UNC CHARLOTTE

The WILLIAM STATES LEE COLLEGE *of* ENGINEERING

Introduction to ML
Lecture 9: Sequence to Sequence and Machine Translation

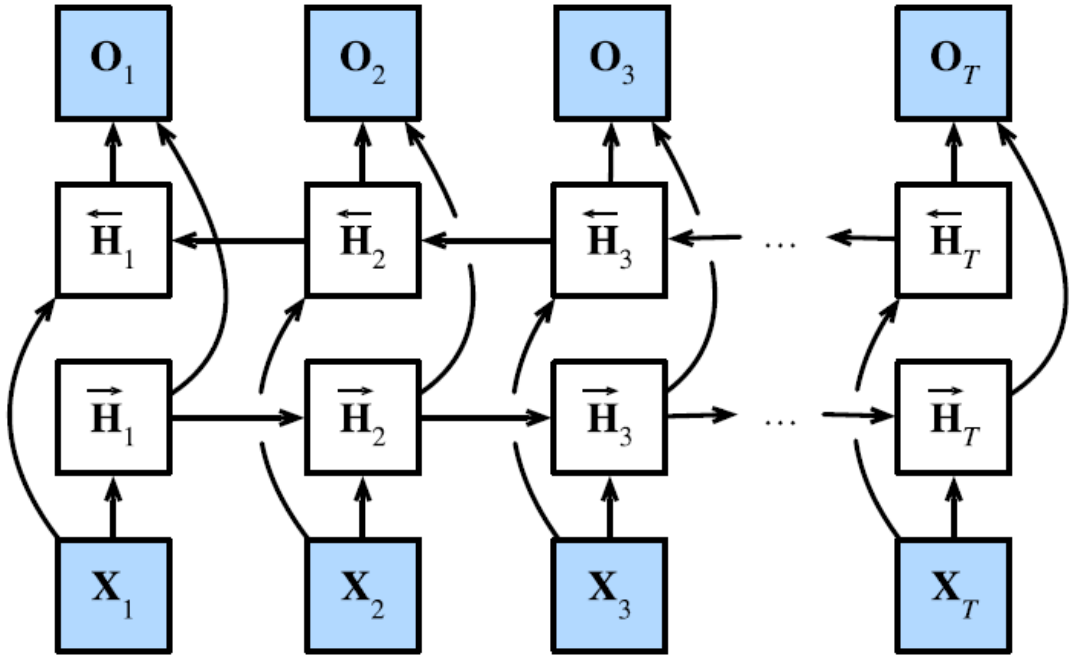
Hamed Tabkhi
Department of Electrical and Computer Engineering,
University of North Carolina Charlotte (UNCC)
htabkhiv@uncc.edu

Bi-directional RNNs

- So far, our working example of a sequence learning task has been language modeling, where we aim to predict the next token given all previous tokens in a sequence.
- In this scenario, we wish only to condition upon the leftward context, and thus the unidirectional chaining of a standard RNN seems appropriate.
- However, there are many other sequence learning tasks contexts where it's perfectly fine to condition the prediction at every time step on both the leftward and the rightward context. Consider, for example, part of speech detection.
- Why shouldn't we take the context in both directions into account when assessing the part of speech associated with a given word?

Bi-directional RNNs

- Fortunately, a simple technique transforms any unidirectional RNN into a bidirectional RNN.
- We simply implement two unidirectional RNN layers chained together in opposite directions and acting on the same input.
- For the first RNN layer, the first input is $\mathbf{x}_1$ and the last input is $\mathbf{x}_T$, but for the second RNN layer, the first input is $\mathbf{x}_T$ and the last input is $\mathbf{x}_1$.
- To produce the output of this bidirectional RNN layer, we simply concatenate together the corresponding outputs of the two-underlying unidirectional
- RNN layers.



UNC CHARLOTTE

Machine Translation

- Among the major breakthroughs that prompted widespread interest in modern RNNs was a major advance in the applied field of statistical *machine translation*.
- Here, the model is presented with a sentence in one language and must predict the corresponding sentence in another language.
- Note that here the sentences may be of different lengths, and that corresponding words in the two sentences may not occur in the same order, owing to differences in the two language's grammatical structure.
- Many problems have this flavor of mapping between two such "unaligned" sequences.
- **Examples include mapping from dialog prompts to replies or from questions to answers. Broadly, such problems are called *sequence-to-sequence* (seq2seq) problems**

Machine Translation

- For decades, statistical formulations of translation between languages had been popular (Brown *et al.*, 1990, Brown *et al.*, 1988), even before researchers got neural network approaches working (methods often lumped together under the term *neural machine translation*).
- Unlike the language modeling that we, here each example consists of two separate text sequences, one in the source language and another (the translation) in the target language.

- To begin, we download an English-French dataset that consists of bilingual sentence pairs from the Tatoeba Project.
- Each line in the dataset is a tab-delimited pair consisting of an English text sequence and the translated French text sequence.
- Note that each text sequence can be just one sentence, or a paragraph of multiple sentences.
- In this machine translation problem where English is translated into French, English is called the *source language* and French is called the *target language*.

Dataset Pre-processing

- After downloading the dataset, we proceed with several preprocessing steps for the raw text data.
- For instance, we replace non-breaking space with space, convert uppercase letters to lowercase ones, and insert space between words and punctuation marks.

Tokenization

- Unlike the character-level tokenization, for machine translation we prefer word-level tokenization here (today's state-of-the-art models use more complex tokenization techniques).
- The following _tokenize method tokenizes the first max_examples text sequence pairs, where each token is either a word or a punctuation mark.
- We append the special "<eos>" token to the end of every sequence to indicate the end of the sequence.
- When a model is predicting by generating a sequence token after token, the generation of the "<eos>" token can suggest that the output sequence is complete. In the end, the method below returns two lists of token lists: src and tgt.
- Specifically, src[i] is a list of tokens from the text sequence in the source language (English here) and tgt[i] is that in the target language (French here).

Loading Sequences of Fixed Length

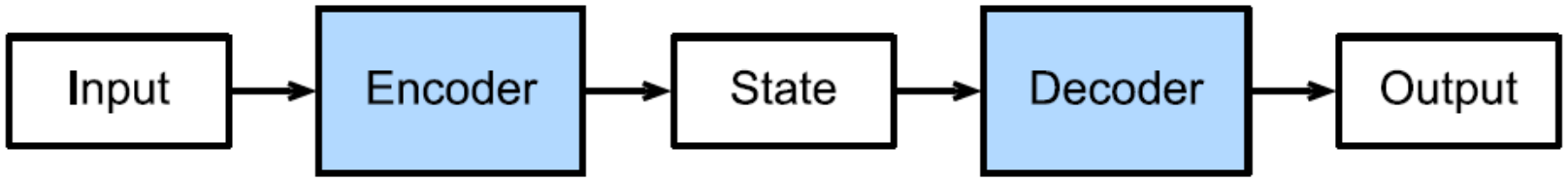
- In machine translation, each sample is a pair of source and target text sequences, where the two text sequences may have different lengths.
- For computational efficiency, we can still process a minibatch of text sequences at one time *By truncation* and *padding*.
- Suppose that every sequence in the same minibatch should have the same length num_steps. If a text sequence has fewer than num_steps tokens, we will keep appending the special “<pad>” token to its end until its length reaches num_steps (similar to zero padding in computer vision problems)
- Otherwise, we will truncate the text sequence by only taking its first num_steps tokens and discarding the remaining.
- In this way, every text sequence will have the same length to be loaded in minibatches of the same shape. Besides, we also record length of the source sequence excluding padding tokens. This information will be needed by some models.

- The length of the sequence will define the complexity of the model!!! (What is the length of sequence in Chat GPT-4????)

Encoder-Decoder Architecture

- In general, seq2seq problems like machine translation, inputs and outputs are of varying lengths that are unaligned.
- The standard approach to handling this sort of data is to design an *encoder-decoder* architecture, consisting of two major components:

1. an *encoder* that takes a variable-length sequence as input
2. a *decoder* that acts as a conditional language model, taking in the encoded input and the leftwards context of the target sequence and predicting the subsequent token in the target sequence.

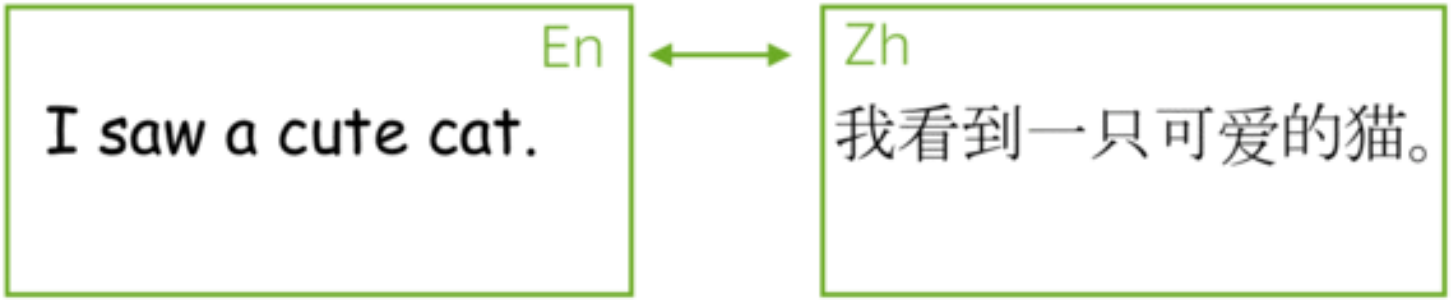


Seq2Seq for Machine Translation

- Let's take machine translation from English to French as an example.
- Given an input sequence in English: "They", "are", "watching", ".", this encoder-decoder architecture first encodes the variable-length input into a state, then decodes the state to generate the translated sequence, token by token, as output: "Ils", "regardent", ".".
- The decoder model, consisting of a separate RNN, will predict each successive target token given both the input sequence and the preceding tokens in the output.

Note that if we ignore the encoder, the decoder in a seq2seq architecture behaves just like a normal language model!

Let's review sequence to sequence modeling again in context of language translation



Hu

$$y^* = \arg\max_y p(y|x)$$

The "probability" is intuitive and is given by a human translator's expertise

model parameters

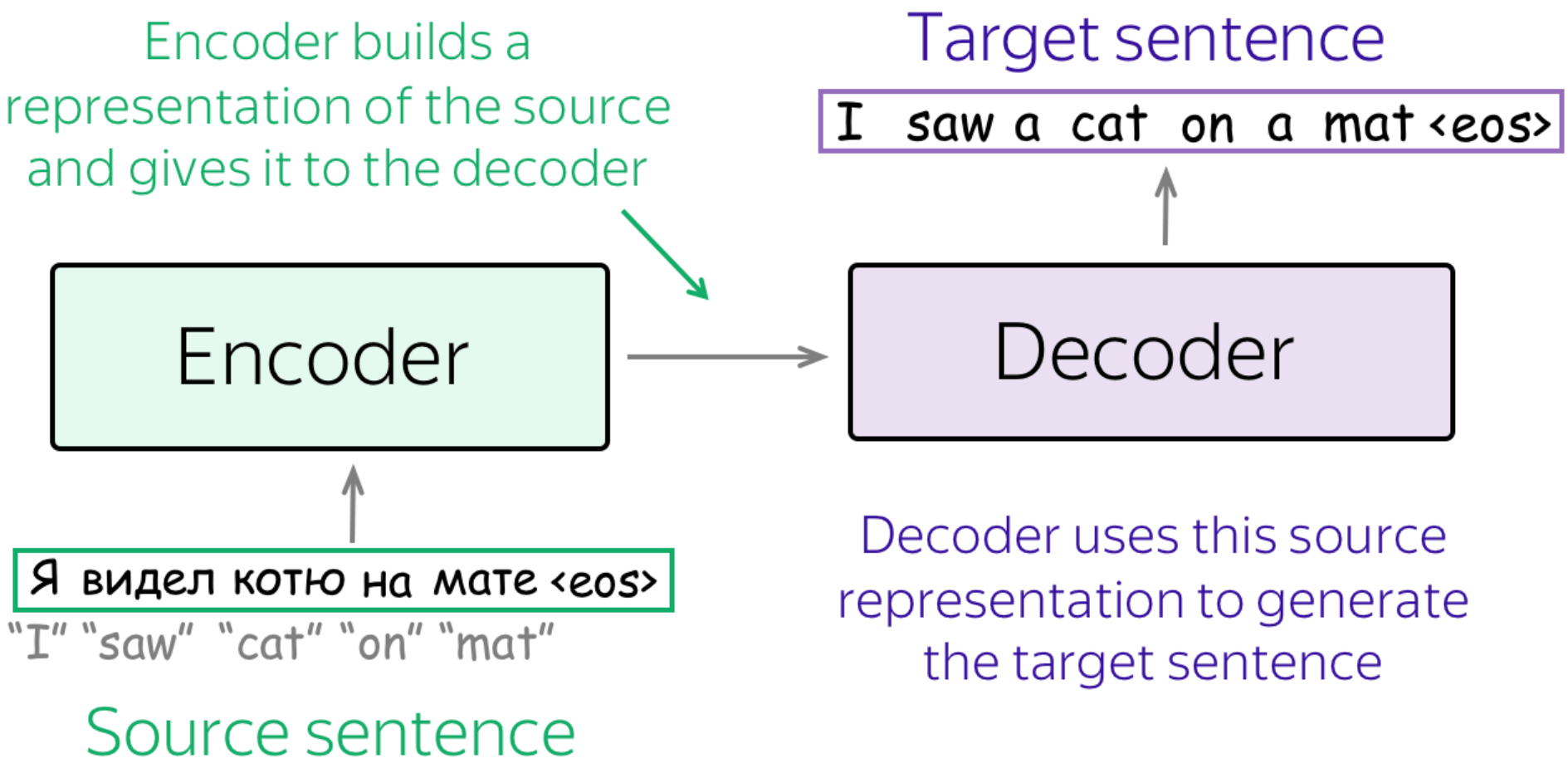
$$y' = \arg\max_y p(y|x,\theta)$$

Questions we need to answer

- modeling
  How does the model for $p(y|x,\theta)$ look like?
- learning
  How to find θ?
- search
  How to find the argmax?

•Argmax is an operation that finds the argument that gives the maximum value from a target function.

UNC CHARLOTTE

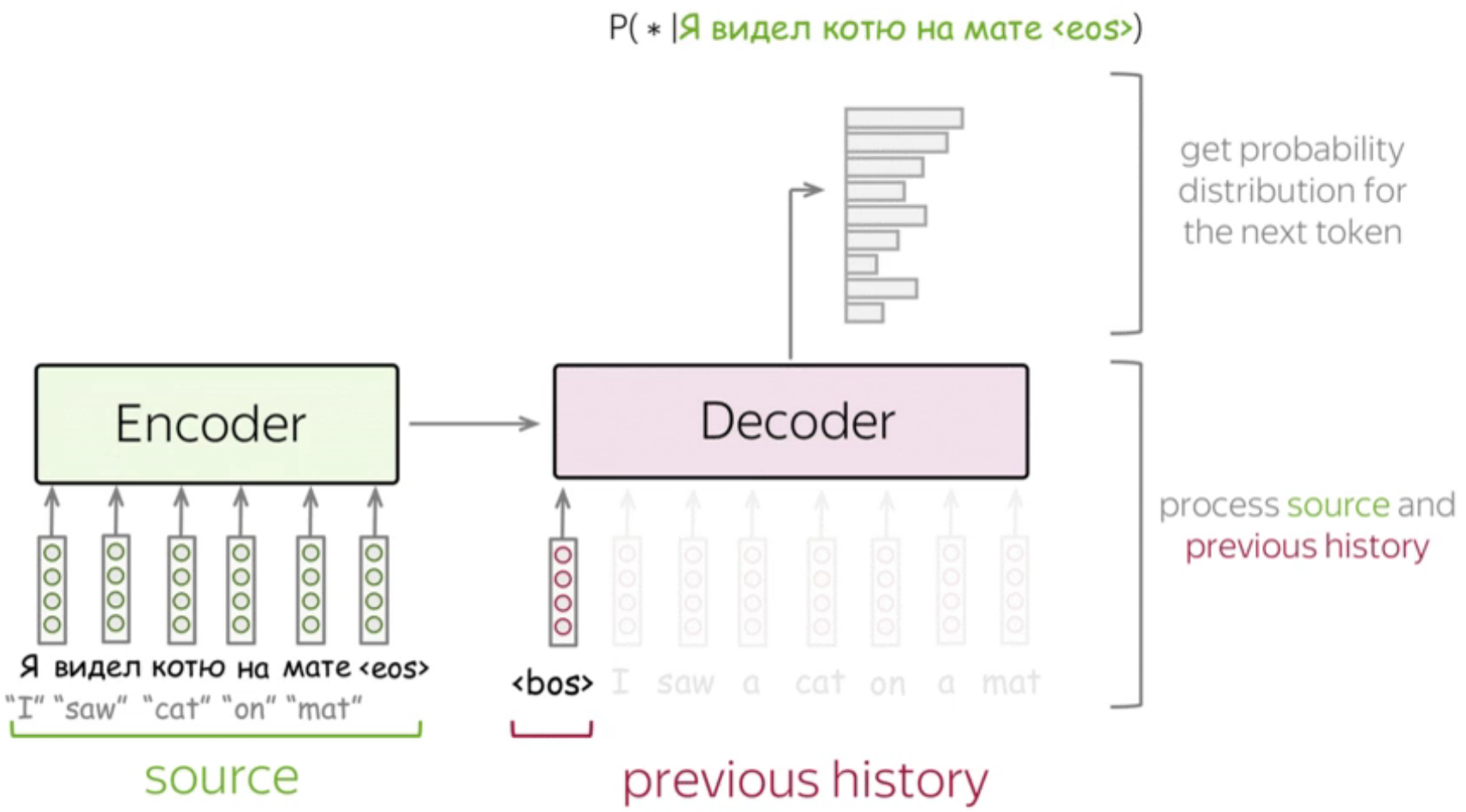


Language Models: $P(y_1, y_2, \ldots, y_n) = \prod_{t=1}^{n} p(y_t | y_{<t})$

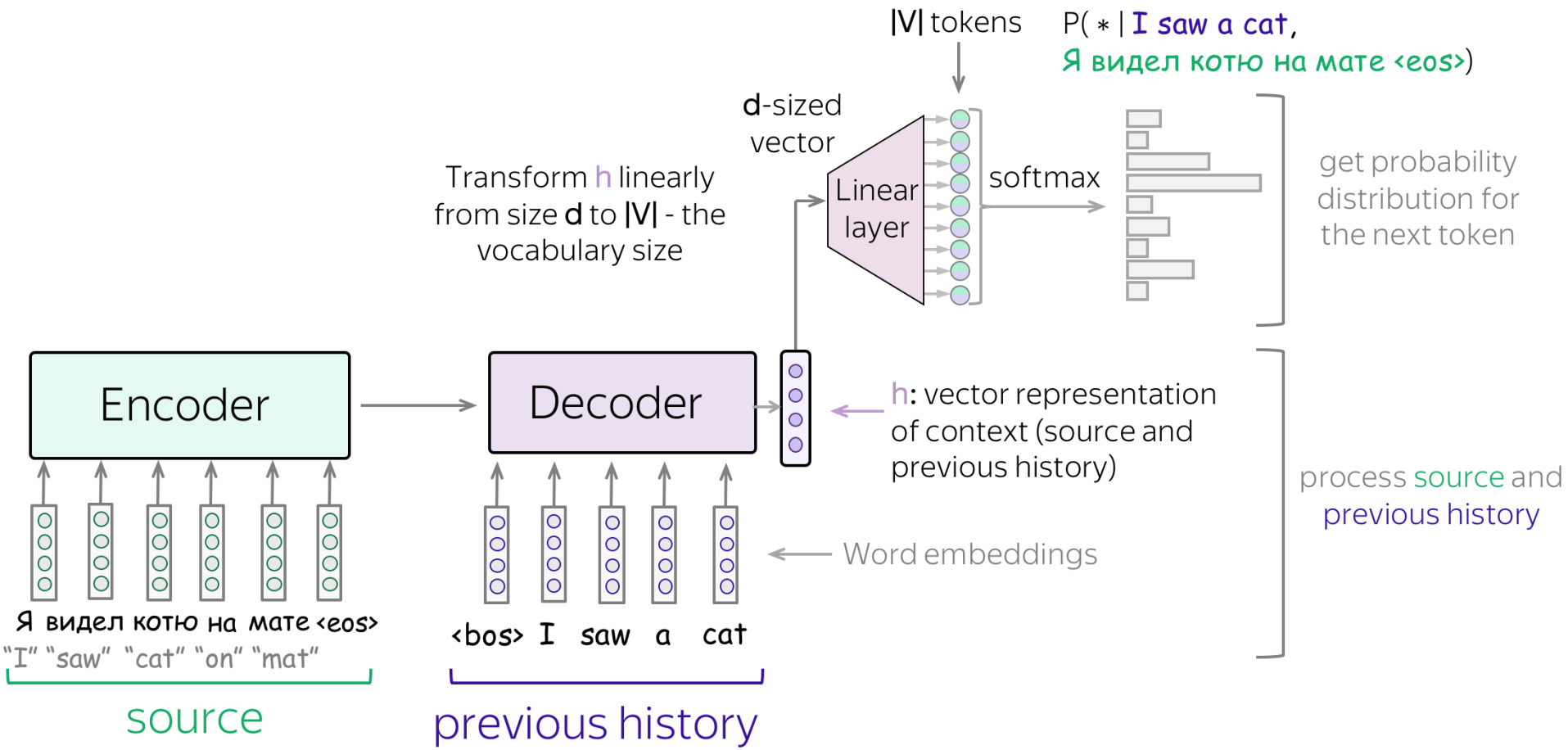
Conditional Language Models: $P(y_1, y_2, \ldots, y_n, | x) = \prod_{t=1}^{n} p(y_t | y_{<t}, x)$

condition on source x

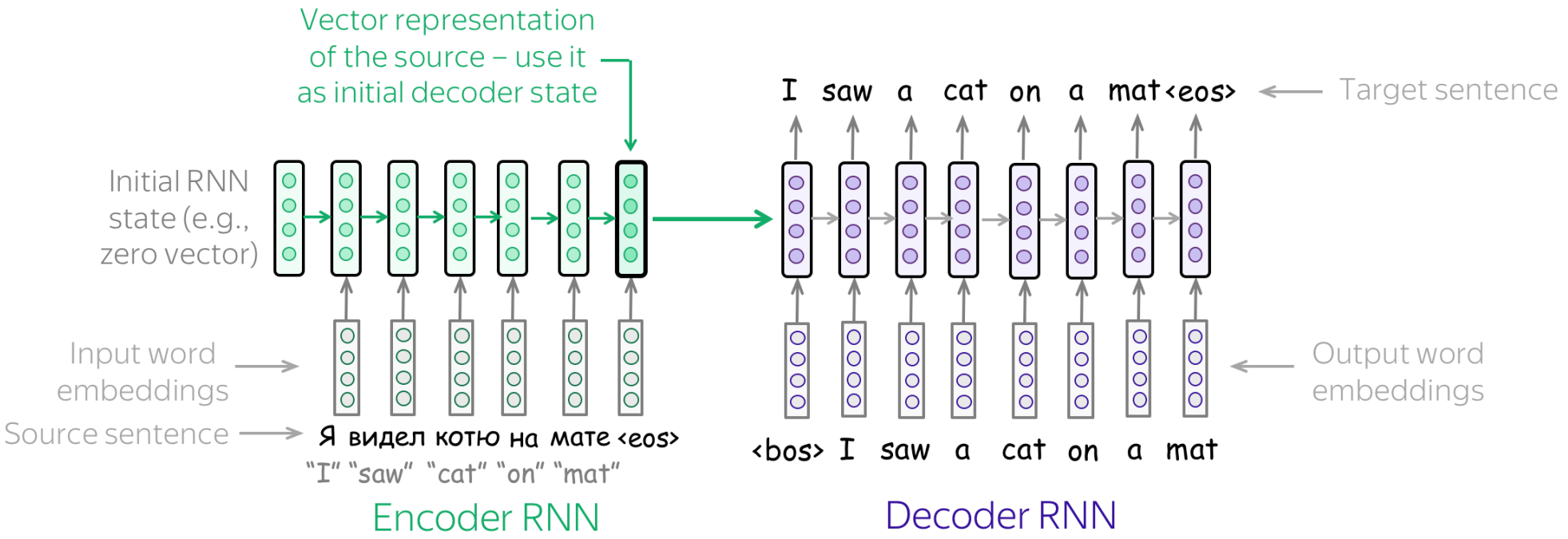
UNC CHARLOTTE



UNC CHARLOTTE



UNC CHARLOTTE



UNC CHARLOTTE

Source sequence:

Я видел котю на мате <eos>

"I" "saw" "cat" "on" "mat"

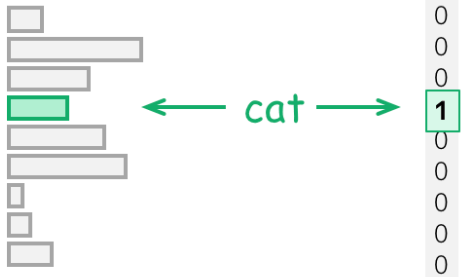
Target sequence: ← one training example

I saw a cat on a mat <eos> ← one step for this example

previous tokens

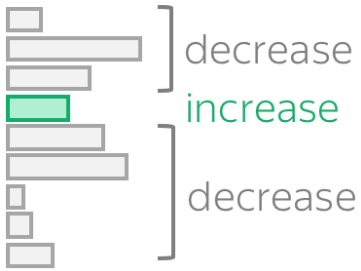
we want the model to predict this

Model prediction: p(* |I saw a, Я ... <eos>)



Target

Loss = -log (p(cat)) → min



UNC CHARLOTTE

Encoder: read source

we are here

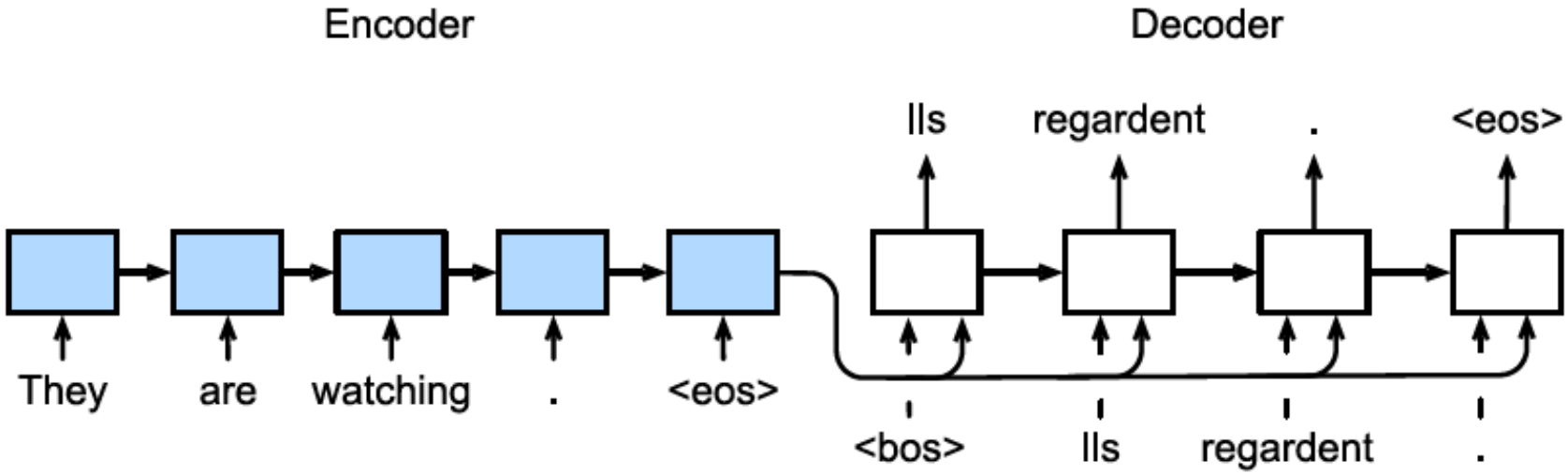
Source: Я видел котю на мате <eos>

"I" "saw" "cat" "on" "mat"

Target: I saw a cat on a mat <eos>

Teacher Forcing

- While running the encoder on the input sequence is relatively straightforward, how to handle the input and output of the decoder requires more care.
- The most common approach is sometimes called *teacher forcing*.
- Here, the original target sequence (token labels) is fed into the decoder as input.

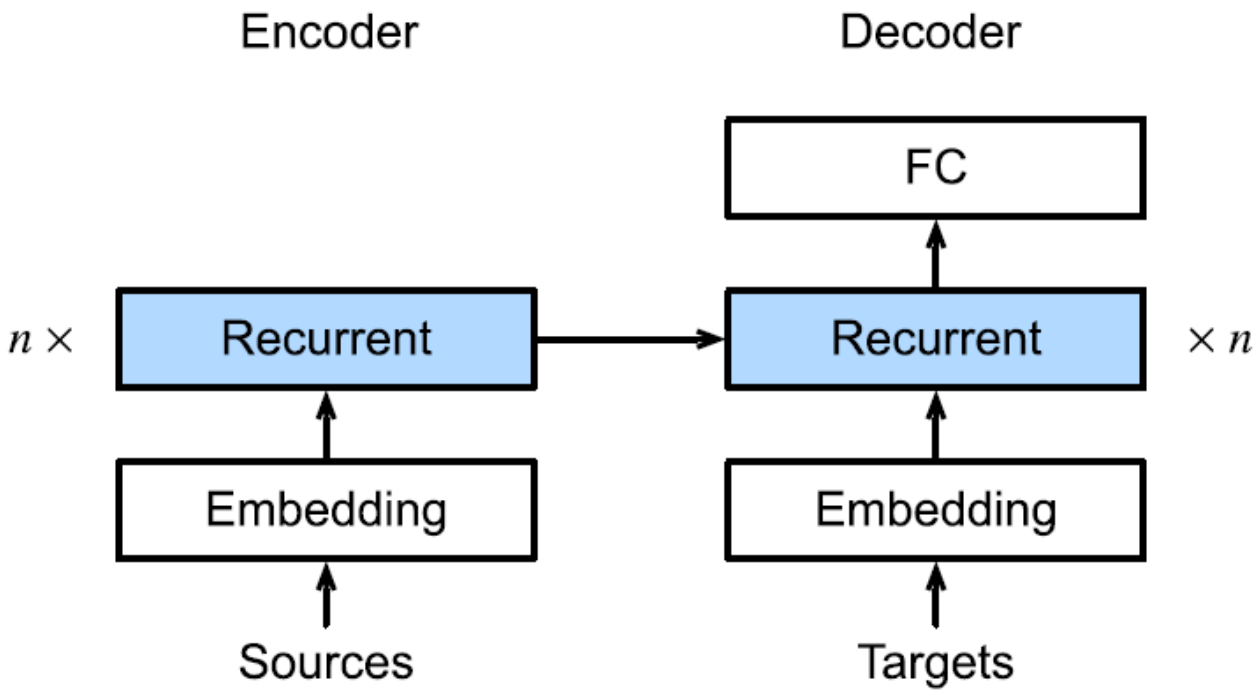


Embedding Layer

- It is a pre-trained model that encodes the language representation, such as characters, words, or sentences, so we can use embedding vectors for other tasks.
- Note that we use an *embedding layer* to obtain the feature vector for each token in the input sequence. The weight of an embedding layer is a matrix, where the number of rows corresponds to the size of the input vocabulary (vocab_size)
- and number of columns corresponds to the feature vector's dimension (embed_size).
- For any input token index *i*, the embedding layer fetches the *i*th row (starting from 0) of the weight matrix to return its feature vector

- The *nn.Embedding* layer takes in two arguments as a minimum. the vocabulary size and the size of the encoded representation for each word. For example, if you have a vocabulary of 10,000 words, then the value of the first argument would be 10,000.
- Each word in the vocabulary will be represented by a vector of fixed size. The second argument is the size of the learned embedding for each word

Embedding Layer Integration



RNN Decoder Implementation

- We directly use the hidden state (output) at the final time step of the encoder to initialize the hidden state of the decoder.
- Notice that there is no fully connected layer for encoder!
- This requires that the RNN encoder and the RNN decoder have the same number of layers and hidden units.
- To further incorporate the encoded input sequence information, the context variable is concatenated with the decoder input at all the time steps
- To predict the probability distribution of the output token, we use a fully connected layer to transform the hidden state at the final layer of the RNN decoder.

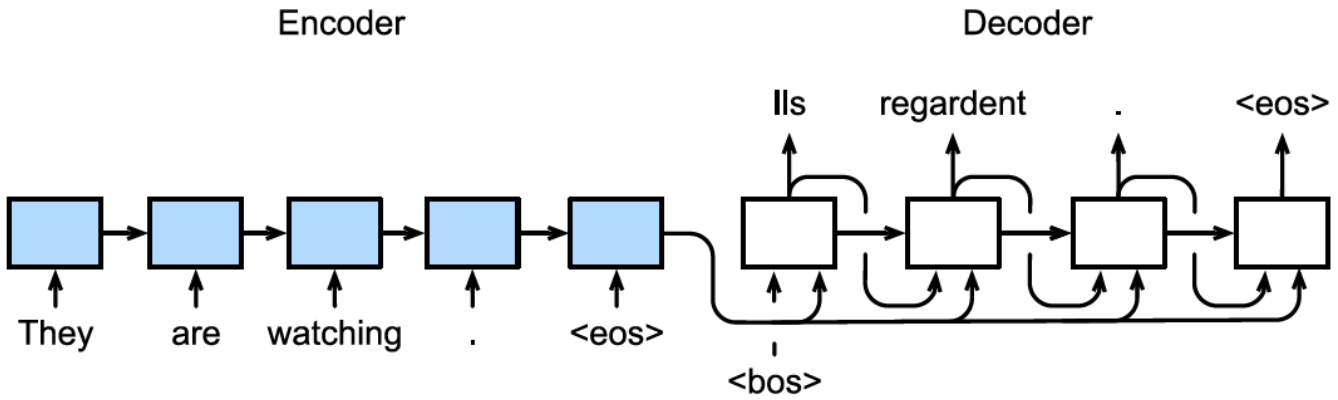
.

Loss Function with Masking

- At each time step, the decoder predicts a probability distribution for the output tokens.
- We apply softmax to obtain the distribution and calculate the cross-entropy loss for optimization.
- Special padding tokens are appended to the end of sequences so sequences of varying lengths can be efficiently loaded in minibatches of the same shape. However, prediction of padding tokens should be excluded from loss calculations.
- To this end, we can mask irrelevant entries with zero values so that multiplication of any irrelevant prediction with zero equals to zero.

Prediction

To predict the output sequence at each step, the predicted token from the previous time step is fed into the decoder as an input.
One simple strategy is to sample whichever token the decoder has assigned the highest probability when predicting at each step.
As in training, at the initial time step the beginning-of-sequence (“<bos>”) token is fed into the decoder.
When the end-of-sequence (“<eos>”) token is predicted, the prediction of the output sequence is complete!



UNC CHARLOTTE