Advanced Pre-training language models a brief introduction

Xiachong Feng

Outline

1. Encoder-Decoder
2. Attention
3. Transformer: 《Attention is all you need》
4. Word embedding and pre-trained model
5. ELMo ：《Deep contextualized word representations》
6. OpenAI GPT: 《Improving Language Understanding by Generative Pre-Training》
7. BERT: 《BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding》
8. Conclusion

Outline

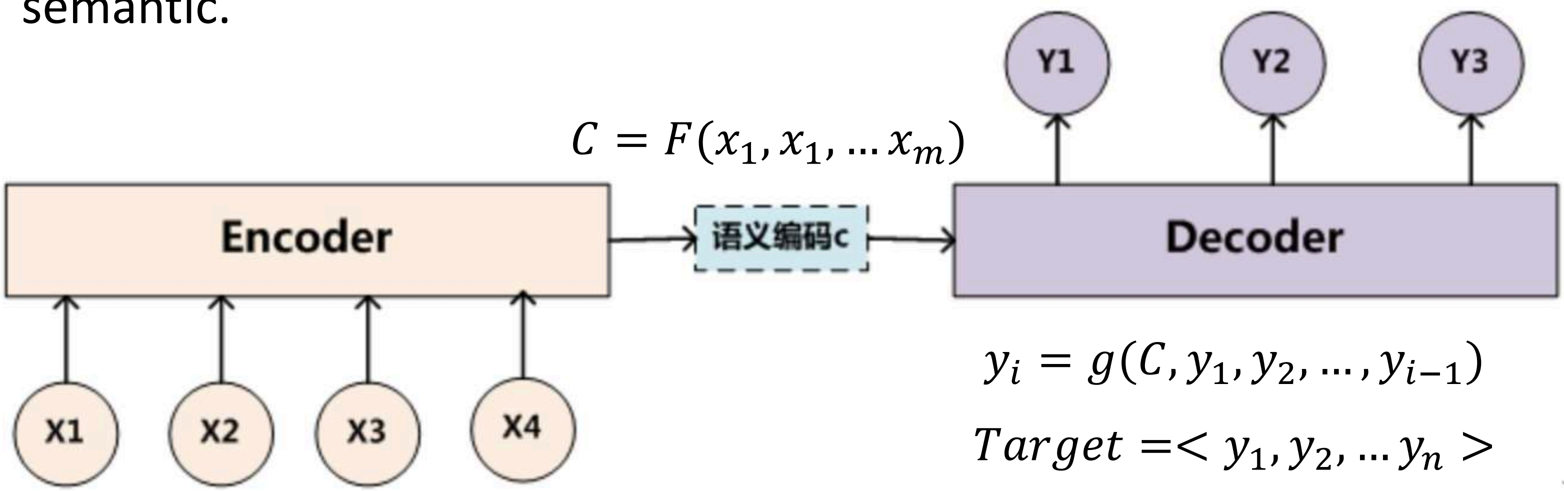
1. Encoder-Decoder
2. Attention
3. Transformer: 《Attention is all you need》
4. Word embedding and pre-trained model
5. ELMo: 《Deep contextualized word representations》
6. OpenAI GPT: 《Improving Language Understanding by Generative Pre-Training》
7. BERT: 《BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding》
8. Conclusion

Encoder-Decoder

- When the sentence is **short**, context vector may retain some important information
- When the sentence is **long**, context vector will lose some information such as semantic.

$$y_1 = f(C)$$
$$y_2 = f(C, y_1)$$
$$y_3 = f(C, y_1, y_2)$$

$$C = F(x_1, x_1, \dots x_m)$$



$$y_i = g(C, y_1, y_2, \dots, y_{i-1})$$
$$Target = < y_1, y_2, \dots y_n >$$

$$Source = < x_1, x_2, \dots x_m >$$

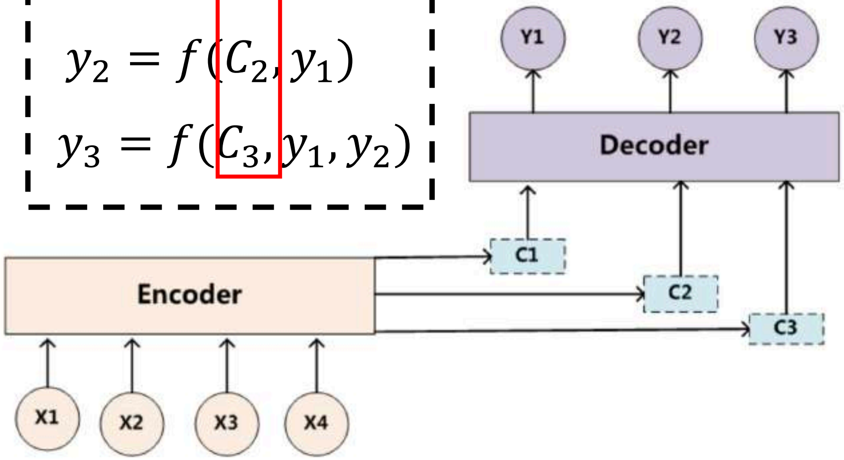
Encoder-Decoder Framework

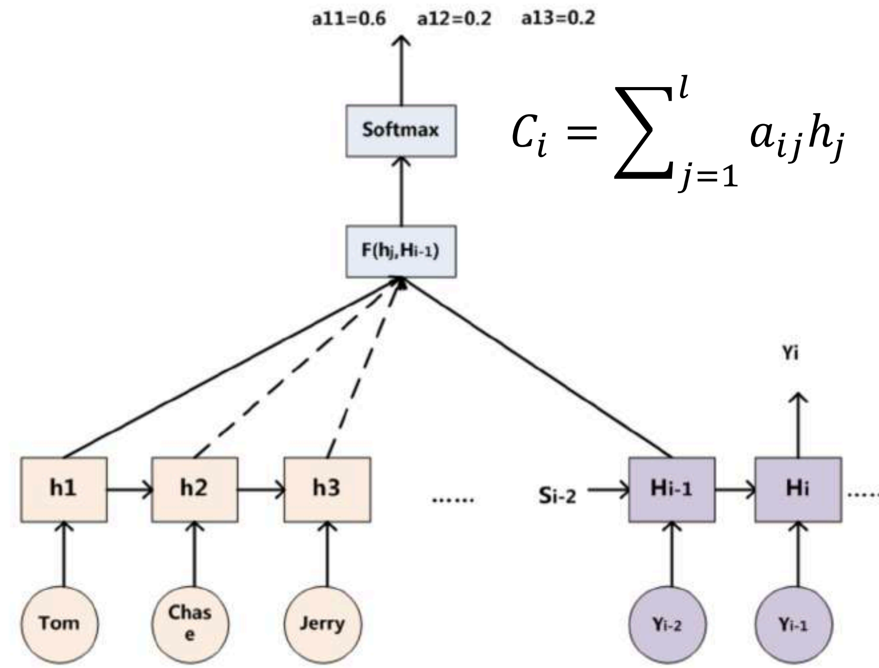
Outline

1. Encoder-Decoder
2. Attention
3. Transformer: 《Attention is all you need》
4. Word embedding and pre-trained model
5. ELMo: 《Deep contextualized word representations》
6. OpenAI GPT: 《Improving Language Understanding by Generative Pre-Training》
7. BERT: 《BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding》
8. Conclusion

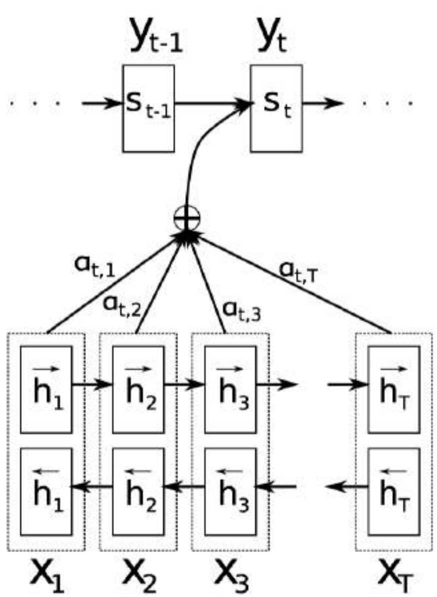
Soft-Attention

$$y_1 = f(C_1)$$
$$y_2 = f(C_2, y_1)$$
$$y_3 = f(C_3, y_1, y_2)$$



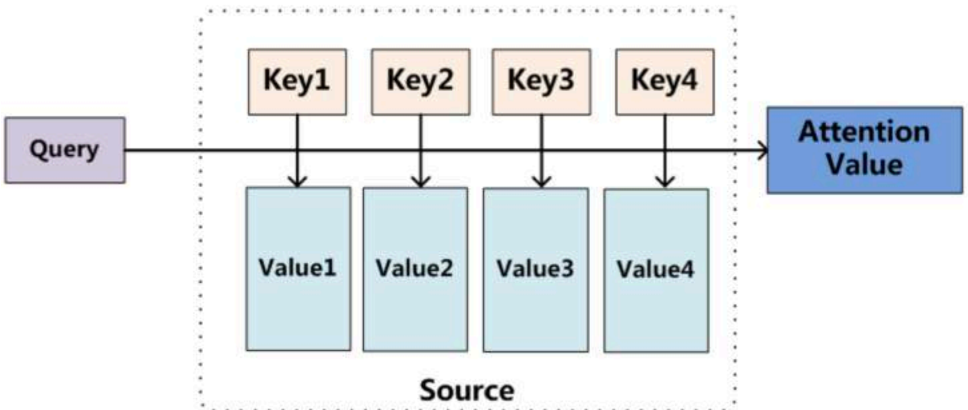


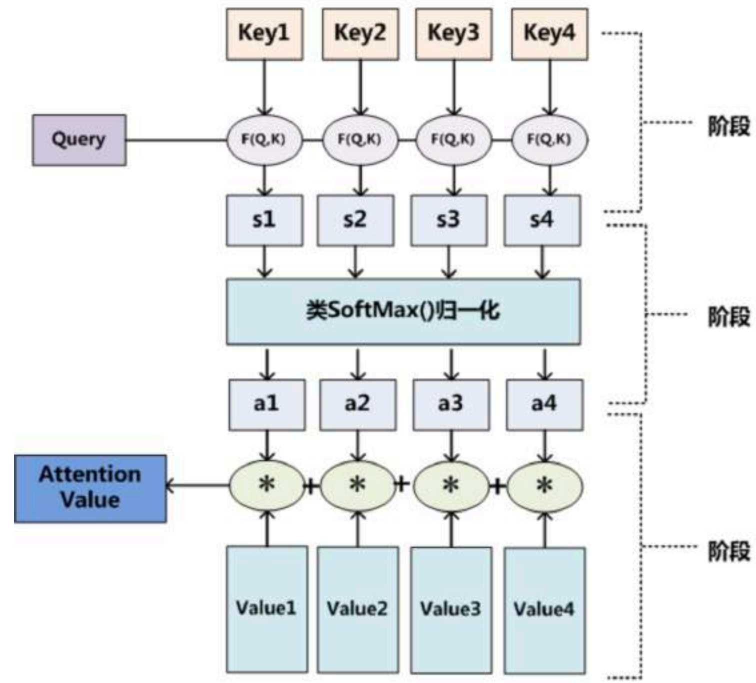
$$C_i = \sum_{j=1}^{l} a_{ij} h_j$$



$$p(y_i | y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i)$$
$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$
$$\boxed{e_{ij} = a(s_{i-1}, h_j)}$$

Core idea of Attention

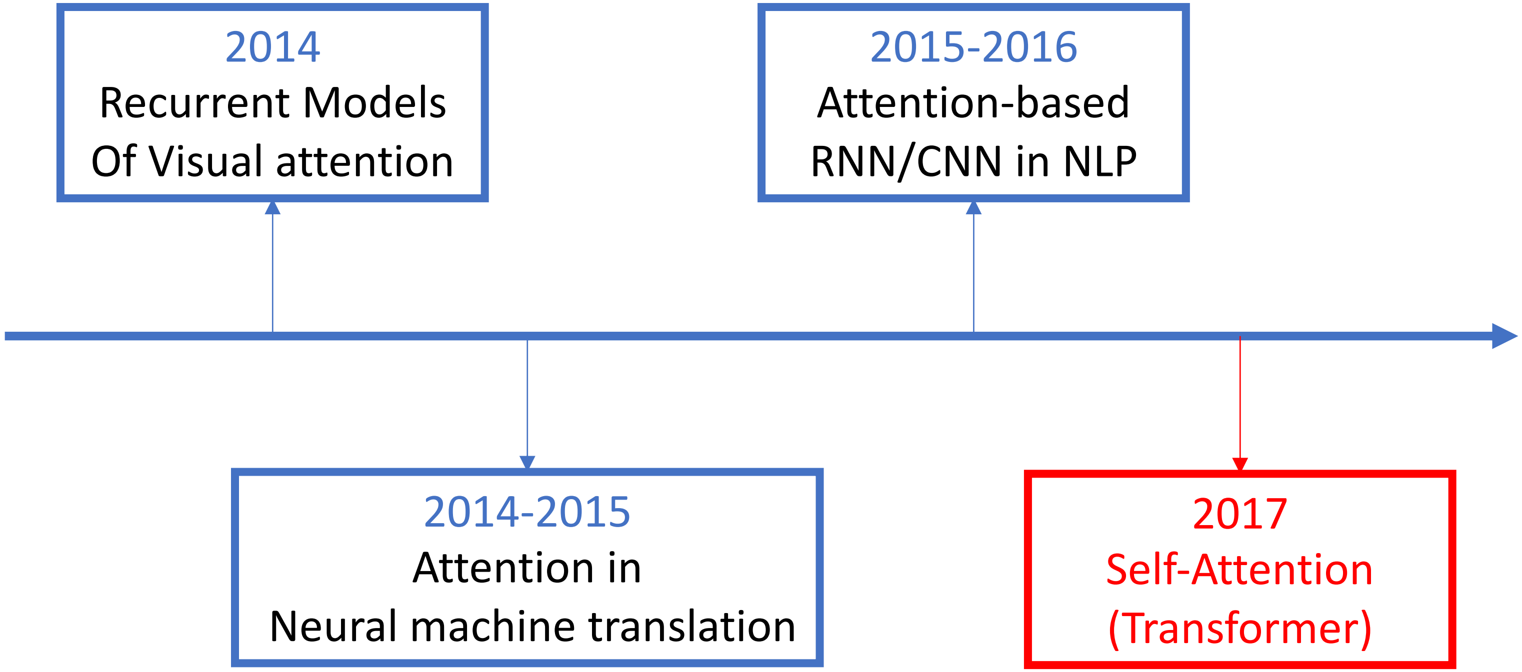




$$\boxed{Attention(Query, Source) = \sum_{i=1}^{l} Similarity(Query, Key_i) * Value_i}$$

$$\begin{cases} \color{red}{Dot}: Similarity(Query, Key_i) = Query \cdot Key_i \\ \color{red}{Cosine}: Similarity(Query, Key_i) = \dfrac{Query \cdot Key_i}{||Query|| \cdot ||Key_i||} \\ \color{red}{MLP}: Similarity(Query, Key_i) = \mathrm{MLP}(Query, Key_i) \end{cases}$$

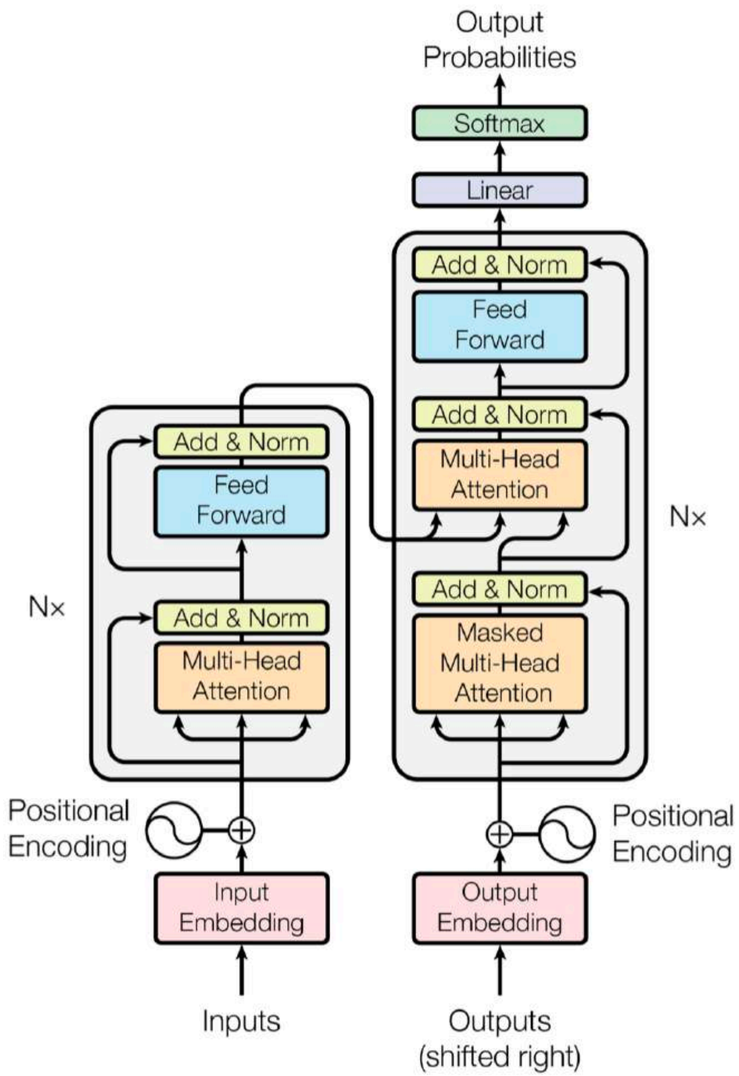
Attention Timeline



Outline

1. Encoder-Decoder
2. Attention
3. **Transformer**: 《Attention is all you need》
4. Word embedding and pre-trained model
5. ELMo: 《Deep contextualized word representations》
6. OpenAI GPT: 《Improving Language Understanding by Generative Pre-Training》
7. BERT: 《BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding》
8. Conclusion

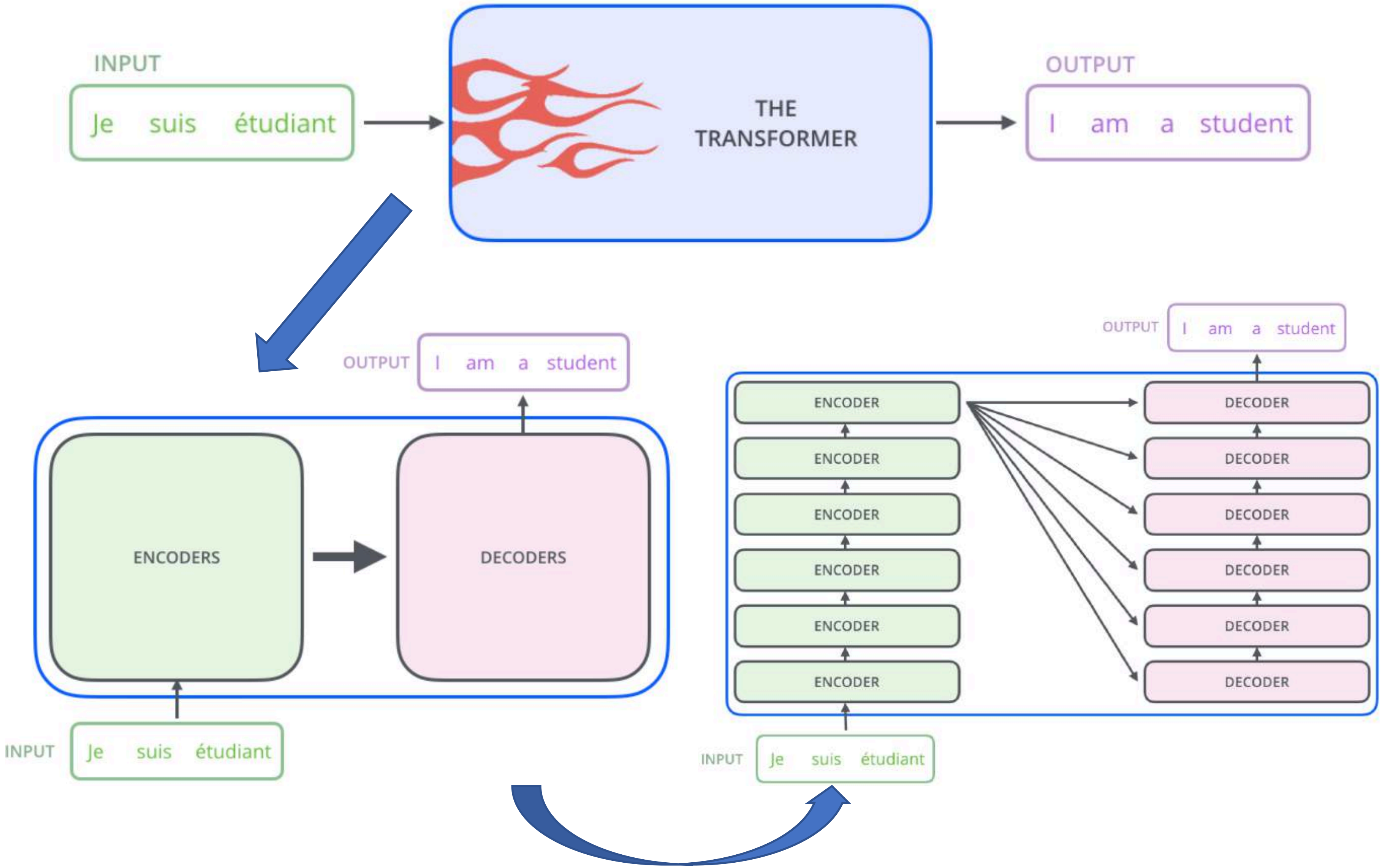
Attention is all you need



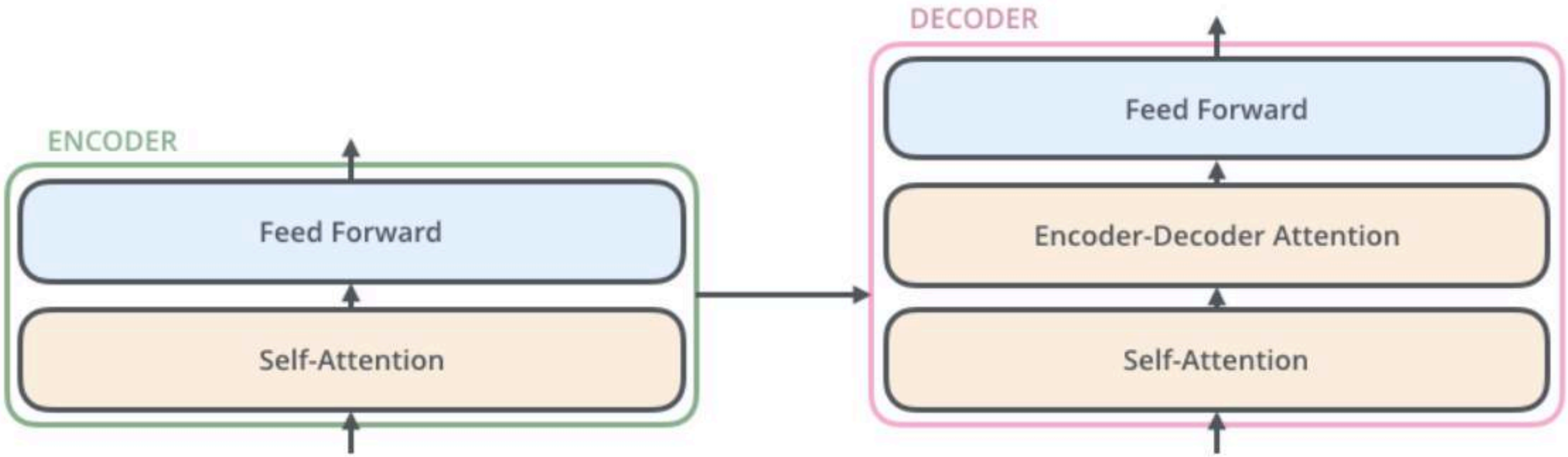
Key words

- Transformer
- Faster
- Encoder-Decoder
- Scaled Dot-Product Attention
- Multi-Head Attention
- Position encoding
- Residual connections

A High-Level Look

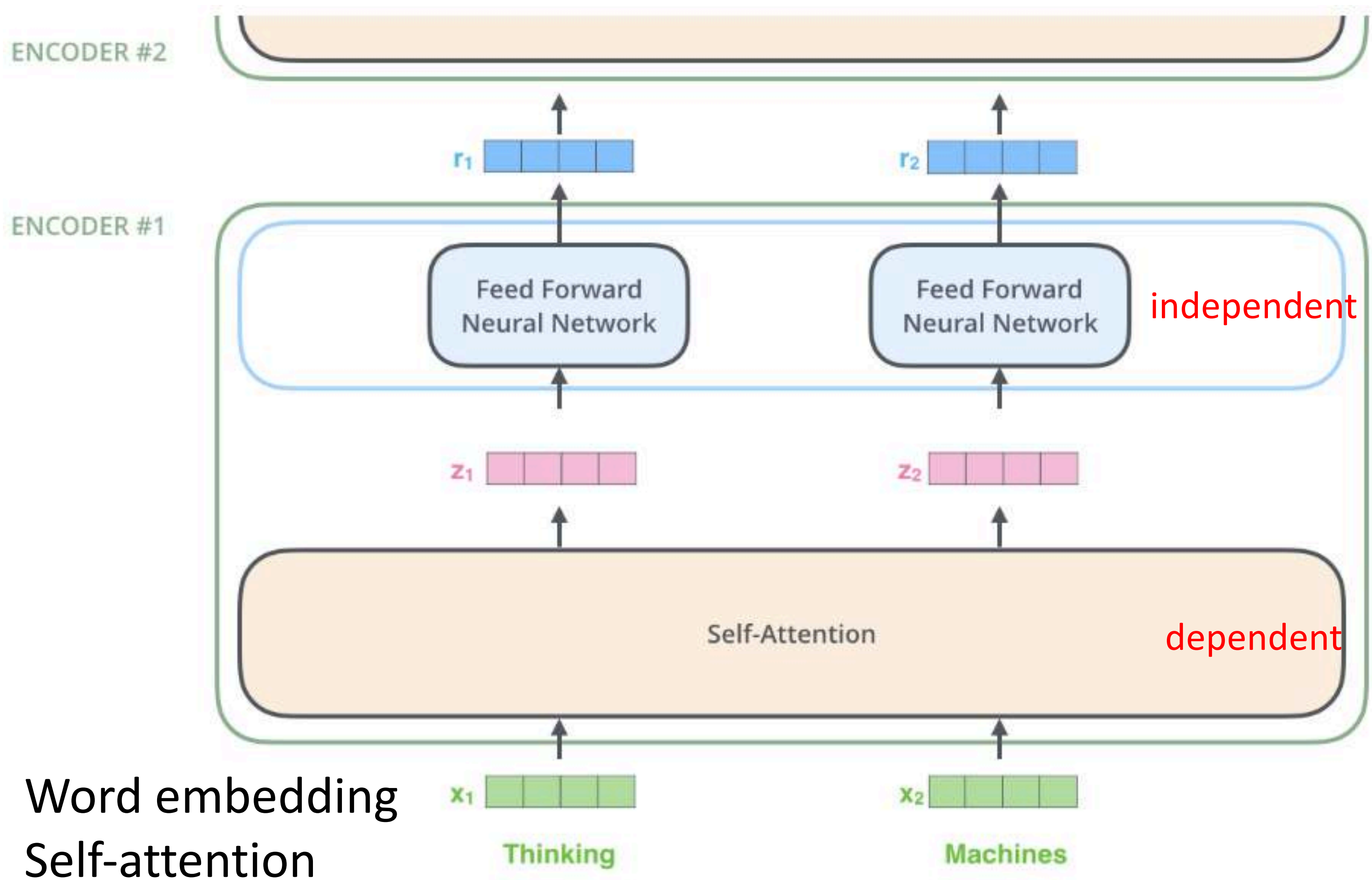


Encoder-Decoder



1. The **encoders** are all identical in structure (yet they do not share weights).
2. The **encoder's** inputs 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.
3. The outputs of the self-attention layer are fed to **a feed-forward neural network**. The exact same feed-forward network is independently applied to each position.
4. 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

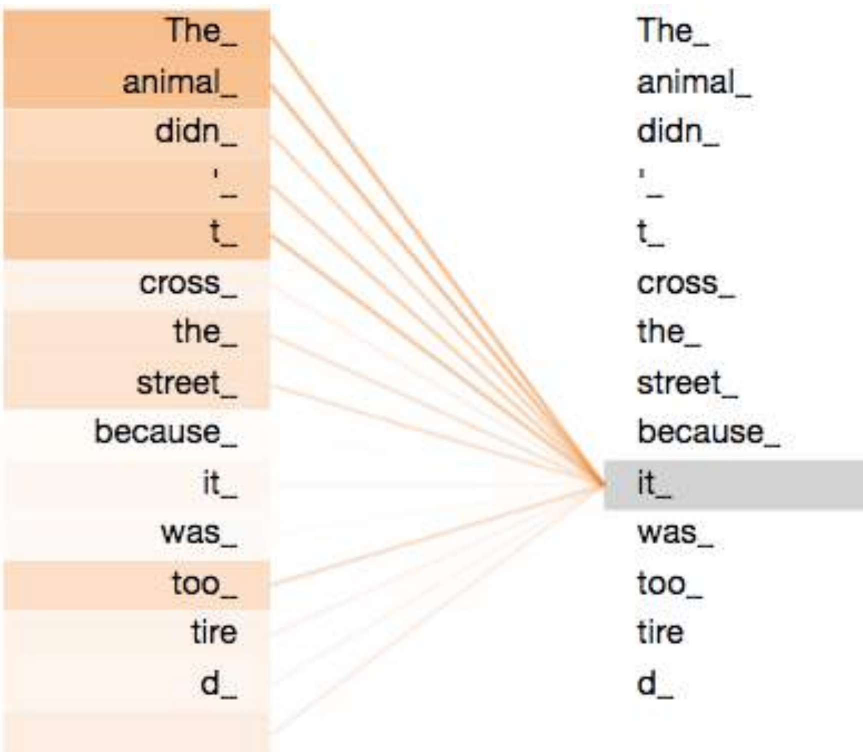
Encoder Detail



1. Word embedding
2. Self-attention
3. FFNN

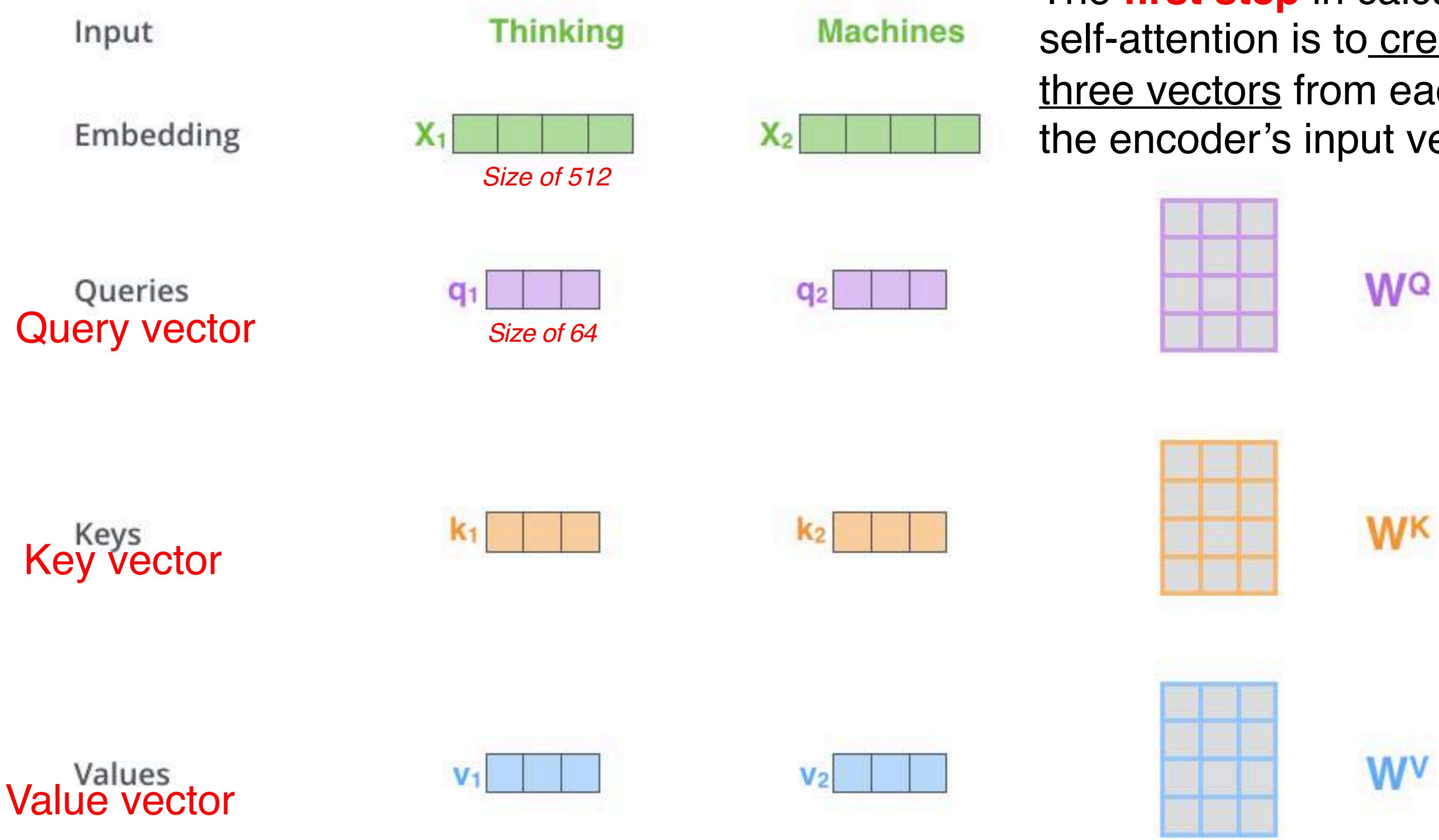
Self-Attention High Level

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



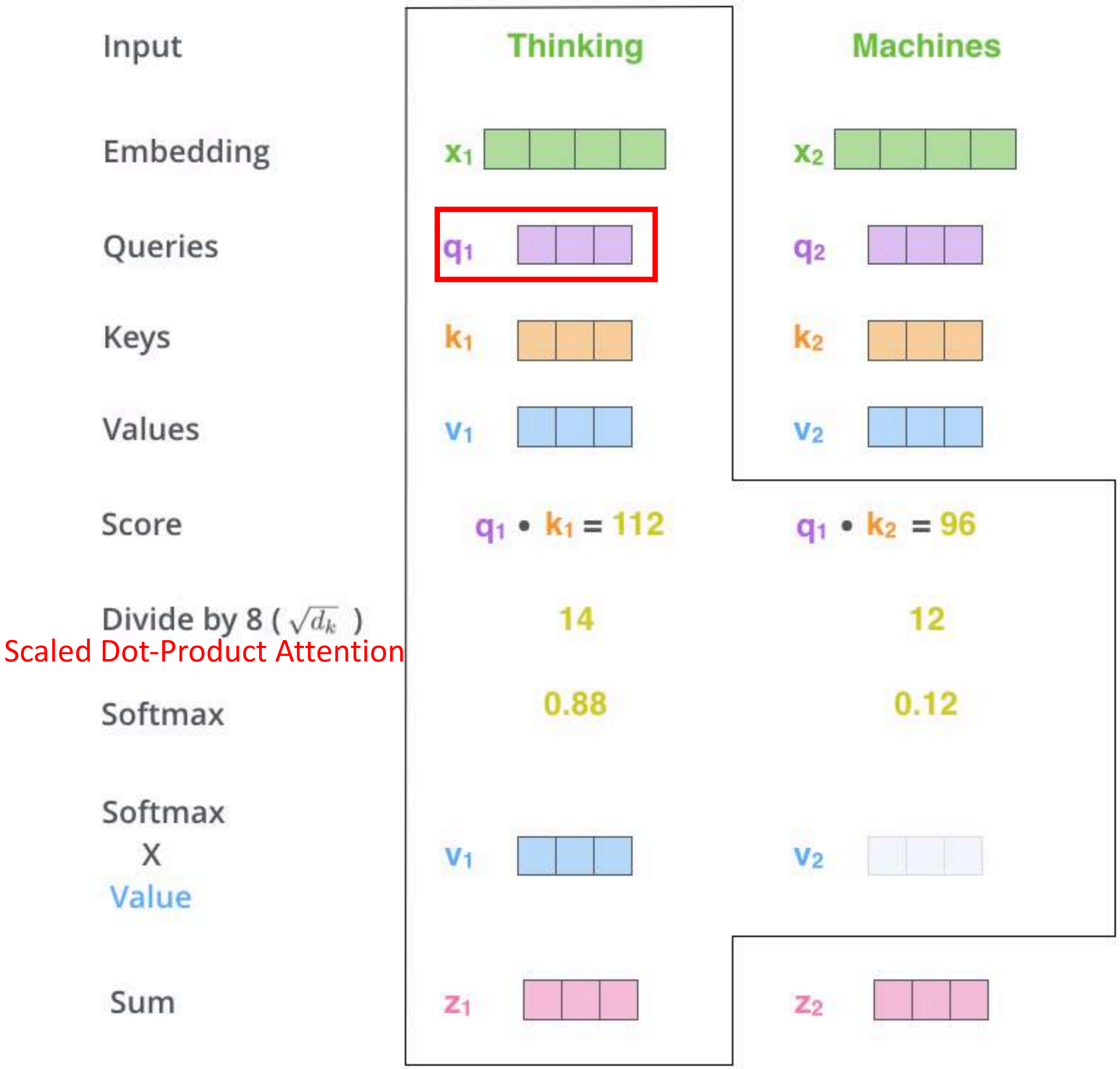
Self-Attention in Detail

The **first step** in calculating self-attention is to create three vectors from each of the encoder's input vectors



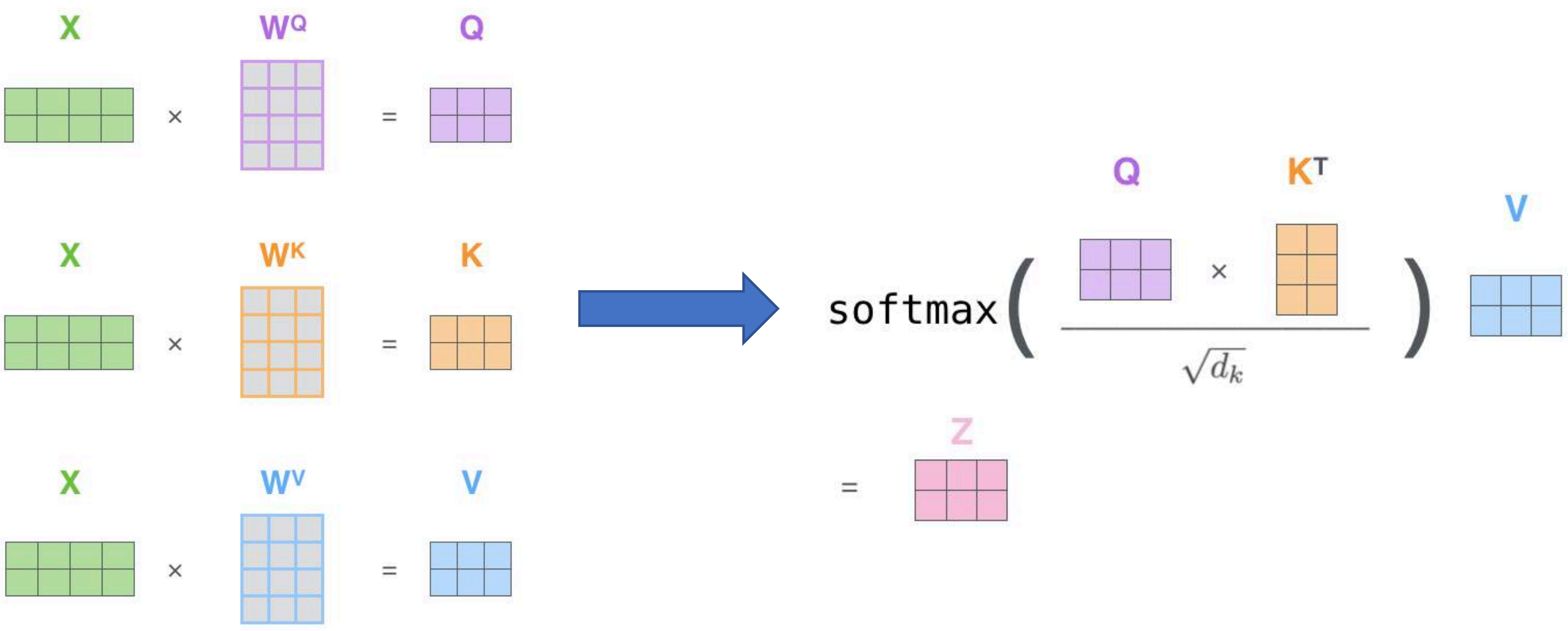
Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

Self-Attention in Detail



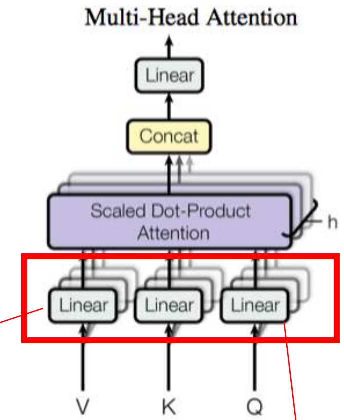
- The **second step** in calculating self-attention is to calculate a score.
- The **third and forth steps** are to divide the scores by 8, then pass the result through a softmax operation.
- The **fifth step** is to multiply each value vector by the softmax score
- The **sixth step** is to sum up the weighted value vectors.

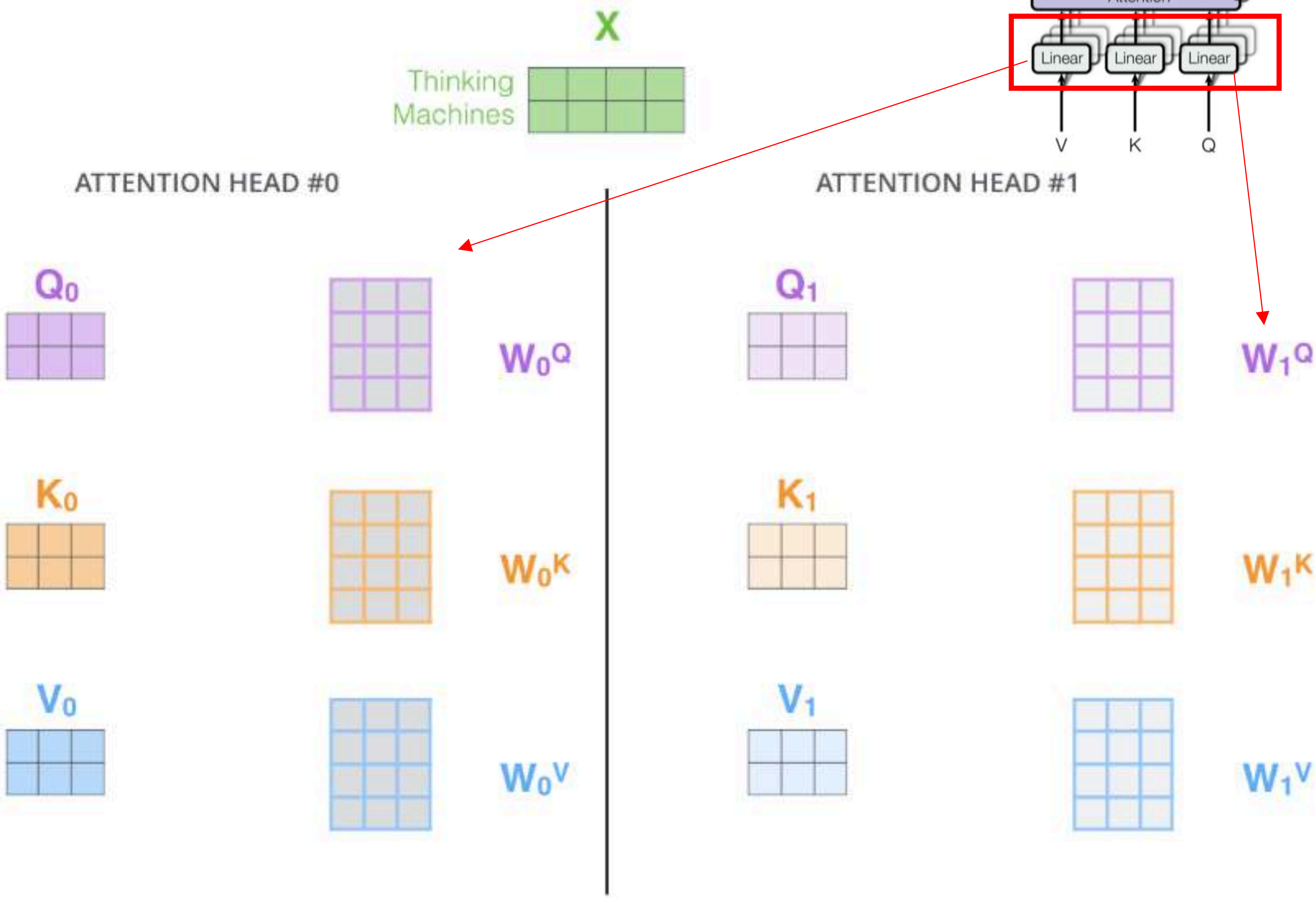
Self-Attention in Detail



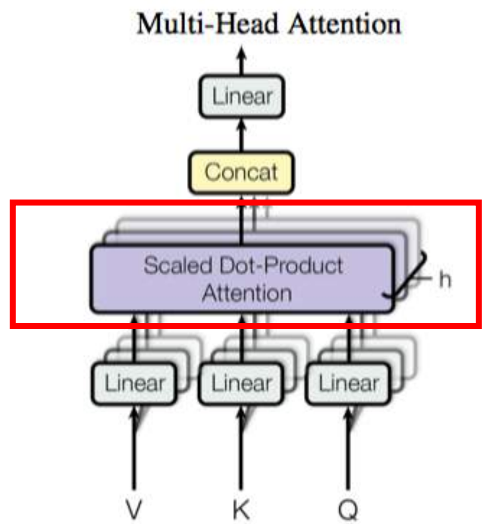
The self-attention calculation in matrix form

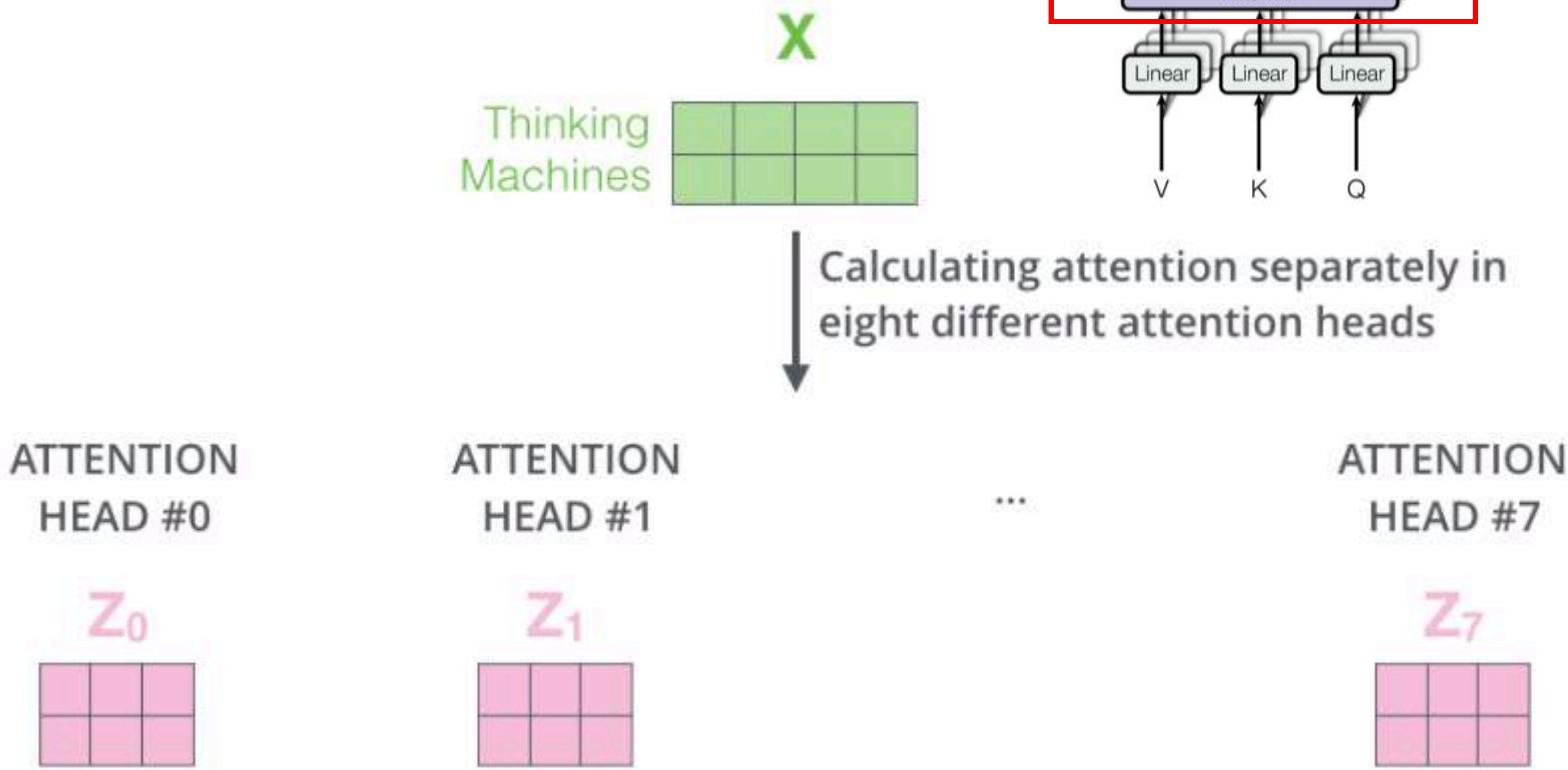
Multi-head attention





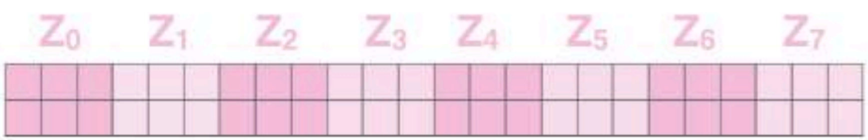
Multi-head attention





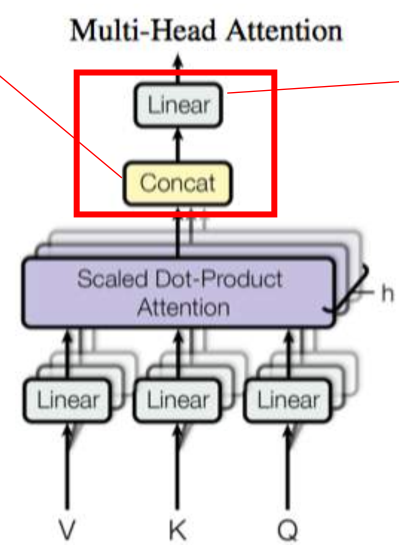
Multi-head attention

1) Concatenate all the attention heads

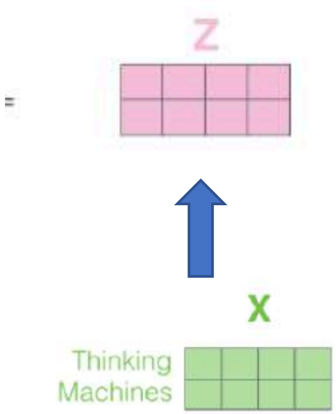


2) Multiply with a weight matrix W^O that was trained jointly with the model

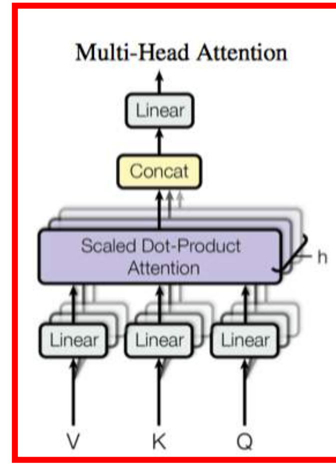
X

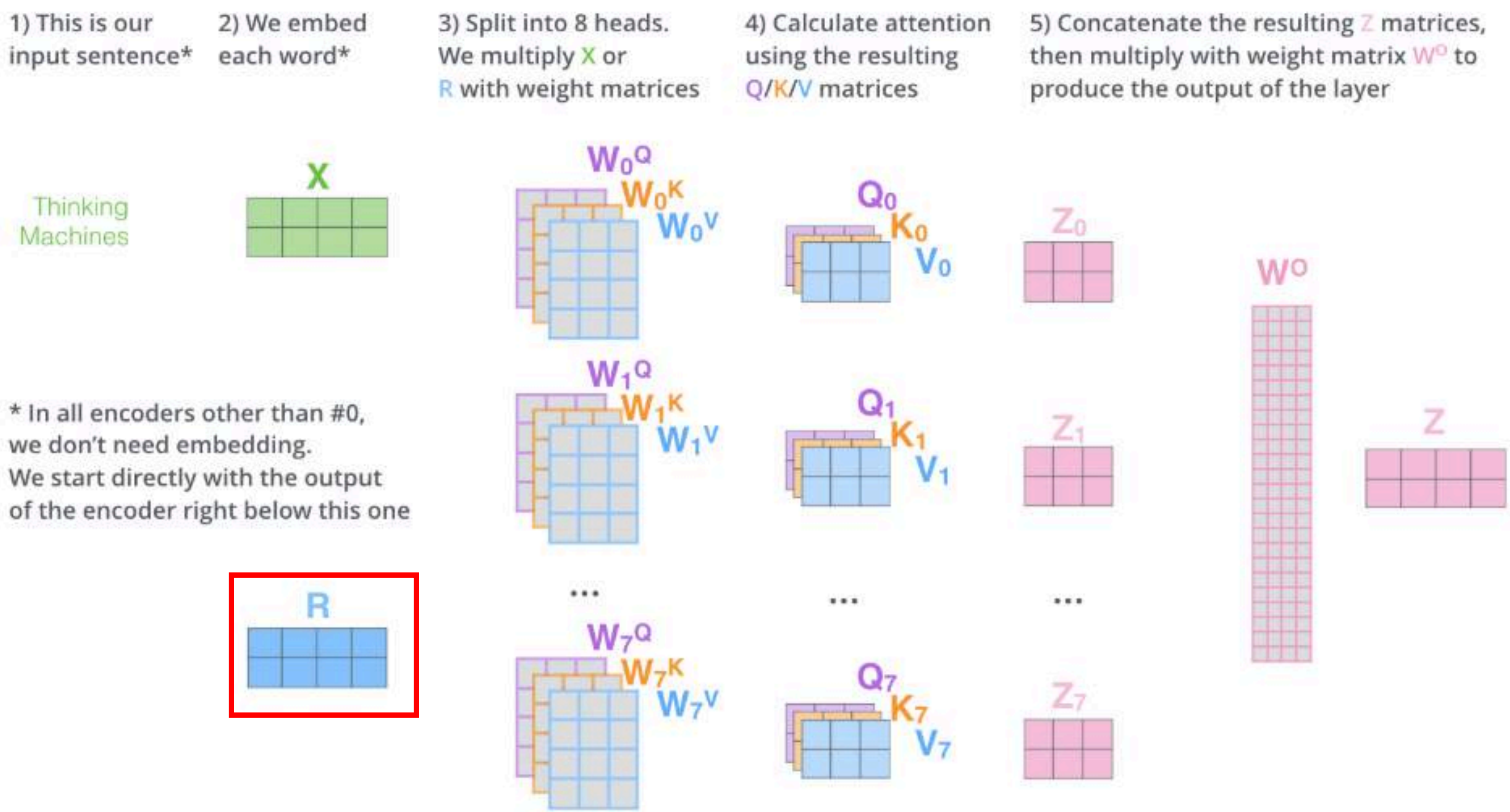


3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

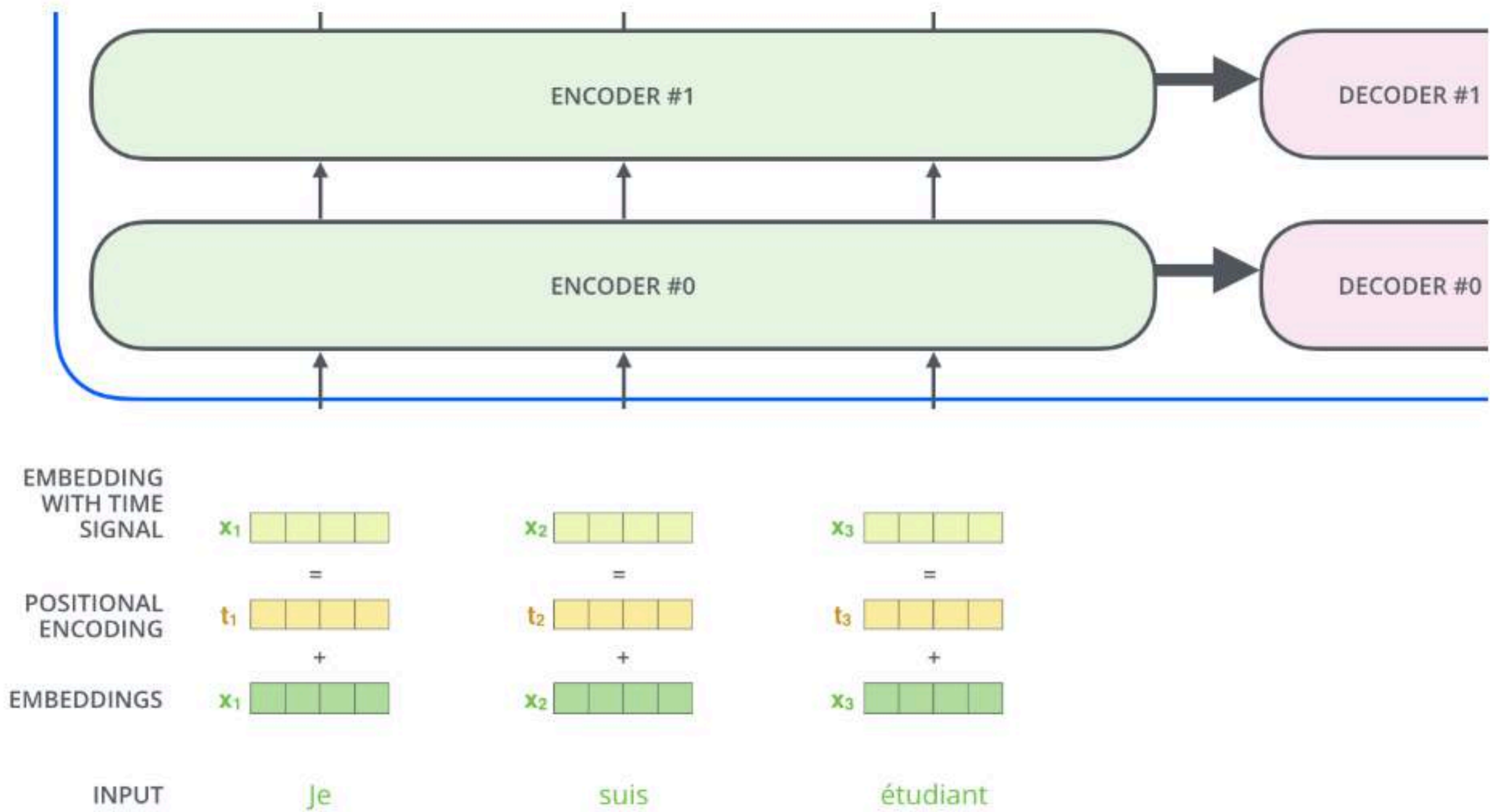


Multi-head attention



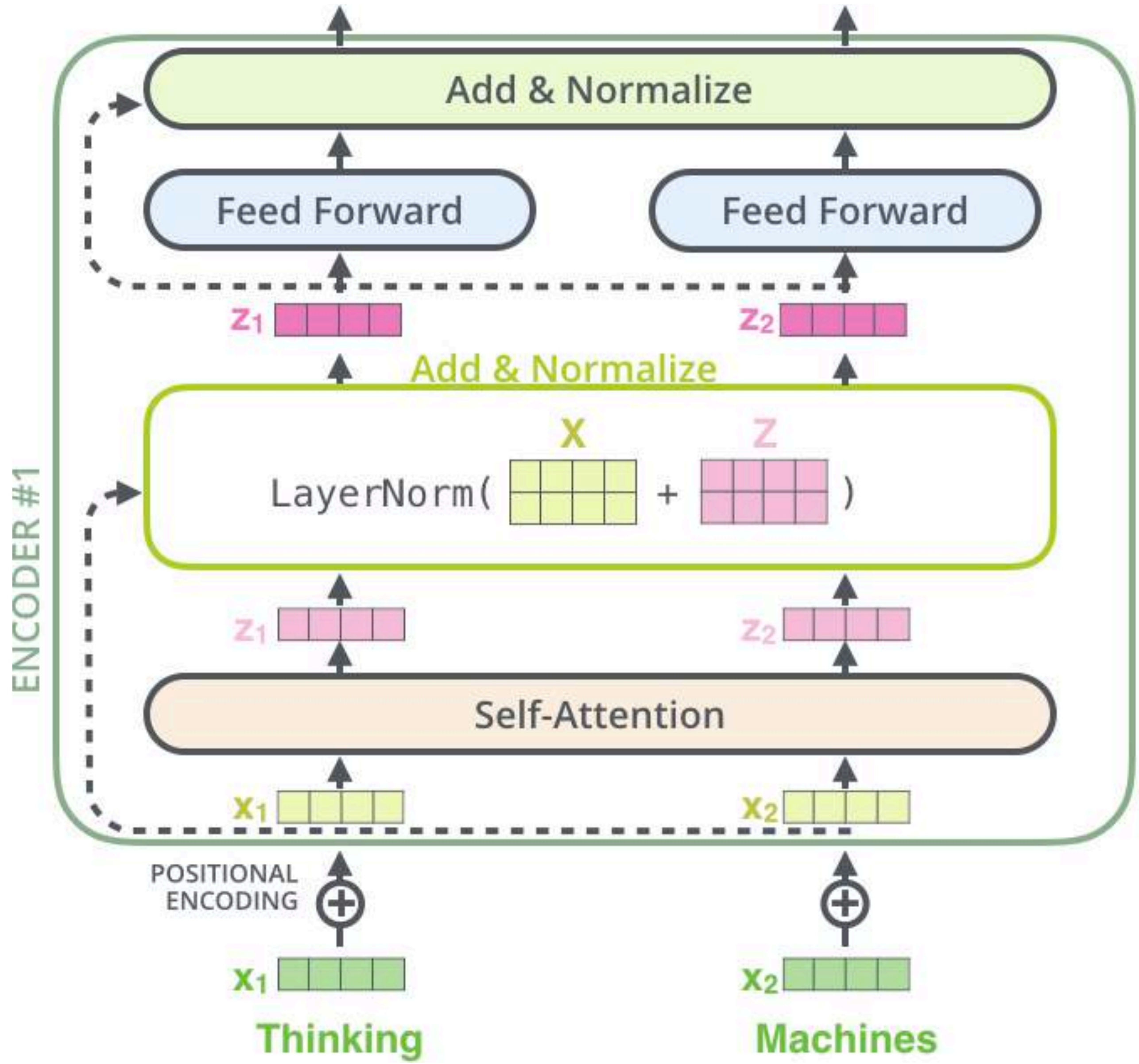


Positional Encoding

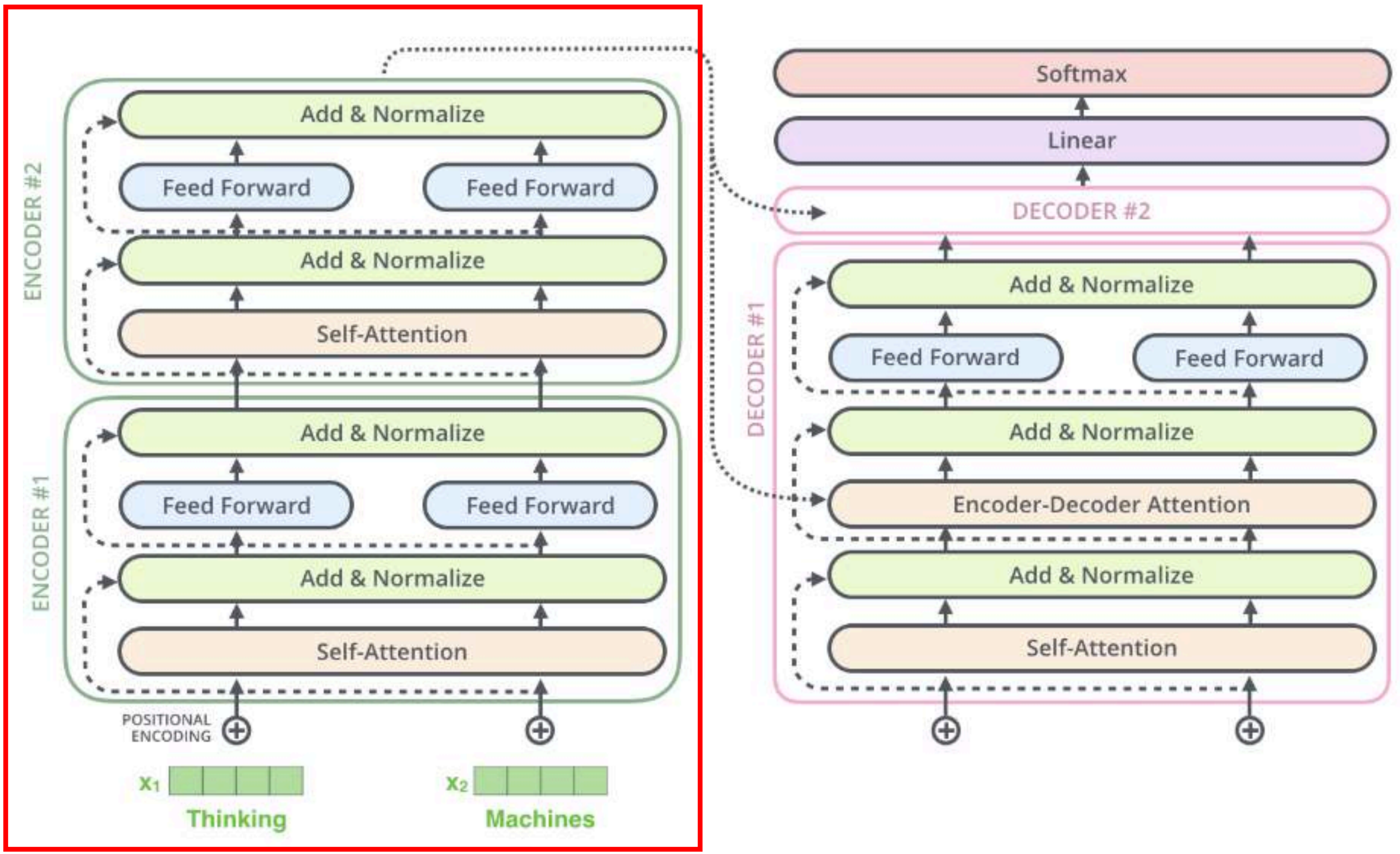


To give the model a sense of the order of the words, we add positional encoding vectors -- the values of which follow a specific pattern.

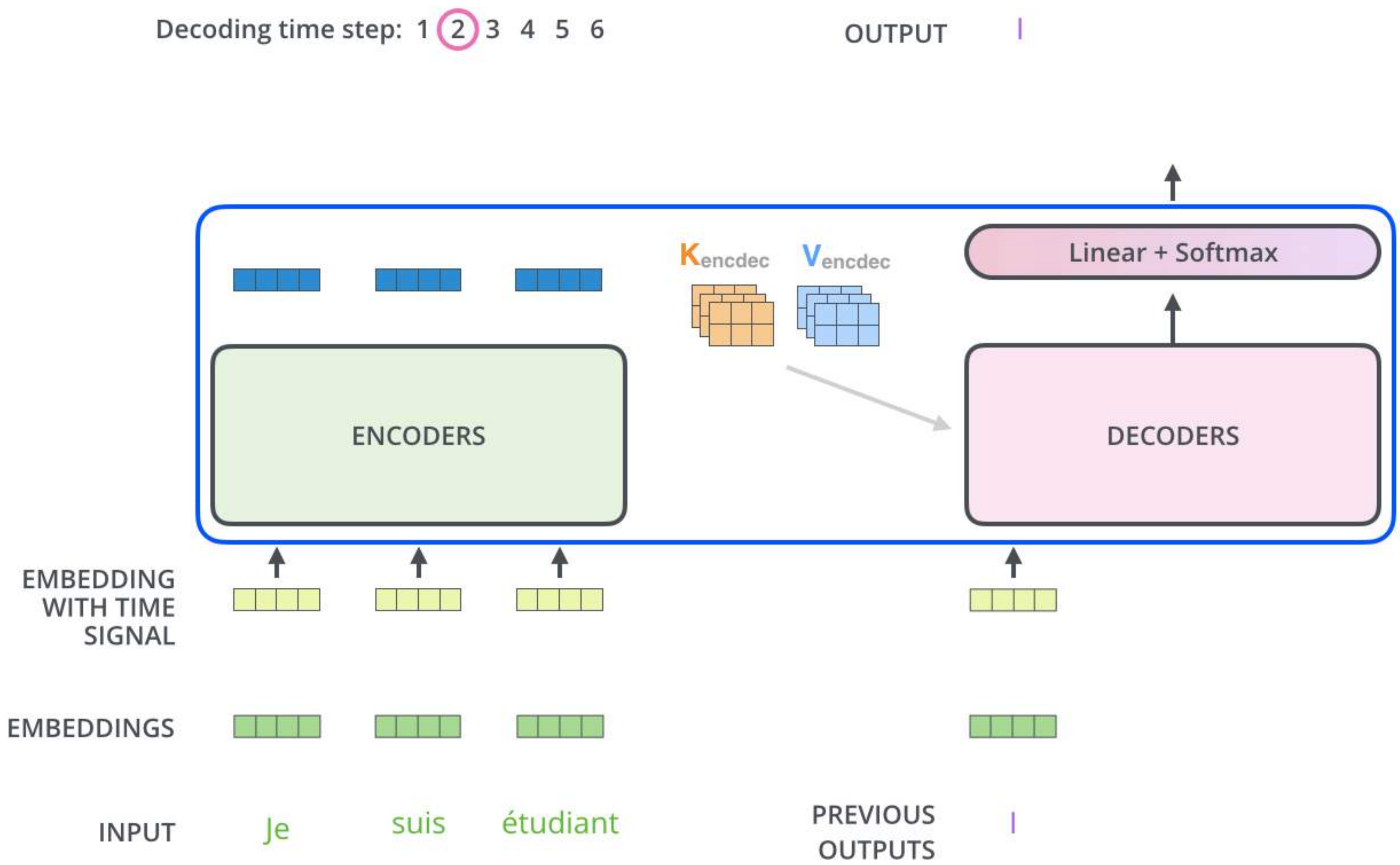
The Residuals



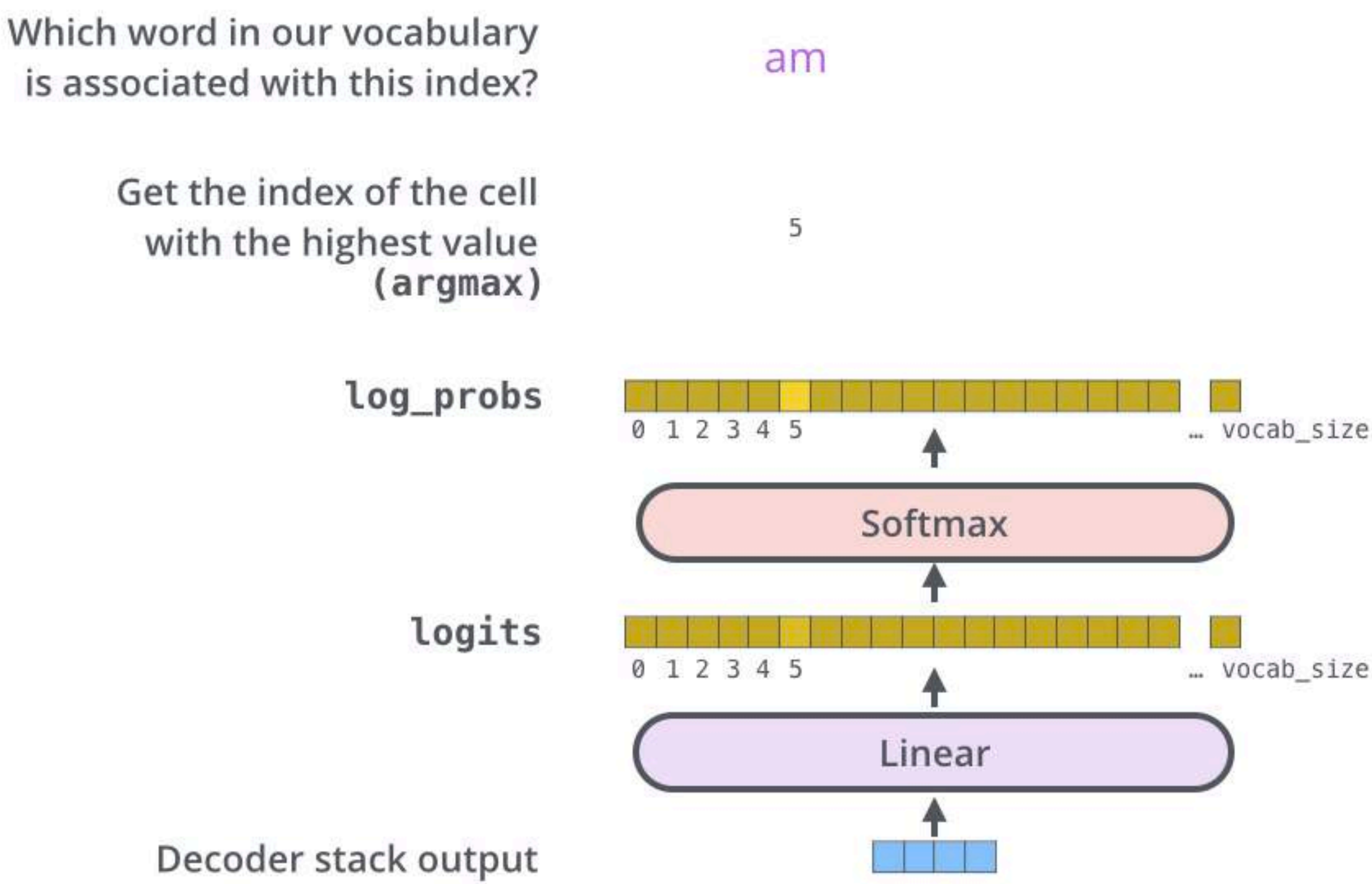
Encoder-Decoder



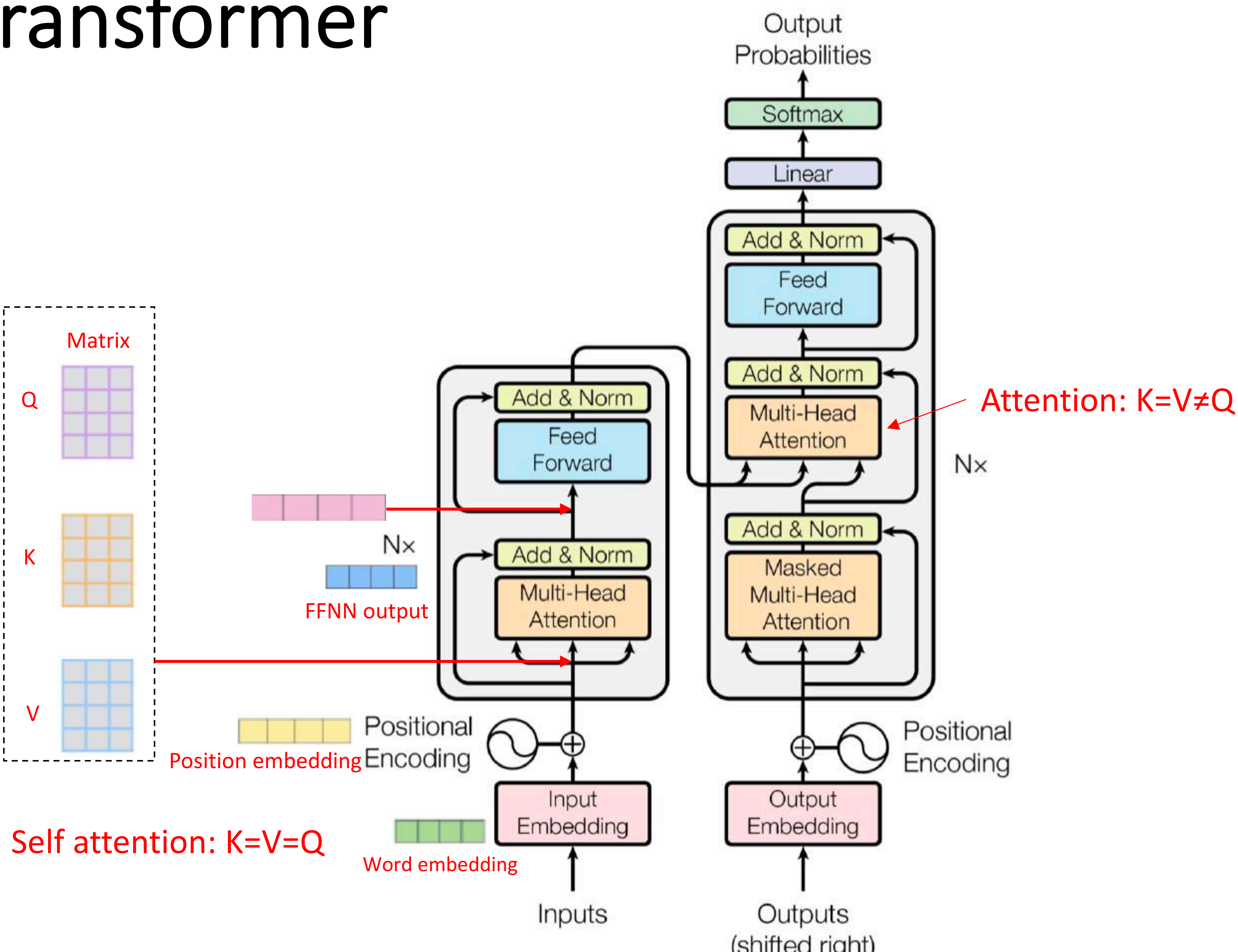
Decoder


Decoding time step: 1 2 3 4 5 6
OUTPUT
I
Kencdec
Vencdec
Linear + Softmax
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

Linear and Softmax Layer



Transformer



Self attention: K=V=Q

Outline

1. Encoder-Decoder
2. Attention
3. Transformer: 《Attention is all you need》
4. Word embedding and pre-trained model
5. ELMo: 《Deep contextualized word representations》
6. OpenAI GPT: 《Improving Language Understanding by Generative Pre-Training》
7. BERT: 《BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding》
8. Conclusion

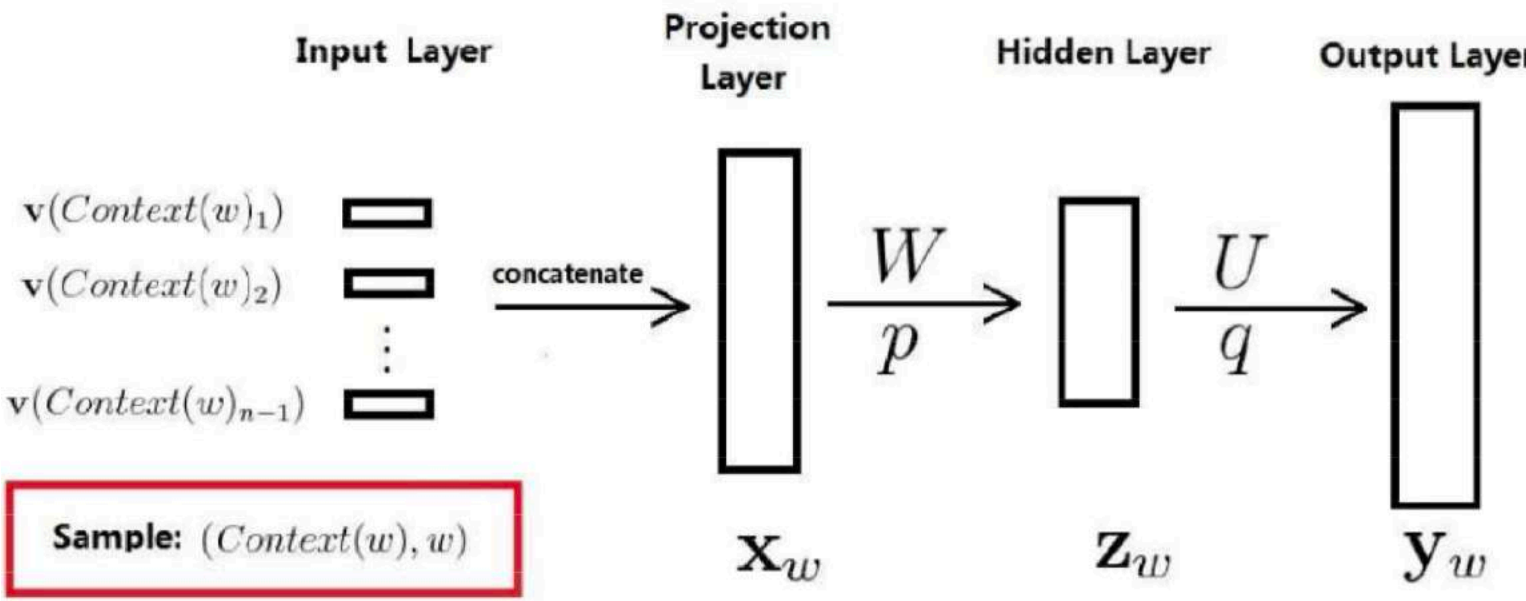
Language model

- **Language model** is a **probability distribution** over a sequences of words.

$$P(w_1, w_2, \ldots, w_m) = p(w_1)p(w_2|w_1)p(w_3|w_1, w_2) \ldots$$

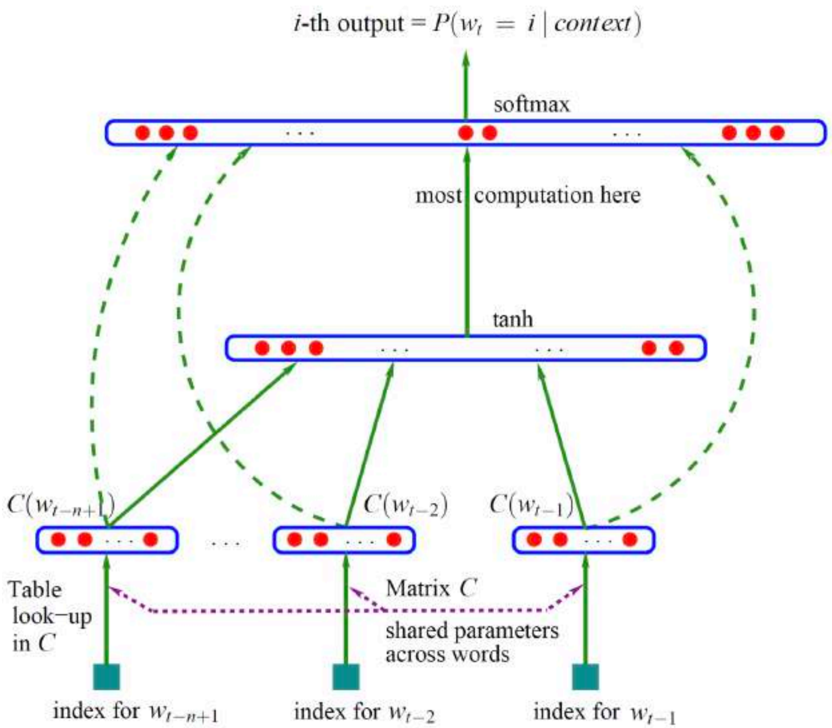
- N-Gram Models
  - Uni-gram
  - Bi-gram
  - Tri-gram
- Neural network language models(NNLM)

NNLM

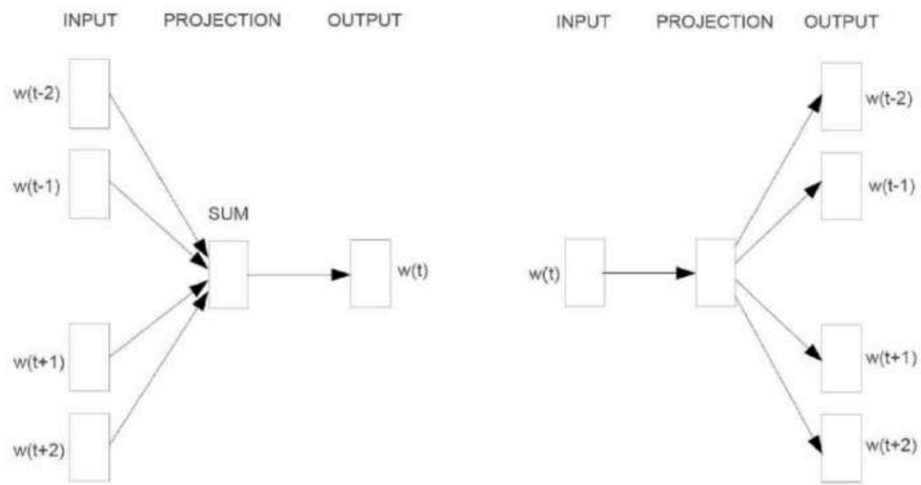


$$\begin{cases} Z_w = \tanh(Wx_w + p) \\ y_w = Uz_w + q \\ softmax(y_w) \end{cases}$$

NNLM and Word2Vec



Neural probabilistic language model(2003)



Word2vec(2013)

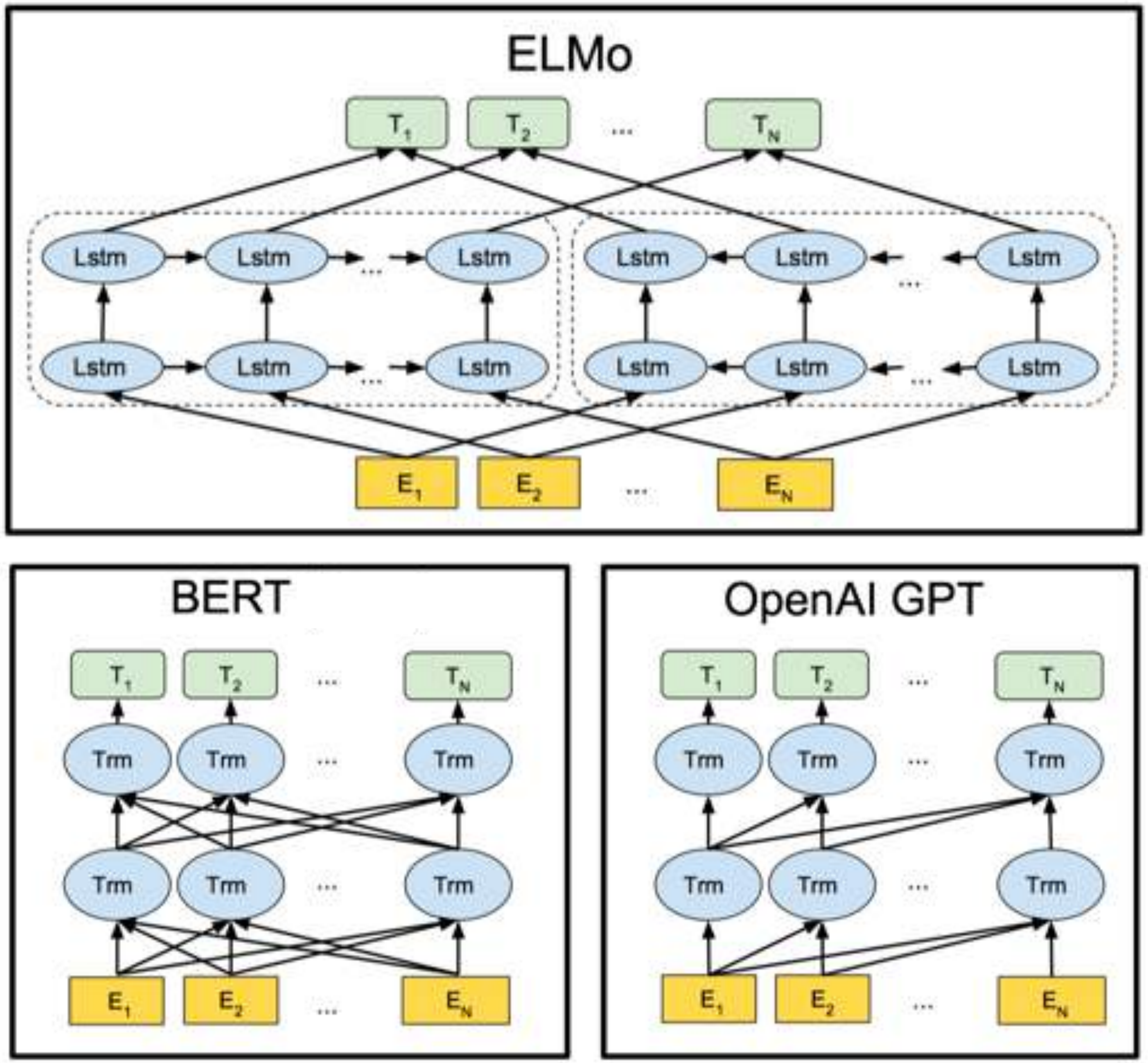
Pre-training

- Word embedding
  - Word2vec
  - Glove
  - FastText
  - …
- Transfer learning

Outline

1. Encoder-Decoder
2. Attention
3. Transformer: 《Attention is all you need》
4. Word embedding and pre-trained model
5. ELMo: 《Deep contextualized word representations》
6. OpenAI GPT: 《Improving Language Understanding by Generative Pre-Training》
7. BERT: 《BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding》
8. Conclusion

Overview



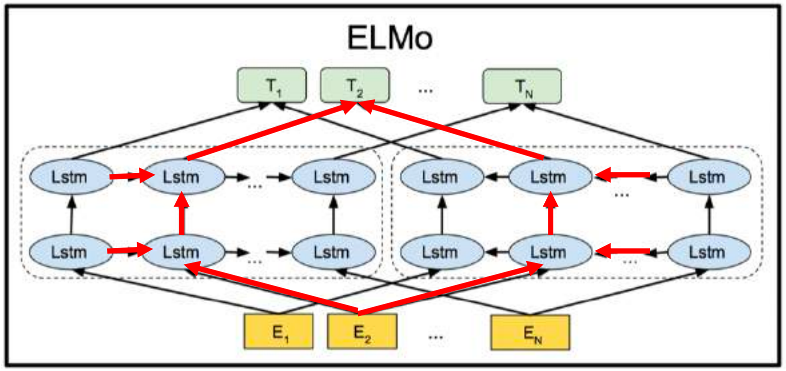
ELMo

- ELMo (**E**mbeddings from **L**anguage **Mo**dels)
  - complex characteristics of word use (syntax and semantics)
  - across linguistic contexts (polysemy)
- **Feature-Based**
- ELMo representations are deep, in the sense that they are a function of all of the internal layers of the biLM.
- The higher-level LSTM states capture context-dependent aspects of word meaning, while lower-level states model aspects of syntax.

Bidirectional language models

- Forward language model

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_1, t_2, \ldots, t_{k-1}).$$



- Backward language model

$$p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_{k+1}, t_{k+2}, \ldots, t_N)$$

- Jointly maximizes the log likelihood of the forward and backward directions

$$\sum_{k=1}^{N} \Big(\log p(t_k \mid t_1, \ldots, t_{k-1}; \boxed{\Theta_x}, \overrightarrow{\Theta}_{LSTM}, \boxed{\Theta_s}) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \boxed{\Theta_x}, \overleftarrow{\Theta}_{LSTM}, \boxed{\Theta_s}) \Big)$$

Θ_x Token representation

Θ_s Softmax layer

share some weights between directions instead of using completely independent parameters.

Embedding from language models

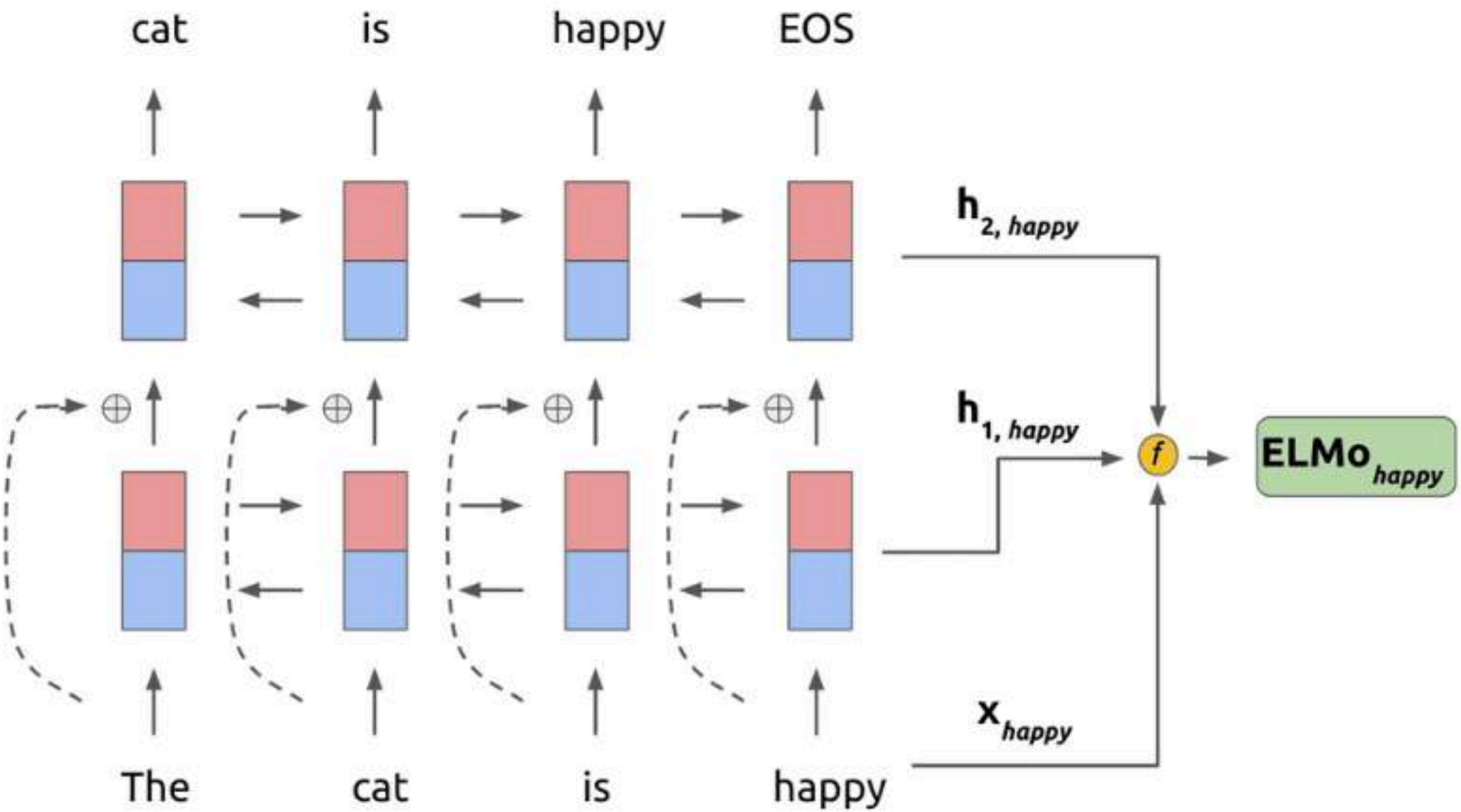
- ELMo is a task specific combination of the intermediate layer representations in the biLM.
- For k-th token, L-layer bi-directional Language models computes 2L+1 representations:

$$\begin{aligned} R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\ &= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\}, \end{aligned}$$

- For a specific down-stream task, ELMo would learn a weight to combine these representations(In the simplest just selects the top layer $E(R_k) = \mathbf{h}_{k,L}^{LM}$)

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} \boxed{s_j^{task}} \mathbf{h}_{k,j}^{LM}.$$

Embedding from language models



$$ELMo_k^{task} = \gamma_k \cdot (s_0^{task} \cdot x_k + s_1^{task} \cdot h_{1,k} + s_2^{task} \cdot h_{2,k})$$

Using biLMs for supervised NLP tasks

- Concatenate the ELMo vector with initial word embedding and pass representation into the task RNN.

$$[\mathbf{x}_k; \mathbf{ELMo}_k^{task}]$$

- Including ELMo at the output of the task RNN by introducing another set of output specific linear weights.

$$[\mathbf{h}_k; \mathbf{ELMo}_k^{task}]$$

- Add a moderate amount of dropout to ELMo, in some cases to regularize the ELMo weights by adding $\lambda\|\mathbf{w}\|_2^2$ to the loss.

Experiment

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

1. Question answering
2. Textual entailment
3. Semantic role labeling
4. Coreference resolution
5. Named entity extraction
6. Sentiment analysis

ELMo

- Including representations from all layers improves overall performance over just using the last layer, and including contextual representations from the last layer improves performance over the baseline.
- A small λ is preferred in most cases with ELMo.
- Including ELMo at the output of the biRNN in task-specific architectures improves overall results for some tasks. but for SRL (and coreference resolution, not shown) performance is highest when it is included at just the input layer.
- The biLM is able to disambiguate both the part of speech and word sense in the source sentence.

Outline

1. Encoder-Decoder
2. Attention
3. Transformer: 《Attention is all you need》
4. Word embedding and pre-trained model
5. Elmo: 《Deep contextualized word representations》
6. OpenAI GPT: 《Improving Language Understanding by Generative Pre-Training》
7. BERT: 《BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding》
8. Conclusion

OpenAI GPT

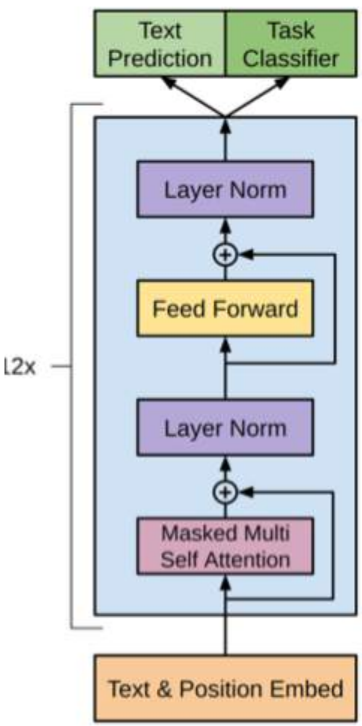
- **G**enerative **P**re-trained **T**ransformer
- Their goal is to learn a **universal representation** that transfers with little adaptation to a wide range of tasks.
  - First, use a language modeling objective on the unlabeled data to learn the initial parameters of a neural network model.
  - Second, adapt these parameters to a target task using the corresponding supervised objective.
- Highlight:
  - Use **transformer networks** instead of LSTM to achieve better capture long-term linguistic structure
  - Include **auxiliary training objectives** in addition to the task objective when fine-tuing.
  - Demonstrate the effectiveness of the approach on a wide range of tasks(significantly improving upon **the state of the art in 9 out of the 12 tasks studied**)

Unsupervised pre-training

- Use a **standard language modeling objective** to maximize the following likelihood:

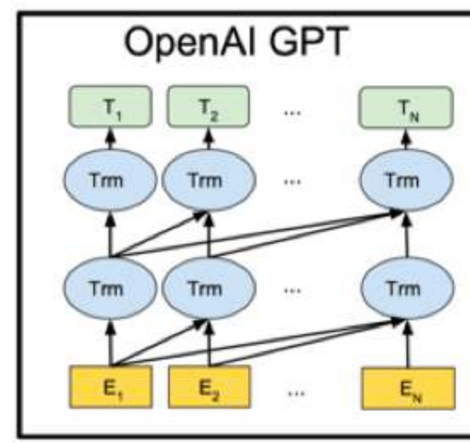
$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta)$$

- A **multi-layer transformer <u>decoder</u>** for the language model



$$h_0 = UW_e + W_p$$
$$h_l = \texttt{transformer\_block}(h_{l-1}) \forall i \in [1, n]$$
$$P(u) = \texttt{softmax}(h_n W_e^T)$$

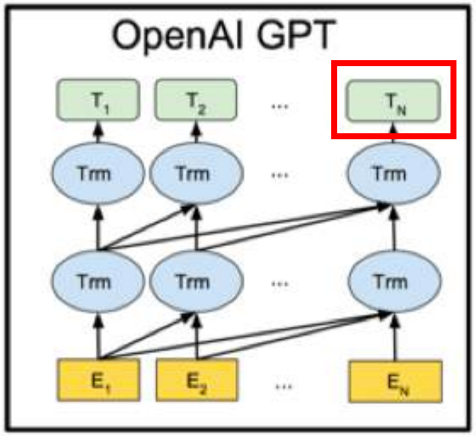
$U = (u_{-k}, \ldots, u_{-1})$	context vector of tokens
W_e	token embedding matrix
W_p	position embedding matrix
n	number of layers



Supervised fine-tuning

- The final transformer block`s activation is fed into an added linear output layer.

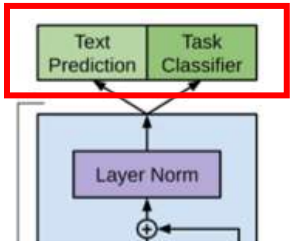
$$P(y|x^1,\ldots,x^m) = \texttt{softmax}(h_l^m W_y).$$



- Objective

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1,\ldots,x^m).$$

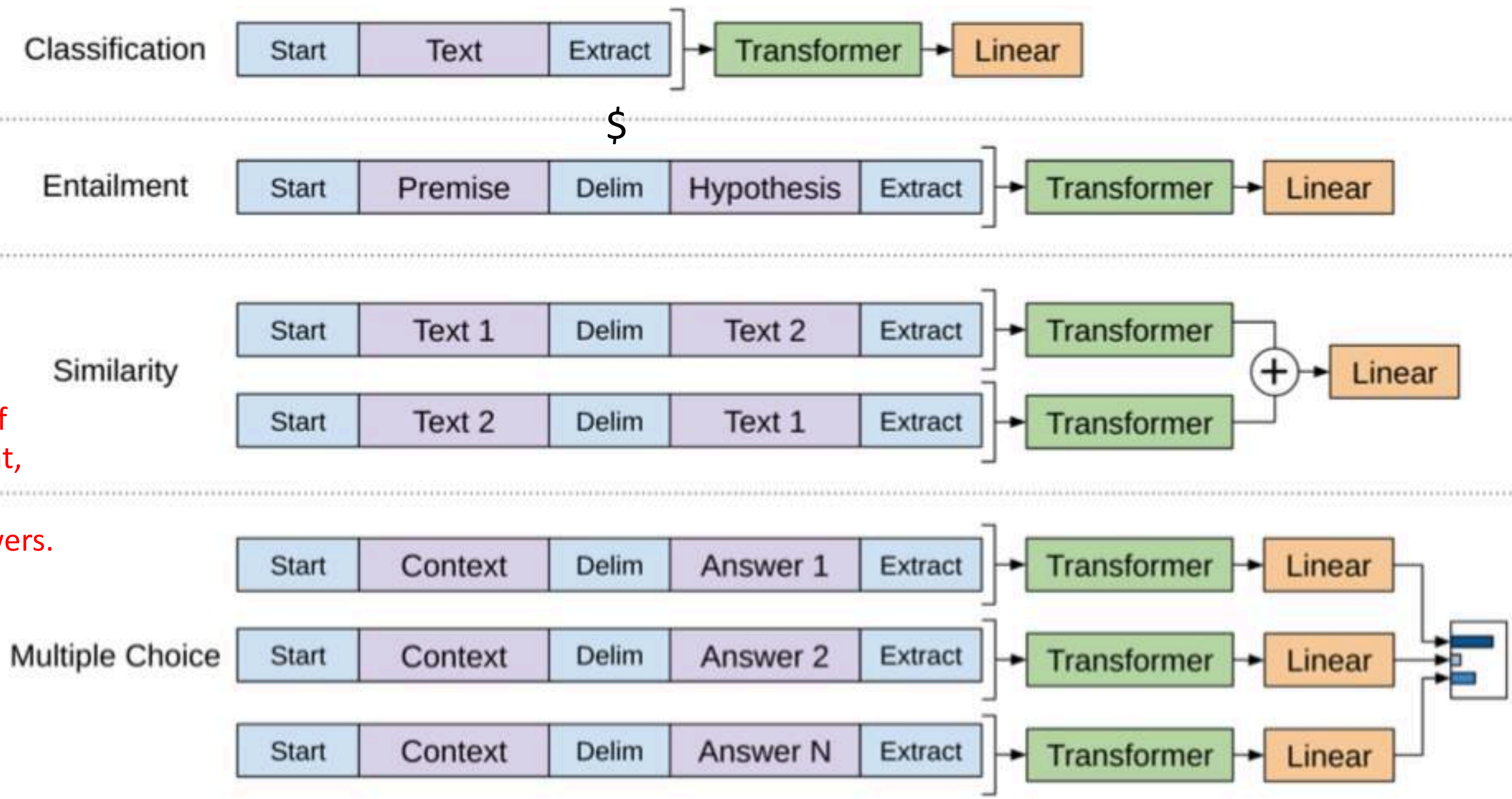
- We additionally found that including language modeling as an auxiliary objective to the fine-tuning helped learning by (a) improving generalization of the supervised model, and (b) accelerating convergence.



$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C})$$

Task specific input transformations

ordered sentence pairs, or triplets of document, question, and answers.



*convert structured inputs into an **ordered sequence** that our pre-trained model can process.*

ELMo vs OpenAI GPT

- **ELMo generalizes traditional word embedding research along a different dimension. integrating contextual word embeddings with existing task-specific architectures.(feature based)**

- **OpenAI GPT is to pre-train some model architecture on a LM objective before fine-tuning that same model for a supervised downstream task.(fine tuning)**

Outline

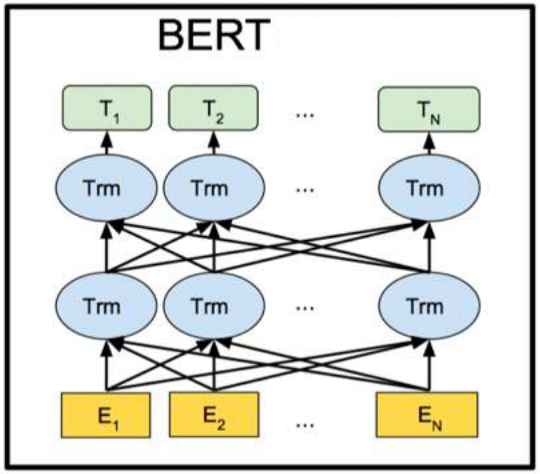
1. Encoder-Decoder
2. Attention
3. Transformer: 《Attention is all you need》
4. Word embedding and pre-trained model
5. ELMo: 《Deep contextualized word representations》
6. OpenAI GPT: 《Improving Language Understanding by Generative Pre-Training》
7. BERT: 《BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding》
8. Conclusion

BERT

- **B**idirectional **E**ncoder **R**epresentations from **T**ransformers.
- **Fine-tuning based**
- New pre-training objective
  - **Masked language model (MLM)**
    - randomly masks some of the tokens from the input, predict the original vocabulary id of the masked word based only on its context.
  - **Next sentence prediction task**
    - Binarized (is or not)
- Pre-trained representations eliminate the needs of many heavily engineered task-specific architectures.
- BERT advances the **state-of-the-art for 11 NLP tasks**.

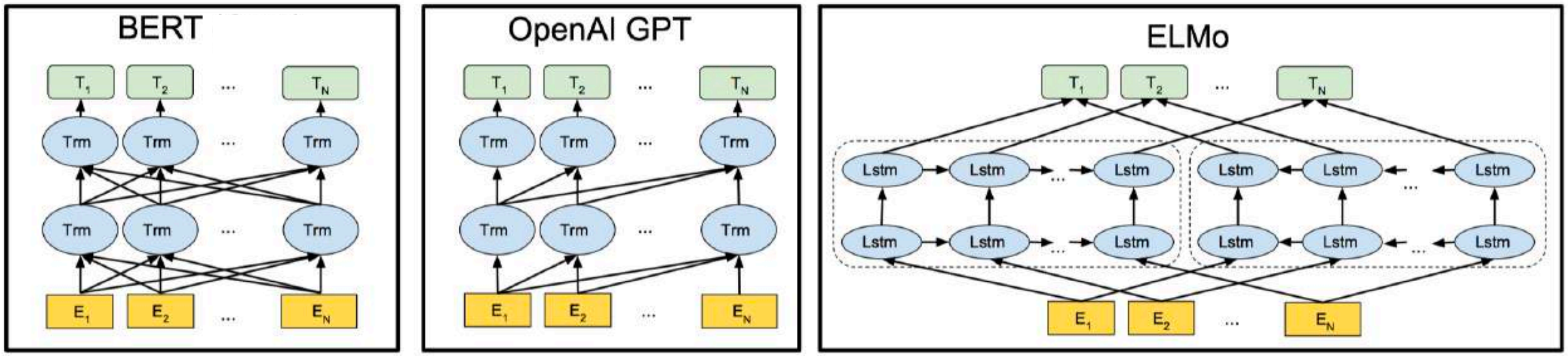
Model Architecture

- BERT's model architecture is a **multi-layer bidirectional Transformer encoder**.
  - L: number of layers
  - H: hidden size
  - A: number of self-attention heads.



- Model
  - $BERT_{BASE}$: L=12, H=768, A=12, Total Parameters=110M *(have an identical model size as OpenAI GPT for comparison purposes)*
  - $BERT_{LARGE}$: L=24, H=1024, A=16, Total Parameters=340M
- Note:
  - BERT: Bidirectional Transformer encoder
  - OpenAI: Left-context-only Transformer decoder

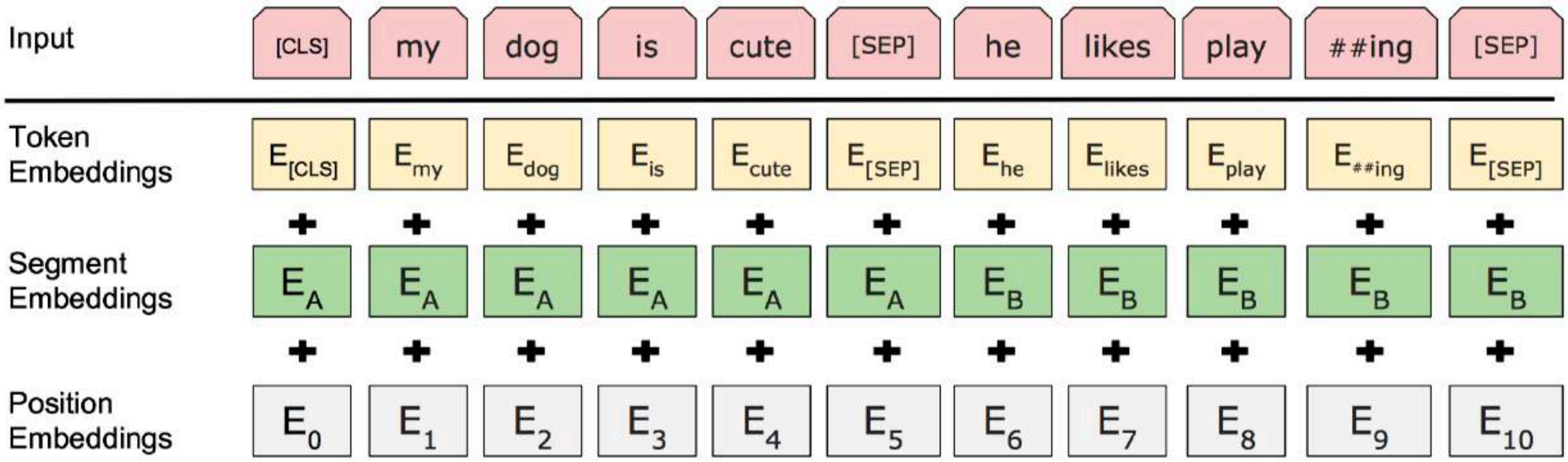
Model Architecture



- **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 LSTM

Input Representation

- For a given token, its input representation is constructed by summing the corresponding **token, segment and position embeddings.**



- **CLS**: Special classification embedding for classification tasks
- **E_A, E_B:** Sentence pairs are packed together into a single sequence. separate them with a special token ([SEP]).
- Learned **positional embeddings**

Tasks #1: Masked LM

- **Definition**: masking some percentage of the input tokens at random, and then predicting only those masked tokens.
- The final hidden vectors corresponding to the mask tokens are fed into an output softmax over the vocabulary, as in a standard LM.
- In practice: 15%
- Downsides:
  - Mismatch between pre-training and finetuning, since the [MASK] token is never seen during fine-tuning.
  - Only 15% of tokens are predicted in each batch, which suggests that more pre-training steps may be required for the model to converge.

Tasks #1: Masked LM

- Mismatch between pre-training and finetuning, since the [MASK] token is never seen during fine-tuning.
  1. 80% of the time: Replace the word with the [MASK] token
     - For training LM *my dog is hairy →my dog is [MASK]*
  2. 10% of the time: Replace the word with a random word
     - For adding noise *my dog is hairy → my dog is apple*
  3. 10% of the time: Keep the word unchanged
     - For the true *my dog is hairy → my dog is hairy*
- Only 15% of tokens are predicted in each batch, which suggests that more pre-training steps may be required for the model to converge.
  - empirical improvements of the MLM model far outweigh the increased training cost.

Tasks #2: Next Sentence Prediction

- In order to train a model that understands **sentence relationships.**
- **Binarized** next sentence prediction task
- Choosing the sentences A and B for each pretraining example, 50% of the time B is the actual next sentence that follows A, and 50% of the time it is a random sentence from the corpus.

Input = `[CLS] the man went to [MASK] store [SEP]`
`he bought a gallon [MASK] milk [SEP]`

Label = `IsNext`

Input = `[CLS] the man [MASK] to the store [SEP]`
`penguin [MASK] are flight ##less birds [SEP]`

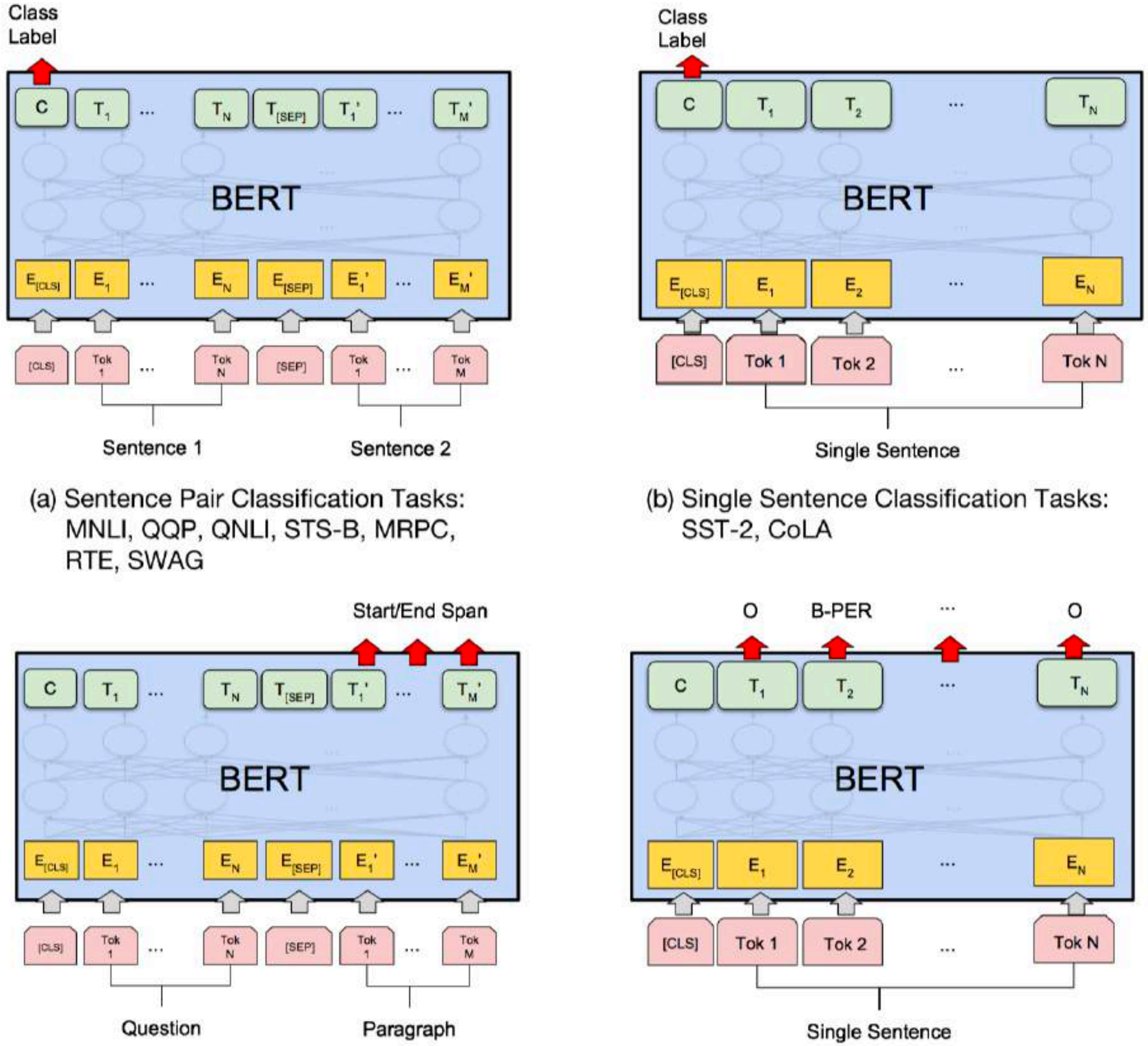
Label = `NotNext`

Training

- The training loss is the sum of the mean masked LM likelihood and mean next sentence prediction likelihood.

- Training of BERTBASE was performed on 4 Cloud TPUs in Pod configuration (16 TPU chips total). 5 Training of BERTLARGE was performed on 16 Cloud TPUs (64 TPU chips total). Each pretraining took 4 days to complete.

Fine-tuning Procedure



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

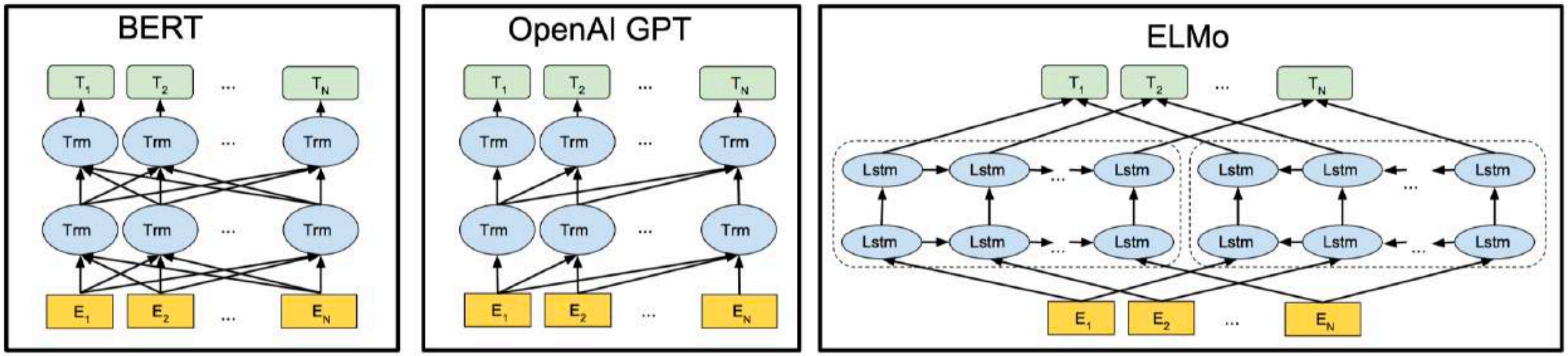
(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Outline

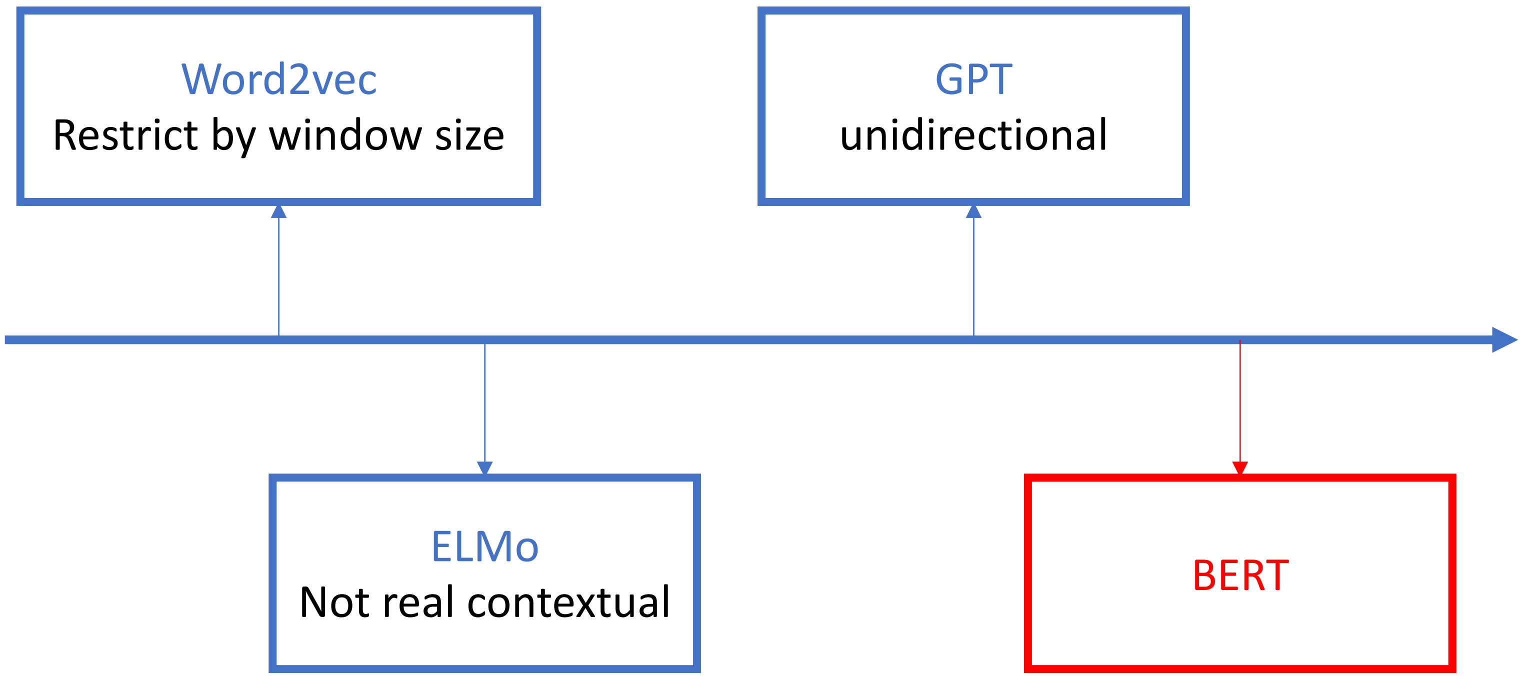
1. Encoder-Decoder
2. Attention
3. Transformer: 《Attention is all you need》
4. Word embedding and pre-trained model
5. ELMo: 《Deep contextualized word representations》
6. OpenAI GPT: 《Improving Language Understanding by Generative Pre-Training》
7. BERT: 《BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding》
8. Conclusion

BERT vs GPT vs ELMo



- **Pre-trained language representations**
  - **Feature based**: ELMO
  - **Fine-tuning**: OpenAI GPT、BERT
- **Direction**
  - **Unidirectional**: Elmo、OpenAI GPT
  - **Bidirectional**: BERT
- **Pre-training objective**
  - **Elmo、OpenAI GPT**：Traditional language model
  - **BERT**：masked language model、next sentence prediction

Conclusion


Word2vec
Restrict by window size
GPT
unidirectional
ELMo
Not real contextual
BERT

Reference

- Peters, M. E. et al. Deep contextualized word representations. naacl (2018).
- Radford, A. & Salimans, T. Improving Language Understanding by Generative Pre-Training. (2018).
- Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. (2018).
- Vaswani, Ashish, et al. Attention is all you need. (2017).
- 深度学习中的注意力机制 https://blog.csdn.net/qq_40027052/article/details/78421155
- 论文笔记：Attention is all you need https://www.jianshu.com/p/3f2d4bc126e6
- 自然语言处理中的自注意力机制 http://ir.dlut.edu.cn/news/detail/485
- Jay Alammar: https://jalammar.github.io/illustrated-transformer/
- [论文笔记]ELMo https://zhuanlan.zhihu.com/p/37684922
- BERT 笔记 http://blog.tvect.cc/archives/799
- 详细解读谷歌新模型 BERT 为什么嗨翻 AI https://mp.weixin.qq.com/s/8uZ2SJtzZhzQhoPY7XO9uw
- 自然语言处理中的语言模型预训练方法 http://ir.dlut.edu.cn/news/detail/485
- NLP的游戏规则从此改写？从word2vec, ELMo到BERT https://mp.weixin.qq.com/s/I315hYPrxV0YYryqsUysXw

If I forget any tutorial, please forgive me, Thanks a lot for all of the excellent materials.

Thanks!