어텐션 매커니즘
for 스터디

Tae Hwan Jung

https://ratsgo.github.io/from%20frequency%20to%20semantics/2017/10/06/attention/ 를 참조하여 만들었습니다.

1. Seq2Seq

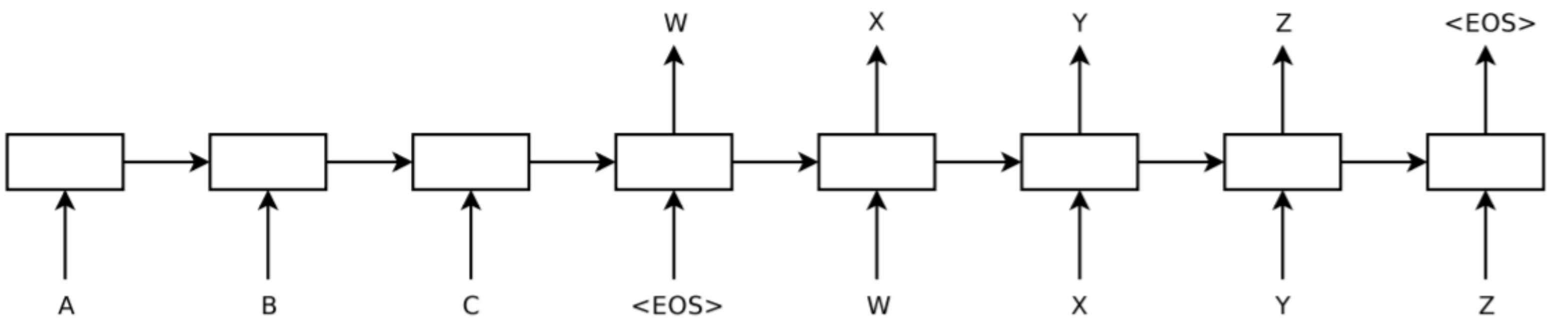


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

- https://arxiv.org/pdf/1409.3215.pdf 출처

1. Seq2Seq

Encoder **Decoder**

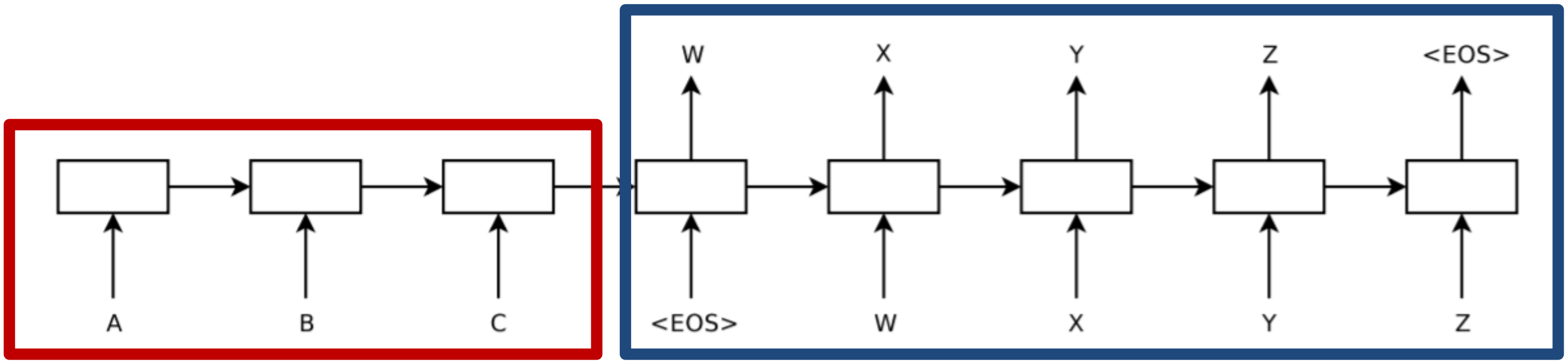


Figure 1: Our model reads an input sentence “ABC” and produces “WXYZ” as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

- https://arxiv.org/pdf/1409.3215.pdf 출처

1. Seq2Seq

Encoder **Decoder**

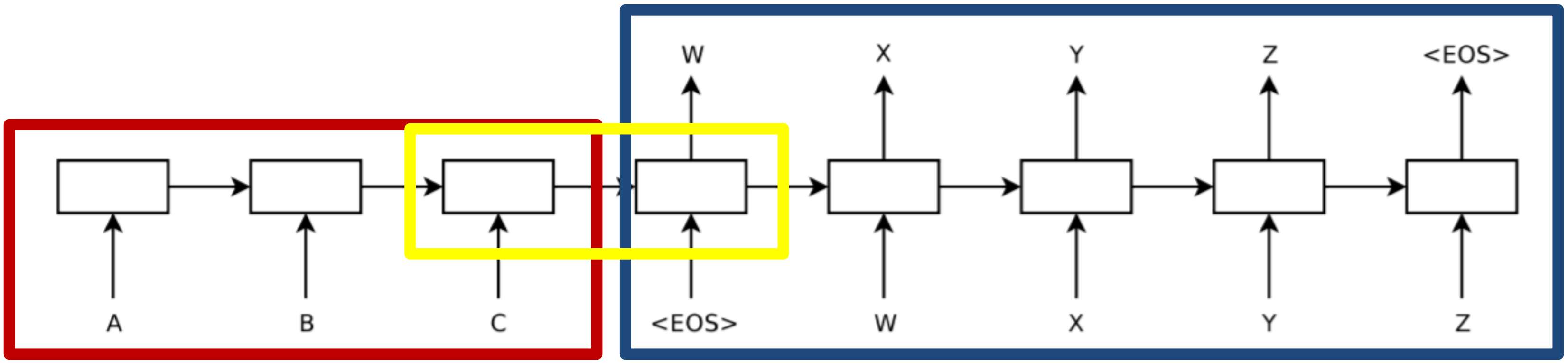


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

Encoder의 최종 hidden State가 Deocder의 initialize Hidden State로 들어간다.

- https://arxiv.org/pdf/1409.3215.pdf 출처

1. Seq2Seq

Inputs: input, h_0

- **input** of shape (*seq_len, batch, input_size*): tensor containing the features of the input sequence. The input can also be a packed variable length sequence. See `torch.nn.utils.rnn.pack_padded_sequence()` or `torch.nn.utils.rnn.pack_sequence()` for details.
- **h_0** of shape (*num_layers * num_directions, batch, hidden_size*): tensor containing the initial hidden state for each element in the batch. Defaults to zero if not provided. If the RNN is bidirectional, num_directions should be 2, else it should be 1.

- **https://pytorch.org/docs/stable/nn.html**

1. Seq2Seq

Outputs: output, h_n

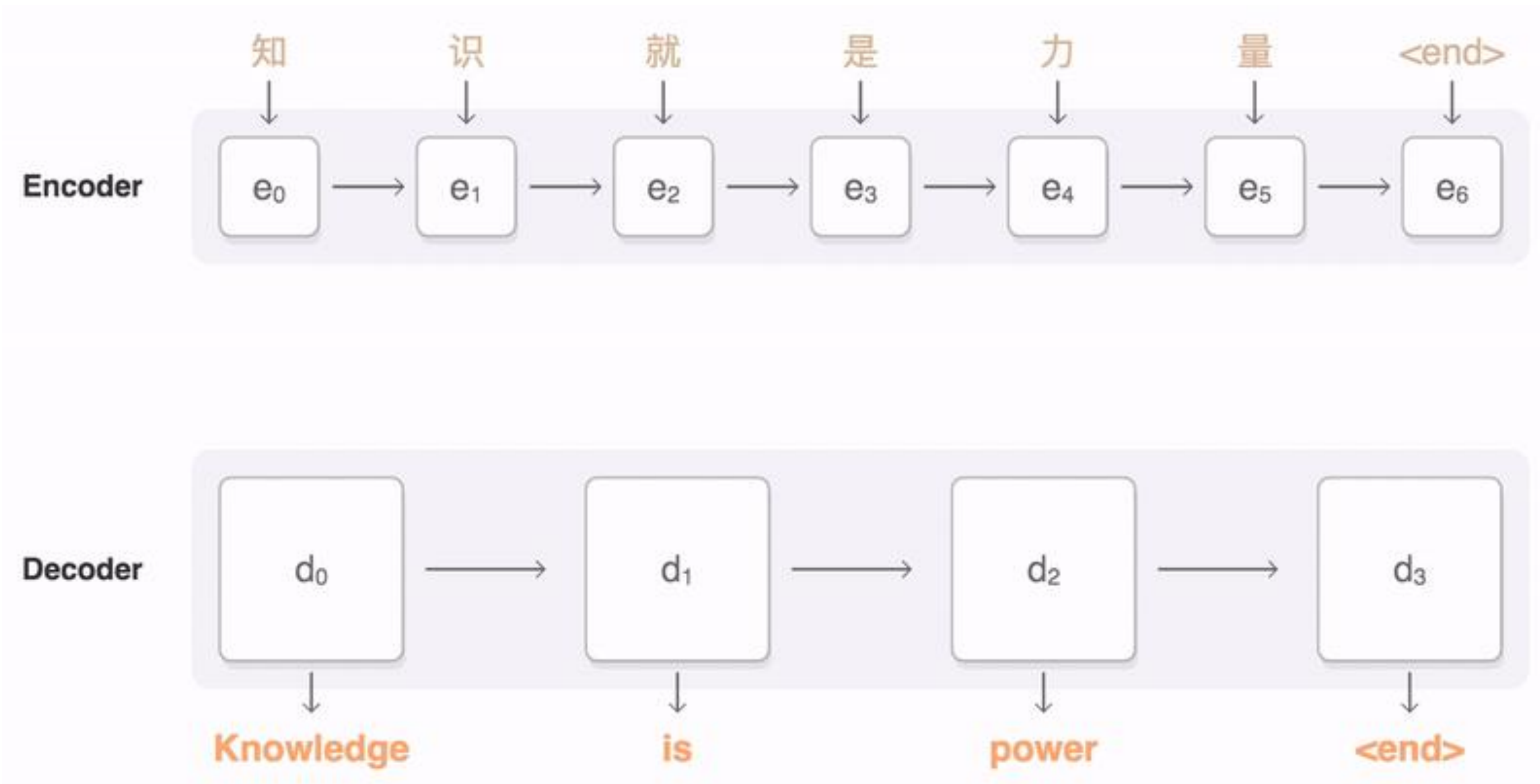
- **output** of shape (*seq_len, batch, num_directions * hidden_size*): tensor containing the output features (*h_k*) from the last layer of the RNN, for each *k*. If a `torch.nn.utils.rnn.PackedSequence` has been given as the input, the output will also be a packed sequence.

  For the unpacked case, the directions can be separated using `output.view(seq_len, batch, num_directions, hidden_size)`, with forward and backward being direction *0* and *1* respectively. Similarly, the directions can be separated in the packed case.
- **h_n** (num_layers * num_directions, batch, hidden_size): tensor containing the hidden state for *k = seq_len*.

  Like *output*, the layers can be separated using `h_n.view(num_layers, num_directions, batch, hidden_size)`.

- https://pytorch.org/docs/stable/nn.html

2. Seq2Seq-Attention



- https://google.github.io/seq2seq/ 출처

2. Seq2Seq-Attention

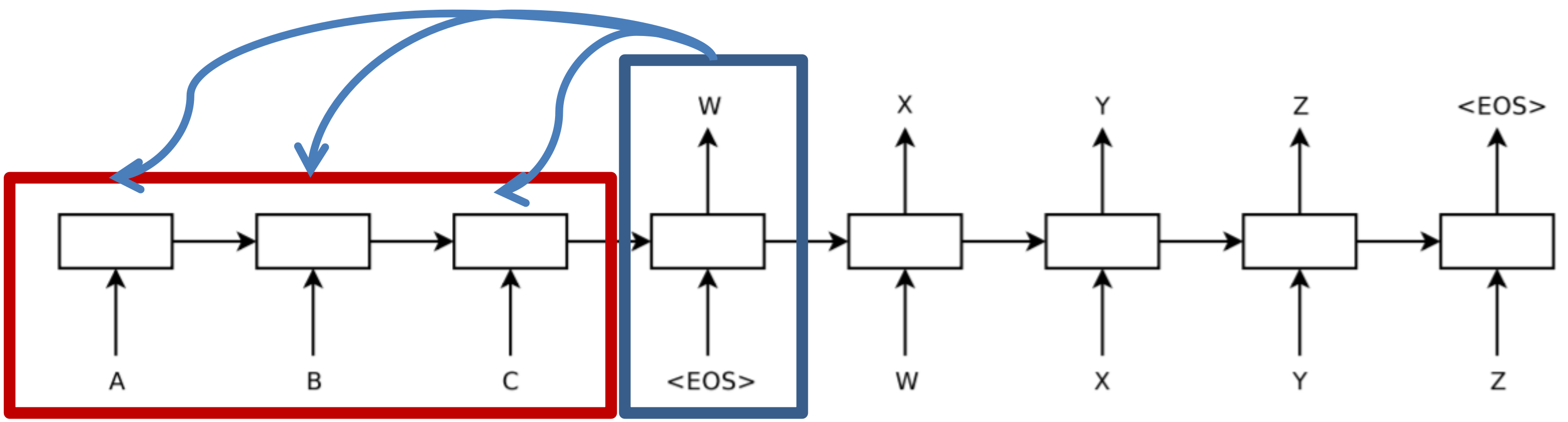


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

$$e_{ij} = a(s_{i-1}, h_j)$$

E_ij = alignment model(디코더의 이전 hidden state, 인코더의 hidden state)

2. Seq2Seq-Attention

$$\overrightarrow{\alpha_i} = [\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{iT_x}]$$

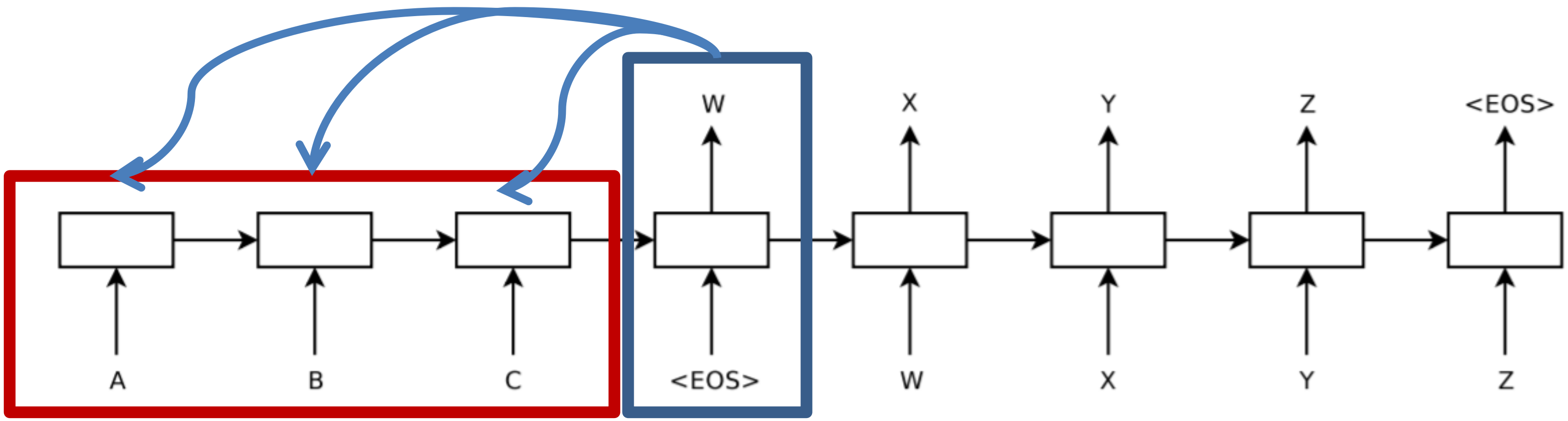


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

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

2. Seq2Seq-Attention

Contextual matrix

Query

$$\overrightarrow{c_i} = \sum_{j=1}^{T_x} \alpha_{ij} h_j = \boxed{F}\,\overrightarrow{\alpha_i}$$

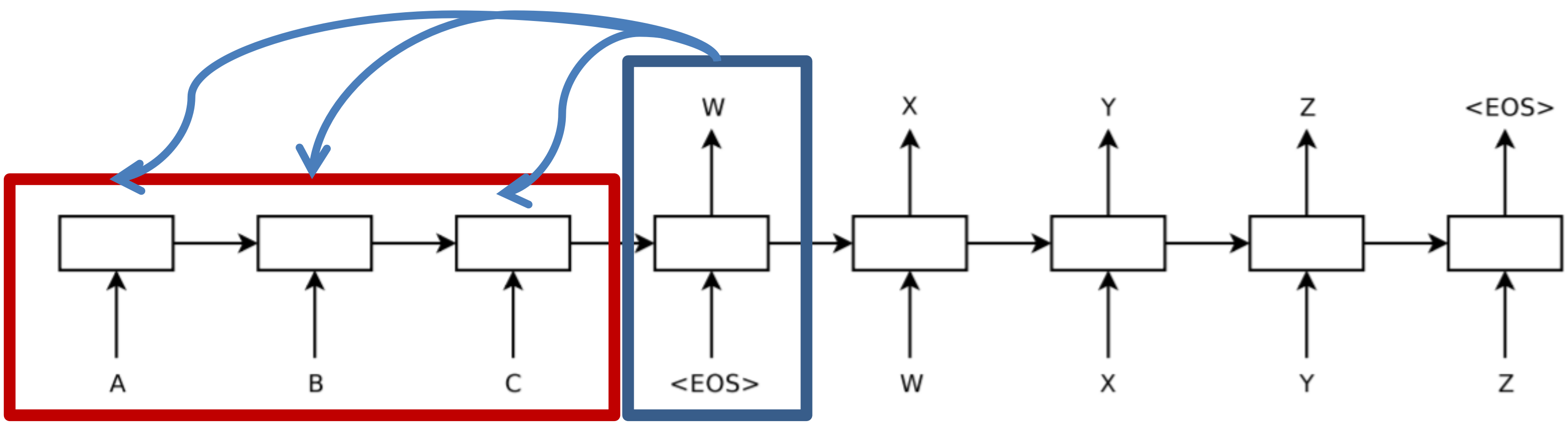


Figure 1: Our model reads an input sentence “ABC” and produces “WXYZ” as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

2. Seq2Seq-Attention

model[i] = self.out(torch.cat((dec_output, context), 1))

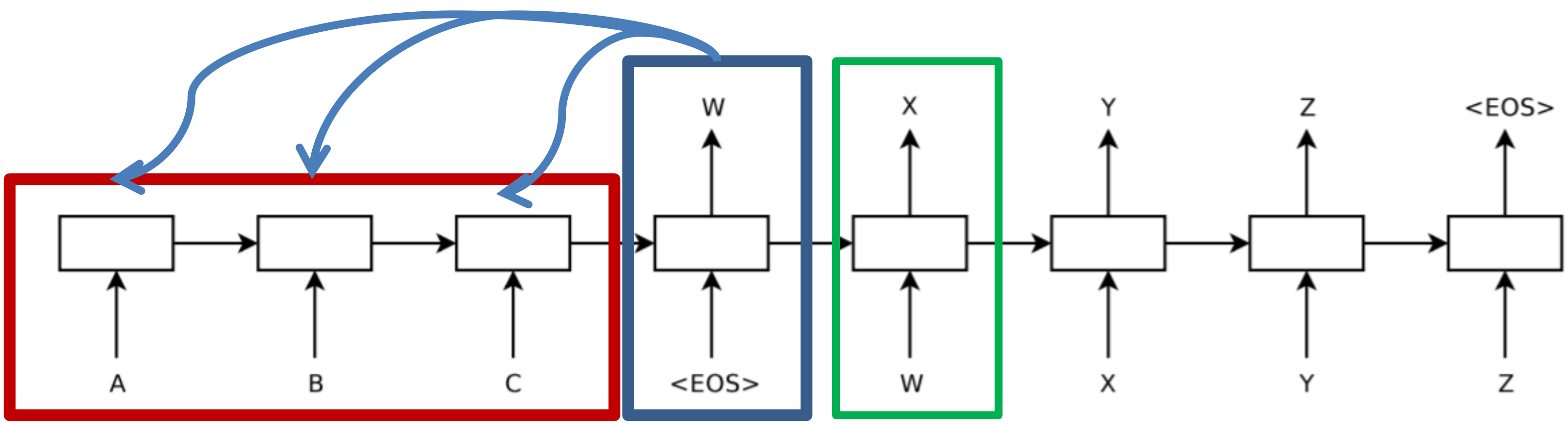


Figure 1: Our model reads an input sentence “ABC” and produces “WXYZ” as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

2. Seq2Seq-Attention

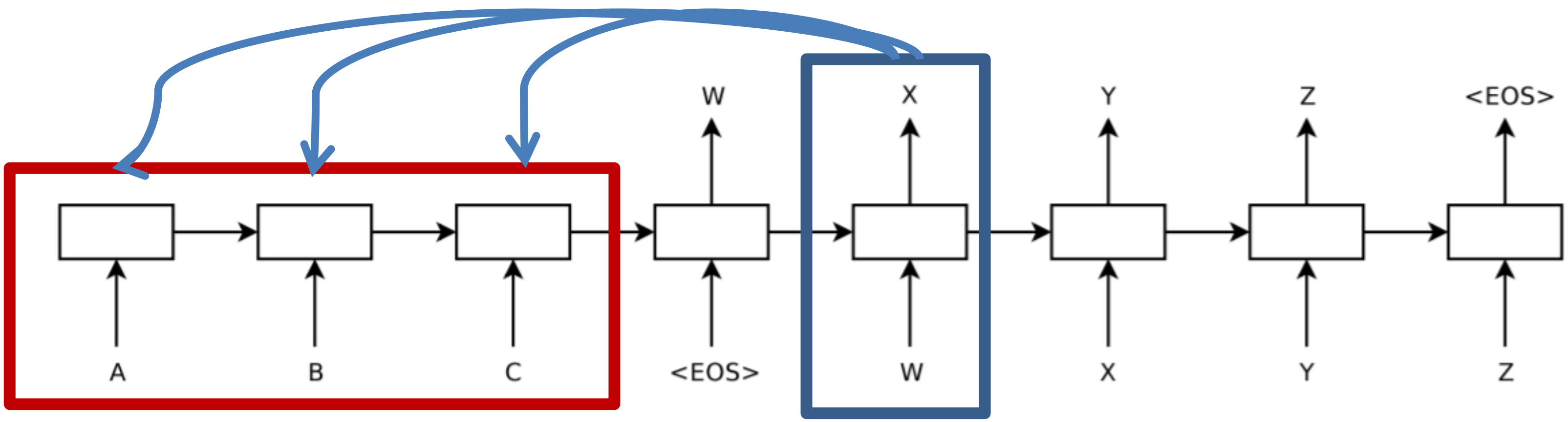


Figure 1: Our model reads an input sentence “ABC” and produces “WXYZ” as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

2. Seq2Seq-Attention

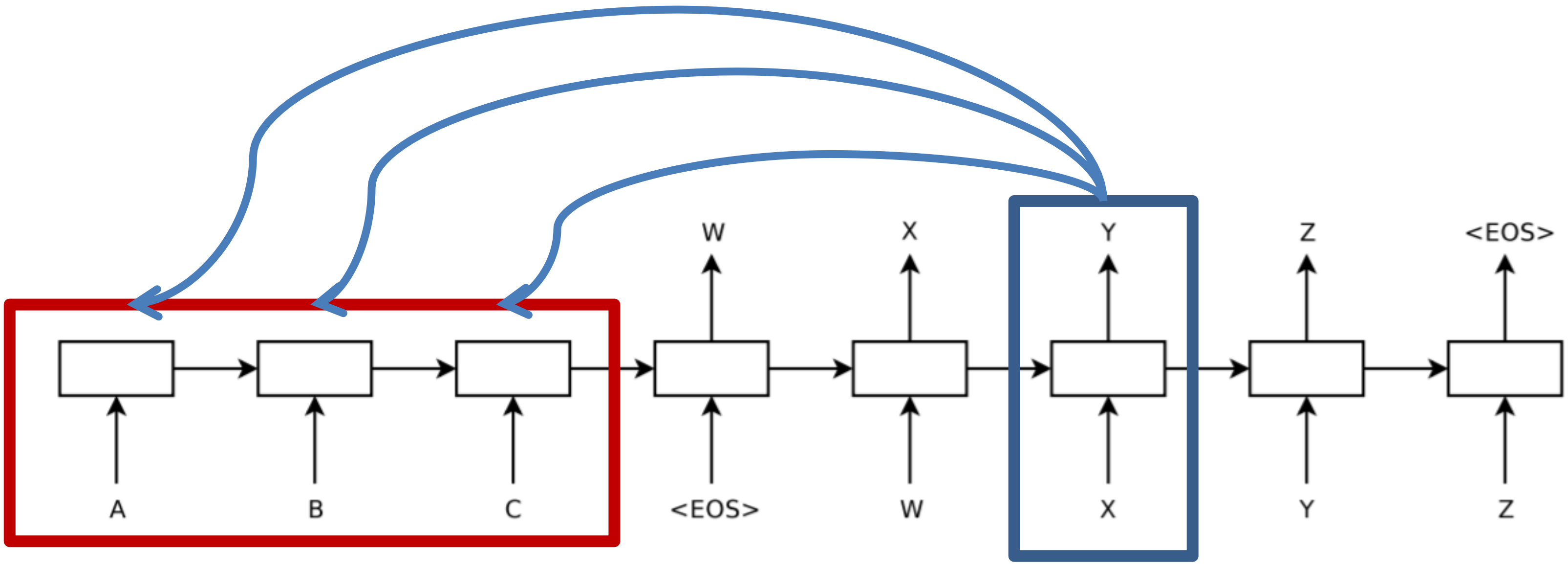


Figure 1: Our model reads an input sentence “ABC” and produces “WXYZ” as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

2. Seq2Seq-Attention

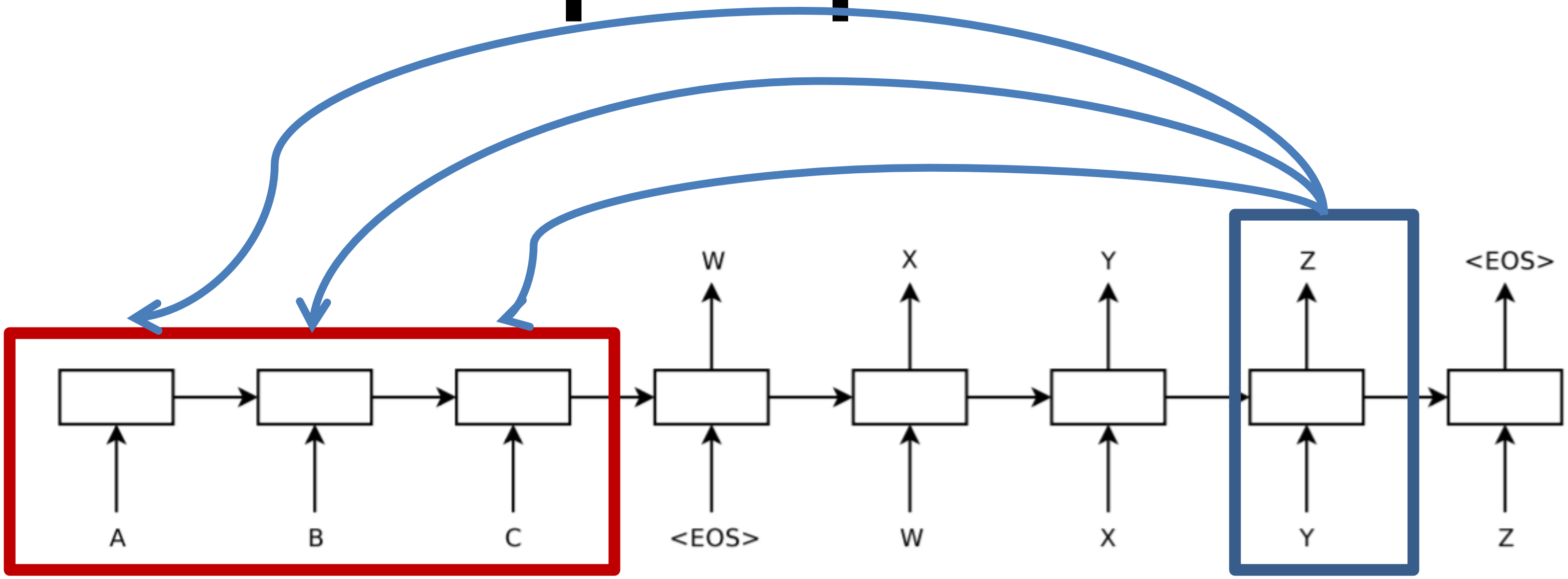


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

2. Seq2Seq-Attention

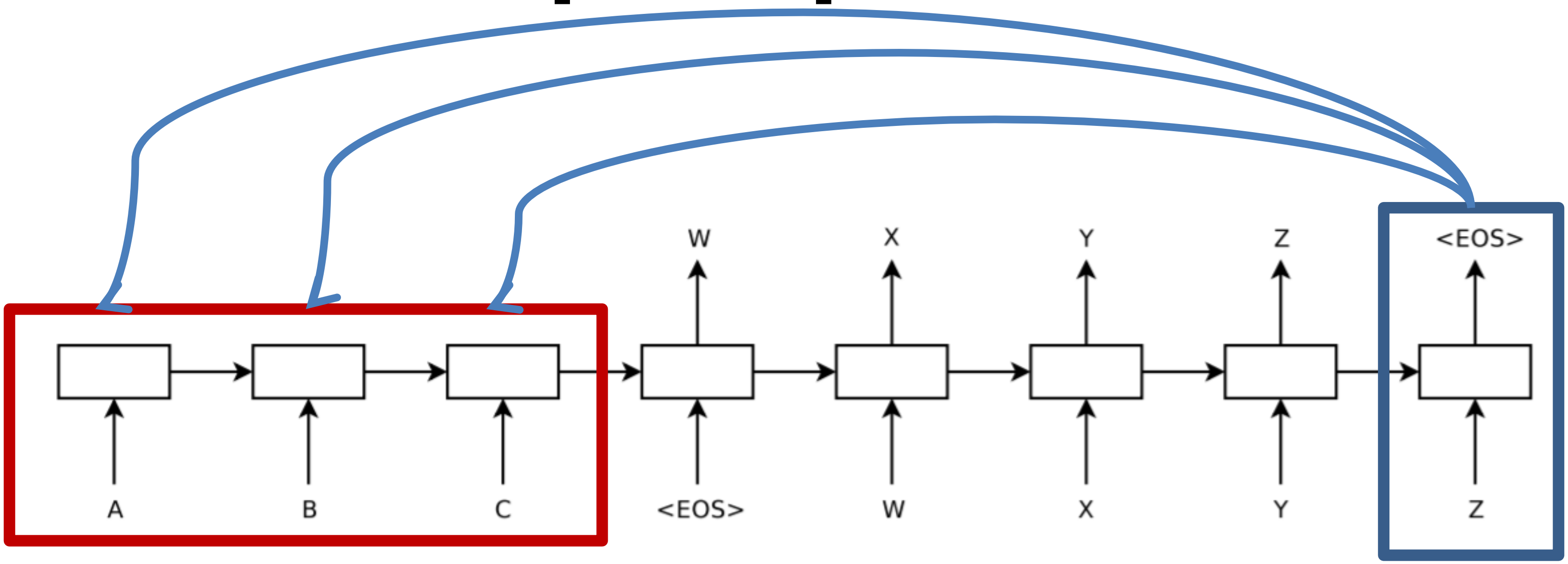


Figure 1: Our model reads an input sentence “ABC” and produces “WXYZ” as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.