Department of Computing

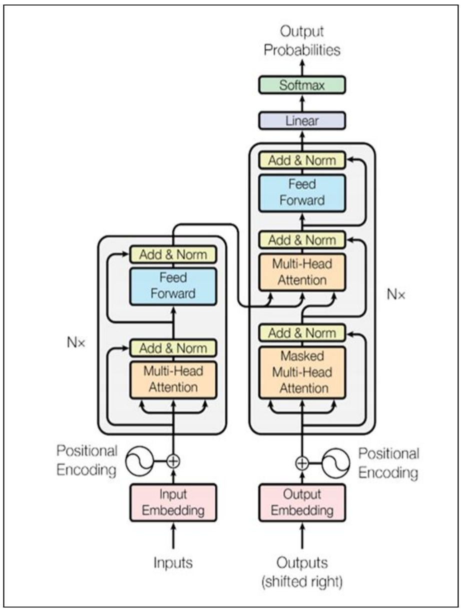
Imperial College of Science, Technology and Medicine

Transformers

Author: Anton Zhitomirsky

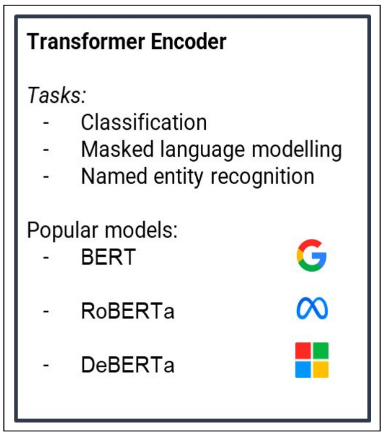
Contents

1 Transformers

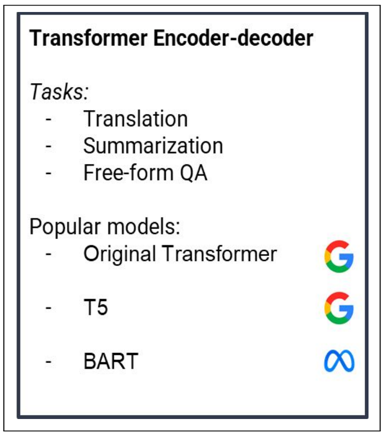


- There are n amounts of encoders and n amounts of decoders.
- Within one layer (encoder or decoders) we have multiple sub-layers
- In the encoder, we have 2 sub-layers, which processes muilti-head attention and the add and norm, the second sub-layer has a feed forward network and add & norm.
- In the decoder we have 3 sublayers, which is the masked multi-head attention, sublayer 2 (which is also known as cross attention) and sublayer three which is a feed-forward network.

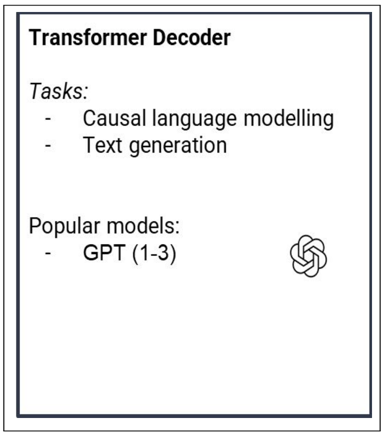
The original paper utilizes this as an encoder decoder network.



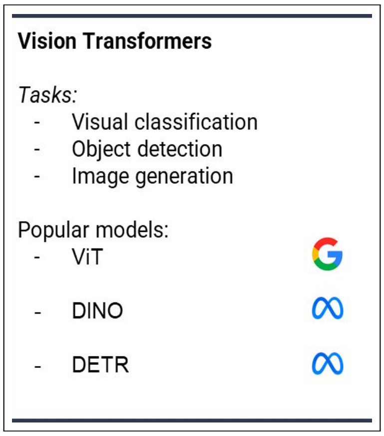
(a) Classificaiton tasks, feeding in a sequence and get a prediction over full input (sentiment classification) or a subset (entity recognition).



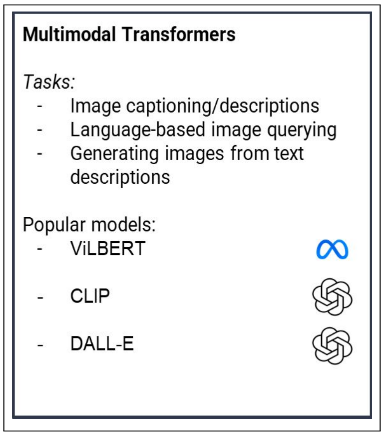
(b) Sequence to sequence tasks



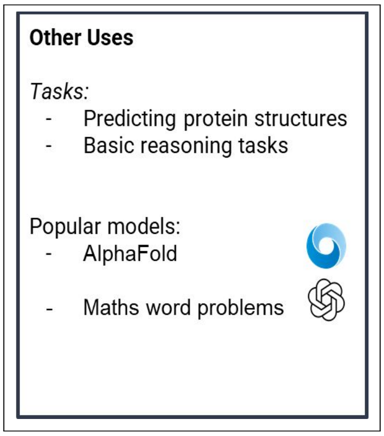
(c) For language specific task such as text generation or language modelling



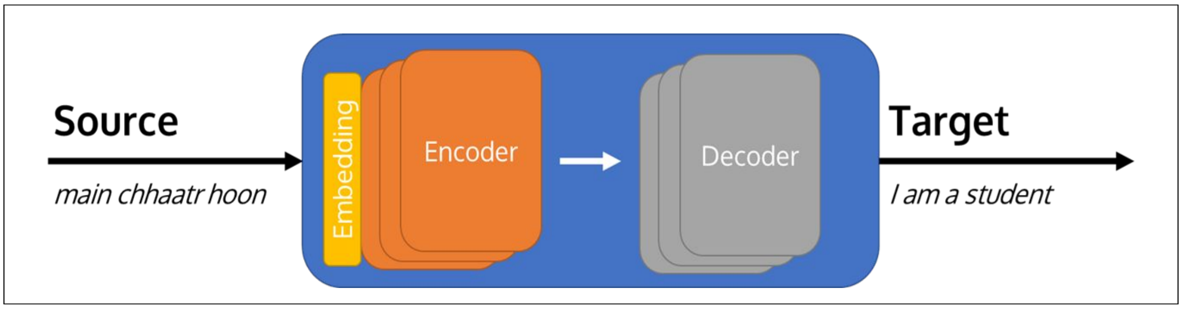
(d) Use in place of CNNs, object detection or generation



(e) Incorparating more than one modality (e.g. text, image, sound). So, generating images from text, or image captioning from an image.

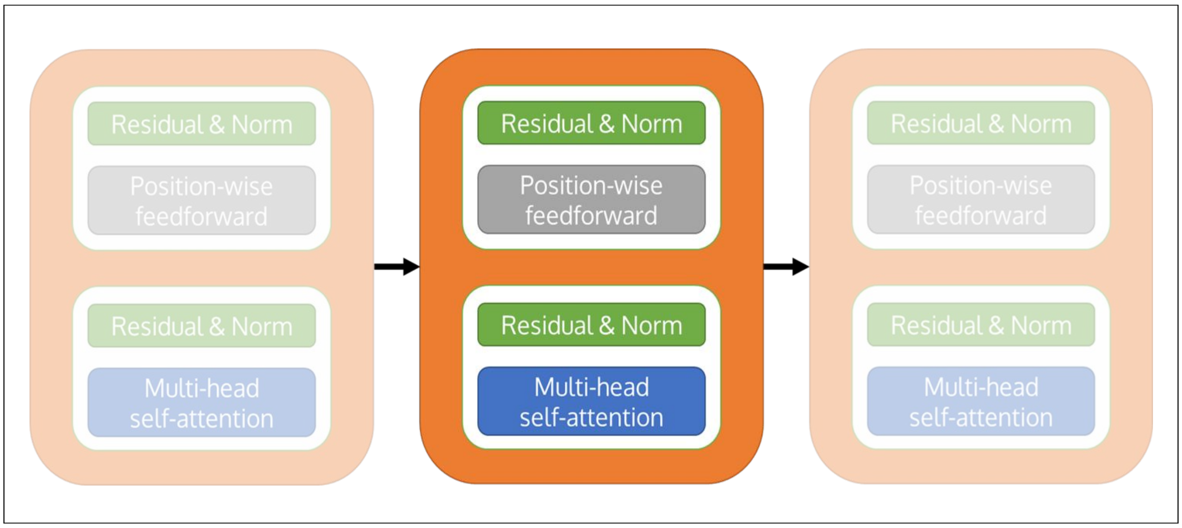


(f)

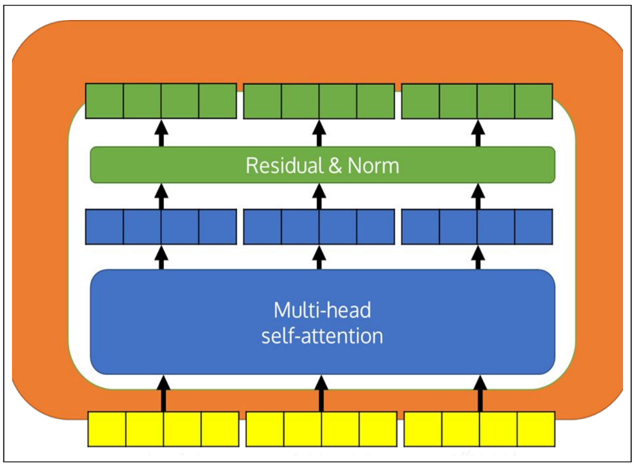


Like with other translation, the Transformer takes in a source sentence and generates a target. Inside, the trasnformer, we have a stack of encoder layers and a stack of decoder layers. At some point, we use the encoded output to help us generate the target. Note that the output of one encoder layer is sent as input to the next encoder layer. Similarly, the output of one decoder layer is sent to the next decoder layer as input

1.1 Encoder



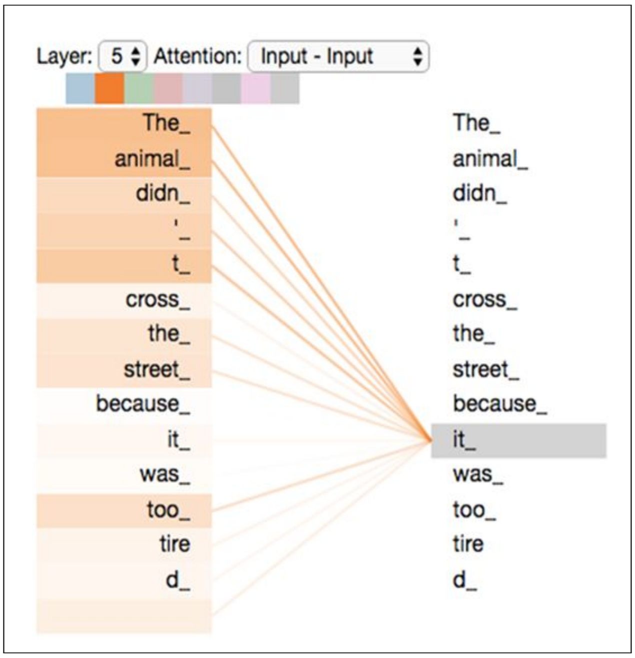
The output of the first encoder serves as the input for the second encoder. There are two sublayers. The first sublayer contains a multi-head self-attention module, and the second contains a position-wise feed forward network. A common theme in Transformers is that each sublayer will have a residual connection and layer normalization



- The multi-head self-attention receives some input, the yellow, which are some word representations (here, with 4 dimensions each). Generally, we have S words, each with D dimensions.
- The multi-head self-attention layer processes the input and outputs another set of $S \times D$ encodings.
- These ecnodings get sent to the Residual & Norm, which outputs another set of $S \times D$ encodings
- Conceptually this should make transformers easy to work with -mostly everything in the encoder stays as $S \times D$.

1.1.1 Self-attention

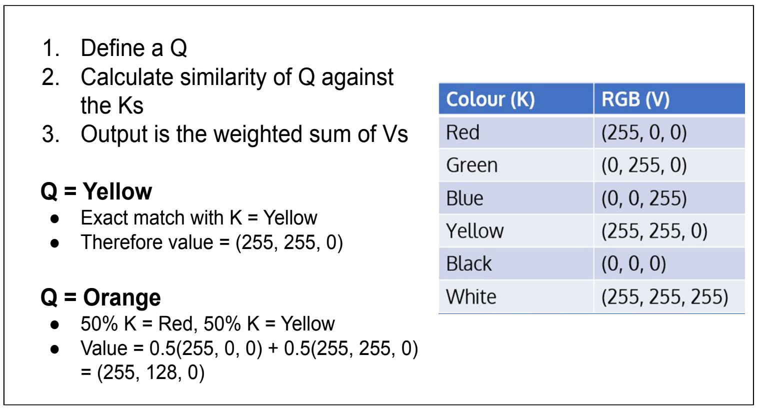
The self-attention mechanism allows each input in a sequence to look at the whole sequnce to compute a representation of the sequence. Each word therefore gets a representation based on all the other words in the input sequence. Each S in an $S \times D$ input has looked at every other word to compute its D-dimensional representation.



- As an example: “the animal didn’t cross the street because it was too tired’
- If we were processing this sequence, we would have some hidden state representation for the word “it”.
- In an RNN, the word “it” hasn’t been explicitly forced to look at other words in the sequence to align itself within them; the importance of words would be equal where we don’t want them to be
- Transformers are designed such that the hidden state representation of the word “it” would be influenced more by “animal” than “because”.
- For self-attention, each input (e.g. “it”) looks at the whole sequence and then computes a representation of that word, based on how important each of the other words are to it.s

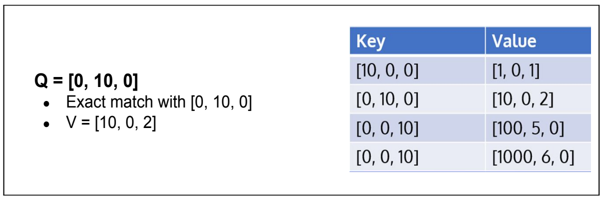
The attention function consists of 3 variables

$$\text{Attention}(Q, K, V) = \sigma(\frac{QK^T}{\sqrt{d_h}})V, \quad \textbf{Q}\text{eries}, \textbf{K}\text{eys}, \textbf{V}\text{alues}, \quad \sigma : \text{softmax} \tag{1.1.1}$$

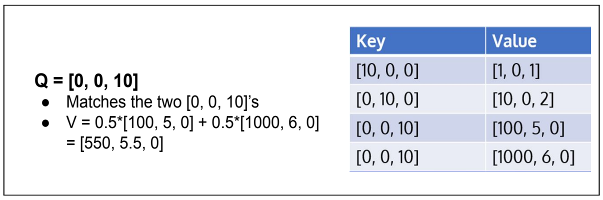


Think of it as a look-up in a dictionary (color to RGB value)

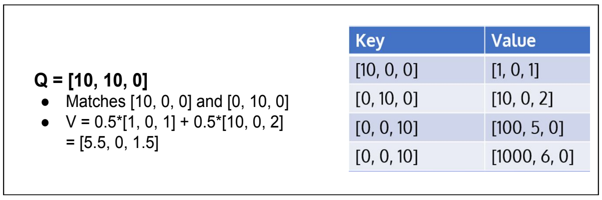
1. The query is the colour we want to find.
2. We would index the dictionary and receive a direct match
3. If the query is “orange” we want 50% red and 50% yellow and combine them



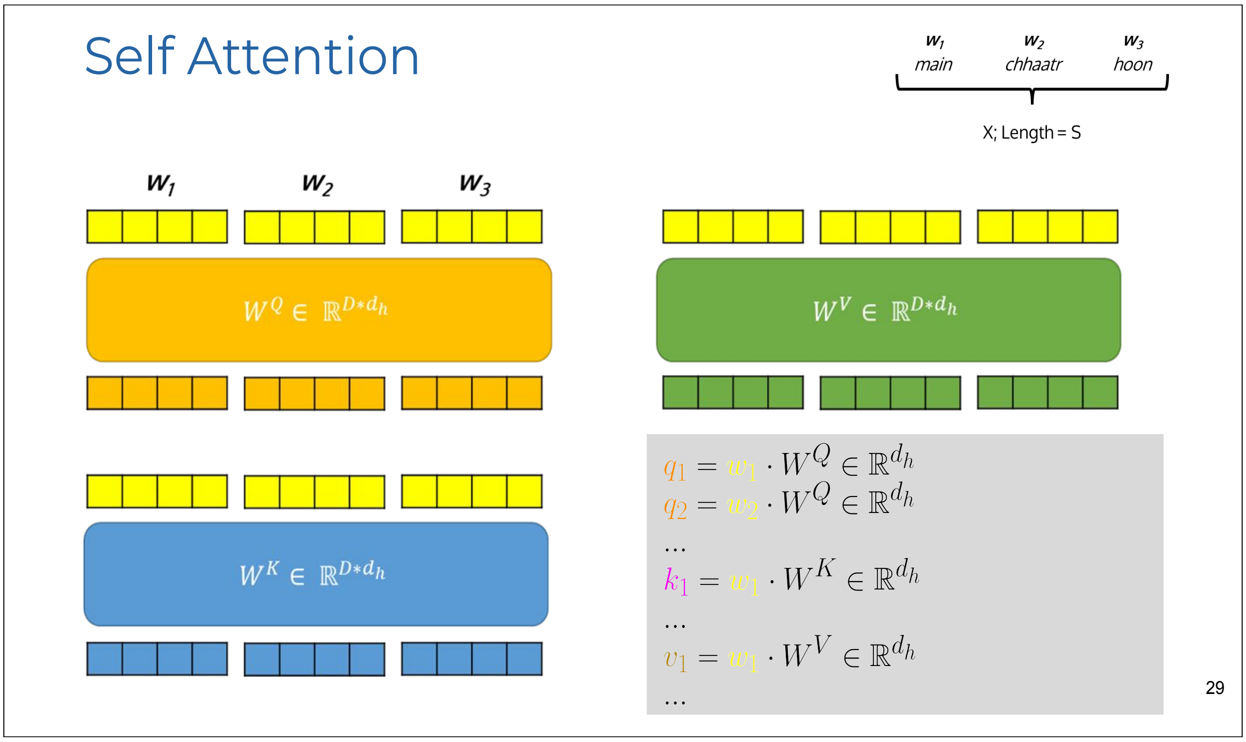
(g) direct match



(h) matches two keys



(i) matches two keys, but in a combination



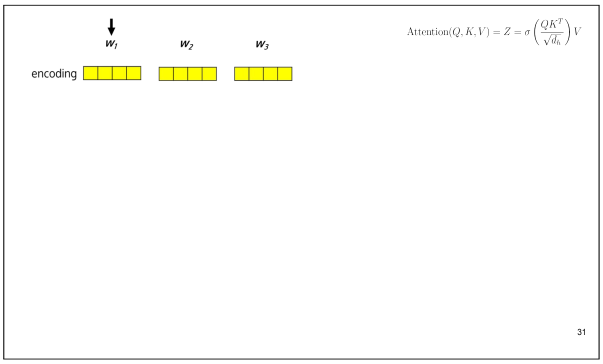
First, we take our word embeddings and project them through a learnt weight matrix, to a representation $D \times d_h$. Firstly, we set d_h equal to D. We do the same for our keys and values.

We can simplify the above to do everything in one-sho in matrix form:

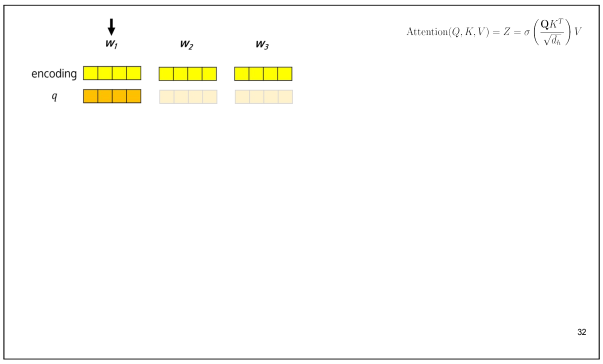
$$Q = W \cdot W^Q \in \mathbb{R}^{S \times d_h} \tag{1.1.2}$$

$$K = W \cdot W^K \in \mathbb{R}^{S \times d_h} \tag{1.1.3}$$

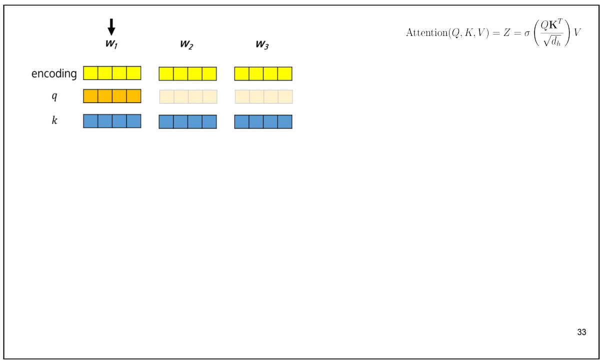
$$V = W \cdot W^V \in \mathbb{R}^{S \times d_h} \tag{1.1.4}$$



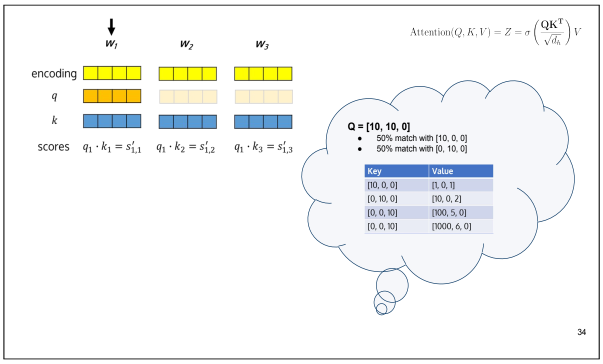
(j) Looking at attention in vector form first, we're going to work out our attention-ized representation for each word. Let's start with w_1



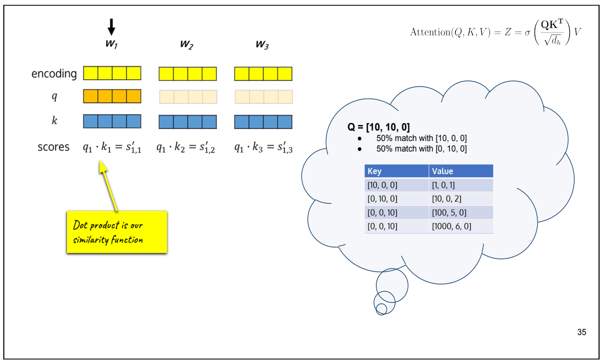
(k) Due to the projections above, we have query vectors for all words. Right now, we only need the query vector for w_1



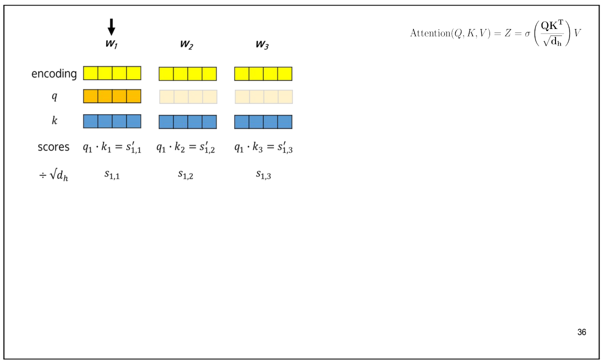
(l) Thinking back to the intuition, we want to compare how similar the query is to all keys we have. This is done by some similarity function. So here, we want to get a representation for w1. We will get this representation by comparing our query vector to all our keys.



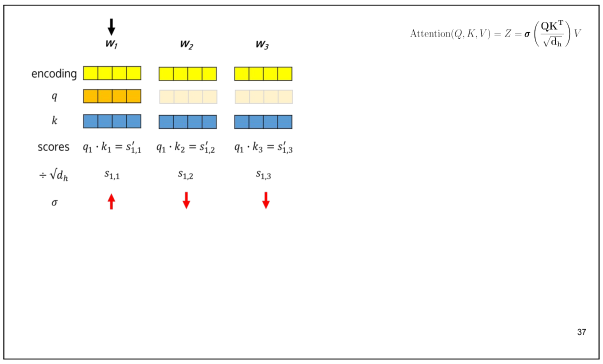
(m) Now we're going to work out the similarity between the query vector we have, and ALL the key vectors. We then obtained a weighting for how similar each of the key vectors were to the query vector. The score here is NOT the weighting, whoeve,r we need it in our calculation of the weighting



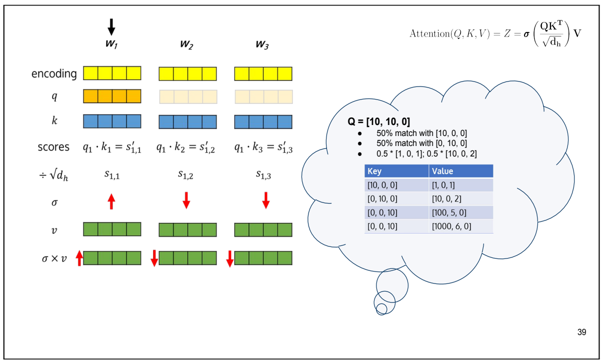
(n) Lets say the similarity is the dot product, so we get an un-normalized similarity between the given query vector and key vectors.



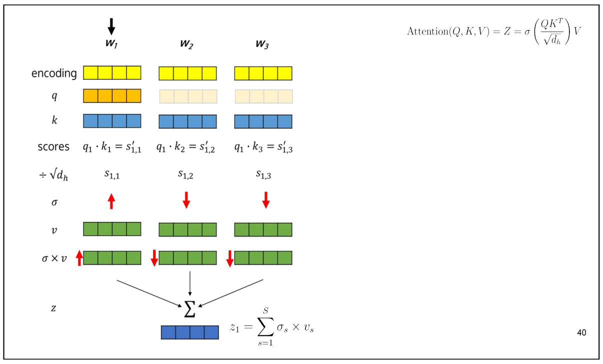
(o) In the next step, apply softmax, but before we do this, divide by $\sqrt{d_h}$. Post-division values are now closer to 0. This makes the softmax operation less peaky. That means that the outputs of the softmax operation will ideally not have an individual value which dominates the weightings



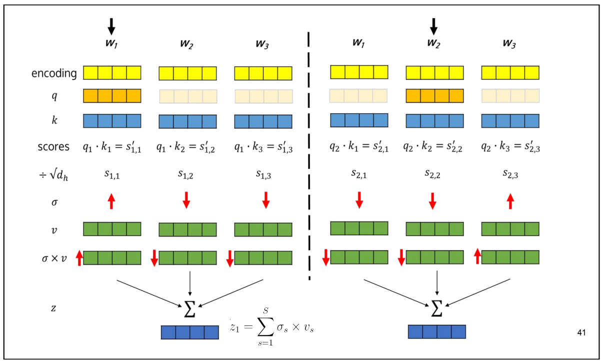
(p) Arrows indicate strong and weak weightings.



(q) The first step is simply retrieving the value vectors we obtained previously. The second step is actually performing the weighting on each of our value vectors. So our value vectors retain more information if the softmax value for that index is high, and retain less information if the softmax value for that index is low



(r) Finally we obtain our contextual representation for the first word. This is the sum over our weighted value vectors.



(s) We repeat the process for every word in our input sequence. Our overall representation for our input sequence would then be all of our z vectors stsacked togheter. Each z-vector is D-dimensional, and obviously we have S amounts of them.