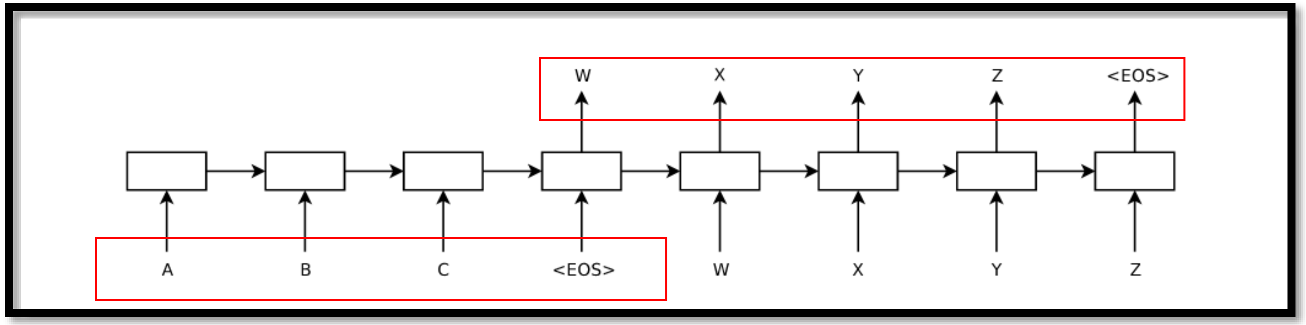
Transformer

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

Recap: sequence-to-sequence 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.

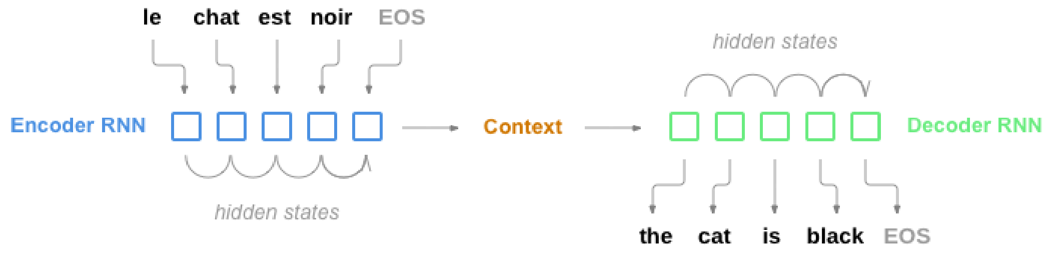
Recap: sequence-to-sequence modeling

Encoder input: variable-length source sentence

Decoder output: variable-length target sentence

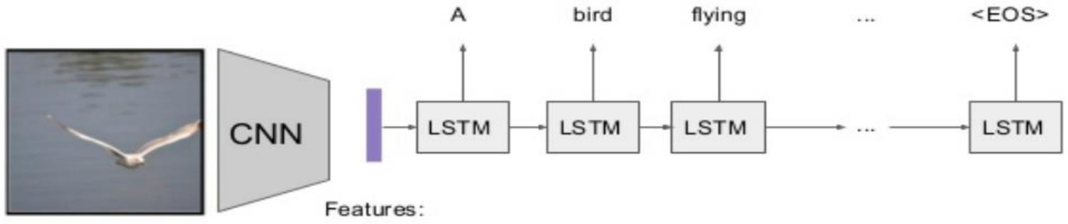
Encoder :

maps input sequence to a context vector



Decoder :

maps the context vector to another sequence



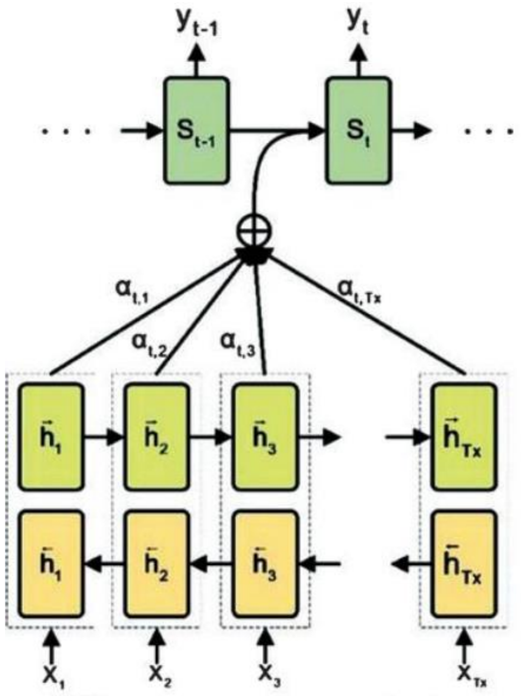
The network compresses all source information into a static fixed-length context vector
And thus all output predictions are based on static output of encoder.

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



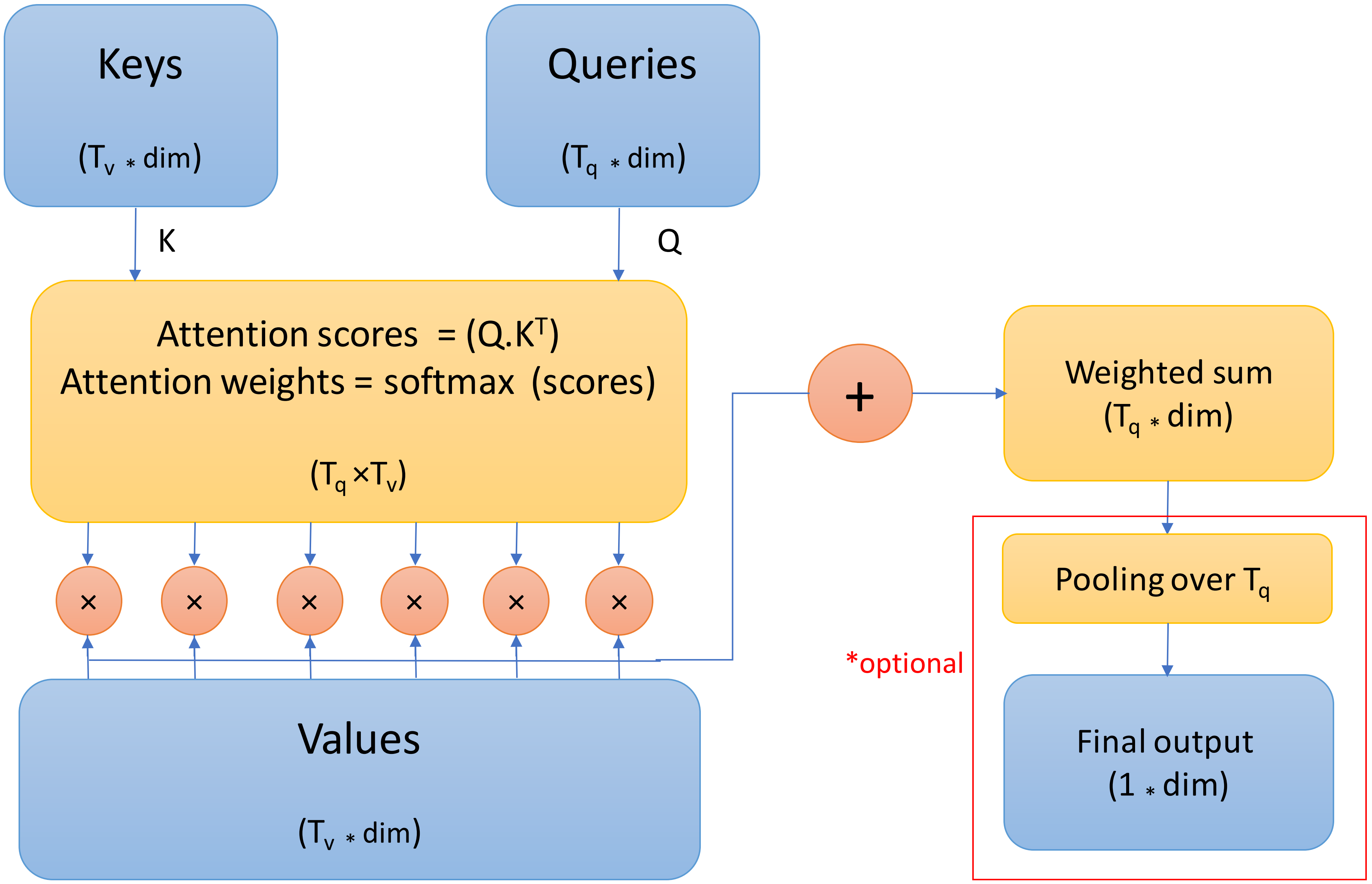
Advantages:

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

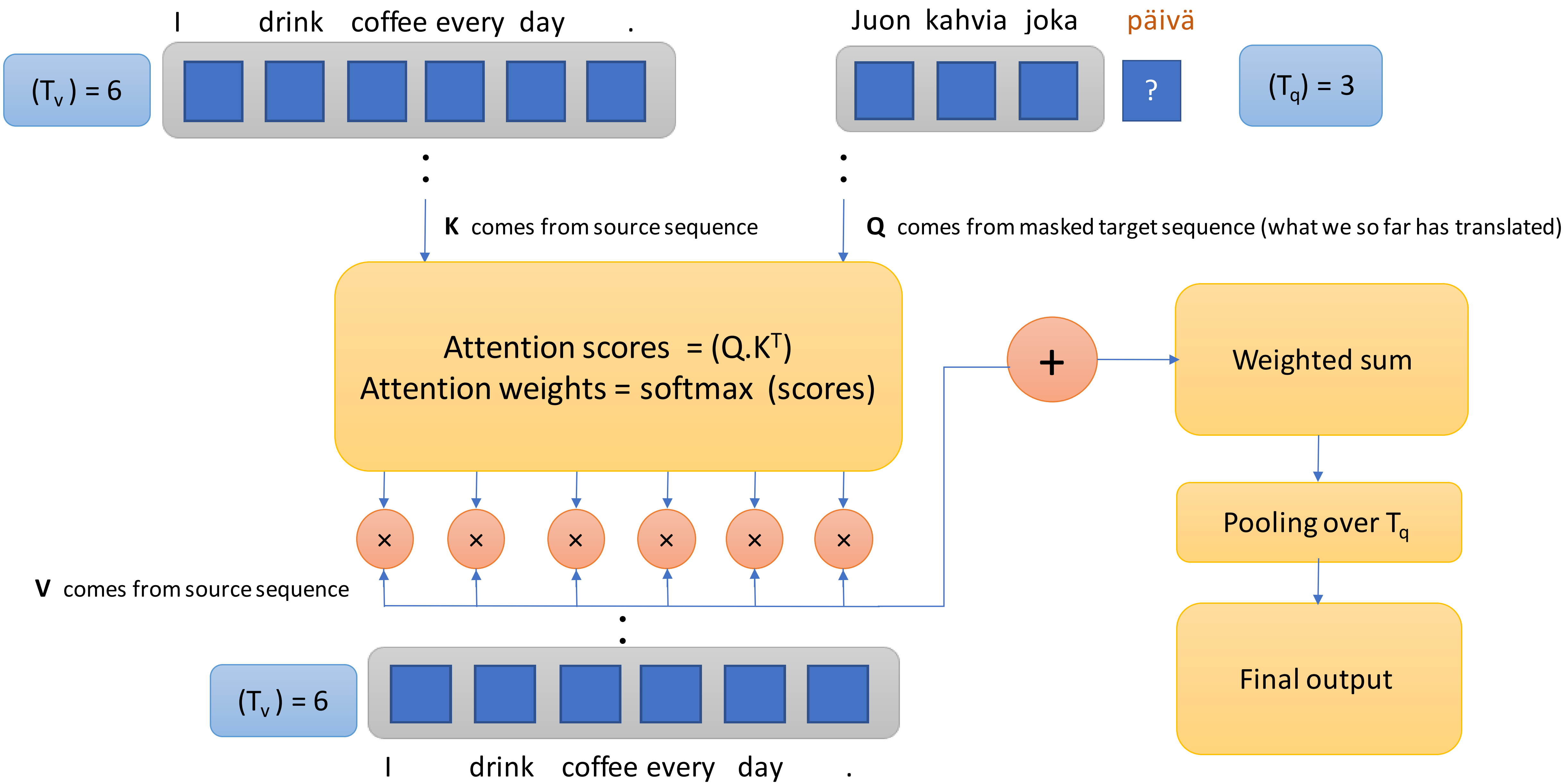
- Suitable for long sequences

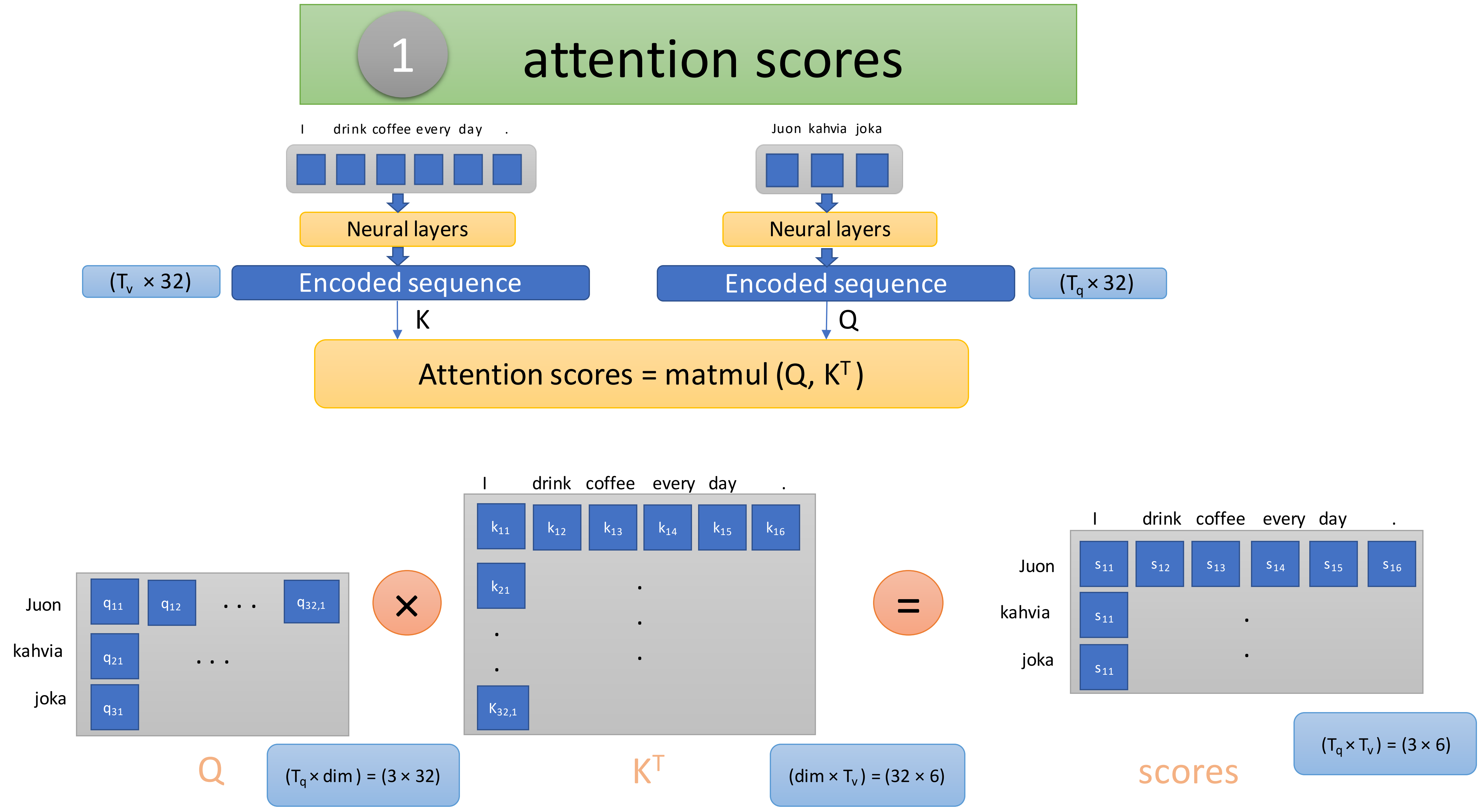
Attention layer returns an output based on input query and its memory.

Dot product attention layer

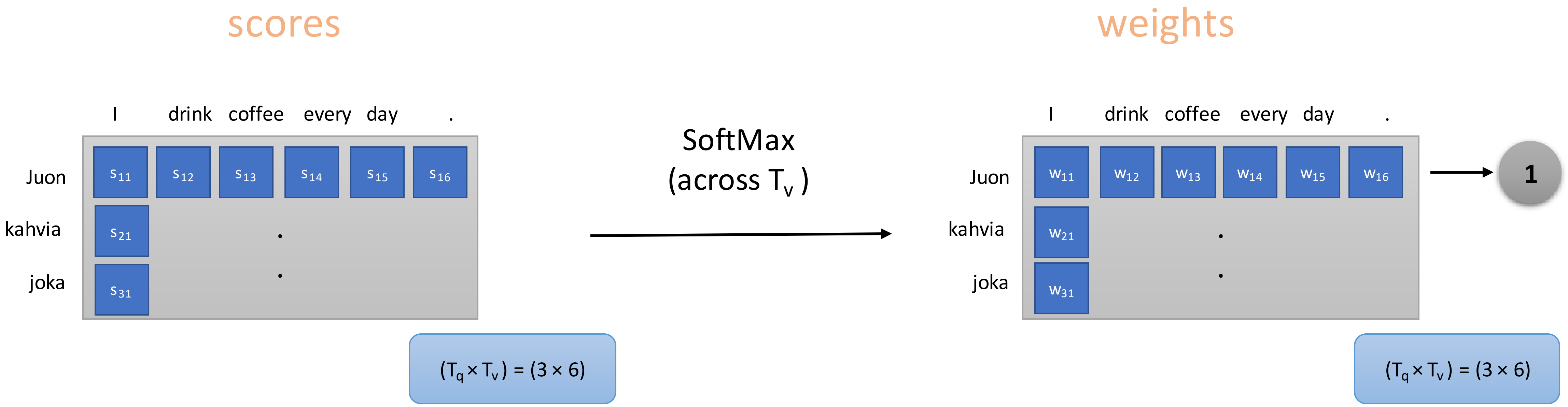


Dot product attention layer



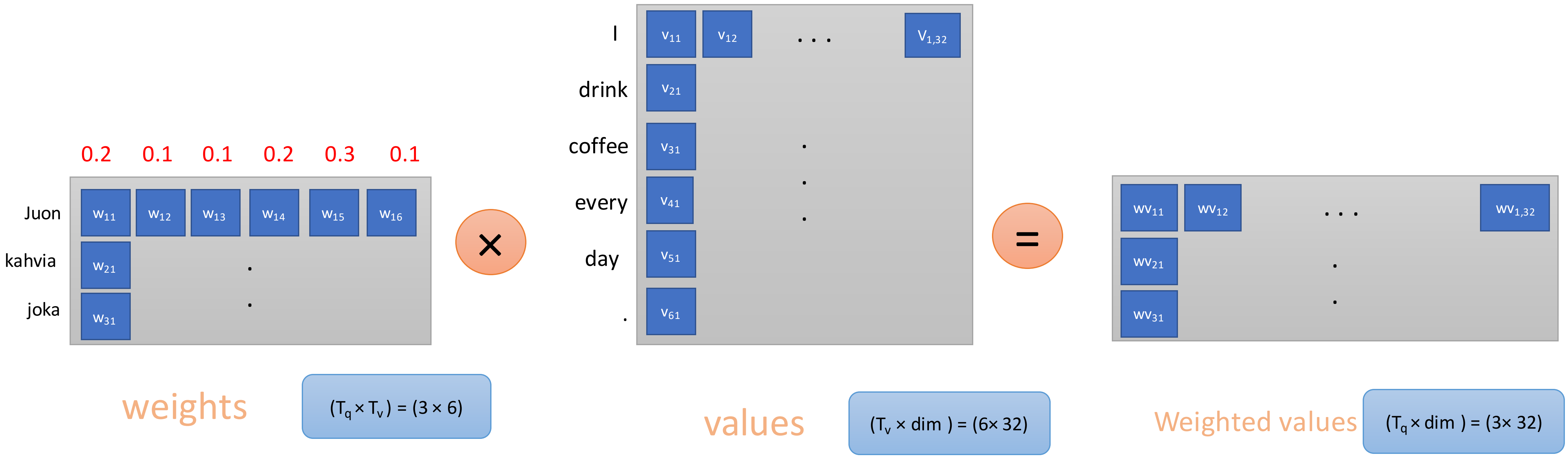

1
attention scores
I drink coffee every day .
Juon kahvia joka
Neural layers
Neural layers
(T_v × 32)
Encoded sequence
Encoded sequence
(T_q × 32)
K
Q
Attention scores = matmul (Q, K^T)
Juon
kahvia
joka
q11
q12
. . .
q32,1
q21
. . .
q31
×
I drink coffee every day .
k11
k12
k13
k14
k15
k16
k21
K32,1
=
I drink coffee every day .
Juon
kahvia
joka
s11
s12
s13
s14
s15
s16
s11
s11
Q
(T_q × dim) = (3 × 32)
K^T
(dim × T_v) = (32 × 6)
scores
(T_q × T_v) = (3 × 6)

2 attention weights

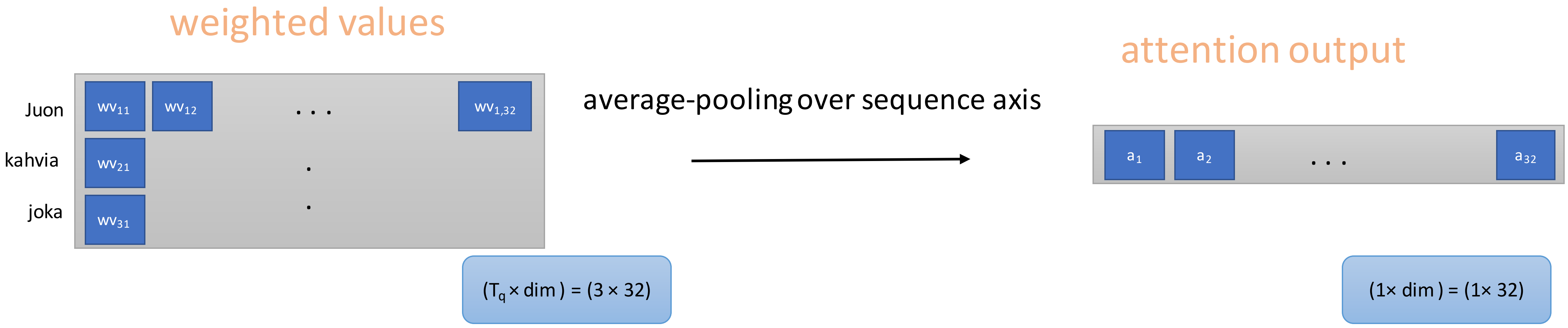


For each query instance (e.g. Juon) we have a distribution of weights over all values, (e.g. I, drink, coffee, every, day)

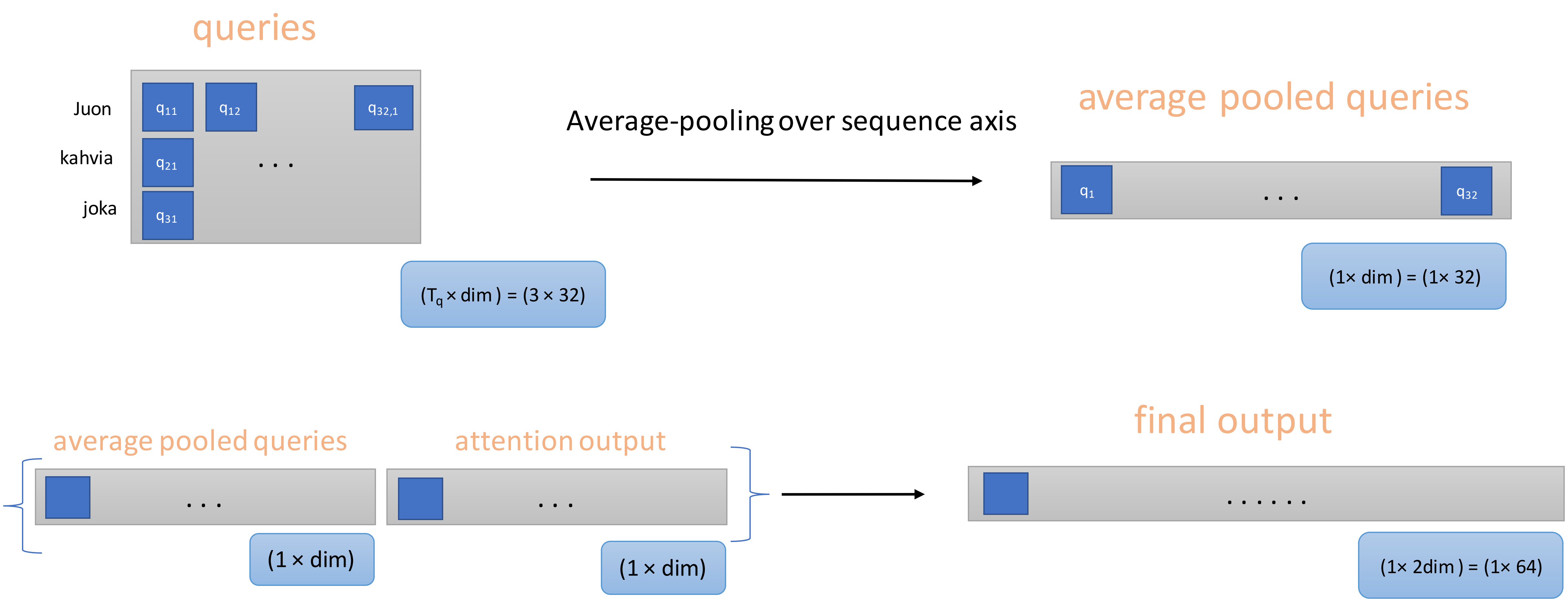
3 Using weights to create a linear combination of Values



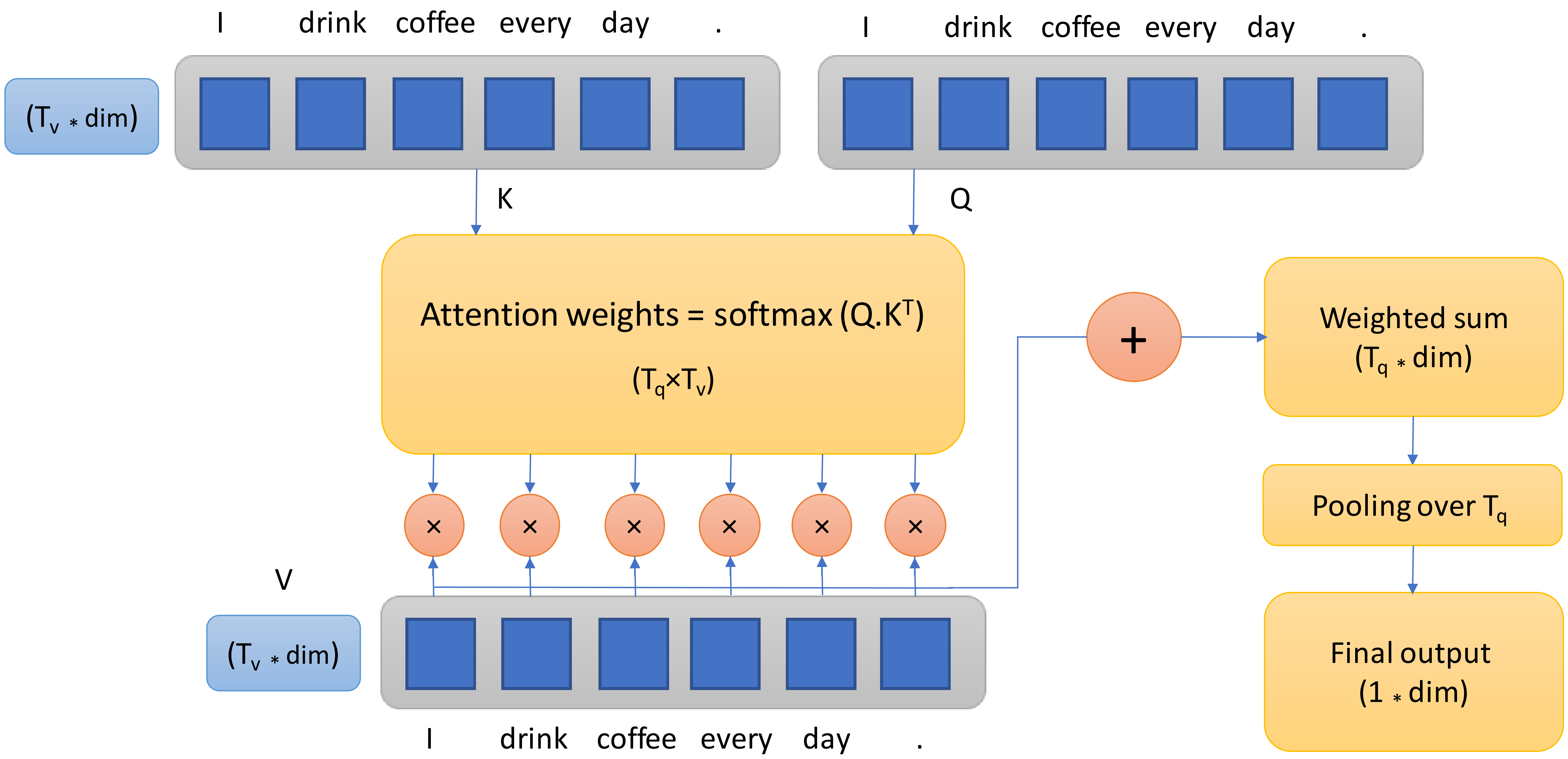
4 average pooling



5 Concatenate attention output with average pooled Queries



Self-attention layer



Aligning functions

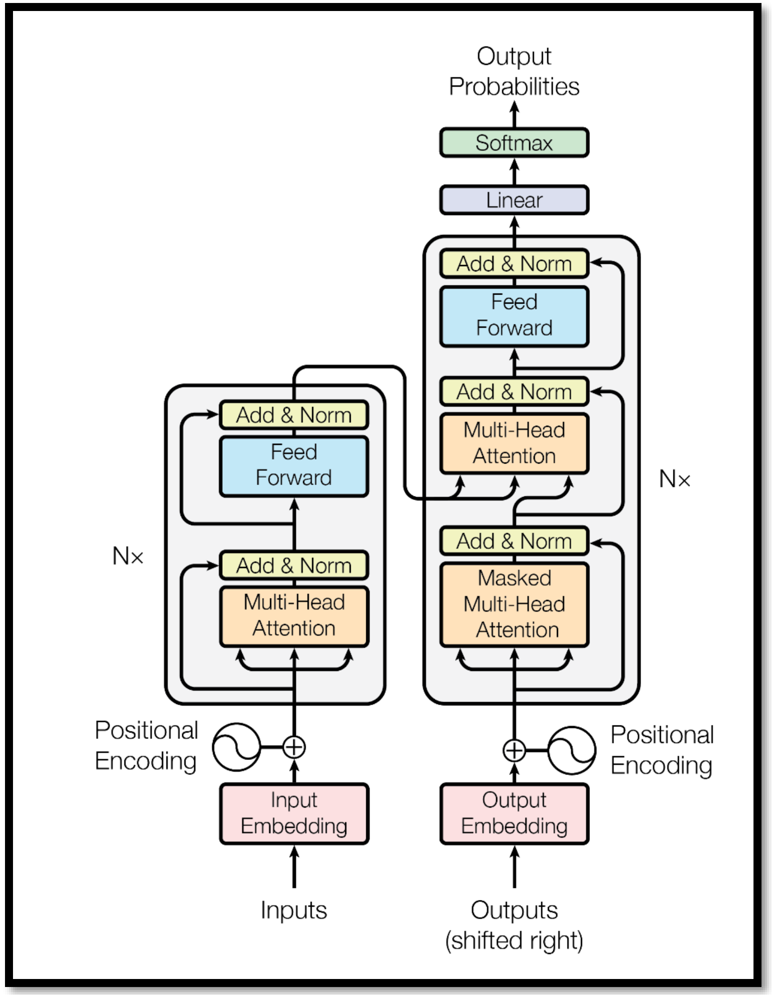
`scores = tf.reduce_sum(tf.tanh(query + value), axis=-1)`

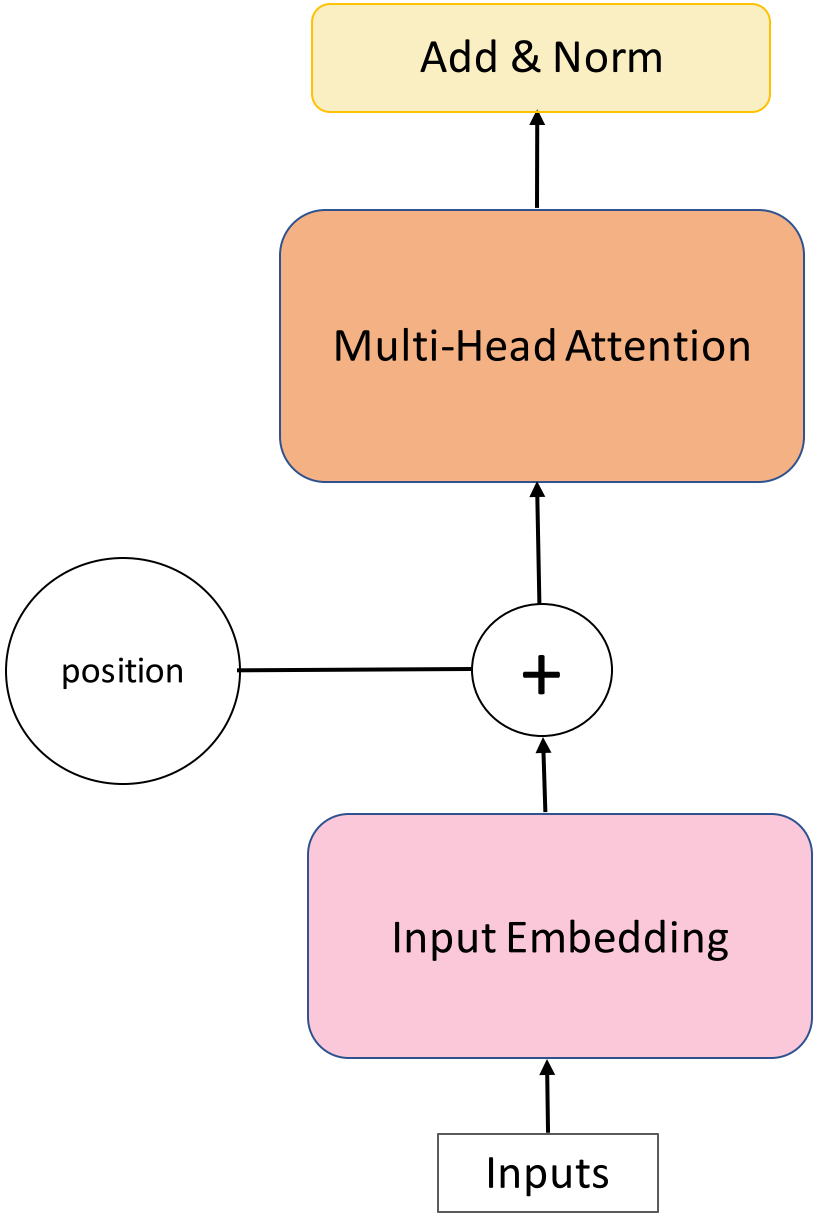
`scores = tf.matmul(query, key, transpose_b=True)`

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)


Output
Probabilities
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)

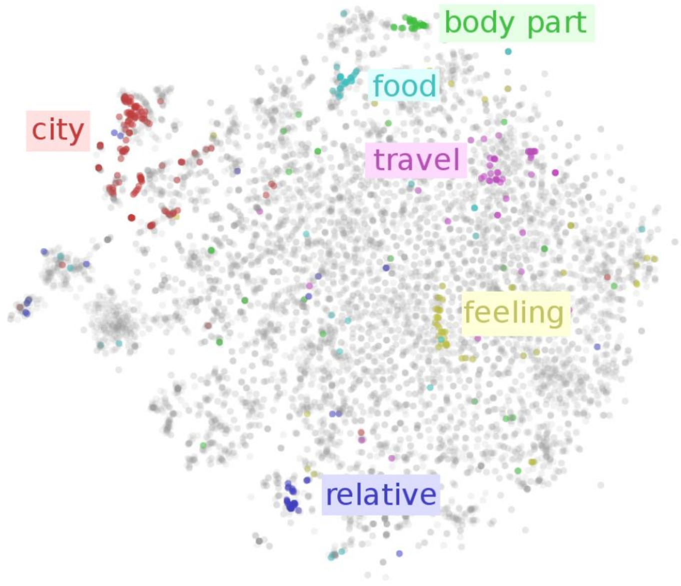

Add & Norm
Multi-Head Attention
position
+
Input Embedding
Inputs

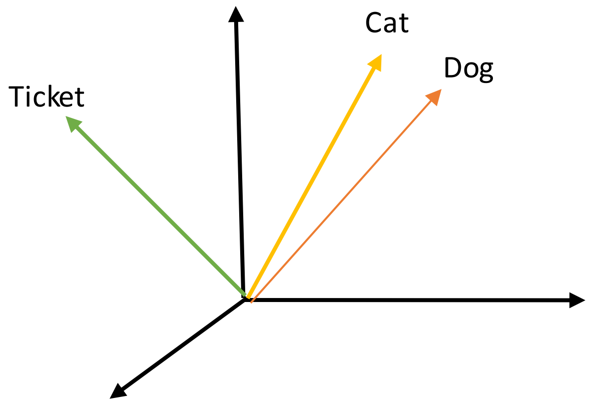
Embedding layer : word vectors

- Each word is indicated by a numerical vector to map "Semantic meanings" to "Geometric space"

- In the mapped geometric space the distance of vectors can be used as an indicator for their semantic distances, e.g.

$D(V_{cat}, V_{dog}) < D(V_{cat}, V_{ticket})$





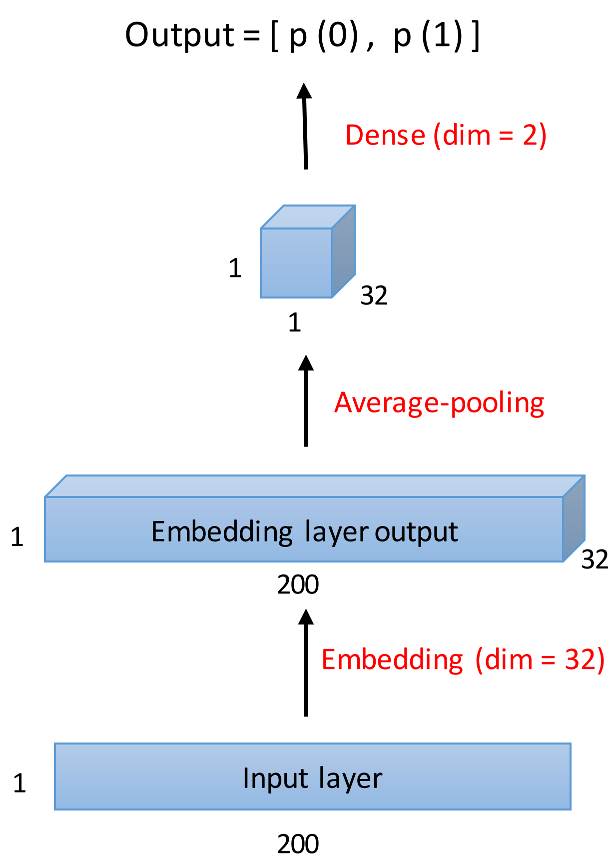
Embedding layer example

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers


vocab_size = 20000  # Only consider the top 20k words
maxlen = 200  # Only consider the first 200 words of each sequence
(x_train, y_train), (x_val, y_val) = keras.datasets.imdb.load_data(num_words=vocab_size)
print(len(x_train), "Training sequences")
print(len(x_val), "Validation sequences")
x_train = keras.preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_val = keras.preprocessing.sequence.pad_sequences(x_val, maxlen=maxlen)
```

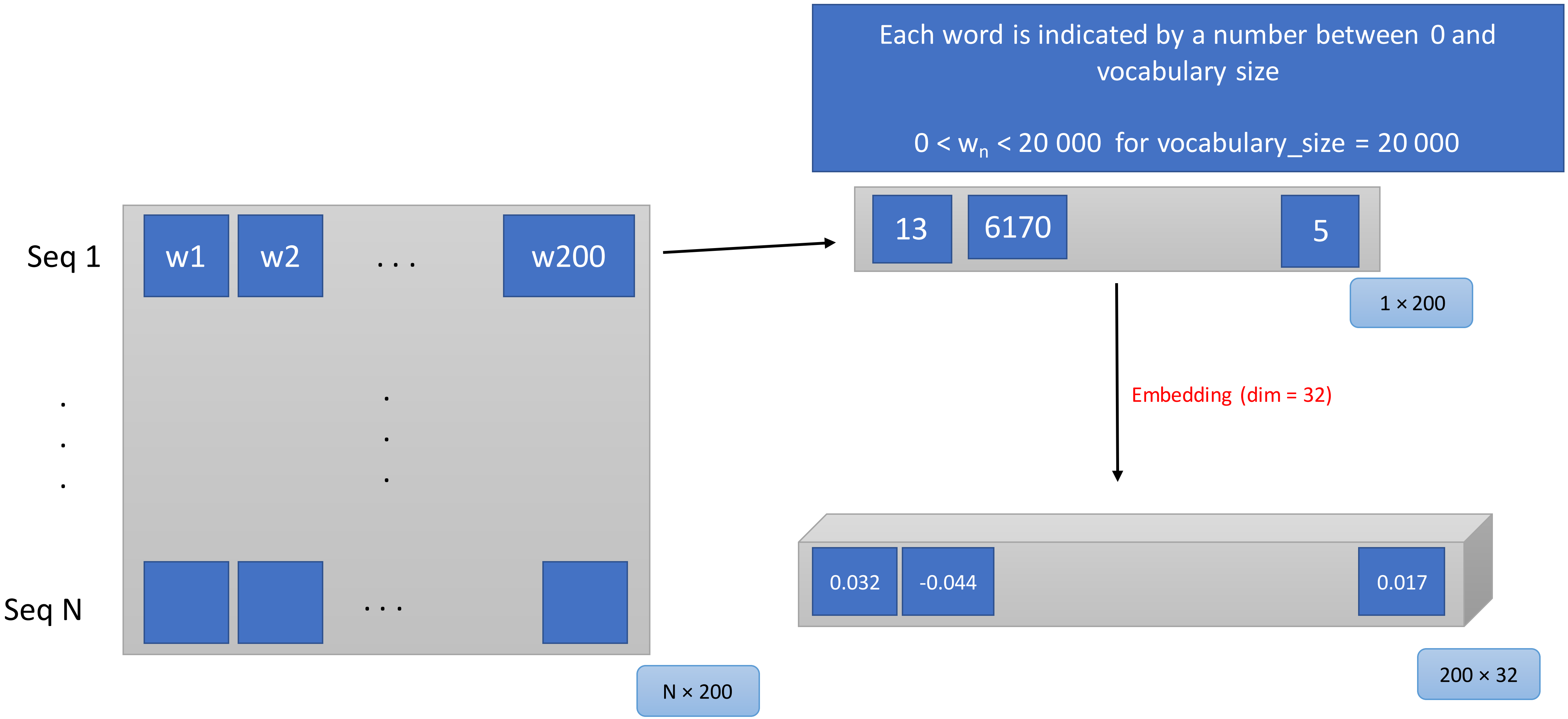
```
25000 Training sequences
25000 Validation sequences
```

```
############################################################################### Training
embed_dim = 32

inputs = layers.Input(shape=(maxlen,))

embedding_layer = layers.Embedding(input_dim=vocab_size, output_dim=embed_dim)
x = embedding_layer(inputs)

pool_layer = layers.GlobalAveragePooling1D( name = 'pool')
x = pool_layer(x)

outputs = layers.Dense(2, activation="softmax")(x)

model = keras.Model(inputs=inputs, outputs=outputs)

print(model.summary())

model.compile("adam", "sparse_categorical_crossentropy", metrics=["accuracy"])
history = model.fit(x_train, y_train, batch_size=32, epochs=5, validation_data=(x_val, y_val))
```

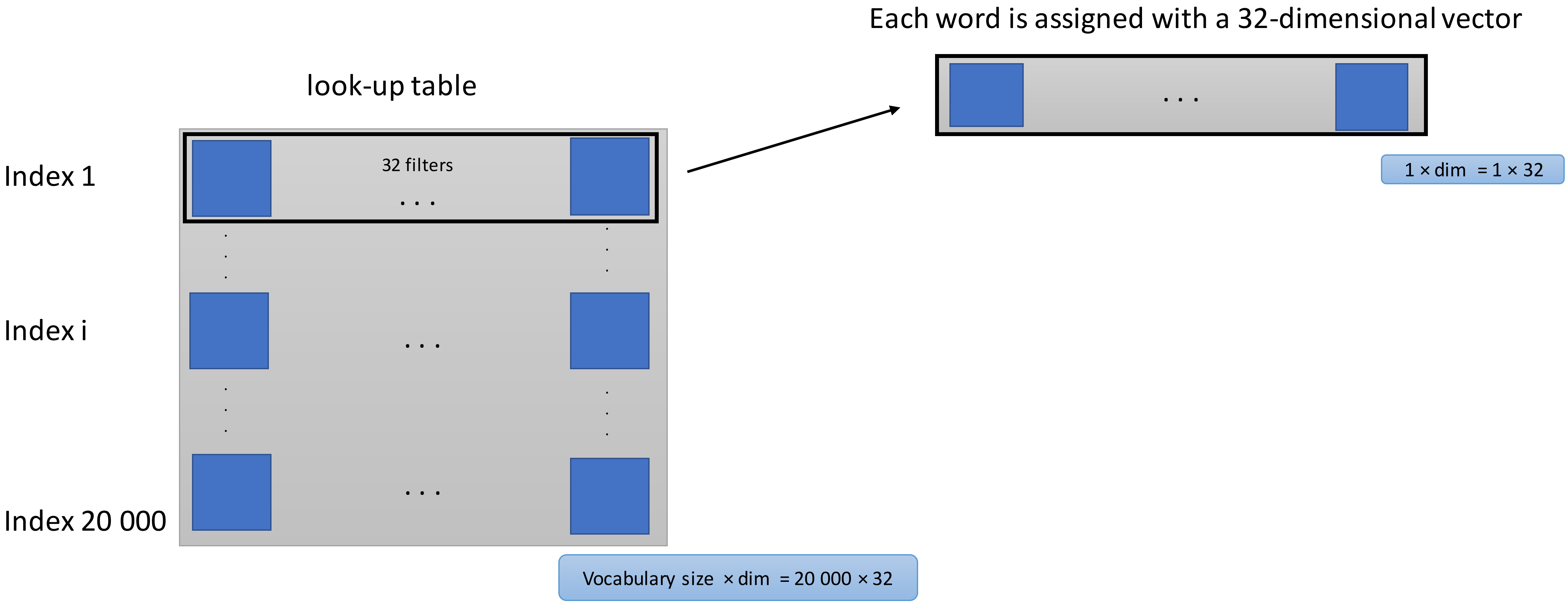
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 200)]             0
_________________________________________________________________
embedding (Embedding)        (None, 200, 32)           640000
_________________________________________________________________
pool (GlobalAveragePooling1D (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 2)                 66
=================================================================
Total params: 640,066
Trainable params: 640,066
Non-trainable params: 0
```



Embedding layer example



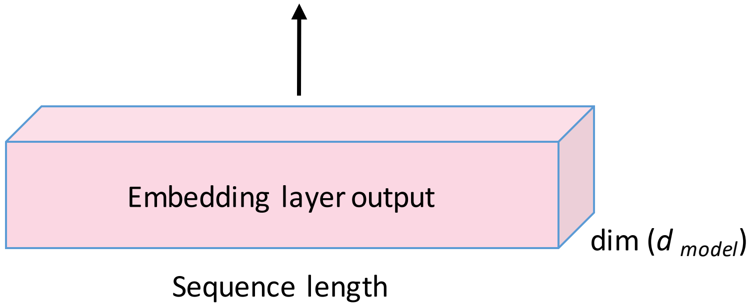
Embedding layer example

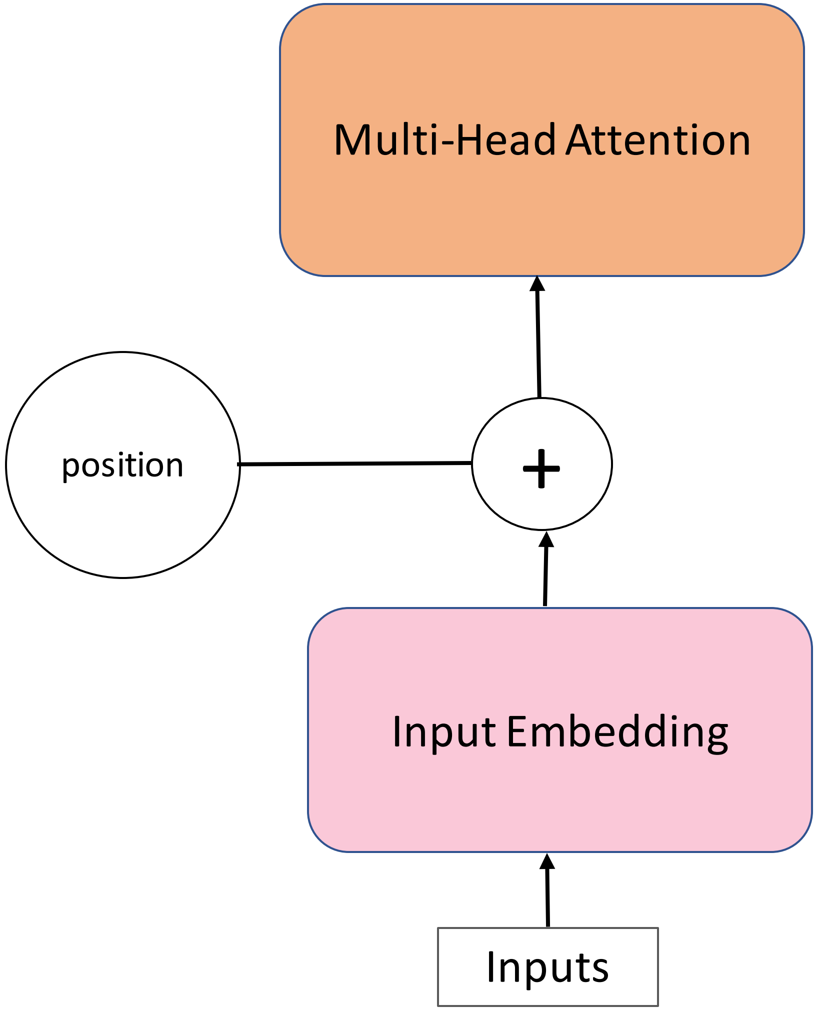


positional encoding

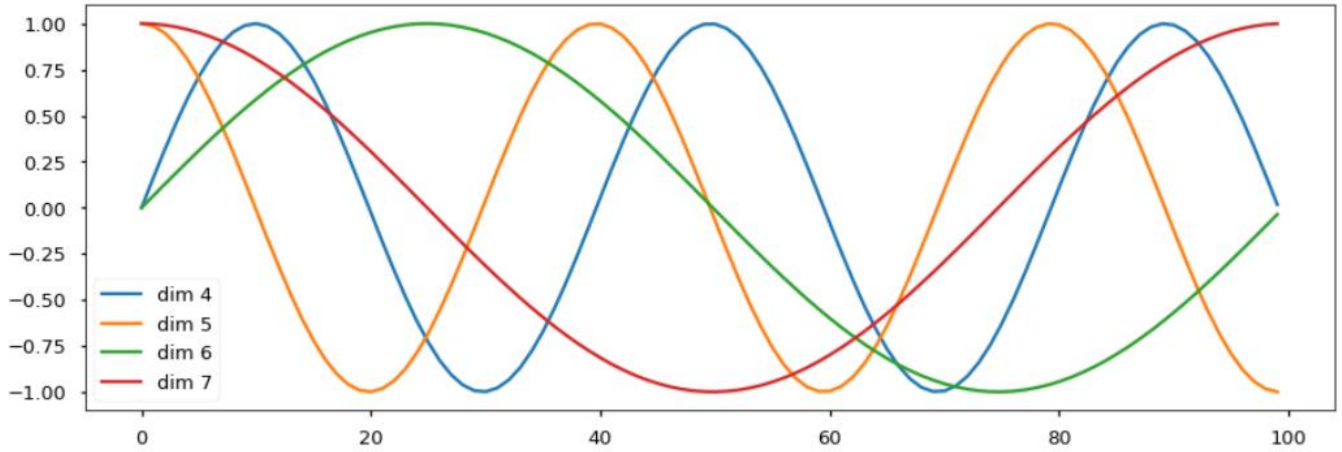
Each filter "i" defines a separate positional component.

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



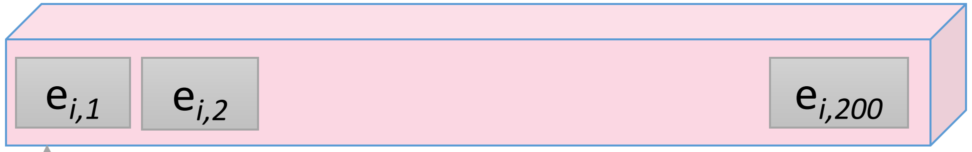


positional encoding



source: http://nlp.seas.harvard.edu/2018/04/03/attention.html

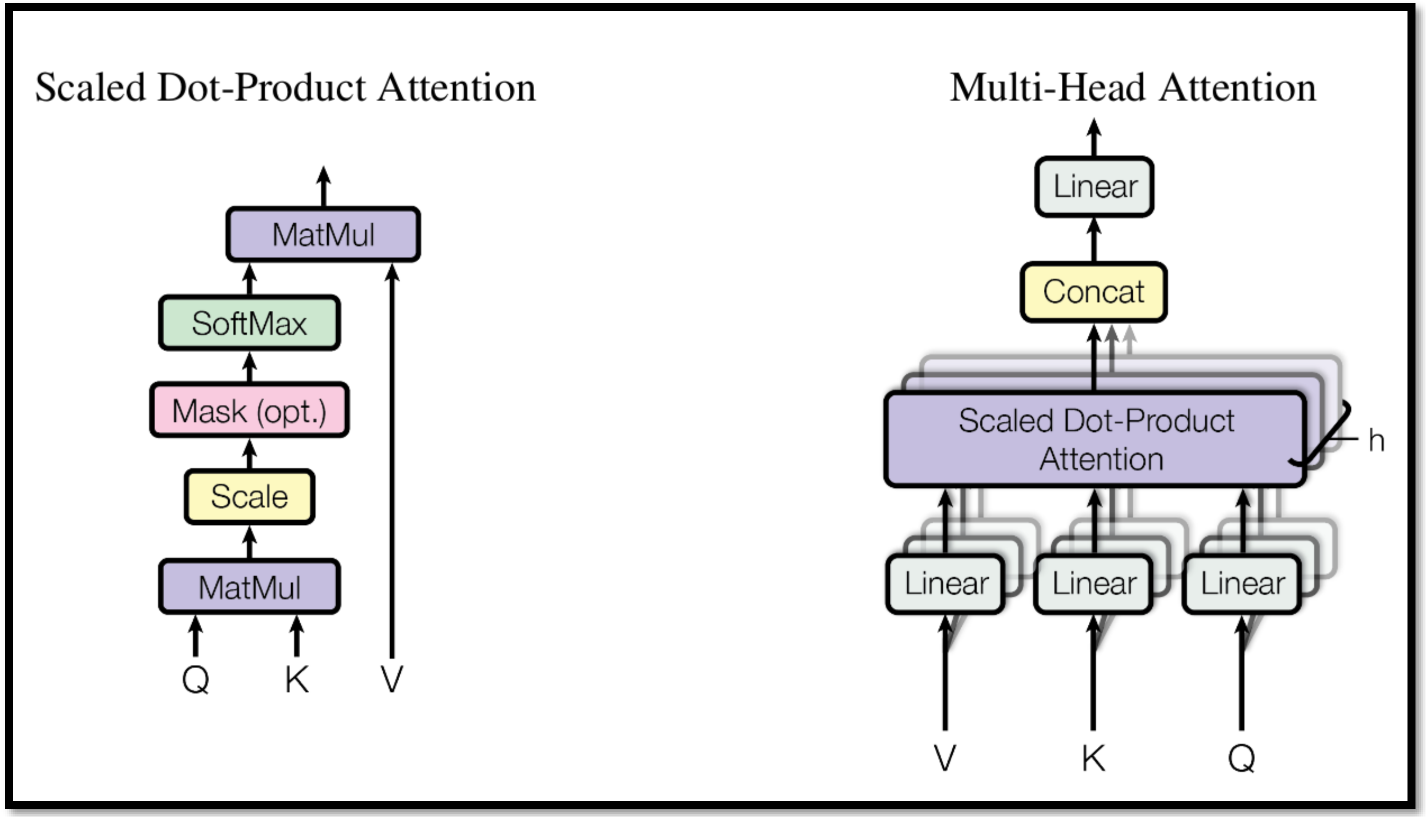
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$



dim (d_{model}) = 32

Embedding layer output

Dot product multi-head attention layer



$d_{(model)}$

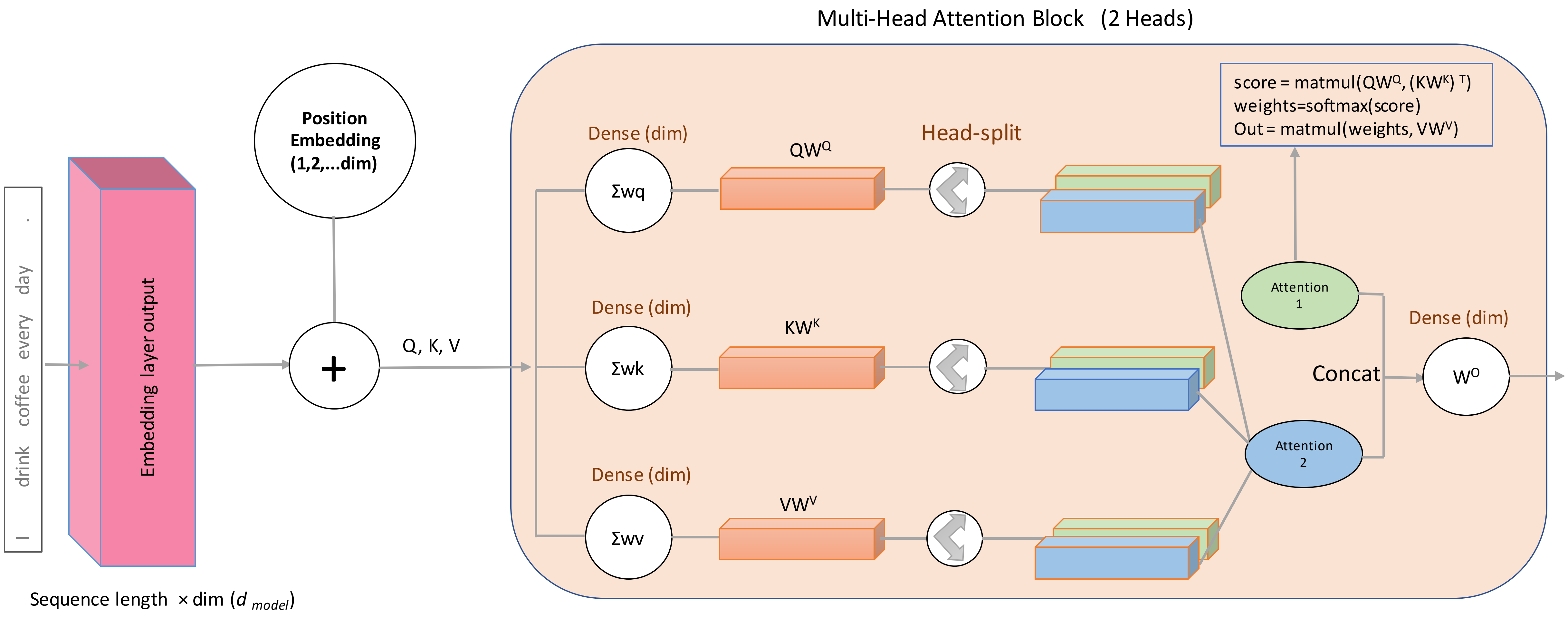
8 parallel channels each of size = $d_{(model)}/8$

- 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

$d_{(model)}$

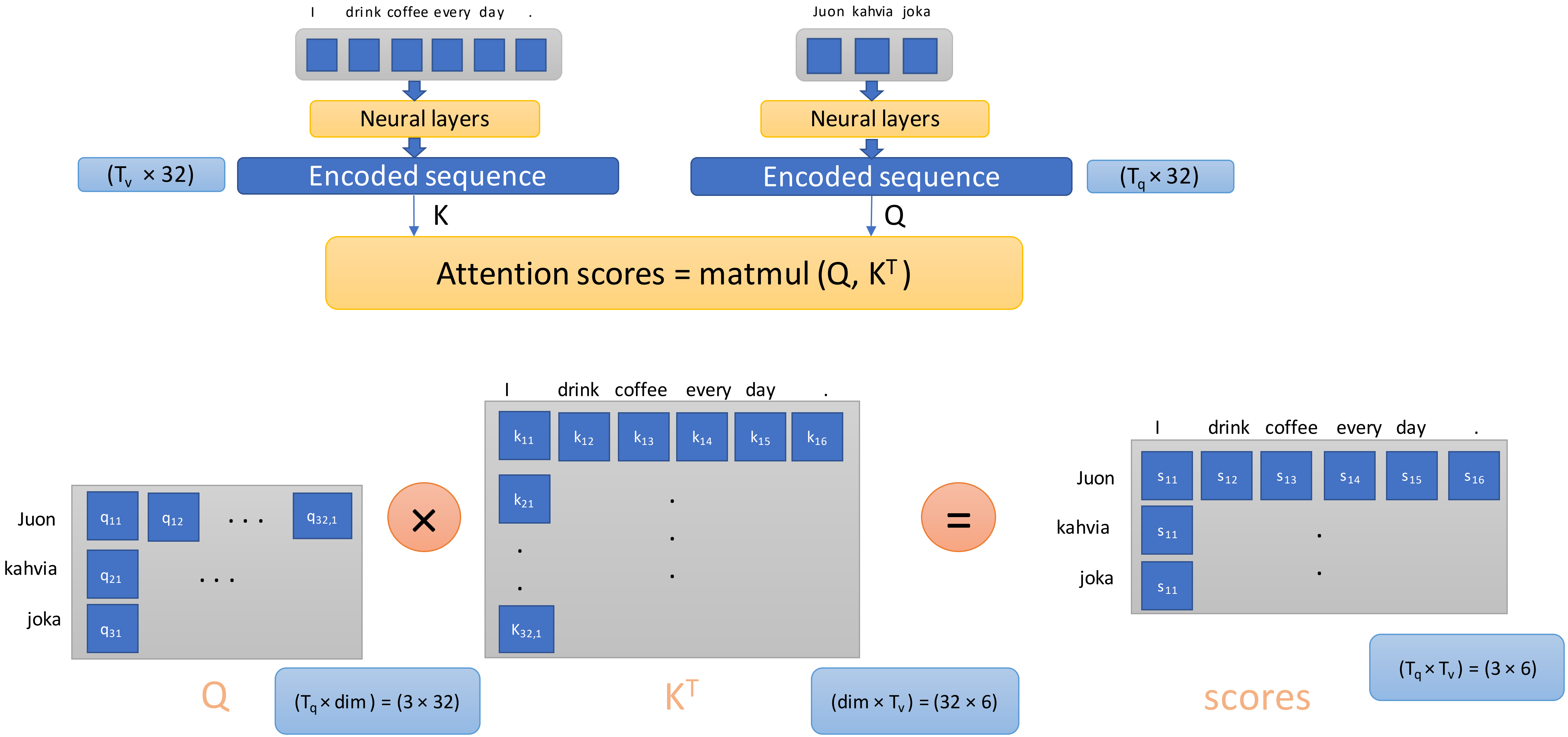
$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Multi-Head attention

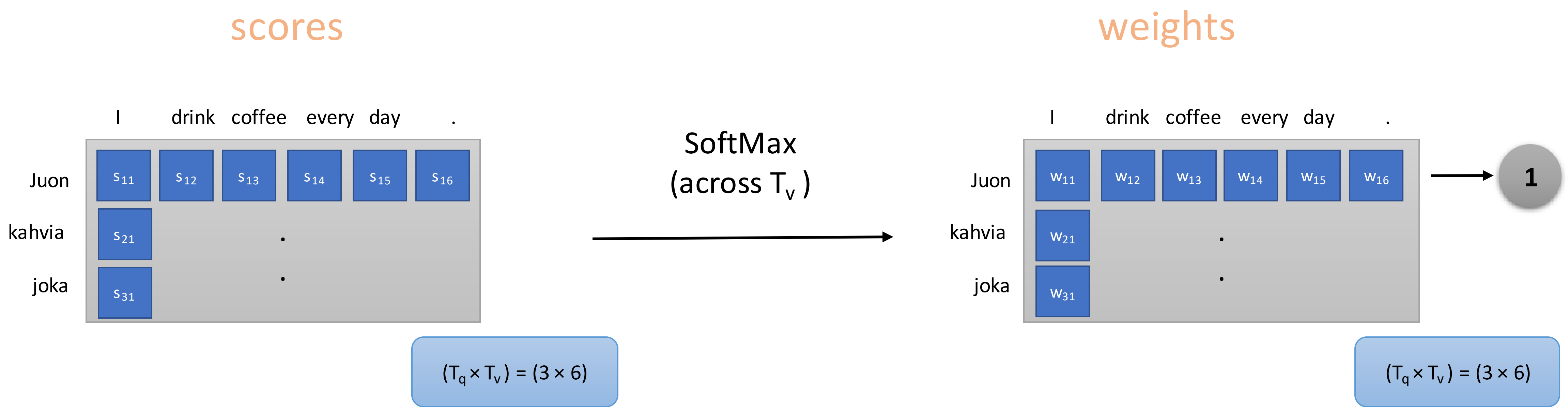


$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

One-Head vs Multi-head



One-Head vs Multi-head



One-Head

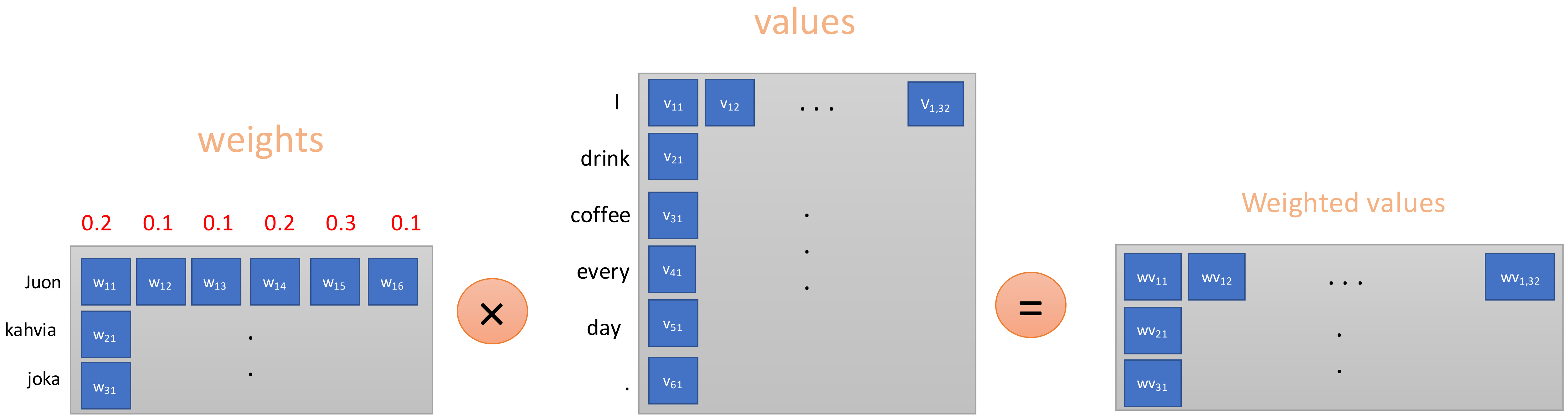
$W_{11} = \mathrm{SoftMax}(q_{1,1}k_{1,1} + \ldots + q_{1,32}k_{1,32})$

Multi-Heads

$W°_{1,1}\,(h1) = \mathrm{SoftMax}(q_{11}k_{11} + \ldots + q_{1,16}k_{1,16})$

$W°_{1,1}\,(h2) = \mathrm{SoftMax}(q_{1,17}k_{1,17} + \ldots + q_{1,32}k_{1,32})$

One-Head vs Multi-head



One-Head $W_{11} = \mathrm{SoftMax}(q_{1,1}k_{1,1} + \ldots + q_{1,32}k_{1,32}) \times v_{11}$

Multi-Heads $W°_{11}\,(h1) = \mathrm{SoftMax}(q_{1,1}k_{1,1} + \ldots + q_{1,16}k_{1,16}) \times v_{11}$ $W°_{11}\,(h2) = \mathrm{SoftMax}(q_{1,17}k_{1,17} + \ldots + q_{1,32}k_{1,32}) \times v_{11}$

Thank you!