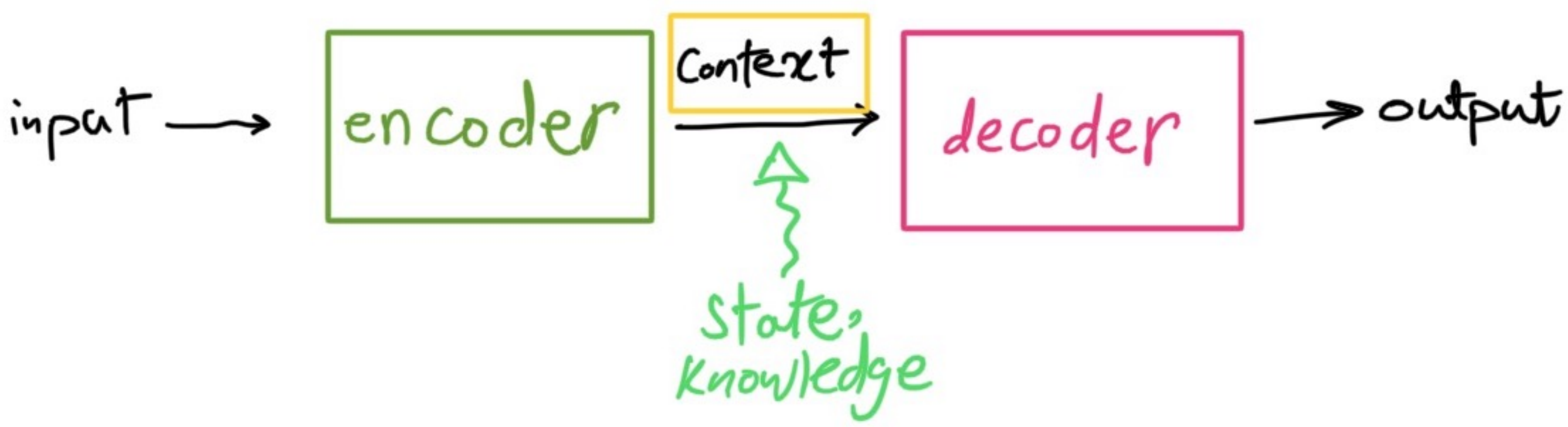
Attention

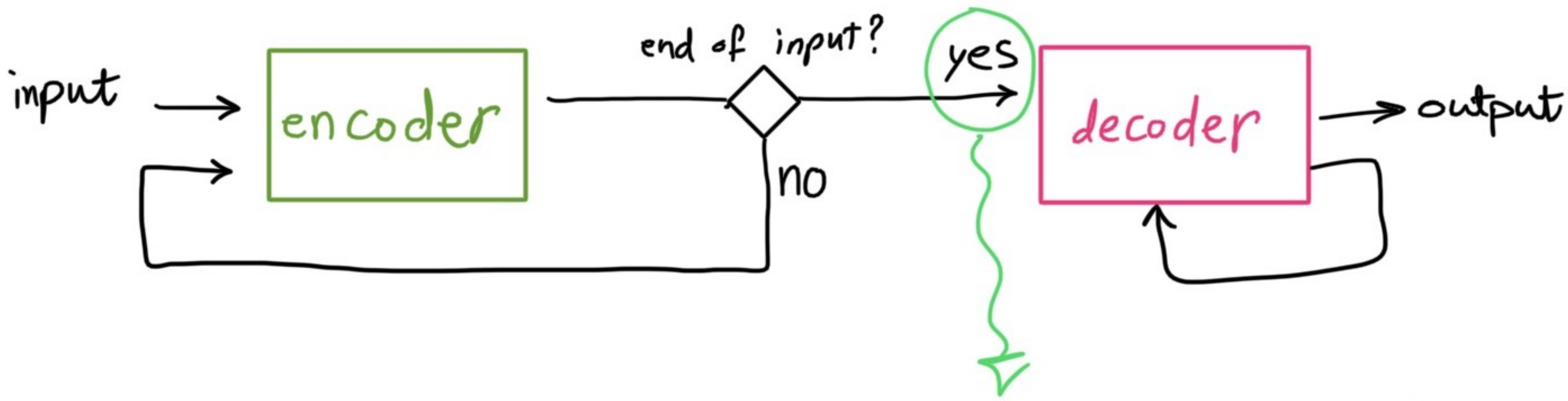
- without attention, a model has to look at the entire input sentence, and then generate the output one by one.

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

seq2seq models:



in more details:



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 send 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 accomodate 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 flexiblity in the Content 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 every thing a little bit; but capture the essence of their corresponding input token the most.

• what does the decoder do?

How does the attention decoder know which parts of the input sequence to focus on at each time step? Learns!

e.g., it learn that the order of words in an English sentence 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)

$$h_1 \times s_1 + h_2 \times s_2 + h_3 \times s_3 =$$

attention context vector for decoder

softmax of scores

- Attention mechanisms
  - additive
  - multiplicative

$\boxed{\text{ADDITIVE}}$

learned weight matrices

$$e_{ij} = v_a^T \tanh\left(W_a s_{i-1} + U_a h_j\right)$$

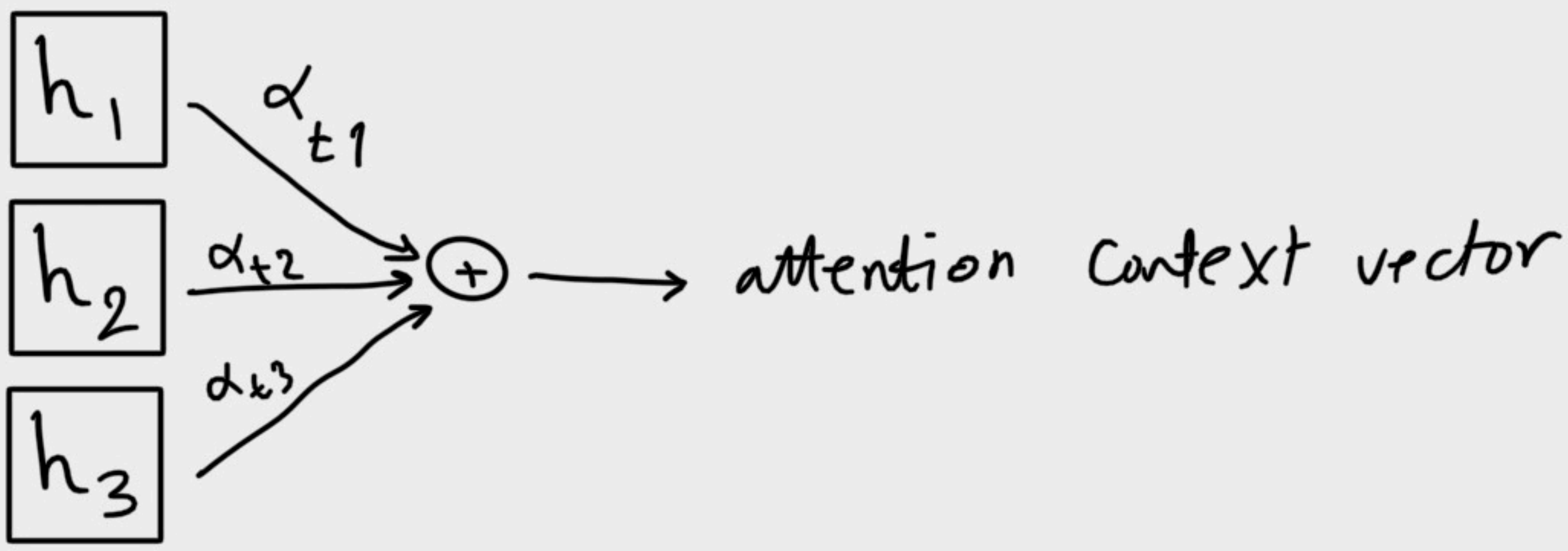
hidden state of decoder at timestep $i-1$

hidden state from the encoder

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

simply softmax

attention context vector ← $c_i = \sum_{j=1}^{T_x} \underbrace{\alpha_{ij}}_{\text{score}} \underbrace{h_j}_{\text{hidden state}}$ weighted sum



MULTIPLICATIVE

decoder hidden state at timestep t

all the encoder hidden states at $\overbrace{\text{time step}}^{\text{previous}}$

$$
\text{score}(h_t, \bar{h}_s) = \begin{cases} h_t^T \bar{h}_s & \rightarrow \text{dot} \\ h_t^T W_a \bar{h}_s & \rightarrow \text{general} \\ v_a^T \tanh(W_a \underbrace{[h_t ; \bar{h}_s]}_{\text{concat}}) & \rightarrow \text{concat} \end{cases}
$$

weight matrix

$$
a_t(s) = \overbrace{\text{align}}^{\text{e.g., softmax}}(h_t, \bar{h}_s)
$$

$$
= \frac{\exp(\text{score}(h_t, \bar{h}_s))}{\sum_{s'} \exp(\text{score}(h_t, \bar{h}_{s'}))}
$$

$$\tilde{h}_t = \tanh\left(W_c\,[\,c_t\,;\,h_t\,]\right)$$

attention context vector

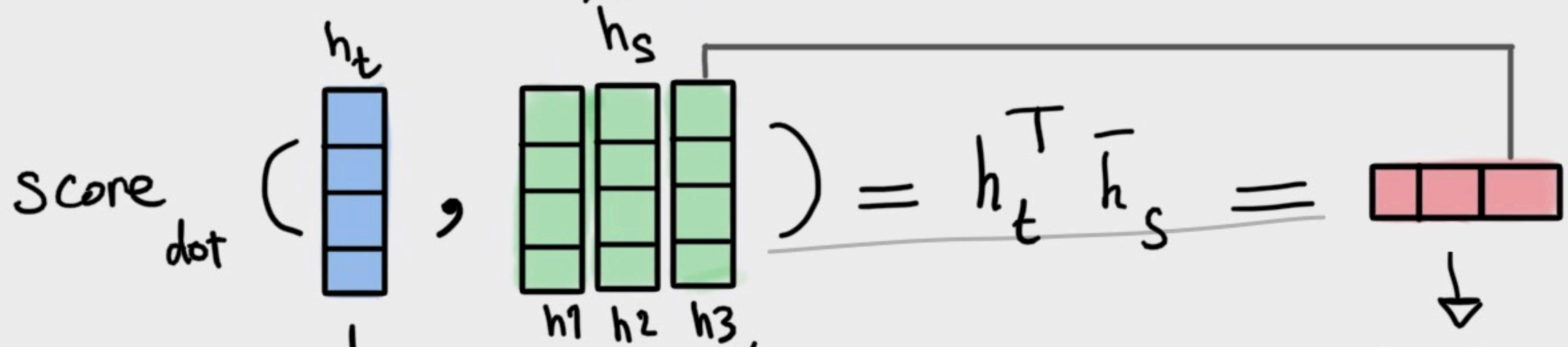
Walk-through Example

Silly example:

$$\text{Score}_{dot}\left(\underbrace{\overset{h_t}{\begin{bmatrix}3\\1\end{bmatrix}},\ \overset{h_1}{\begin{bmatrix}2\\6\end{bmatrix}}}\right) = 3\times 2 + 1\times 6 = 12$$

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:



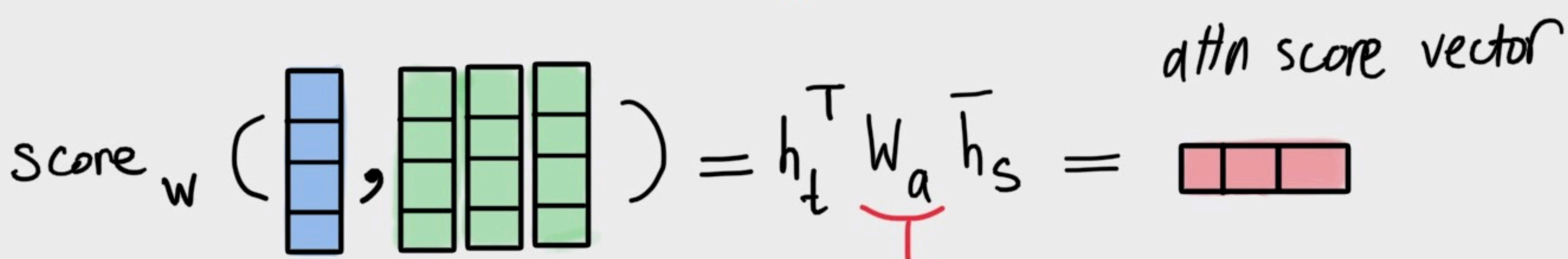
$n \times 1$ → # hidden units in the RNNs

$n \times t$ → # elements in input sequence

$1 \times t$ → attention score for each encoder hidden state wrt current decoder hidden state

ASSUMPTION: encoder and decoder have the same embedding space. → solution: use another score function

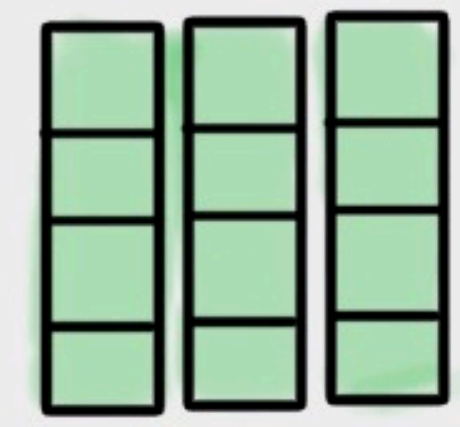
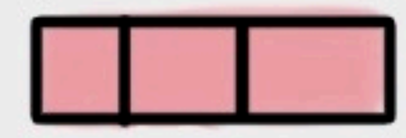
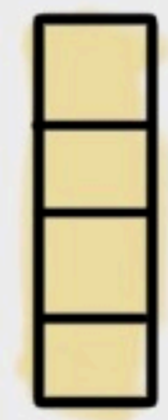
The second score function: aka (general)

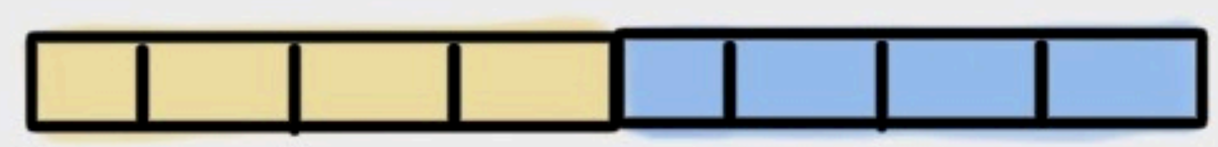


W_a → $m \times n$

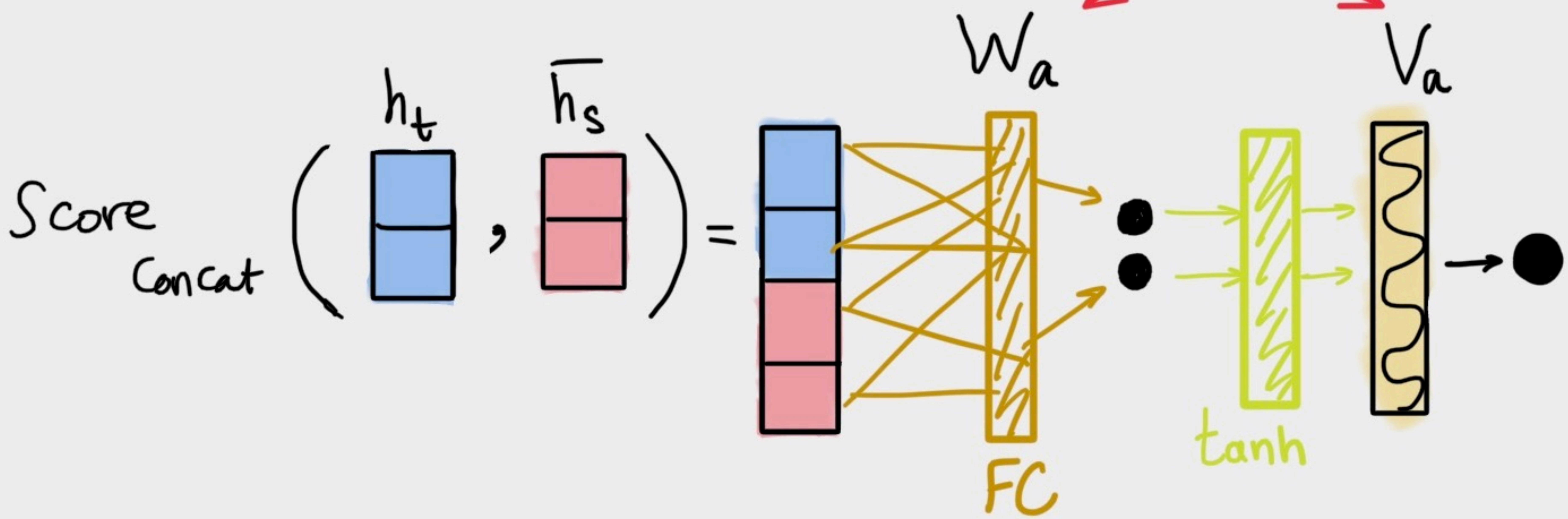
- weight matrix for a linear transformation
- trained jointly withe model

Where does it appear in the decoder?

we take sum of $\bar{h}_s$ () 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:



$$\rightsquigarrow v_a^T \tanh(W_a[h_t; \bar{h}_s])$$

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.