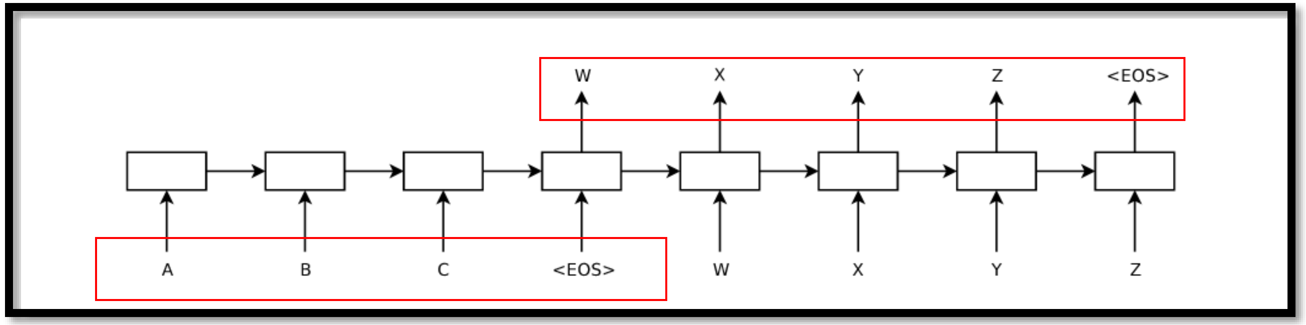
Transformer

Introduced at "Attention is all you need" by Ashish Vaswani et al. (2017)

Recap

seq2seq modeling



Task:

Source sequence ⟶ target sequence

- Introduced for language modeling
Sequence to Sequence Learning with Neural Networks, by Sutskever et. al. , 2014

- Found application at:

* Machine translation (audio/text)
* QA dialogue generation
* Image caption generation

Method:

- Sequential modeling (RNN/CNN)
- Encoder-decoder architecture with fixed-length context vector

Limitations:

- No explicit mechanism for reasoning over structure (imposes an inductive bias to the structure of data).
- Not suitable for long sequences.

Recap

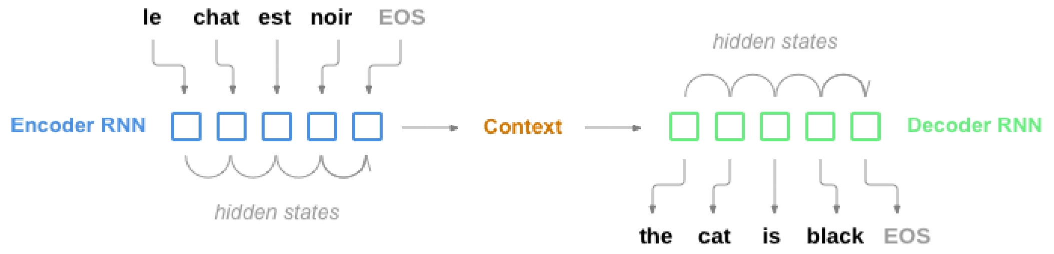
seq2seq modeling

Encoder input: variable-length source sentence

Decoder output: variable-length target sentence

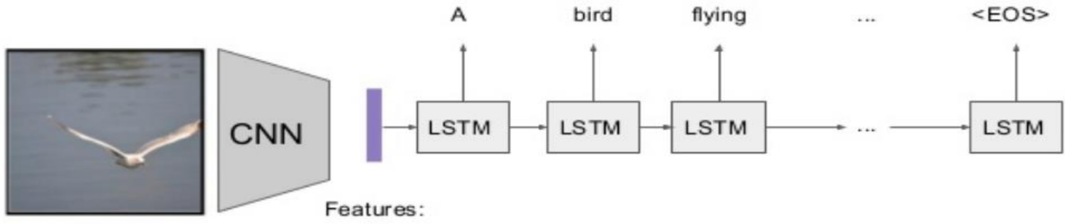
Encoder :

maps input sequence to a context vector



Decoder :

maps the context vector to another sequence



The network compresses all source information into a static fixed-length context vector
And thus all output predictions are based on static output of encoder.

Recap

attention layer

- Introduced for NMT :

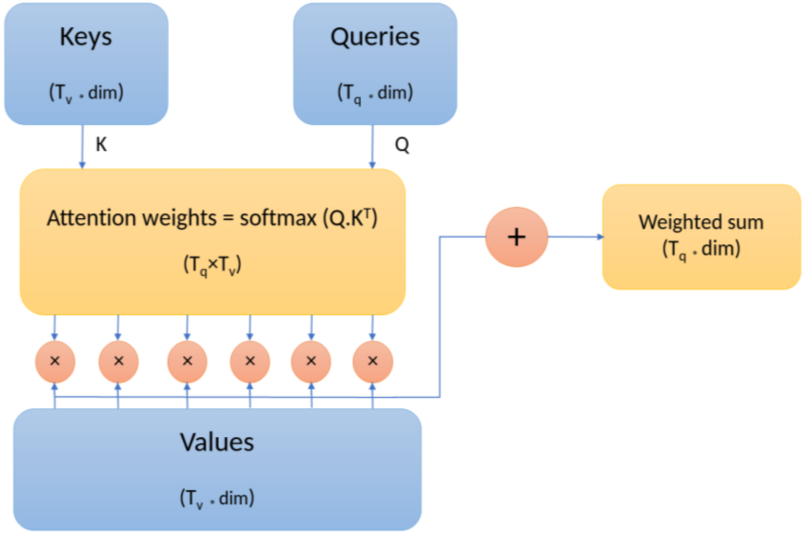
To automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word. The model then predicts a target word based on the context vectors associated with these source positions and all the previous generated target words.

Neural Machine Translation by Jointly Learning to Align and Translate, Bahdanau et. al., 2015

Advantages:

- The model automatically finds correspondence between source and target sequences (alignment)

- Suitable for long sequences



Attention layer returns an output based on input query and its memory.

Recap

attention layer

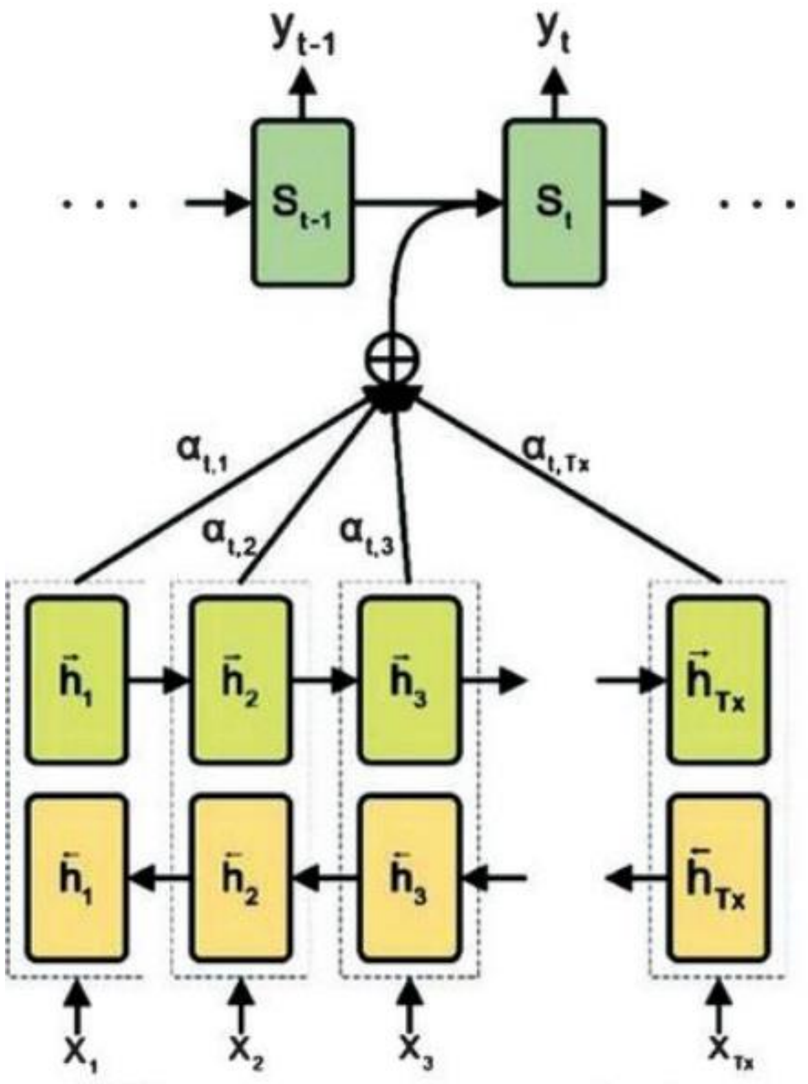
$$p(y_i | y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i),$$

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

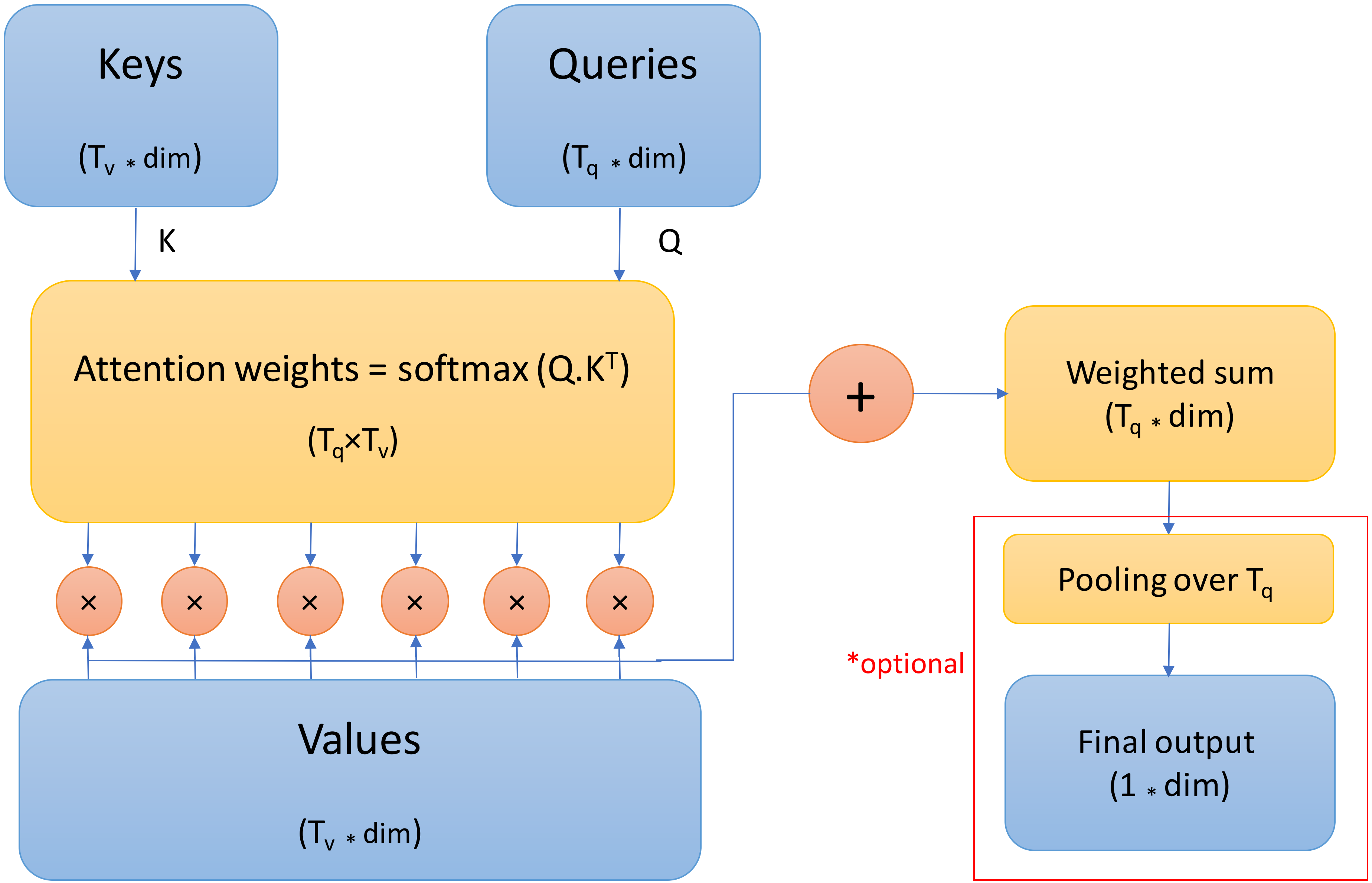
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

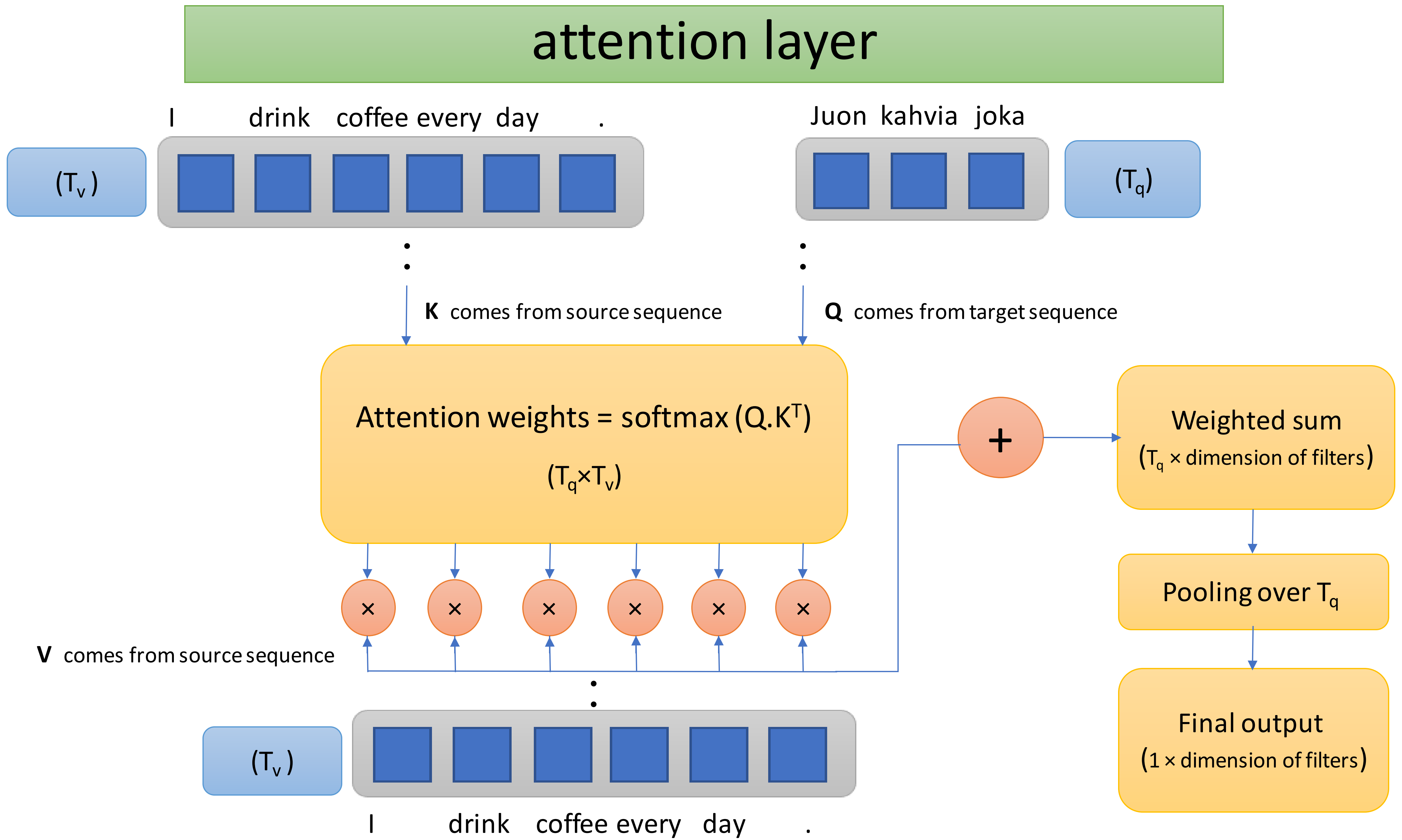
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

$$e_{ij} = a(s_{i-1}, h_j)$$

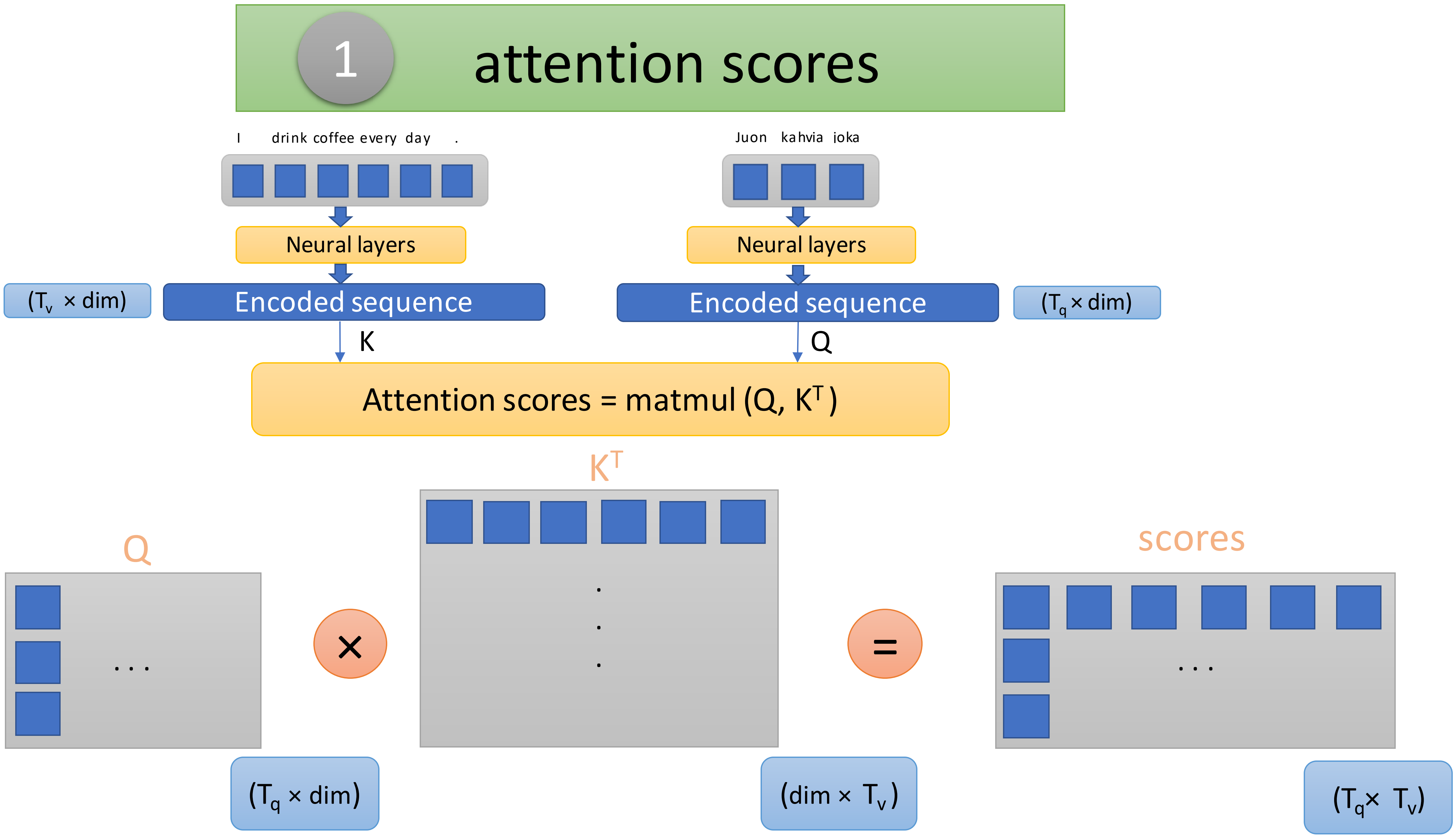


attention layer

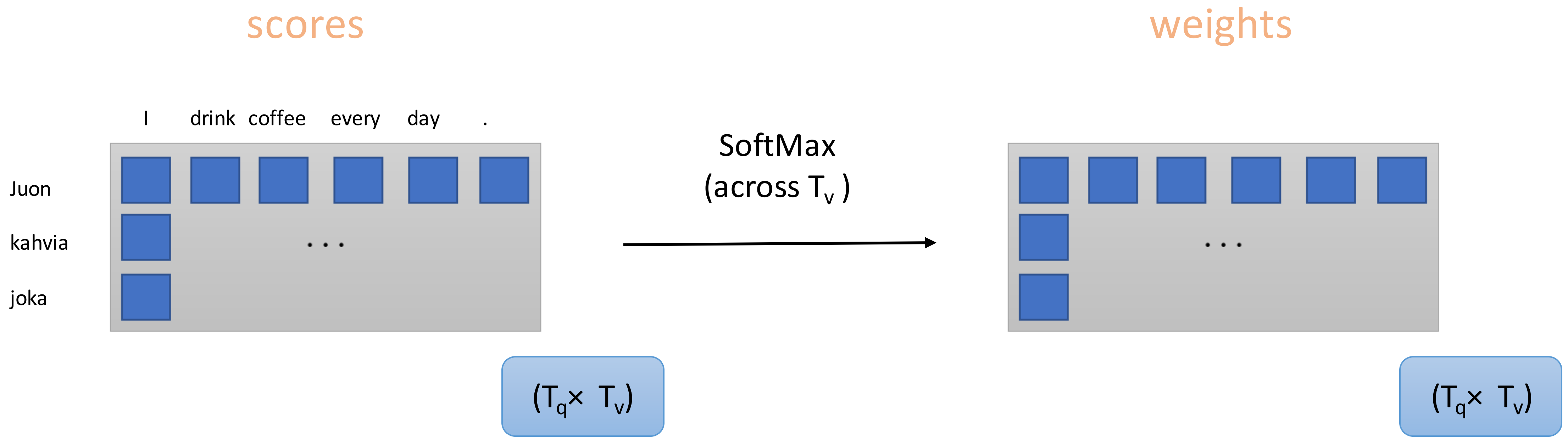




attention layer
I drink coffee every day .
Juon kahvia joka
(T_v)
(T_q)
K comes from source sequence
Q comes from target sequence
Attention weights = softmax $(Q.K^T)$
$(T_q \times T_v)$
+
Weighted sum
$(T_q \times$ dimension of filters$)$
Pooling over T_q
Final output
$(1 \times$ dimension of filters$)$
×
V comes from source sequence
(T_v)
I drink coffee every day .


1
attention scores
I drink coffee every day .
Juon kahvia joka
Neural layers
Neural layers
(T_v × dim)
Encoded sequence
Encoded sequence
(T_q × dim)
K
Q
Attention scores = matmul (Q, K^T)
K^T
Q
scores
...
...
(T_q × dim)
(dim × T_v)
(T_q× T_v)

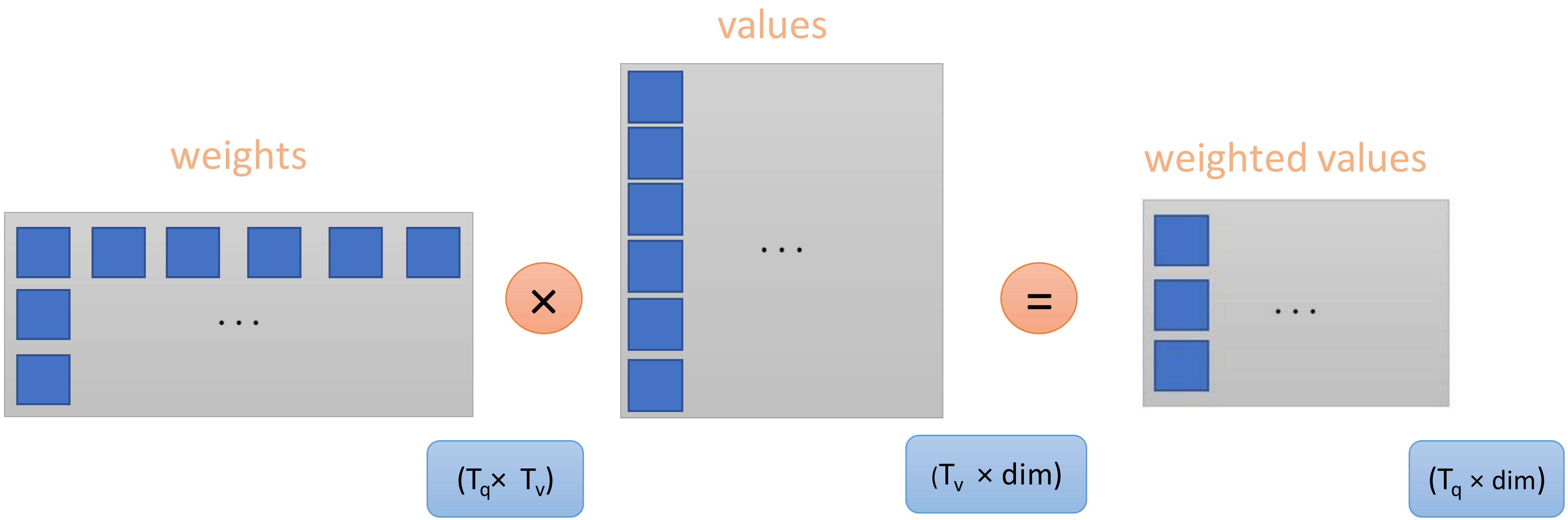
2 attention weights



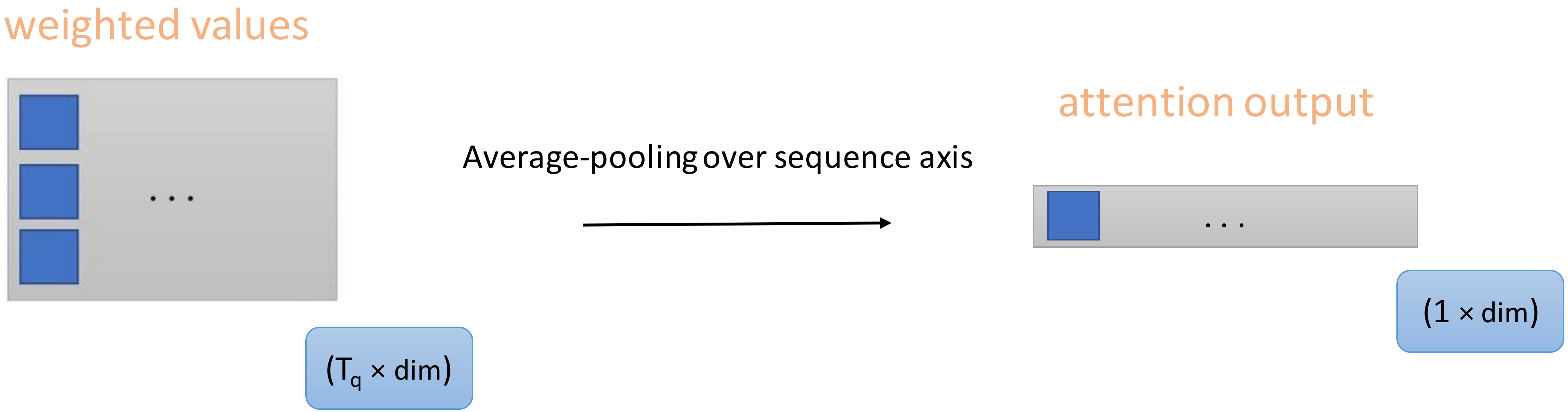
SoftMax = exp(logits) / reduce_sum(exp(logits), axis=-1)

For each query instance (e.g. kahvia) we have a distribution of weights over values.

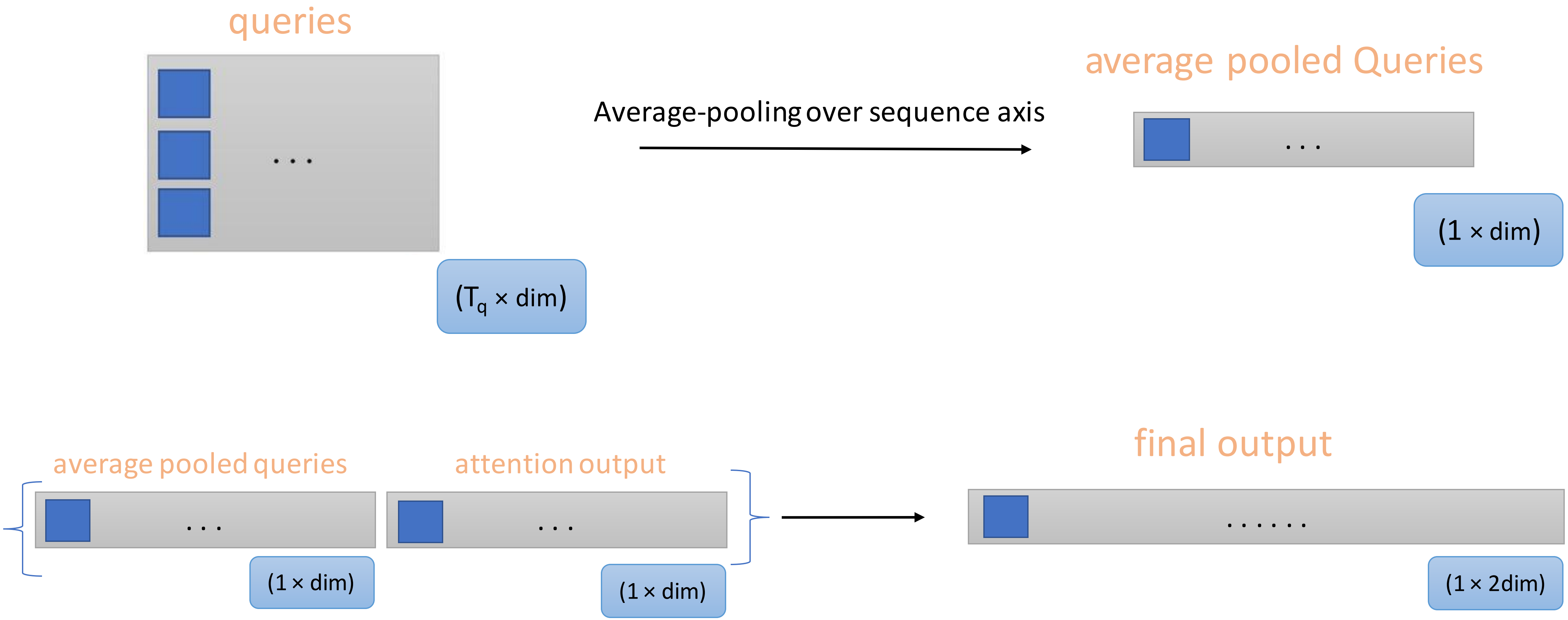
Using weights to create a linear combination of Values



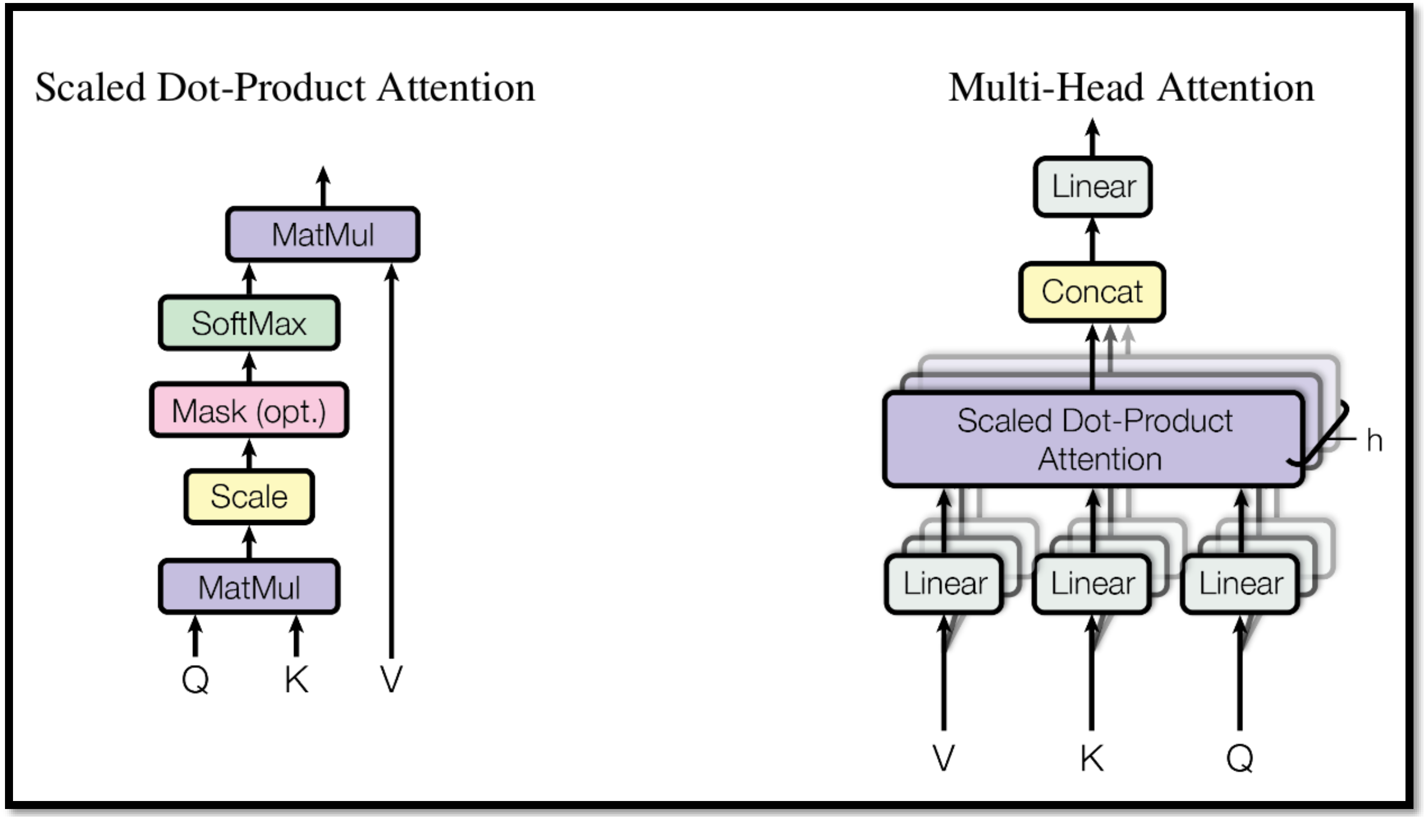
4 average pooling



5 Concatenate attention output with average pooled Queries

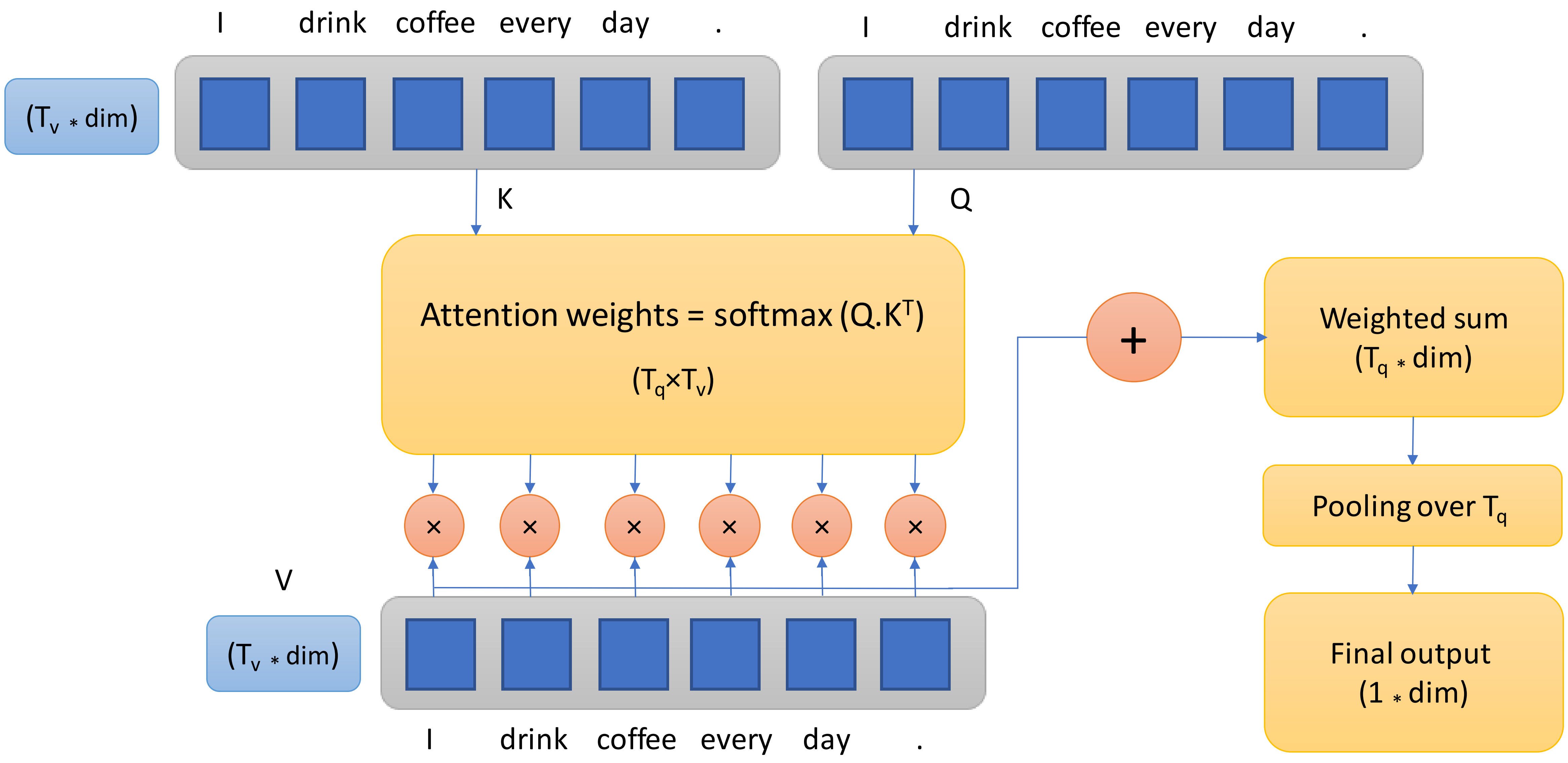


attention layer



$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Self-attention layer



Story of attention mechanism: Aligning functions

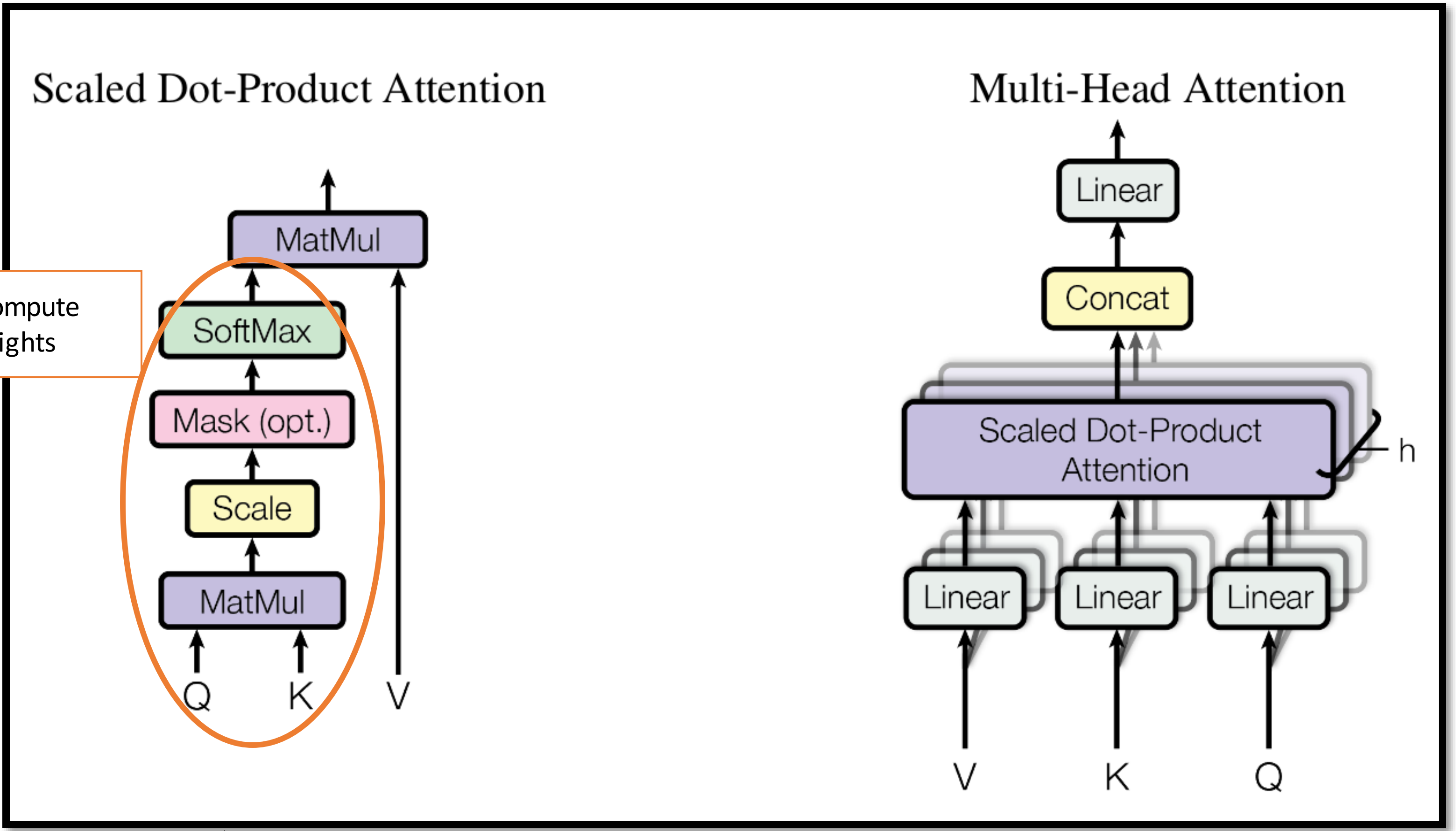
```
scores = tf.reduce_sum(tf.tanh(query  + value), axis=-1)
```

```
scores = tf.matmul(query,  key, transpose_b=True)
```

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

Transformer

(multi-head self-attention)



To compute weights

Weighted sum of values

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

back to d (model) = 512

8 parallel channels of size 64

- attention function is performed in each parallel channel separately
- allows the model to jointly attend to information from different subsets
- works better than averaging
- total computation cost is smaller

d (model) = 512

Thank you!