2.2 Attention

Let's discuss attention in terms of machine translation. Some problems with the standard encoder-decoder architecture are that long-term dependencies between words that are hard to learn, and that the hidden state attempts to store information of any length into a hidden vector of fixed size. In fact, when generating the next translated word in a sentence, we do not necessarily need to consider the entire input. We can instead choose to only attend to relevant words in an input.

In a vanilla sequence to sequence model, we generate a next word $y^{\langle t \rangle}$ using the hidden state $s^{\langle t \rangle}$, which is a function of $s^{\langle t-1 \rangle}$ and $y^{\langle t-1 \rangle}$. With attention, however, we consider $s^{\langle t \rangle}$ as a function of three things: $s^{\langle t-1 \rangle}$, $y^{\langle t-1 \rangle}$, and some *context* $c^{\langle t \rangle}$. The context vector is a weighted sum of all input activations $a^{\langle t' \rangle}$:

$$c^{\langle t \rangle} = \sum_{t'} \alpha^{\langle t,t' \rangle} a^{\langle t' \rangle} \tag{20}$$

where $\alpha^{\langle t,t' \rangle}$ is the amount of attention that $y^{\langle t \rangle}$ should pay to $a^{\langle t' \rangle}$. Omitting the softmax for simplicity, one way to compute attention is to train a small neural network to compute $\alpha^{\langle t,t' \rangle}$ given $s^{\langle t-1 \rangle}$ and $a^{\langle t' \rangle}$.

Another way of thinking of attention is that given a *query* $\boldsymbol{q}$ and a set of *key, value* pairs ($\boldsymbol{k}, \boldsymbol{v}$ pairs), we compute a context as a weighted sum of the values, where weights are computed using compatibility function between the query and each key. In the scenario described above, the query is the last decoder state $s^{\langle t-1 \rangle}$, and the keys and values are the same and equal the encoder states (activations) $a^{\langle t' \rangle}$. As for computing the compatibility function between $\boldsymbol{q}$ and $\boldsymbol{k}$, there are several ways:

- **Dot product.** $\text{attn\_score}(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^T \boldsymbol{k}$. This method forces input and output encodings to be in the same space.
- **Scaled dot product.** Scale of dot product increases with larger dimensions, so $\text{attn\_score}(\boldsymbol{q}, \boldsymbol{k}) = \frac{\boldsymbol{q}^T \boldsymbol{k}}{\sqrt{|\boldsymbol{k}|}}$
- **Bilinear function.** Relaxing the requirement for input and output encodings to be in the same space, we have $\text{attn\_score}(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^T W_a \boldsymbol{k}$.
- **Multi-layer perceptron (additive similarity).** $\text{attn\_score}(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{w}_2^T \tanh\left(W_1[\boldsymbol{q}; \boldsymbol{k}]\right)$.