Attention

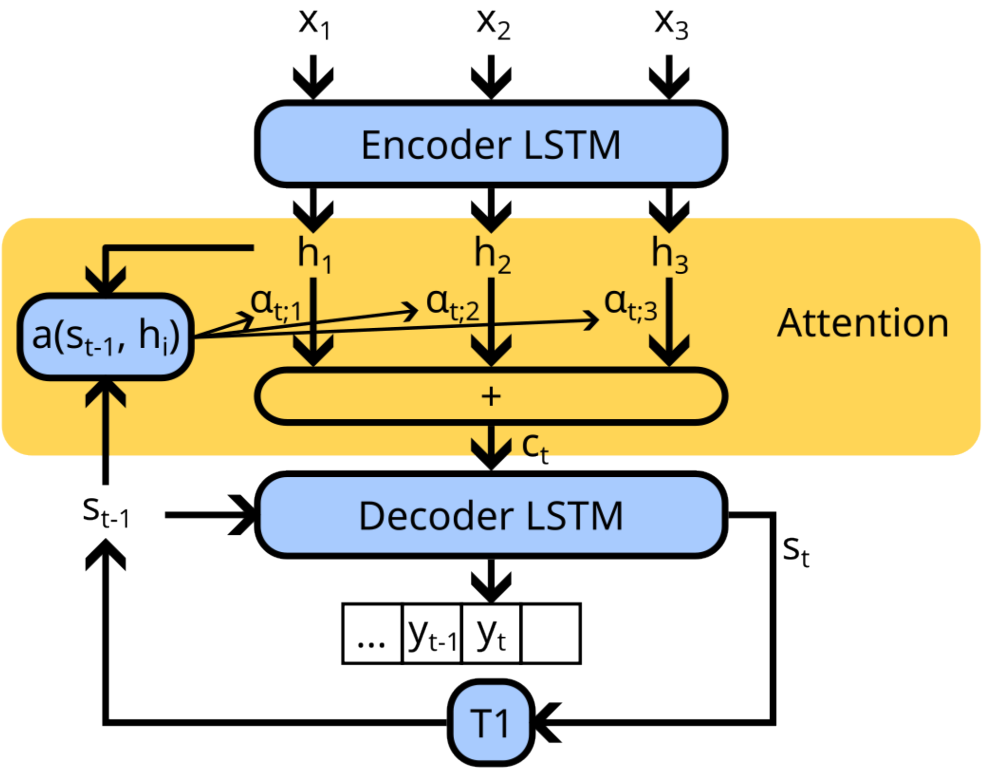
is all you need

Ascii Paper Reading Group

Goal

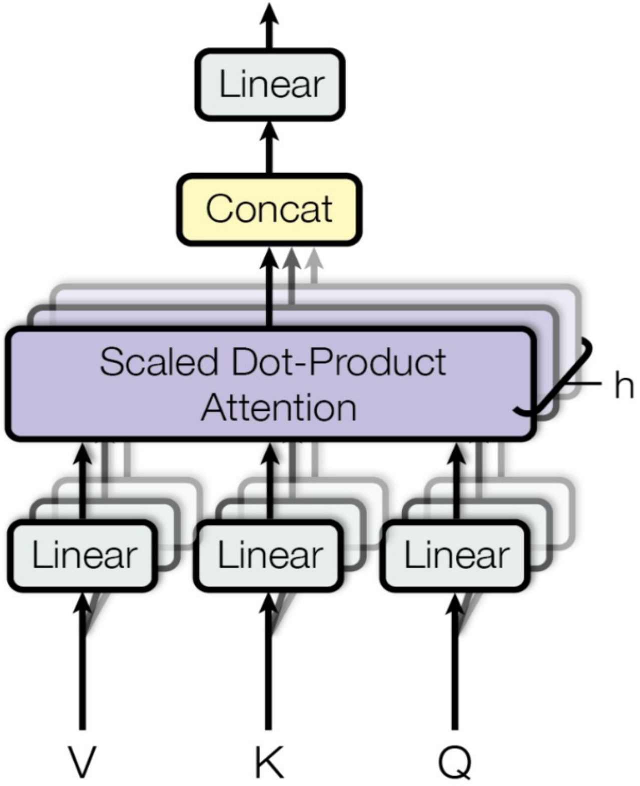
- Machine Translation
  - requires alignment of source with output sentence
- Parallel calculation

Previous Works



Alignment using

Multi Head Attention



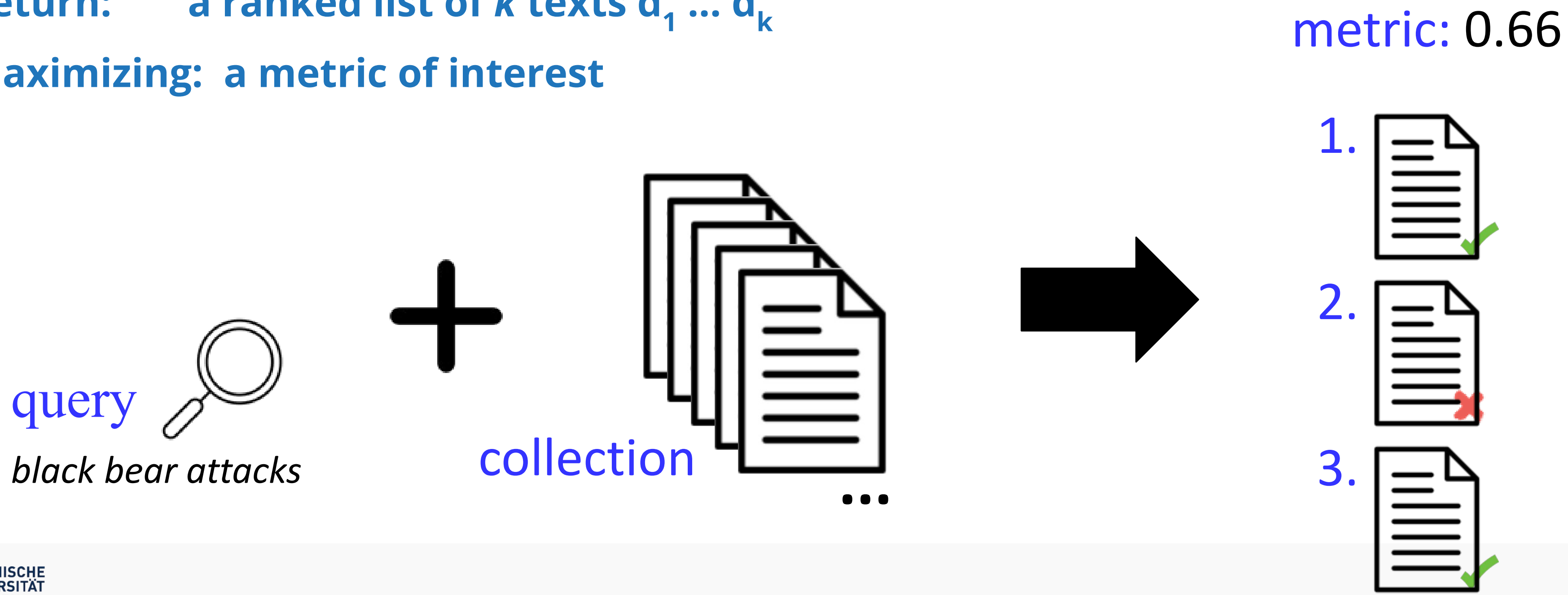
Focus: Ad hoc Retrieval

Given: query q

collection of texts

Return: a ranked list of k texts $d_1 \ldots d_k$

Maximizing: a metric of interest

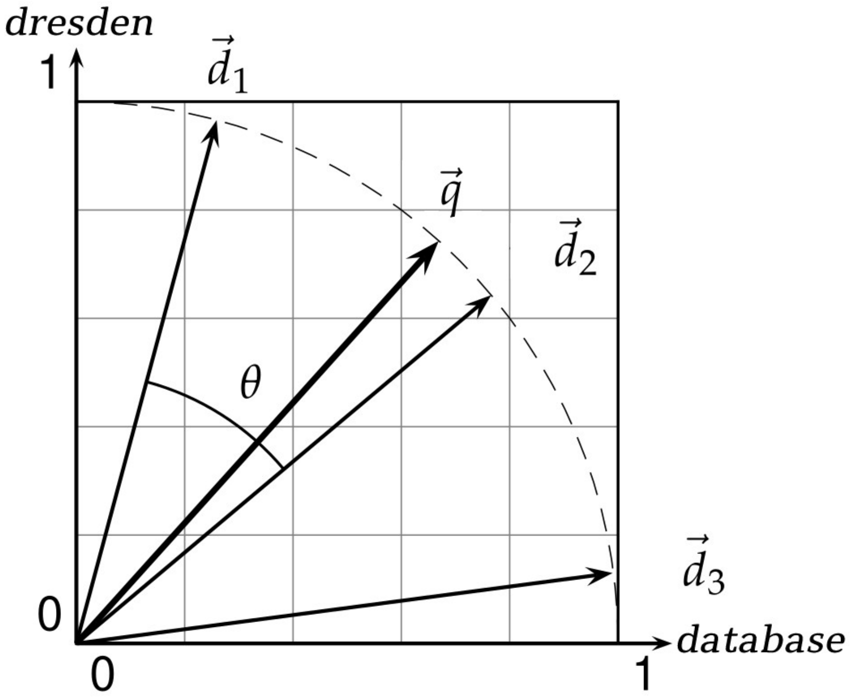


Information Retrieval Basics I

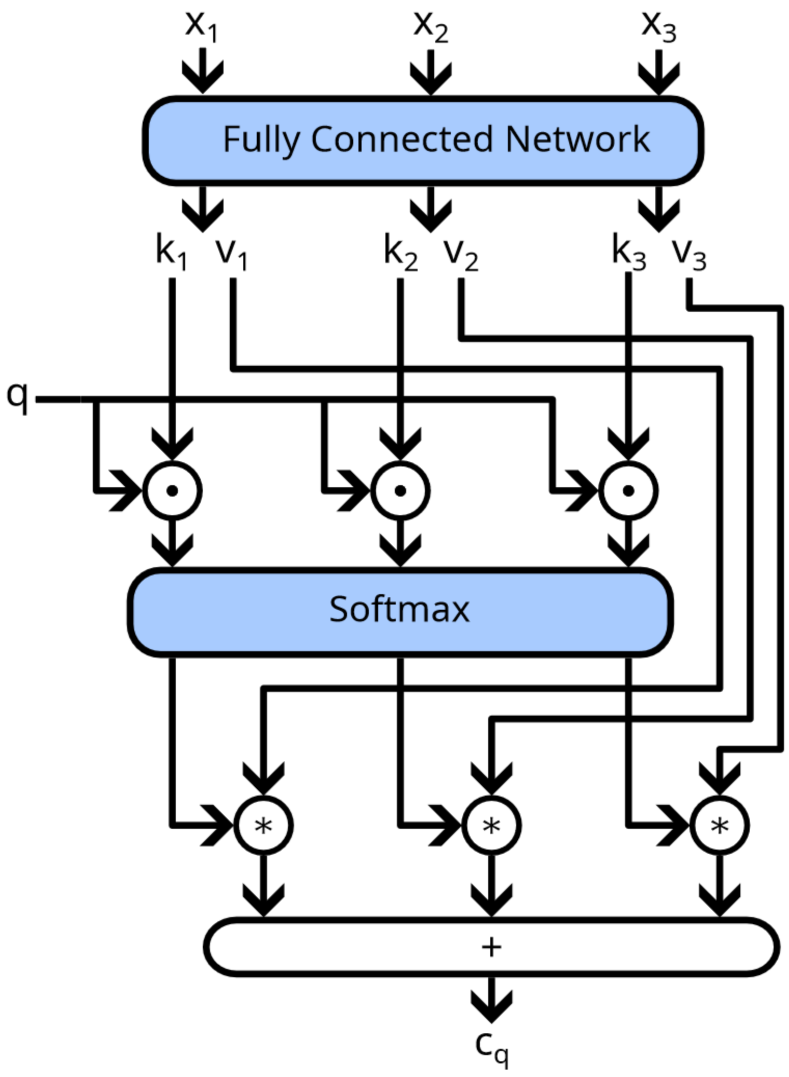
Vector Space Model

- queries *q* + documents *d* represented as vectors
- use cosine similarity between query and documents for ranking

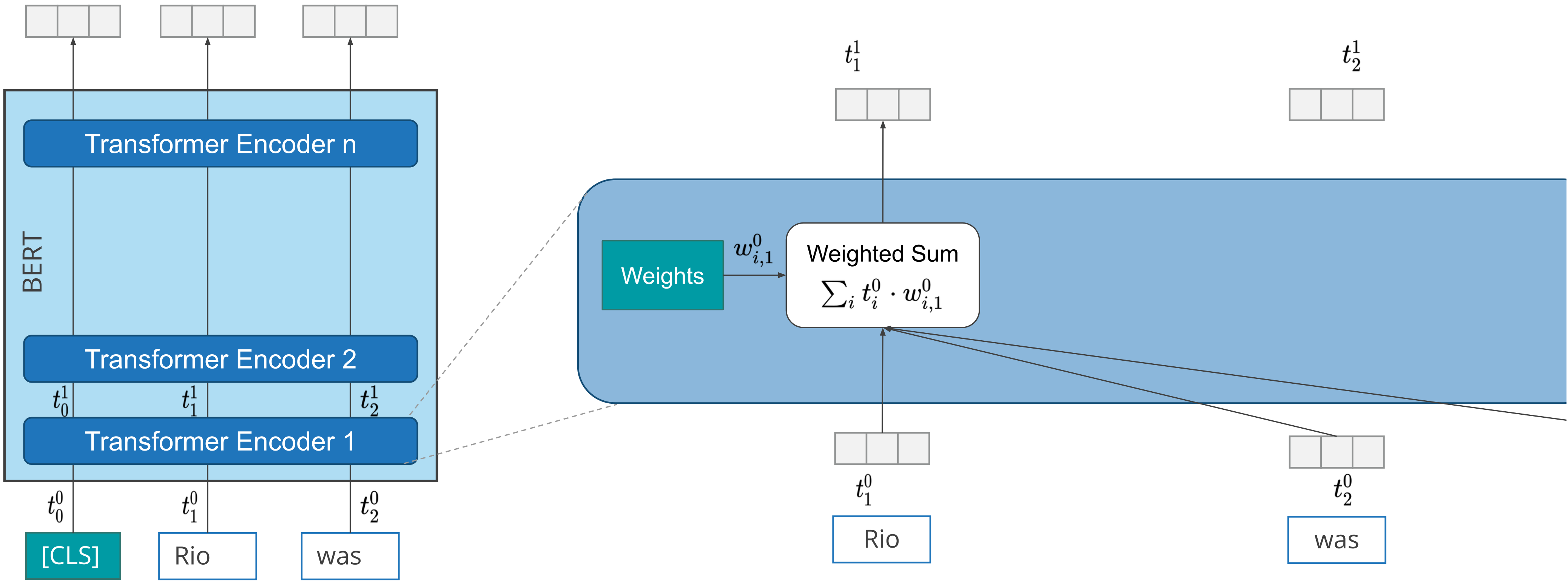
$$\mathrm{sim}(d_1, d_2) = \cos\theta = \frac{\vec{d_1} \cdot \vec{d_2}}{\left|\vec{d_1}\right| \left|\vec{d_2}\right|}$$



Dot Product Attention

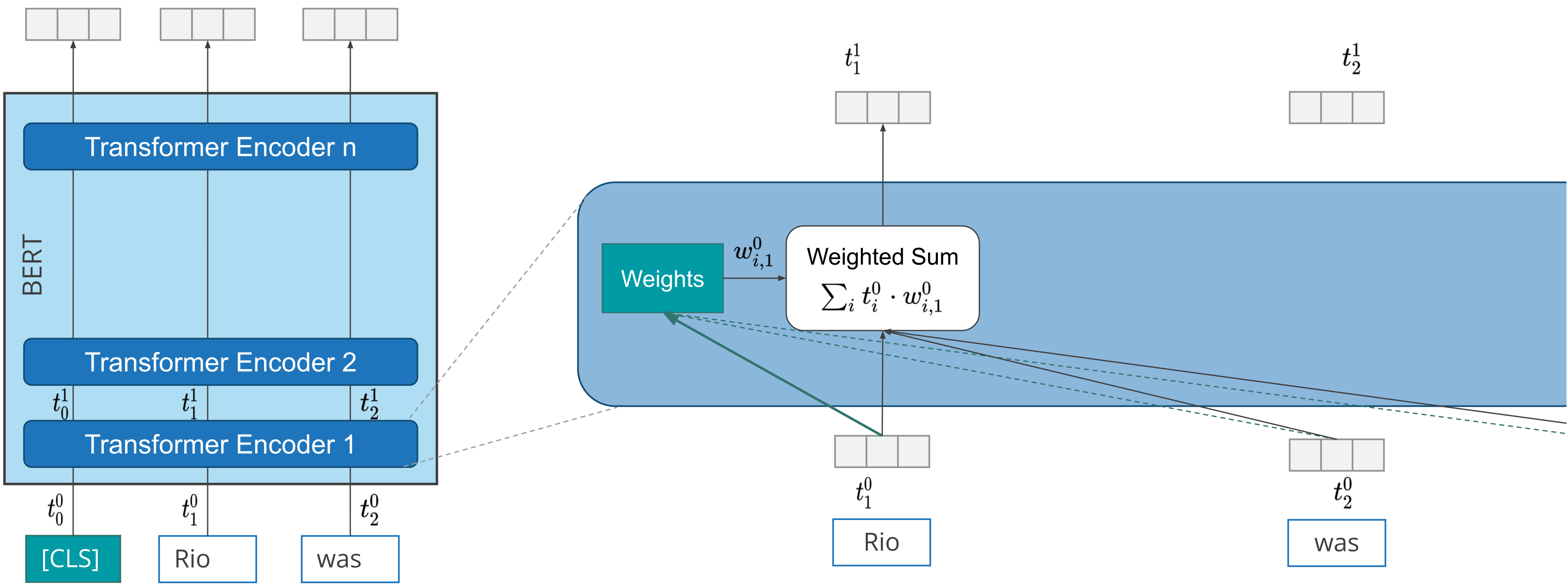


Inside BERT - Attention



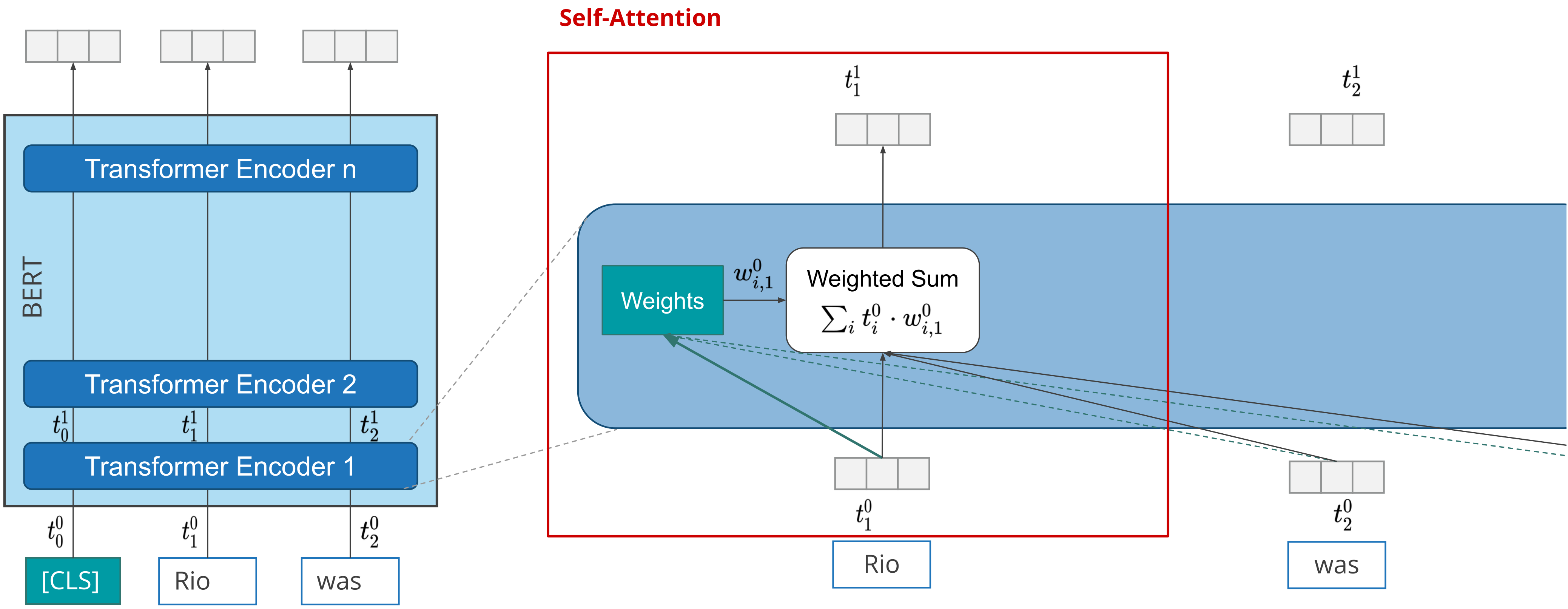
→ Weights determined by current vector (mini neural network is trained here)
→ Context of all other words influences the output vector of token t_i

Inside BERT - Attention



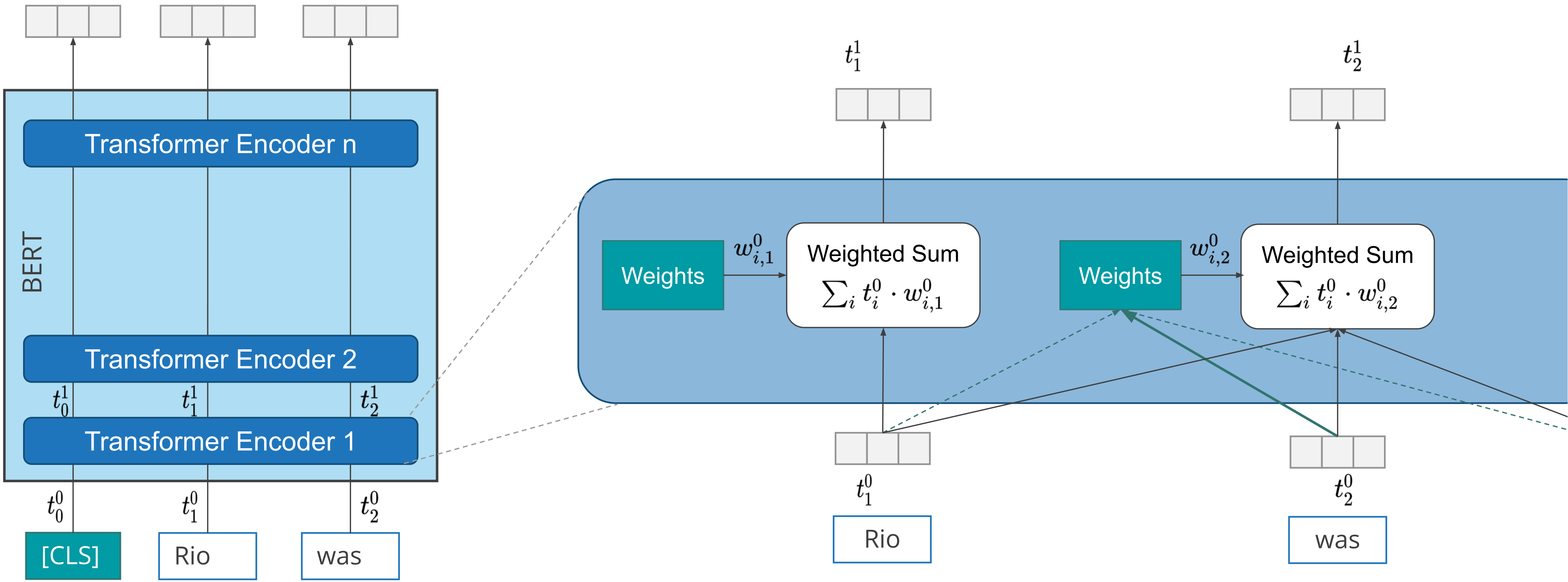
→ Weights determined by current vector (mini neural network is trained here)
→ Context of all other words influences the output vector of token t_i

Inside BERT - Attention



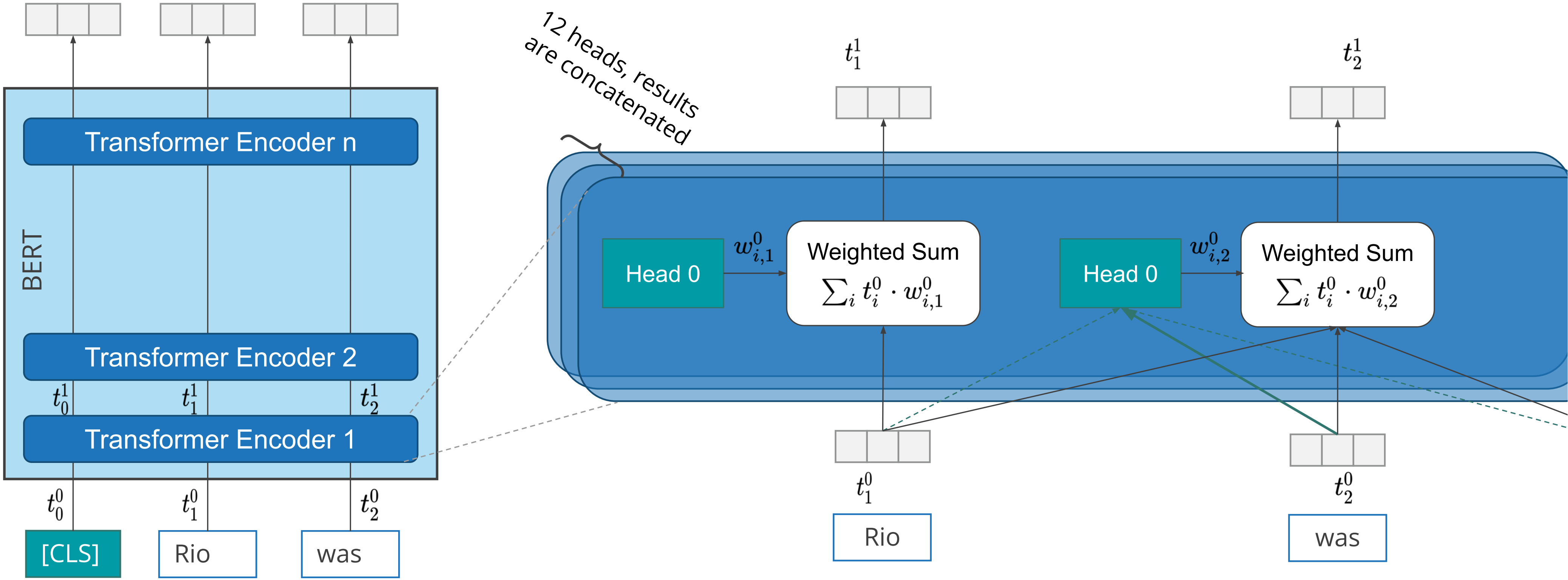
→ Weights determined by current vector (mini neural network is trained here)
→ Context of all other words influences the output vector of token ti

Inside BERT - Attention



→ Weights determined by current vector (mini neural network is trained here)
→ Context of all other words influences the output vector of token t_i

Inside BERT - Attention

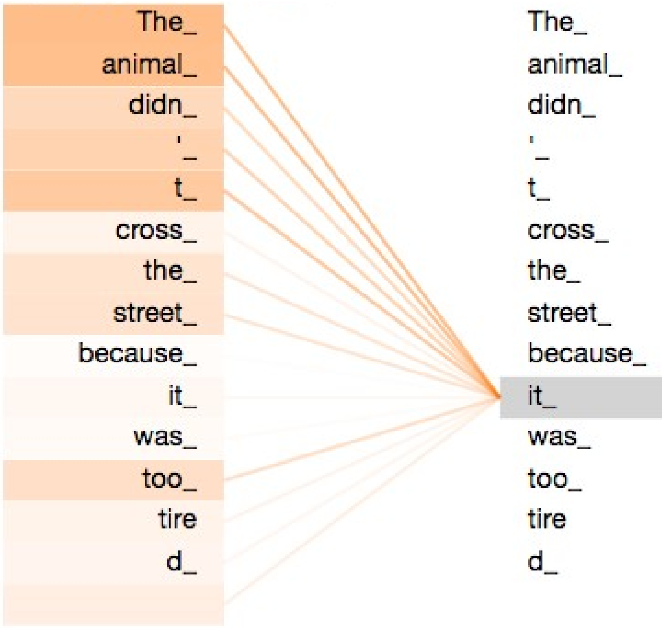


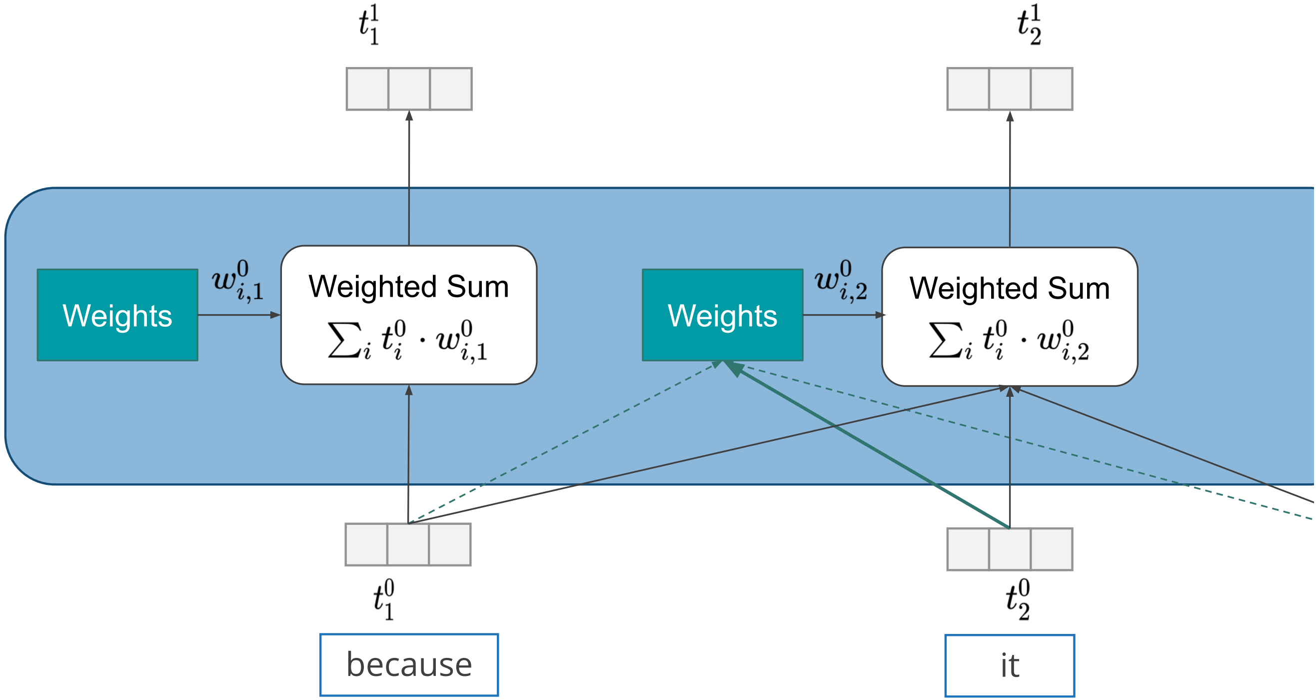
→ Weights determined by current vector (mini neural network is trained here)
→ Context of all other words influences the output vector of token t_i

Inside BERT - Attention

Dresden Database Research Group

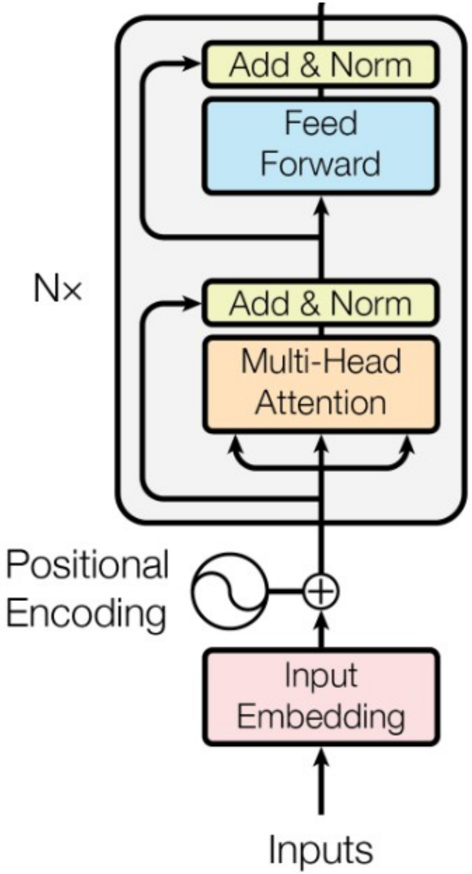
Visualization of weights of one attention head:





→ Weights determined by current vector (mini neural network is trained here)
→ Context of all other words influences the output vector of token t_i

Encoder



- Repeated self attention
- Skip connections for gradient flow
- Normalization for
  - gradient flow
  - Cosine distance

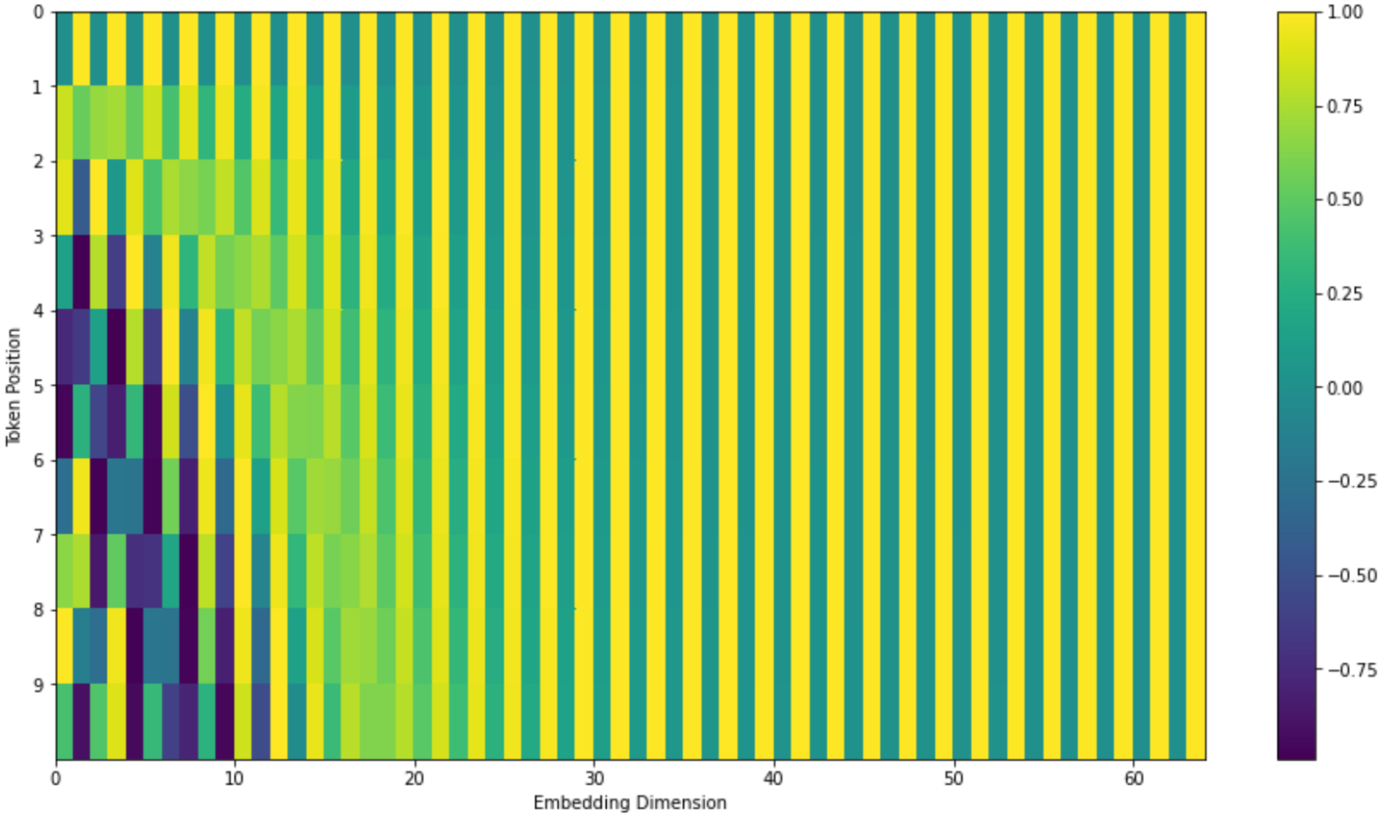
Positional Encoding

- Problem: Dot-Product attention doesn't consider distance of words in a document.
- Add a positional encoding to each token

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

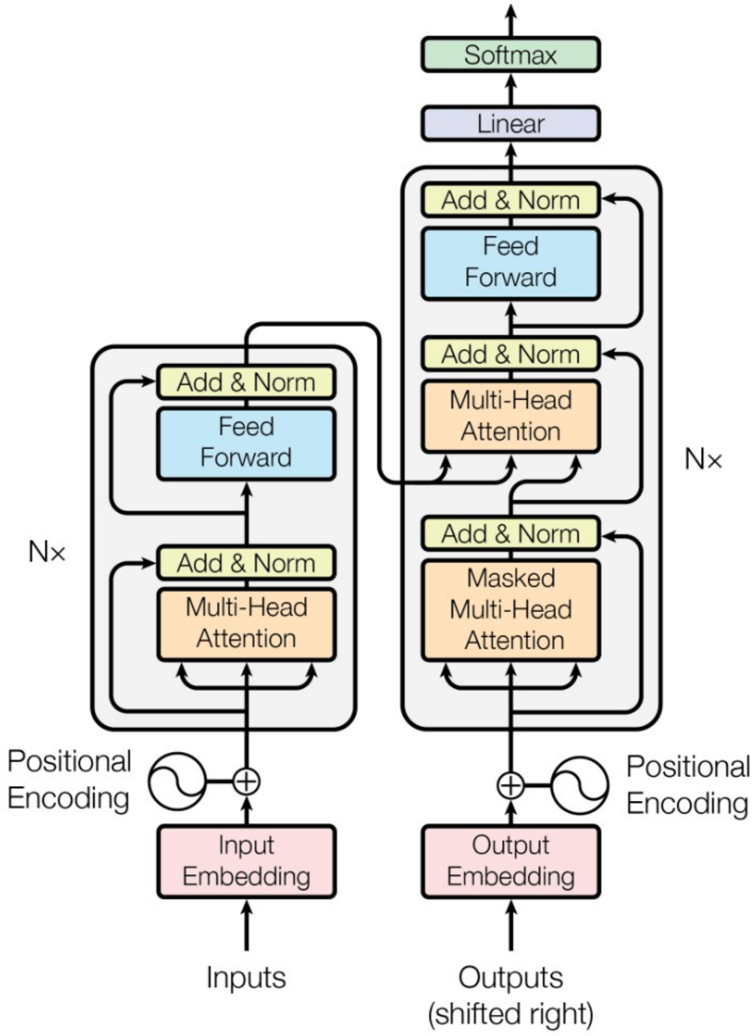
i … dimension

Positional Encoding



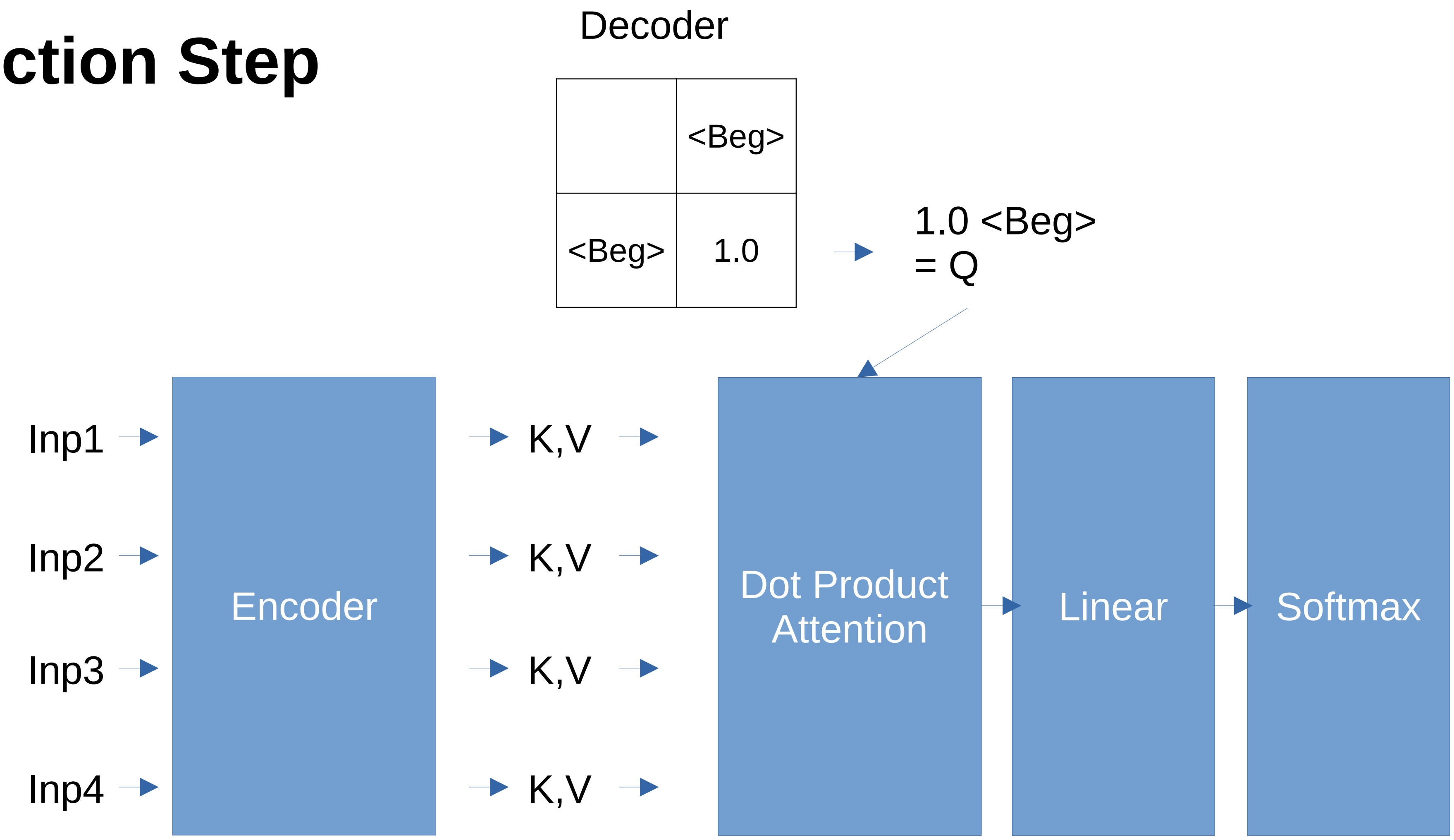
https://jalammar.github.io/illustrated-transformer/

Decoder

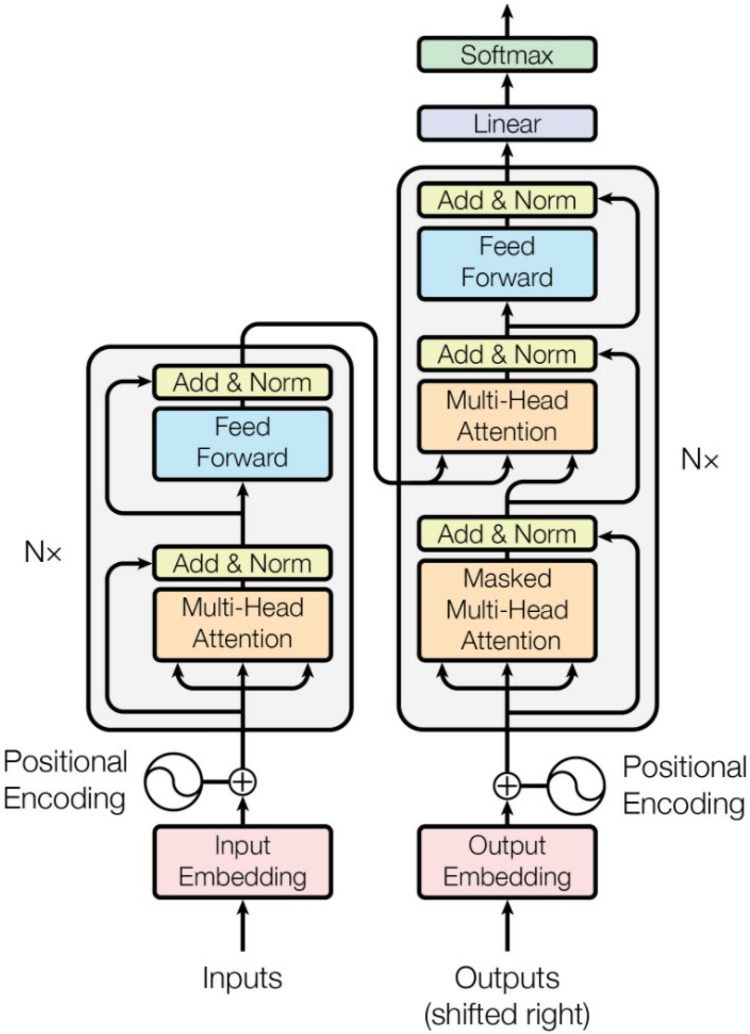


Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
N×
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
N×
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

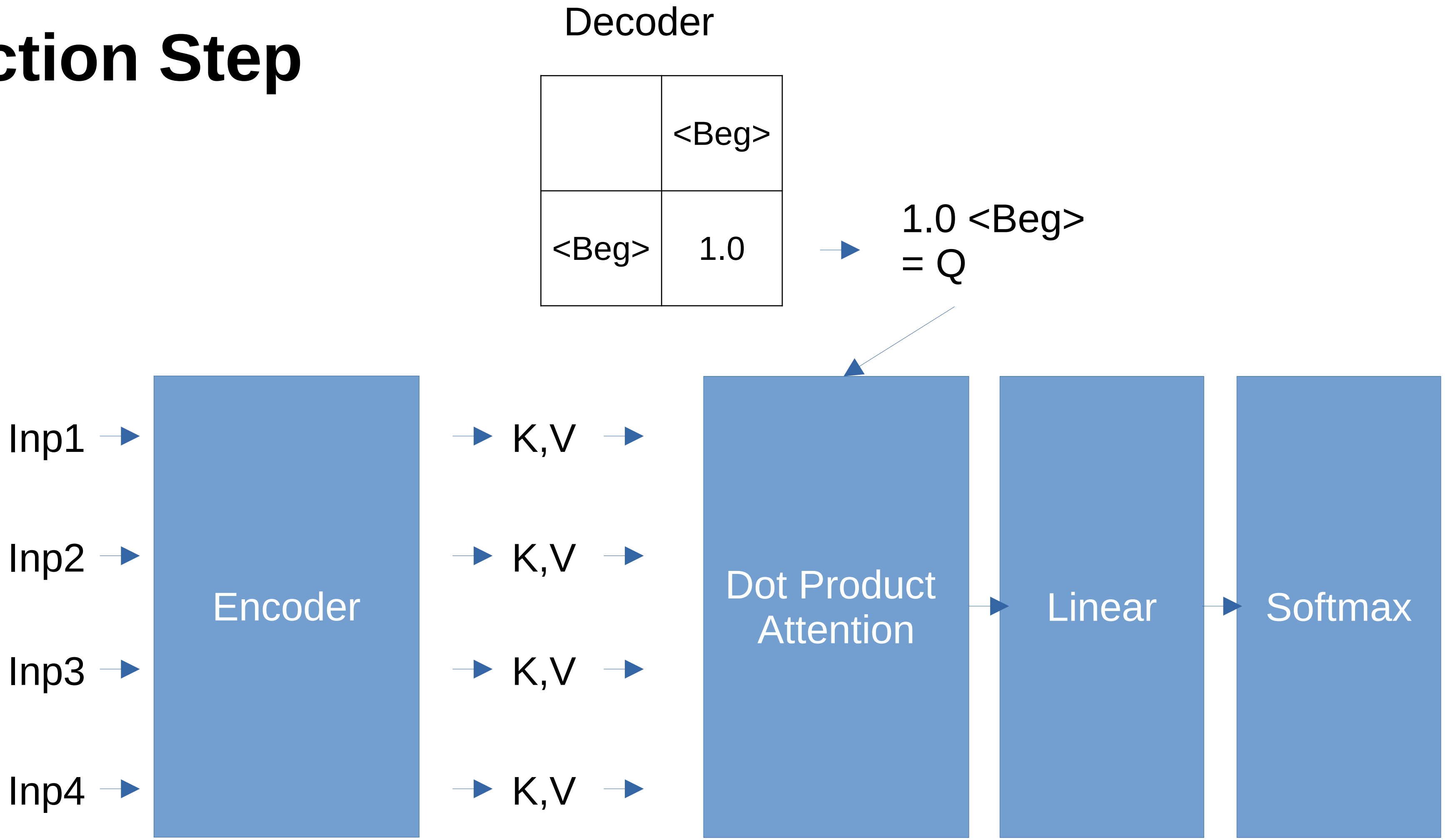
Prediction Step

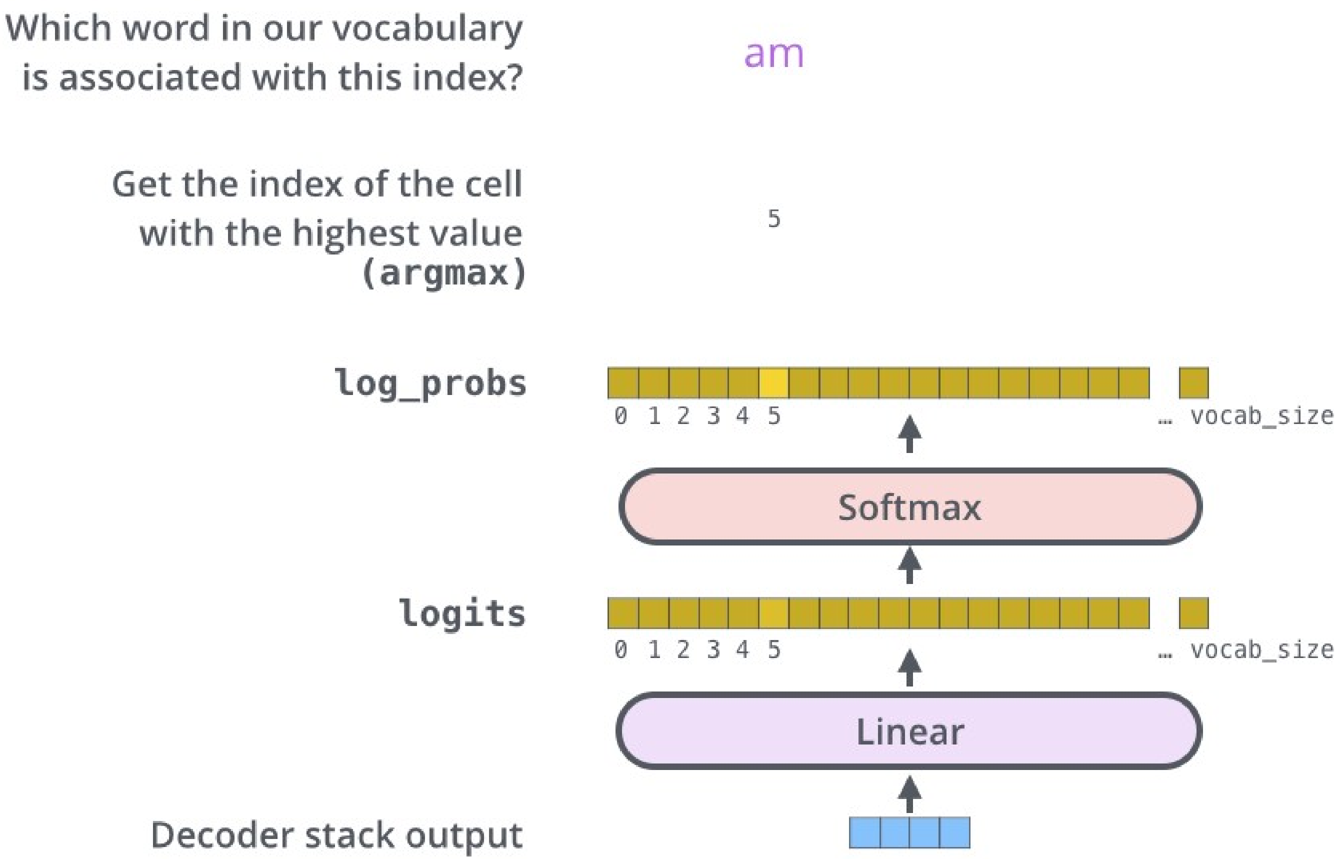


Decoder


Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Prediction Step





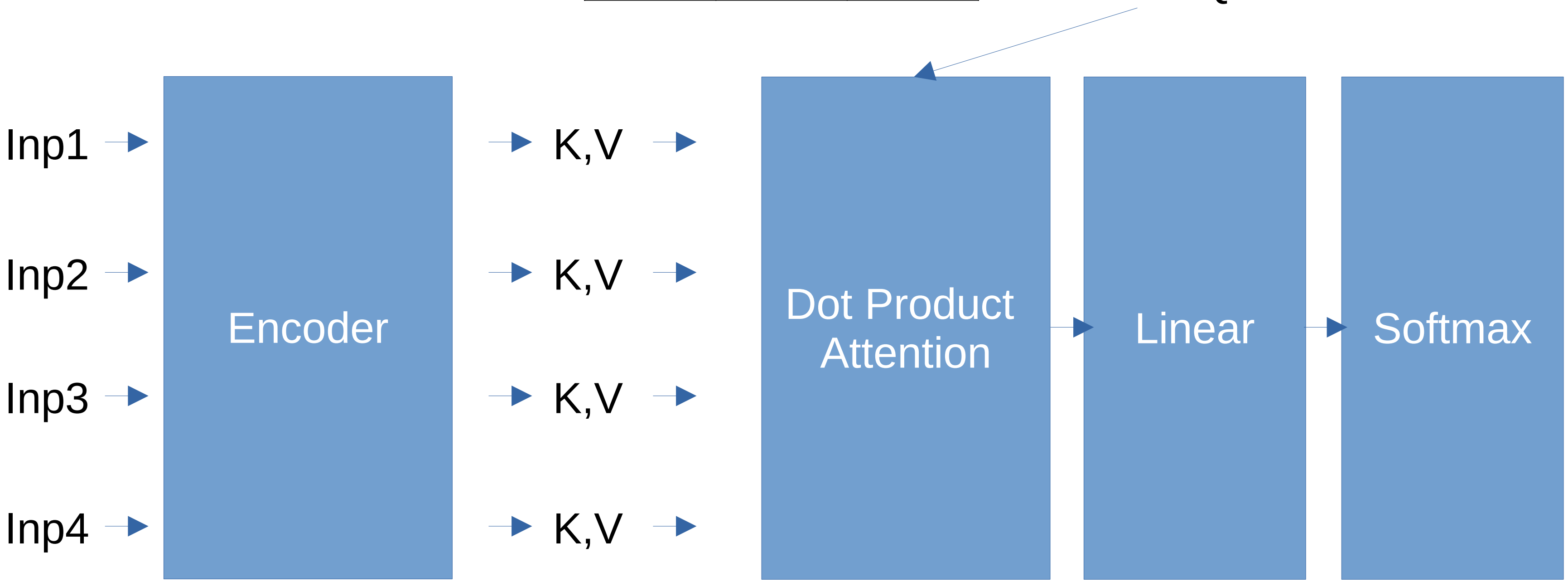
https://jalammar.github.io/illustrated-transformer/

Prediction Step

Decoder

	<Beg>	Out1
Out1	0.3	0.7

0.3 <Beg>
+ 0.7 Out1
= Q

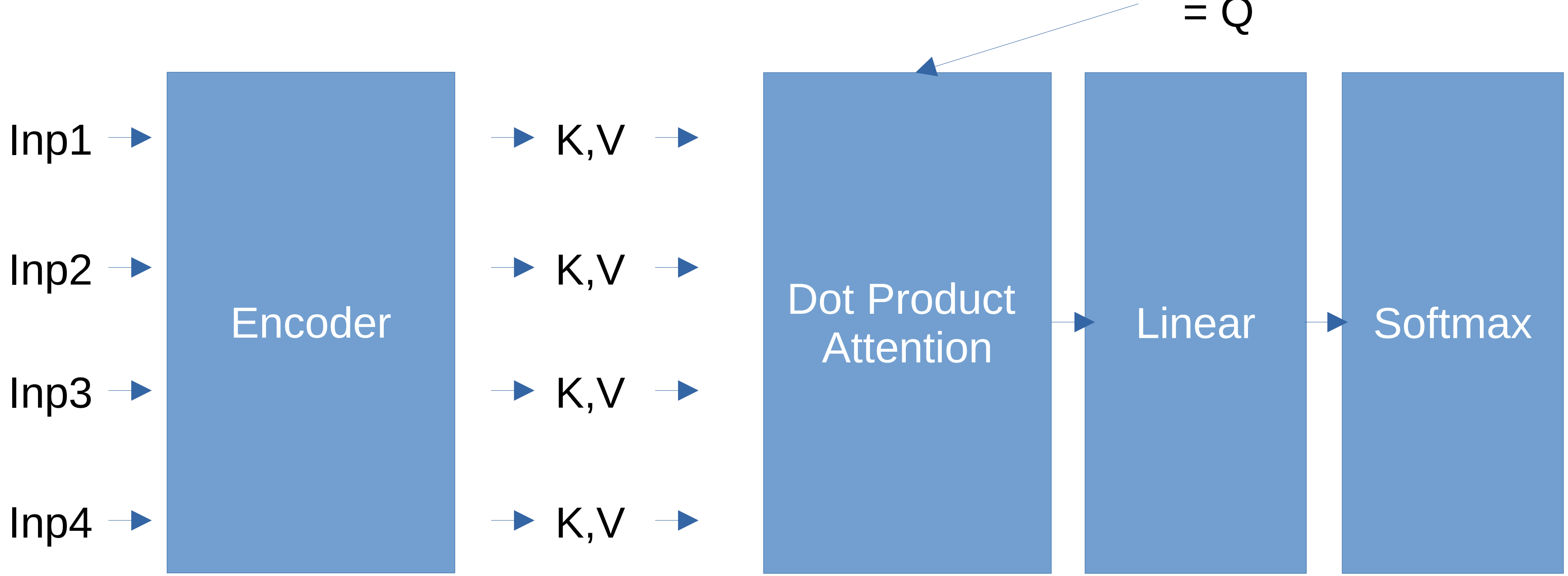


Prediction Step

Decoder

	<Beg>	Out1	Out2
Out2	0.3	0.1	0.6

0.3 <Beg>
+ 0.1 Out1
+ 0.6 Out2
= Q



Training Optimization

- What we saw before
  - Iterative prediction
- During training we know the target sentence
  - We can make all predictions in parallel

Masked Self Attention

	Input1	Input2	Input3	Input4	
Input1	...	...	...	...	→ K,V
Input2	...	...	...	...	→ K,V
Input3	...	...	...	...	→ K,V
Input3	...	...	...	...	→ K,V

	<Beg>	Out1	Out2	Out3	
<Beg>	...				→ Q1
Out1	...	...			→ Q2
Out2	...	...	...		→ Q3
Out3	...	...	...	...	→ Q4

-> parallel prediction of all tokens (training only)