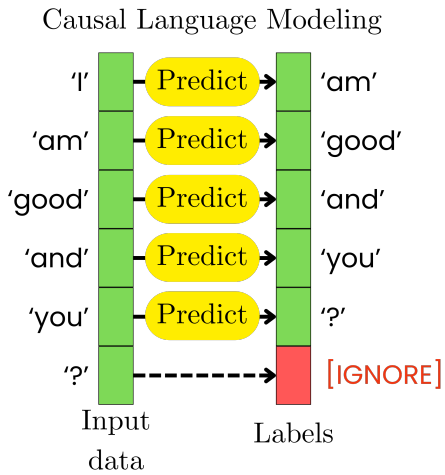
Why Do We Need Masks For the Attention Layer?

Damien Benveniste

The AiEdge

1 The Masked Attention

In the decoder, a mask is applied to the self-attention layers to enforce causality by preventing the decoder from accessing future tokens. In the encoder-decoder architecture, we train the model for causal language modeling. This means the model learns to predict the next token for every token in the output sequence. By masking the attention matrices, we ensure that the model cannot pick into the future tokens to predict the next token, which would be a form of data leakage. To predict the next token, only the past and current tokens can be used by the model.



The mask takes the form of a $N' \times N'$ matrix M (the same size as the alignment scores and the attention weights), where N' is the current number of tokens in the output sequence. It has for value:

$$M_{ij} = \begin{cases} 0 & \text{if } j \leq i, \\ -\infty & \text{if } j > i, \end{cases} \tag{1}$$

or, in matrix notation:

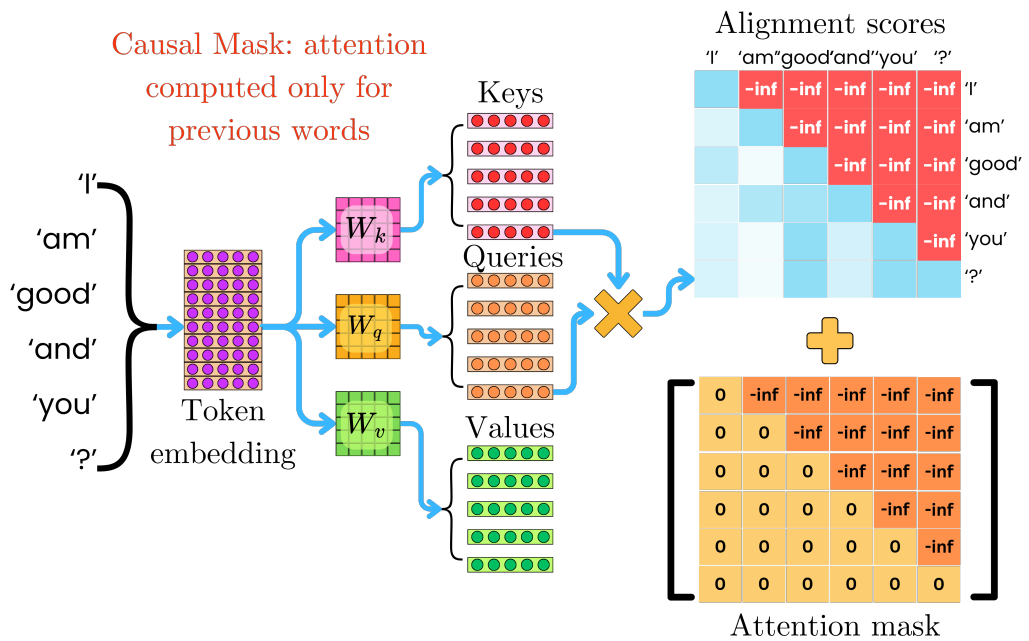
$$M = \begin{bmatrix} 0 & -\infty & \cdots & -\infty & -\infty \\ 0 & 0 & \cdots & -\infty & -\infty \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & -\infty \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{2}$$

- $j < i$ means the j-th token appears before the i-th token in the sequence (an earlier token).
- $j = i$ means the j-th token is the i-th token itself.
- $j > i$ means the j-th token appears after the i-th token (a future token).

So M "mask out" any keys $\mathbf{k}_j$ where $j > i$. This forces the model to only attend to the tokens from positions $[1, 2, \ldots, i]$, that is, the current and previous tokens, but never future ones.

The attention weights are computed by adding the mask matrix to the alignment scores:

$$A = \text{Softmax}\left(\frac{QK^\top}{\sqrt{d_{\text{head}}}} + M\right) \tag{3}$$



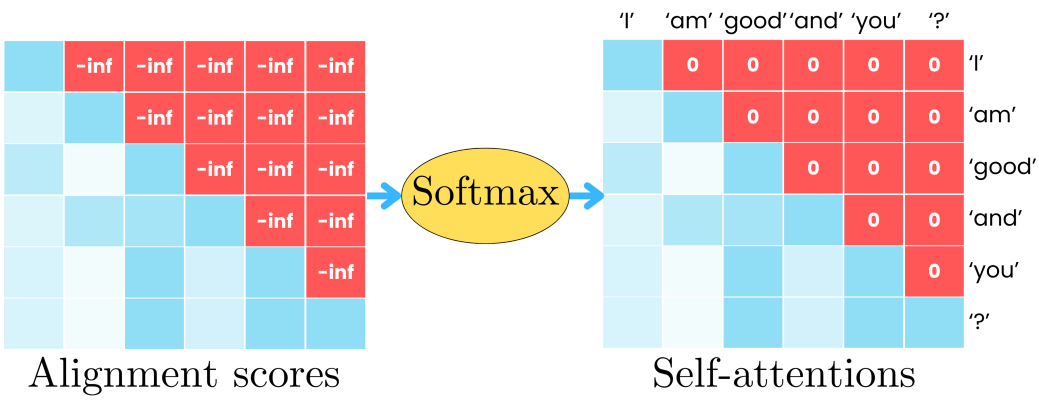
The reason we use $-\infty$ is due to the specific analytical form of the softmax function used to normalize the attention vectors. The softmax function has the following form:

$$\text{Softmax}(e_{ij}) = \frac{\exp e_{ij}}{\sum_{k=1}^{N'} \exp e_{ik}} \tag{4}$$

Because $\exp -\infty = 0$, we have:

$$\text{Softmax}(-\infty) = 0 \tag{5}$$

which prevents the attention weights from being computed for future tokens.



Furthermore, the normalization factors computed for each vector will not account for the future tokens and will behave as if the future tokens were not present:

$$\text{Softmax}(e_{\text{good, am}}) = \frac{\exp e_{\text{good, am}}}{\exp e_{\text{good, I}} + \exp e_{\text{good, am}} + \exp e_{\text{good, good}} + 0 + 0 + 0} \tag{6}$$

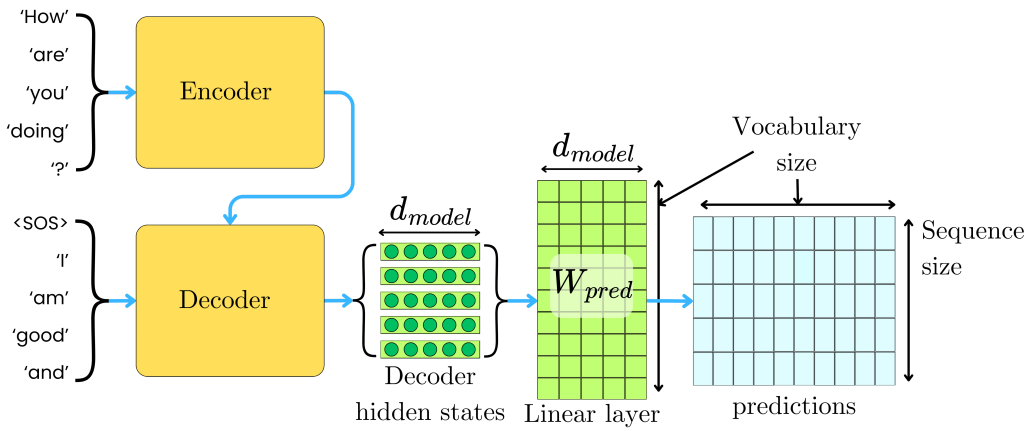
2 The Decoding Process

In the transformer, the prediction head is a classifier that uses the decoder's output hidden states as features and estimates the probabilities that the next token is one of the tokens in the token vocabulary. It is a linear layer W_{pred} that maps vectors from hidden size d_{model} to the vocabulary size, followed by a softmax transformation to convert from logits to probabilities. If $\mathbf{s}_i$ is an incoming decoder hidden state:

$$\mathbf{logit}_i = W_{\text{pred}}\mathbf{s}_i \qquad \text{Project to logits} \tag{7}$$

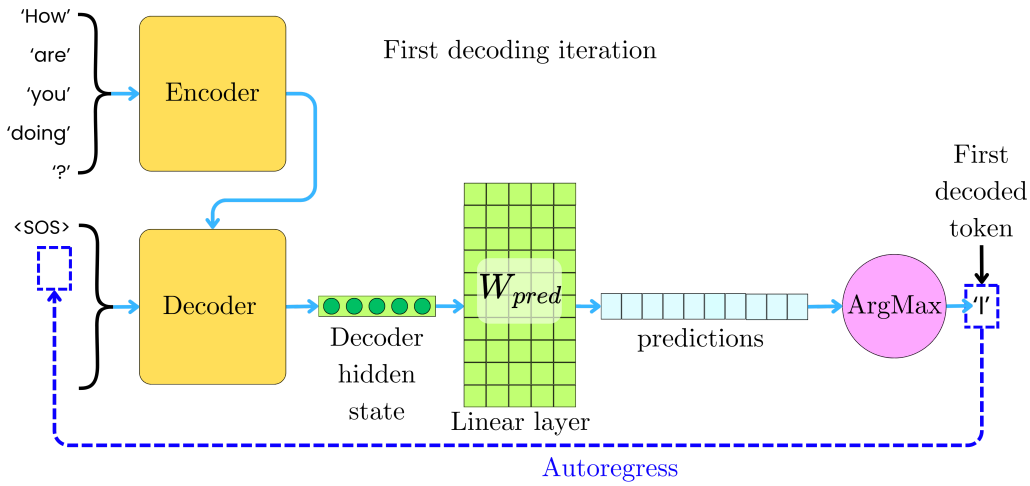
$$\mathbf{p}_i = \text{Softmax}\left(\mathbf{logit}_i\right) \qquad \text{Convert to probabilities} \tag{8}$$

Each token in the output sequence will correspond to an output hidden state, which in turn corresponds to a prediction vector.

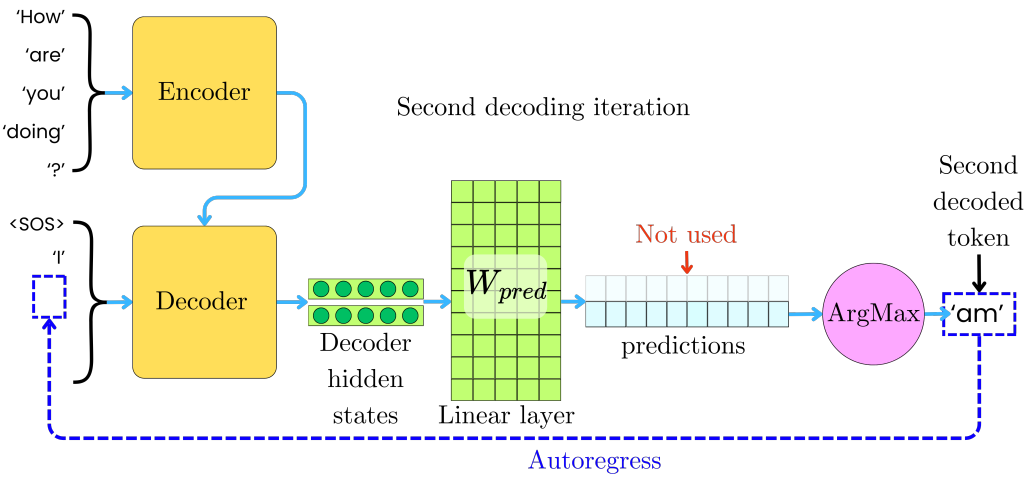


At the beginning of the decoding process, the decoder is initialized with the start of sequence token <SOS>, and its corresponding prediction vector is used to predict the first token in the output sequence. Since the softmax transformation is monotonic, we can directly choose the token with the maximum predicting logit without the need to convert to probabilities:

$$\text{First token} = \arg\max_{\text{Vocab}} \textbf{logit}_1 \tag{9}$$



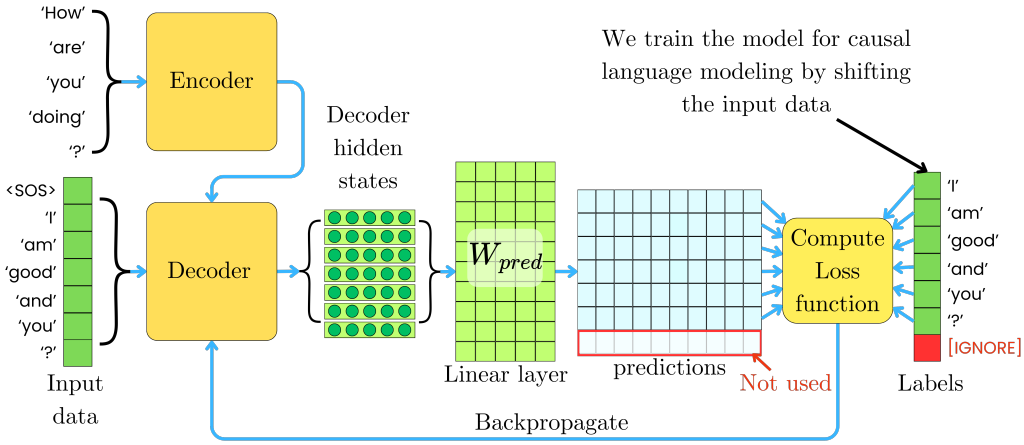
This new predicted token is appended to the output sequence and we continue the decoding process by autoregressing iteratively. During the second iteration, since we only need to predict the next token, we use the prediction vector corresponding to the last token.



We continue this process until the model predicts the end of sequence token <EOS>. During inference, we typically only use the last prediction vector in the sequence, but it is important to ensure that the prediction vectors corresponding to the previous tokens remain the same at each iteration. For example, the logit vector $\textbf{logit}_1$ is going to be re-predicted at each iteration, and there is no reason for it to change during the decoding process. That is main reason for the attention masks in the attention layers, as they ensure that the model cannot use future tokens to infer on the first token in the sequence.

3 Training For Causal Language Modeling

The model is trained during training to predict the next token. The labels used during training are the tokens of the output sequence shifted by one.

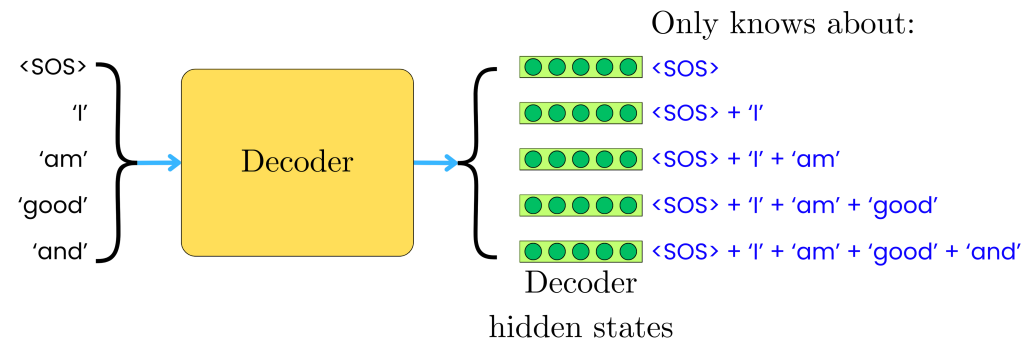


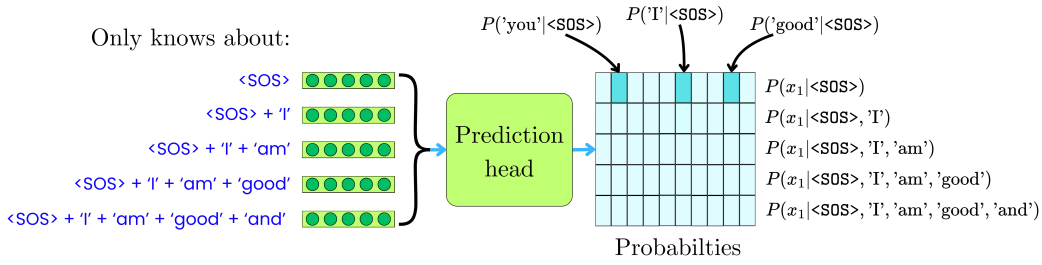
Because the model can output predictions for all the tokens at once, we can directly compute the loss function for all the tokens in the sequence. The probability for the next token x_t knowing

the previous token $[x_0, x_1, \ldots, x_{t-1}]$ is:

$$P(x_t|x_0, x_1, \ldots, x_{t-1}) \stackrel{\text{def}}{=} P(x_t|x_{<t}) \tag{10}$$

Once again, the masked attention allows to estimate those probabilities because all the hidden states in the model on only account for the tokens up to their position in the sequence.





Because we are training a classifier, we use the crossentropy loss function to train the model and average the loss across all the tokens in each sample sequence in the training data:

$$\mathcal{L} = -\frac{1}{T}\sum_{t=1}^{T} P(x_t|x_{<t}) \tag{11}$$

where x_t and $x_{<t}$ are the ground truth tokens found as labels in the training data.