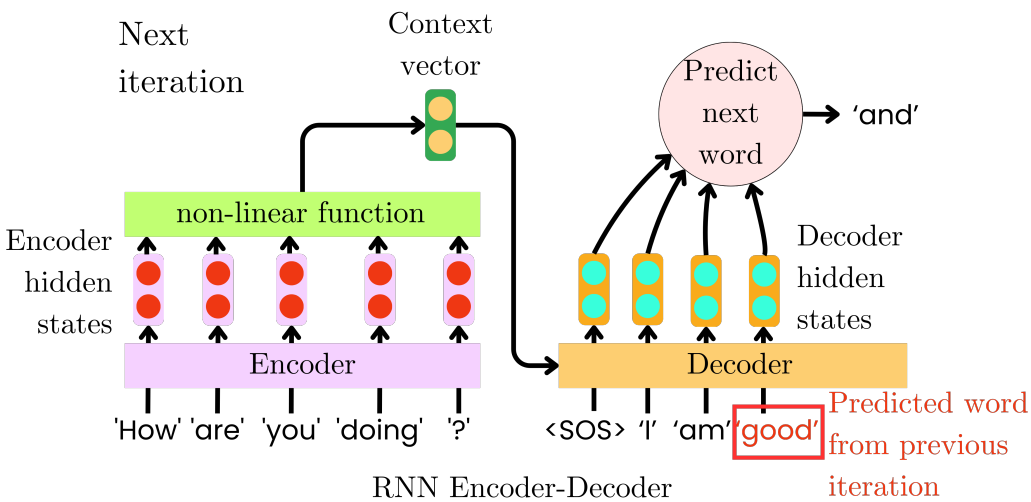
The First Sequencet-to-Sequence Model: The RNN Encoder-Decoder Architecture

Damien Benveniste
The AiEdge



The first sequence-to-sequence model was introduced in 2014 by the seminal paper by Sutskever et al.[1] with the RNN encoder-decoder framework (the same paper introducing the GRU layer). The RNN encoder-decoder architecture was developed as an autoregressive model to predict a new text sequence from an input text sequence. There are three main components to the model:

- The encoder: it compresses the input sequence into a fixed-dimensional context vector that captures its semantic and syntactic information.
- The decoder: it generates the output sequence step-by-step using the context vector from the encoder
- The prediction head: it converts the decoder's hidden state into a probability distribution over the target vocabulary to predict the next word.

The decoder decodes the output text in an autoregressive manner. This means that the decoding process is an iterative process where each predicted word at the current iteration is appended at the next iteration to the text used as input to the decoder.

1. The Encoder

The encoder is composed of a word embedding and multiple layers of recurrent units. Let us choose GRUs as it was done in the original paper. For each word in the input

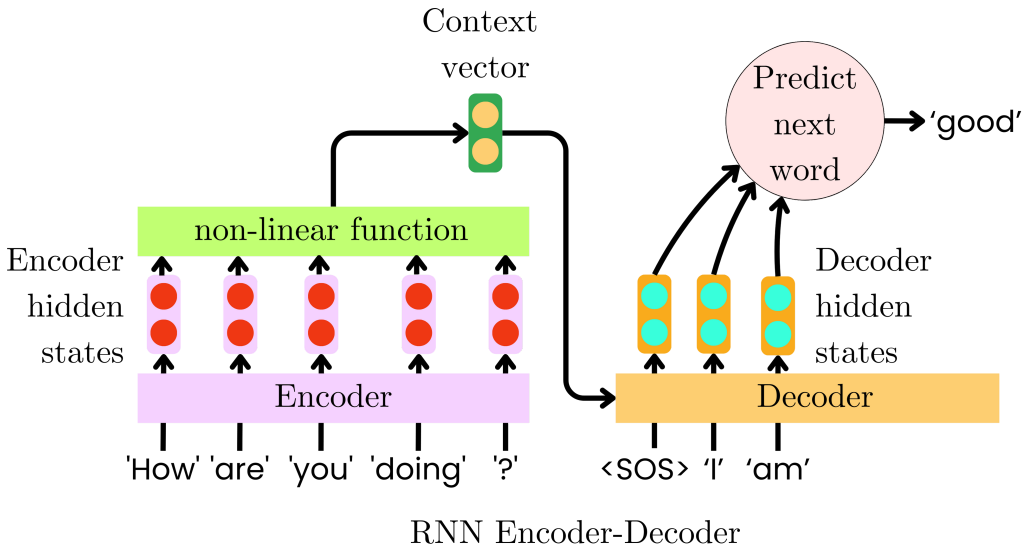


Figure 1: In the RNN encoder-decoder architecture, the encoder generates a context vector that represents a summary of the information contained in the input sequence, and it is used as input to the decoder. The decoder then decodes step by step the different tokens by using as input the tokens predicted in the previous iterations.

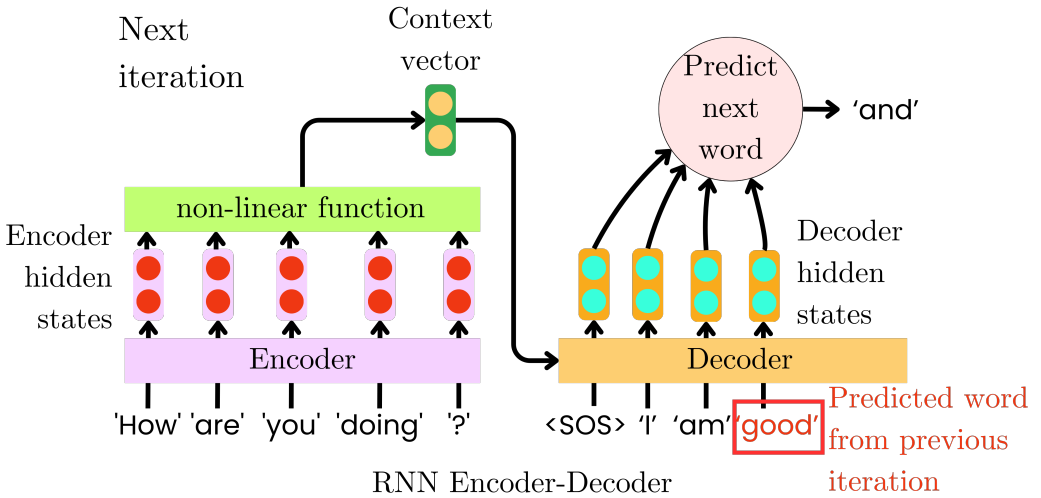


Figure 2: Once a token is predicted by the prediction head, it is appended to the output sequence and used as input to the decoder in the next iteration.

sequence, its vector representation $\mathbf{x}_t$ is used as input to the GRU network, which outputs its corresponding hidden states. We call $\mathbf{h}_t^i$ the vector generated by the ith GRU layer at the time-step t.

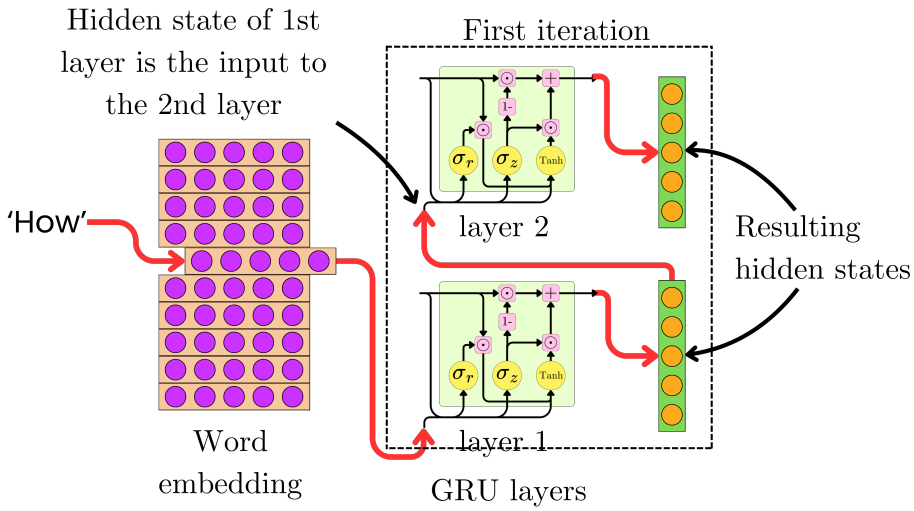


Figure 3: During the first iteration, the vector representation of the first word in the input sequence is used as input to the RNN. The RNN yields one hidden state per layer.

The inputs to the first layer are the vector representation $\mathbf{x}_t$ and the hidden state $\mathbf{h}^1_{t-1}$ of the previous iteration. The inputs to the subsequent layers i are the hidden state $\mathbf{h}^{i-1}_t$ of the previous layer and the hidden state $\mathbf{h}^i_{t-1}$ of the previous iteration.

$$\mathbf{h}^i_t = \begin{cases} \text{GRU}^i_{\text{enc}}(\mathbf{x}_t, \mathbf{h}^i_{t-1}) & \text{if } i = 1, \\ \text{GRU}^i_{\text{enc}}(\mathbf{h}^{i-1}_t, \mathbf{h}^i_{t-1}) & \text{if } i > 1, \end{cases} . \tag{1}$$

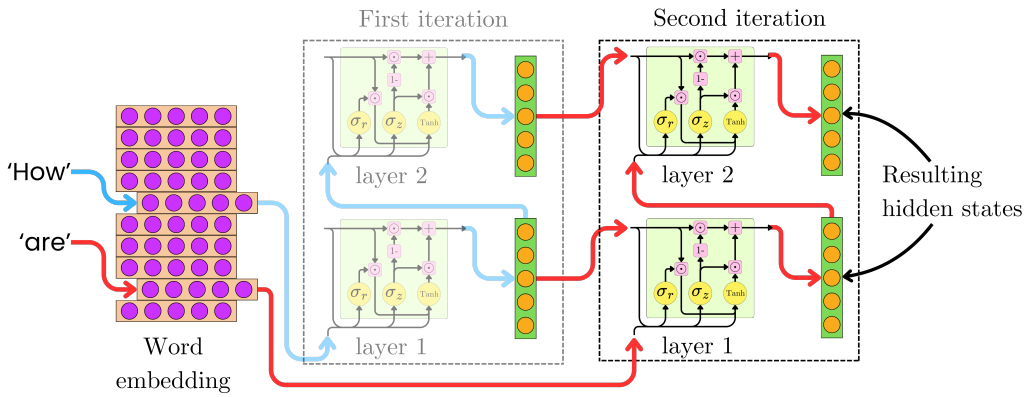


Figure 4: In the second iteration, the vector representation of the second word in the input sequence is used as input to the RNN as well as the hidden states of the previous iteration.

At each iteration, the resulting hidden states encode the full input sequence up to the

word related to the current iteration:

$$\mathbf{h}_t^i = \text{RNN}_{\text{enc}}^i(\mathbf{x}_1, \ldots, \mathbf{x}_t) \tag{2}$$

Once we iterated through all the words within the input sequence, we obtain two sets of vectors: the hidden states $[\mathbf{h}_1^L, \ldots, \mathbf{h}_T^L]$ for each iteration t of the last layer L, and the hidden states $[\mathbf{h}_T^1, \ldots, \mathbf{h}_T^L]$ of the last iteration T for all the layers i. We often refer to the former set of vectors as the "output states" and to the latter as the "hidden states."

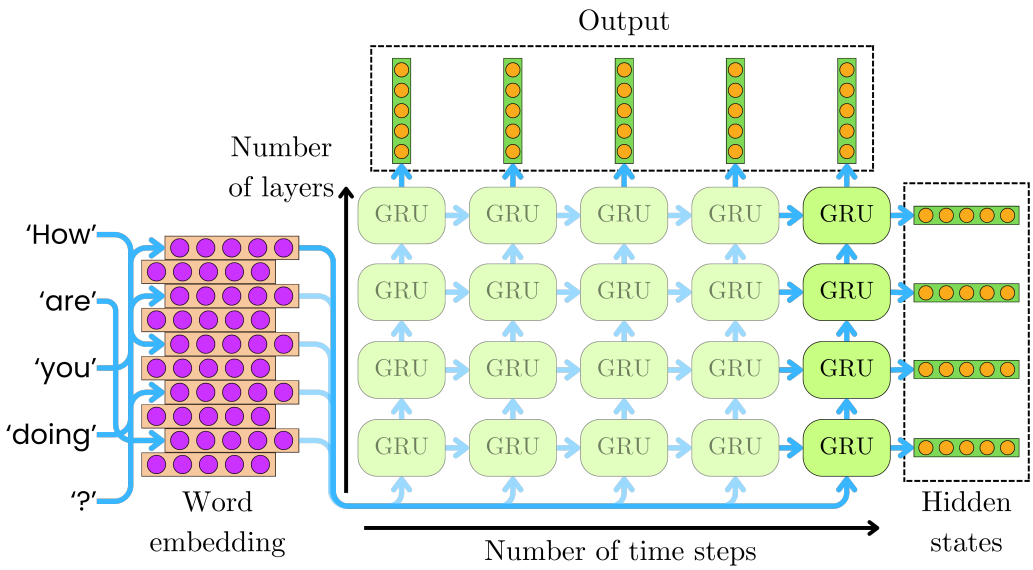


Figure 5: We can extract two sets of hidden state vectors from the encoder: the hidden states $[\mathbf{h}_1^L, \ldots, \mathbf{h}_T^L]$ for each iteration t of the last layer L, and the hidden states $[\mathbf{h}_T^1, \ldots, \mathbf{h}_T^L]$ of the last iteration T for all the layers i.

In the context of the RNN encoder-decoder architecture, we are going to use the ones referred as the "hidden states". Those hidden states encode the full input sequence and are used as the context vector for the decoder to generate a response.

2. The Decoder

The decoder is very similar to the encoder and has the same architecture. We have another word embedding, and it has multiple layers of recurrent units. The main difference is that we initialize its input text by the start of sequence token `<SOS>`, and the input hidden states $\mathbf{s}_0^i$ to the GRU network are initialized by the hidden states $\mathbf{h}_T^i$ generated by the encoder:

$$\mathbf{s}_1^i = \text{GRU}_{\text{dec}}^i(\mathbf{y}_1, \mathbf{h}_T^i) \tag{3}$$

The resulting output vector $\mathbf{s}_1^L$ is used by the predicting head to predict the first word of the output sequence. In our example, the word `'I'`

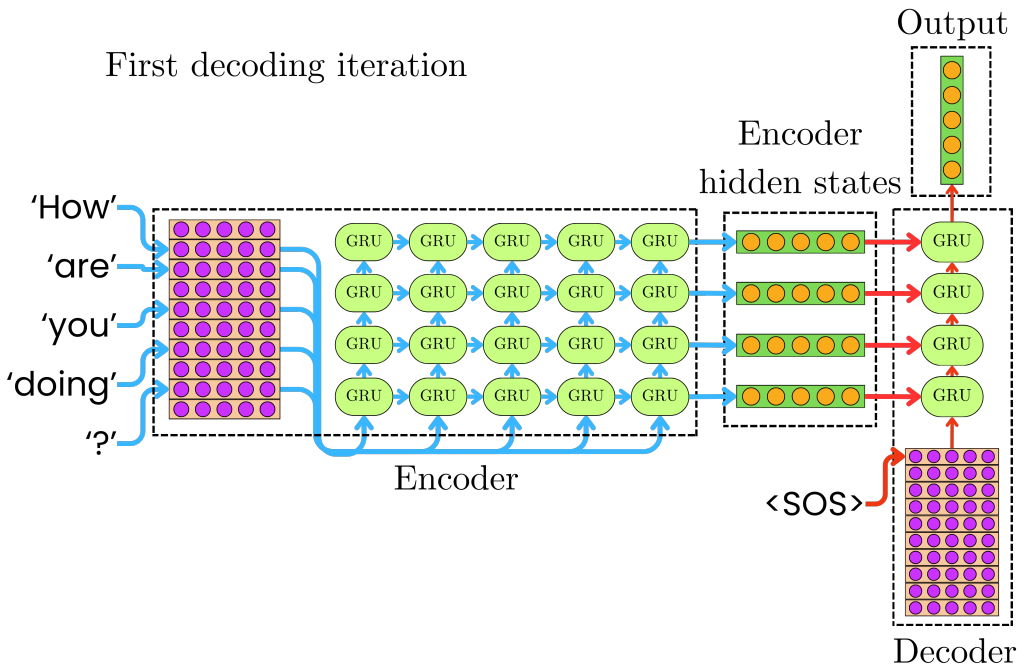


Figure 6: At the first iteration of the decoding process, the decoder takes as input the encoder's hidden states $\mathbf{h}_T^i$ and the vector representation of the `<SOS>` token.

This decoded word is then appended to the input tokens to the decoder. As for the encoder, at each iteration t, the GRU network uses as inputs the vector representation $\mathbf{y}_t$ of the latest decoded token and the hidden state $\mathbf{s}_{t-1}^i$ of the previous iteration:

$$\mathbf{s}_t^i = \begin{cases} \text{GRU}_{\text{dec}}^i(\mathbf{y}_t, \mathbf{s}_{t-1}^i) & \text{if } i = 1, \\ \text{GRU}_{\text{dec}}^i(\mathbf{s}_t^{i-1}, \mathbf{s}_{t-1}^i) & \text{if } i > 1, \end{cases} . \tag{4}$$

The resulting vector output $\mathbf{s}_t^T$ used by the prediction head encodes the whole output sequence up to the word associated with the current decoding iteration and its interaction with the input sequence through the encoder hidden states:

$$\mathbf{s}_t^L = \text{RNN}_{\text{dec}}^L(\mathbf{y}_1, \ldots, \mathbf{y}_t, \mathbf{h}_1^L, \ldots, \mathbf{h}_T^L) \tag{5}$$

We iterate this decoding process until the model emits an end-of-sequence token `<EOS>`, in which case we interrupt the decoding process and return the output sequence.

3. The Prediction Head

The prediction head is just a classifier that outputs a probability (or a logit) for each of the words in the vocabulary. At each time step, the decoder produces an output vector $\mathbf{s}_t^L$ with a hidden size dimension. To predict the next word, this vector is mapped through a

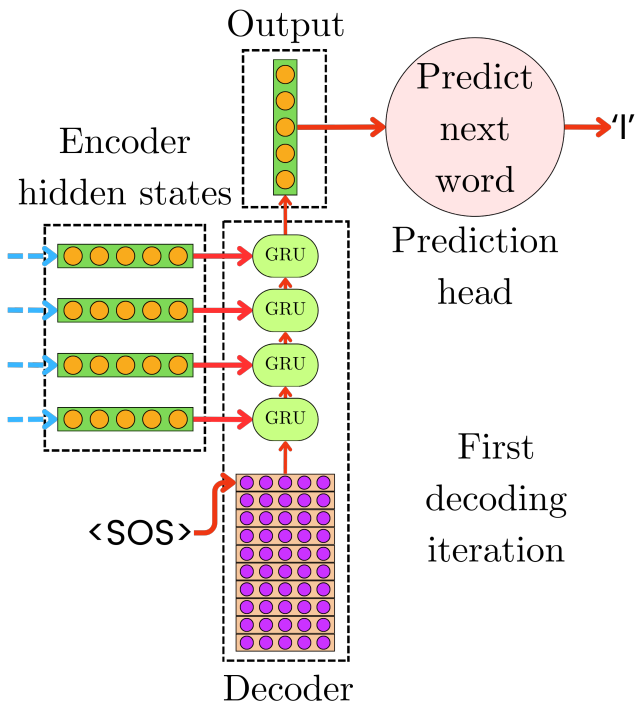


Figure 7: The prediction head uses the decoder output to predict the next word.

linear layer W_p into a vector of logits with the size of the vocabulary. The linear layer has an input dimension of hidden size and an output dimension of vocabulary size. To obtain probabilities, we could simply apply the Softmax transformation, but because the Softmax transformation is monotonic, we can directly extract the most likely word by choosing the one with the highest logit:

$$\text{Next word} = \arg\max_{\text{vocab}} W_p \mathbf{s}_t^L \tag{6}$$

Because it is a classifier, this model is trained using the cross-entropy computed on the target sequence given the source sequence:

$$\mathcal{L} = -\sum_{i=1}^{T'} P(y_t^* | y_1^*, \ldots, y_{t-1}^*, \mathbf{h}_T^1, \ldots, \mathbf{h}_T^L) \tag{7}$$

where T' is the length of the target (output) sequence and y_t^* is the ground-truth token at step t in the training data. In practice, we use the Softmax transformation to estimate the probabilities. If $\mathbf{l}_t = W_p \mathbf{s}_t^L$ is the logit vector, we obtain the probability vector $\mathbf{p}_t$ by applying the Softmax transformation:

$$\mathbf{p}_t = \text{Softmax}(\mathbf{l}_t) = \frac{e^{\mathbf{l}_t}}{\sum_{i=1}^{|V|} e^{l_t^i}} \tag{8}$$

where $|V|$ is the dimension of the vocabulary. To train our model, we just need to pick the probability corresponding to the token y_t^* and compute the loss function.

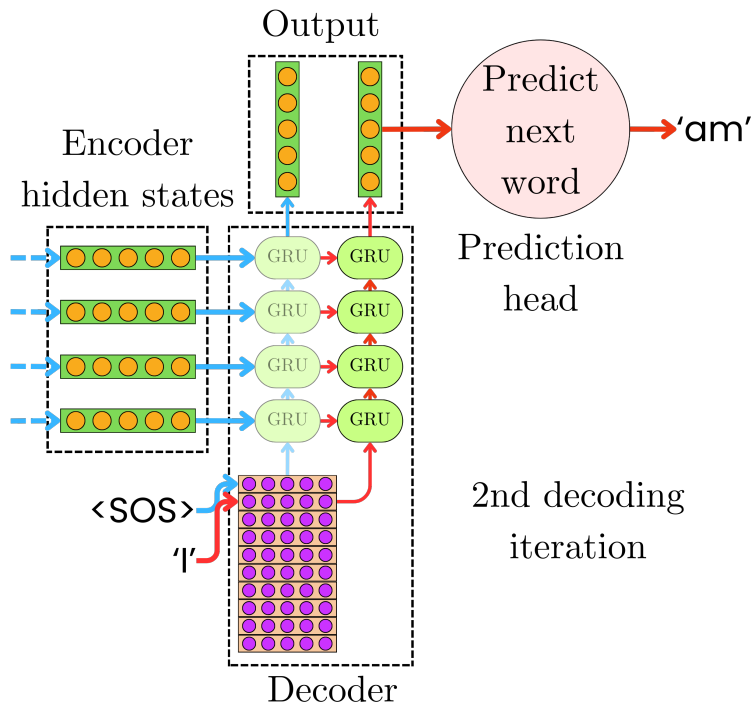


Figure 8: In the second iteration, we use the predicted word of the previous iteration to continue the decoding process.

References

[1] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

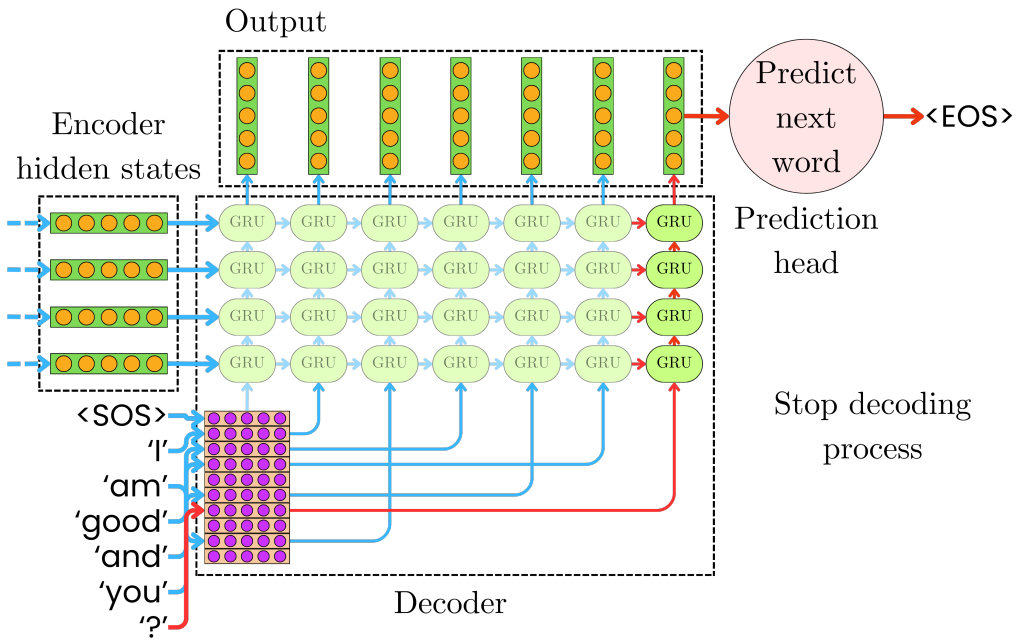


Figure 9: The decoding process stops when the model predicts that the next word is the <EOS> token.

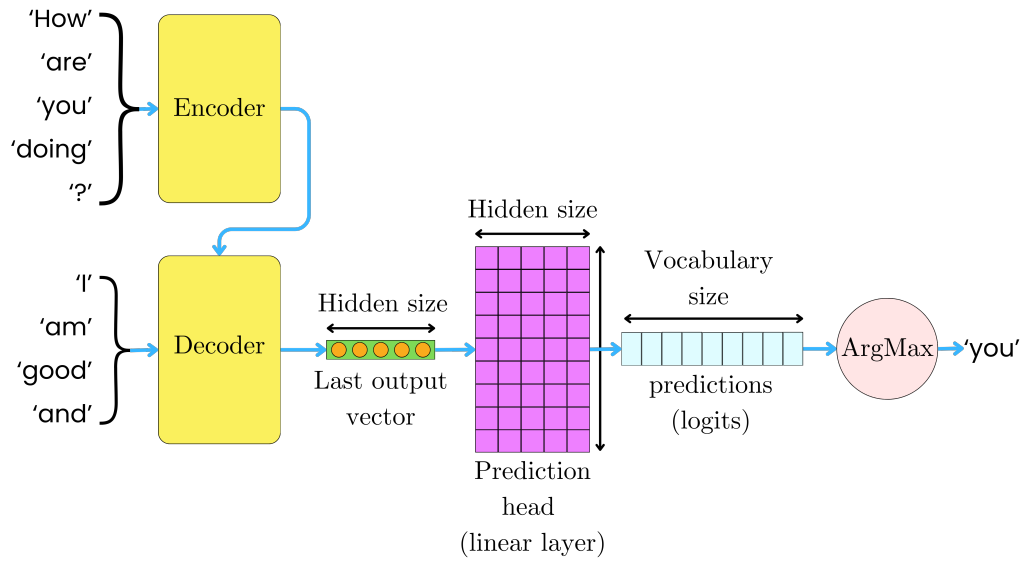


Figure 10: The prediction head is mostly a linear layer mapping from the decoder output to the logit prediction vector. We then choose as the next word, the word with the highest logit value.