Attention

- Without attention, a model has to look at the entire input sentence, and then generate the output one by one.

- With attention, a model can look at different parts of the sentence, and output based on each part.

**Seq2Seq models:**

![Diagram of Seq2Seq model]

**In more details:**

![Detailed diagram of Seq2Seq model]

This means that the encoder processes all the input first, then computes a representation to be fed to the decoder.
more details about the context vector. The encoder takes the newest hidden state and the n-th input token and generates a new hidden state until all of the input tokens are consumed. Then, the encoder sends the final hidden state to the decoder.

The fact that the encoder only sends a single, fixed-size vector to the decoder—no matter the length of the input—is a limitation of this architecture. If we set the size of this vector too long to accommodate large input sequences, the model would overfit on small inputs. → Attention solves this problem.

In the seq2seq model that uses attention, the encoder sends all the hidden states to the decoder; not just the last one. → Provides flexibility in the context size.

Each hidden state that the encoder computes, is related most to a token, i.e., the n-th hidden state captures the essence of n-th token in the input sequence. Note that all the hidden states capture the essence of everything a little bit, but capture the essence of their corresponding input token the most.
what does the decoder do?

h1, h2, h3 → decoder → output

How does the attention decoder know which parts of the input sequence to focus on at each time step? *Learns!* E.g., it learns that the order of words in an English sentence is different from that of a French sentence.

A more formal look at the Decoder

At each time step, decoder computes a score vector, giving each hidden state a different rank. It then feeds the scores to a softmax function to get them as probabilities. These weights determine how important each hidden state is in the attention vector.
The simplest way to compute the context vector for the decoder: (at every timestep)

\[
\begin{align*}
    h_1 \times s_1 & + \ h_2 \times s_2 & + \ h_3 \times s_3 & = \\
    & \text{attention context vector for decoder}
\end{align*}
\]

- Attention mechanisms
  - additive
  - multiplicative

**ADDITIVE**

\[
E_{ij} = v^T_a \tanh \left( W_a s_{i-1} + U_a h_j \right)
\]

- learned weight matrices
- hidden state from the encoder
- hidden state of decoder at time step \(i-1\)

\[
\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_a} \exp(e_{ik})}
\]

- Simply softmax
attention Context Vector

\[ c_i = \sum_{j=1}^{T_y} \alpha_{i,j} h_j \]  weighted sum

\[ h_1 \]
\[ h_2 \]
\[ h_3 \]

\[ \alpha_{t1} \]
\[ \alpha_{t2} \]
\[ \alpha_{t3} \]

attention Context vector

---

**MULTIPLICATIVE**

decoder hidden state at timestep \( t \)

score \((h_t, \overline{h_s})\) = \[
\begin{cases} 
 h_t^T \overline{h_s} \\
 \overline{h_t}^T W_a \overline{h_s} \\
 v_a^T \tanh(W_a h_t ; \overline{h_s}) \rightarrow \text{Concat}
\end{cases}
\]

weight matrix
e.g., softmax

\[ a_t(s) = \text{align}(h_t, \overline{h_s}) \]

\[ = \frac{\exp(\text{score}(h_t, \overline{h_s}))}{\sum_{s'} \exp(\text{score}(h_t, \overline{h_{s'}}))} \]
\[ \tilde{h}_t = \tanh ( W_c \ [ c_t ; h_t ] ) \]

- attention
- context
- vector

**Walk-through Example**

**Silly example:**

\[
\text{Score}_{\text{dot}} \left( \begin{pmatrix} 3 \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ 6 \end{pmatrix} \right) = 3 \times 2 + 1 \times 6 = 12
\]

The dot product, intuitively, is like a similarity measure. If the angle between two vectors is large, the similarity, i.e., dot product, is smaller.
More realistic example:

$$\text{score}(h_t, \overline{h_s}) = h_t^T \overline{h_s}$$

**ASSUMPTION:** encoder and decoder have the same embedding space.

The second score function: aka (general)

$$\text{score}_w(h_t, \overline{h_s}) = h_t^T W_a \overline{h_s}$$

- weight matrix for a linear transformation
- trained jointly with the model

Use another score function
Where does it appear in the decoder?

We take sum of $\overline{h}_S$ with weights that come from the aligned attn score vector to obtain attention context vector.

Then, we concat the attn context vector and the hidden state of the decoder that has been computed in this time step and pass it through a FC layer which basically performs the $(W_c[c_t; h_t])$ part of the equation. Then we take tanh, and the output is our first output token.
Concat scoring function:

\[ \text{Score} \{ h_t, \bar{h}_s \} = W_a \begin{bmatrix} h_t & \bar{h}_s \end{bmatrix} V_a \]

\[ \approx U_a^T \tanh(W_a [h_t; \bar{h}_s]) \]

This looks similar to ADDITIVE, but it is not.
- The additive has more learned parameters.
- The additive uses \( h_t \) that is from the previous step whereas concat method uses the \( h_t \) from current step.