The most straightforward way of applying self-attention is where the attention matrix is directly formed by the dot product of the input vectors. However, this is problematic. The dot product between a vector and itself will always be high, so each hidden state will be biased towards attending to itself. The transformer solves this by first projecting the input into three different representations using three different weight matrices:

- The **query vector** $\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i$ is the one being *attended from*, like the target in the standard attention mechanism. Query vector
- The **key vector** $\mathbf{k}_i = \mathbf{W}_k \mathbf{x}_i$ is the one being *attended to*, like the source in the basic attention mechanism. Key vector
- The **value vector** $\mathbf{v}_i = \mathbf{W}_v \mathbf{x}_i$ is the context that is being generated. Value vector

In the standard attention mechanism, the key and value networks are identical, but intuitively it makes sense for these to be separate representations. The encoding results of the ith word, $\mathbf{c}_i$, can be calculated by applying an attention mechanism to the projected vectors:

$$\begin{aligned} r_{ij} &= (\mathbf{q}_i \cdot \mathbf{k}_j)/\sqrt{d} \\ a_{ij} &= e^{r_{ij}}/(\sum_k e^{r_{ik}}) \\ \mathbf{c}_i &= \sum_j a_{ij} \cdot \mathbf{v}_j \,, \end{aligned}$$

where d is the dimension of $\mathbf{k}$ and $\mathbf{q}$. Note that i and j are indexes in the same sentence, since we are encoding the context using self-attention. In each transformer layer, self-attention uses the hidden vectors from the previous layer, which initially is the embedding layer.

There are several details worth mentioning here. First of all, the self-attention mechanism is *asymmetric*, as r_{ij} is different from r_{ji}. Second, the scale factor $\sqrt{d}$ was added to improve numerical stability. Third, the encoding for all words in a sentence can be calculated simultaneously, as the above equations can be expressed using matrix operations that can be computed efficiently in parallel on modern specialized hardware.

The choice of which context to use is completely learned from training examples, not prespecified. The context-based summarization, $\mathbf{c}_i$, is a sum over all previous positions in the sentence. In theory, any information from the sentence could appear in $\mathbf{c}_i$, but in practice, sometimes important information gets lost, because it is essentially averaged out over the whole sentence. One way to address that is called **multiheaded attention**. We divide the sentence up into m equal pieces and apply the attention model to each of the m pieces. Each piece has its own set of weights. Then the results are concatenated together to form $\mathbf{c}_i$. By concatenating rather than summing, we make it easier for an important subpiece to stand out. Multiheaded attention

24.4.2 From self-attention to transformer

Self-attention is only one component of the transformer model. Each transformer layer consists of several sub-layers. At each transformer layer, self-attention is applied first. The output of the attention module is fed through feedforward layers, where the same feedforward weight matrices are applied independently at each position. A nonlinear activation function, typically ReLU, is applied after the first feedforward layer. In order to address the potential vanishing gradient problem, two residual connections are added into the transformer layer. A single-layer transformer in shown in Figure 24.9. In practice, transformer models usually