Language Understanding

04 - Sequence-to-Sequence Models with Attention

Hossein Zeinali

Introduction

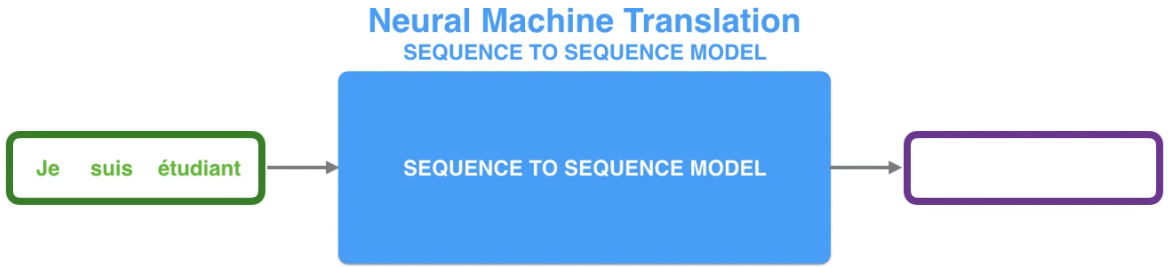
- We are going to learn how to model conditional probability like $P(y \mid x)$, where x and y are both sequences of discrete symbols
  - e.g. machine translation: x = "**I am a teacher**" and y = "من معلم هستم"
- Let x be the source (English) sentence, y be the target (Persian) sentence.

$$x = x_1 x_2 \dots x_n$$
$$y = y_1 y_2 \dots y_m$$

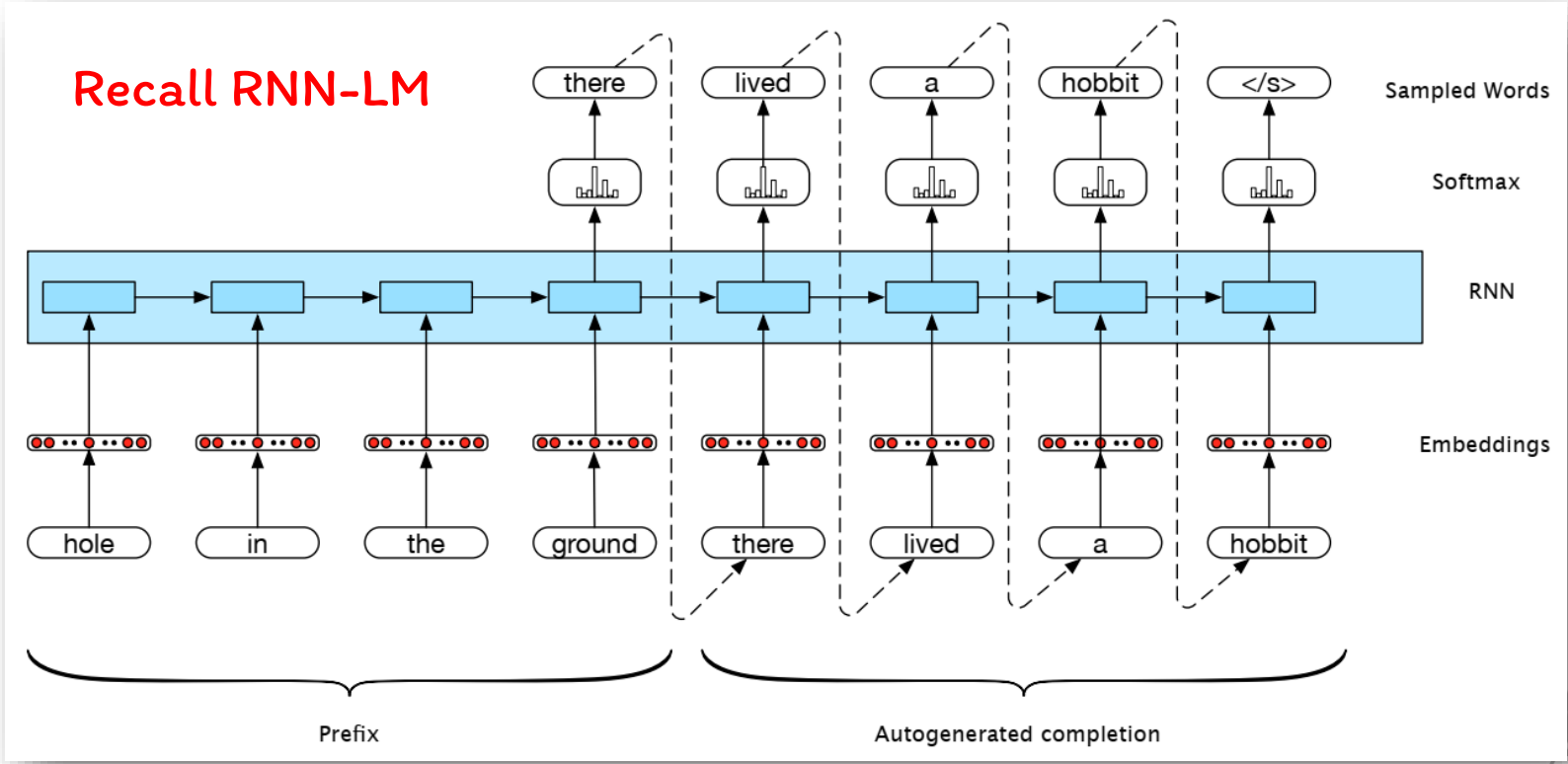
- How can we define $P(y|x)$?
- Expand it using the chain rule:

$$P(y|x) = \prod_{i=1}^{m+1} P(y_i | y_1 \dots y_{i-1} x_1 \dots x_n)$$

Introduction



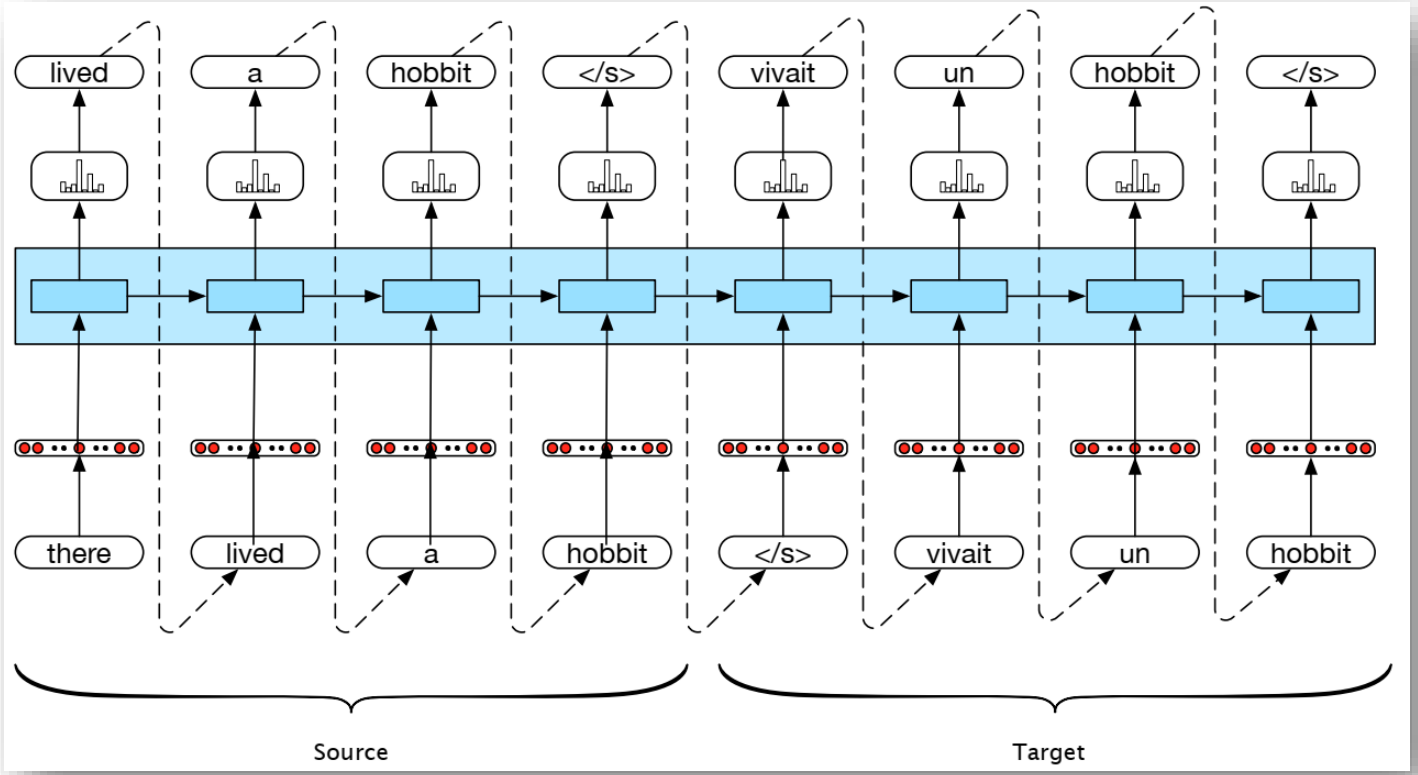
Using RNN-LM for Translation



Using RNN-LM for Translation

- LM vs MT:
  - In LM we are interested in $P(y_1, y_2, \dots, y_m)$ while in MT are interested in $P(y_1, y_2, \dots, y_m | x_1, x_2, \dots, x_n)$
- Training data for MT is parallel texts, or **bitexts**.
  - The text being translated from the **source** to the **target**.
- How to extend language models and autoregressive generation to machine translation:
  - Add an end-of-sentence marker at the end of each bitext's source sentence
  - Concatenate the corresponding target to it
  - Use concatenated source-target pairs as training data of NN
  - This is a model of $P(x, y) = P(x)P(y|x)$. But we don't care about $P(x)$.

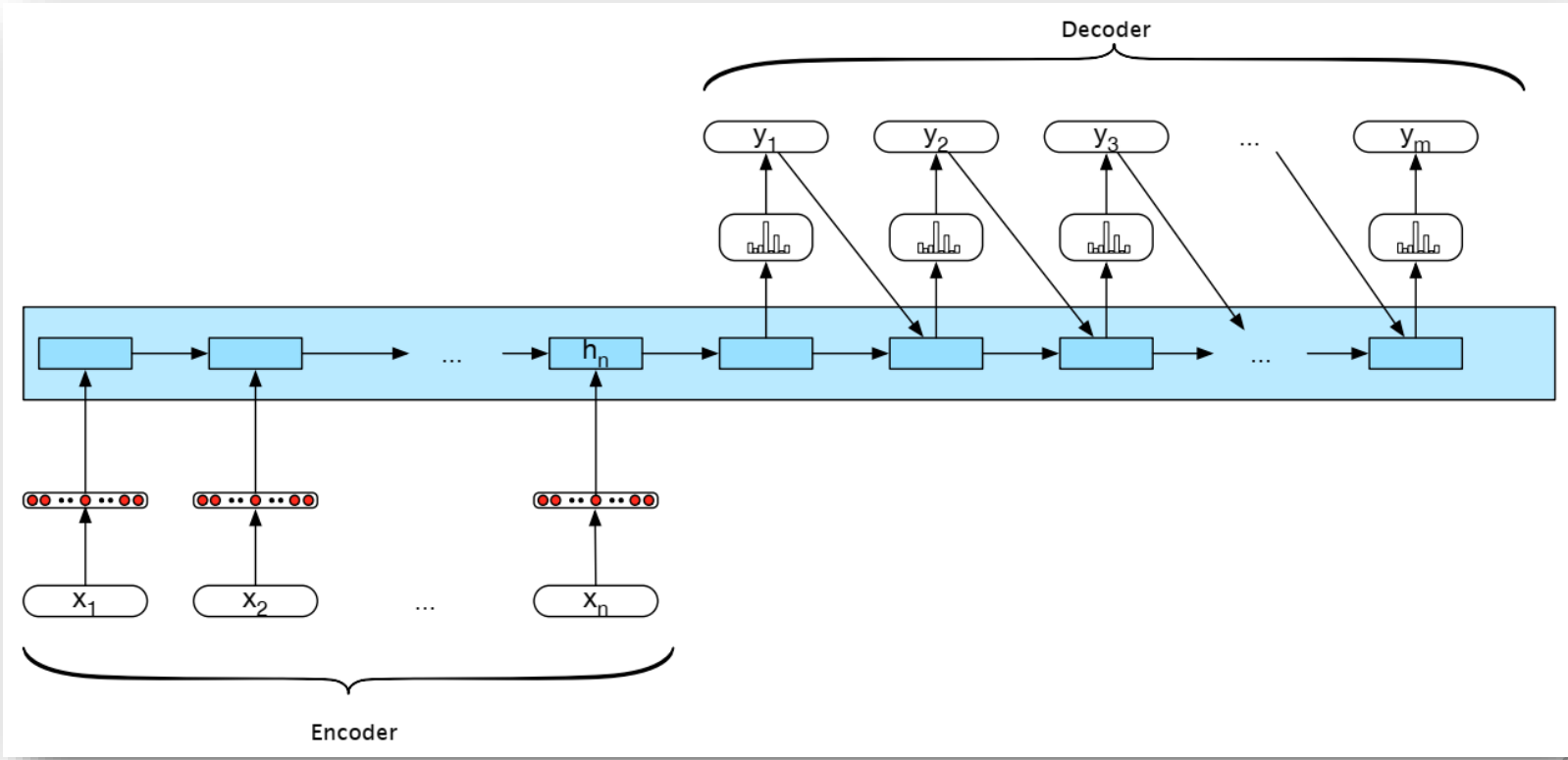
Using RNN-LM for Translation



Using RNN-LM for Translation

- This clever approach demonstrated surprisingly good results
- But, is this the best way for doing MT using RNNs?
  - In many cases, the source and target language may have very different vocabulary and morphological/syntactic structure.
- **Insight.** We need two RNNs, one for source and one for target.
  - We will see Encoder-Decoder topology for this problem.

Encoder-Decoder Networks

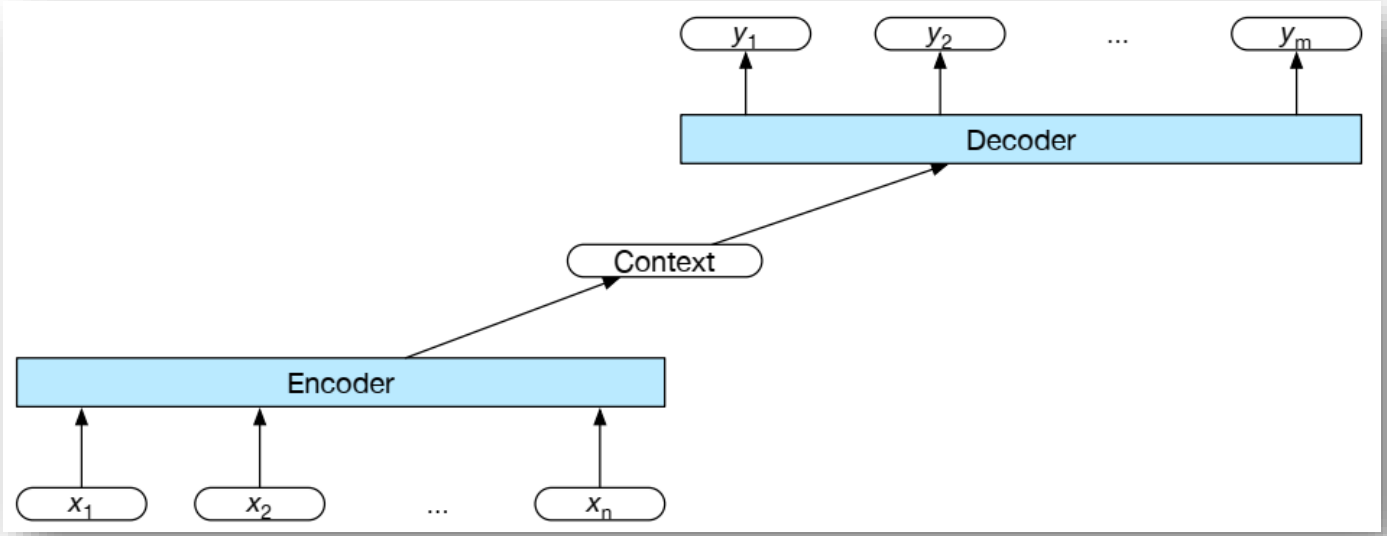


Encoder-Decoder Networks

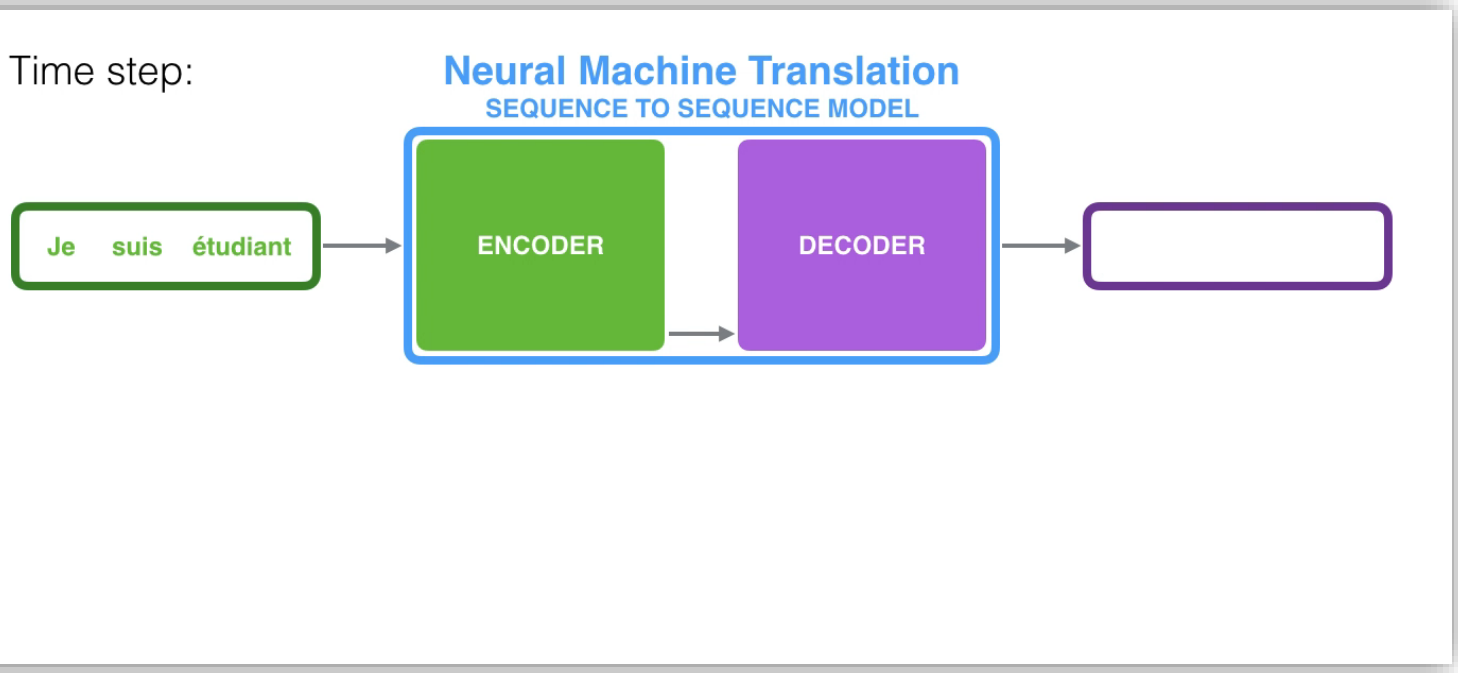
- **Encoder**: generates a contextualized representation of the input that is embodied in the final hidden state of the encoder, h_n, which in turn feeds into the first hidden state of the decoder.
- **Decoder:** takes the last state of the encoder and autoregressively generates a sequence of outputs.
- Problems:
  - The encoder and the decoder are assumed to have the same internal structure
  - The final state of the encoder is the only context available to the decoder
  - The context is only available to the decoder as its initial hidden state

Encoder-Decoder Networks

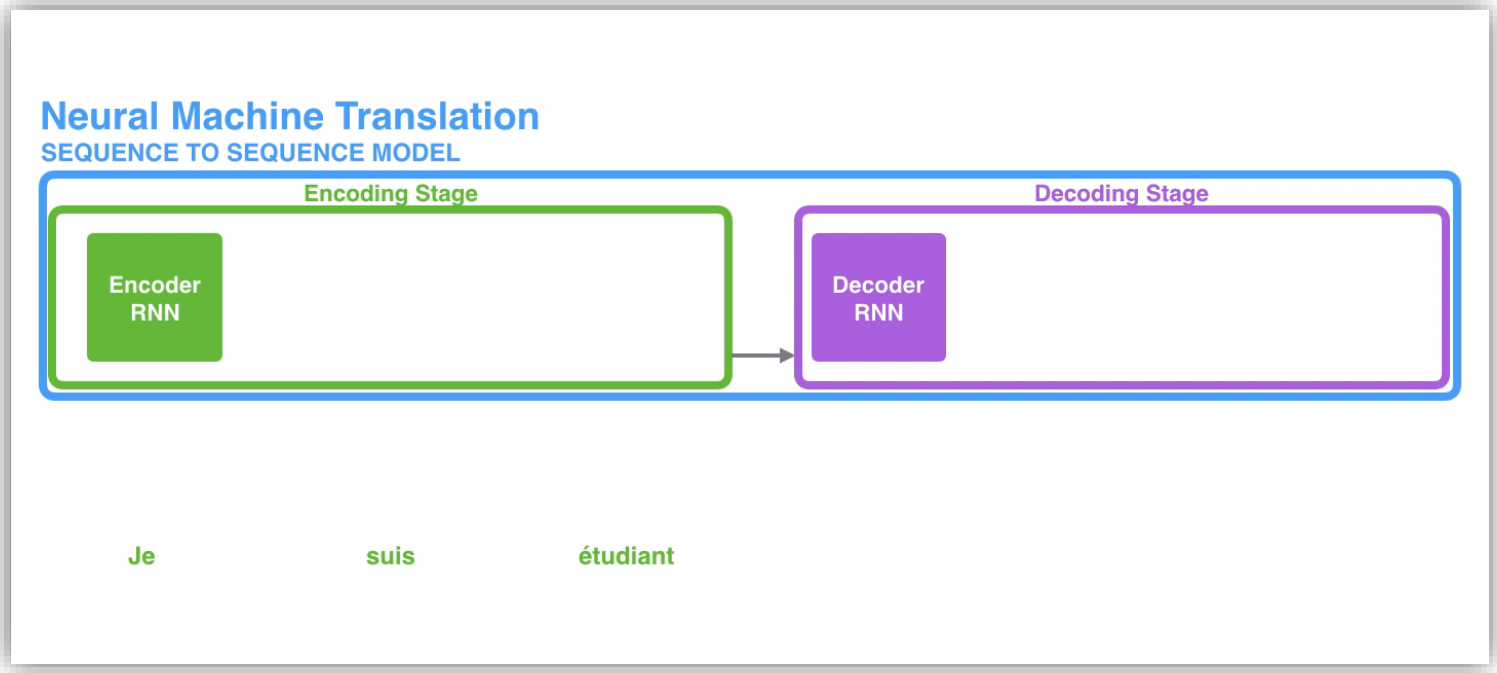
- **Encoder**: Simple RNNs, LSTMs, GRUs and convolutional networks can all be been employed as encoders.
- **Decoder**: LSTM or GRU-based RNN can be used.



Encoder-Decoder Networks

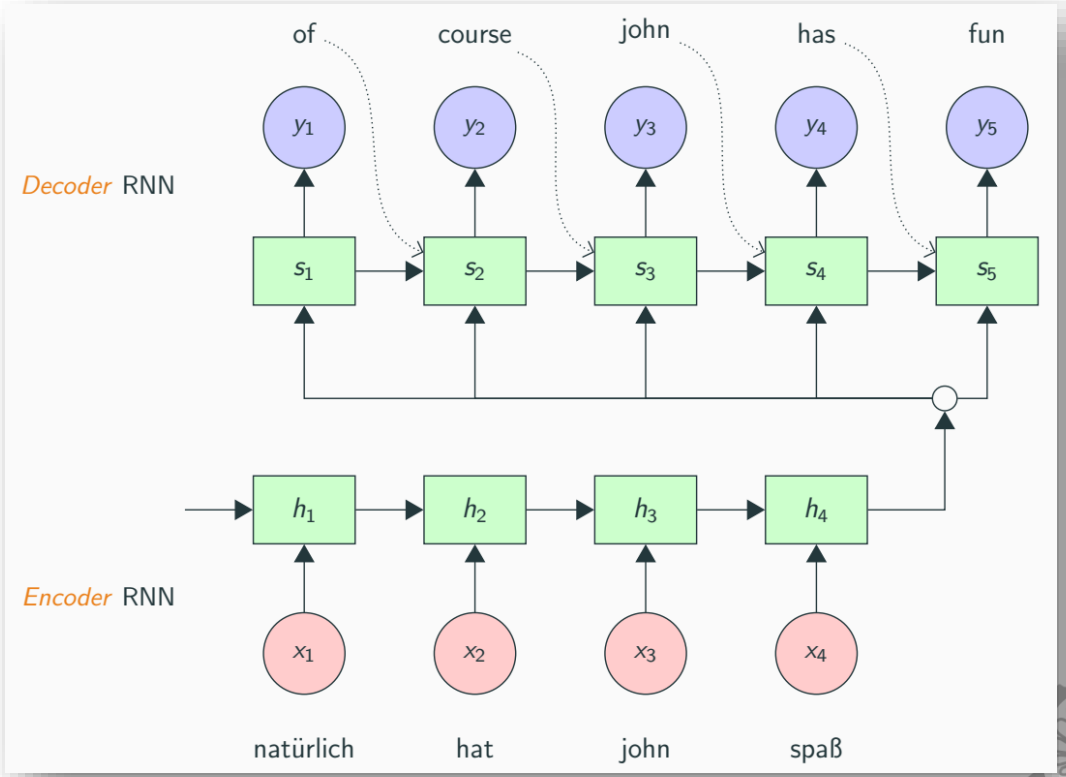


Encoder-Decoder Networks

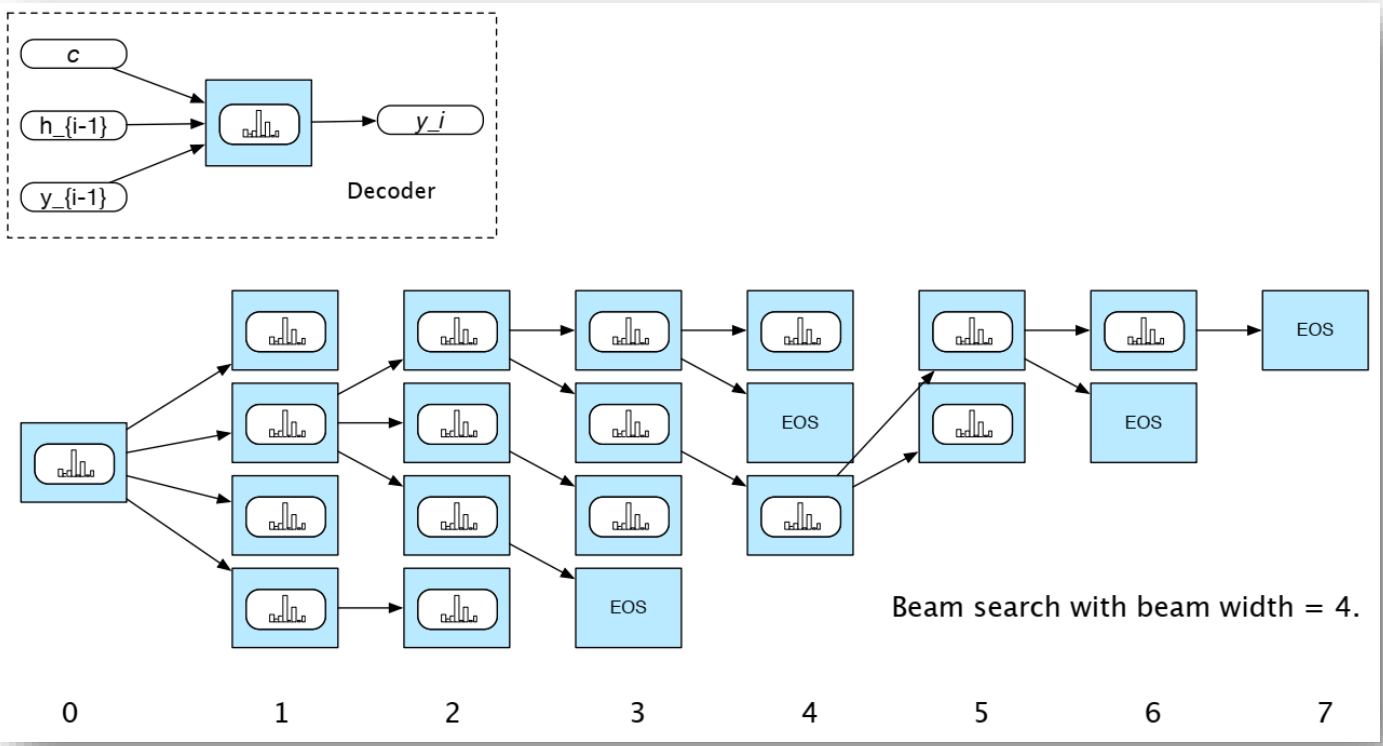


Encoder-Decoder Networks

- Better strategy: make the context vector c available at each step in the decoding process.



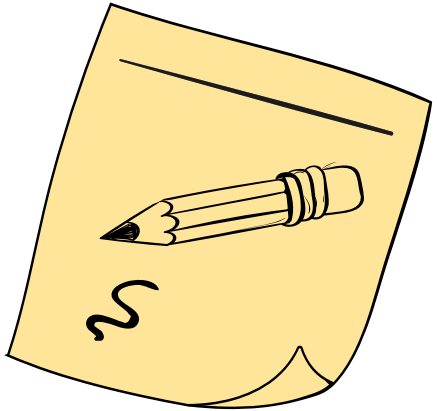
Beam Search Decoding



Context Modeling

- The context vector c is defined as a function of the hidden states of the encoder, that is, $c = f(h_1^n)$
  - Problem: the number of hidden states varies with the size of the input
- First non-genius solutions:
  - Using the last hidden state of the encoder as the context
    - Despite its simplicity, the final hidden state inevitably is more focused on the latter parts of input sequence, rather than the input as whole.
  - Using Bi-RNNs and its forward and backward last hidden states
    - Concatenation, Sum or Average of two vectors can be used
- The best solution:
  - Using **attention mechanism**

Attention Mechanism



Attention Mechanism

- The best context in each decoder state is using a combination of all encoder hidden states

$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$$

- A vector of scores is needed that capture the relevance of each encoder hidden state (h^e) to the decoder state captured in h_{i-1}^d.
- Simplest score is dot product:

$$score(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e$$

- Dot product is a static measure that does not facilitate adaptation during the training. Let's change it to:

$$score(h_{i-1}^d, h_j^e) = h_{i-1}^d W_s h_j^e$$

Attention Mechanism

- Normalize the scores with a softmax

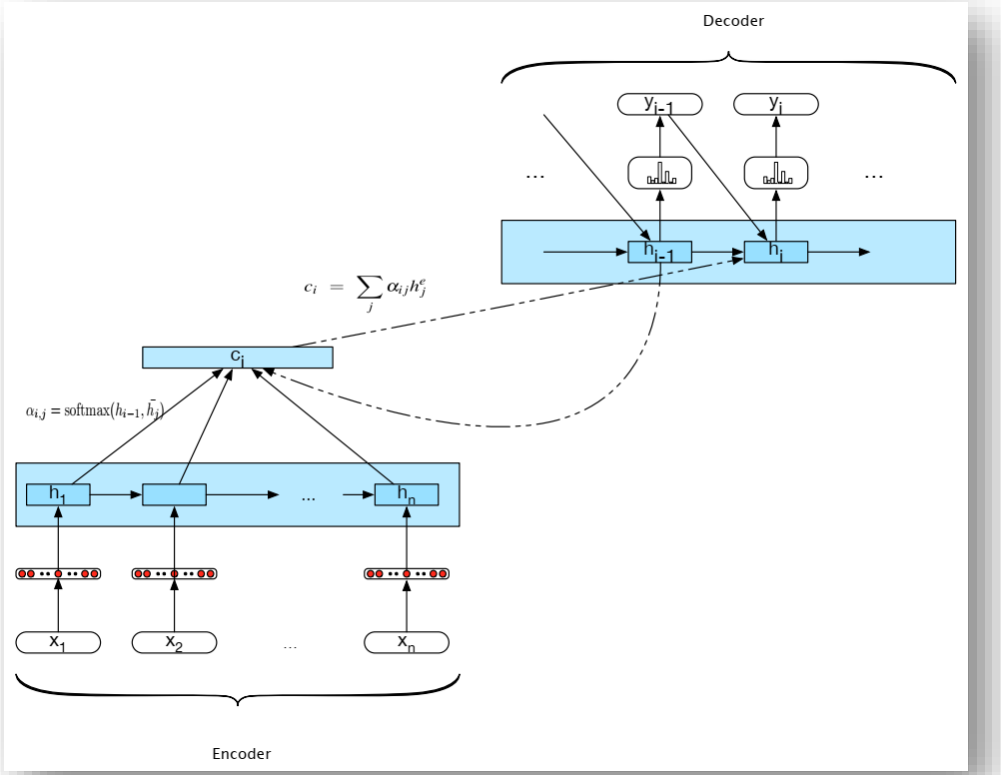
$$\alpha_{i,j} = \text{softmax}\left(score\left(h_{i-1}^{d}, h_{j}^{e}\right) \forall j \in e\right) = \frac{\exp\left(score\left(h_{i-1}^{d}, h_{j}^{e}\right)\right)}{\sum_{k} \exp\left(score\left(h_{i-1}^{d}, h_{k}^{e}\right)\right)}$$

- Finally, a fixed-length context vector is calculated by taking a weighted average over all the encoder hidden states

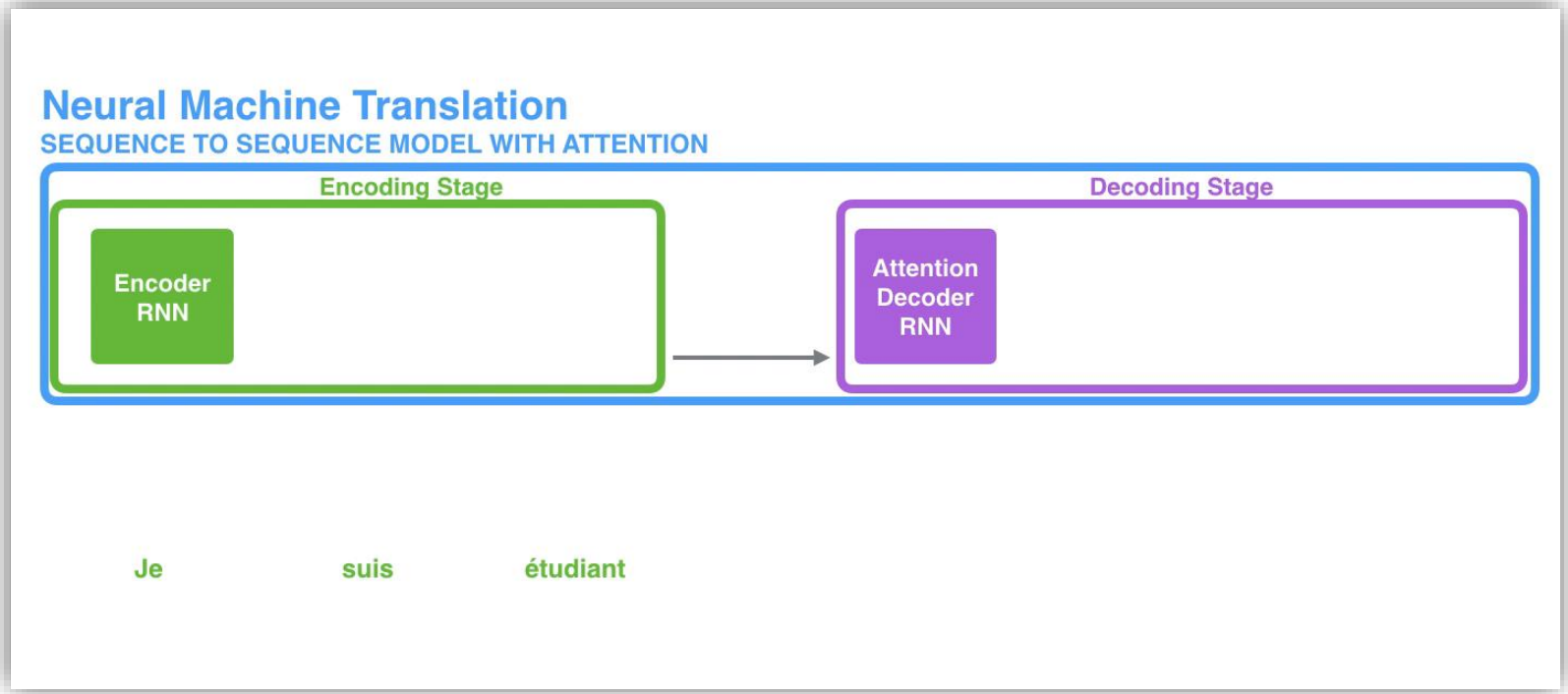
$$c_i = \sum_{j} \alpha_{i,j} h_{j}^{e}$$

- c_i is a fixed-length context vector that takes into account information from the entire encoder state that is dynamically update to reflect the needs of the decoder at each step of decoding.

Attention Mechanism

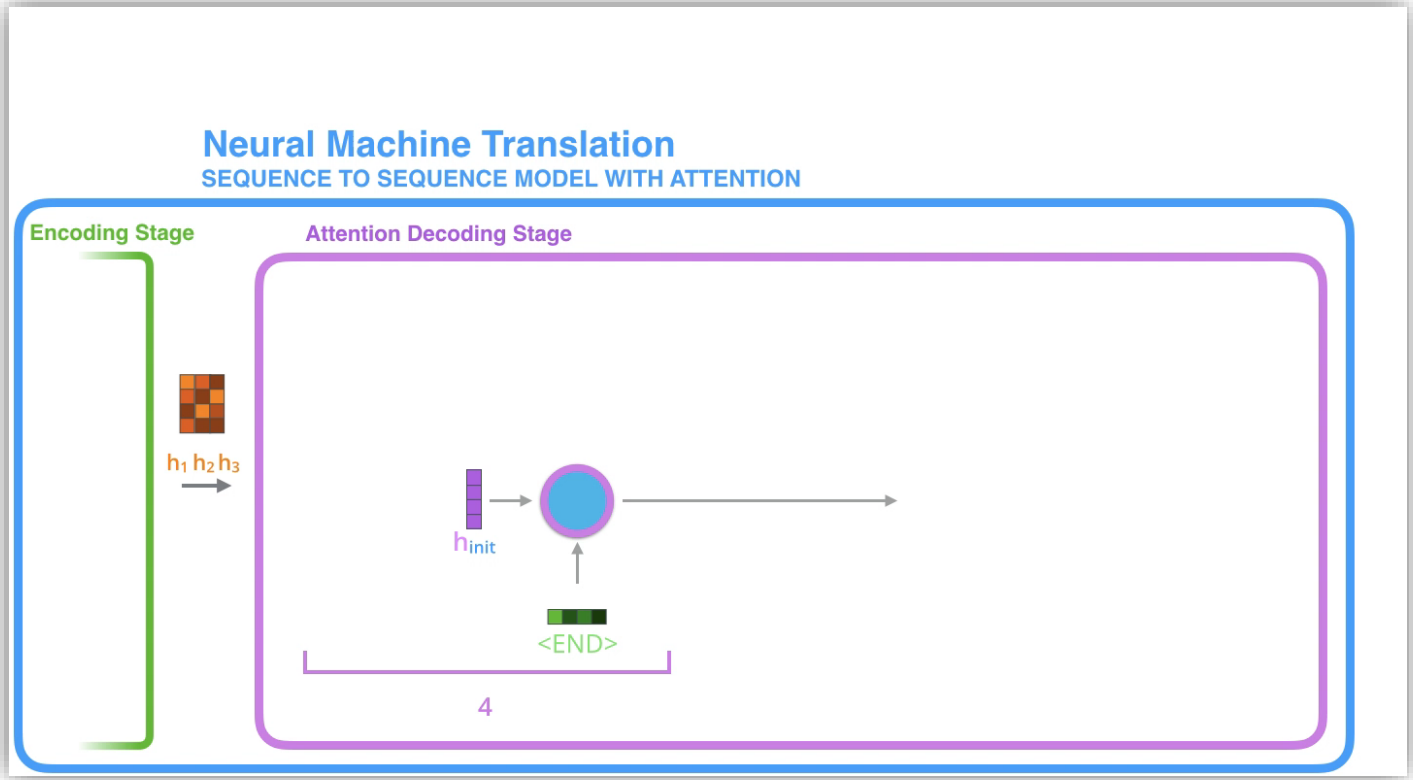


Attention Mechanism

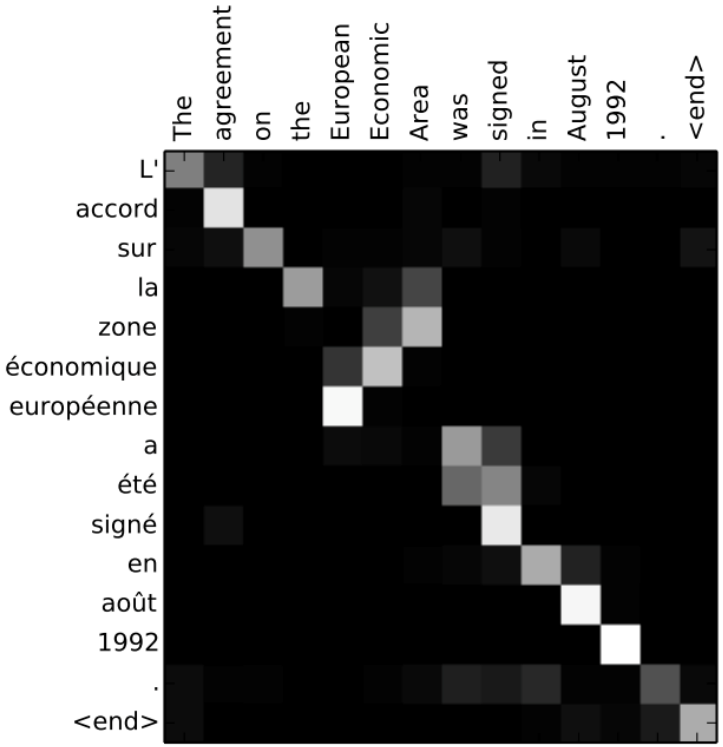


Attention Mechanism

Attention: All Together



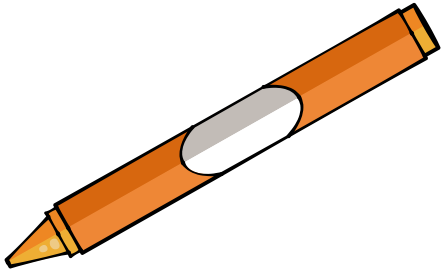
Attention Matrix



Attention Aside

- Attention treats each hidden state of an RNN as a representation of the corresponding source word.
- RNNs is a way of summarizing the complete history of a sequence. So h_i represents $x_1 \dots x_i$.
  - Actually h_i mostly represents x_i, but in the context of $x_1 \dots x_i$.
  - This is useful for disambiguate different meanings of x_i,
- What about context $x_{i+1} \dots x_n$?
  - Use a bidirectional RNN, one left-to-right, one right-to-left and concatenate their states at position i.

Thanks for your attention



References and IP Notice

- [1] Daniel Jurafsky and James H. Martin, “Speech and Language Processing”, 3rd ed., 2019
- [2] Graham Neubig, "Neural machine translation and sequence-to-sequence models: A tutorial." (2017).

- Animation videos are from https://jalammar.github.io web site.
  - https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/
- Some graphics were selected Slidesgo template