawei Zhou

Outline

- Recurrent Neural Networks (RNN)
- Long-Short Term Memory (LSTM)
- Transformer
    - Encoder
    - Multi-head Attention
    - Decoder

Recurrent Neural Networks

- The Recurrent Neural Network $\boldsymbol{A}$ processes some input x_i and outputs h_i. A loop allows information to be passed from one state to the next.
- It aims to learn the dependencies of words.



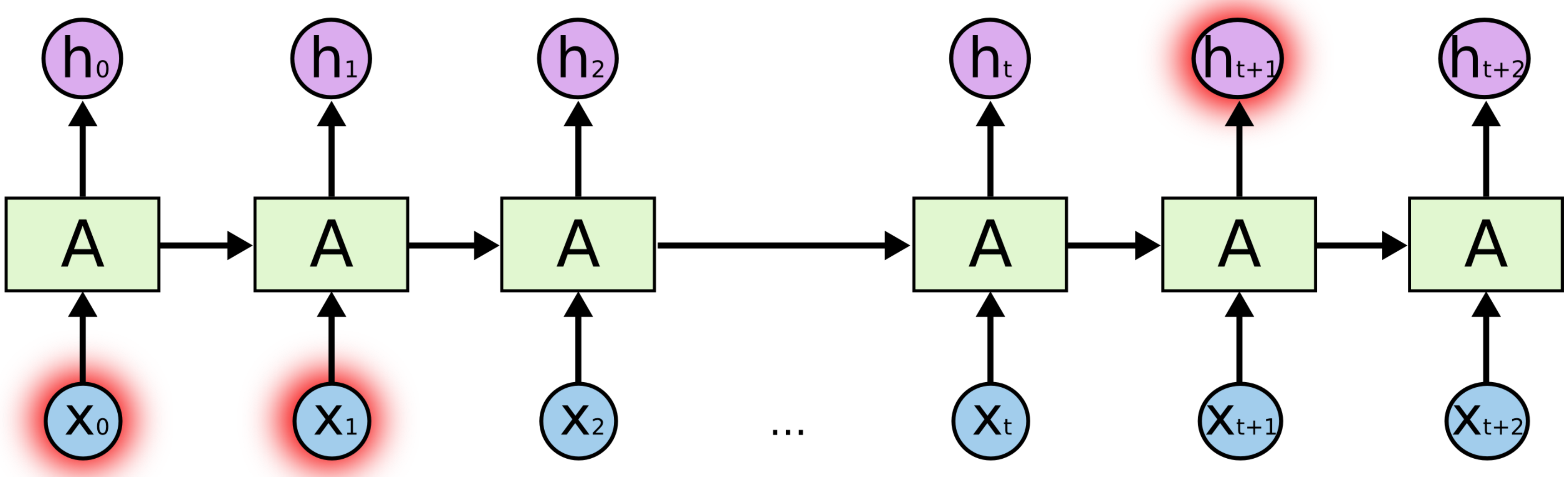
The problem of long-term dependencies

- If we are trying to predict the next word of the sentence **"the clouds in the __"**, we don't need further context. It's pretty obvious that the next word is going to be **sky.**

- However**,** you are trying to predict the last word of the text: "Tom grew up in France… Tom can speak fluent __?"**.**
  - Recent information suggests that the next word is probably a language.
  - If we want to narrow down which language, we need context of France, that is further back in the text.

The problem of long-term dependencies

- In theory, RNNs could learn this long-term dependencies.
- In practice, they don't seem to learn them, as the model often forgets the content of distant positions in the sequence.

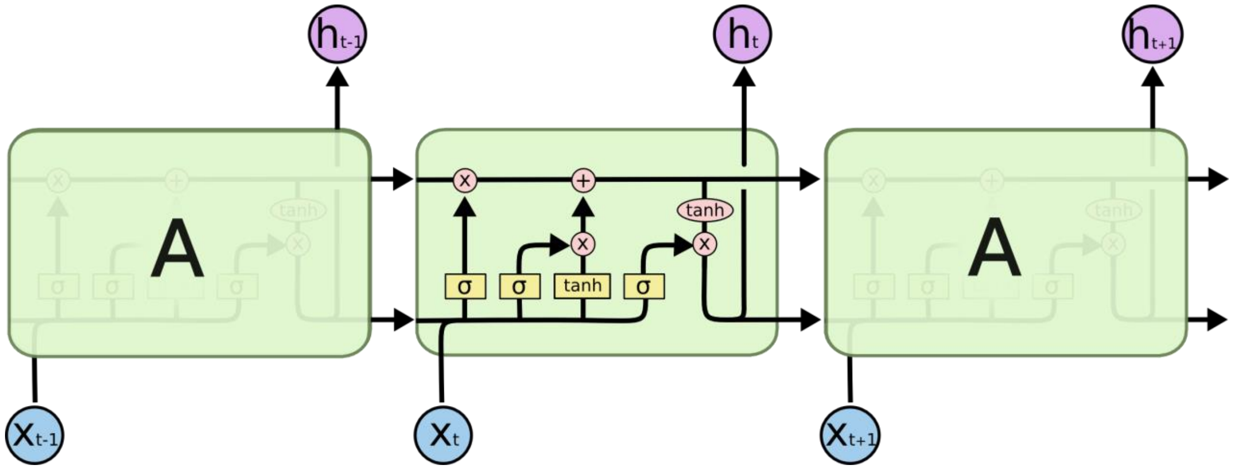


Outline

- Recurrent Neural Networks (RNN)
- Long-Short Term Memory (LSTM)
- Transformer
    - Encoder
    - Multi-head Attention
    - Decoder

Long-Short Term Memory (LSTM)

- Each cell takes as inputs x_t, the previous cell state c_{t-1} and the output of the previous cell h_{t-1}.
- It generates a new cell state c_t, and an output h_t.
- The forget gate is responsible for removing irrelevant information and remember important information.



The problem with LSTMs

- The same problem that happens to RNNs generally, happen with LSTMs, i.e. when sentences are too long (maybe a paragraph) LSTMs still don’t do too well.

- The reason is that the probability of keeping the context from a word that is far away from the current word being processed decreases exponentially with the distance from it.

- That means that when sentences are too long, the model often forgets the content of distant positions in the sequence.

Outline

- Recurrent Neural Networks (RNN)
- Long-Short Term Memory (LSTM)
- Transformer
  - Encoder
  - Multi-head Attention
  - Decoder

What is a transformer?

- Transformer is a type of neural network architecture, which is developed to solve the problem of **neural machine translation.**
  - Speech recognition
  - Text-to-speech transformation
  - Language translation

Why transformer?

- **Efficiency is the key!**

- Feed forward layer of transformer could be computed in parallel.

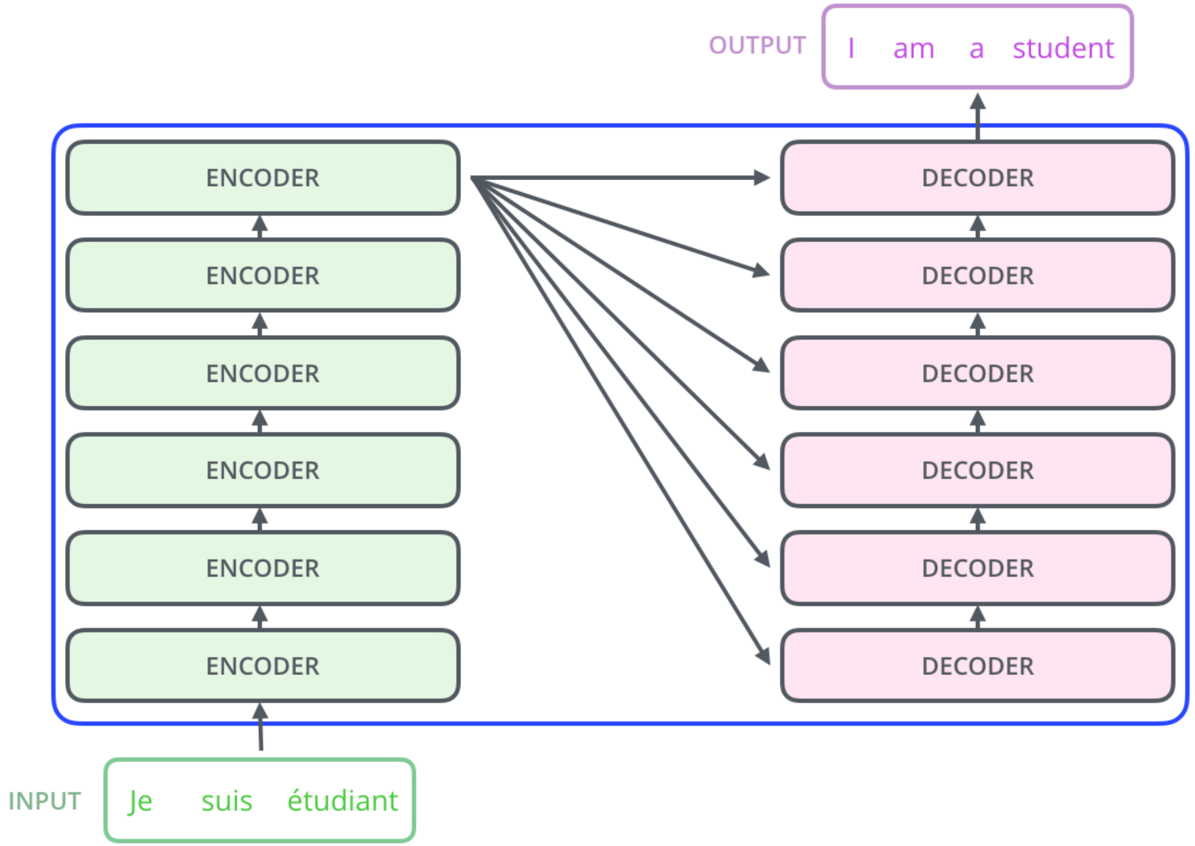
- Transformers try to solve the problem by using Convolutional Neural Networks together with attention models.

- Attention boosts the speed of how fast the model can translate from one sequence to another.

Structure of Transformer

- The transformer consists of encoders and decoders.

- The transformer shown in the figure on the right hand side stacks six encoders and six decoders.



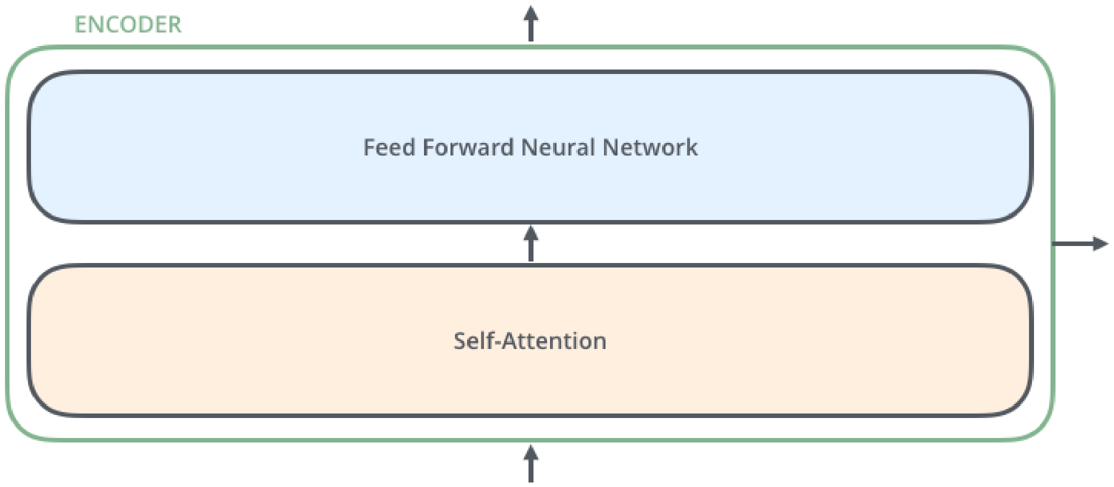
Outline

- Recurrent Neural Networks (RNN)
- Long-Short Term Memory (LSTM)
- Transformer
  - Encoder
  - Multi-head Attention
  - Decoder

Encoder of a transformer

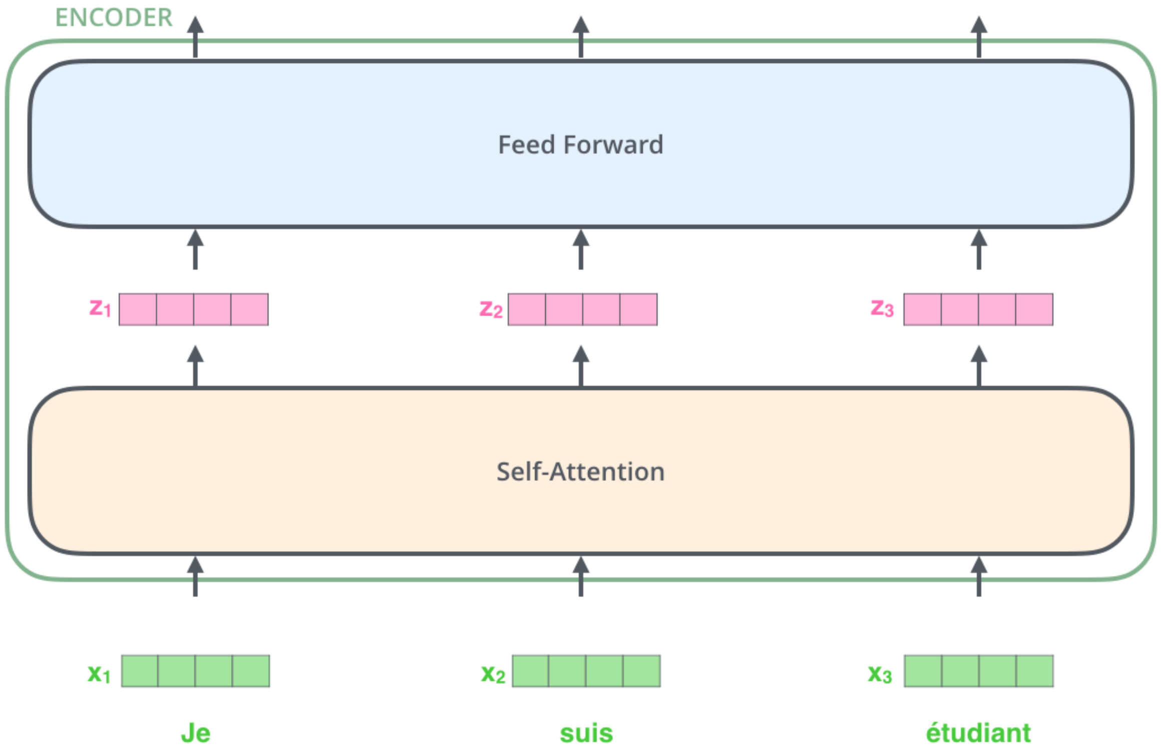
- Each encoder consists of two layers: Self-Attention Layer and Feed Forward Neural Network.

- The goals of self-attention layer
  - To learn the dependencies between the words in the sentence.
  - To use that information to capture the internal structure of the sentence.



Encoder

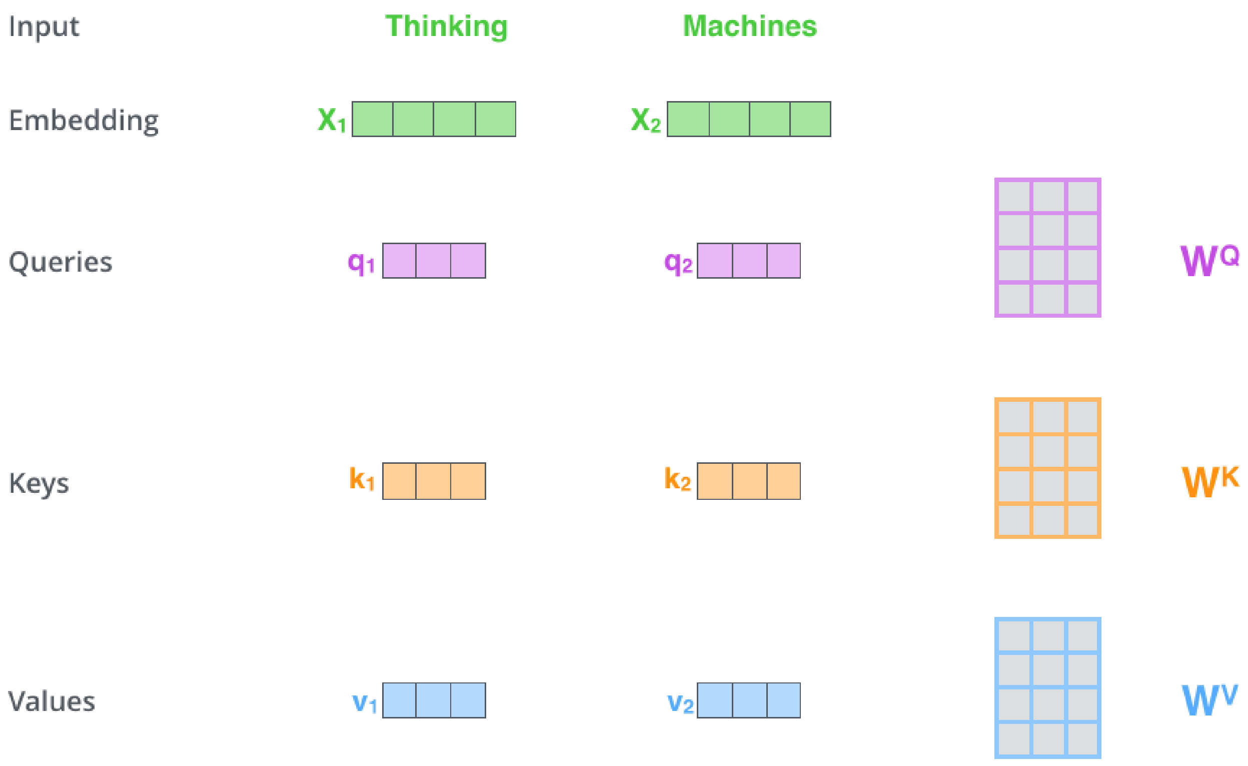
- The input of the encoder is the word embeddings.
- There are dependencies between words in the self-attention layer.
- The feed-forward layer does not have those dependencies, and thus the various paths can be executed in parallel.



Self-Attention Layer of Encoder

- The first step: for each word, we create a Query vector, a Key vector, and a Value vector, where $X_i \in R^{n\times d}$ is input word embedding, $W^Q \in R^{d\times p}$, $W^k \in R^{d\times p}$, $W^v \in R^{d\times p}$ are three weight matrices (hyperparameters), $q_i \in R^{1\times p}$, $k_i \in R^{1\times p}$, $v_i \in R^{1\times p}$ are three vectors.

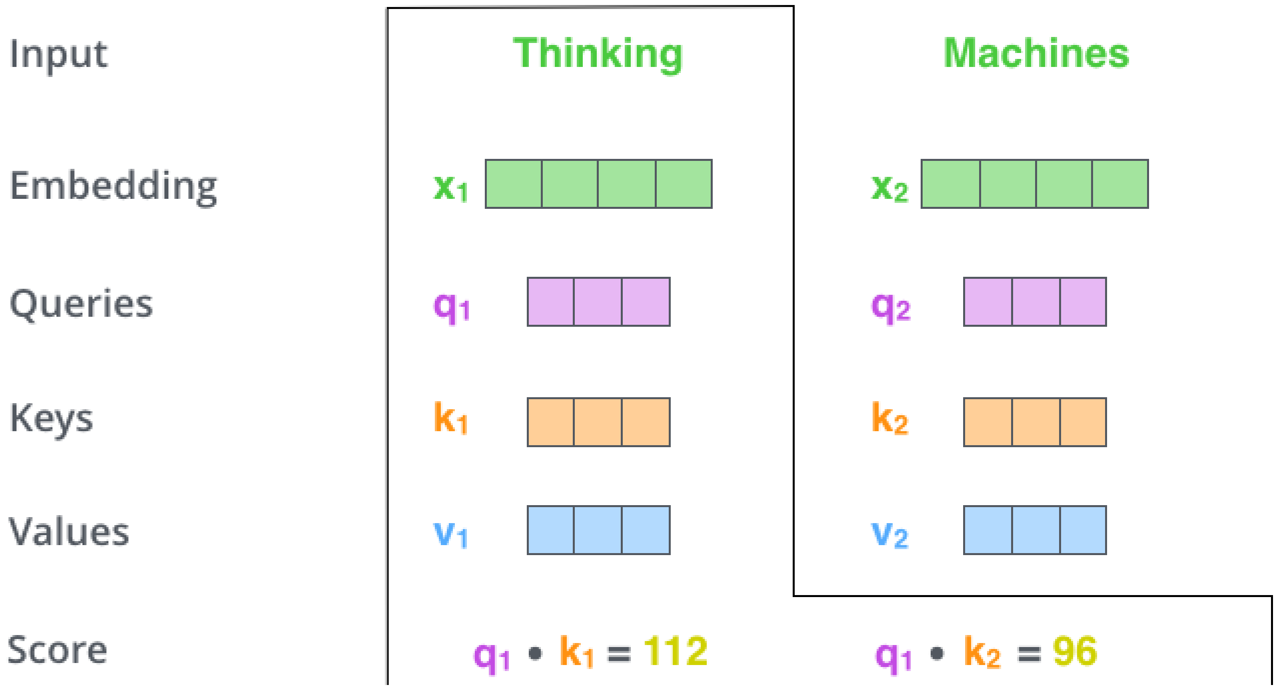
- Multiplying X_1 by the weight matrix W^Q produces q_1 (similar for v_1 and k_1).



Self-Attention Layer of Encoder

- The second step: we need to score each word of the input sentence against this word.

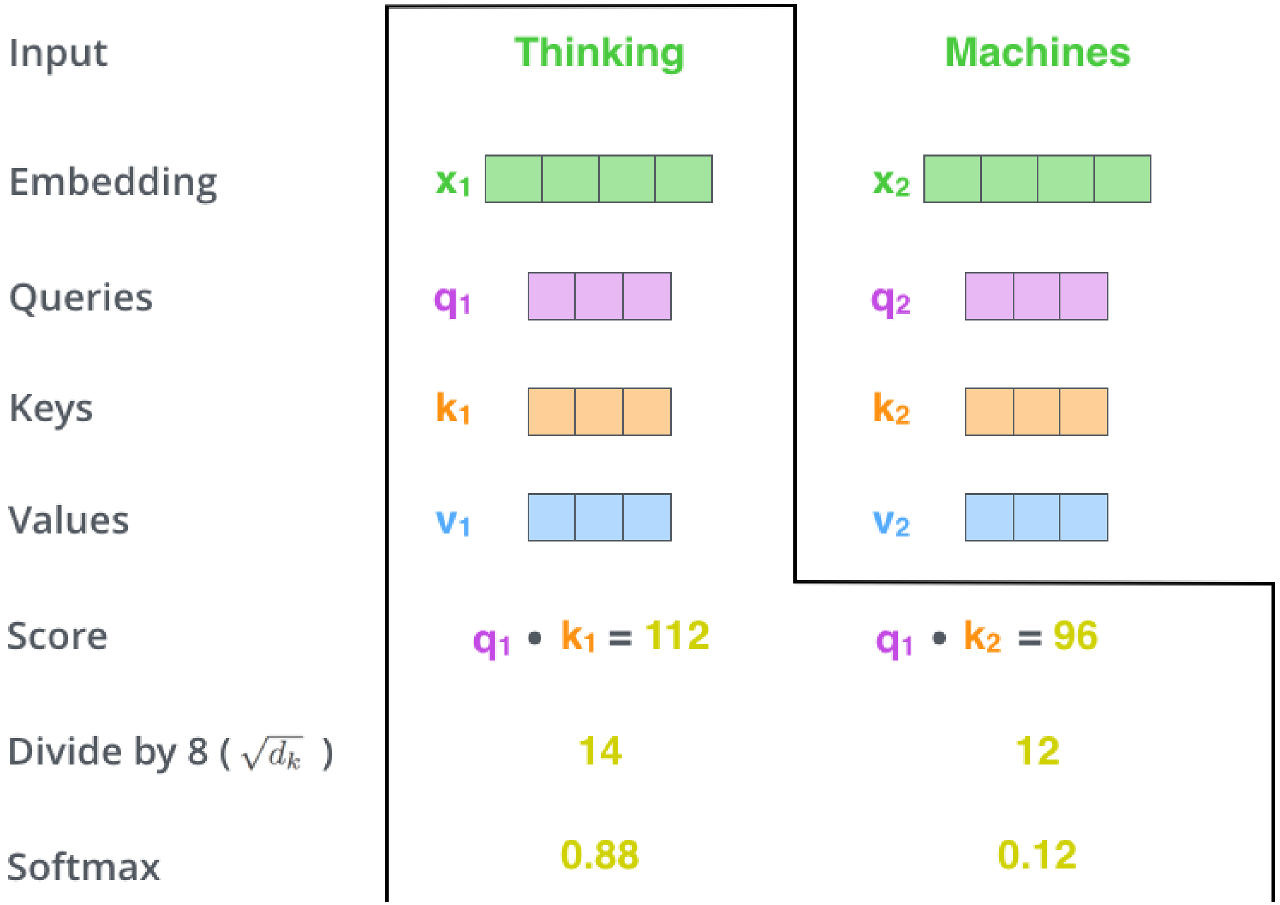
- The score determines how much focus to place on other words as we encode a word at a certain position.



Self-Attention Layer of Encoder

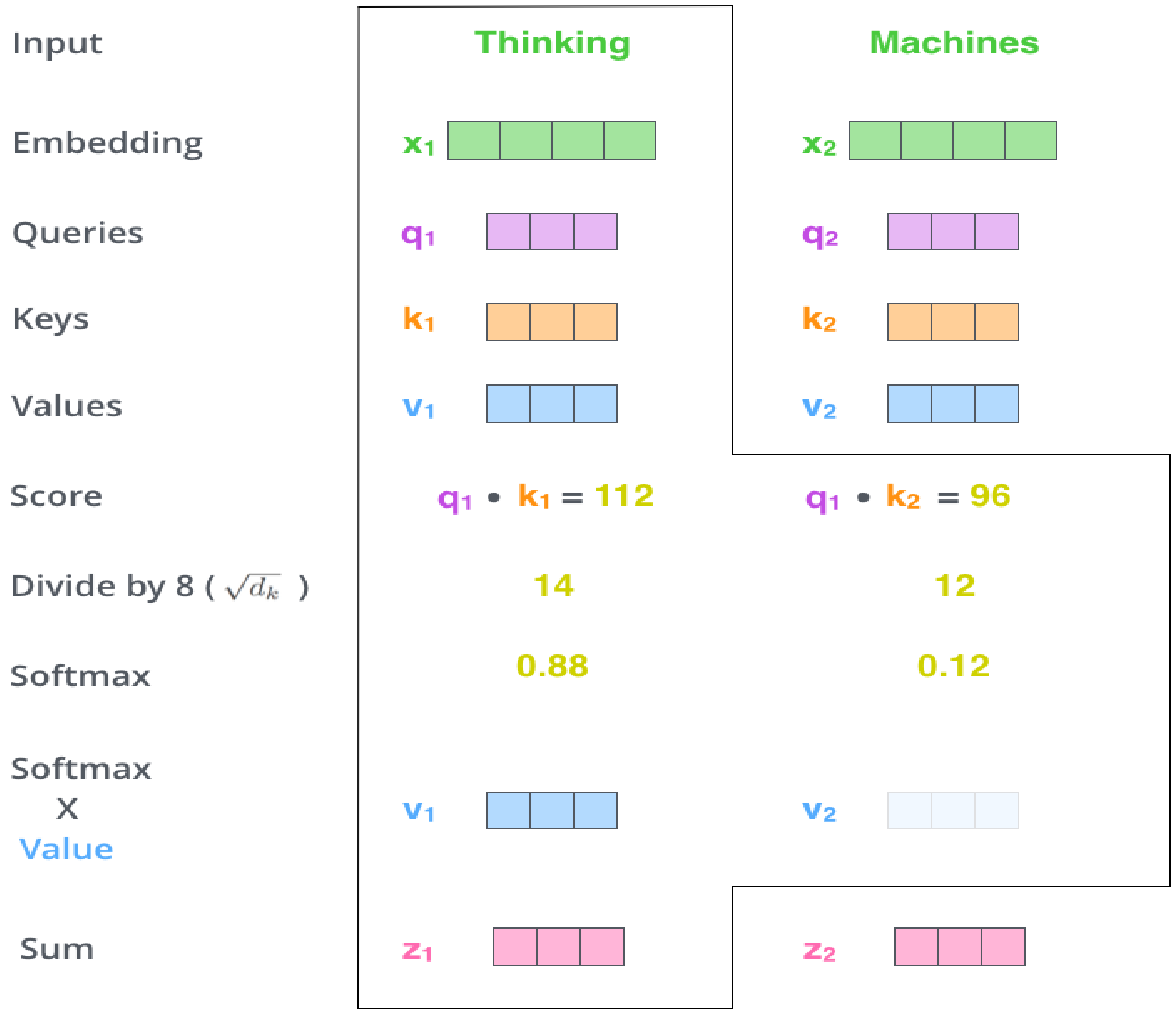
- The third step is to divide the scores by the square root of the dimension of the key vectors. This leads to having more stable gradients.

- The fourth step is to pass the result through a Softmax operation.



Self-Attention Layer of Encoder

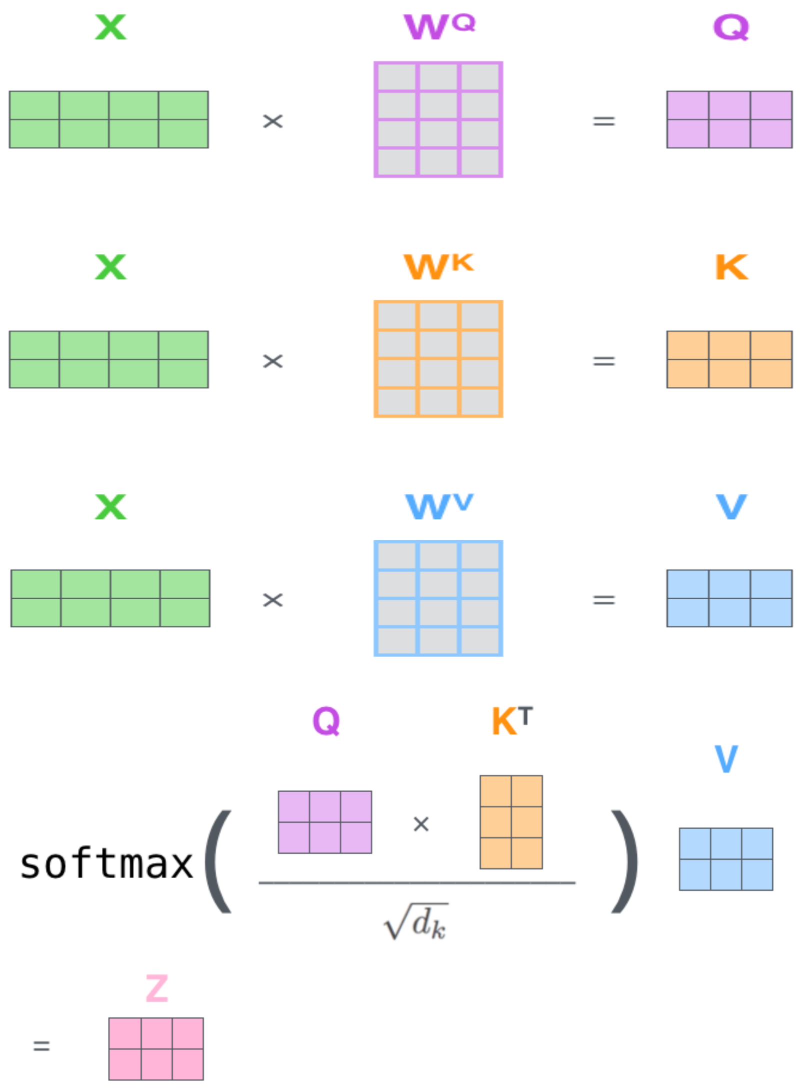
- The fifth step is to multiply each value vector v_i by the Softmax score.
- The intuition here is to keep intact values of the word(s) we want to focus on, and dropout irrelevant words.
- The sixth step is to sum up the weighted value vectors.



Recap: Self-Attention in Encoder

- The first step is to calculate the Query, Key, and Value matrices.

- Steps two through six, we calculate the outputs of the self-attention layer.



A Family of Attention Score Functions

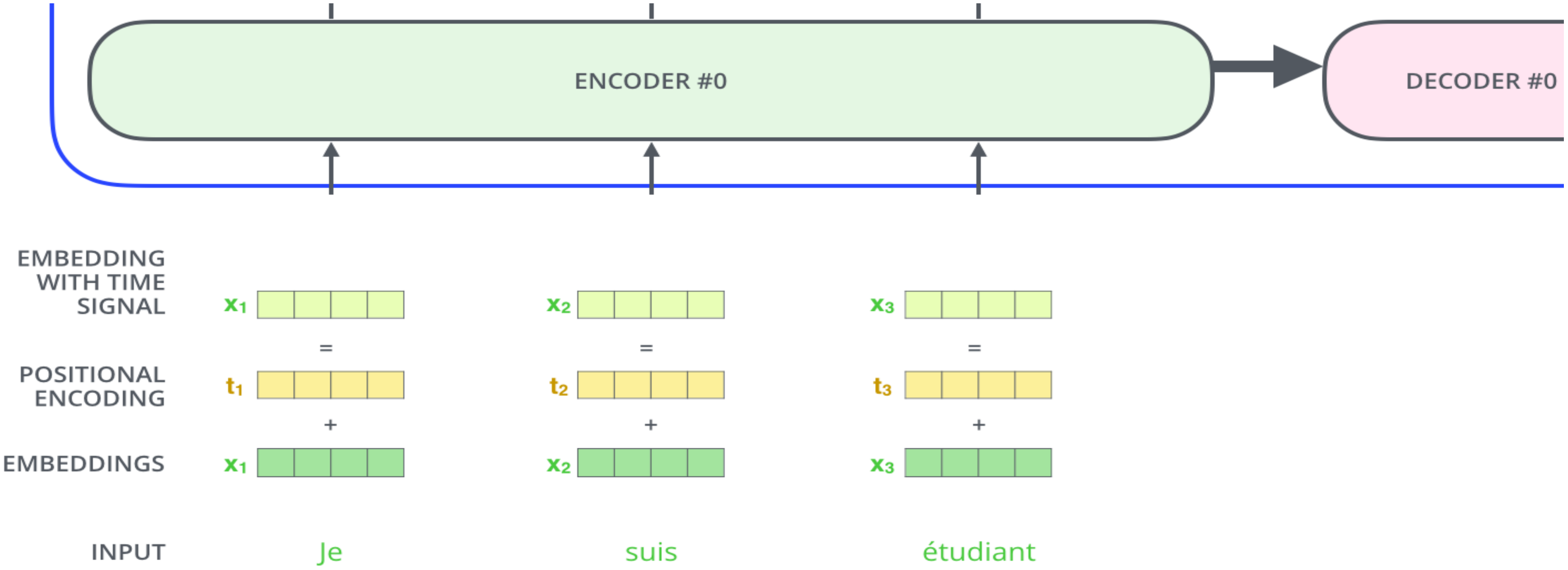
Summary

Below is a summary table of several popular attention mechanisms and corresponding alignment score functions:

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

Positional Encoding

- To account for the order of the words in the input or output sequence, the transformer adds a vector to each input embedding to determine the position of each word.

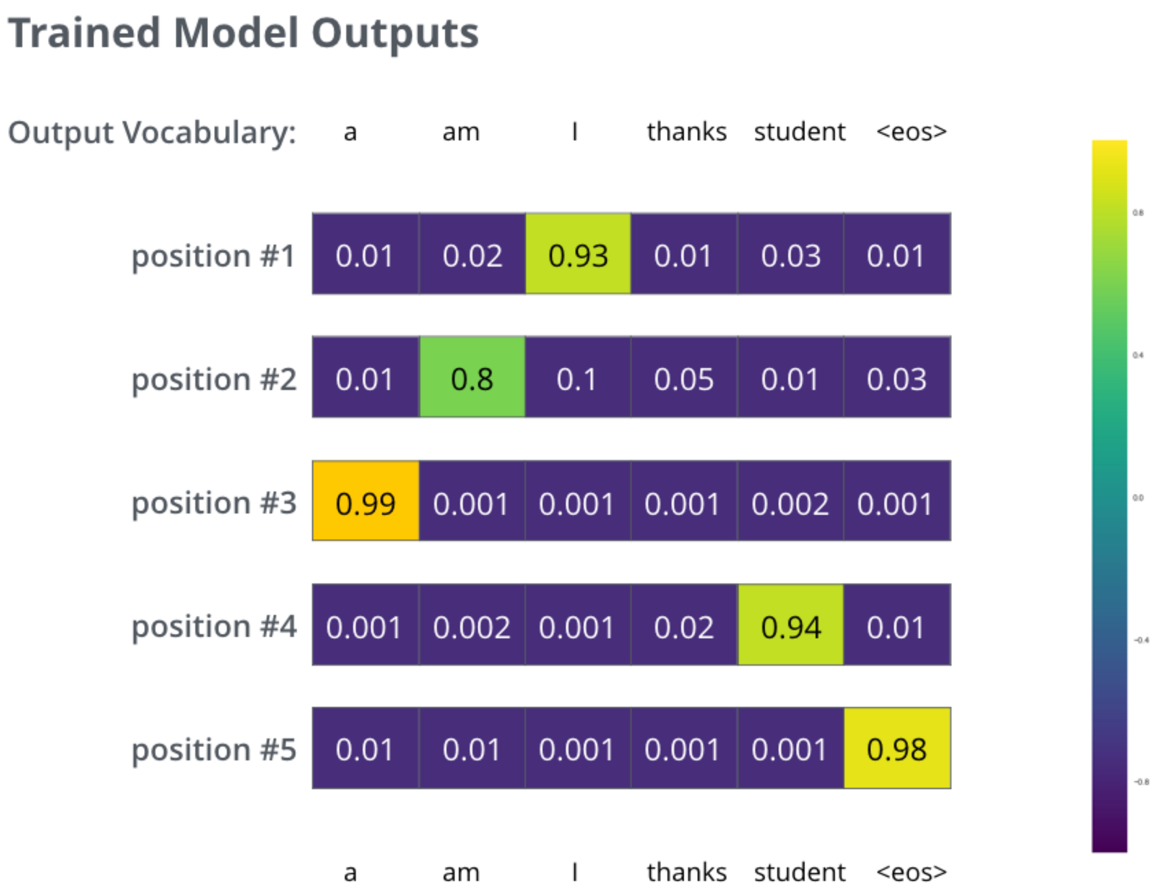


The intuition Behind Positional Encoding

- Adding these values to the embeddings provides meaningful distances between the embedding vectors once they’re projected into Q/K/V vectors and during dot-product attention.

Positional Encoding

- In the positional encodings $U \in R^{L \times d}$, L is length of sequence and d is vocabulary size, and the i-th row U_i corresponds to the i-th word in the input/output sequence.
- Example of French to English translation:
    - input: “je suis étudiant”.
    - output: “I am a student”.
    - Token <eos> means the end of the sentence.



Outline

- Recurrent Neural Networks (RNN)
- Long-Short Term Memory (LSTM)
- Transformer
  - Encoder
  - Multi-head Attention
  - Decoder

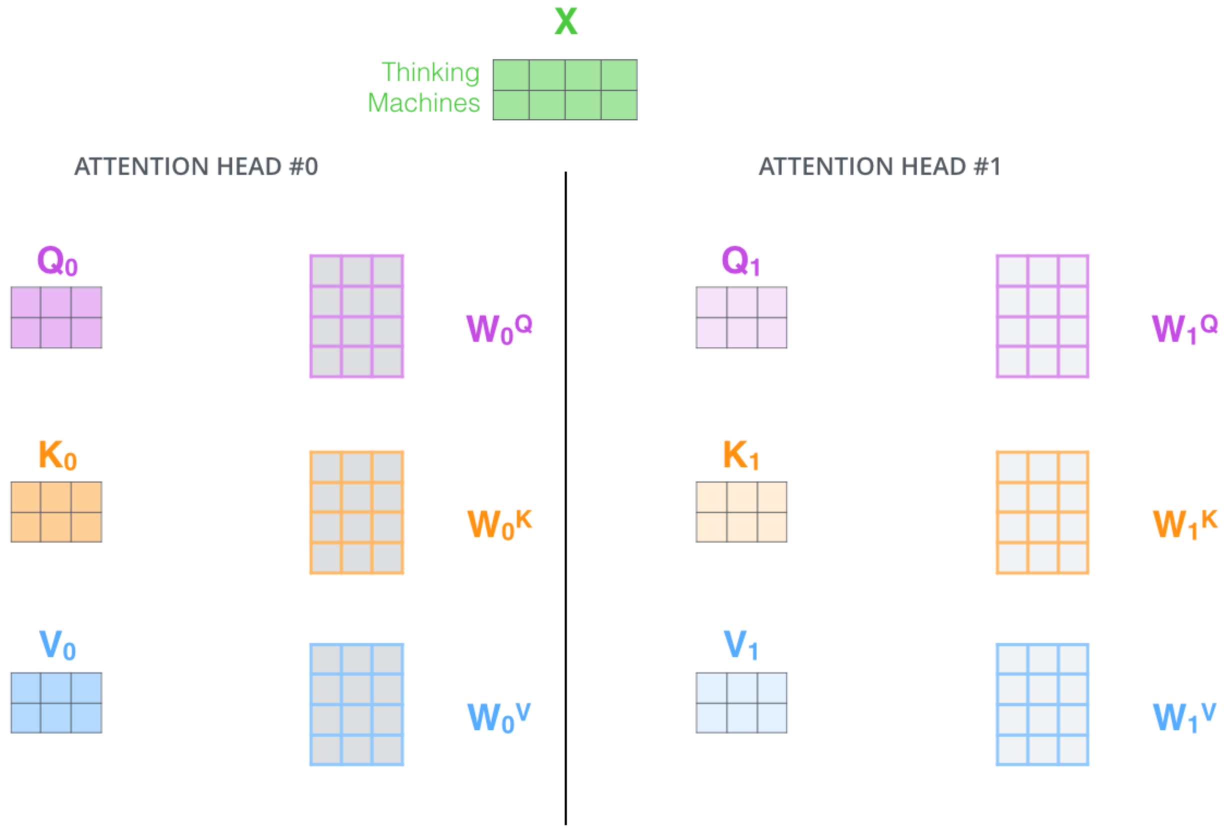
Benefits of Multi-head Attention

- It expands the model's ability to focus on different positions.
  - In a single attention layer, output vector z contains a little bit of every other encoding, but it could be still dominated by the actual word itself.
  - If we're translating a sentence like "The animal didn't cross the street because it was too tired", we would want to know which word "it" refers to.

- It gives the attention layer multiple "representation subspaces".
  - With multi-headed attention we have multiple sets of Query/Key/Value weight matrices, and different representation subspace.

Multi-head Self Attention

- If we do the same self-attention calculation we outlined before, just eight different times with different weight matrices, we end up with eight different Z matrices or eight different representation subspace.



Multi-head Self Attention

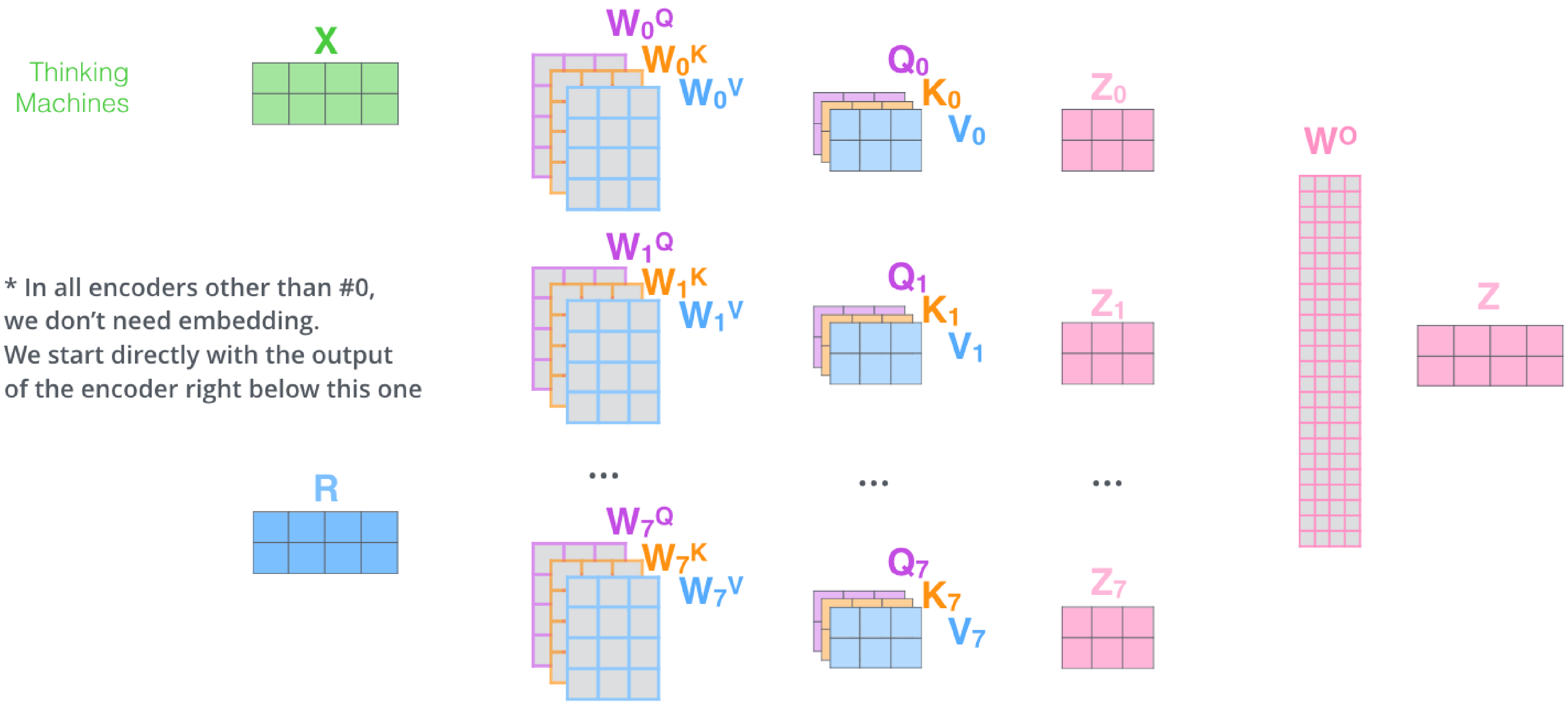
1) This is our input sentence*

2) We embed each word*

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

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

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



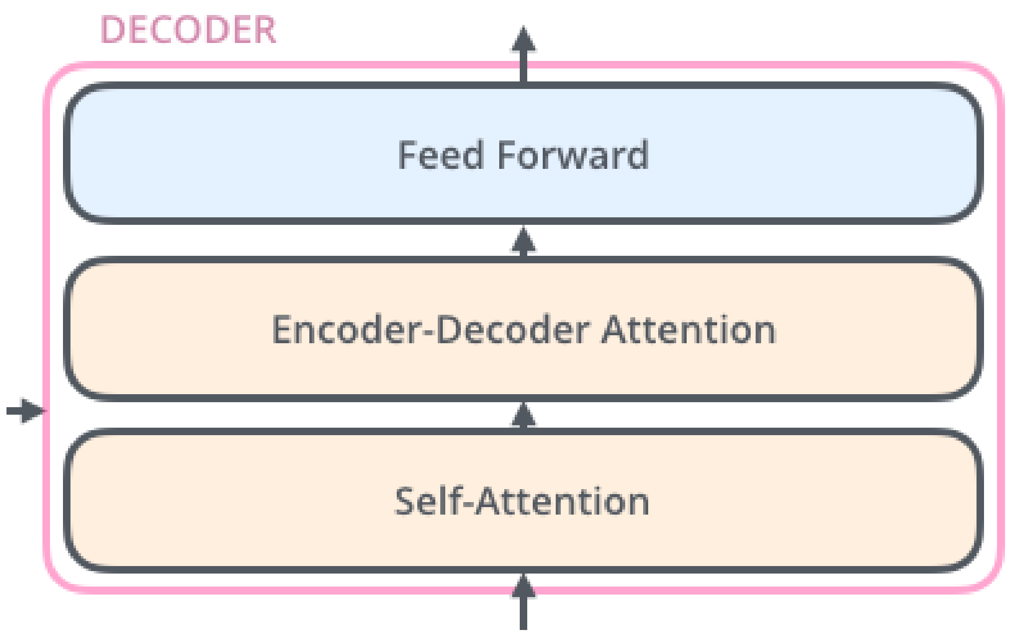
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Outline

- Recurrent Neural Networks (RNN)
- Long-Short Term Memory (LSTM)
- Transformer
  - Encoder
  - Multi-head Attention
  - Decoder

Decoder of a transformer

- The decoder has both those layers, but between them is an another attention layer (Encoder-Decoder Attention Layer) that helps the decoder focus on relevant parts of the input sentence.

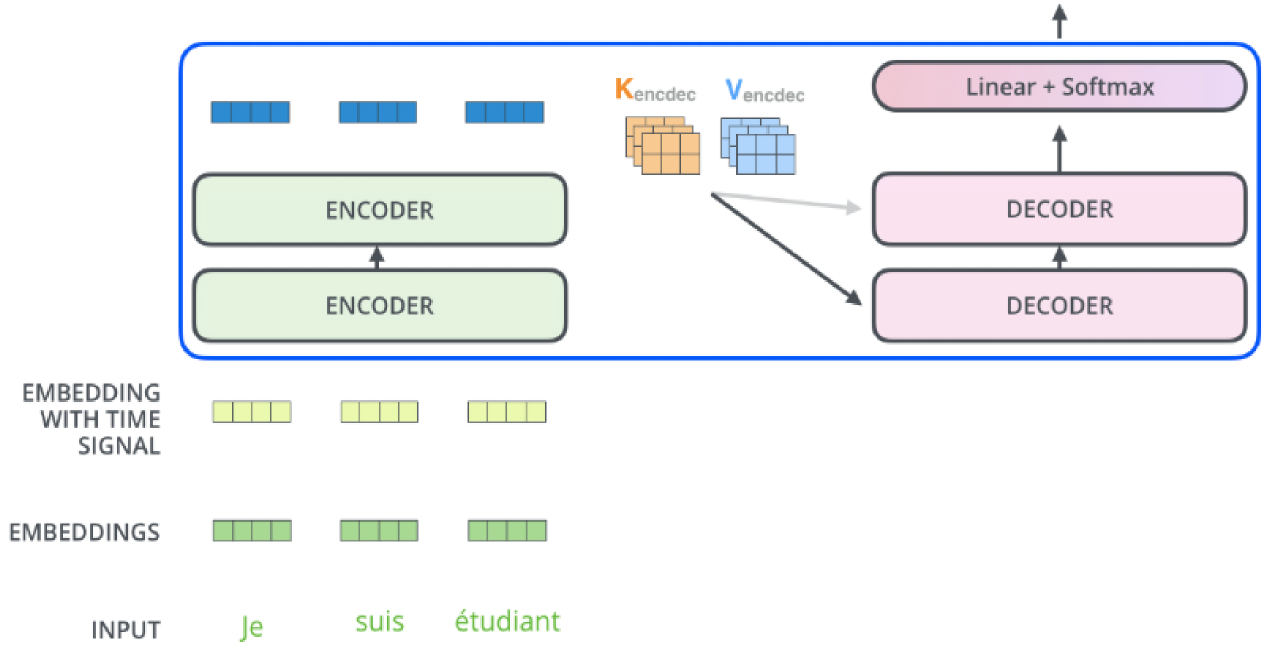


- The self-attention layer and feed forward network of a decoder work similarly to that of encoder.

Encoder-Decoder Attention Layer of Decoder

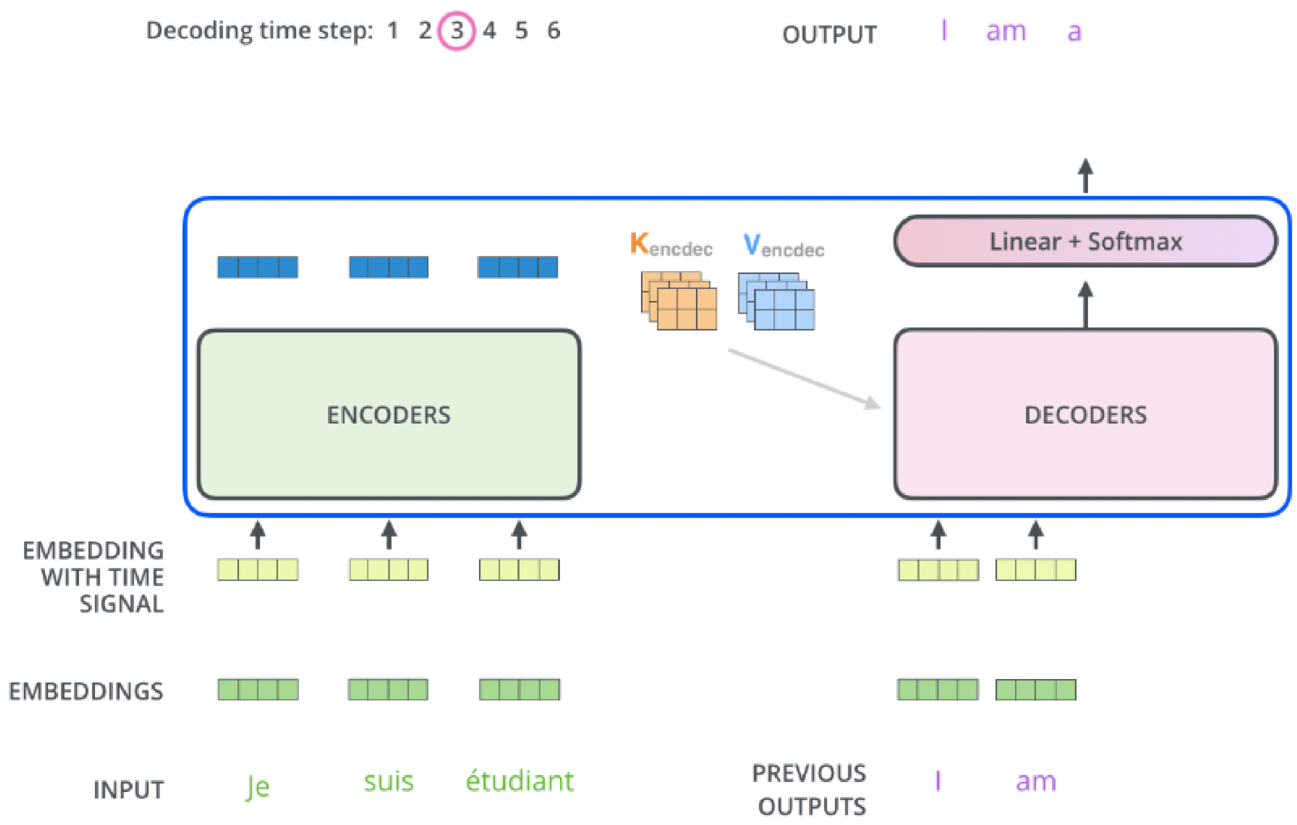
- The output of the top encoder is transformed into a set of attention vectors K and V.

- Just like we did with the encoder inputs, we embed and add positional encoding to those decoder inputs to indicate the position of each word.



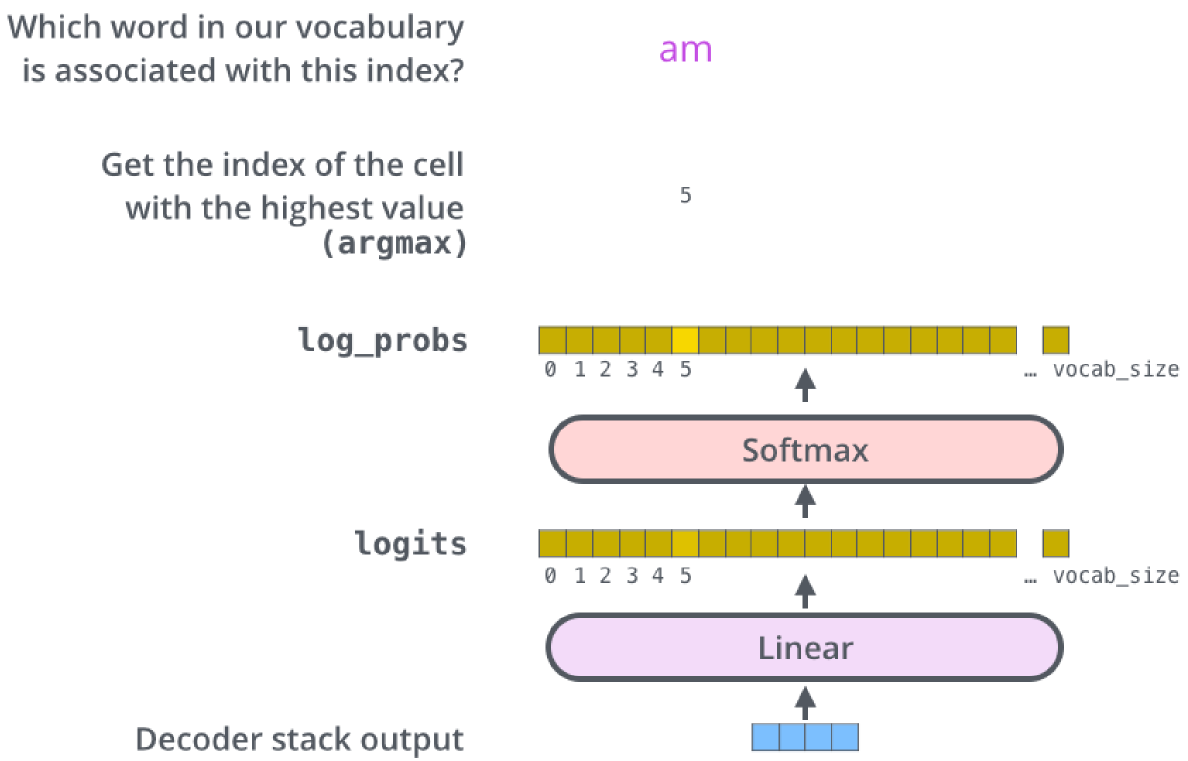
Encoder-Decoder Attention Layer of Decoder

- These output vectors are used by each decoder in its encoder-decoder attention layer to help the decoder focus on appropriate places in the input sequence.



Linear and Softmax Layer of Decoder

- The Linear layer projects the output of decoders to a logits vector.
- The Softmax layer then turns those scores into probabilities.



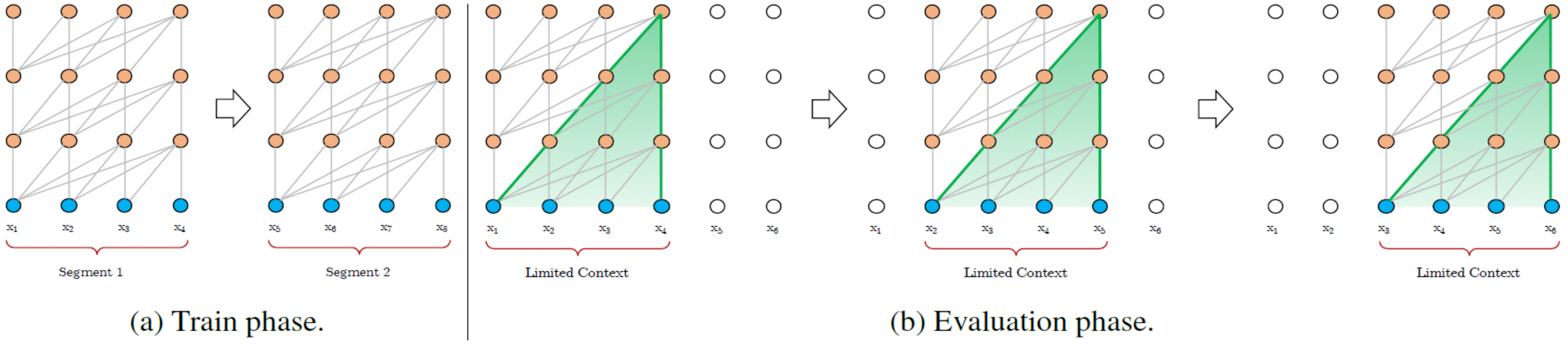
Useful Tutorial

- How Transformers Work
- http://jalammar.github.io/illustrated-transformer/
- Attention? Attention!
- http://nlp.seas.harvard.edu/2018/04/03/attention.html

Transformer XL: Attentive Language Models Beyond a Fixed-Length Context

Vanilla model

- The central problem is how to effectively encode an arbitrarily long context into a fixed size representation.
- Vanilla model splits the entire corpus into shorter segments of manageable sizes, and ignore all contextual information from previous segments.



(a) Train phase. (b) Evaluation phase.

Figure 1: Illustration of the vanilla model with a segment length 4.

Major limitations of vanilla model

- The largest possible dependency length is upper bounded by the segment length.

- The model lacks necessary contextual information needed to well predict the first few symbols.

- Simply chunking a sequence into fixed-length segments lead to inefficient optimization and inferior performance.

Transformer-XL: Segment-Level Recurrence with State Reuse

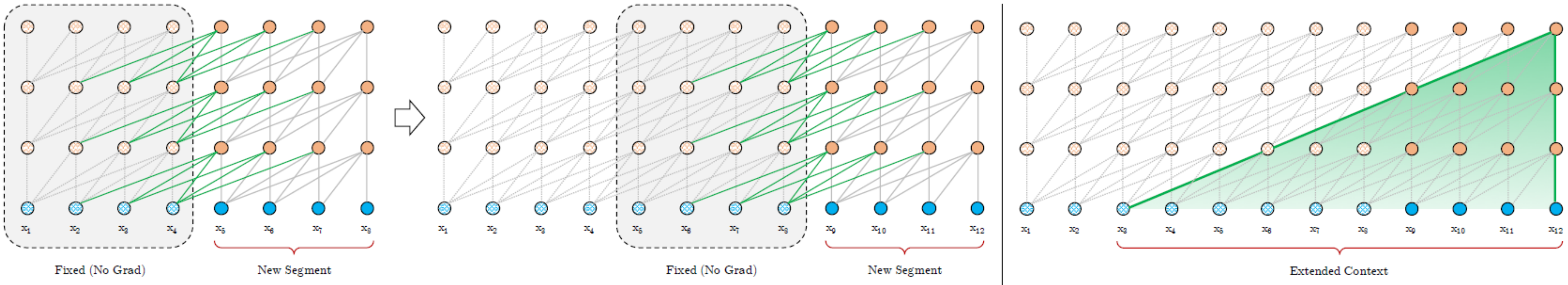
- To address these limitations, they take the advantage of the recurrence mechanism and apply it to the Transformer architecture.

- The goal of this mechanism is to enable long-term dependencies with the information from previous segments.

- Transformer-XL processes the first segment of tokens but keeps the output of the hidden layers. When the next segment is processed, each hidden layer receives two inputs.

Transformer-XL: Segment-Level Recurrence with State Reuse

- Two inputs:
  - The output of the previous hidden layer of that segment.
  - The output of the previous hidden layer from the previous segment.
- Advantages: Incorporate longer dependencies and speed up the evaluation.



(a) Training phase.

(b) Evaluation phase.

Transformer-XL: Segment-Level Recurrence with State Reuse

- Let the two consecutive segments of length L be $s_\tau = [x_{\tau,1}, \ \ldots, x_{\tau,L}]$ and $s_{\tau+1} = [x_{\tau+1,1}, \ \ldots, x_{\tau+1,L}]$, respectively.
- Denote the n-th layer hidden state sequence produced for the τ -th segment s_τ as $h_\tau^n \in R^{L\times d}$, where d is the hidden dimension.
- The n-th layer hidden state for segment $s_{\tau+1}$ is produced as follows,

$$
\begin{aligned}
\widetilde{\mathbf{h}}_{\tau+1}^{n-1} &= \left[\mathrm{SG}(\mathbf{h}_{\tau}^{n-1}) \circ \mathbf{h}_{\tau+1}^{n-1}\right], && \text{(extended context)} \\
\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n} &= \mathbf{h}_{\tau+1}^{n-1}\mathbf{W}_{q}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{k}^{\top}, \widetilde{\mathbf{h}}_{\tau+1}^{n-1}\mathbf{W}_{v}^{\top}, && \text{(query, key, value vectors)} \\
\mathbf{h}_{\tau+1}^{n} &= \text{Transformer-Layer}\left(\mathbf{q}_{\tau+1}^{n}, \mathbf{k}_{\tau+1}^{n}, \mathbf{v}_{\tau+1}^{n}\right). && \text{(self-attention + feed-forward)}
\end{aligned}
$$

, where $\mathrm{SG}(\cdot)$ stands for stop-gradient.

Transformer-XL: Relative Positional Encodings

- Challenge to reuse the hidden states - keep the positional information coherent.
- In Transformer, the information of sequence order is provided by a set of positional encodings $U \in R^{L_{max} \times d}$, where the i-th row U_i corresponds to the i-th absolute position within a segment and L_{max} is the maximum possible length to be modeled.

$$\mathbf{h}_{\tau+1} = f(\mathbf{h}_\tau, \mathbf{E}_{\mathbf{s}_{\tau+1}} + \mathbf{U}_{1:L}) \quad \text{and} \quad \mathbf{h}_\tau = f(\mathbf{h}_{\tau-1}, \mathbf{E}_{\mathbf{s}_\tau} + \mathbf{U}_{1:L}),$$

- Notice that , both $E_{s_{\tau+1}}$ and E_{s_τ} are associated with the same positional encoding $U_{1:L}$.

Transformer-XL: Relative Positional Encodings

- To avoid this failure mode, they encode relative positional information in the hidden states.
  - Technically, it expands the simple multiplication of the Attention Head's $Score(q_i \cdot k_j)$ to include the four parts mentioned previously.

$$\mathbf{A}_{i,j}^{\mathrm{abs}} = q_i^\top k_j = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(b)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{E}_{x_j}}_{(c)} + \underbrace{\mathbf{U}_i^\top \mathbf{W}_q^\top \mathbf{W}_k \mathbf{U}_j}_{(d)}.$$

$$\mathbf{A}_{i,j}^{\mathrm{rel}} = \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(a)} + \underbrace{\mathbf{E}_{x_i}^\top \mathbf{W}_q^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(b)} + \underbrace{u^\top \mathbf{W}_{k,E} \mathbf{E}_{x_j}}_{(c)} + \underbrace{v^\top \mathbf{W}_{k,R} \mathbf{R}_{i-j}}_{(d)}.$$

  - Relative positional encoding $R \in R^{L_{max} \times d}$ substitutes the positional encoding $U \in R^{L_{max} \times d}$, since we only need to know the relative distance of two words rather than the real order when calculating the self-attention.

Transformer-XL: Overall

- Equipping the recurrence mechanism with this relative positional embedding,

$$
\begin{aligned}
\text{For } n = 1, \ldots, N: \quad \widetilde{\mathbf{h}}_\tau^{n-1} &= \left[\mathrm{SG}(\mathbf{m}_\tau^{n-1}) \circ \mathbf{h}_\tau^{n-1}\right] \\
\mathbf{q}_\tau^n, \mathbf{k}_\tau^n, \mathbf{v}_\tau^n &= \mathbf{h}_\tau^{n-1} {\mathbf{W}_q^n}^\top, \widetilde{\mathbf{h}}_\tau^{n-1} {\mathbf{W}_{k,E}^n}^\top, \widetilde{\mathbf{h}}_\tau^{n-1} {\mathbf{W}_v^n}^\top \\
\mathbf{A}_{\tau,i,j}^n &= {\mathbf{q}_{\tau,i}^n}^\top \mathbf{k}_{\tau,j}^n + {\mathbf{q}_{\tau,i}^n}^\top \mathbf{W}_{k,R}^n \mathbf{R}_{i-j} + u^\top \mathbf{k}_{\tau,j} + v^\top \mathbf{W}_{k,R}^n \mathbf{R}_{i-j} \\
\mathbf{a}_\tau^n &= \text{Masked-Softmax}(\mathbf{A}_\tau^n)\mathbf{v}_\tau^n \\
\mathbf{o}_\tau^n &= \mathrm{LayerNorm}(\mathrm{Linear}(\mathbf{a}_\tau^n) + \mathbf{h}_\tau^{n-1}) \\
\mathbf{h}_\tau^n &= \text{Positionwise-Feed-Forward}(\mathbf{o}_\tau^n)
\end{aligned}
$$

Experimental Results

Model	#Param	PPL
Grave et al. (2016b) - LSTM	-	48.7
Bai et al. (2018) - TCN	-	45.2
Dauphin et al. (2016) - GCNN-8	-	44.9
Grave et al. (2016b) - LSTM + Neural cache	-	40.8
Dauphin et al. (2016) - GCNN-14	-	37.2
Merity et al. (2018) - QRNN	151M	33.0
Rae et al. (2018) - Hebbian + Cache	-	29.9
Ours - Transformer-XL Standard	151M	**24.0**
Baevski and Auli (2018) - Adaptive Input$^\diamond$	247M	20.5
Ours - Transformer-XL Large	257M	**18.3**

Table 1: Comparison with state-of-the-art results on WikiText-103. $^\diamond$ indicates contemporary work.

Model	#Param	bpc
Ha et al. (2016) - LN HyperNetworks	27M	1.34
Chung et al. (2016) - LN HM-LSTM	35M	1.32
Zilly et al. (2016) - RHN	46M	1.27
Mujika et al. (2017) - FS-LSTM-4	47M	1.25
Krause et al. (2016) - Large mLSTM	46M	1.24
Knol (2017) - cmix v13	-	1.23
Al-Rfou et al. (2018) - 12L Transformer	44M	1.11
Ours - 12L Transformer-XL	41M	**1.06**
Al-Rfou et al. (2018) - 64L Transformer	235M	1.06
Ours - 18L Transformer-XL	88M	1.03
Ours - 24L Transformer-XL	277M	**0.99**

Table 2: Comparison with state-of-the-art results on enwik8.

Model	#Param	bpc
Cooijmans et al. (2016) - BN-LSTM	-	1.36
Chung et al. (2016) - LN HM-LSTM	35M	1.29
Zilly et al. (2016) - RHN	45M	1.27
Krause et al. (2016) - Large mLSTM	45M	1.27
Al-Rfou et al. (2018) - 12L Transformer	44M	1.18
Al-Rfou et al. (2018) - 64L Transformer	235M	1.13
Ours - 24L Transformer-XL	277M	**1.08**

Table 3: Comparison with state-of-the-art results on text8.

Model	#Param	PPL
Shazeer et al. (2014) - Sparse Non-Negative	33B	52.9
Chelba et al. (2013) - RNN-1024 + 9 Gram	20B	51.3
Kuchaiev and Ginsburg (2017) - G-LSTM-2	-	36.0
Dauphin et al. (2016) - GCNN-14 bottleneck	-	31.9
Jozefowicz et al. (2016) - LSTM	1.8B	30.6
Jozefowicz et al. (2016) - LSTM + CNN Input	1.04B	30.0
Shazeer et al. (2017) - Low-Budget MoE	~5B	34.1
Shazeer et al. (2017) - High-Budget MoE	~5B	28.0
Shazeer et al. (2018) - Mesh Tensorflow	4.9B	24.0
Baevski and Auli (2018) - Adaptive Input$^\diamond$	0.46B	24.1
Baevski and Auli (2018) - Adaptive Input$^\diamond$	1.0B	23.7
Ours - Transformer-XL Base	0.46B	23.5
Ours - Transformer-XL Large	0.8B	**21.8**

Table 4: Comparison with state-of-the-art results on One Billion Word. $^\diamond$ indicates contemporary work.