CS5424 : Neural Networks and Deep Learning

Lecture 9: Attention Neural Networks

Semester 1 2022/23

Ai Xin

aixin@comp.nus.edu.sg

Department of Computer Science National
University of Singapore (NUS)

Outline

- **An introduction on Attention**
- **General Attention**
- **How attention is used in Transformer**
- **Understanding Transformer**
- **Why Transformer works?**
- **Implement Transformer for Language Model**
- **Implement Transformer for Seq2Seq Model**
- **Conclusion**

Outline

- **An introduction on Attention**
- General Attention
- How attention is used in Transformer
- Understanding Transformer
- Why Transformer works?
- Implement Transformer for Language Model
- Implement Transformer for Seq2Seq Model
- Conclusion

End of RNNs ?

- Attention neural networks are taking over NLP !
  - 2019 breakthrough in AI

Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[*] [†]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

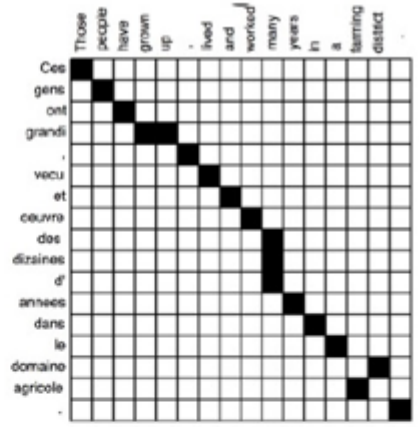
Illia Polosukhin[*] [‡]
illia.polosukhin@gmail.com

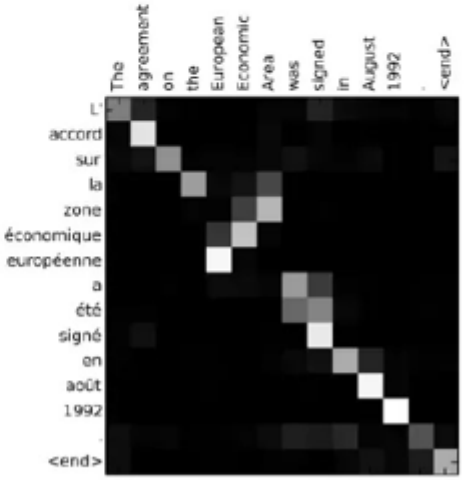
Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Attention Applications

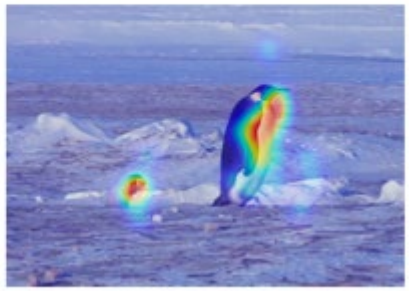
- Attention in NLP :
  - Alignment in Machine Translation (MT) :
    - Hard attention :
      - (Binary) alignment is a (difficult) combinatorial optimization problem.
    - Soft attention :
      - For each word in target sequence, get a probability over words in the source sequence [Brown-et-al'93].
      - Better approach as continuous relaxation of the combinatorial matching can be solved by backpropagation !



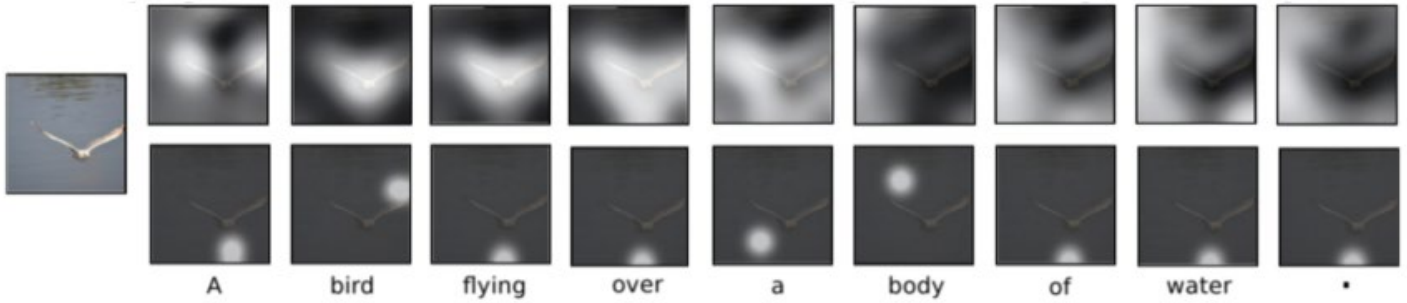


Attention Applications

- Attention in CV :
  - Attention over grid



Visual attention



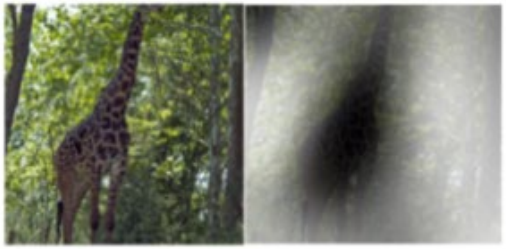
A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

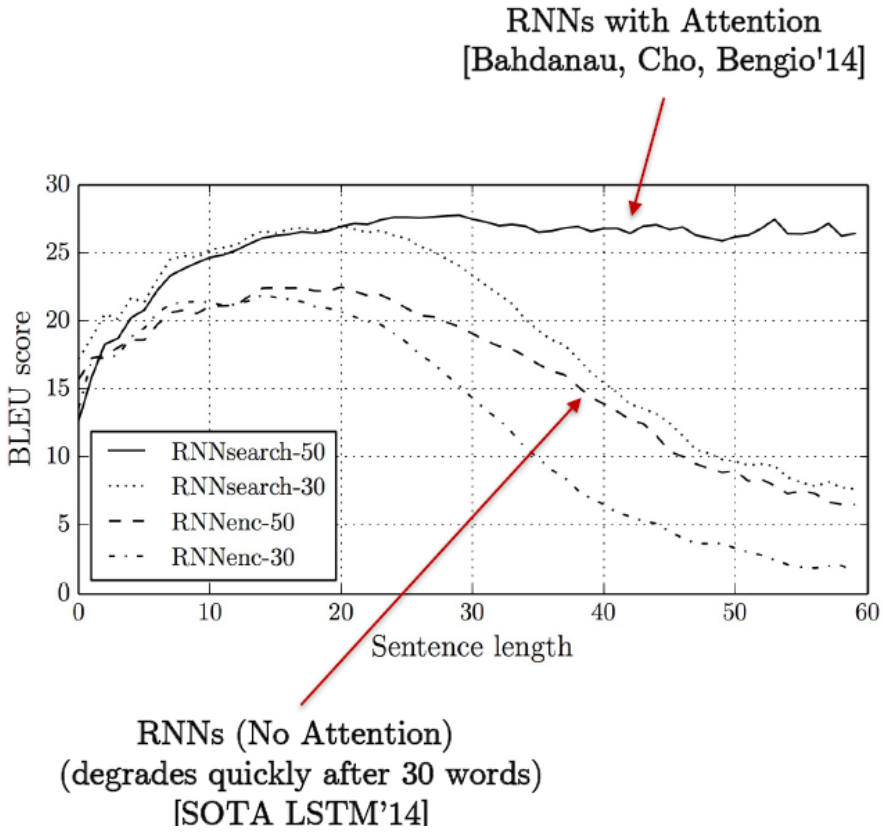
A group of people sitting on a boat in the water.



A giraffe standing in a forest with trees in the background.

Attention in NLP

- Attention in NLP :
  - First paper to do soft alignment with attention in MT is [Bahdanau, Cho, Bengio'14], followed by [Luong, Pham, Manning'15].
    - Single hop attention.
    - Improved SOTA in MT.
    - Precursor of Transformers [Vaswani-et-al'17].
    - Start competing RNNs/LSTM.
- Breakthrough idea in NLP !
  - As revolutionary as ConvNets for CV.

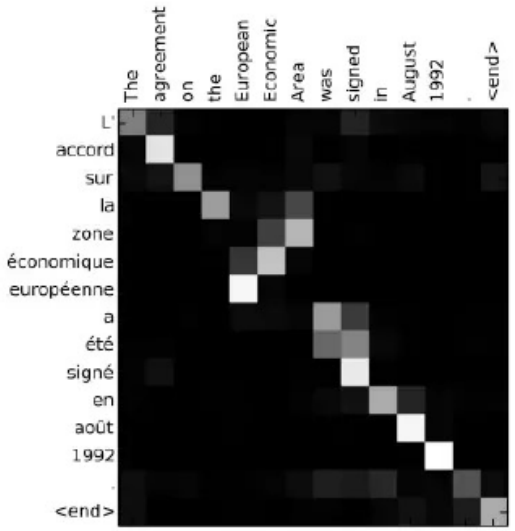


Attention in NLP

- Why casting MT as a soft alignment/attention problem is a great idea ?
  - With RNNs, the very long sequence requires to be memorized and represented by a single vector (that will be later decoded).
    - (The memory idea is nice but) RNN architectures of a memory module cannot simply deal with long sequences (limit of non-linear dynamic systems).
    - We ask too much to memorize everything with one vector !
  - With attention, we distribute the memorization load over each word.
    - Each word in the target sequence only needs to find its match with the word (or a few words) in the source target.
      - It solves the limitation of long-term dependencies in RNNs (any word in the target sequence communicates with any word in the source sequence).
    - The matching is made easy by transforming the words with hidden representations.
    - Attention is a key mathematical structure/property for NLP !
      - SOTA for all NLP tasks in 2019.

Attention in NLP

- Illustrations of attention



  - Example 1 : MT
    - Easy to match source-target words.

  - Example 2 : Sentiment Analysis
    - Easy to focus on relevant words.
      - Is the book's review good ?
        - Steve is arrogant but his book is awesome.

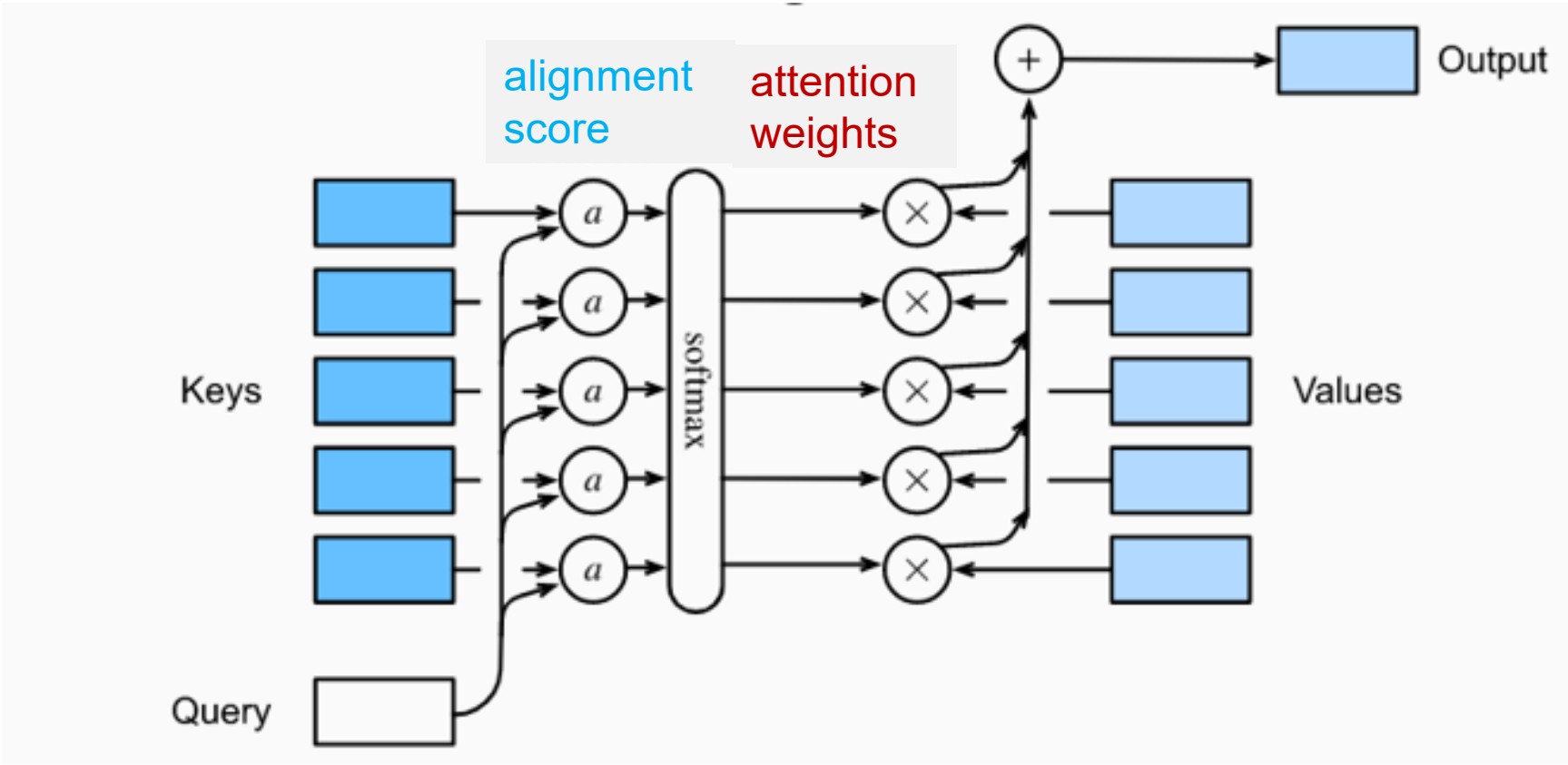
Focus on this positive word,
and ignore everything else !

Attention in NLP

Outline

- An introduction on Attention
- **General Attention**
- How attention is used in Transformer
- Understanding Transformer
- Why Transformer works?
- Implement Transformer for Language Model
- Implement Transformer for Seq2Seq Model
- Conclusion

General Attention



Context Vector (Output)

$$\mathbf{c}_t = \sum_{i=1}^{T} \underbrace{\text{Softmax}\big(a(\mathbf{s}_{t-1}, \mathbf{h}_i)\big)}_{\text{attention weights}} \mathbf{h}_i = \sum_{i=1}^{T} \alpha(\mathbf{s}_{t-1}, \mathbf{h}_i)\mathbf{h}_i$$

alignment score

Query Key Value

General Attention

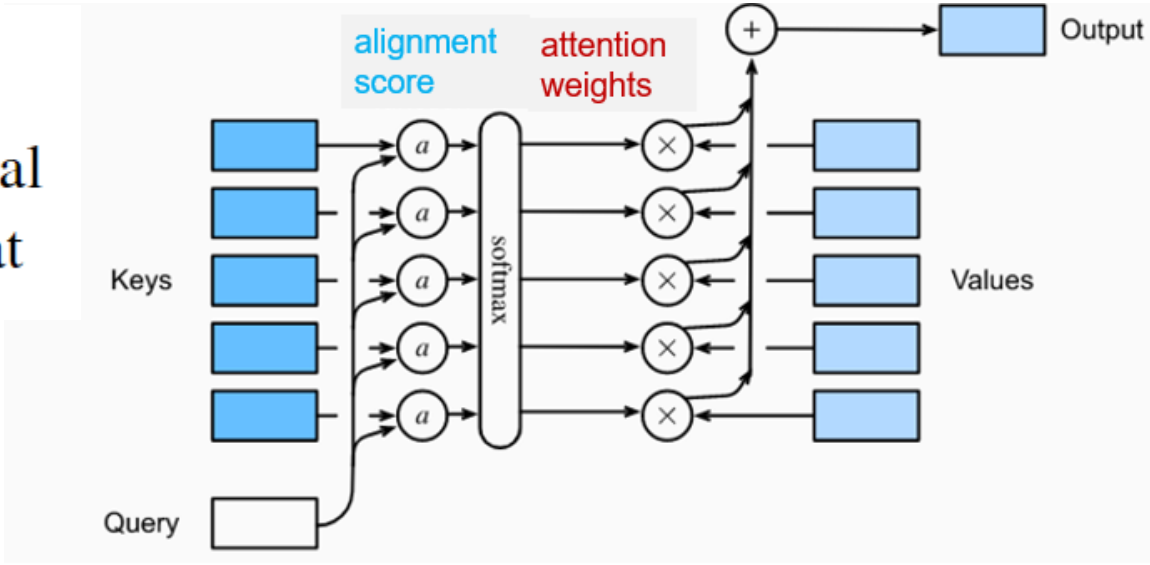
$$\mathbf{q} \in \mathbb{R}^{d_k}, \mathbf{k}_i \in \mathbb{R}^{d_k}, \mathbf{v}_i \in \mathbb{R}^{d_v}$$

Attention (q, K, V) for a query q and n key-value pairs:

$$\text{Attention}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = f(\mathbf{q}, (\mathbf{k}_1, \mathbf{v}_1) \ldots (\mathbf{k}_n, \mathbf{v}_n)) = \sum_{i=1}^{n} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i \in \mathbb{R}^{d_v}$$

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \text{softmax}(a(\mathbf{q}, \mathbf{k}_i) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{i=1}^{n} \exp(a(\mathbf{q}, \mathbf{k}_i))} \in \mathbb{R}$$

$$a(\mathbf{q}, \mathbf{k}_i) = \begin{cases} \mathbf{q}^T \mathbf{k}_i & \text{dot} \\ \mathbf{q}^T \mathbf{W} \mathbf{k}_i & \text{general} \\ \mathbf{v}^T \tanh(\mathbf{W}[\mathbf{k}_i; \mathbf{q}]) & \text{concat} \end{cases}$$



Scaled dot product attention

Attention (q, K, V) for a query q and n key-value pairs:

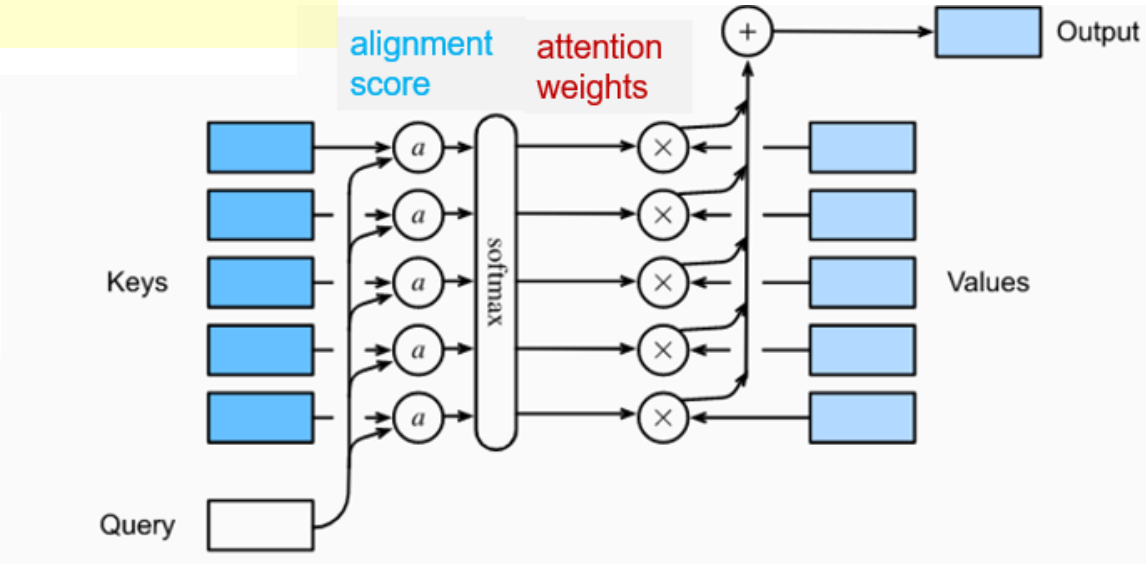
$$f(\mathbf{q}, (\mathbf{k}_1, \mathbf{v}_1) \ldots (\mathbf{k}_n, \mathbf{v}_n)) = \sum_{i=1}^{n} \alpha(\mathbf{q}, \mathbf{k}_i)\mathbf{v}_i \qquad \mathbf{q} \in \mathbb{R}^{d_k}, \mathbf{k}_i \in \mathbb{R}^{d_k}, \mathbf{v}_i \in \mathbb{R}^{d_v}$$

$$\alpha(\mathbf{q}, \mathbf{k}_i) = \mathrm{softmax}(a(\mathbf{q}, \mathbf{k}_i) = \frac{\exp(a(\mathbf{q}, \mathbf{k}_i))}{\sum_{i=1}^{n} \exp(a(\mathbf{q}, \mathbf{k}_i))} \in \mathbb{R}$$

$$a(\mathbf{q}, \mathbf{k}_i) = \frac{\mathbf{q}^T \mathbf{k}_i}{\sqrt{d_k}} \in \mathbb{R}, \quad \mathbf{q} \in \mathbb{R}^{d_k}, \mathbf{k}_i \in \mathbb{R}^{d_k}$$

$$\mathrm{Attention}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \mathrm{Softmax}(\frac{\mathbf{q}^T \mathbf{K}^T}{\sqrt{d_k}})\mathbf{V} \in \mathbb{R}^{d_v}$$

$$\mathbf{q} \in \mathbb{R}^{d_k}, \mathbf{K} \in \mathbb{R}^{n \times d_k}, \mathbf{V} \in \mathbb{R}^{n \times d_v}$$



Scaled dot product attention

- **Attention (q, k, v) for a query q and n key-value pairs :**

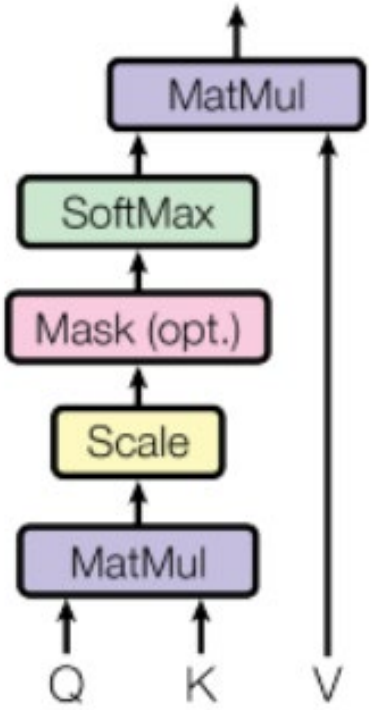
$$\text{Attention}(\mathbf{q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \in \mathbb{R}^{d_v}$$

$$\mathbf{q} \in \mathbb{R}^{d_k}, \mathbf{K} \in \mathbb{R}^{n \times d_k}, \mathbf{V} \in \mathbb{R}^{n \times d_v}$$

- **Attention for m Queries and n Key-Value pairs**

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \in \mathbb{R}^{m \times d_v}$$

$$\mathbf{Q} \in \mathbb{R}^{m \times d_k}, \mathbf{K} \in \mathbb{R}^{n \times d_k}, \mathbf{V} \in \mathbb{R}^{n \times d_v}$$



le daetr

<sos>
the
black
the
cat

$$\mathbf{QK}^\mathsf{T} = \underbrace{\begin{bmatrix} q_{11} & q_{12} & \dots & q_{1d_k} \\ q_{21} & q_{22} & \dots & q_{2d_k} \\ \vdots & \vdots & \ddots & \vdots \\ q_{m1} & q_{m2} & \dots & q_{md_k} \end{bmatrix}}_{\mathbf{Q}} \cdot \underbrace{\begin{bmatrix} k_{11} & k_{12} & \dots & k_{1n} \\ k_{21} & k_{22} & \dots & k_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ k_{d_k1} & k_{d_k2} & \dots & k_{d_kn} \end{bmatrix}}_{\mathbf{K}^\mathsf{T}} = \underbrace{\begin{bmatrix} e_{11} & e_{12} & \dots & e_{1n} \\ e_{21} & e_{22} & \dots & e_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{m1} & e_{m2} & \dots & e_{mn} \end{bmatrix}}_{\mathbf{QK}^\mathsf{T}}$$

attention to be paid on each french work

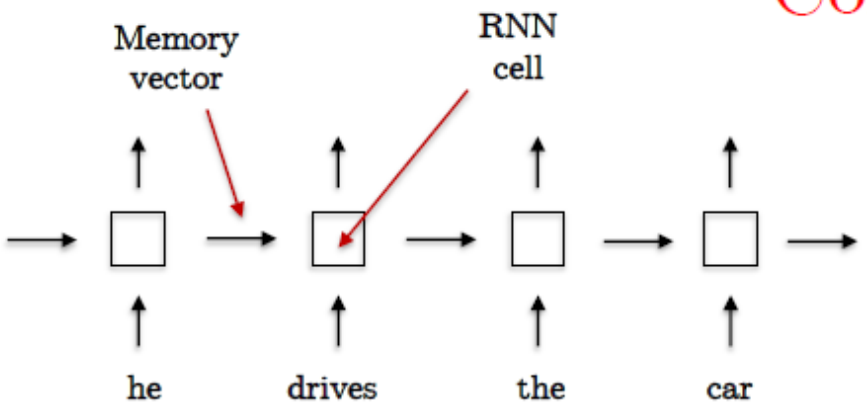
$$\text{softmax}(\frac{\mathbf{QK}^\mathsf{T}}{\sqrt{d_k}}) = \begin{bmatrix} \text{softmax}(e_{11}/\sqrt{d_k} & e_{12}/\sqrt{d_k} & \dots & e_{1n}/\sqrt{d_k}) \\ \text{softmax}(e_{21}/\sqrt{d_k} & e_{22}/\sqrt{d_k} & \dots & e_{2n}/\sqrt{d_k}) \\ \vdots & \vdots & \ddots & \vdots \\ \text{softmax}(e_{m1}/\sqrt{d_k} & e_{m2}/\sqrt{d_k} & \dots & e_{mn}/\sqrt{d_k}) \end{bmatrix}$$

$\Sigma = 1$

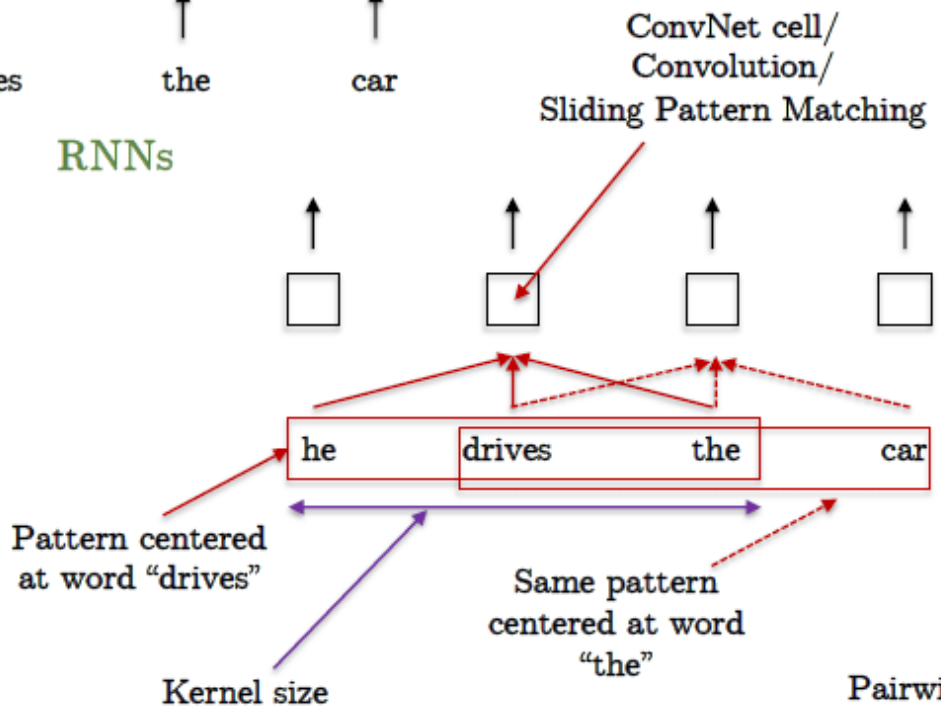
$$\text{softmax}(\frac{\mathbf{QK}^\mathsf{T}}{\sqrt{d_k}})\mathbf{V} = \begin{bmatrix} \text{softmax}(e_{11}/\sqrt{d_k} & e_{12}/\sqrt{d_k} & \dots & e_{1n}/\sqrt{d_k}) \\ \text{softmax}(e_{21}/\sqrt{d_k} & e_{22}/\sqrt{d_k} & \dots & e_{2n}/\sqrt{d_k}) \\ \vdots & \vdots & \ddots & \vdots \\ \text{softmax}(e_{m1}/\sqrt{d_k} & e_{m2}/\sqrt{d_k} & \dots & e_{mn}/\sqrt{d_k}) \end{bmatrix} \cdot \begin{bmatrix} v_{11} & v_{12} & \dots & v_{1d_v} \\ v_{21} & v_{22} & \dots & v_{2d_v} \\ \vdots & \vdots & \ddots & \vdots \\ v_{n1} & v_{n2} & \dots & v_{nd_v} \end{bmatrix} = \mathbf{A} \in \mathbb{R}^{m \times d_v}$$

Each row represents a word-embedded french word

Comparing Layers

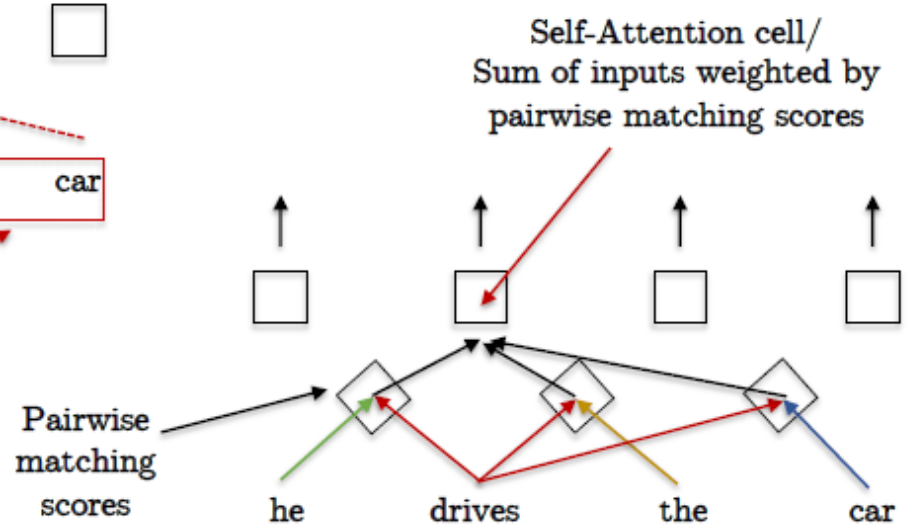


RNNs



ConvNets

All these layers can learn representation of sequences of different length/size and different context.



ANNs

Computational Cost

- RNN layer : $O(n.d^2)$
- ConvNet layer : $O(n.d^2.k)$
- Transformer layer : $O(n^2.d)$

Seems scary !

with n : sequence length, d : hidden feature size, k : kernel size

- Attention networks have actually less parameters as long as $d \geq n$!
  - Example 1 : n=100, d=1000, k=3

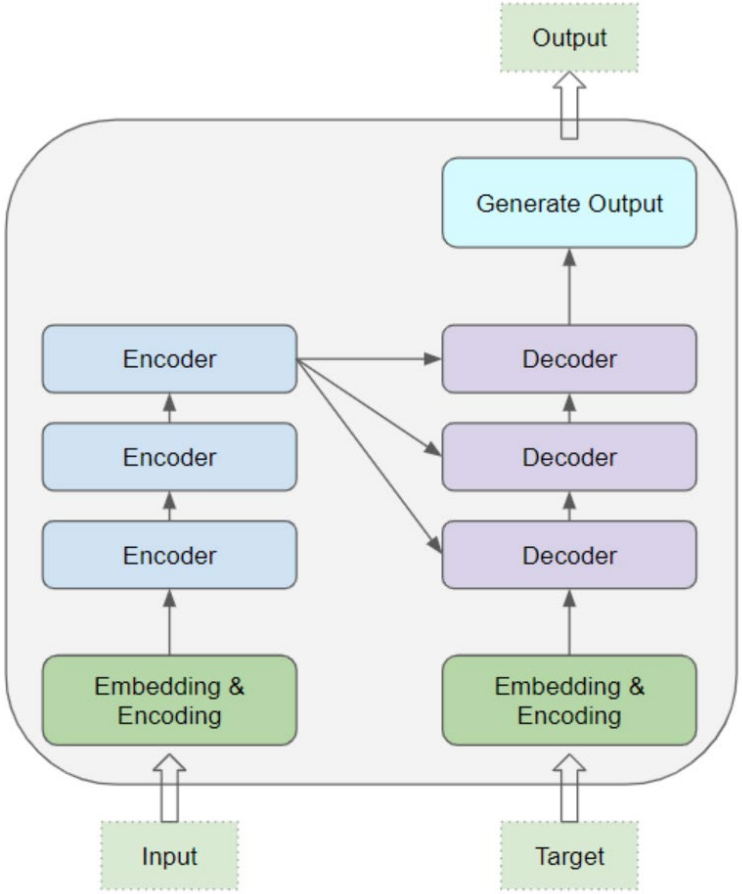
    RNN: $O(10^8)$, ConvNet: $O(3.10^8)$, Transformer: $O(10^7)$
  - Example 2 : n=1000, d=1000, k=3

    RNN: $O(10^9)$, ConvNet: $O(3.10^9)$, Transformer: $O(10^9)$

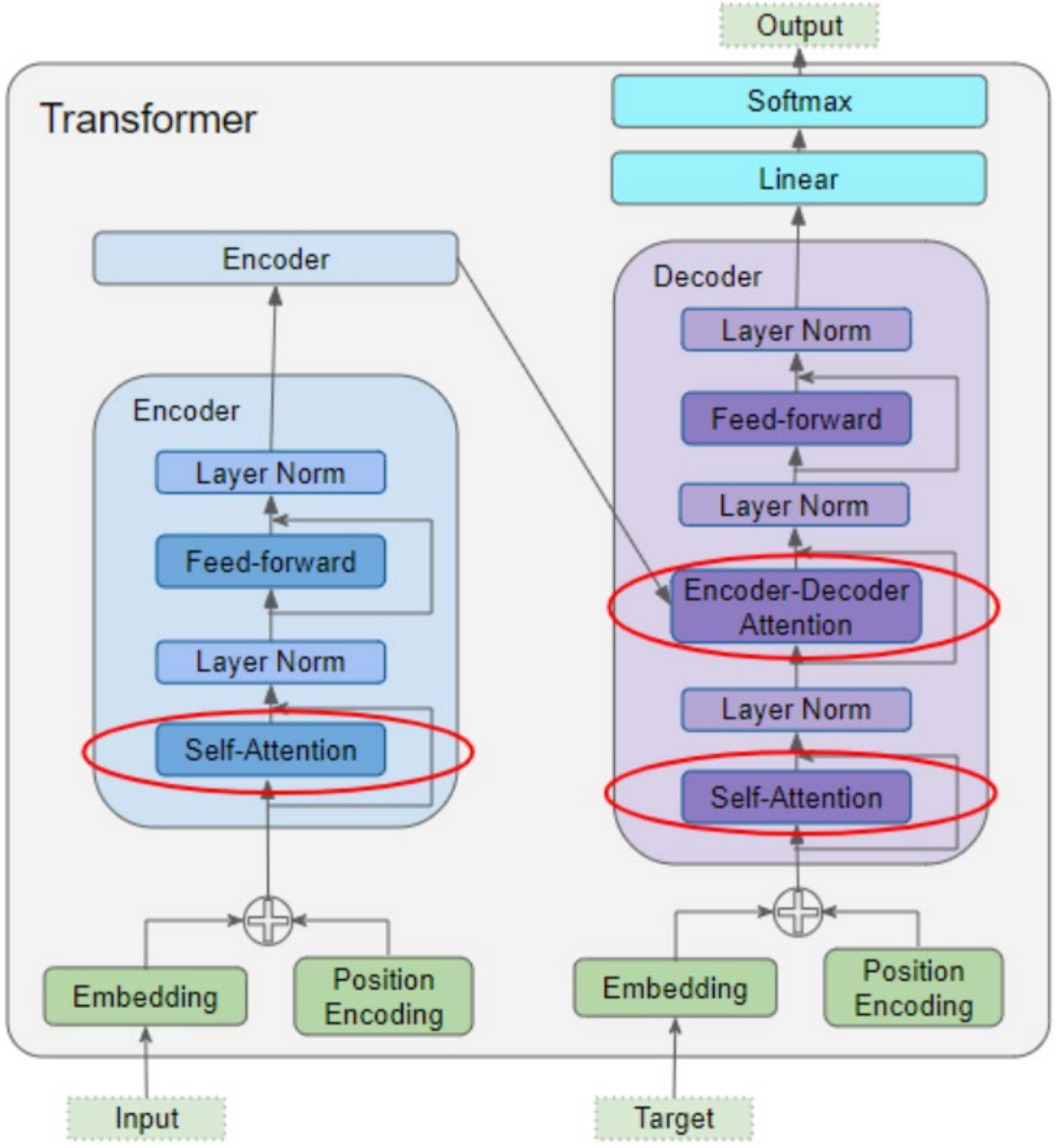
Outline

- An introduction on Attention
- General Attention
- **How attention is used in Transformer**
- Understanding Transformer
- Why Transformer works?
- Implement Transformer for Language Model
- Implement Transformer for Seq2Seq Model
- Conclusion

Transformer Overview

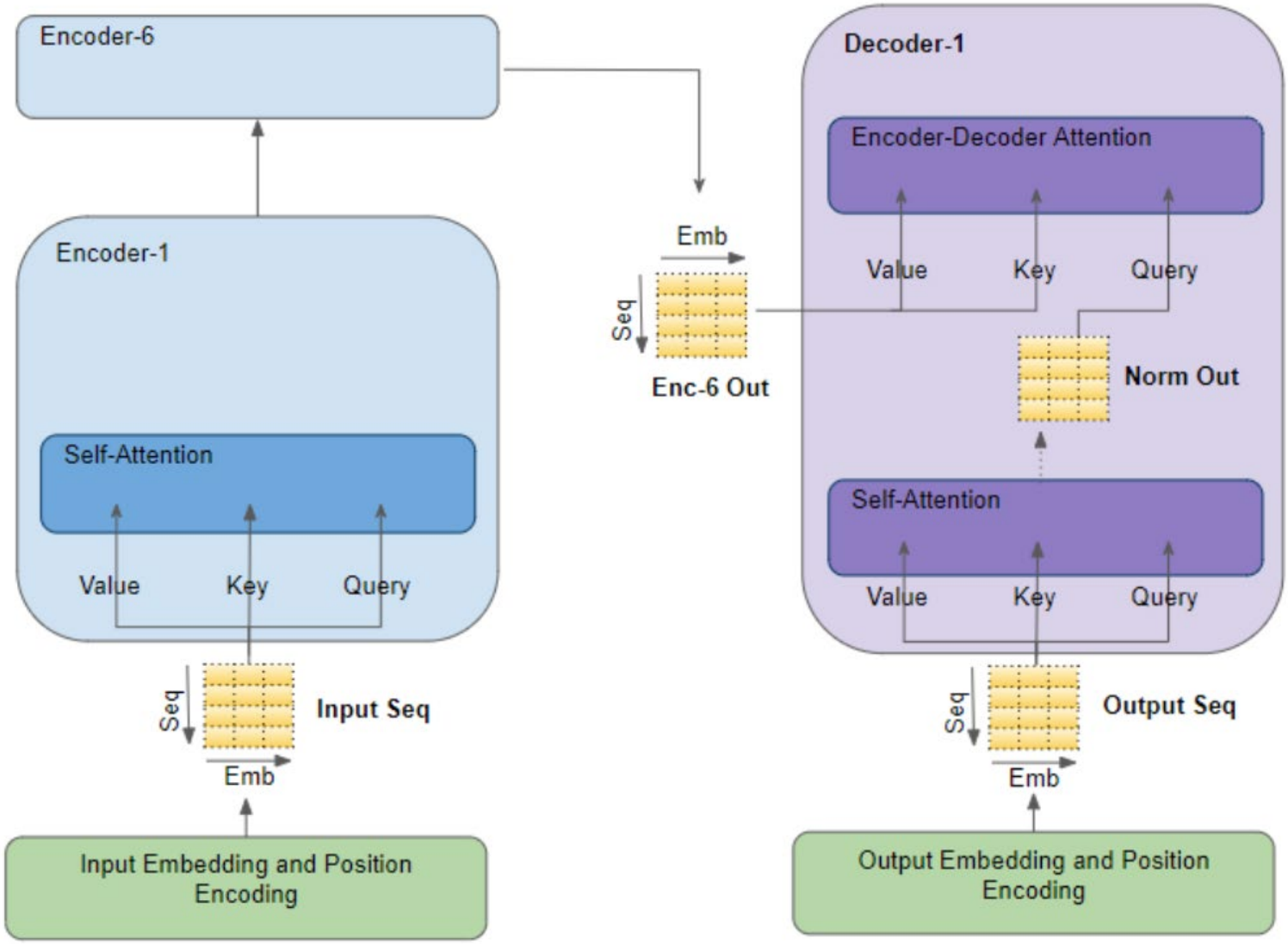


Shifted Right, i.e.
<START> + Target

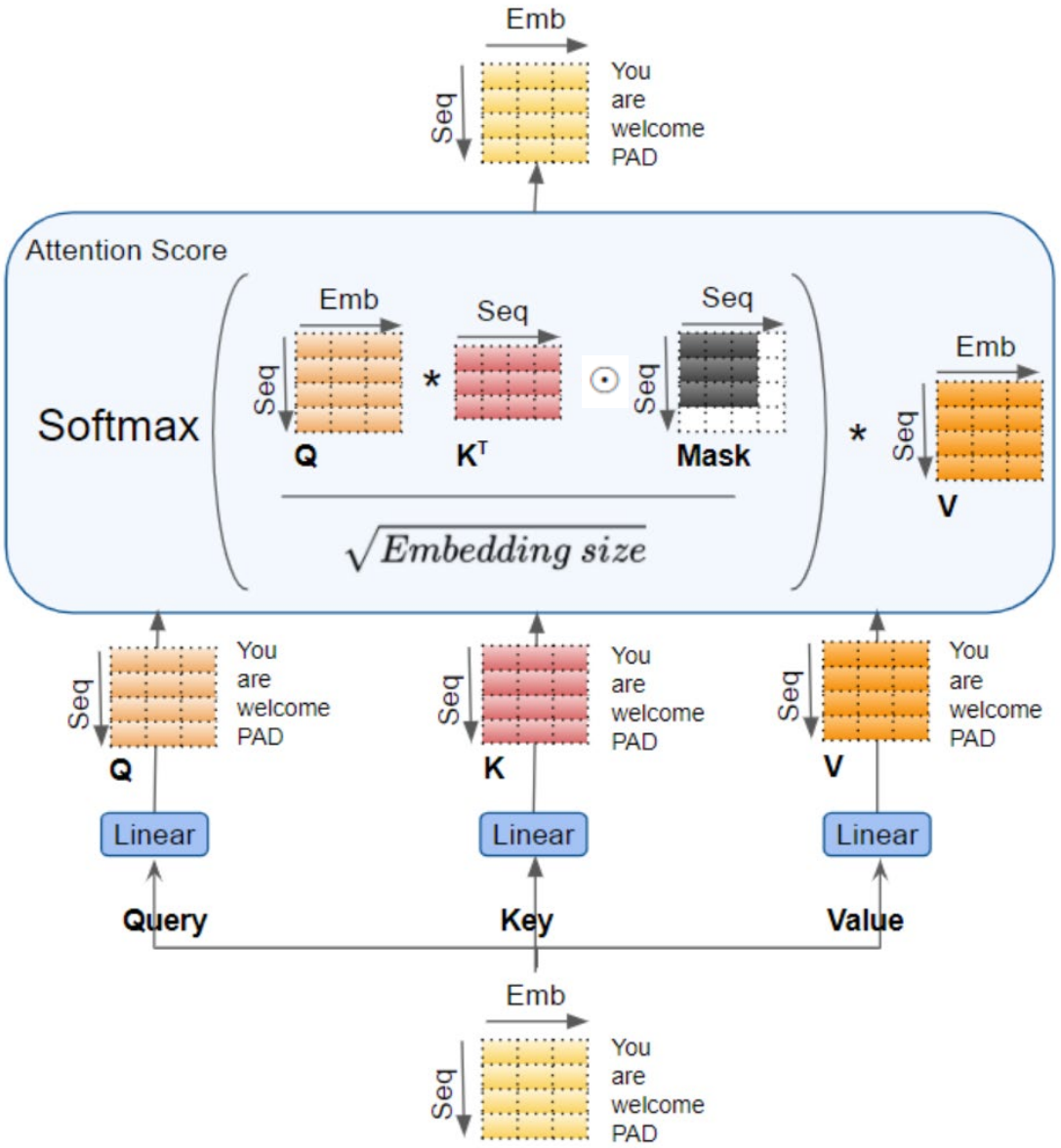


Shifted Right

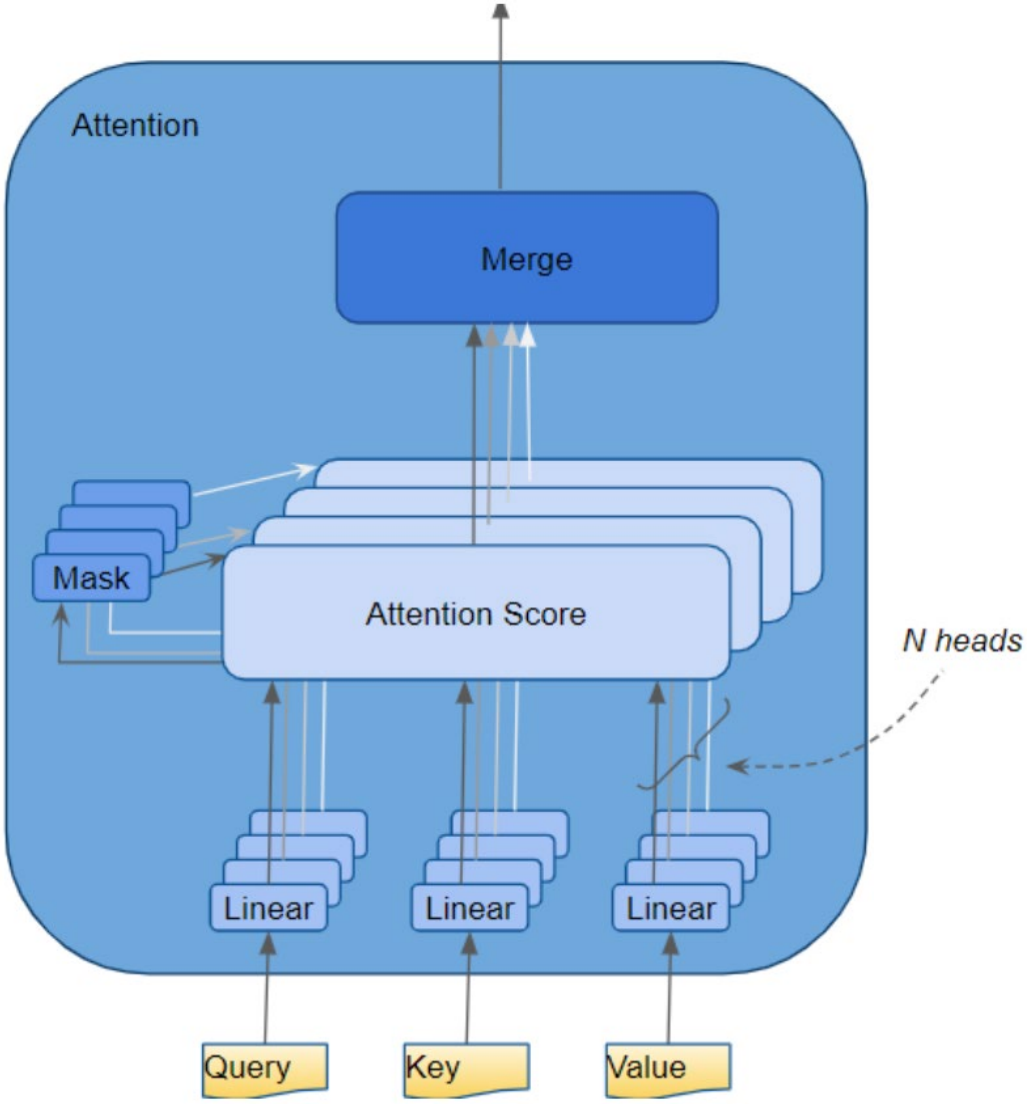
Encoder and Decoder with Attention



Scaled dot product attention

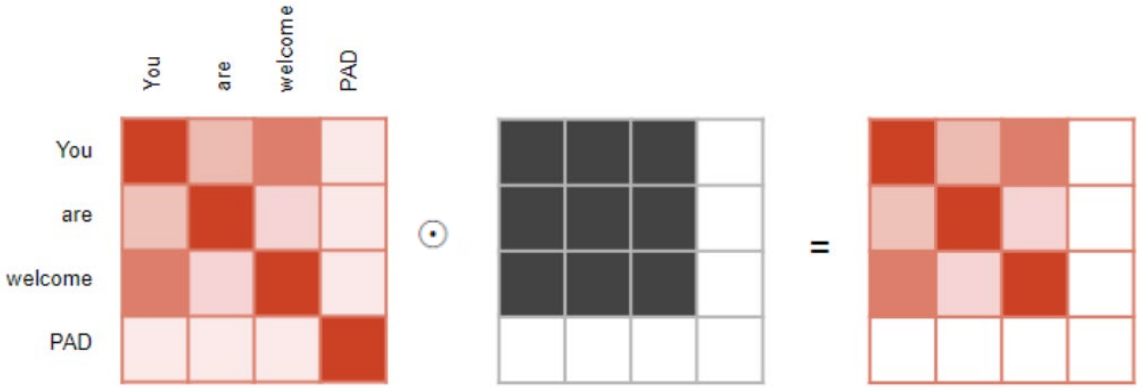


Multi-Head Attention

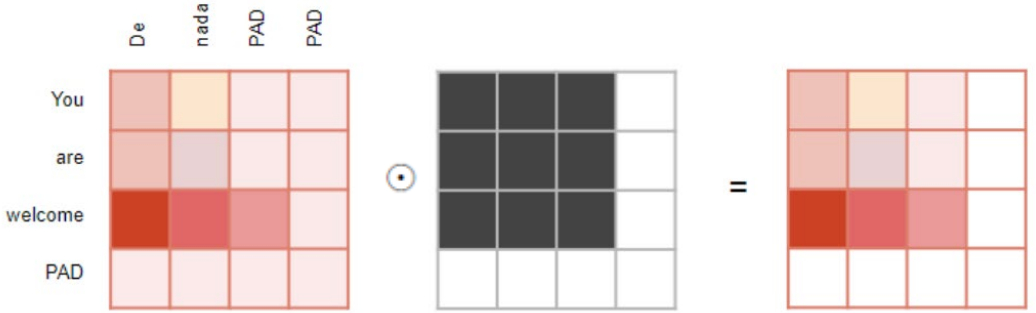


Attention Mask

- **In the Encoder Self-attention and in the Encoder-Decoder-attention:**
  - zero attention outputs for padding in the input sentences
  - ensure that padding doesn't contribute to the self-attention



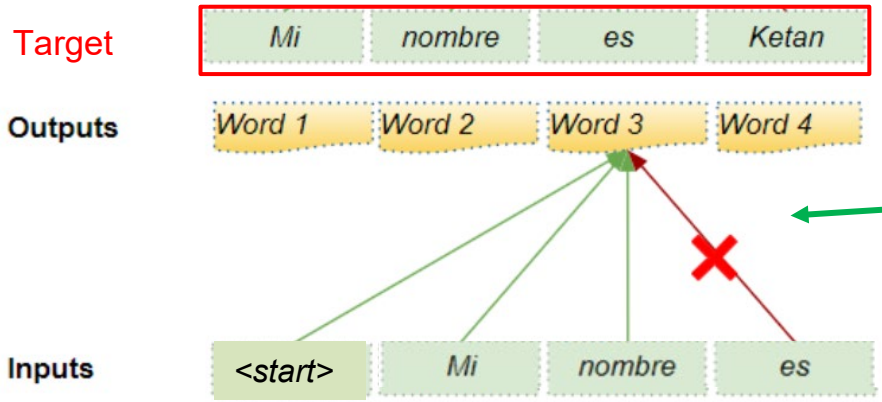
Encoder Self-Attention Scores



Encoder-Decoder Attention Scores

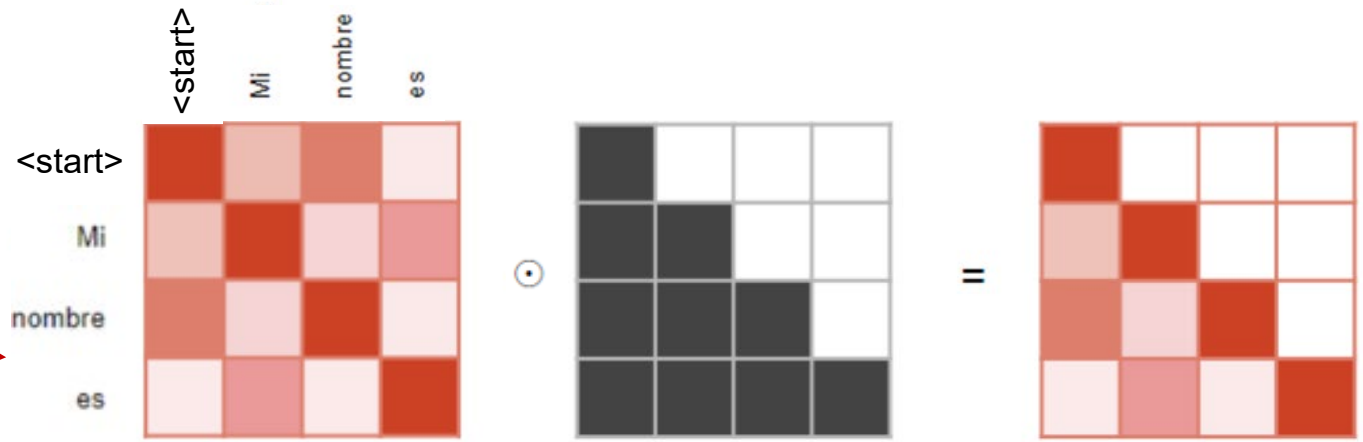
Attention Mask

- **In the Decoder Self-attention:**
  - Prevent decoder "peeking" ahead at the rest of target sentence when predicting the next word



when predicting "Word 3", the decoder should only refer to the first 3 input words

Therefore, the Decoder masks out input words that appear later in the sequence.

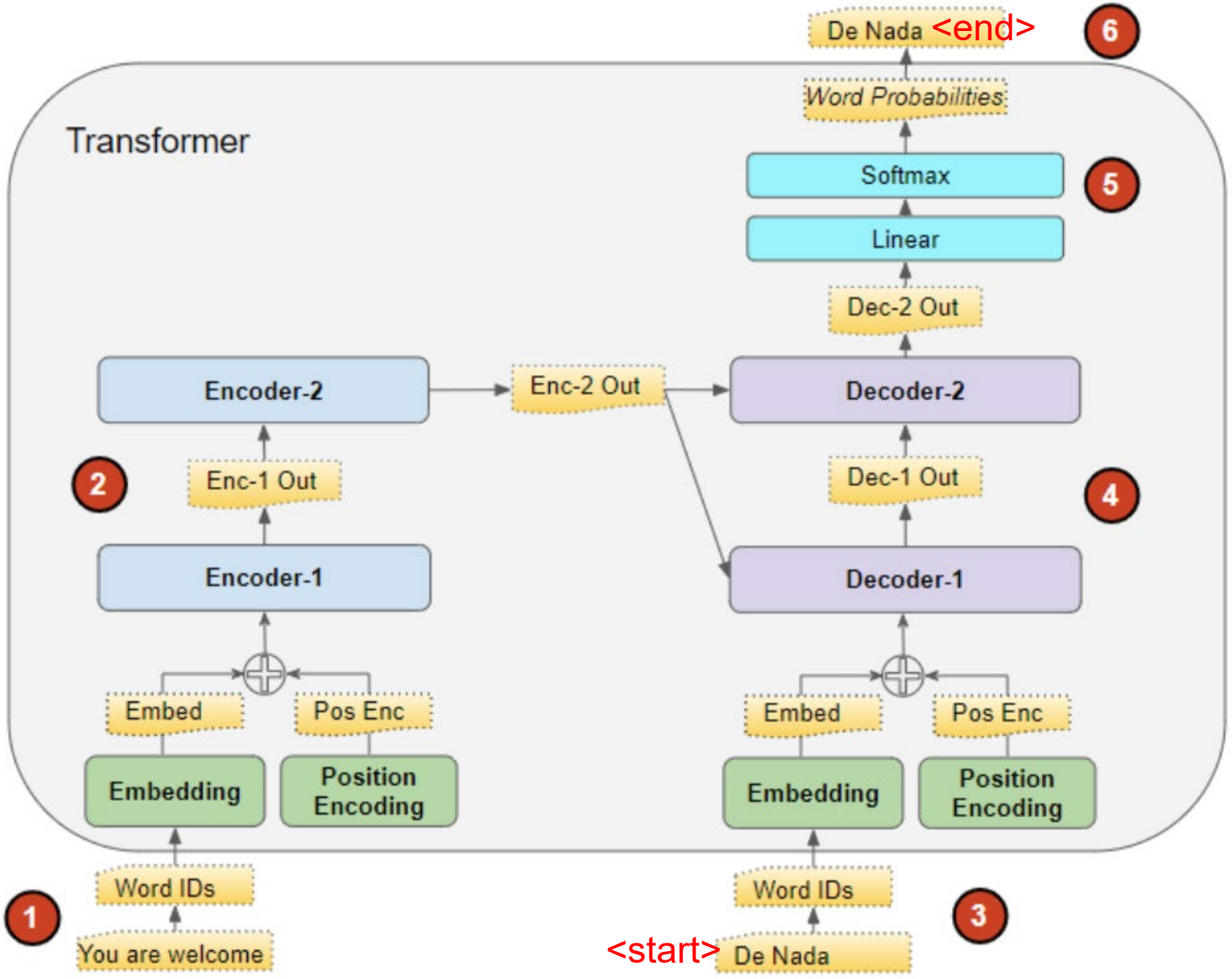


Decoder Self-Attention Scores

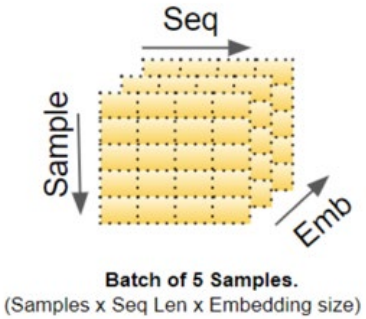
Outline

- An introduction on Attention
- General Attention
- How attention is used in Transformer
- **Understanding Transformer**
- Why Transformer works?
- Implement Transformer for Language Model
- Implement Transformer for Seq2Seq Model
- Conclusion

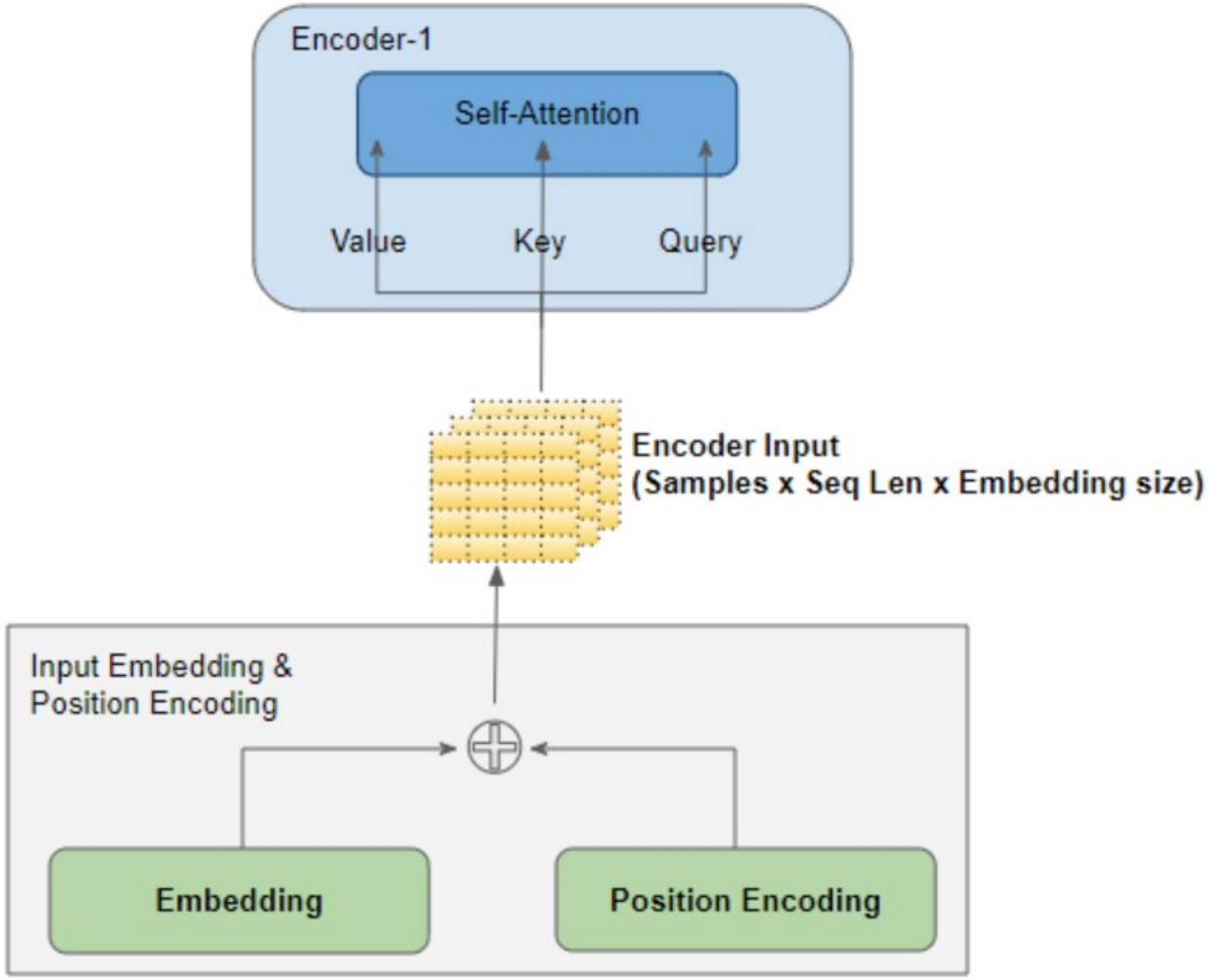
Overview


De Nada <end>
6
Word Probabilities
Transformer
Softmax
5
Linear
Dec-2 Out
Encoder-2
Enc-2 Out
Decoder-2
2
Enc-1 Out
Dec-1 Out
4
Encoder-1
Decoder-1
Embed
Pos Enc
Embed
Pos Enc
Embedding
Position Encoding
Embedding
Position Encoding
Word IDs
Word IDs
1
3
You are welcome
<start> De Nada

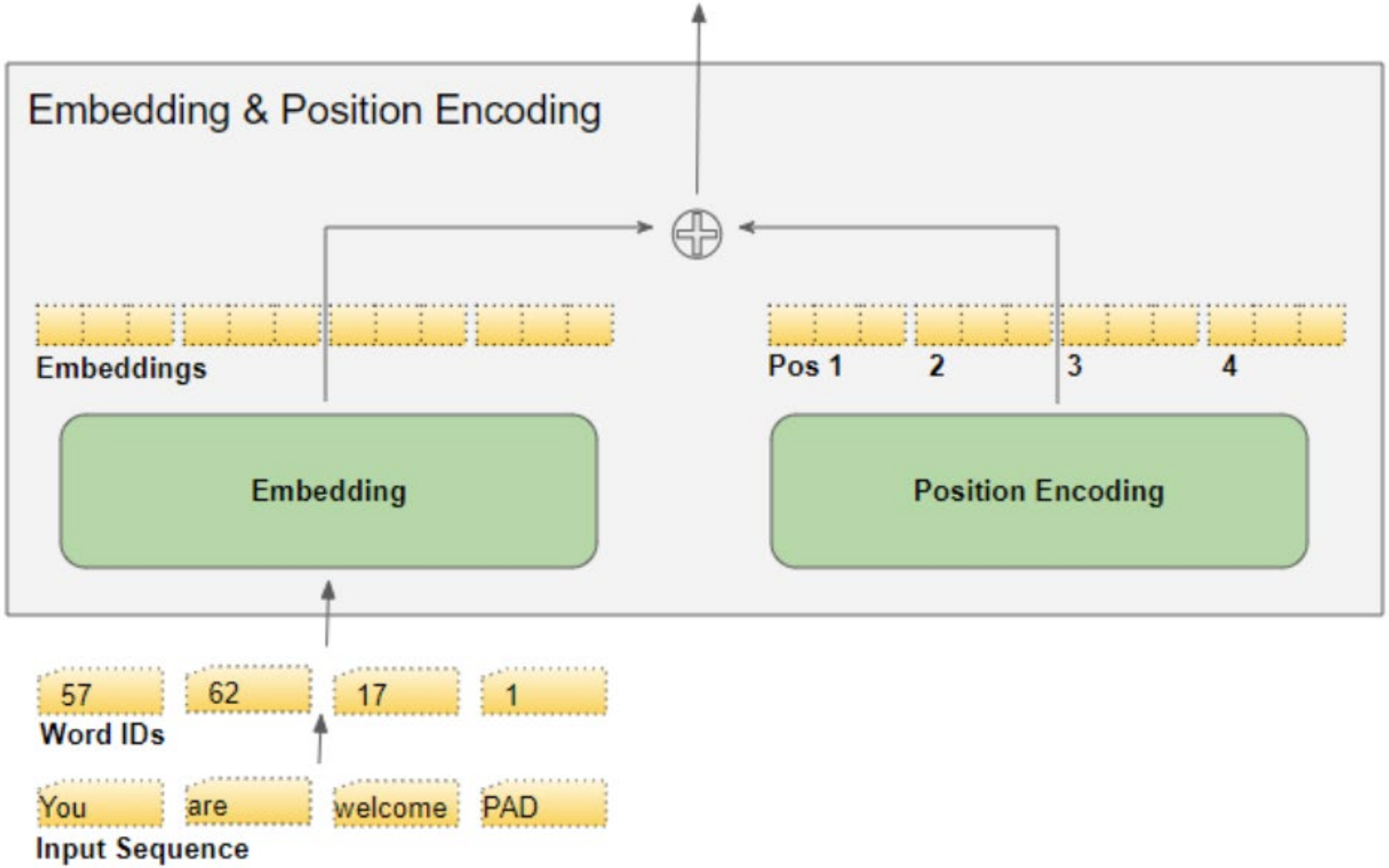
1: Input Layer (Encoder)



Batch of 5 Samples.
(Samples x Seq Len x Embedding size)



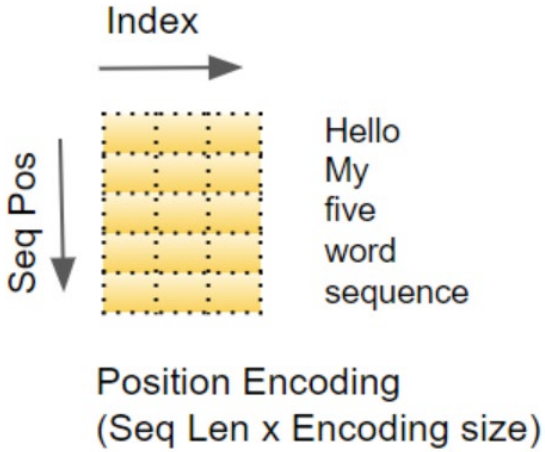
Embedding


Embedding & Position Encoding
Embeddings
Pos 1
2
3
4
Embedding
Position Encoding
57
62
17
1
Word IDs
You
are
welcome
PAD
Input Sequence

Positional Encoding

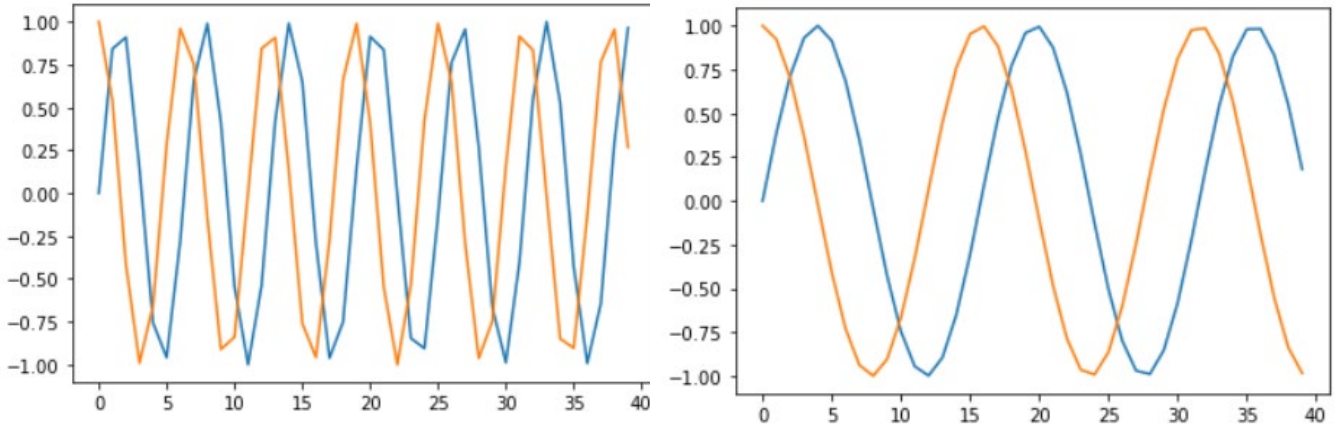
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

- *pos* is the position of the word in the sequence
- *d_model* is the length of the encoding vector (same as the embedding vector) and
- 2i (even index) and 2i +1(odd index) is the index position into this vector

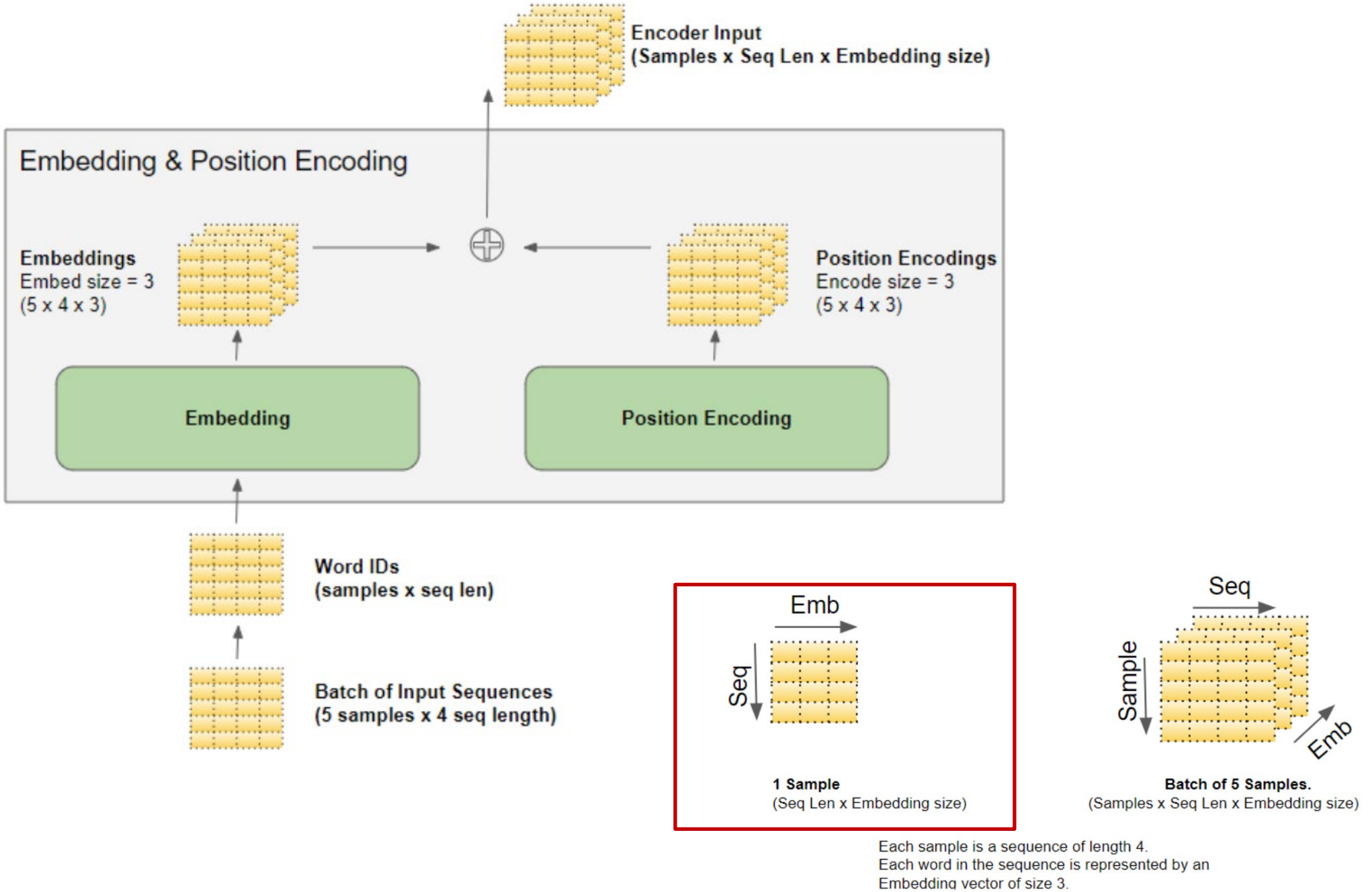


Example

- encode a sequence of 40 words into a vector of d =20
- show encoding values d_model
  - i =0: 0th index and 1st index
  - i =1: 2nd index and 3rd index

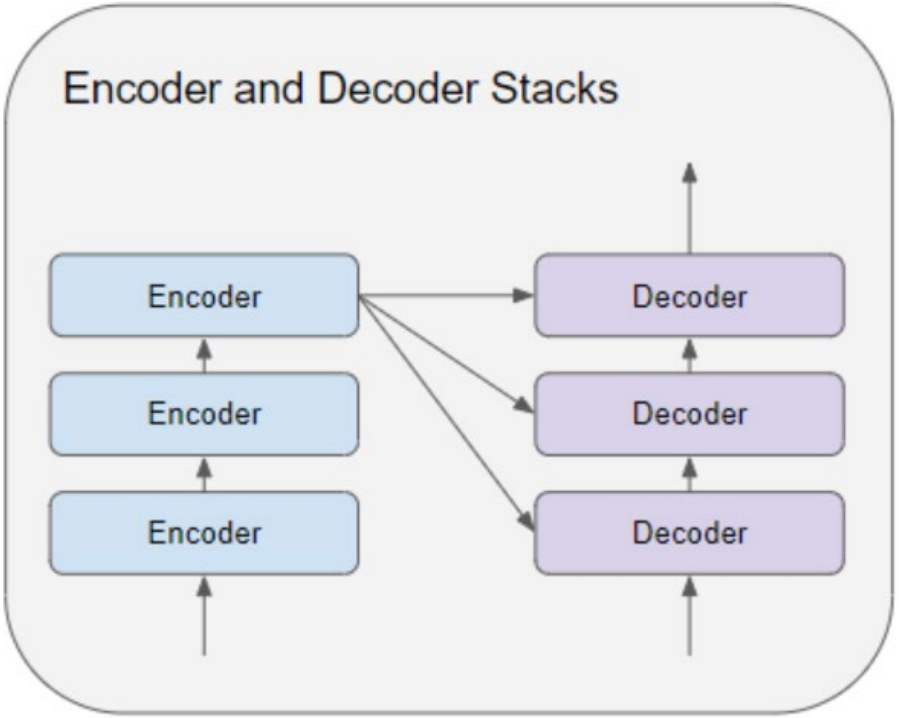


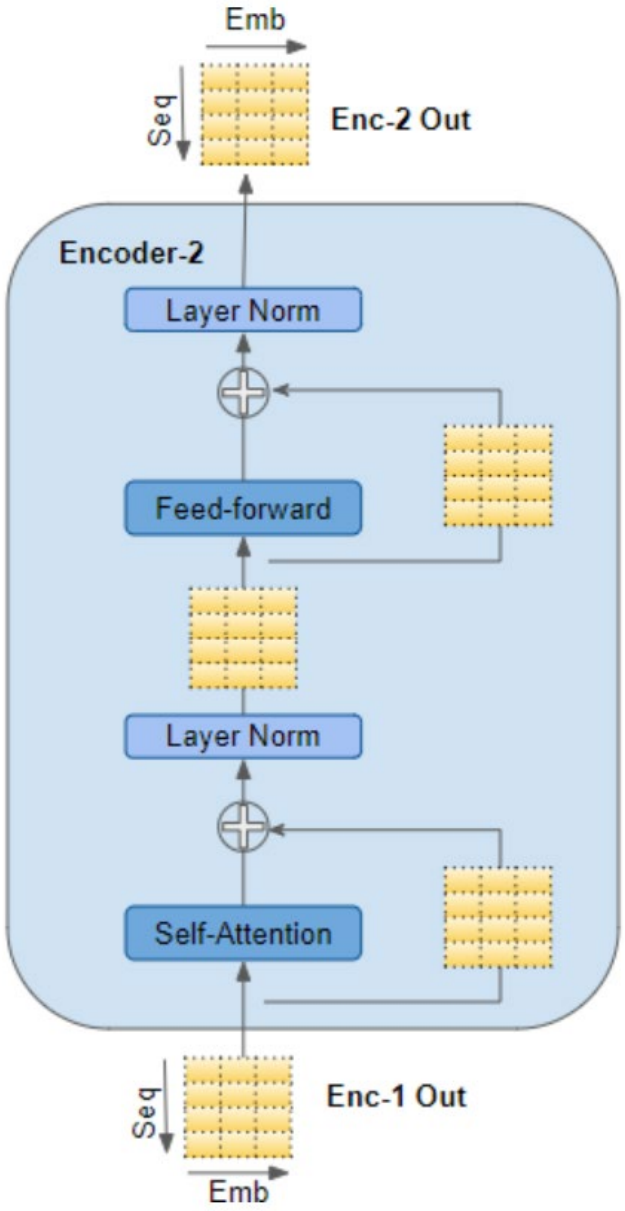
Matrix Dimensions



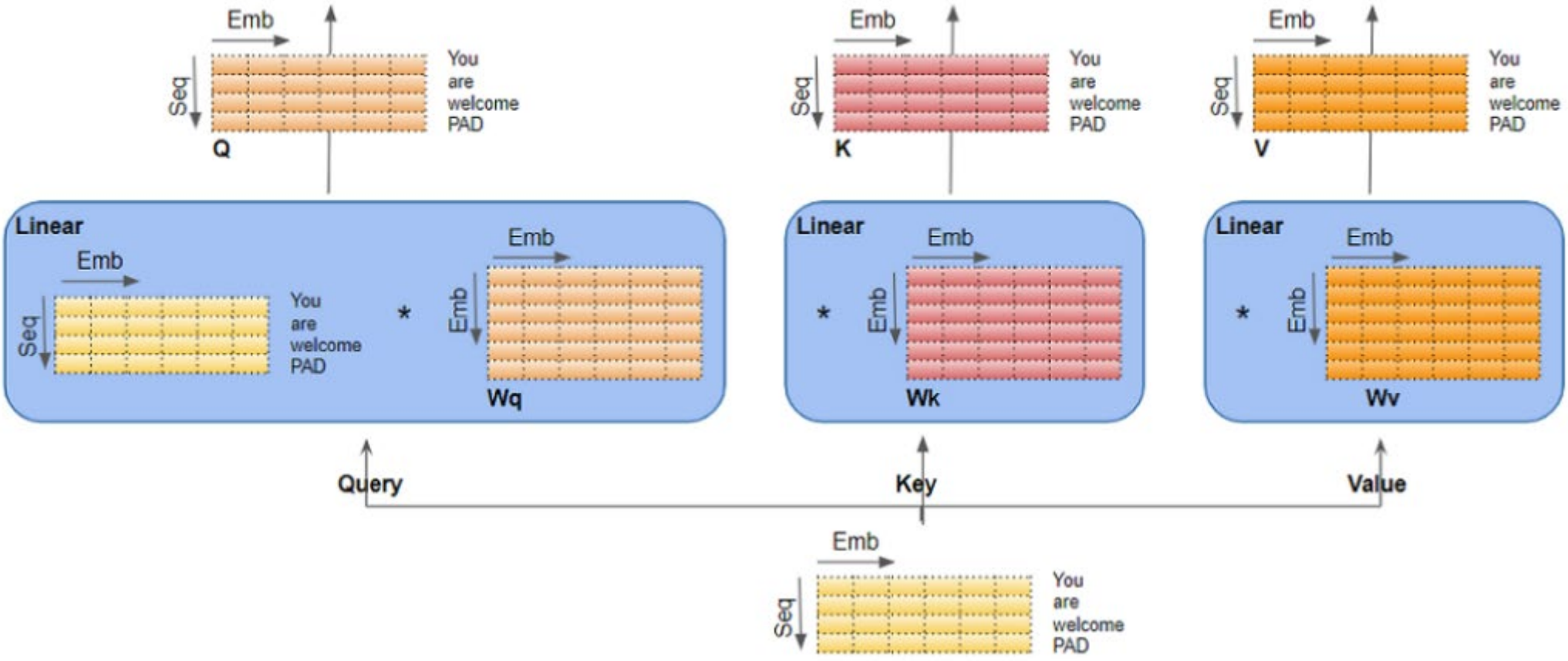
Each sample is a sequence of length 4.
Each word in the sequence is represented by an Embedding vector of size 3.

2. Encoder

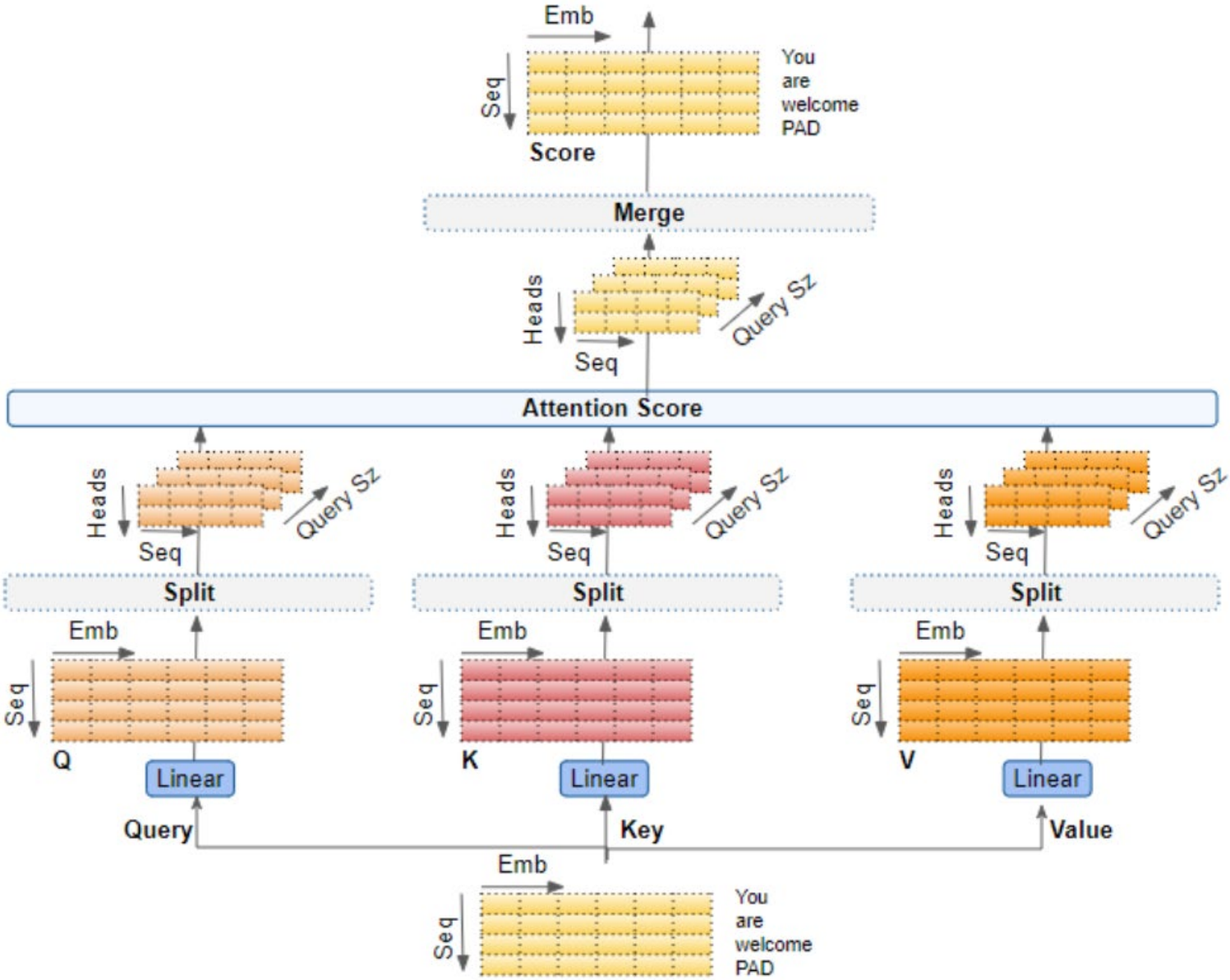

Encoder and Decoder Stacks
Encoder
Encoder
Encoder
Decoder
Decoder
Decoder


Emb
Seq
Enc-2 Out
Encoder-2
Layer Norm
Feed-forward
Layer Norm
Self-Attention
Seq
Enc-1 Out
Emb

2a: Linear Layers



2b. Multi-head Attention

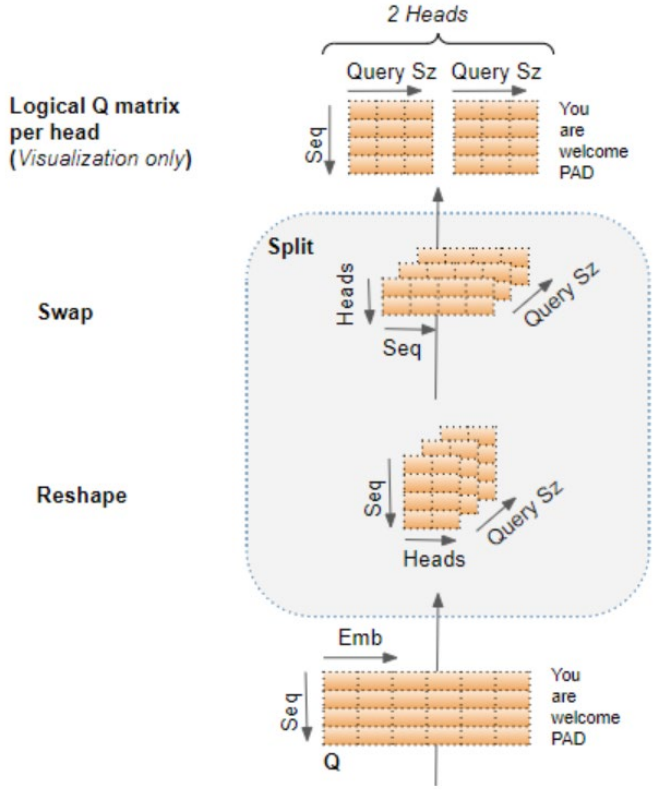


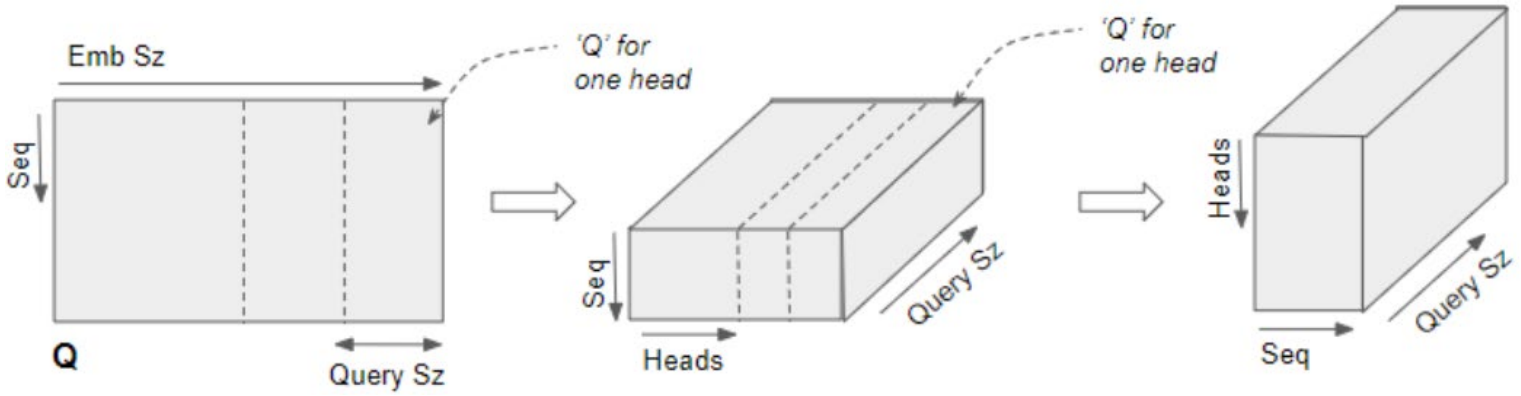
Splitting data across Attention heads

Query Size = Embedding Size / Number of heads

Q, K, V matrices are not physically split into multiple matrices.

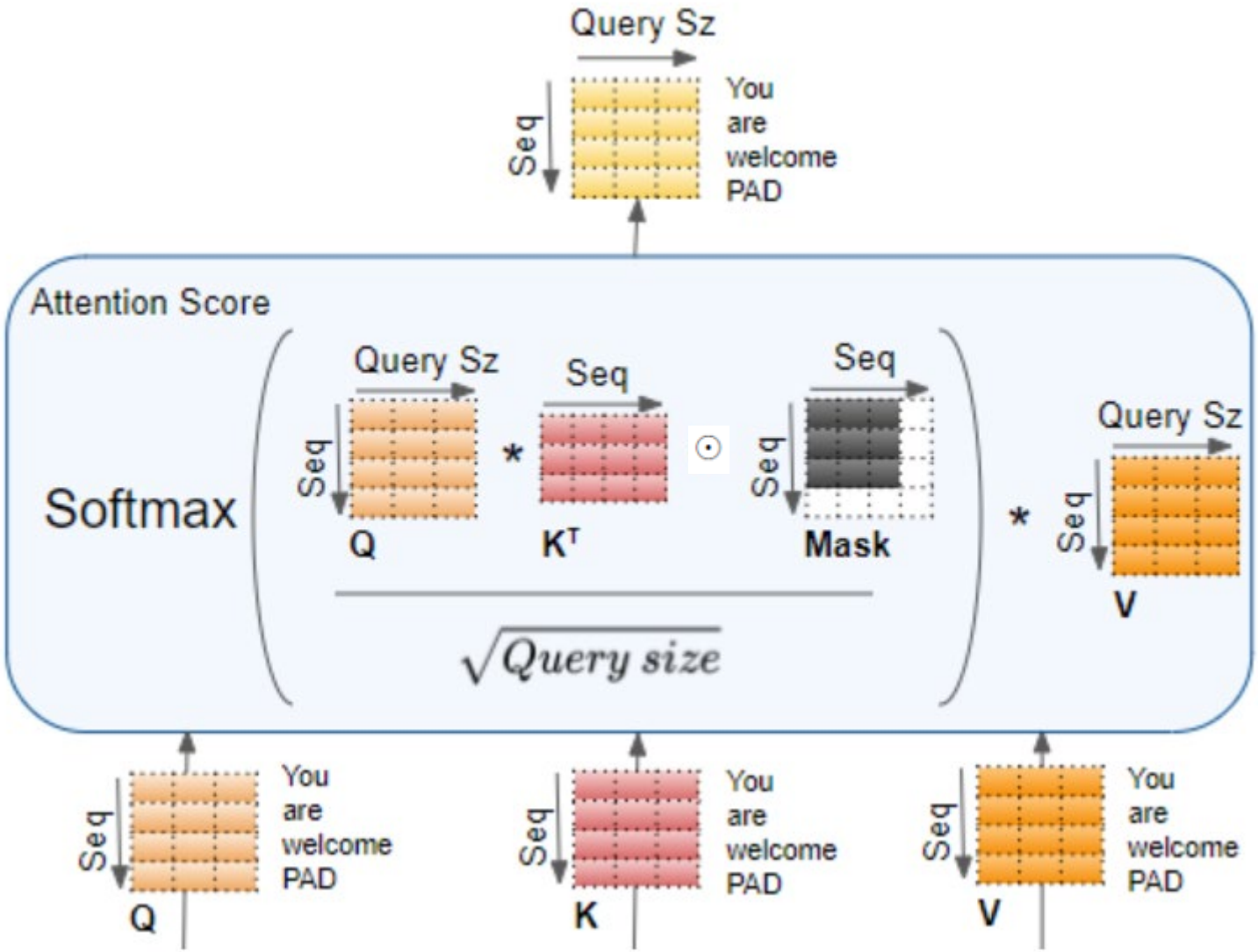
A single matrix is used for Q, K, V respectively, with logically separate sections of the matrix for each Attention head.



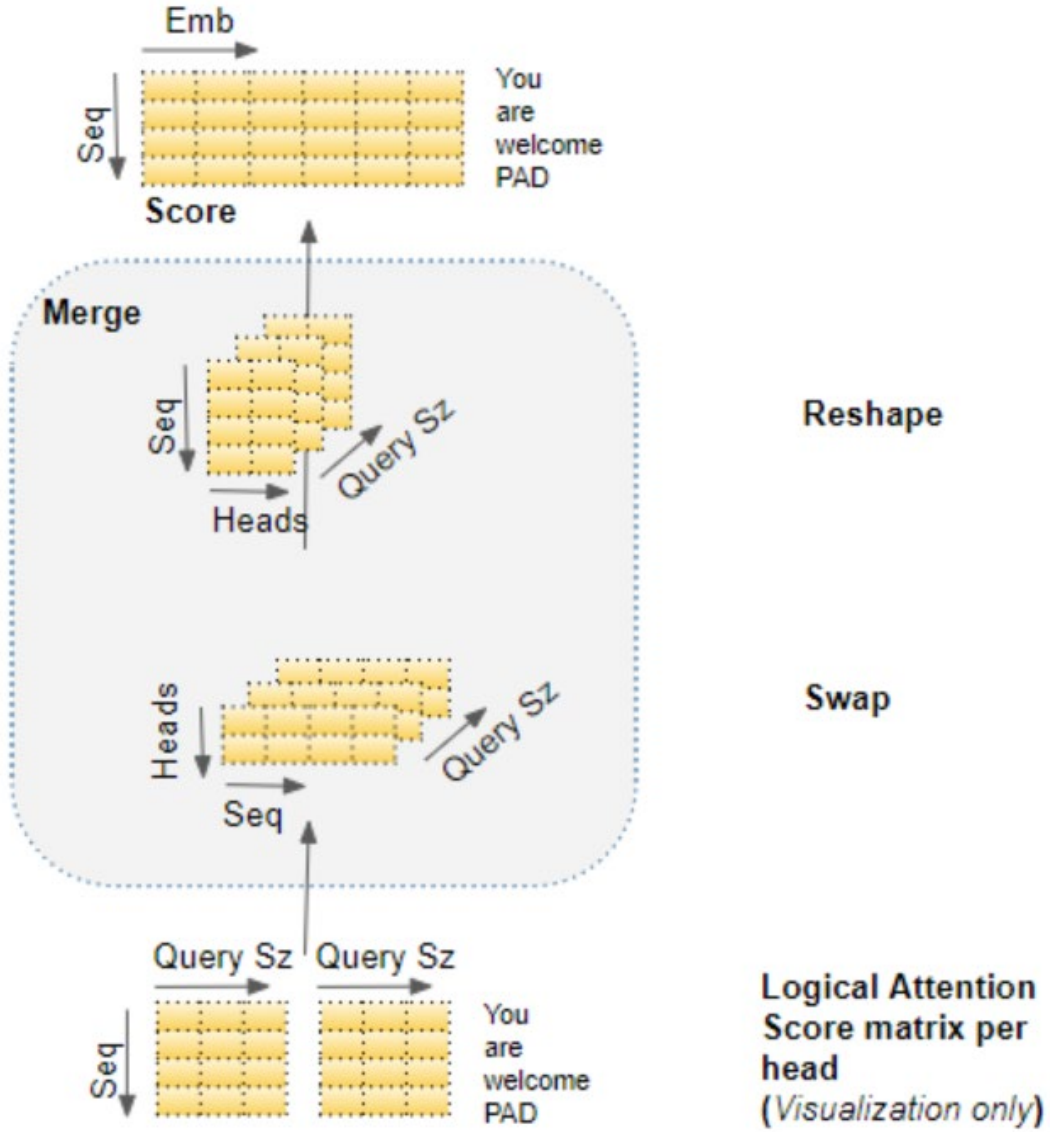


Reshaping Q, K, V matrices to include an explicit Head dimension

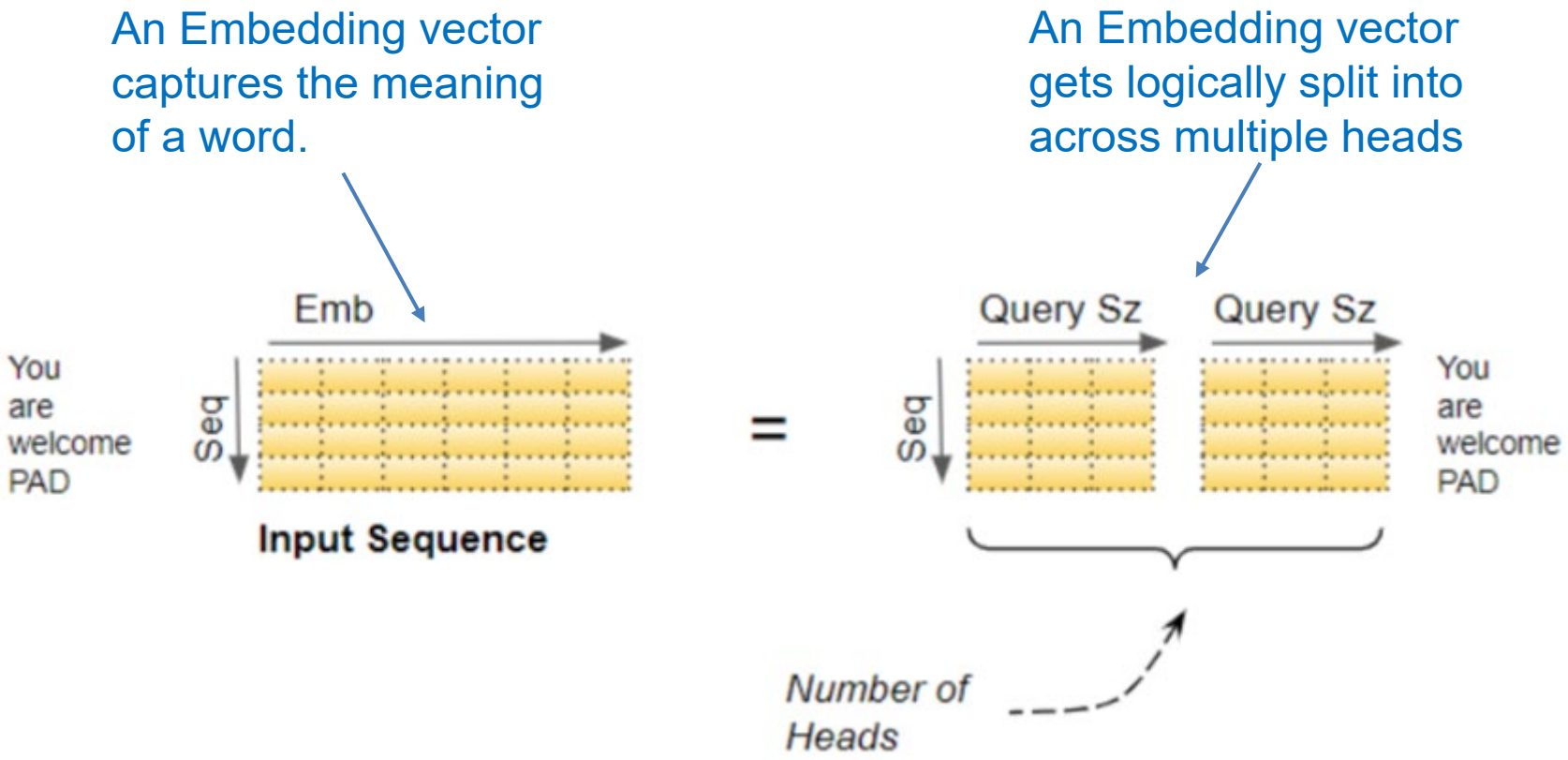
Compute the Attention Score for each head



Merge each Head's Attention Scores together



Multi-head split captures richer interpretations



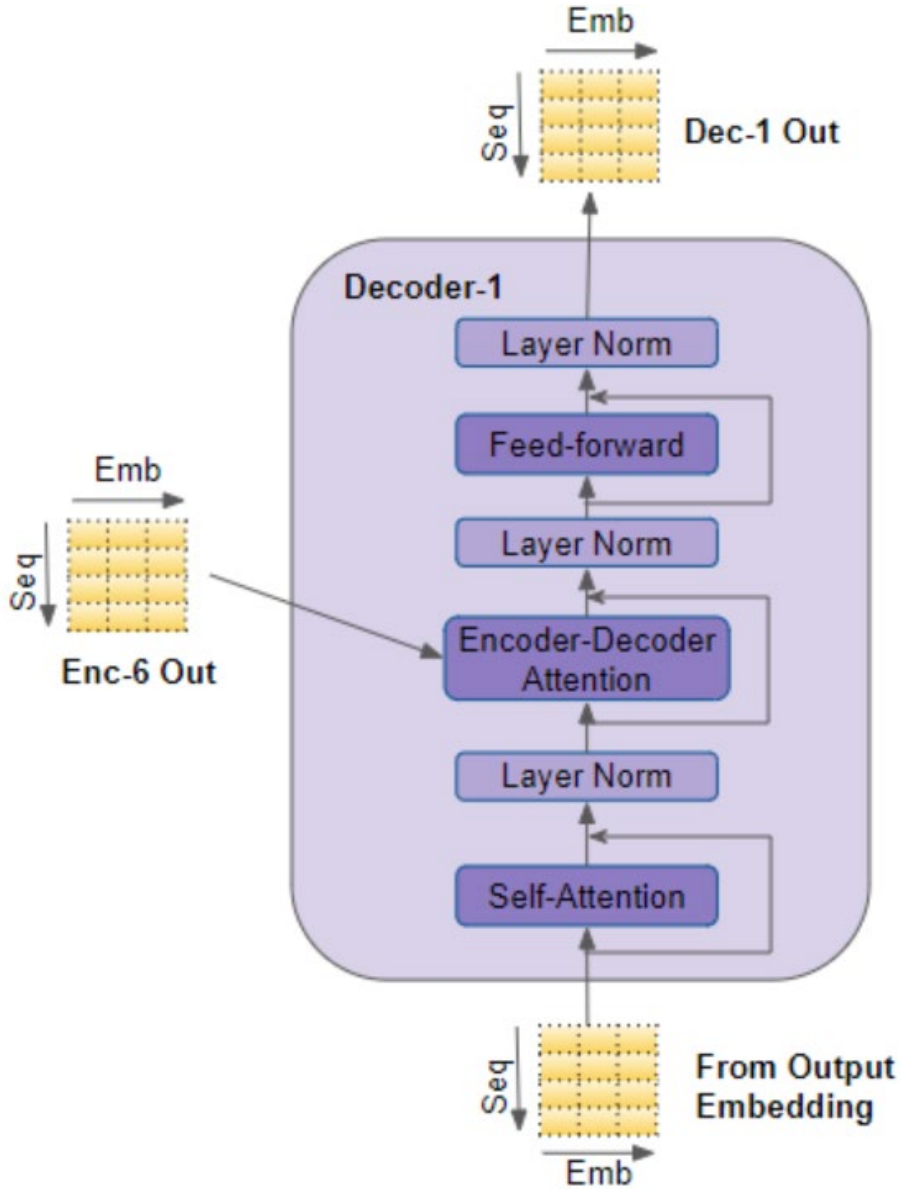
- Separate sections of the Embedding can learn different aspects of the meanings of each word, as it relates other words in the sequence
- Allows the Transformer to capture richer interpretations of the sequence.

3. Input Layer (Decoder)

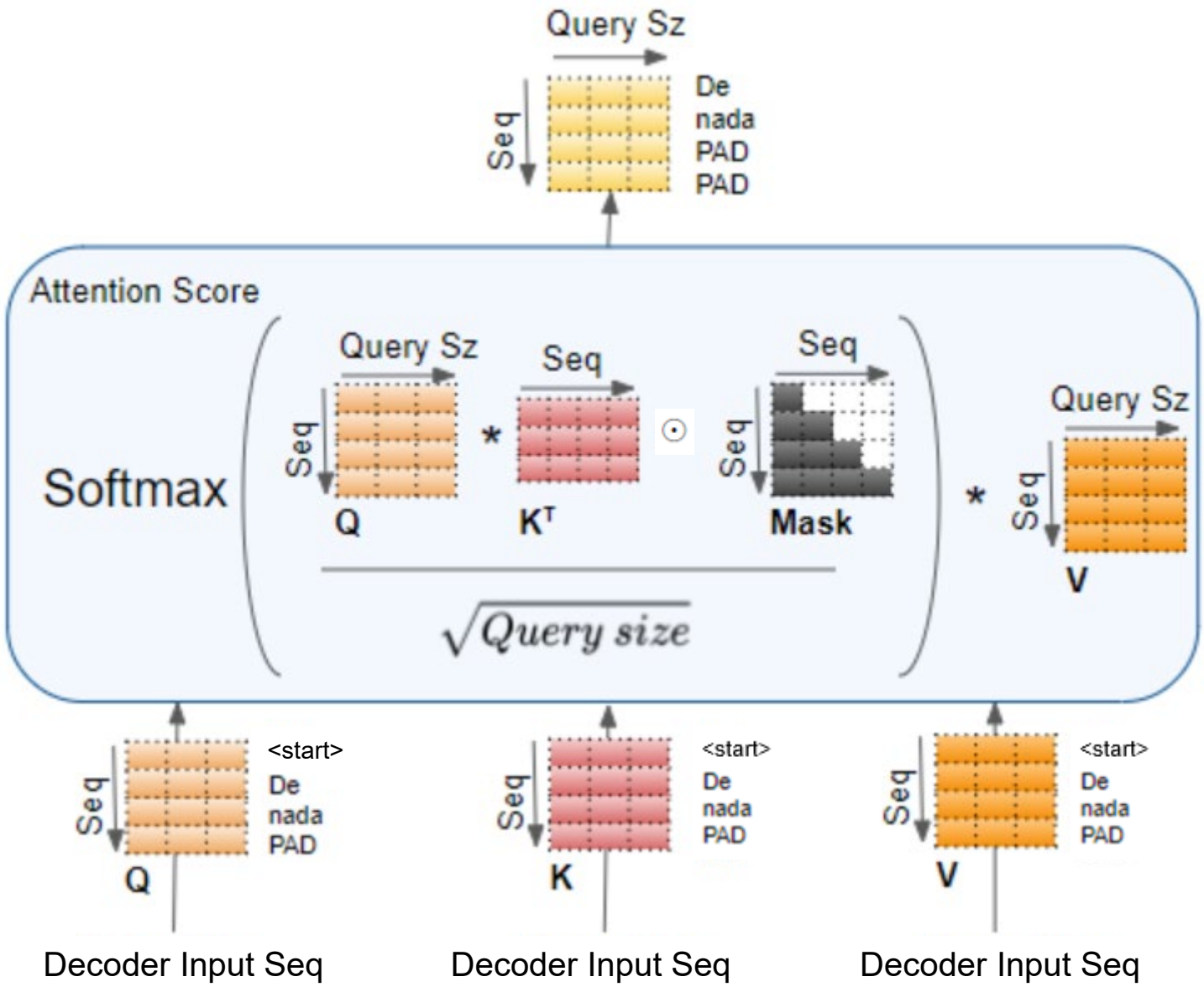
Teacher Forcing:

- feeding the target sequence to the Decoder during training.
- Transformer is able to output all the words in parallel without looping
- greatly speed up training

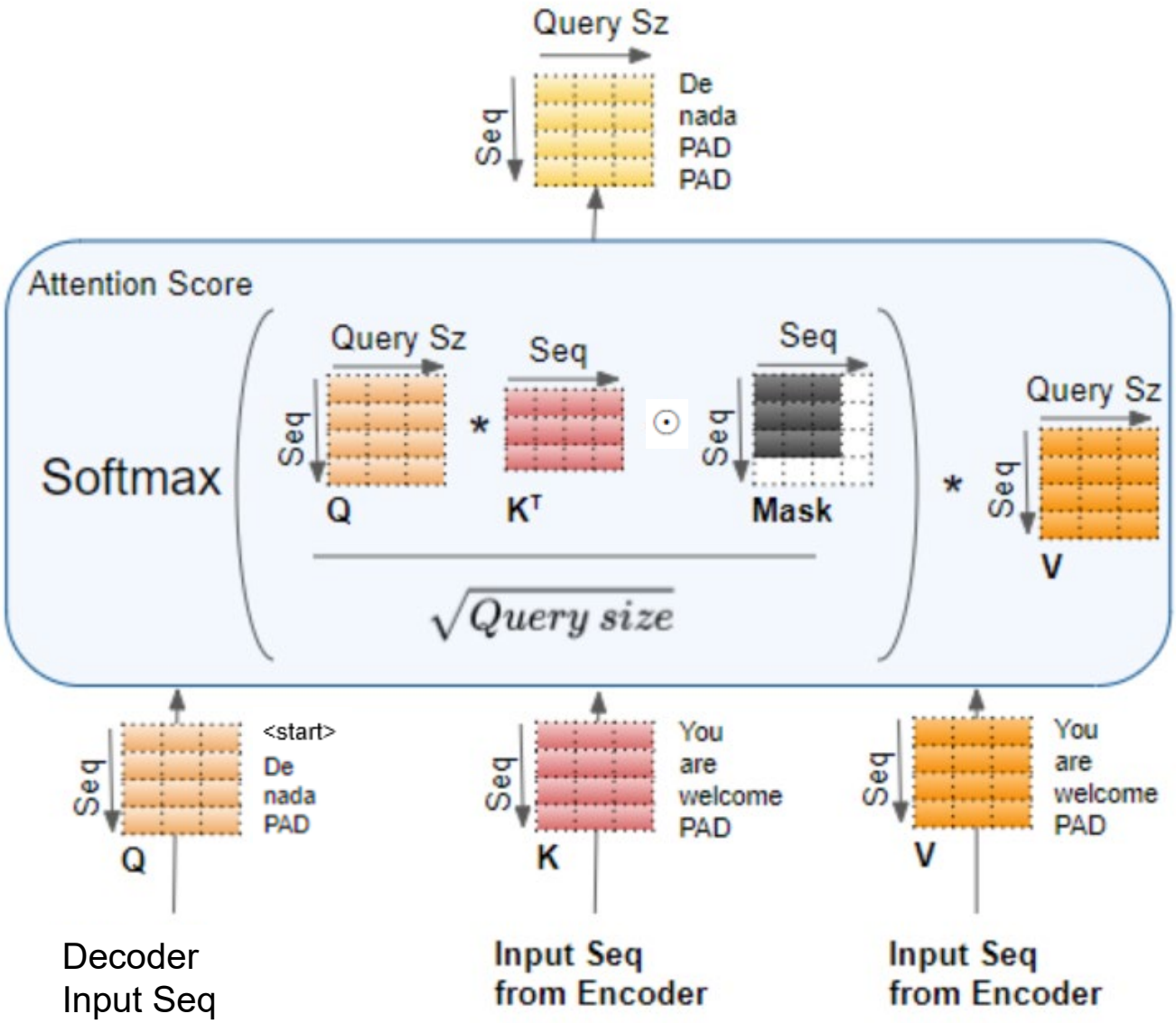
4. Decoder


Emb
Seq
Dec-1 Out
Decoder-1
Layer Norm
Feed-forward
Layer Norm
Encoder-Decoder Attention
Layer Norm
Self-Attention
Emb
Seq
Enc-6 Out
Seq
From Output Embedding
Emb

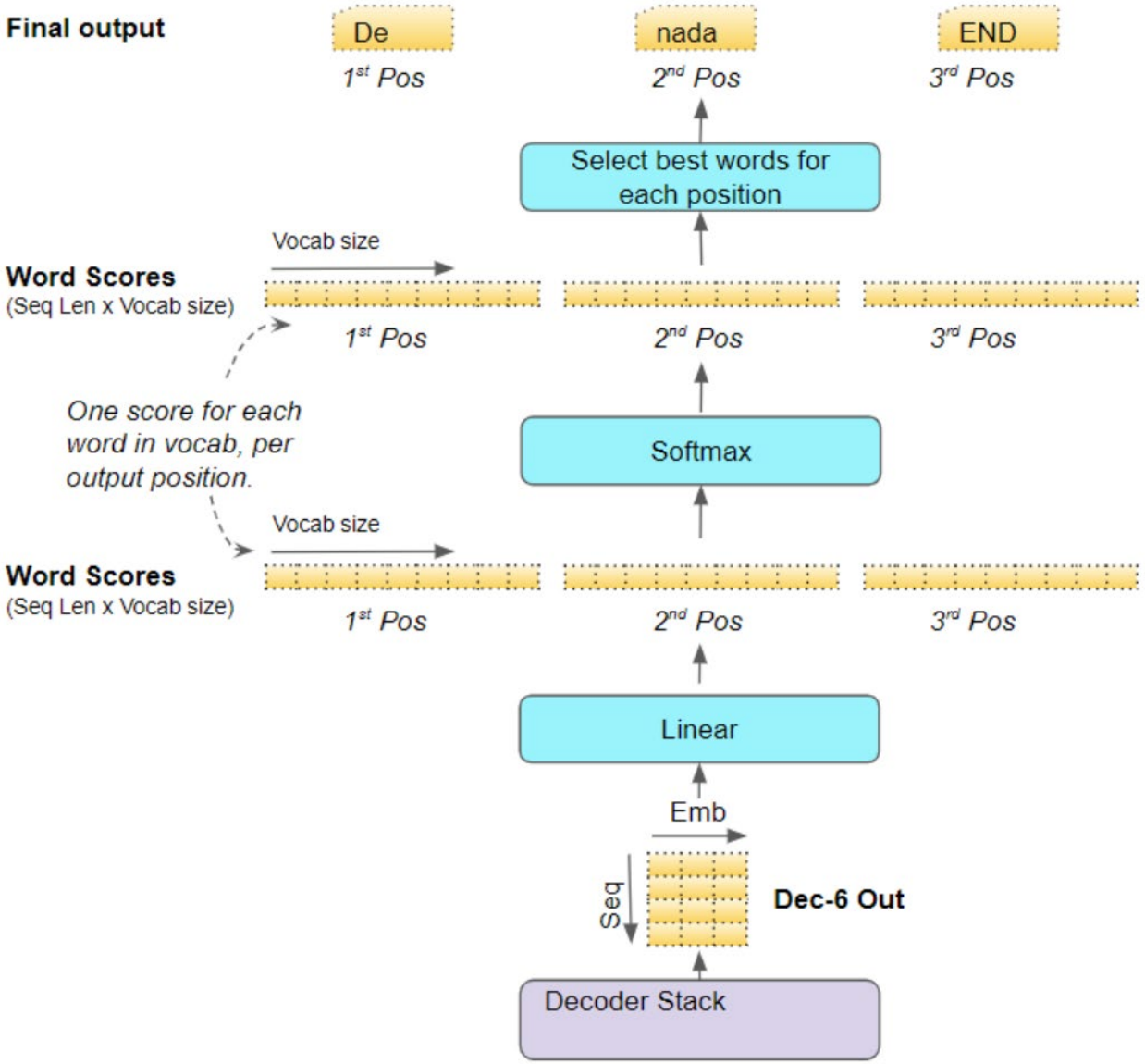
4a: Decoder Self-Attention and Masking



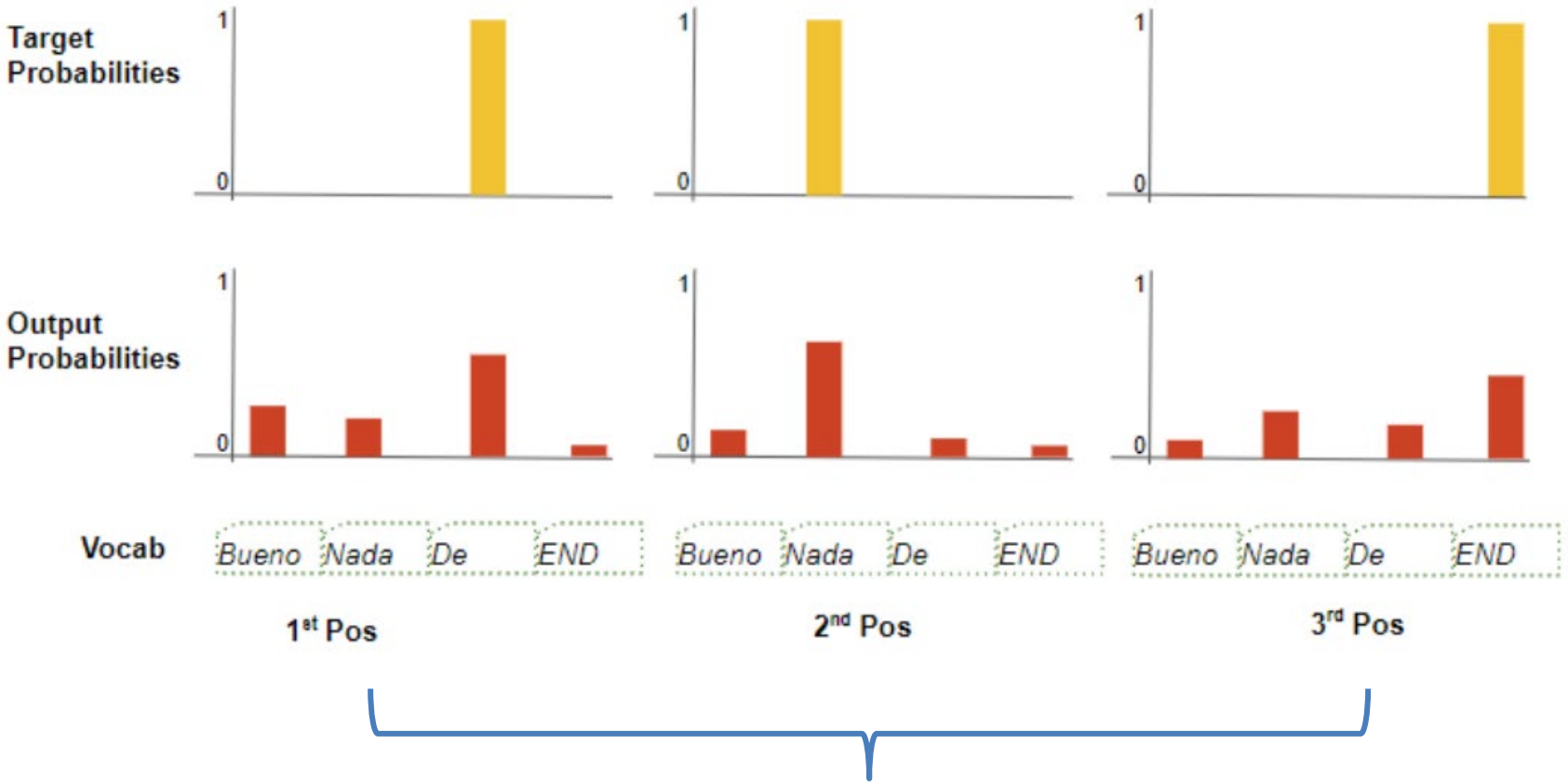
4b: Encoder-Decoder Attention and Masking



5. Generate Output



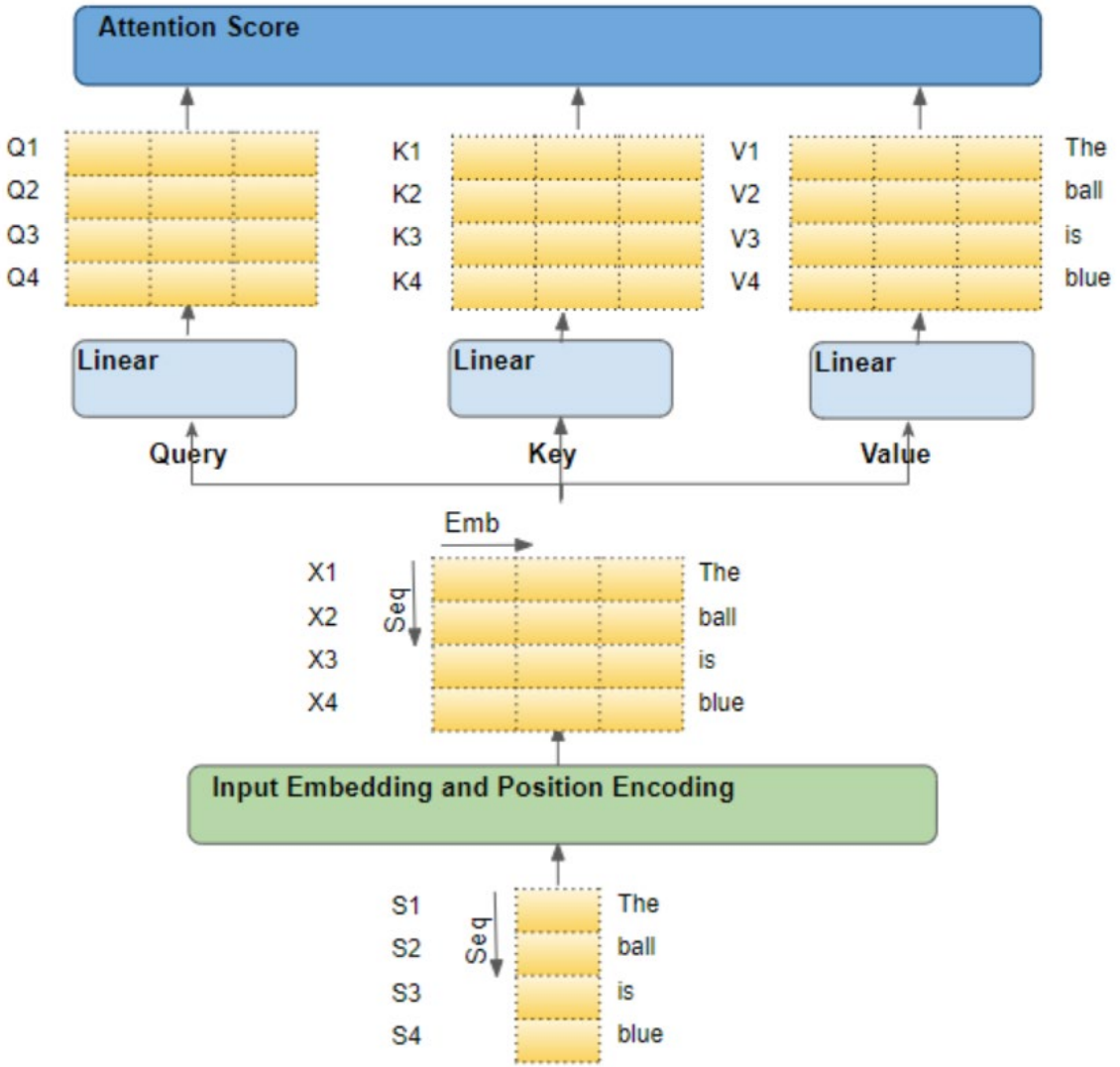
6. Training and Loss Function



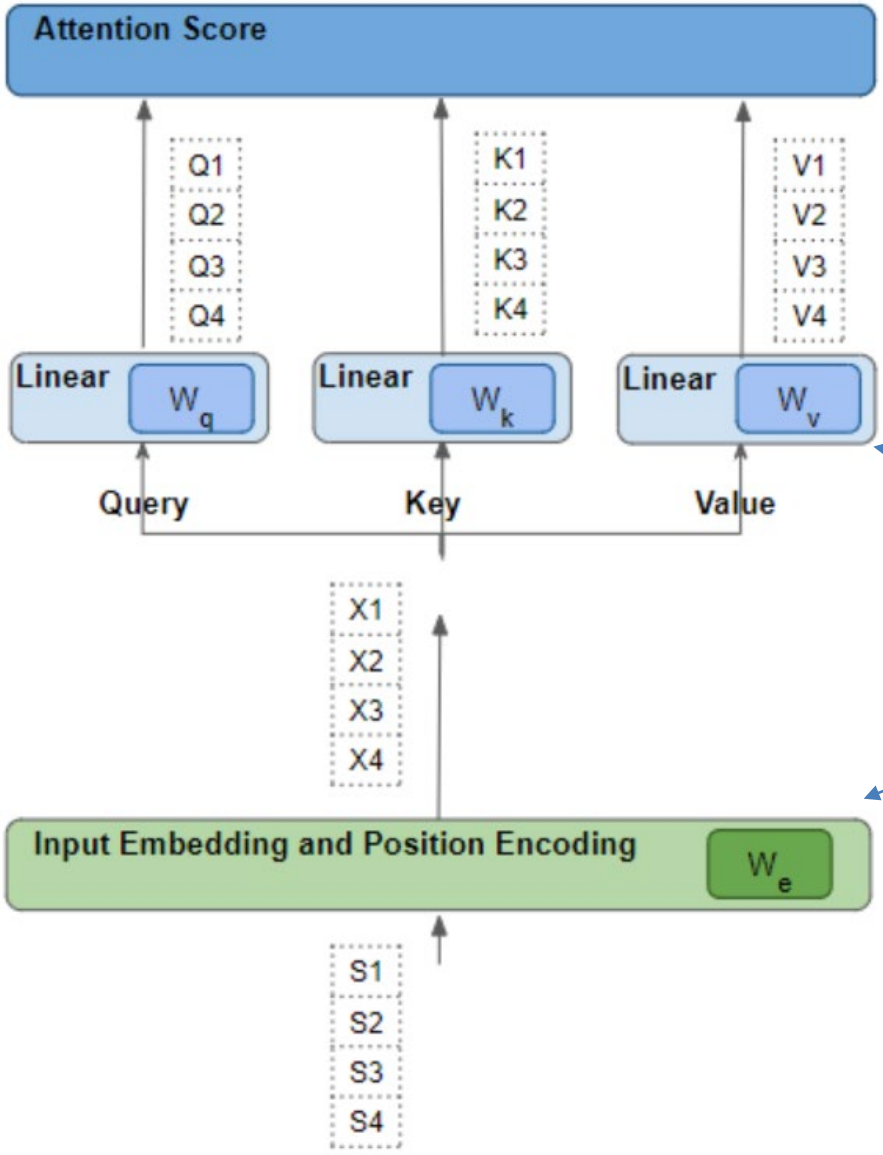
Outline

- An introduction on Attention
- General Attention
- How attention is used in Transformer
- Understanding Transformer
- **Why Transformer works?**
- Implement Transformer for Language Model
- Implement Transformer for Seq2Seq Model
- Conclusion

The flow of source sequence

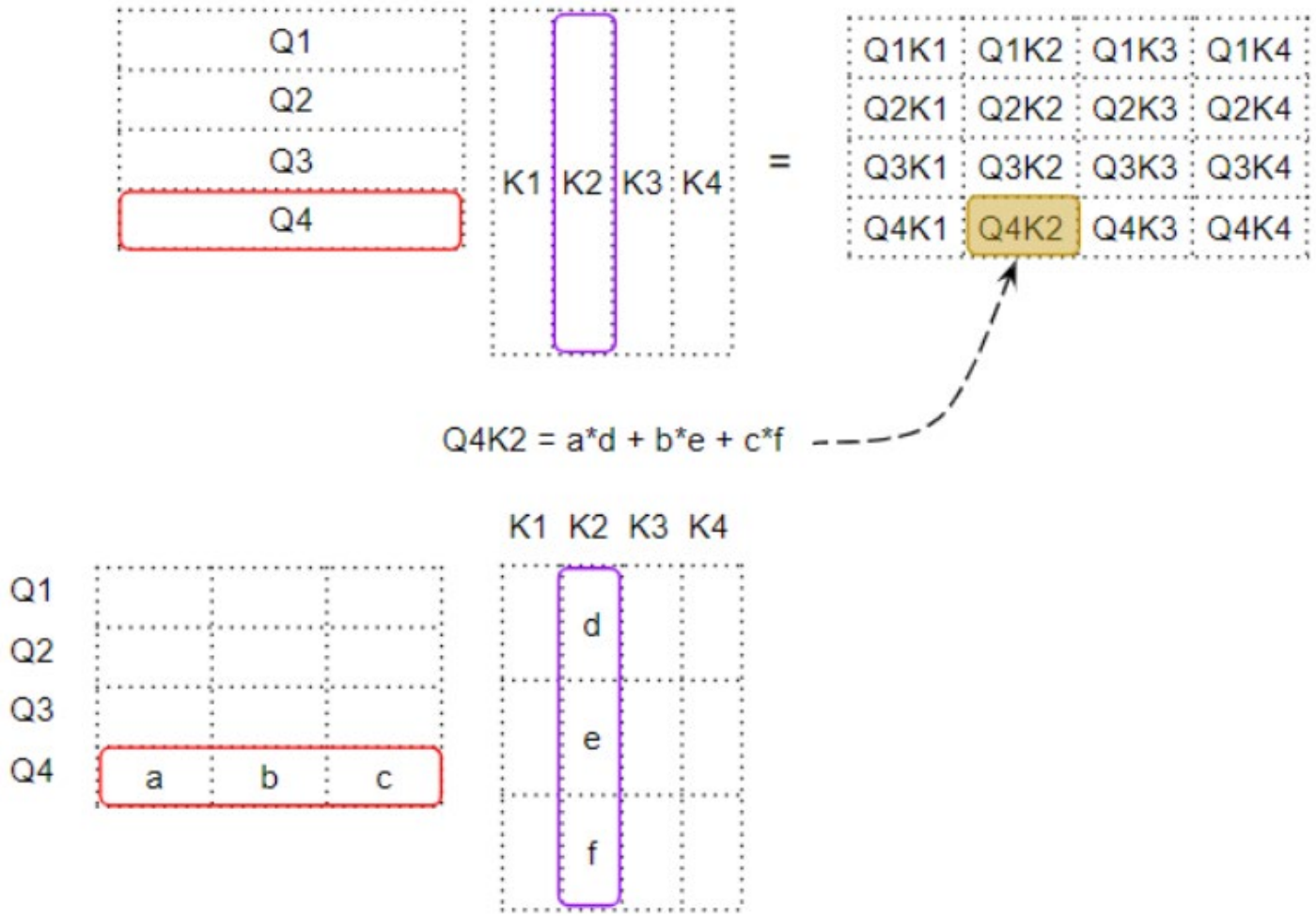


Each word goes through a series of learnable transformations

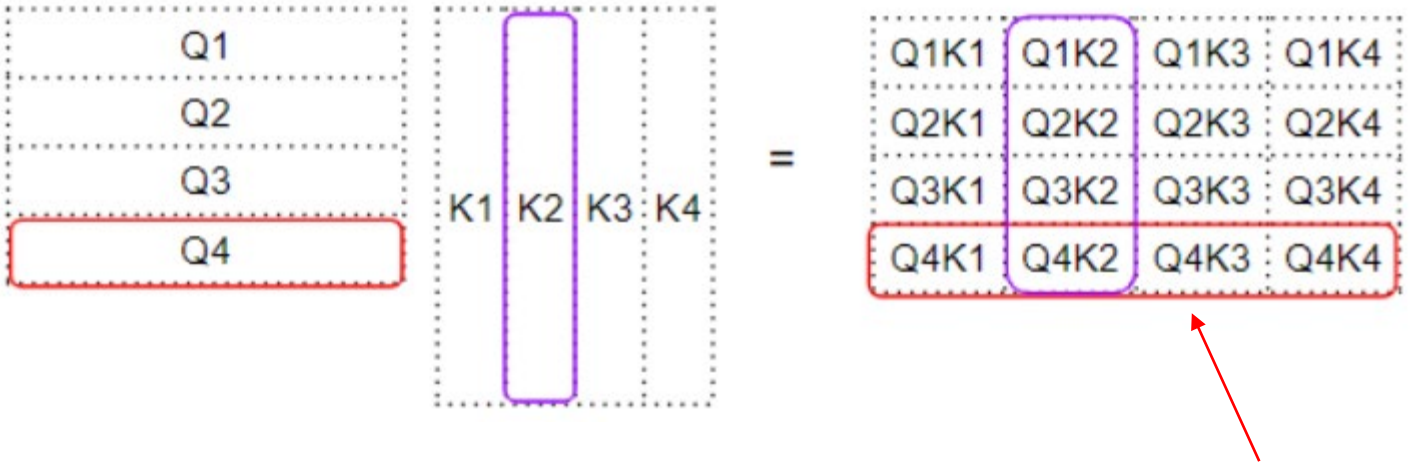


Linear and Embedding weights are learned

Dot Product tells us the similarity between words

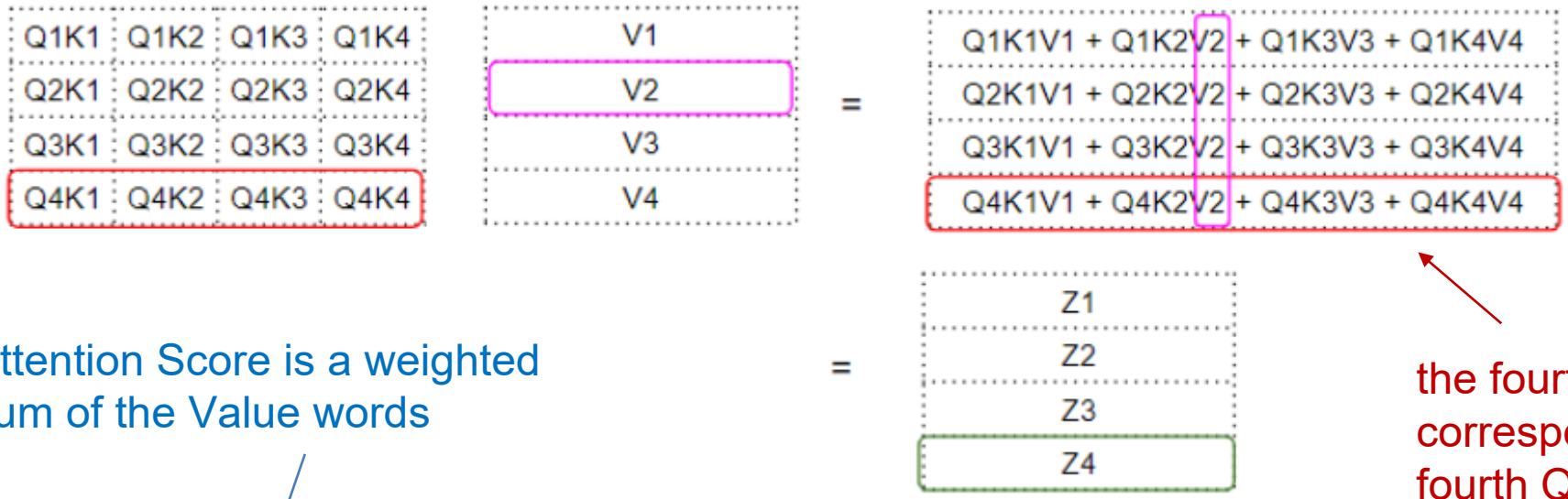


Dot Product between Query and Key matrices



the fourth row corresponds to a dot product between the fourth Query word with every Key word

Dot Product between Query-Key and Value matrices



Attention Score is a weighted sum of the Value words

the fourth row corresponds to the fourth Query word multiplied with all other Key and Value words.

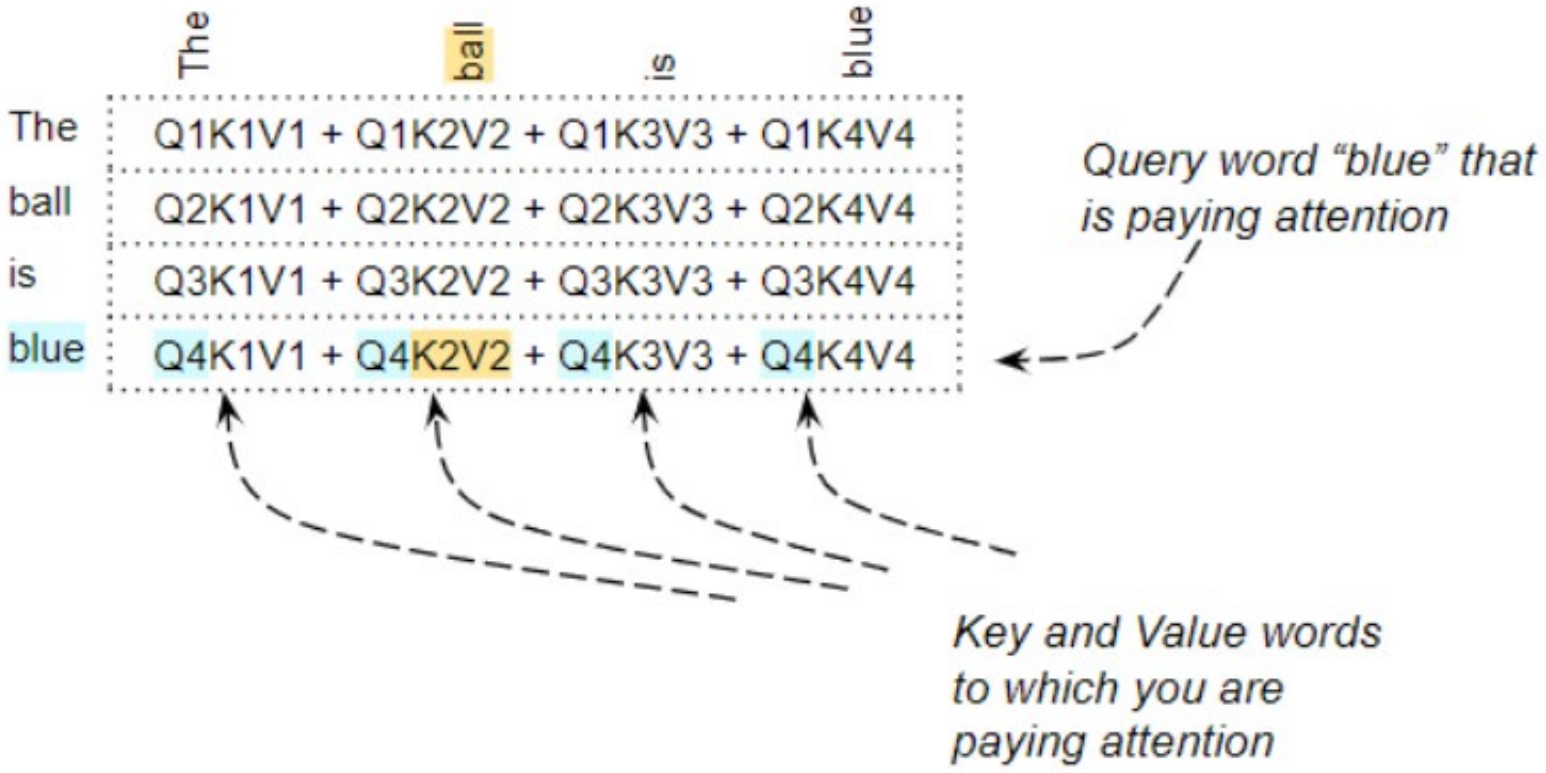
Fourth word Score

*Fourth Query word * first Key word*

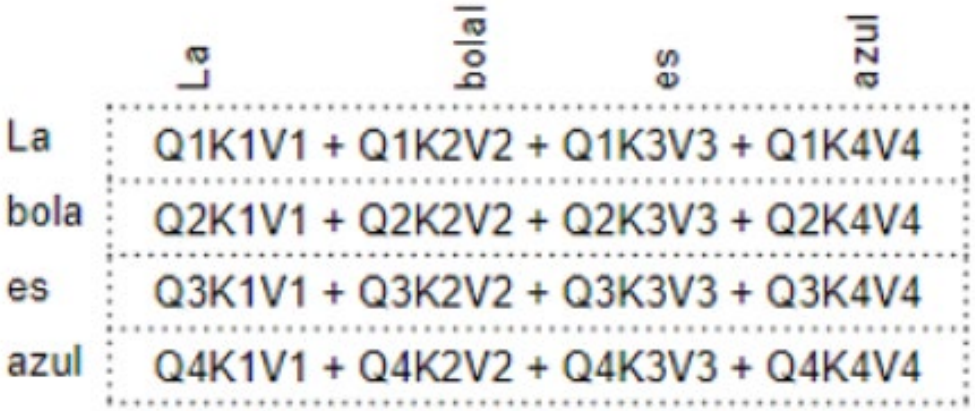
$$Z_4 = (Q_4K_1)\,V_1 + (Q_4K_2)\,V_2 + (Q_4K_3)\,V_3 + (Q_4K_4)\,V_4$$

*Fourth Query word * second Key word*

The role of the Query, Key, and Value words

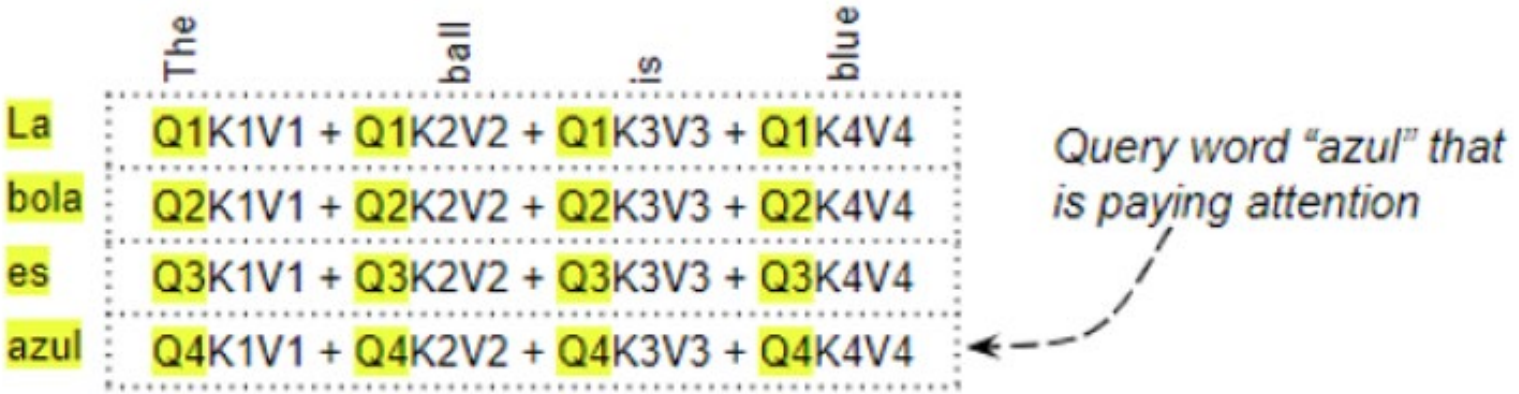


Attention Score for the word "blue" pays attention to every other word



Decoder Self Attention

Target sentence paying attention to itself



Encoder-Decoder Attention

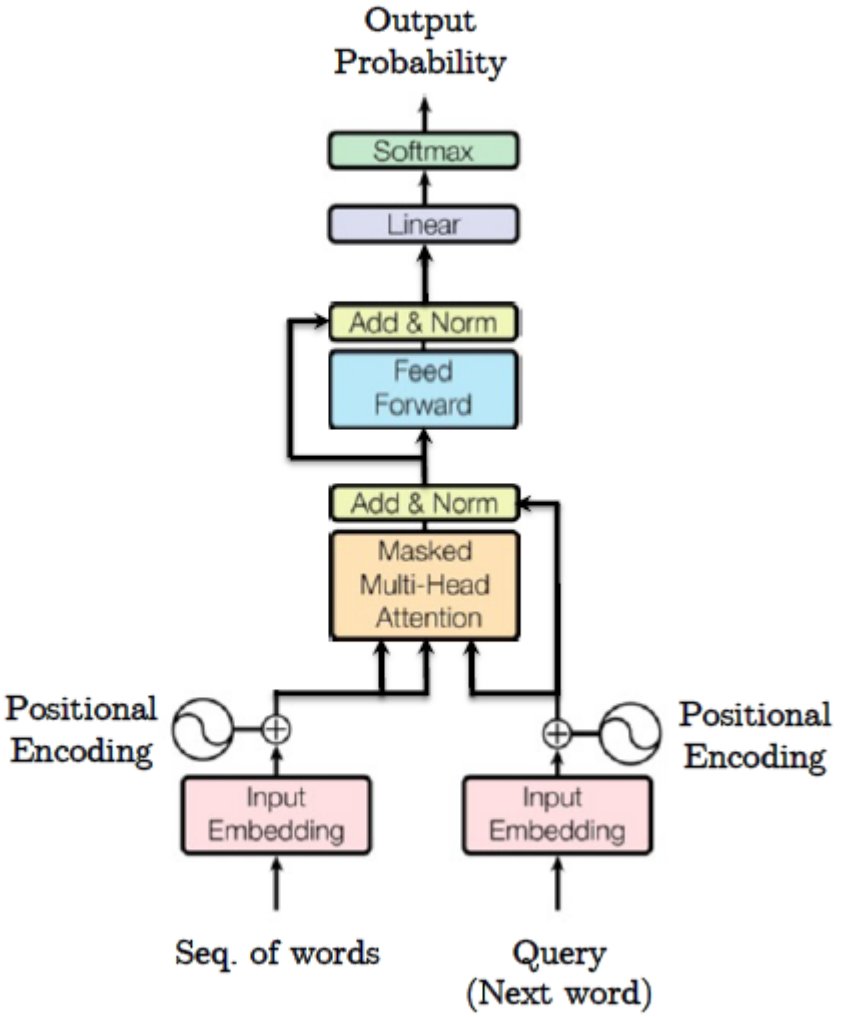
Target sentence paying attention to source sentence

Outline

- An introduction on Attention
- General Attention
- How attention is used in Transformer
- Understanding Transformer
- Why Transformer works?
- **Implement Transformer for Language Model**
- Implement Transformer for Seq2Seq Model
- Conclusion

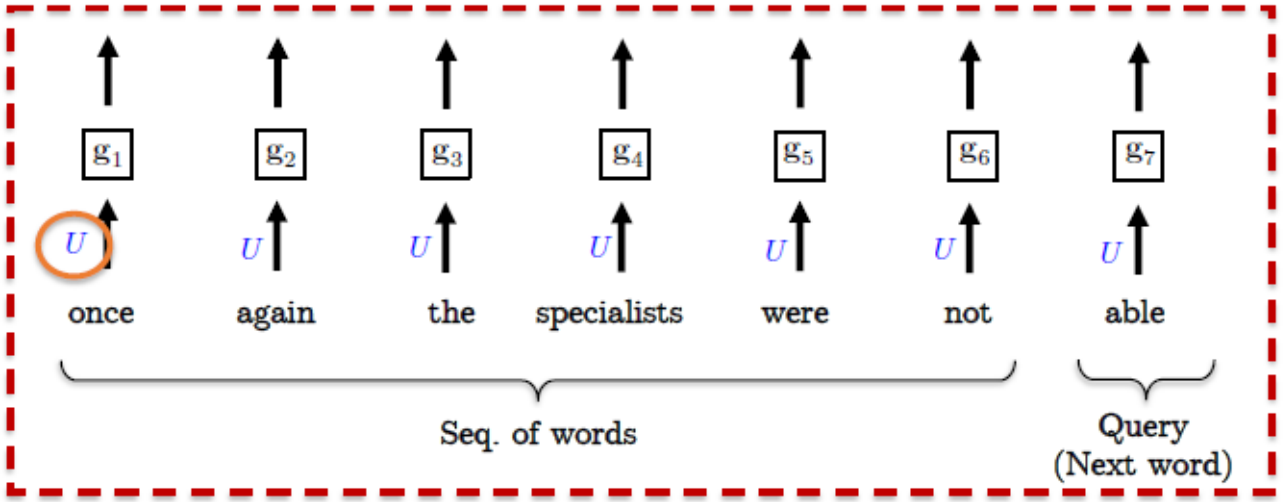
Language Model Transformers

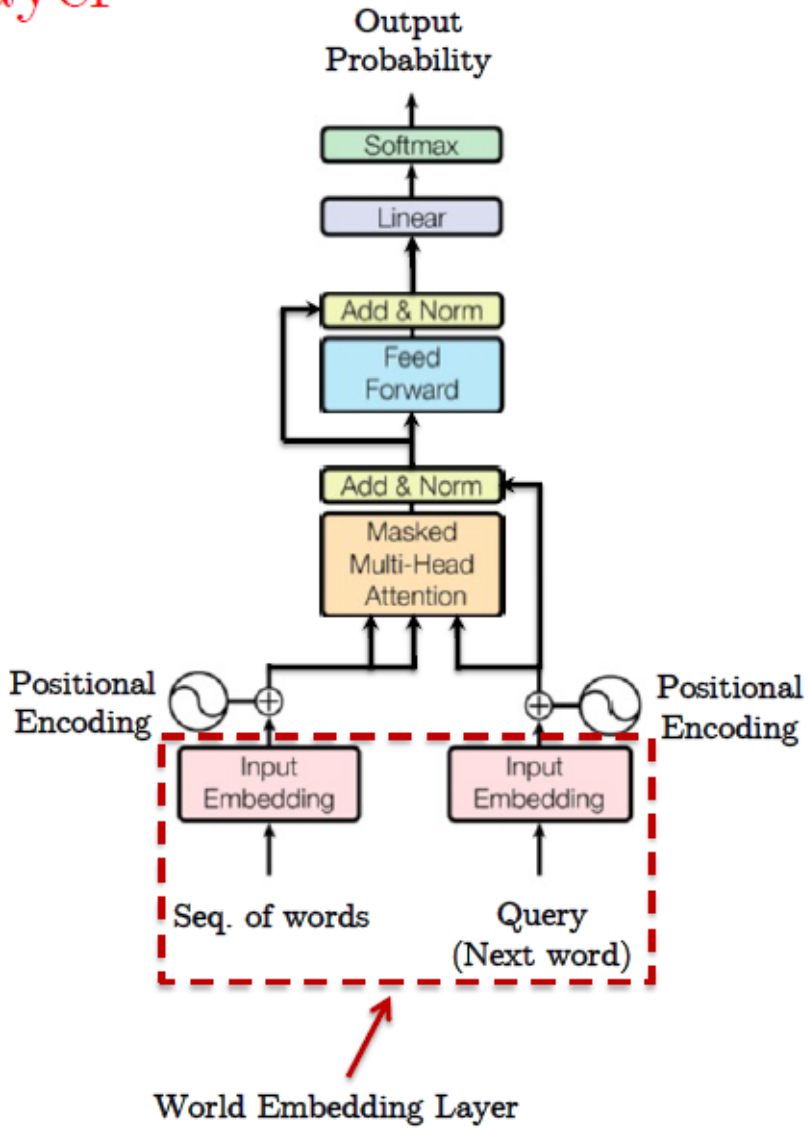
- Given a sequence of words, predict the next word in the sequence.
- Basic task in NLP
  - Requires a world representation that can be flexible/changed in different contexts.
  - Transformers improved word representation to word-in-context representation.
- A Language Model Transformer is composed of three layers :
  - Word embedding layer
  - Attention layer
  - Classification layer



World Embedding Layer

- Categorical variables (dictionary of 10,000 words) are
  - represented by one-hot vectors and then
  - embedded in a linear space.
- This is the same input embedding as in RNNs.
  - PyTorch nn.Embedding()





Multi-Head Attention

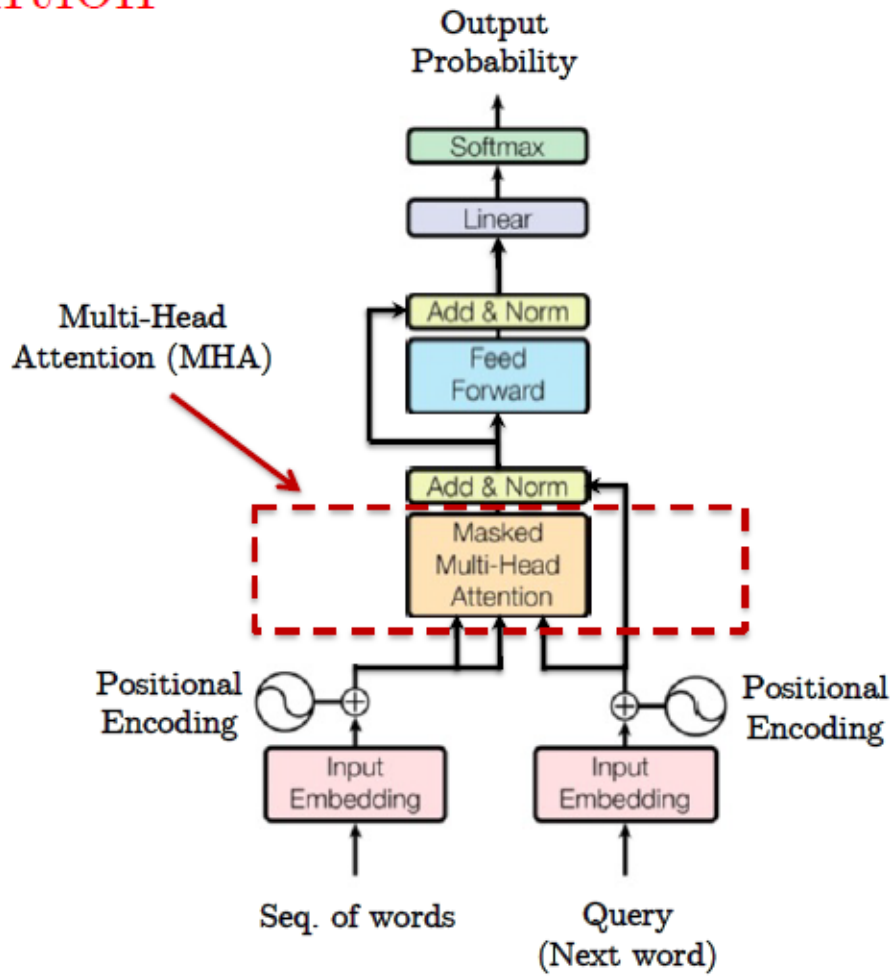
- MHA update equation
  - PyTorch nn.MultiheadAttention()

$$h = \mathrm{MHA}(q, K, V) \in \mathbb{R}^d,\ q \in \mathbb{R}^d, K \in \mathbb{R}^{n \times d}, V \in \mathbb{R}^{n \times d}$$
$$= \Big(\|_{h=1}^{H}\ \mathrm{HA}_h(q, K, V)\Big) W^O,\ W^O \in \mathbb{R}^{d \times d}$$

Concatenation operation

$$\text{with } q = \mathrm{g}_t \in \mathbb{R}^d,\ K = V = \{\mathrm{g}_{t-1}, ..., \mathrm{g}_{t-n}\} \in \mathbb{R}^{n \times d}$$

$$\mathrm{HA}_h(q, K, V) = \mathrm{Softmax}\Big(\frac{q_h K_h^T}{\sqrt{d/H}}\Big) V_h \in \mathbb{R}^{d/H}$$
$$\text{with } q_h = qW_h^q \in \mathbb{R}^{d/H}, W_h^q \in \mathbb{R}^{d \times d/H}$$
$$K_h = KW_h^K \in \mathbb{R}^{n \times d/H}, W_h^K \in \mathbb{R}^{d \times d/H}$$
$$V_h = VW_h^V \in \mathbb{R}^{n \times d/H}, W_h^V \in \mathbb{R}^{d \times d/H}$$

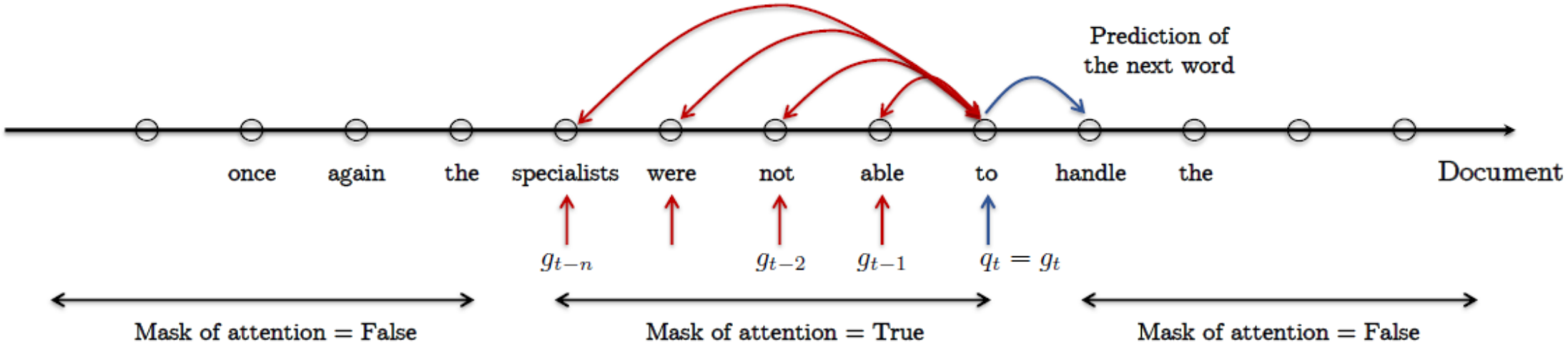


Masked MHA

- Masked MHA equation :

$$\mathrm{HA}(q, K, V) = \mathrm{Mask} \odot \mathrm{Softmax}\left(\frac{qK^T}{\sqrt{d}}\right)V$$

$$\text{with } \mathrm{Mask}_{ij} = \begin{cases} 1 \text{ if attention between } i \text{ and } j \\ 0 \text{ if no attention} \end{cases}$$



Attention Layer

- Attention layer update equation :

$$\text{for } \ell = 0, ..., L-1$$

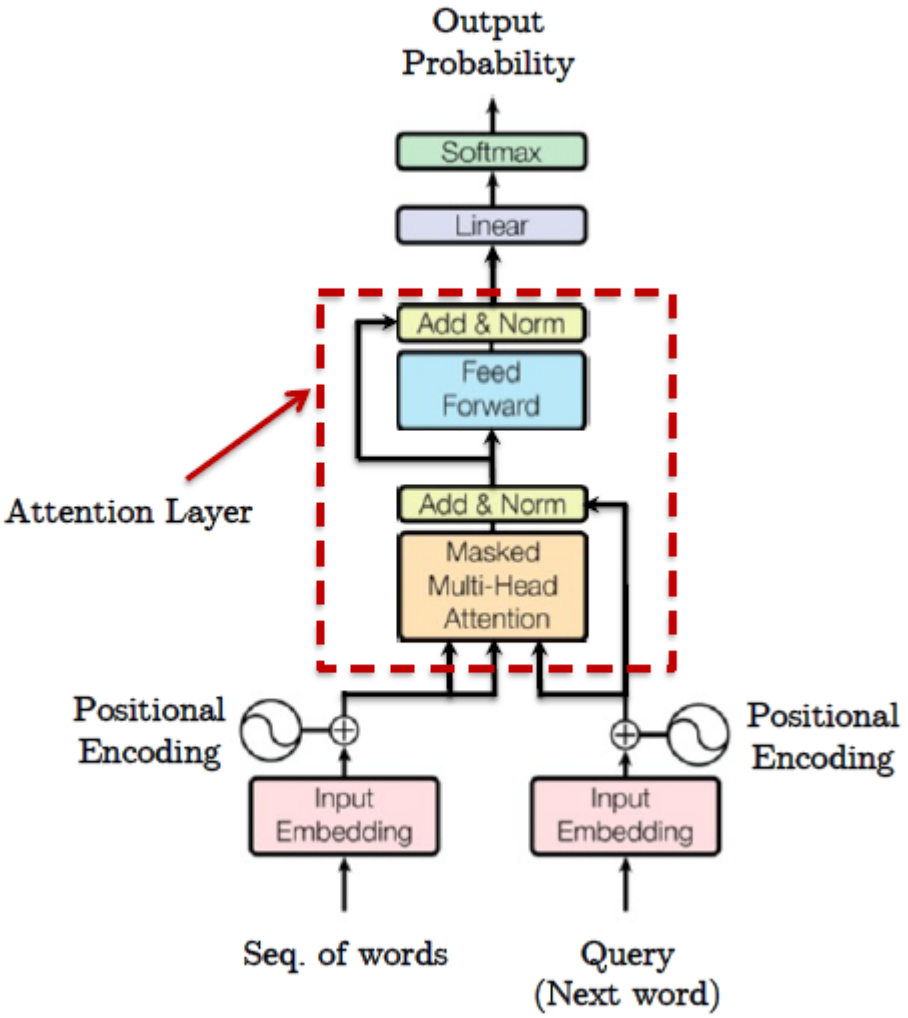
$$\bar{q}^{\ell+1} = \text{LN}\big(q^\ell + \text{MHA}(q^\ell, K^\ell, V^\ell)\big) \in \mathbb{R}^d$$

$$q^{\ell+1} = \text{LN}\big(\bar{q}^{\ell+1} + \text{MLP}(\bar{q}^{\ell+1})\big) \in \mathbb{R}^d$$

Layer normalization
z-scoring with learnable parameters
nn.LayerNorm()

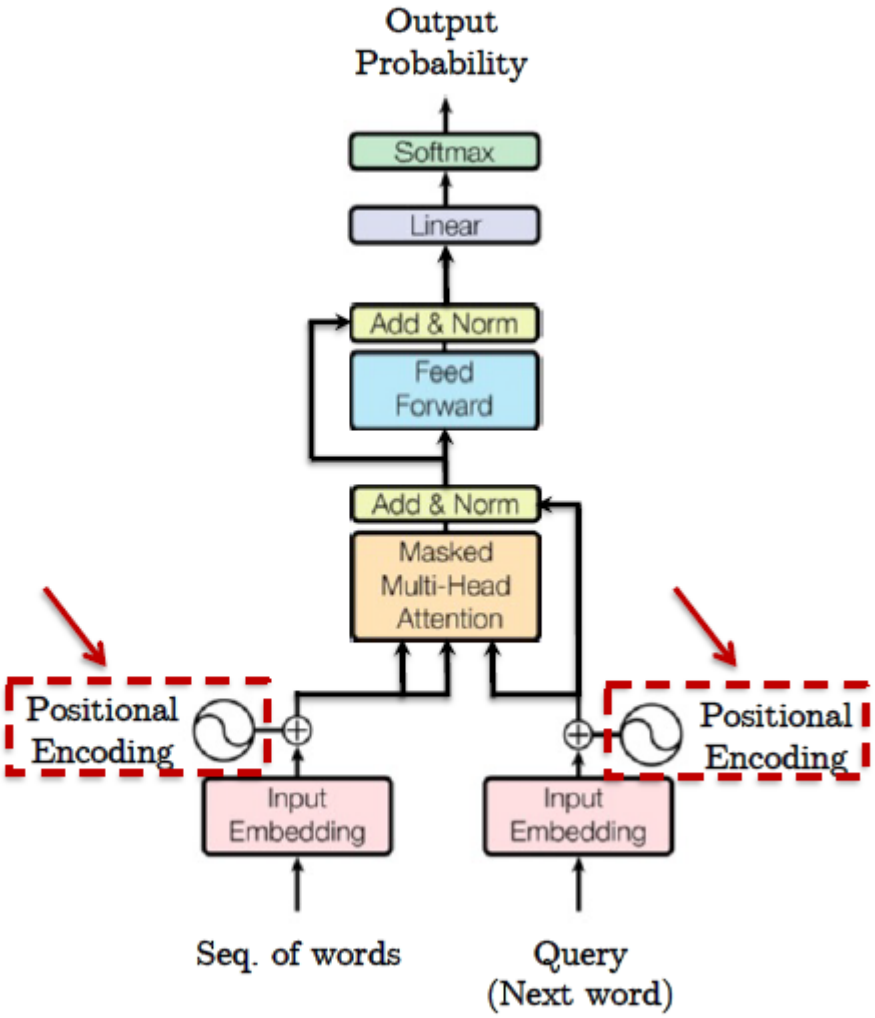
$$\text{LN}(q) = a \odot \left(\frac{q-\mu}{\sigma}\right) + b \in \mathbb{R}^d$$

$$\text{with } \mu = \text{Mean}(q) \in \mathbb{R}, \sigma = \text{Std}(q) \in \mathbb{R}, a, b \in \mathbb{R}^d$$



Positional Encoding

- ANNs like Transformers are designed to process sets (items in a set are not ordered).
- This is an issue for NLP tasks.
  - An additional ordering feature is required to inject temporal information in the attention network.
- Two classes of Positional Encoding (PE) :
  - Learnable vs non-learnable PE
- Learnable PE :
  - Embedding of discrete ordering index 0,1,2,3,...,L-1, with L is the sequence length.
  - Two issues :
    - Requires to know the maximum L value among all training sequences.
    - Some test sequences may have lengths not present in the train set.

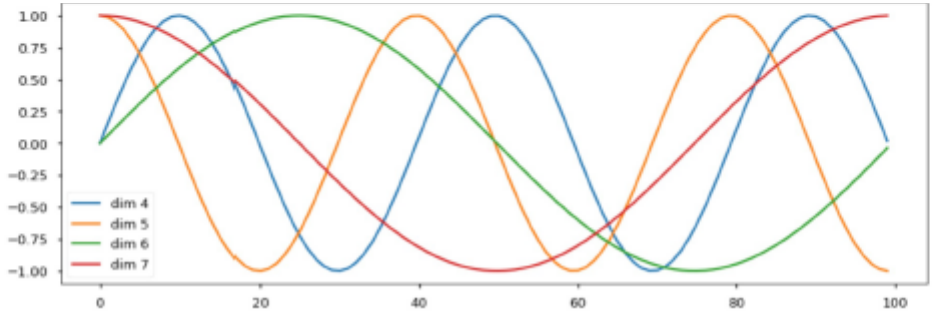


Positional Encoding

- Non-learnable PE :
  - Continuous ordering with sin and cos functions.
  - Advantages :
    - No training necessary
    - No need to know the maximum length in the train set.
    - Test sequences may have lengths not present in the train set.

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

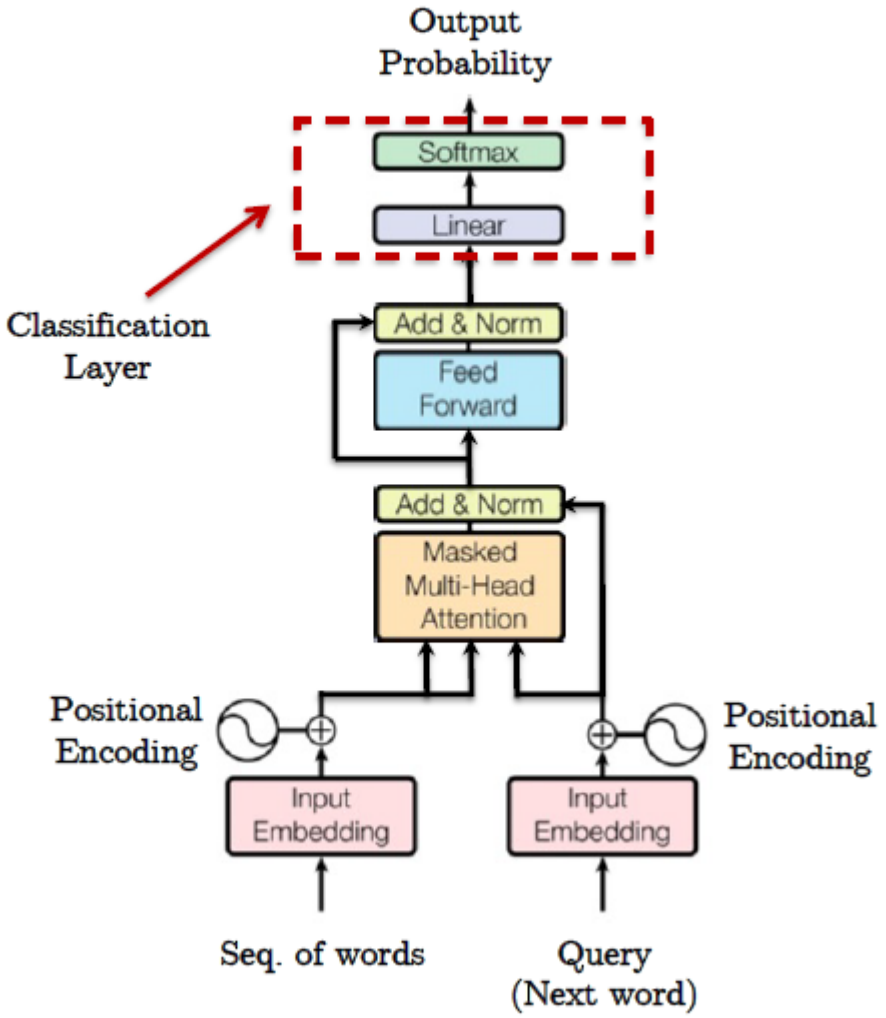
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



- pos is the word position in the input sequence (len = 100 in the above figure)
- 2i or 2i +1 is the index / dim position in the output vector (d = 20 in the above figure)

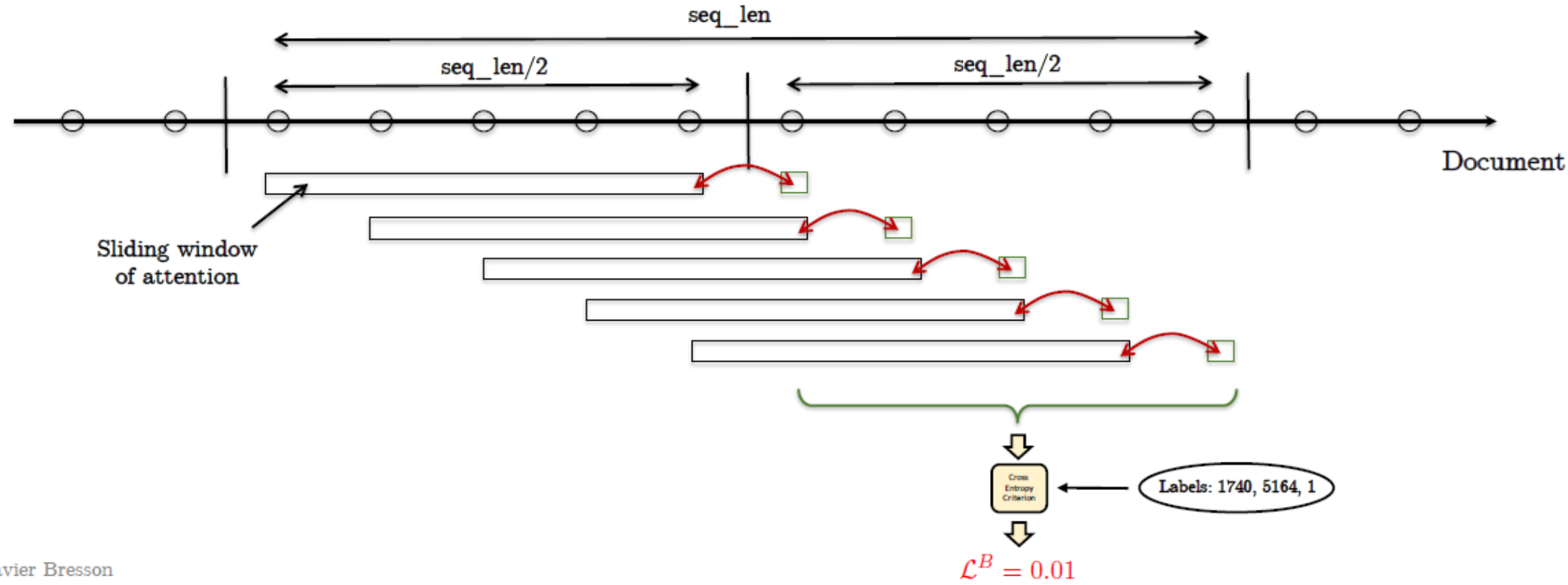
Classification Layer

- The last layer is a standard linear layer to generate the scores of the next word in the sequence, followed by a Softmax operation to produce the probability vector.



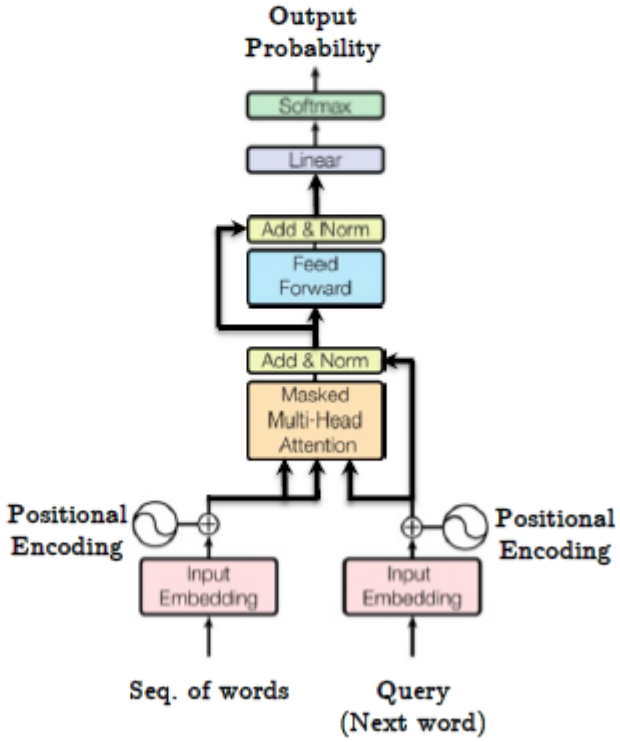
Training

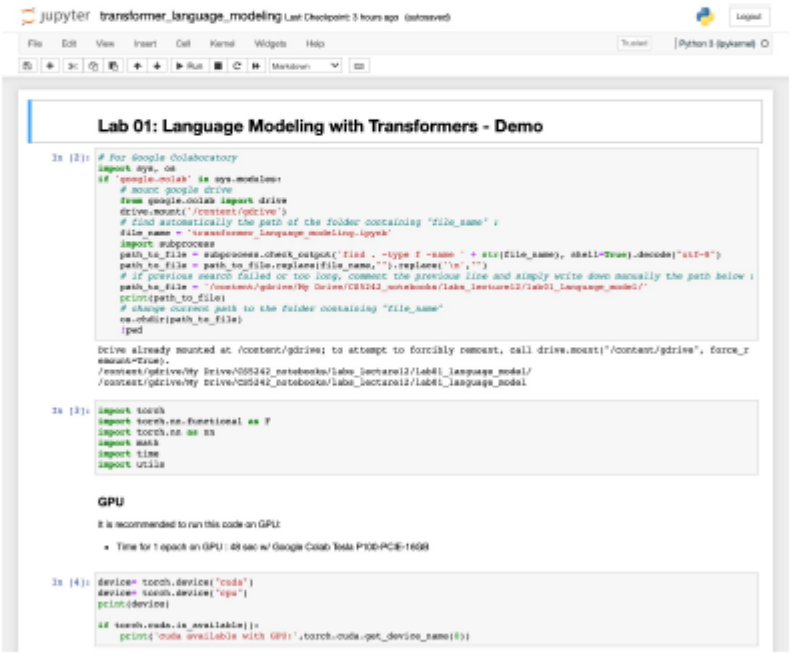
- Training is done with a fixed-size and sliding window of attention.
  - If train_data size is seq_len x batch_size then the size of the window of attention is seq_len/2.



Lab 01

- PyTorch implementation of Language Model Transformers





Lab 01

- Numerical results on PTB :
  - Vanilla RNNs: $\exp(\text{train\_loss}) = 111$ $\exp(\text{test\_loss}) = 155$ 3 million parameters
  - LSTM: $\exp(\text{train\_loss}) = 59$ $\exp(\text{test\_loss}) = 106$ 7 million parameters
  - LSTM state-of-the-art: $\exp(\text{test\_loss}) = 50$ 50 million parameters
  - Vanilla LM Transformers:

    $\exp(\text{train\_loss}) = 54$ $\exp(\text{test\_loss}) = 174$ 3 million parameters
  - LM Transformers state-of-the-art (with pre-training):

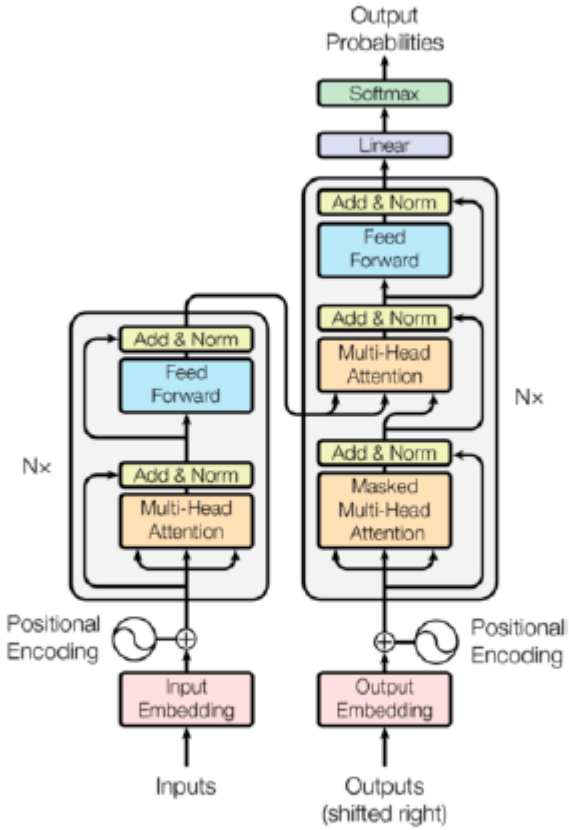
    $\exp(\text{test\_loss}) = 20.5$ 175,000 million parameters

Outline

- An introduction on Attention
- General Attention
- How attention is used in Transformer
- Understanding Transformer
- Why Transformer works?
- Implement Transformer for Language Model
- **Implement Transformer for Seq2Seq Model**
- Conclusion

Seq2Seq Transformers

- Given a sequence of words, convert it into a different sequence.
- Basic task in NLP
  - Translations
  - Questions and Answers
  - Summarization
- A Seq2Sep Transformer is composed of
  - World embedding layer
  - Self-Attention encoding layer
  - Self-Attention decoding layer
  - Cross-Attention decoding layer
  - Classification layer



Input(EN): He drives the car. Output(FR): Il conduit la voiture.

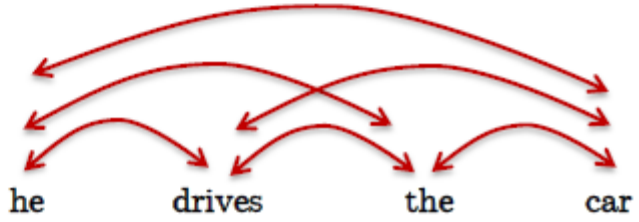
Seq2Seq Transformer Architecture
Vaswani-etal, "Attention is all you need", 2017

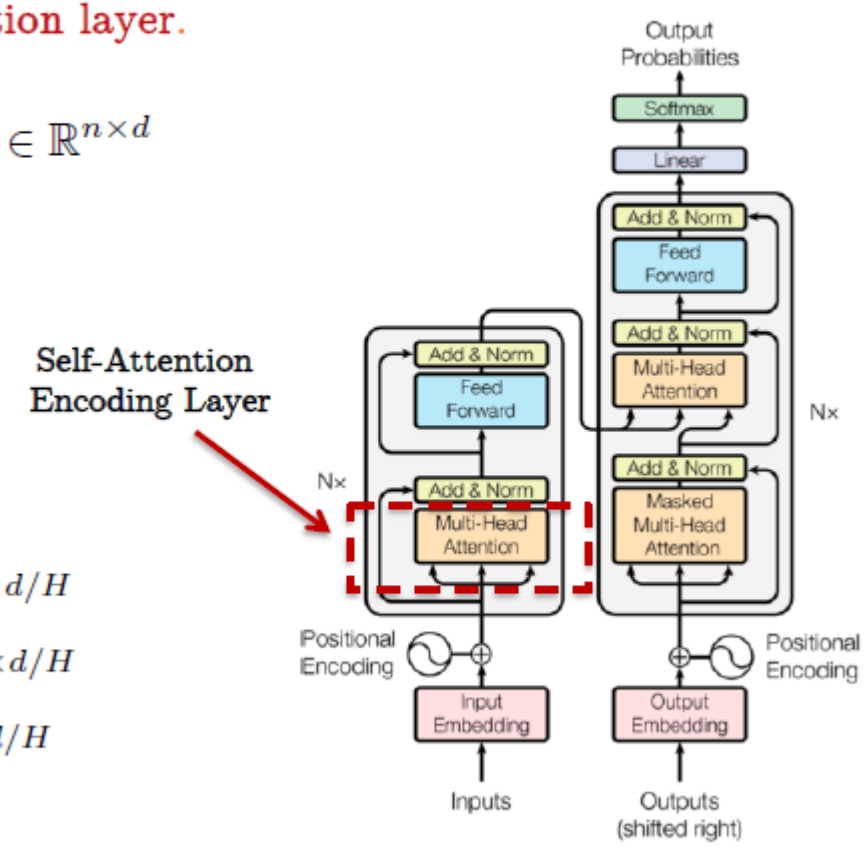
Self-Attention Encoding Layer

- Represent the whole input sequence with a self-attention layer.

$$H = \text{MHA}(Q, K, V) \in \mathbb{R}^{n \times d},\ Q \in \mathbb{R}^{n \times d}, K \in \mathbb{R}^{n \times d}, V \in \mathbb{R}^{n \times d}$$
$$= \Big(\|_{h=1}^{H} \text{HA}_h(Q, K, V) \Big) W^O,\ W^O \in \mathbb{R}^{d \times d}$$
$$\text{with } Q = K = V = \{\text{g}_1^{\text{in}}, ..., \text{g}_n^{\text{in}}\} \in \mathbb{R}^{n \times d}$$

$$\text{HA}_h(Q, K, V) = \text{Softmax}\Big(\frac{Q_h K_h^T}{\sqrt{d/H}}\Big) V_h \in \mathbb{R}^{n \times d/H}$$
$$\text{with } Q_h = QW_h^Q \in \mathbb{R}^{n \times d/H}, W_h^Q \in \mathbb{R}^{d \times d/H}$$
$$K_h = KW_h^K \in \mathbb{R}^{n \times d/H}, W_h^K \in \mathbb{R}^{d \times d/H}$$
$$V_h = VW_h^V \in \mathbb{R}^{n \times d/H}, W_h^V \in \mathbb{R}^{d \times d/H}$$

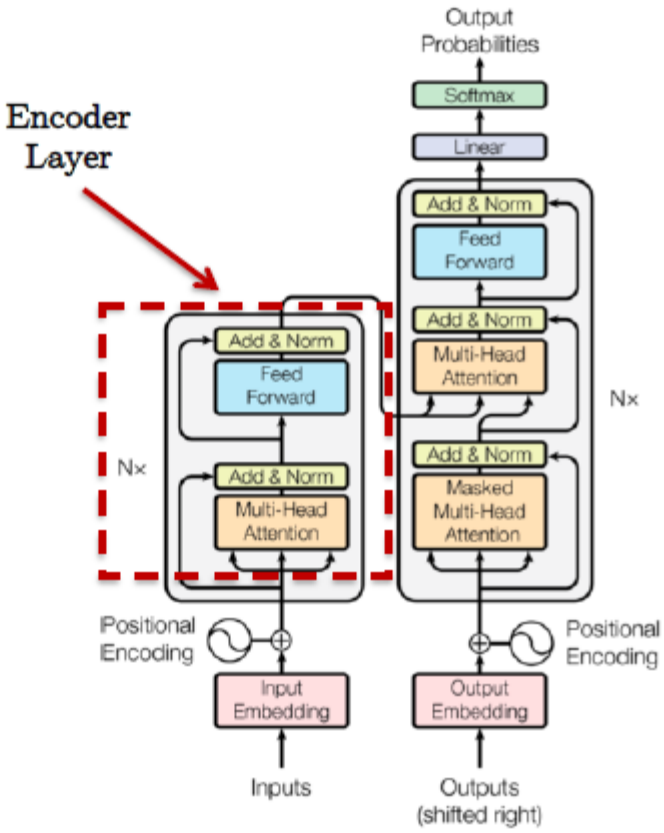




Encoder

- Encoder layer is composed of MHA, MLP, Residual Connection, Layer Normalization, and possibly stacking multiple layers.

$$\text{for } \ell = 0, ..., L^{\text{Enc}} - 1$$
$$\bar{H}^{\ell+1} = \text{LN}\big(H^\ell + \text{MHA}(H^\ell, H^\ell, H^\ell)\big) \in \mathbb{R}^{n \times d}$$
$$H^{\ell+1} = \text{LN}\big(\bar{H}^{\ell+1} + \text{MLP}(\bar{H}^{\ell+1})\big) \in \mathbb{R}^{n \times d}$$
$$\text{with initialization } H^{\ell=0} = \text{LL}(\{\text{g}_1^{\text{in}}, ..., \text{g}_n^{\text{in}}\} + \text{PE}) \in \mathbb{R}^{n \times d}$$

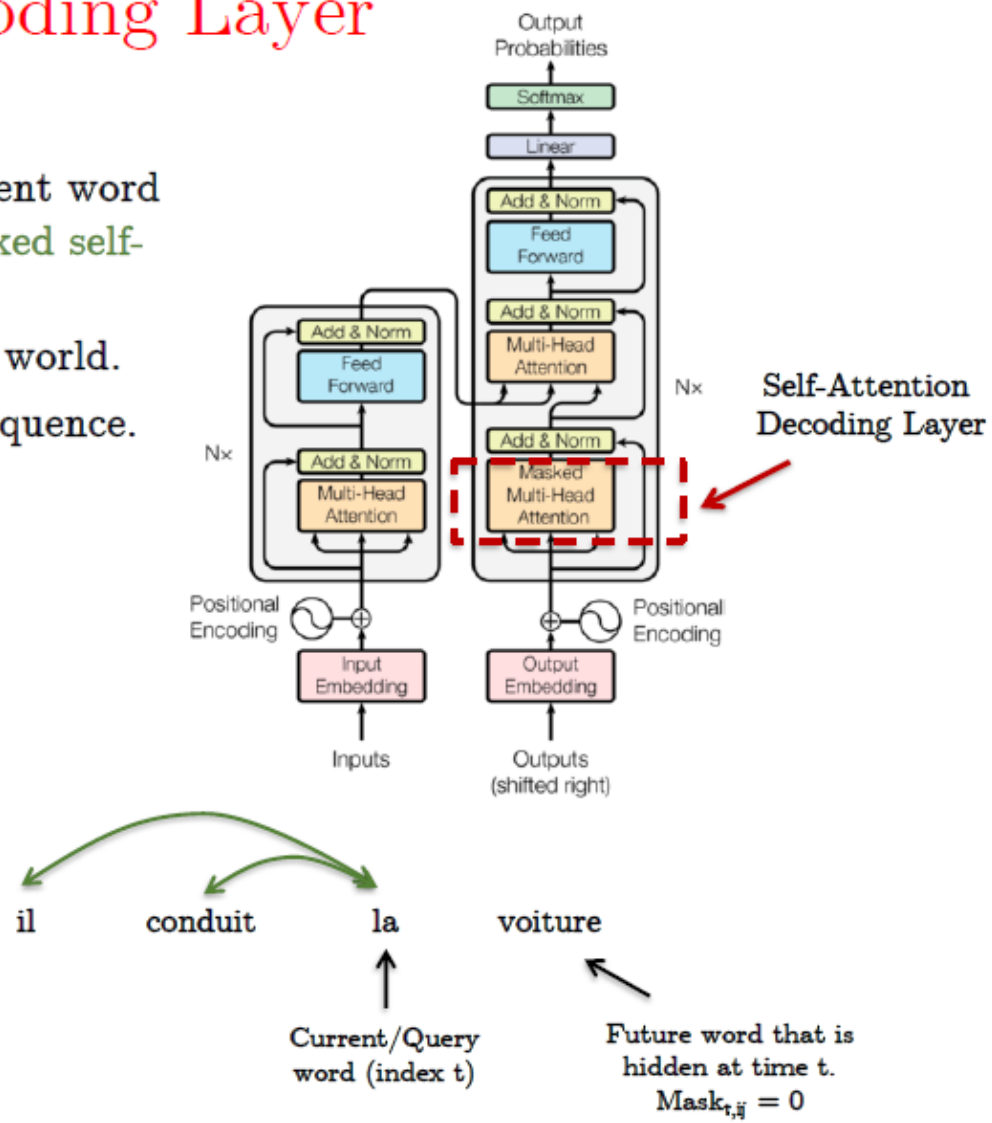


Self-Attention Decoding Layer

- Represent the query (the word embedding of the current word in the partially decoded output sequence) with a masked self-attention layer.
  - Only make attention to words before the current world.
  - Use a mask to hide future words in the output sequence.

$$\bar{q}_t = \mathrm{MHA}(q_t, K, V) \in \mathbb{R}^d,\ q_t \in \mathbb{R}^d, K \in \mathbb{R}^{n \times d}, V \in \mathbb{R}^{n \times d}$$
$$= \left(\|_{h=1}^{H} \mathrm{HA}_h(q_t, K, V) \right) W^O,\ W^O \in \mathbb{R}^{d \times d}$$
$$\text{with } q_t = \mathrm{g}_t^{\mathrm{out}} \in \mathbb{R}^d,\ K = V = \{\mathrm{g}_{t-1}^{\mathrm{out}}, ..., \mathrm{g}_{t-n}^{\mathrm{out}}\} \in \mathbb{R}^{n \times d}$$

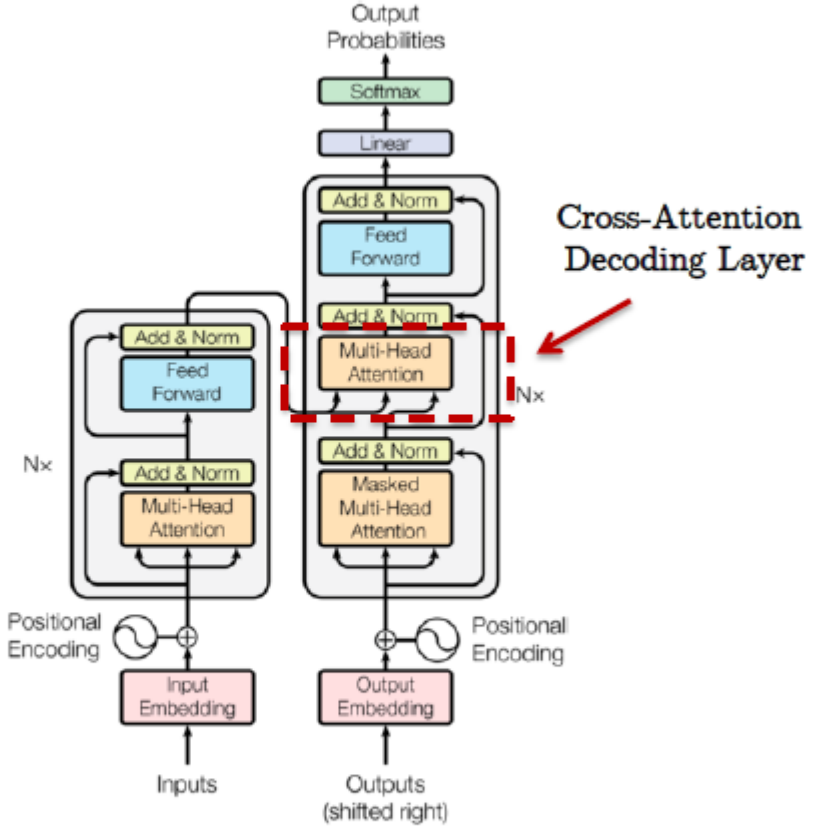
$$\mathrm{HA}(q_t, K, V) = \mathrm{Mask}_t \odot \mathrm{Softmax}\left(\frac{q_t K^T}{\sqrt{d/H}}\right) V$$
$$\text{with } \mathrm{Mask}_{t,ij} = \begin{cases} 1 \text{ if attention between } i \text{ and } j \\ 0 \text{ if no attention} \end{cases}$$

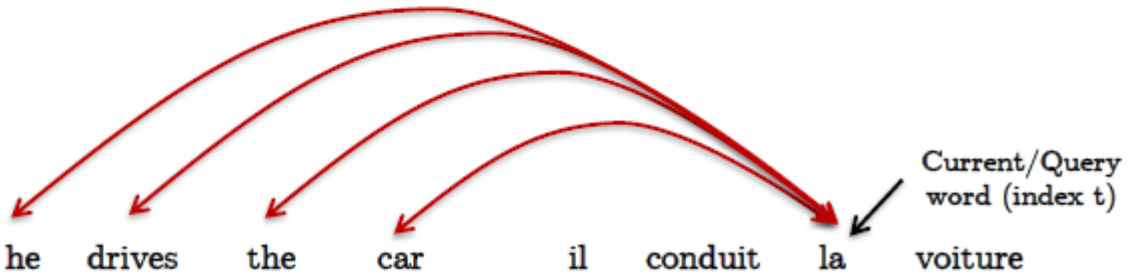


Cross-Attention Decoding Layer

- Draw attention between pairs of words coming from the whole encoded input sequence and the query in the output sequence.

$$\hat{q}_t = \mathrm{MHA}(\bar{q}_t, K, V) \in \mathbb{R}^d,\ \bar{q}_t \in \mathbb{R}^d, K \in \mathbb{R}^{n\times d}, V \in \mathbb{R}^{n\times d}$$
$$= \Big(\|_{h=1}^{H}\ \mathrm{HA}_h(\bar{q}_t, K, V)\Big) W^O,\ W^O \in \mathbb{R}^{d\times d}$$
$$\text{with } \bar{q}_t \in \mathbb{R}^d,\ K = V = H^{\mathrm{Enc}} \in \mathbb{R}^{n\times d}$$





Input sequence
Encoded by self-attention
H^{Enc}

Output sequence

Cross-Attention Decoding Layer

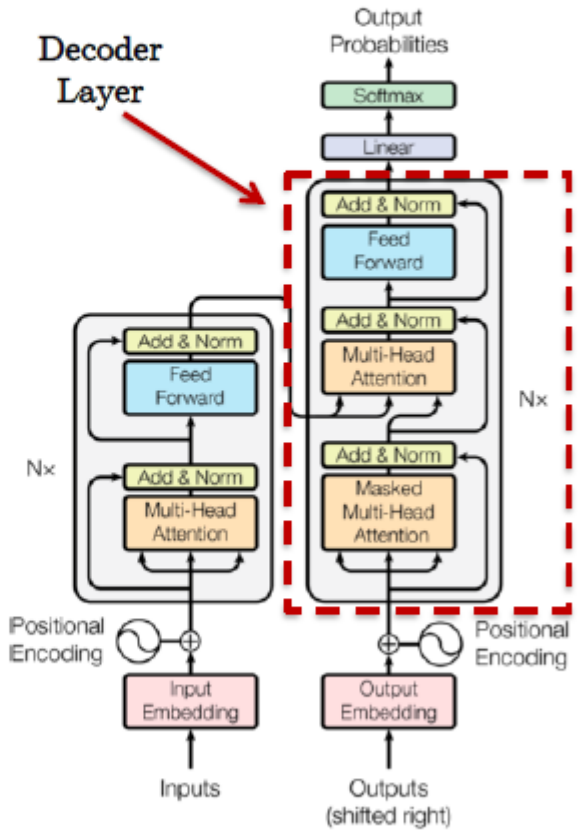
- Output of the decoder :

for $\ell = 0, ..., L^{\text{Dec}} - 1$

$$\bar{q}^{\ell+1} = \text{LN}\big(q^\ell + \text{MHA}(q^\ell, K^\ell, V^\ell)\big) \in \mathbb{R}^d$$
$$\tilde{q}^{\ell+1} = \text{LN}\big(\bar{q}^{\ell+1} + \text{MLP}(\bar{q}^{\ell+1})\big) \in \mathbb{R}^d$$
$$\hat{q}^{\ell+1} = \text{LN}\big(\tilde{q}^\ell + \text{MHA}(\tilde{q}^\ell, H^{\text{Enc}}, H^{\text{Enc}})\big) \in \mathbb{R}^d$$
$$q^{\ell+1} = \text{LN}\big(\hat{q}^{\ell+1} + \text{MLP}(\hat{q}^{\ell+1})\big) \in \mathbb{R}^d$$

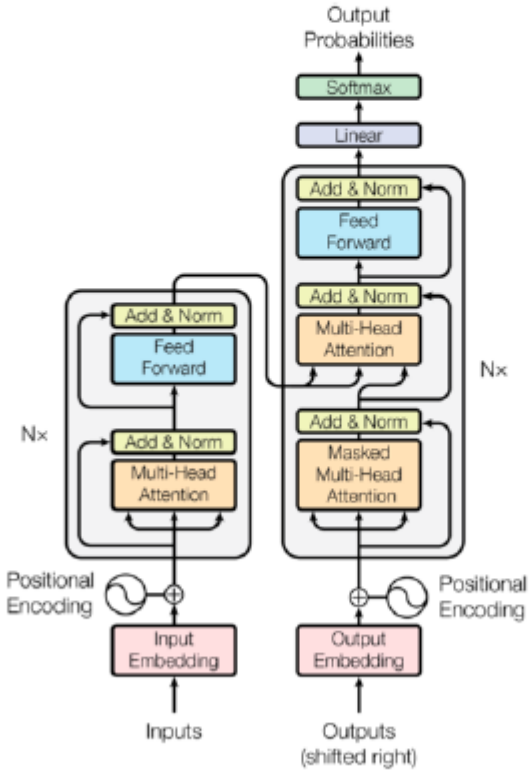
with initialization $q^{\ell=0} = \text{LL}(\text{g}_t^{\text{out}} + \text{PE}_t) \in \mathbb{R}^d$,

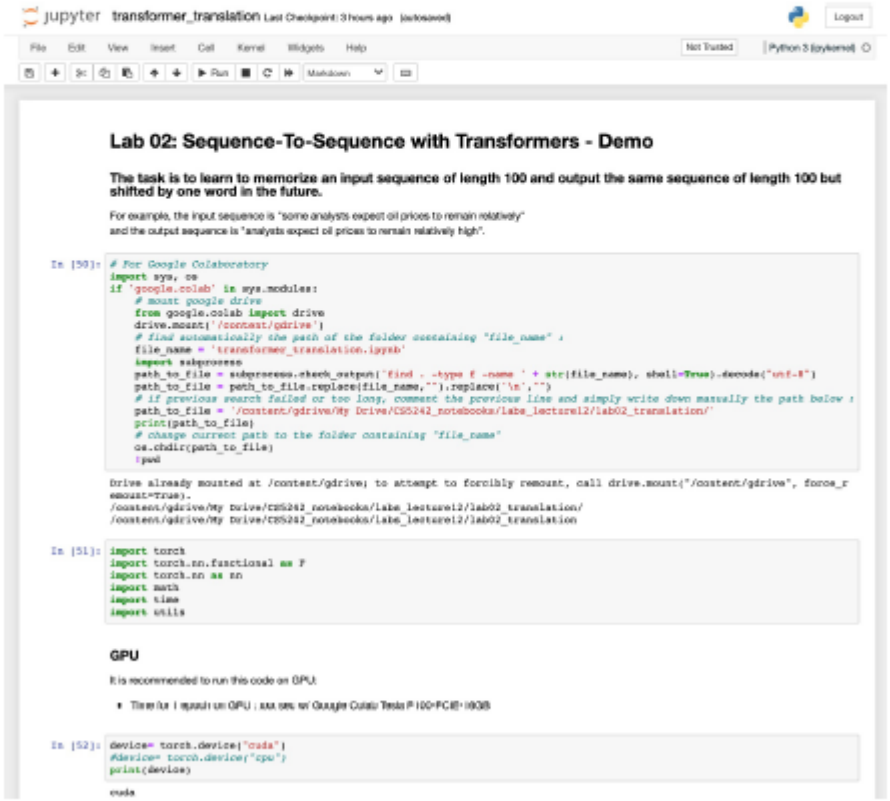
and $K^{\ell=0} = V^{\ell=0} = \text{LL}(\{\text{g}_{t-1}^{\text{out}}, ..., \text{g}_{t-n}^{\text{out}}\} + \text{PE}) \in \mathbb{R}^{n \times d}$



Lab 02

- PyTorch implementation of Seq2Seq Transformers





jupyter transformer_translation

Lab 02: Sequence-To-Sequence with Transformers - Demo

The task is to learn to memorize an input sequence of length 100 and output the same sequence of length 100 but shifted by one word in the future.

For example, the input sequence is "some analysts expect oil prices to remain relatively"
and the output sequence is "analysts expect oil prices to remain relatively high".

```
In [50]: # For Google Colaboratory
import sys, os
if 'google.colab' in sys.modules:
    # mount google drive
    from google.colab import drive
    drive.mount('/content/gdrive')
    # find automatically the path of the folder containing "file_name" :
    file_name = 'transformer_translation.ipynb'
    import subprocess
    path_to_file = subprocess.check_output('find . -type f -name ' + str(file_name), shell=True).decode("utf-8")
    path_to_file = path_to_file.replace(file_name,"").replace('\n',"")
    # if previous search failed or too long, comment the previous line and simply write down manually the path below :
    path_to_file = '/content/gdrive/My Drive/CS5242_notebooks/labs_lecture12/lab02_translation/'
    print(path_to_file)
    # change current path to the folder containing "file_name"
    os.chdir(path_to_file)
    !pwd
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
/content/gdrive/My Drive/CS5242_notebooks/labs_lecture12/lab02_translation/
/content/gdrive/My Drive/CS5242_notebooks/labs_lecture12/lab02_translation
```

```
In [51]: import torch
import torch.nn.functional as F
import torch.nn as nn
import math
import time
import utils
```

GPU

It is recommended to run this code on GPU.

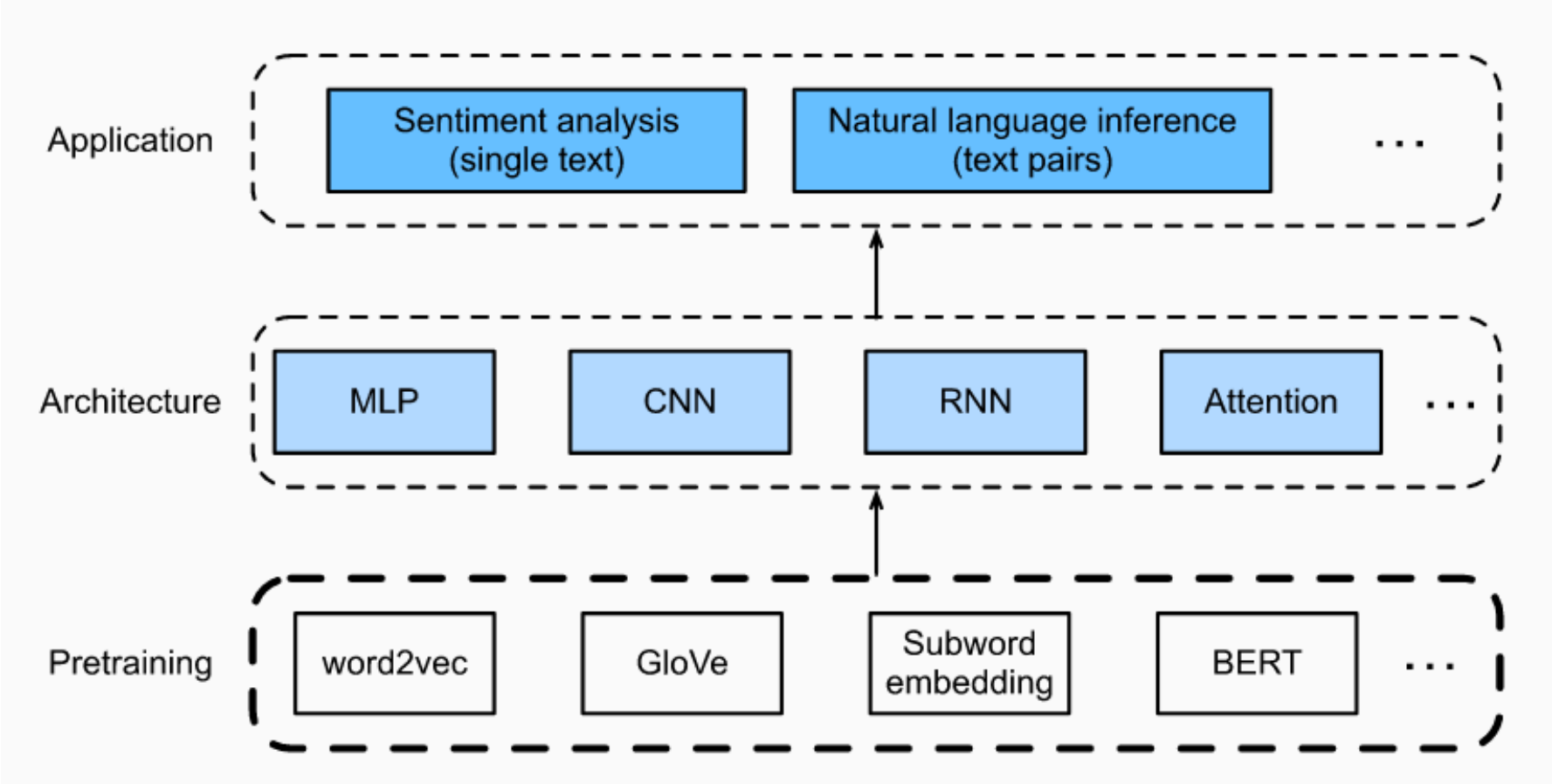
```
In [52]: device= torch.device("cuda")
#device= torch.device("cpu")
print(device)
```

```
cuda
```

Outline

- An introduction on Attention
- General Attention
- How attention is used in Transformer
- Understanding Transformer
- Why Transformer works?
- Implement Transformer for Language Model
- Implement Transformer for Seq2Seq Model
- **Conclusion**

NLP Architecture



Reference

- **Ketan Doshi, "Transformers Explained Visually",**
https://towardsdatascience.com/transformers-explained-visually-part-1-overview-of-functionality-95a6dd460452

https://towardsdatascience.com/transformers-explained-visually-part-2-how-it-works-step-by-step-b49fa4a64f34

https://towardsdatascience.com/transformers-explained-visually-part-3-multi-head-attention-deep-dive-1c1ff1024853

https://towardsdatascience.com/transformers-explained-visually-not-just-how-but-why-they-work-so-well-d840bd61a9d3

- **Some Slides are created by Xavier Bresson**

- **"Dive into Deep Learning" https://d2l.ai/**