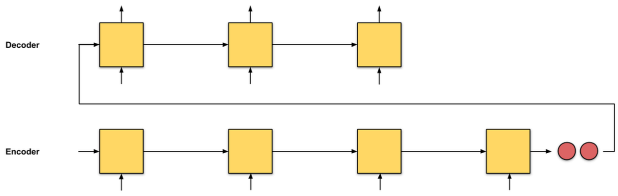
Sequence-to-Sequence

From Seq2Seq to Transformers

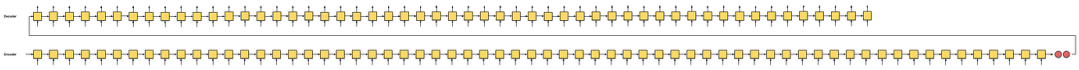
- based on https://github.com/bentrevett/pytorch-seq2seq

1. Sequence to Sequence Learning with Neural Networks

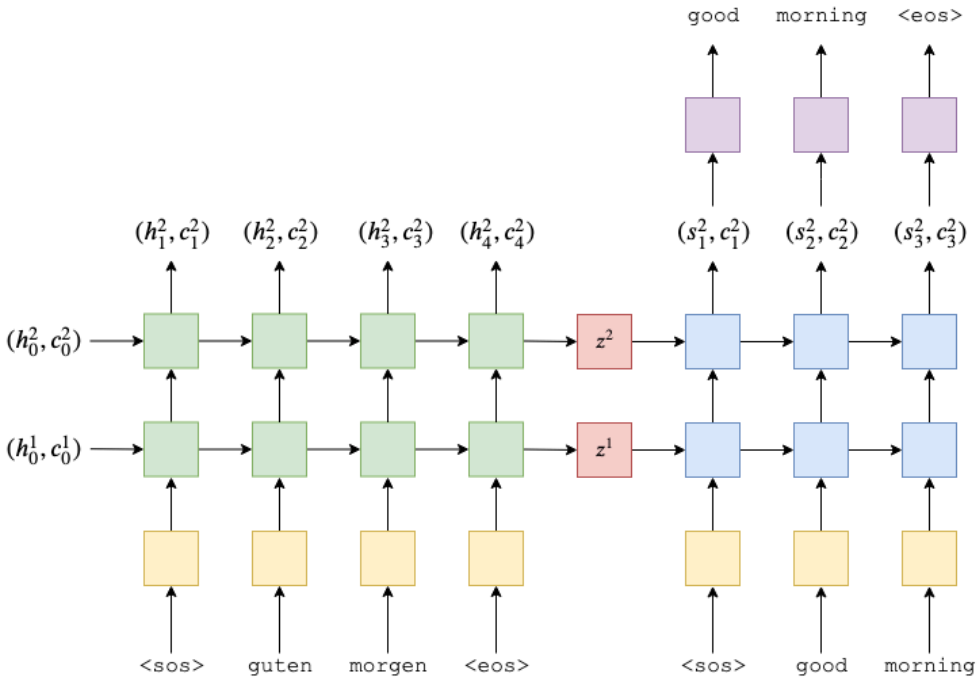
This introduced the basics of seq2seq networks using encoder-decoder models. The model itself will be based off an implementation of Sequence to Sequence Learning with Neural Networks, which uses multi-layer LSTMs.



- seq2seq with an input sequence of length 4

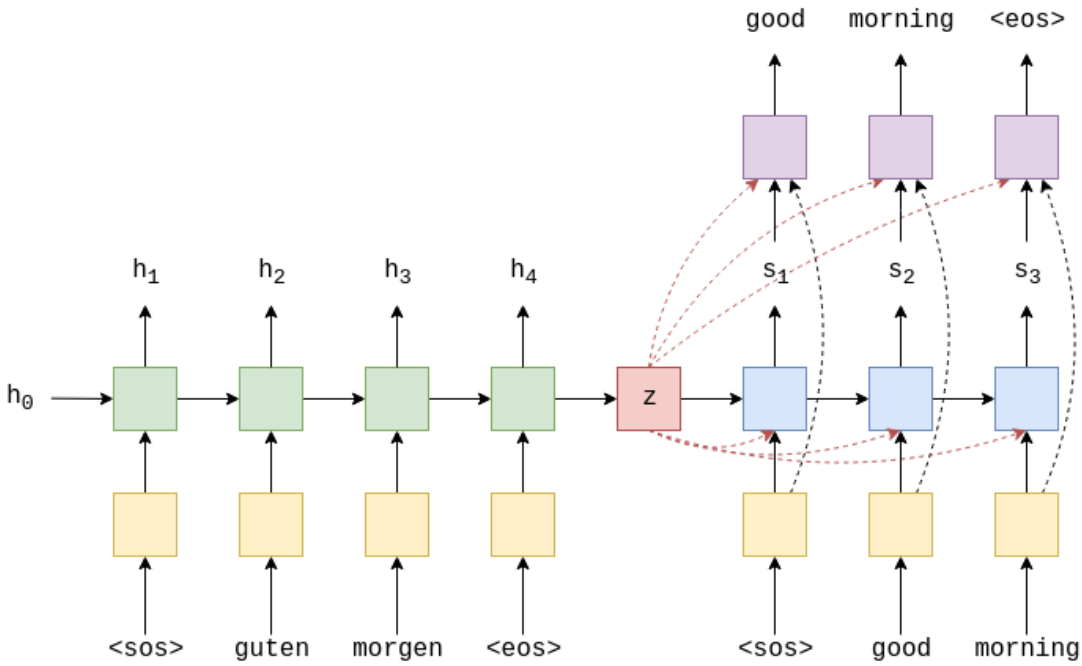


- seq2seq with an input sequence of length 64



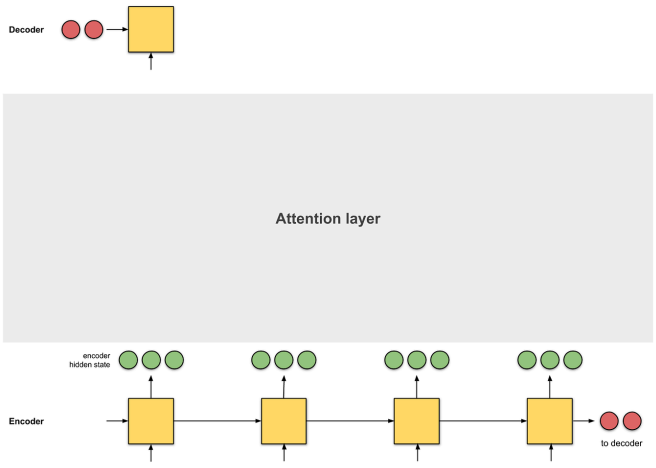
2. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation

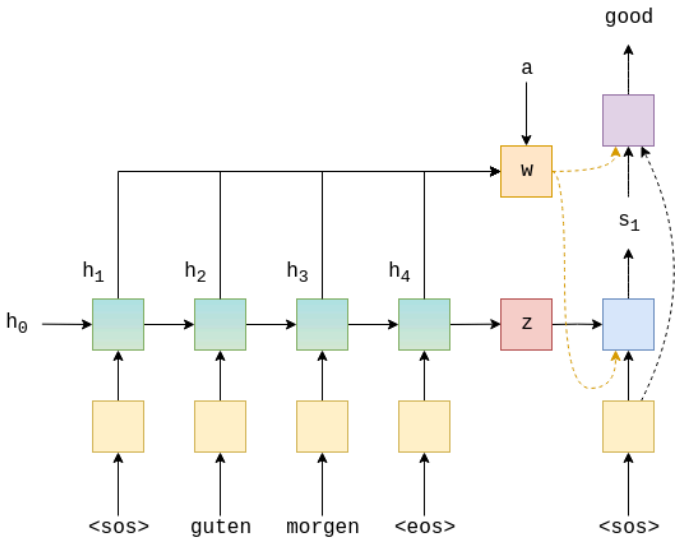
This covers a second model, which helps with the information compression problem faced by encoder-decoder models. This model will be based off an implementation of Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, which uses GRUs.



3. Neural Machine Translation by Jointly Learning to Align and Translate

We learn about **attention** by implementing Neural Machine Translation by Jointly Learning to Align and Translate. This further allievates the information compression problem by allowing the decoder to "look back" at the input sentence by creating context vectors that are weighted sums of the encoder hidden states. The weights for this weighted sum are calculated via an attention mechanism, where the decoder learns to pay attention to the most relevant words in the input sentence.



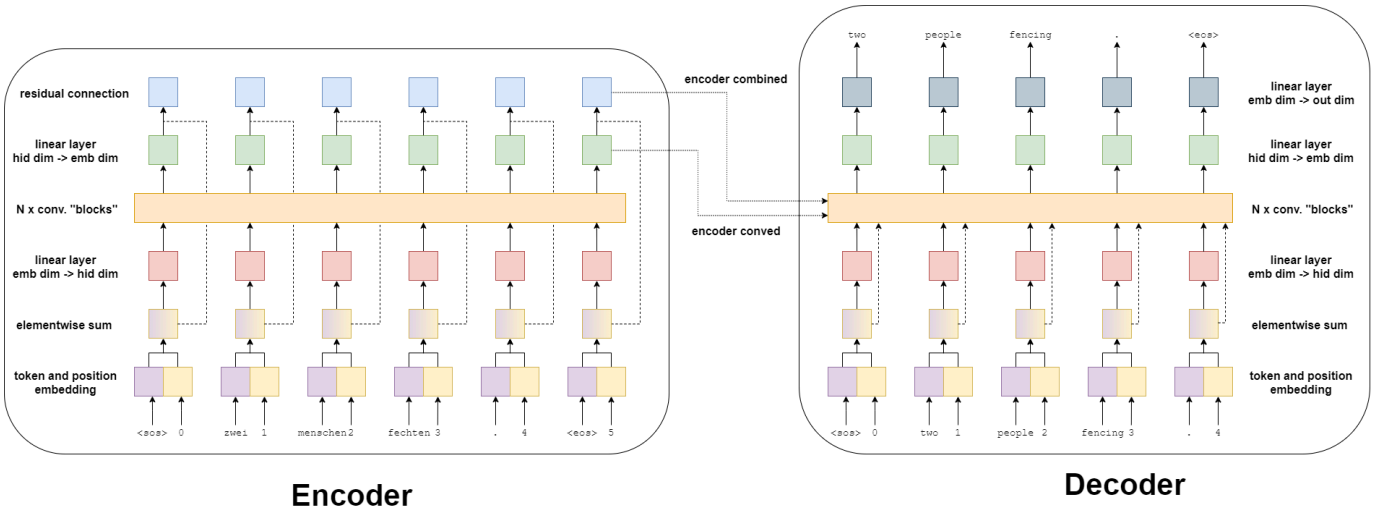


4. Packed Padded Sequences, Masking, Inference and BLEU Open In Colab

We will improve the previous model architecture by adding packed padded sequences and masking. These are two methods commonly used in NLP. Packed padded sequences allow us to only process the non-padded elements of our input sentence with our RNN. Masking is used to force the model to ignore certain elements we do not want it to look at, such as attention over padded elements. Together, these give us a small performance boost. We also cover a very basic way of using the model for inference, allowing us to get translations for any sentence we want to give to the model and how we can view the attention values over the source sequence for those translations. Finally, we show how to calculate the BLEU metric from our translations.

5. Convolutional Sequence to Sequence Learning

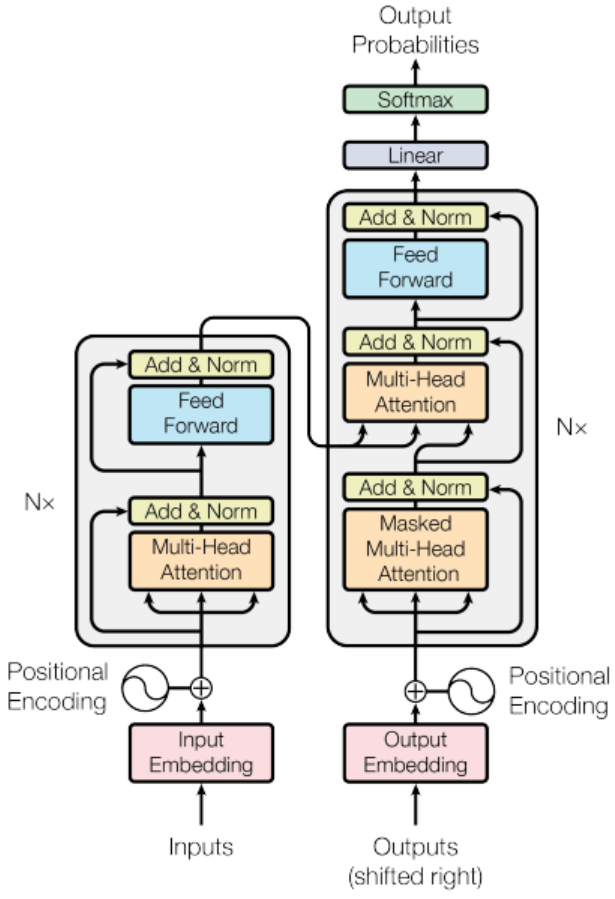
We finally move away from RNN based models and implement a fully convolutional model. One of the downsides of RNNs is that they are sequential. That is, before a word is processed by the RNN, all previous words must also be processed. Convolutional models can be fully parallelized, which allow them to be trained much quicker. We will be implementing the Convolutional Sequence to Sequence model, which uses multiple convolutional layers in both the encoder and decoder, with an attention mechanism between them.



Encoder **Decoder**

6. Attention Is All You Need

Continuing with the non-RNN based models, we implement the Transformer model from Attention Is All You Need. This model is based soley on attention mechanisms and introduces Multi-Head Attention. The encoder and decoder are made of multiple layers, with each layer consisting of Multi-Head Attention and Positionwise Feedforward sublayers. This model is currently used in many state-of-the-art sequence-to-sequence and transfer learning tasks.



Sequence to Sequence Basics

Formally, in the machine translation task, we have an input sequence $x_1, x_2, \dots, x_m$ and an output sequence $y_1, y_2, \dots, y_n$ (note that their lengths can be different). Translation can be thought of as finding the target sequence that is the most probable given the input; formally, the target sequence that maximizes the conditional probability $p(y|x)$: $y^* = \arg\max_y p(y|x)$.

If you are bilingual and can translate between languages easily, you have an intuitive feeling of $p(y|x)$ and can say something like "...well, this translation is kind of more natural for this sentence". But in machine translation, we learn a function $p(y|x, \theta)$ with some parameters θ, and then find its argmax for a given input: $y' = \arg\max_y p(y|x, \theta)$.

To define a machine translation system, we need to answer three questions:

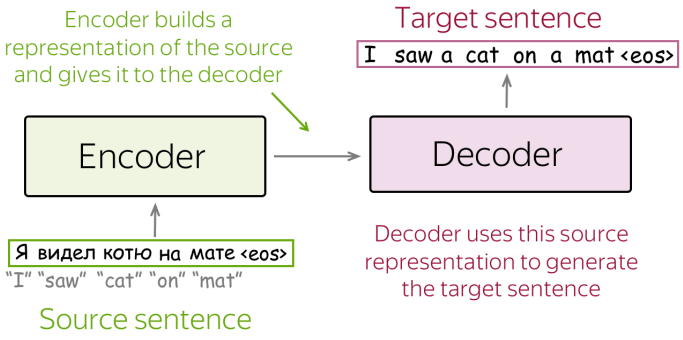
- modeling - how does the model for $p(y|x, \theta)$ look like?
- learning - how to find the parameters θ?
- inference - how to find the best y?

In this section, we will answer the second and third questions in full, but consider only the simplest model. The more "real" models will be considered later in sections Attention and Transformer.

Encoder-Decoder Framework

Encoder-decoder is the standard modeling paradigm for sequence-to-sequence tasks. This framework consists of two components:

- encoder - reads source sequence and produces its representation;
- decoder - uses source representation from the encoder to generate the target sequence.



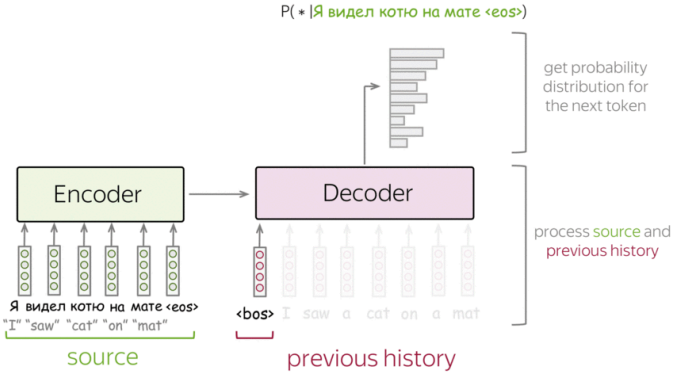
Conditional Language Models

In the Language Modeling lecture, we learned to estimate the probability $p(y)$ of sequences of tokens $y = (y_1, y_2, \dots, y_n)$. While language models estimate the unconditional probability $p(y)$ of a sequence y, sequence-to-sequence models need to estimate the conditional probability p(y|x) of a sequence y given a source x. That's why sequence-to-sequence tasks can be modeled as Conditional Language Models (CLM) - they operate similarly to LMs, but additionally receive source information x.

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

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

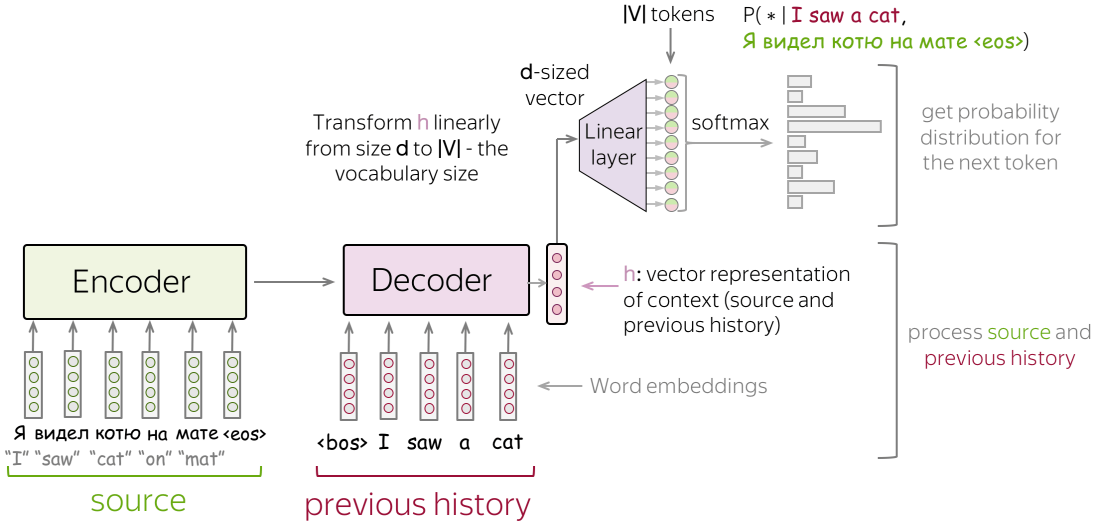
condition on source x



Since the only difference from LMs is the presence of source , the modeling and training is very similar to language models. In particular, the high-level pipeline is as follows:

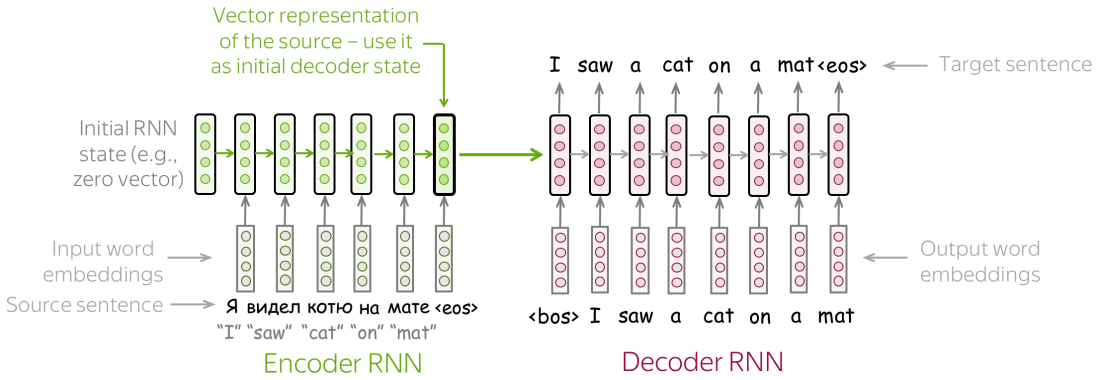
- feed source and previously generated target words into a network;
- get vector representation of context (both source and previous target) from the networks decoder;

- from this vector representation, predict a probability distribution for the next token.



Similarly to neural classifiers and language models, we can think about the classification part (i.e., how to get token probabilities from a vector representation of a text) in a very simple way. Vector representation of a text has some dimensionality , but in the end, we need a vector of size (probabilities for tokens/classes). To get a -sized vector from a -sized, we can use a linear layer. Once we have a -sized vector, all is left is to apply the softmax operation to convert the raw numbers into token probabilities.

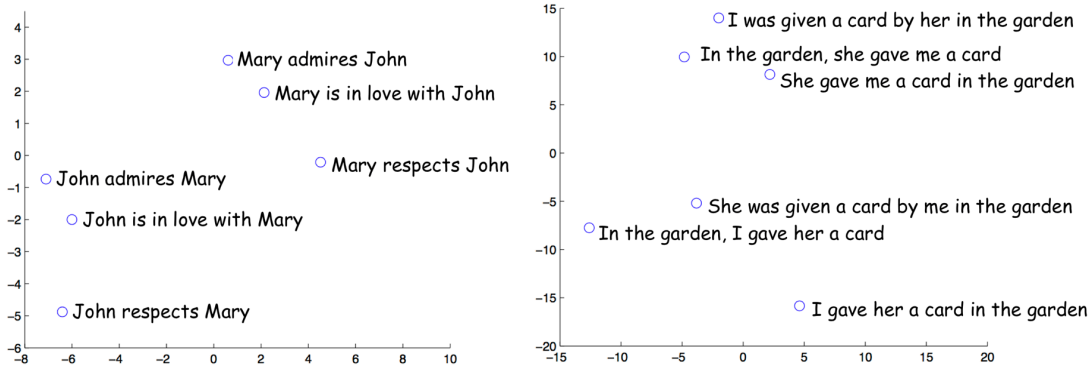
The Simplest Model: Two RNNs for Encoder and Decoder



The simplest encoder-decoder model consists of two RNNs (LSTMs): one for the encoder and another for the decoder. Encoder RNN reads the source sentence, and the final state is used as the initial state of the decoder RNN. The hope is that the final encoder state "encodes" all information about the source, and the decoder can generate the target sentence based on this vector.

This model can have different modifications: for example, the encoder and decoder can have several layers. Such a model with several layers was used, for example, in the paper Sequence to Sequence Learning with Neural Networks - one of the first attempts to solve sequence-to-sequence tasks using neural networks.

In the same paper, the authors looked at the last encoder state and visualized several examples - look below. Interestingly, representations of sentences with similar meaning but different structure are close!



The examples are from the paper Sequence to Sequence Learning with Neural Networks.

Training: The Cross-Entropy Loss (Once Again)

Similarly to neural LMs, neural seq2seq models are trained to predict probability distributions of the next token given previous context (source and previous target tokens). Intuitively, at each step we maximize the probability a model assigns to the correct token.

Formally, let's assume we have a training instance with the source $x = (x_1, \dots, x_m)$ and the target $y = (y_1, \dots, y_n)$. Then at the timestep t, a model predicts a probability distribution $p^{(t)} = p(*|y_1, \dots, y_{t-1}, x_1, \dots, x_m)$. The target at this step is $p^* = \text{one-hot}(y_t)$, i.e., we want a model to assign probability 1 to the correct token, y_t, and zero to the rest.

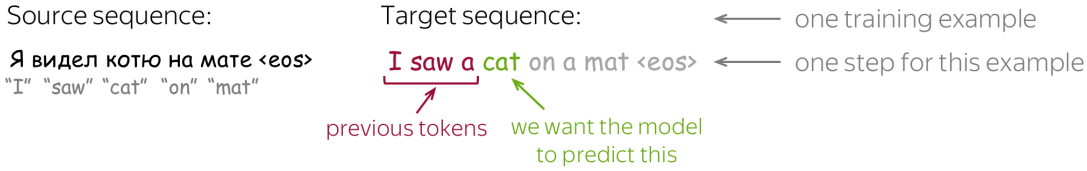
The standard loss function is the cross-entropy loss. Cross-entropy loss for the target distribution p^* and the predicted distribution p is

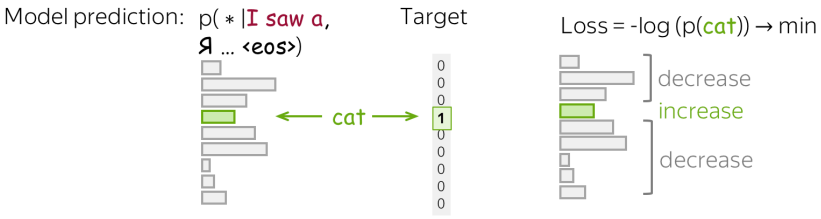
$$Loss(p^*, p) = -p^* \log(p) = -\sum_{i=1}^{|V|} p_i^* \log(p_i).$$

Since only one of p_i^* is non-zero (for the correct token y_t), we will get

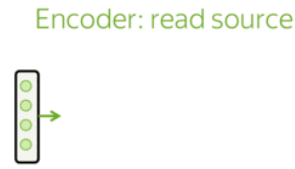
$$Loss(p^*, p) = -\log(p_{y_t}) = -\log(p(y_t|y_{<t}, x)).$$

At each step, we maximize the probability a model assigns to the correct token. Look at the illustration for a single timestep.





For the whole example, the loss will be $-\sum_{t=1}^{n} \log(p(y_t|y_{<t}, x))$. Look at the illustration of the training process (the illustration is for the RNN model, but the model can be different).



Inference: Greedy Decoding and Beam Search

Now when we understand how a model can look like and how to train this model, let's think how to generate a translation using this model. We model the probability of a sentence as follows:

$$y' = \arg\max_y p(y|x) = \arg\max_y \prod_{t=1}^{n} p(y_t|y_{<t}, x)$$ How to find the argmax?

Now the main question is: how to find the argmax?

Note that **we can not find the exact solution.** The total number of hypotheses we need to check is $|V|^n$, , which is not feasible in practice. Therefore, we will find an approximate solution.

• **Greedy Decoding:** At each step, pick the most probable token The straightforward decoding strategy is greedy - at each step, generate a token with the highest probability. This can be a good baseline, but this method is inherently flawed: the best token at the current step does not necessarily lead to the best sequence.

$$\arg\max_y \prod_{t=1}^{n} p(y_t|y_{<t}, x) \neq \prod_{t=1}^{n} \arg\max_{y_t} p(y_t|y_{<t}, x)$$

• **Beam Search:** Keep track of several most probably hypotheses Instead, let's keep several hypotheses. At each step, we will be continuing each of the current hypotheses and pick top-N of them. This is called beam search.

<bos>

Start with the begin of sentence token or with an empty sequence

Usually, the beam size is 4-10. Increasing beam size is computationally inefficient and, what is more important, leads to worse quality.