$$p_{\theta_{dec}}(\mathbf{Y}_{1:m}|\mathbf{c}).$$

By Bayes' rule the distribution can be decomposed into conditional distributions of single target vectors as follows:

$$p_{\theta_{dec}}(\mathbf{Y}_{1:m}|\mathbf{c}) = \prod_{i=1}^{m} p_{\theta_{\text{dec}}}(\mathbf{y}_i|\mathbf{Y}_{0:i-1}, \mathbf{c}).$$

Thus, if the architecture can model the conditional distribution of the next target vector, given all previous target vectors:

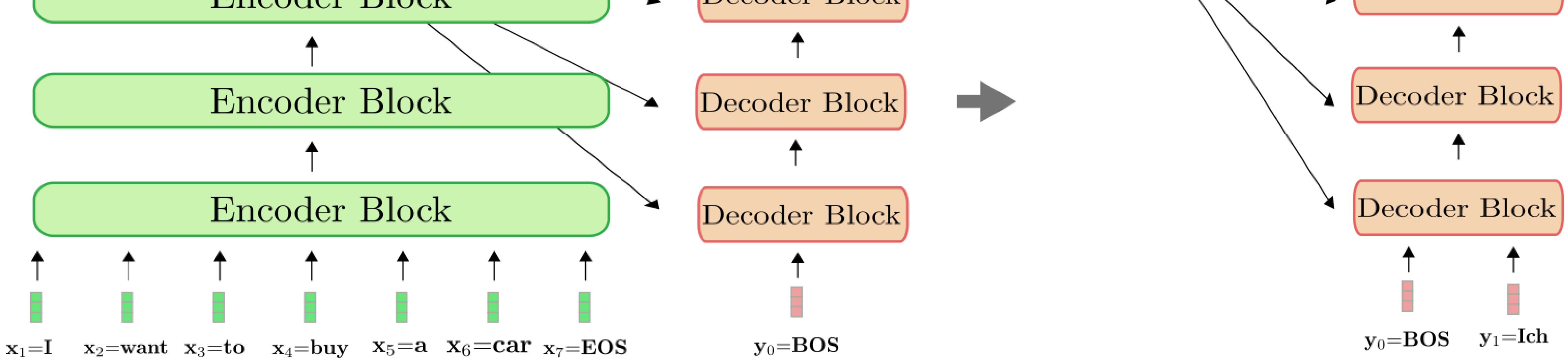
$$p_{\theta_{\text{dec}}}(\mathbf{y}_i|\mathbf{Y}_{0:i-1}, \mathbf{c}), \forall i \in \{1, \ldots, m\},$$

then it can model the distribution of any target vector sequence given the hidden state $\mathbf{c}$ by simply multiplying all conditional probabilities.

So how does the RNN-based decoder architecture model $p_{\theta_{\text{dec}}}(\mathbf{y}_i|\mathbf{Y}_{0:i-1}, \mathbf{c})$?

In computational terms, the model sequentially maps the previous inner hidden state $\mathbf{c}_{i-1}$ and the previous target vector $\mathbf{y}_i$ to the current inner hidden state $\mathbf{c}_i$ and a *logit vector* $\mathbf{l}_i$ (shown in dark red below):

$$f_{\theta_{\text{dec}}}(\mathbf{y}_{i-1}, \mathbf{c}_{i-1}) \to \mathbf{l}_i, \mathbf{c}_i.$$



As can be seen, only in step $i = 1$ do we have to encode "I want to buy a car EOS" to $\overline{\mathbf{X}}_{1:7}$. At step $i = 2$, the contextualized encodings of "I want to buy a car EOS" are simply reused by the decoder.

In 🤗Transformers, this auto-regressive generation is done under-the-hood when calling the `.generate()` method. Let's use one of our translation models to see this in action.

Encoder

As mentioned in the previous section, the *transformer-based* encoder maps the input directional self-attention.

As in bi-directional self-attention, in uni-directional self-attention, the query vectors $\mathbf{q}_0, \ldots, \mathbf{q}_{m-1}$ (shown in purple below), key vectors $\mathbf{k}_0, \ldots, \mathbf{k}_{m-1}$ (shown in orange below), and value vectors $\mathbf{v}_0, \ldots, \mathbf{v}_{m-1}$ (shown in blue below) are projected from their respective input vectors $\mathbf{y'}_0, \ldots, \mathbf{y}_{m-1}$ (shown in light red below). However, in uni-directional self-attention, each query vector $\mathbf{q}_i$ is compared *only* to its respective key vector and all previous ones, namely $\mathbf{k}_0, \ldots, \mathbf{k}_i$ to yield the respective *attention weights*.
This prevents an output vector $\mathbf{y''}_j$ (shown in dark red below) to include any information about the following input vector $\mathbf{y}_i, \text{ with } i > 1$ for all $j \in \{0, \ldots, m-1\}$. As is the case in bi-directional self-attention, the attention weights are then multiplied by their respective value vectors and summed together.

We can summarize uni-directional self-attention as follows:

$$\mathbf{y''}_i = \mathbf{V}_{0:i}\mathbf{Softmax}(\mathbf{K}_{0:i}^\intercal \mathbf{q}_i) + \mathbf{y'}_i.$$

Note that the index range of the key and value vectors is $0:i$ instead of $0:m-1$

[1] The word embedding matrix $\mathbf{W}_{\text{emb}}$ gives each input word a unique *context-independent* vector representation. This matrix is often fixed as the "LM Head" layer. However, the "LM Head" layer can very well consist of a completely independent "encoded vector-to-logit" weight mapping.

[2] Again, an in-detail explanation of the role the feed-forward layers play in transformer-based models is out-of-scope for this notebook. It is argued in Yun et. al, (2017) that feed-forward layers are crucial to map each contextual vector $\mathbf{x'}_i$ individually to the desired output space, which the *self-attention* layer does not manage to do on its own. It should be noted here, that each output token $\mathbf{x}'$ is processed by the same feed-forward layer. For more detail, the reader is advised to read the paper.