Machine Translation. Seq2Seq. Attention. Transformers

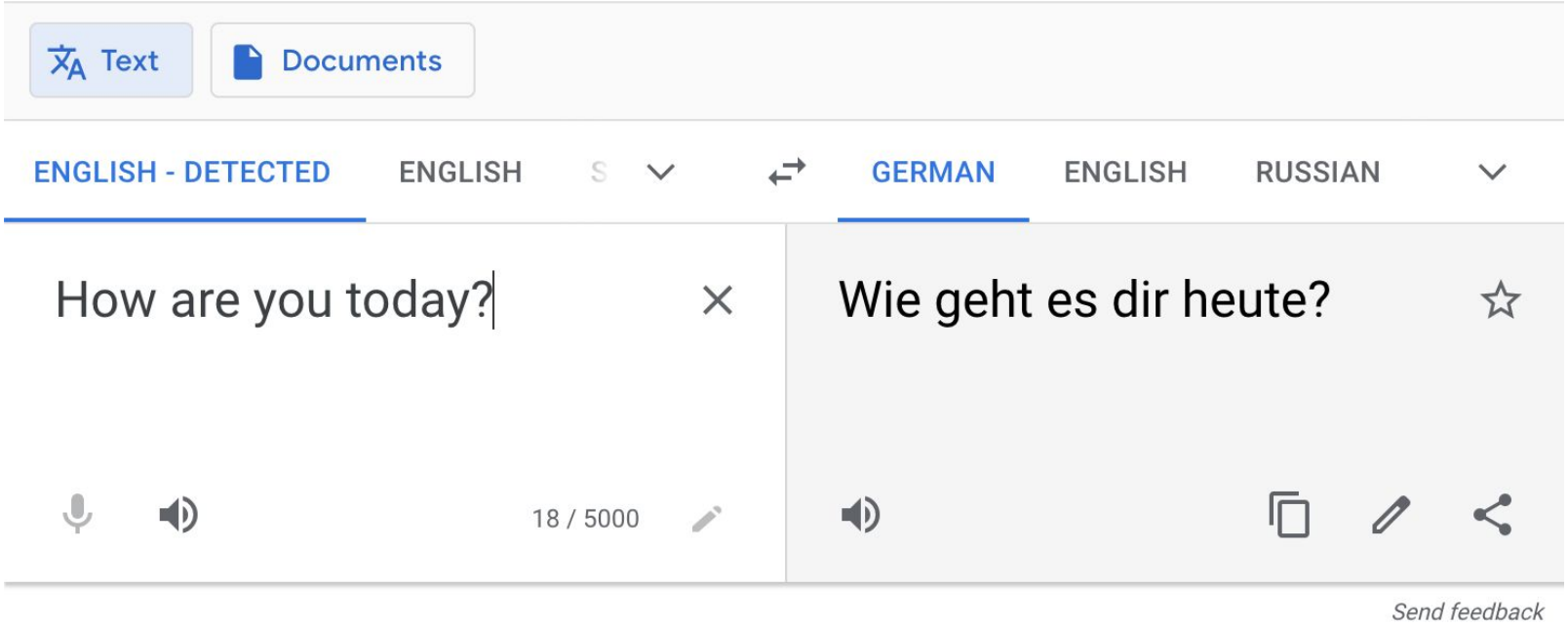
Alena Fenogenova

Lecture 6 15/10/2021

Today

- Machine Translation
  - Seq2seq
  - Decoding techniques
  - Evaluation
- Attention
  - Limitations of RNNS
  - Attention mechanism
  - Early attention models
- Transformer
  - High-level
  - Deeper

Machine translation



- Machine Translation is very popular NLP task
- Georgetown–IBM experiment
- Statistical models
- NMT models (from 2016 industrial standard)

Machine translation task

Source sequence ⟹ Target sequence

Machine translation task – finding the target sequence that is the most probable given the input.

Machine translation can be applied to sequences of any nature:

- between natural languages
- between programming languages
- any sequences of tokens

Any general sequence-to-sequence task

Machine translation task

Source sentence $(x_1, x_2, \ldots\ x_n)$

Target sentence $(y_1, y_2, \ldots y_n)$

Machine translation systems learn a function: $\boldsymbol{p(y|x, \theta)}$

We try to find the target sequence that maximizes the

conditional probability: $\boldsymbol{y = arg\ \max_{y} p(y|x, \theta)}$

where θ – model parameters that determine probability distribution

Statistical approach

1. Large parallel corpora (pairs of x, y)
2. Learn probabilistic model from data
3. Alignment

Bayes rule:

translation model

$$arg\max_y P(y|x) = arg\max_y P(x|y)P(y)$$

language model

Model p(y|x, θ) - Statistical approach

Alignment is the correspondence between source sentence x and target sentence y.

Learn translation model: $P(x|y)$

Introduce latent a variable into the model: $P(x, a|y)$

where a is the **alignment**

Tomorrow I will fly to the conference

Я полечу на конференцию завтра

NMT approach

Encoder reads source sequence and produces its representation;

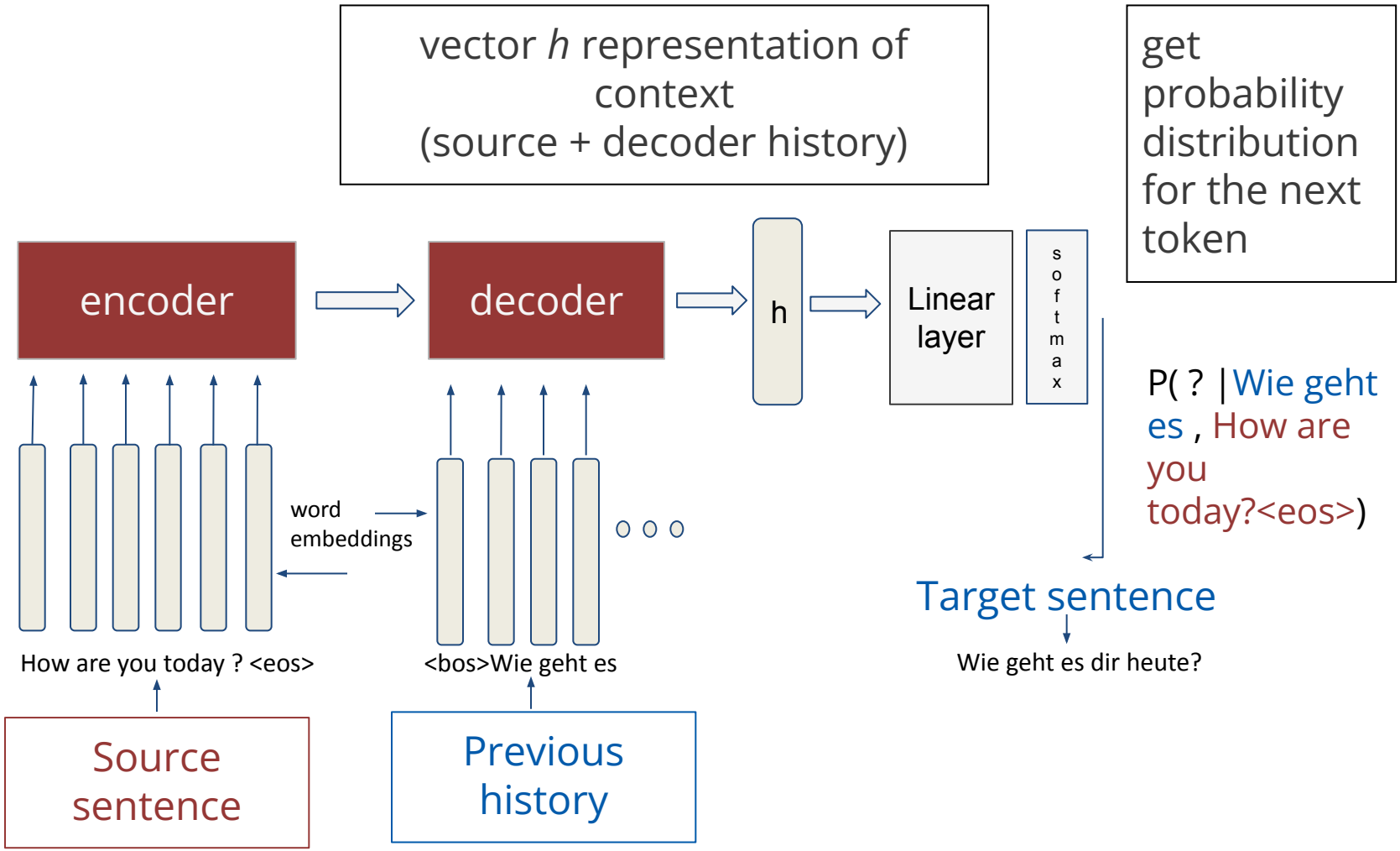
Decoder uses source representation from the encoder to generate the target sequence.

NMT approach

Seq2Seq pipeline:

- 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.

NMT approach



NMT approach

Language models

estimate the *unconditional probability* $p(y)$ of a sequence y

VS

Sequence-to-sequence models

need to estimate the *conditional probability* $p(y|x)$ of a sequence y given a source x

How to find parameter θ?

We train to predict probability distributions of the next token given previous context (source and previous target tokens).

At each step we maximize the probability a model assigns to the correct token.

Backpropagation. Cross-entropy loos

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

How to find the best y? Inference

How to find argmax?

$$y' = \arg\max_y p(y|x) = \arg\max_y \prod_{t=1}^{n} p(y_t|y_{<t}, x)$$

The total number of hypotheses is $|V|^n$

We don't try to find exact solution, we approximate it.

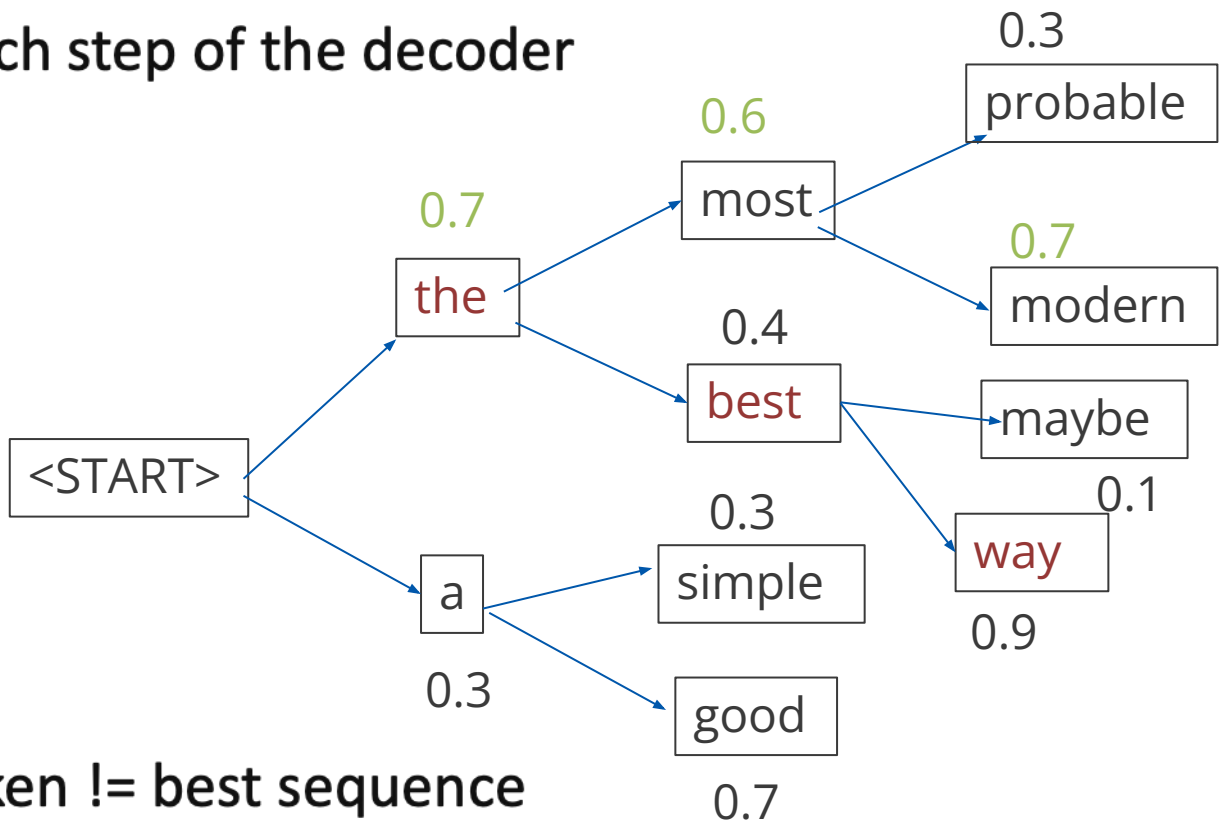
1. Greedy decoding
2. Beam search
3. Sampling

Decoding techniques

Greedy decoding

At each step, pick the most probable token

Take $argmax$ on each step of the decoder



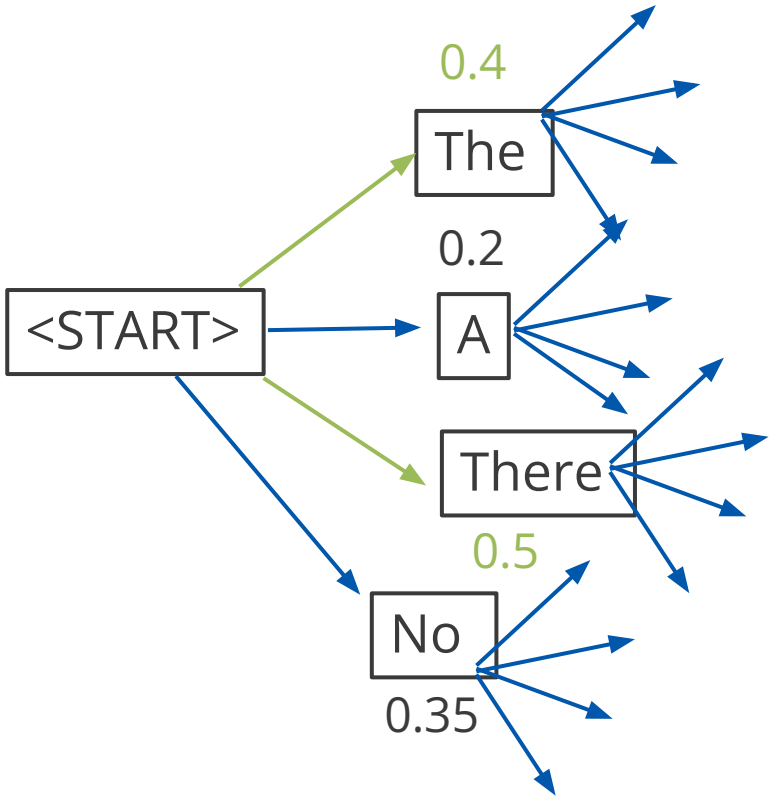
Problems:

- the best next token != best sequence
- has no way to undo decisions

Beam search

Keep track of K most probable partial translations (hypotheses)

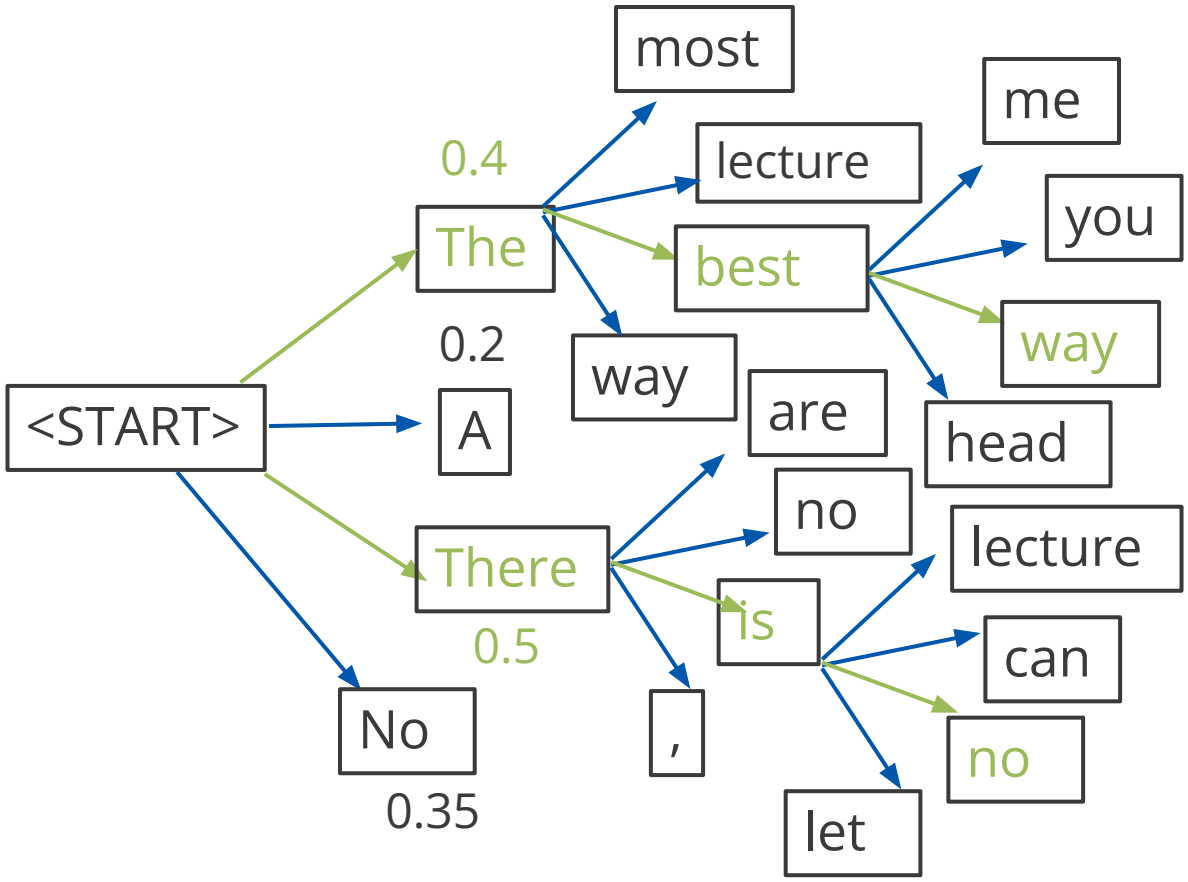
- k is the beam size (in practice 4 to 10)



Beam search

Keep track of K most probable partial translations (hypotheses)

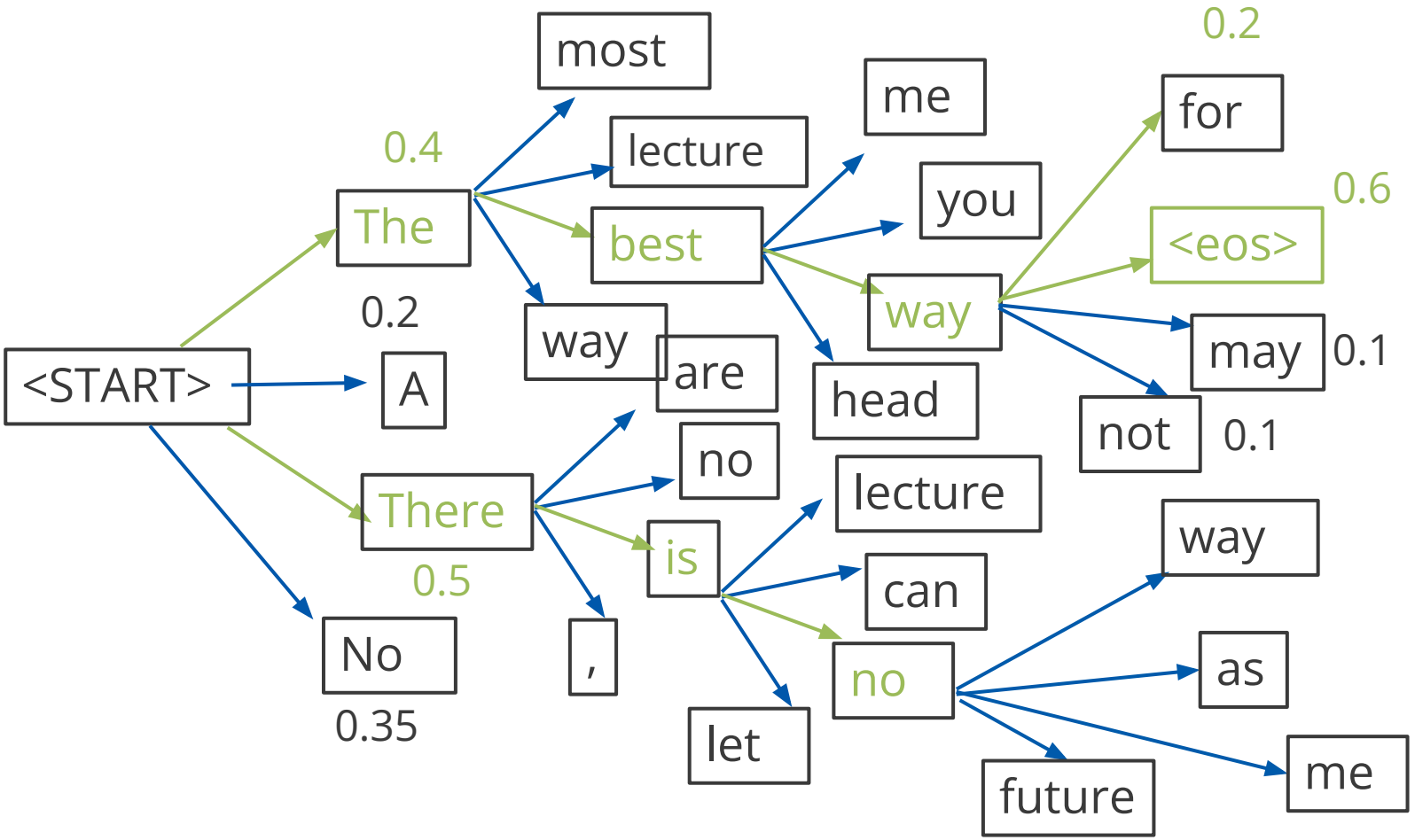
- k is the beam size (in practice 4 to 10)



Beam search

Keep track of K most probable partial translations (hypotheses)

- k is the beam size (in practice 4 to 10)



Sampling

Use the distributions predicted by a model.

We can modify the distributions predicted by a model in different ways, depending of what we need from final text.

Usually we want from generated text:

- to be coherent and meaningfull
- several different but appropriate examples

Sampling

Random sampling utilizes the probability of each token from the softmax function to generate the next token.

It's random, thats why:

+ may be creative

- lack of control

Randomly picking the next word according to its conditional probability distribution:

$$wt \sim P(w \mid w_{1:t-1})$$

Sampling

Temperature sampling.

Temperature is used to increase the probability of probable tokens while reducing the one that is not.

Before applying the final softmax, its inputs are divided by the temperature parameter.

$$\frac{\exp(h^T w)}{\sum_{w_i \in V} \exp(h^T w_i)} \Rightarrow \frac{\exp(\frac{h^T w}{\tau})}{\sum_{w_i \in V} \exp(\frac{h^T w_i}{\tau})}$$

- temperature < 1 — sharp, small variance
- temperature > 1 — not sharp, big variance

Sampling

Top-k sampling.
Only top K probable tokens should be considered for a generation.

- difficult to choose k, as on each step different distributions
- cover very small part of the total probability mass

Top-p sampling or Nucleus
Consider the words which have a cumulative probability mass larger than the hyperparameter *p*.

$$\sum_{x \in V^{(p)}} P(x|x_{1:i-1}) \geq p.$$

Evaluation of MT models

Evaluation of MT

BLEU (Bilingual Evaluation Understudy)

BLEU compares the machine-written translation to one or several human-written translation(s), and computes a similarity score based on:

- n-grams precision (usually for 1, 2, 3 and 4-grams)
- plus a penalty for too-short system translations

$$\frac{Sum\ over\ unique\ ngram\ counts\ in\ the\ candidate}{total\ number\ of\ tokens\ in\ candidate}$$

Limitations: semantics, the order of the n-grams in the sentence.

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right).$$

Evaluation of MT

ROUGE (Recall Oriented Understudy for Gisting Evaluation)

ROUGE-N: Overlap of N-grams refers to the overlap of ***n-grams*** between the system and reference summaries;

- compute unique n-grams
- check overlap and length
- => F1 measure

Recall: $$\frac{|ngrams(ref) \& ngrams(hyp)|}{|ngrams(ref)|}$$

Precision: $$\frac{|ngrams(ref) \& ngrams(hyp)|}{|ngrams(hyp)|}$$

F1: $$2\frac{P*R}{R+P}$$

Evaluation of MT

ROUGE-L: Longest Common Subsequence (LCS) based statistics.

Longest common subsequence problem takes into account sentence level structure similarity naturally and identifies longest co-occurring in sequence n-grams automatically.

Limitations: meaning (does not understand concepts and themes)

Evaluation of MT

The Meteor automatic evaluation metric scores machine translation and other generation tasks hypotheses by aligning them to one or more references.

Alignments are based on exact, stem, synonym, and paraphrase matches between words and phrases.

Weighted F-score $$F = \frac{PR}{\alpha P + (1-\alpha)R)}$$

Penalty function $Penalty = \gamma(\frac{c}{m})^{\beta}$, where $0 \leq \gamma \leq 1$

$$Score = Fmean * (1 - Penalty)$$

Evaluation of MT

Automatic:

- General Text Matcher (GTM)
- Word Error Rate (WER)
- Translation Edit Rate (TER)
- CDER
- ...

https://www.cs.cmu.edu/~alavie/papers/GALE-book-Ch5.pdf

Tools

Statistical:

- https://github.com/moses-smt/mosesdecoder
- https://github.com/apertium

NN:

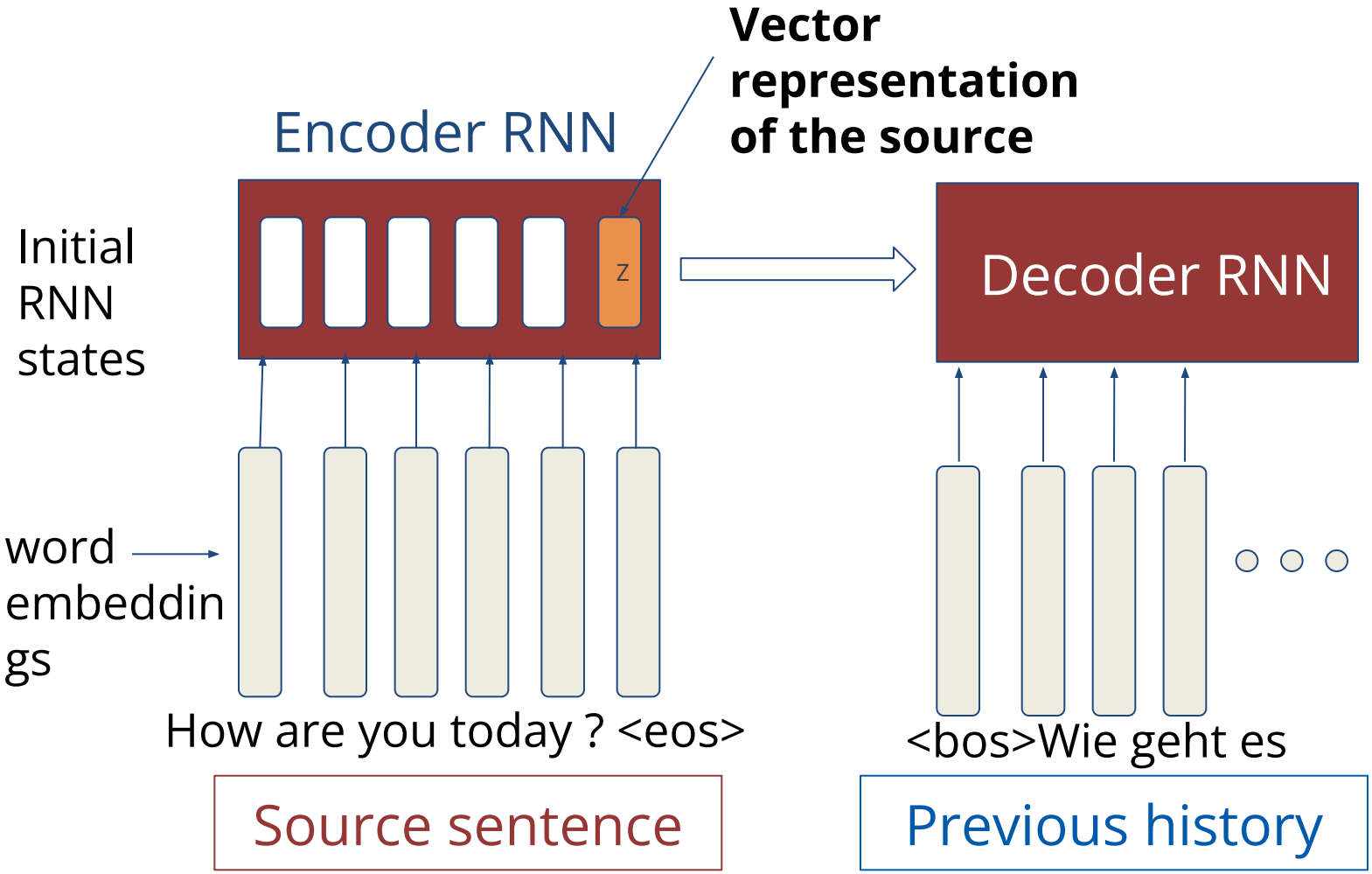
- https://github.com/UKPLab/EasyNMT (opus)
- https://huggingface.co/transformers/model_doc/marian.html

Apps:

- Yandex Translator
- Google Translator API

Limitations

Limitations of seq2seq



Limitations of seq2seq

- for the encoder, it is hard to compress the sentence;
- for the decoder, different information may be relevant at different steps;
- how far we can see in the past is finite.

The intermediate representation z cannot encode information from all the input timesteps **bottleneck problem**.

Limitations of RNNs

Linear interaction distance

RNNs are unrolled from left-to-right. This encodes linear locality.

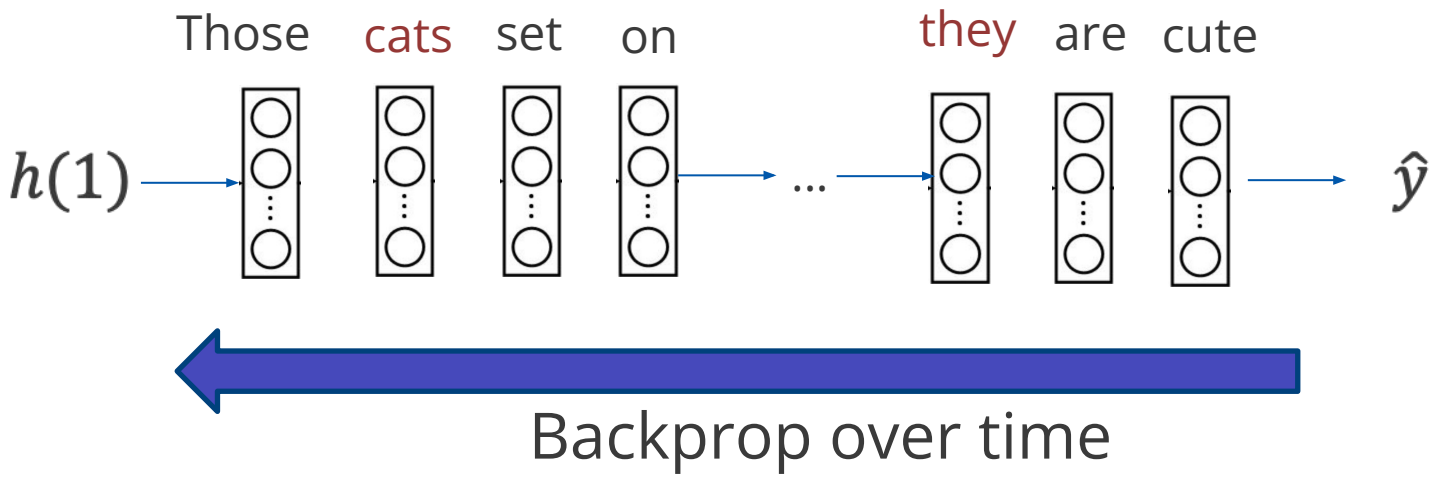
Nearby words often affect each other's meanings

Problem: RNNs take **O(sequence length)** steps for distant word pairs to interact.

- Linear order of words isn't the right way to think about sentences
- Hard to learn long-distance dependencies (vanishing gradient problems!)

Limitations of RNNs

Vanishing gradient problem



When the signals that are far away they are small!

The gradient signal gets smaller and smaller as it backpropagates further.

Model weights are updated only with respect to near effects.

Limitations of RNNs

Lack of parallelizability

- Forward and backward passes have O(sequence length) unparallelizable operations
  - GPUs can perform a bunch of independent computations at once!
  - But future RNN hidden states can't be computed in full before past RNN hidden states have been computed.
  - Inhibits training on very large datasets!

Attention

Attention

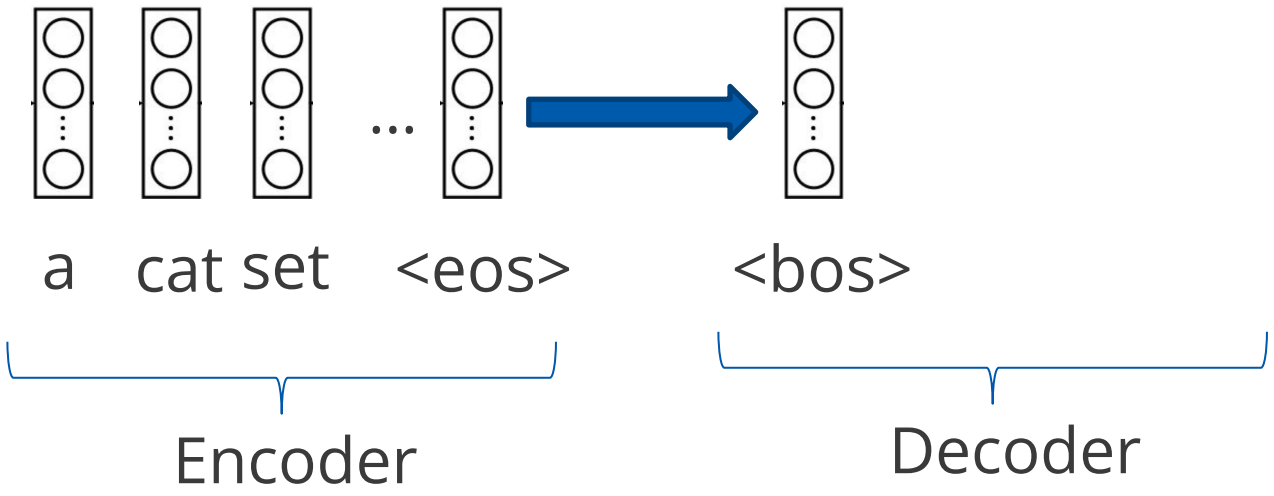
Attention:

At different steps, let a model "focus" on different parts of the source tokens (more relevant ones).

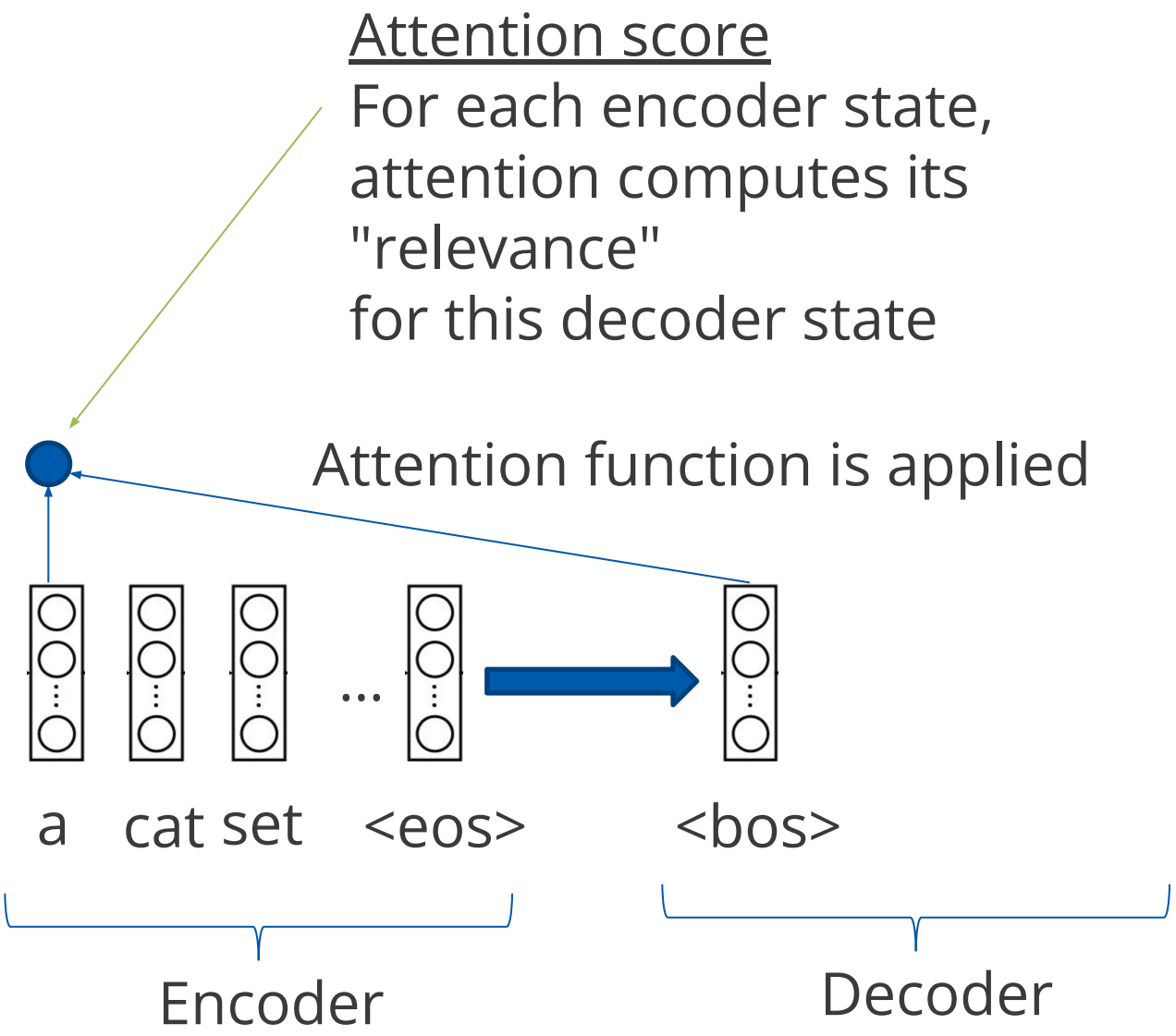
Core idea:

on each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence

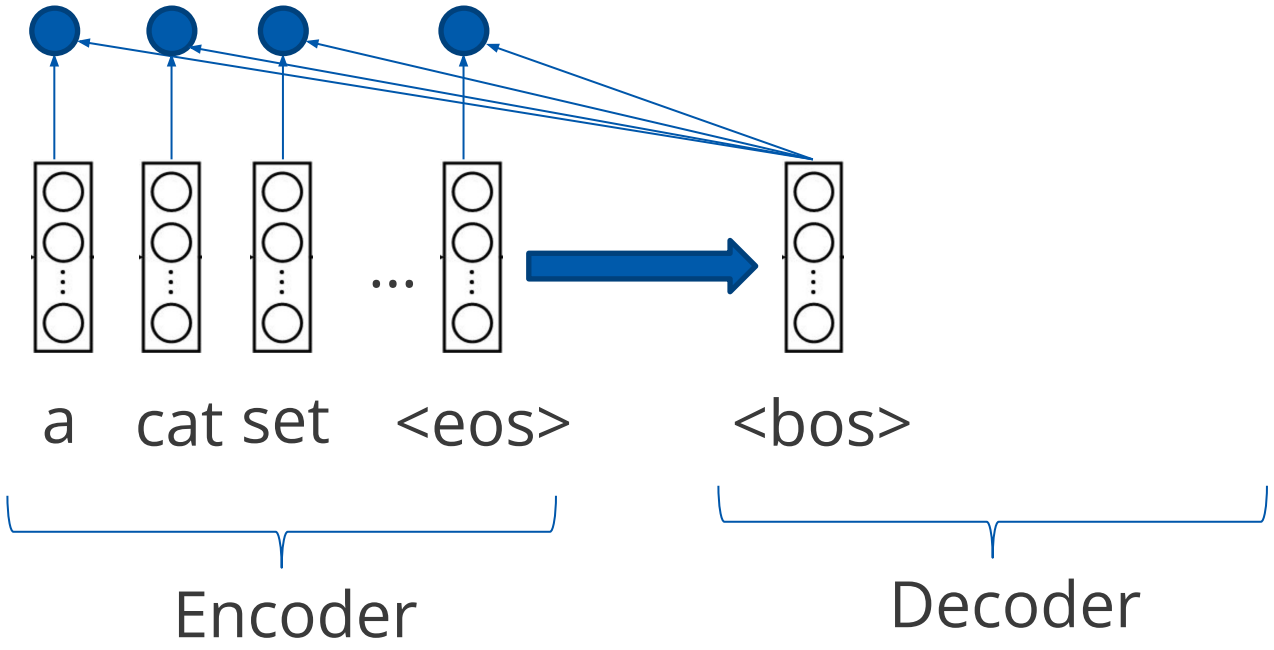
Attention



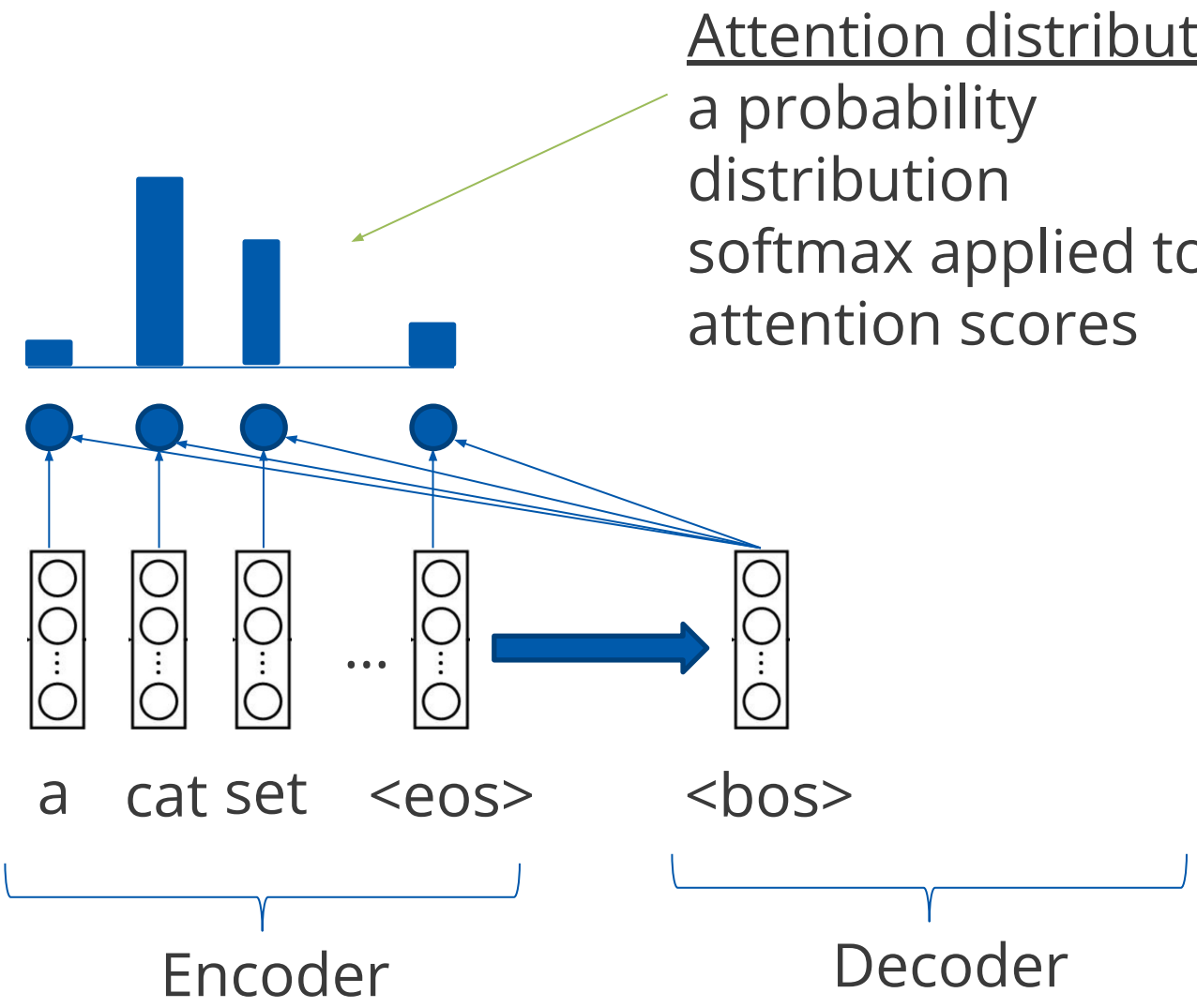
Attention


Attention score
For each encoder state, attention computes its "relevance" for this decoder state
Attention function is applied
...
a
cat
set
<eos>
<bos>
Encoder
Decoder

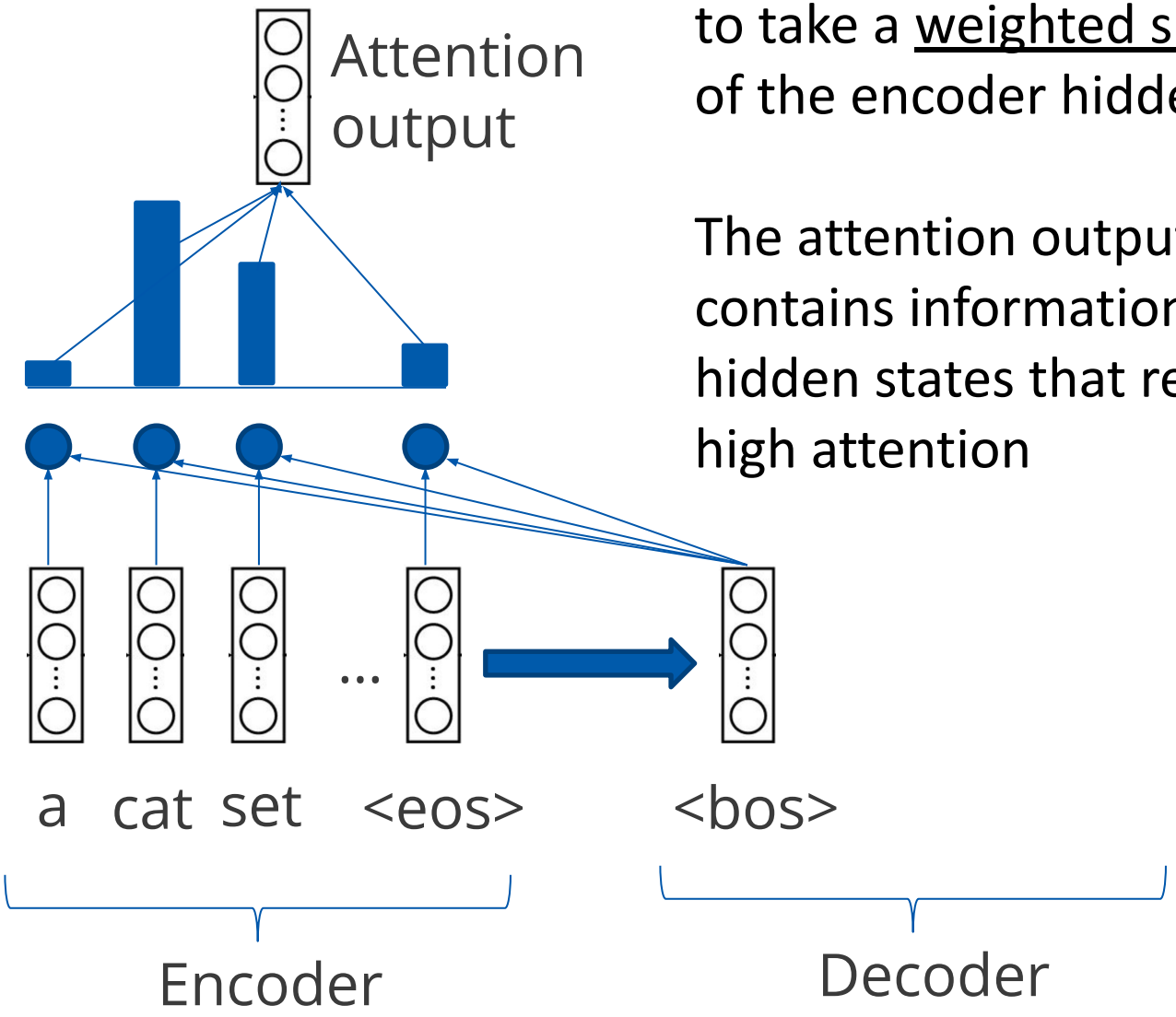
Attention



Attention


Attention distribution
a probability distribution
softmax applied to attention scores
a
cat
set
<eos>
<bos>
Encoder
Decoder

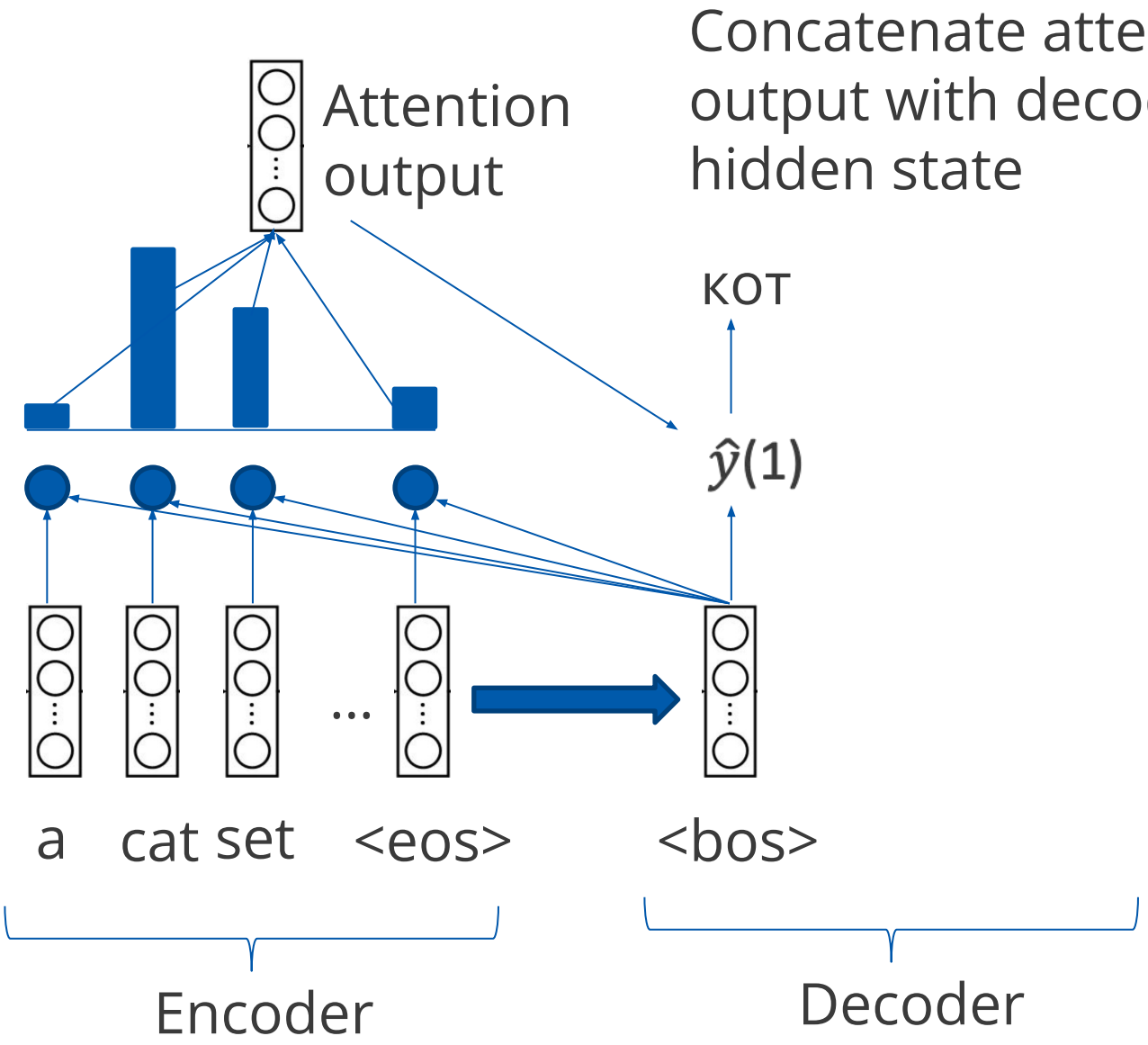
Attention



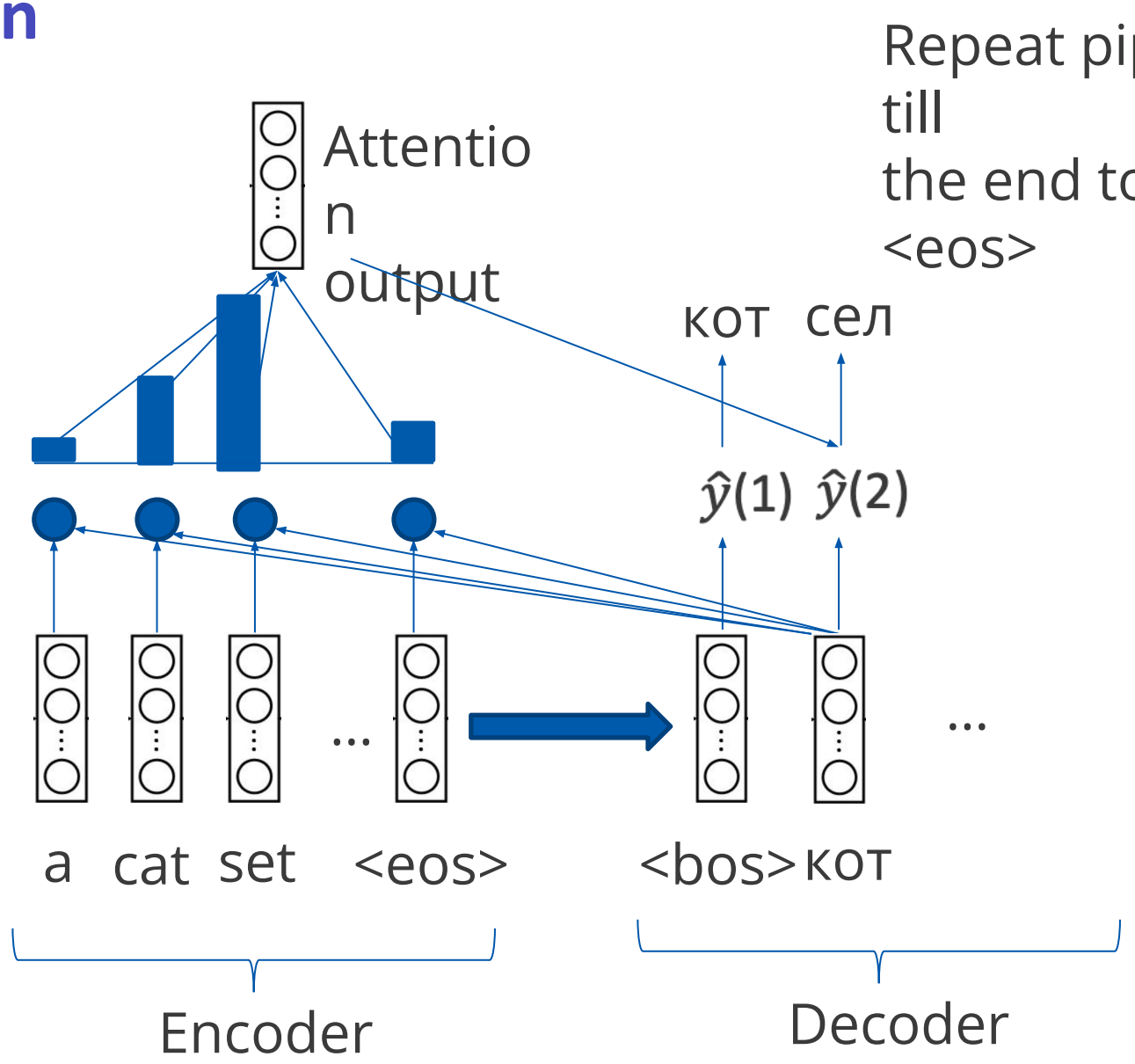
Use the attention distribution to take a weighted sum of the encoder hidden states.

The attention output mostly contains information from the hidden states that received high attention

Attention


Attention output
Concatenate attention output with decoder hidden state
KOT
$\hat{y}(1)$
a
cat
set
<eos>
<bos>
Encoder
Decoder

Attention


Repeat pipeline
till
the end token
<eos>
Attentio
n
output
кот сел
$\hat{y}(1)$ $\hat{y}(2)$
...
...
a cat set <eos>
<bos> кот
Encoder
Decoder

Attention

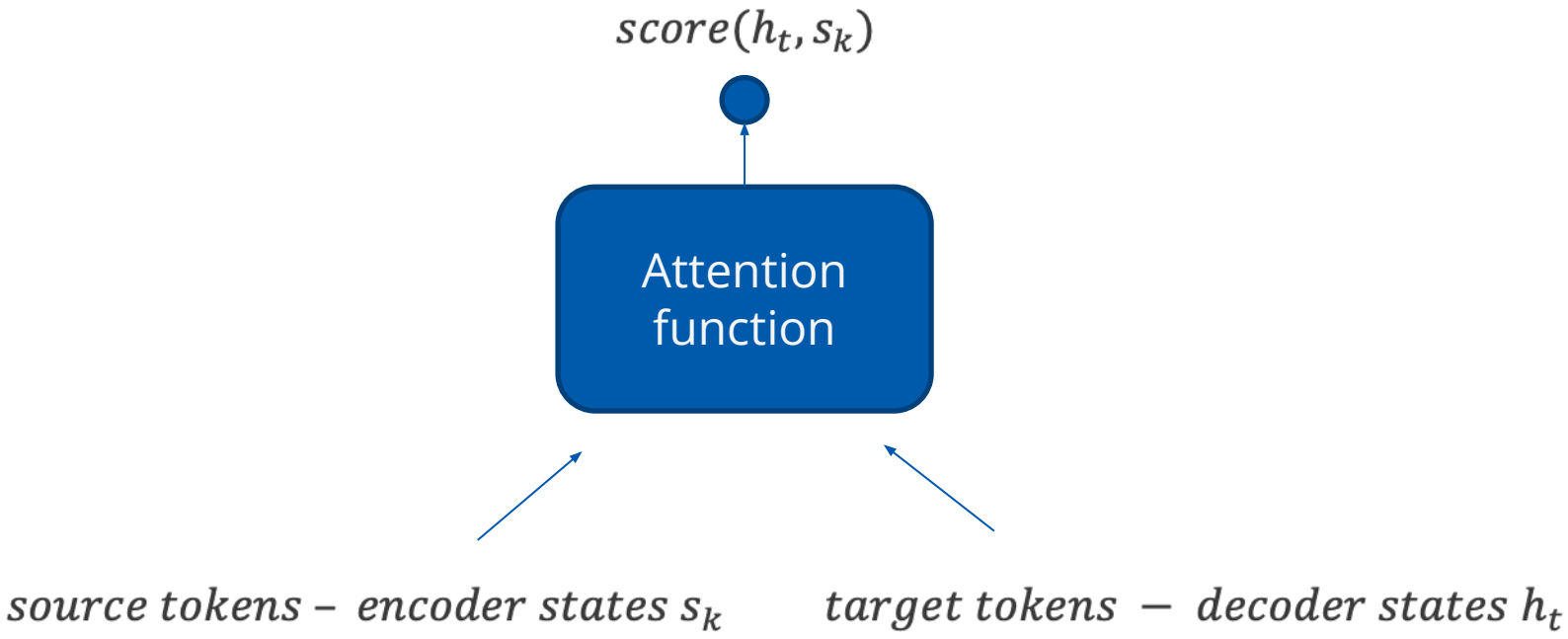
At each decoder step, attention:

1. receives attention input
2. computes attention scores
3. computes attention weights:

   a probability distribution - softmax applied to attention scores;
4. computes attention output:

   the weighted sum of encoder states with attention weights.

Attention scores



Attention scores

- dot-product - the simplest method;

$$score(h_t, s_k) = h_t^T s_k$$

- general or bilinear function ("Luong attention")

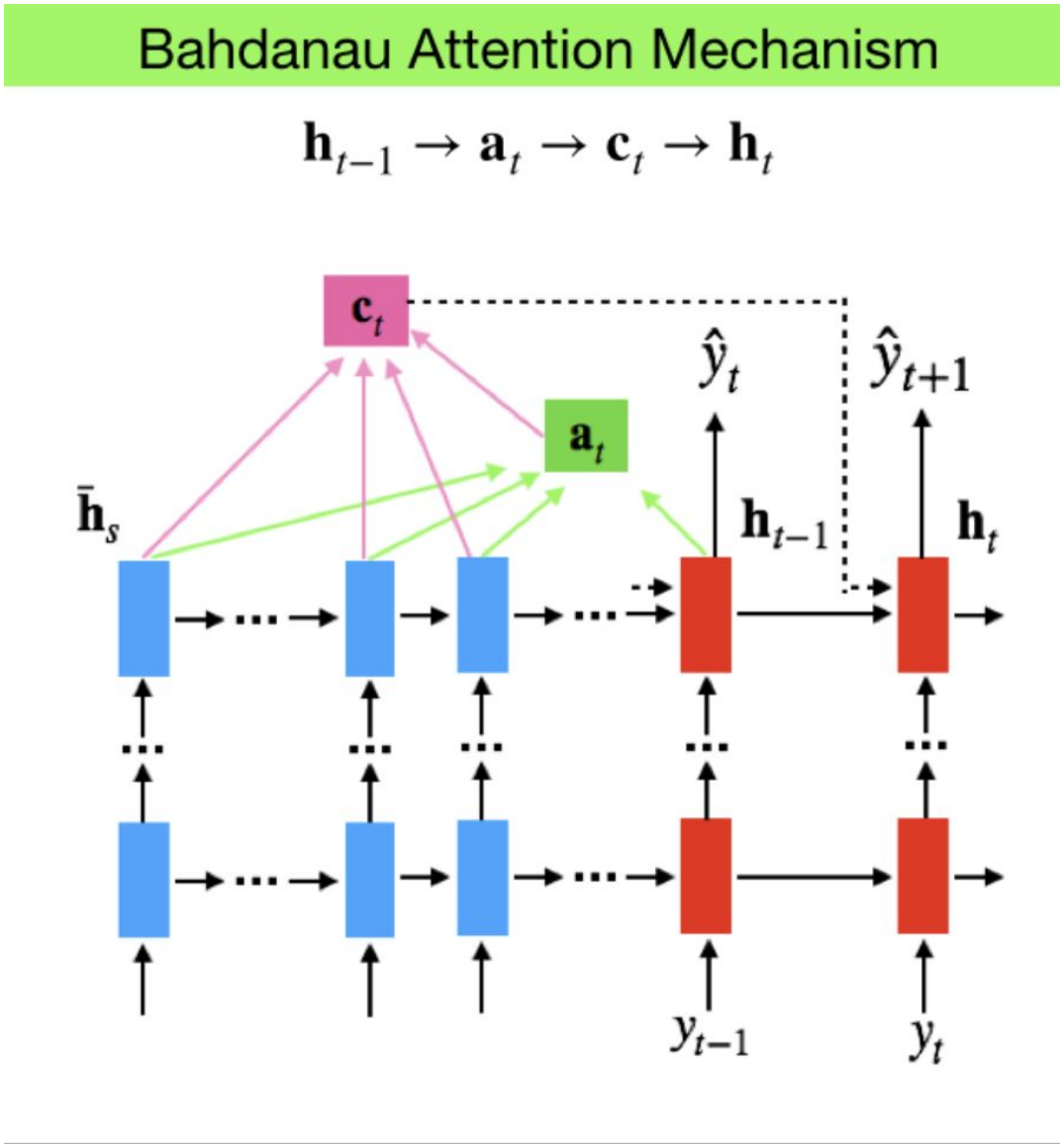
$$score(h_t, s_k) = h_t^T W s_k$$

- additive or multi-layer perceptron ("Bahdanau attention");

$$score(h_t, s_k) = w_2^t \cdot tahn(W_1[h_t, s_k])$$

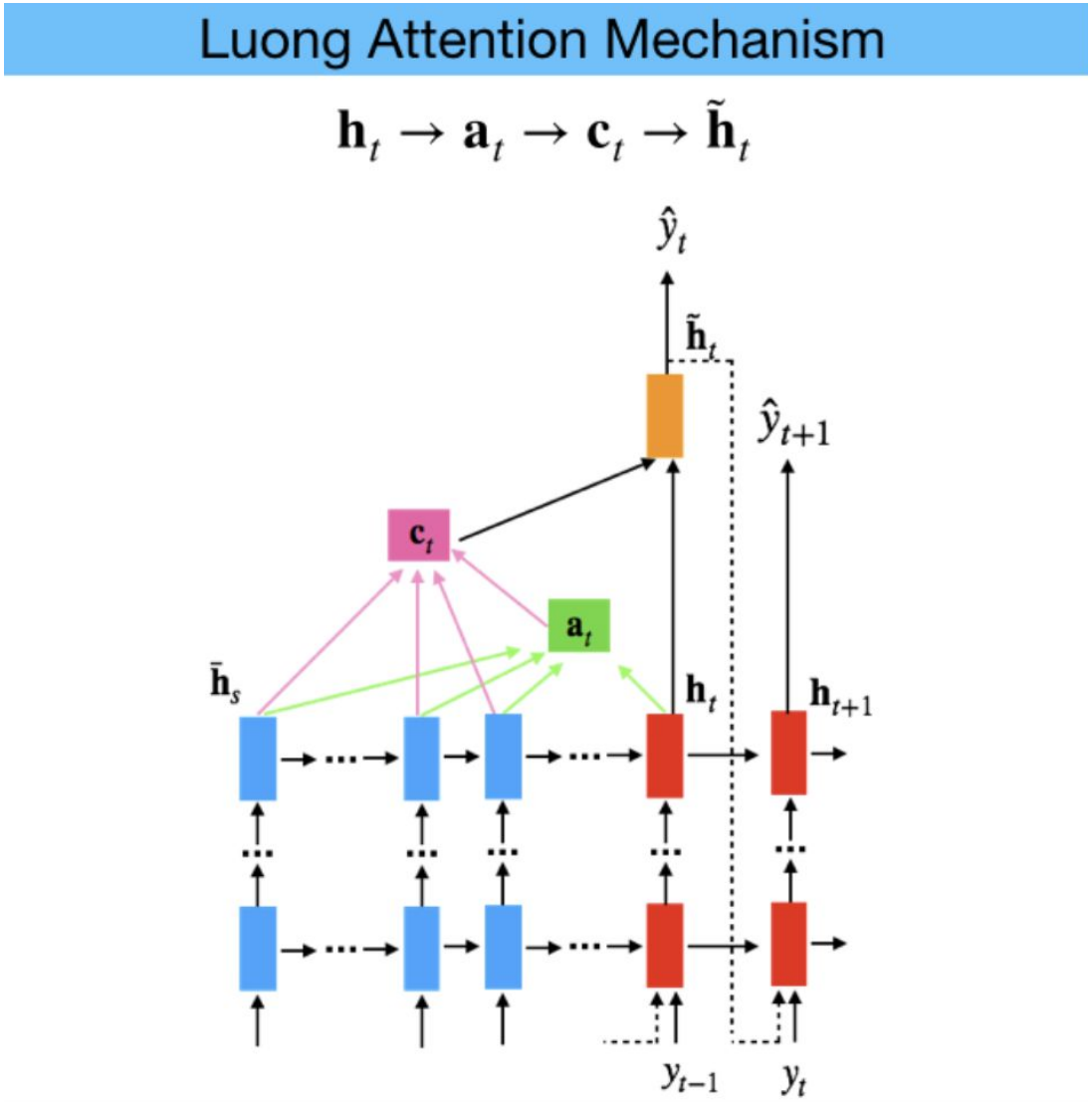
- actually any function you want =)

Early attention models



Bahdanau attention (paper Neural Machine Translation by Jointly Learning to Align and Translate);

Early attention models



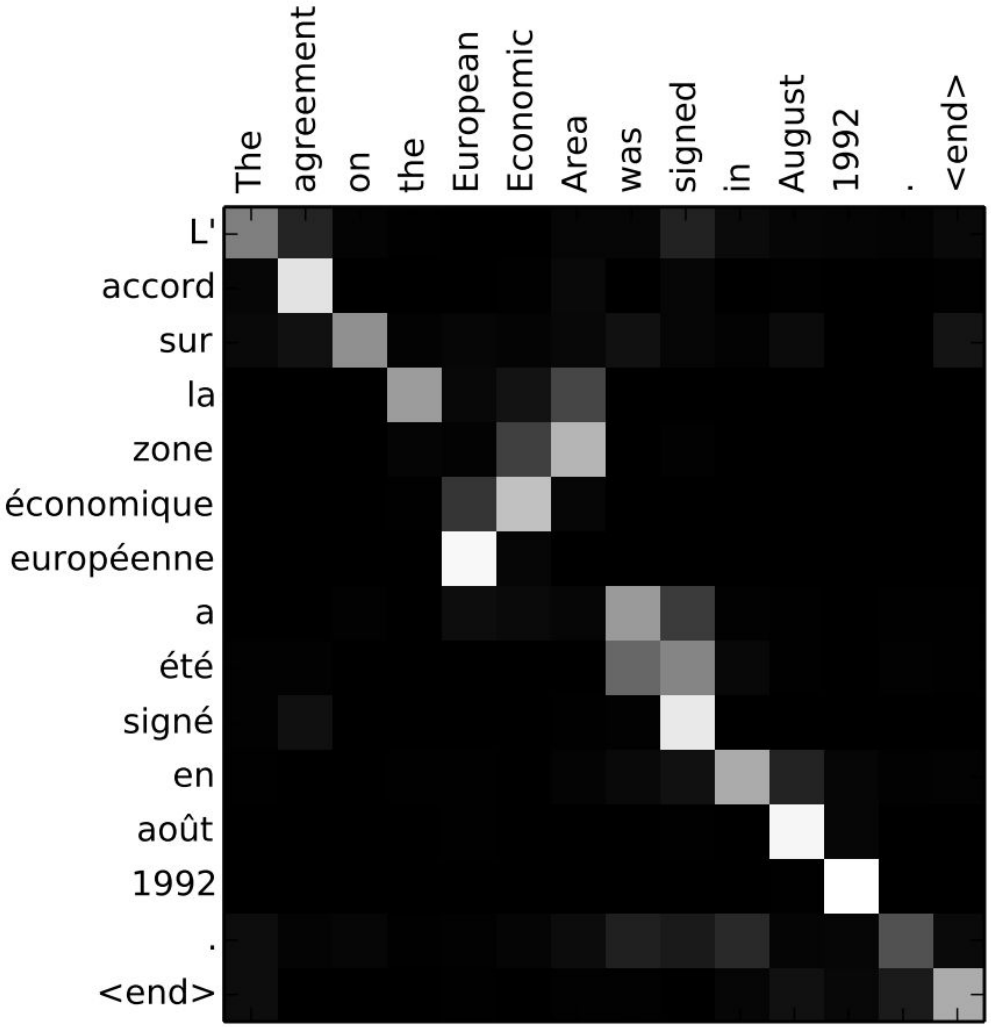
Luong attention

(the paper Effective Approaches to Attention-based Neura Machine Translation).

Attention is good (spoiler - all you need)

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - attention distribution shows what the decoder was focusing on
  - Alignment

Attention is good. Alignment



Transformers

Attention is everywhere

- Attention is all you need =) 2017

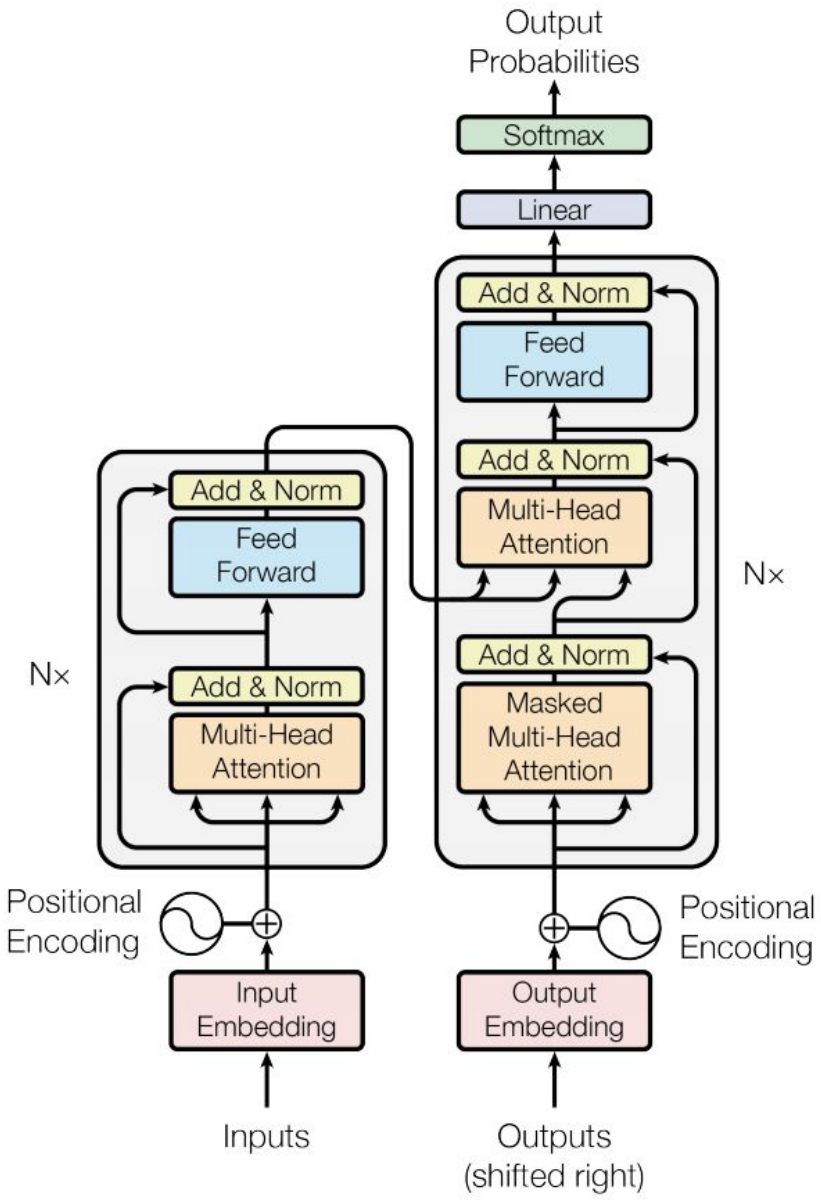
Previously:

- RNN encoder + RNN decoder, interaction via fix-sized vector
- RNN encoder + RNN decoder, interaction via attention

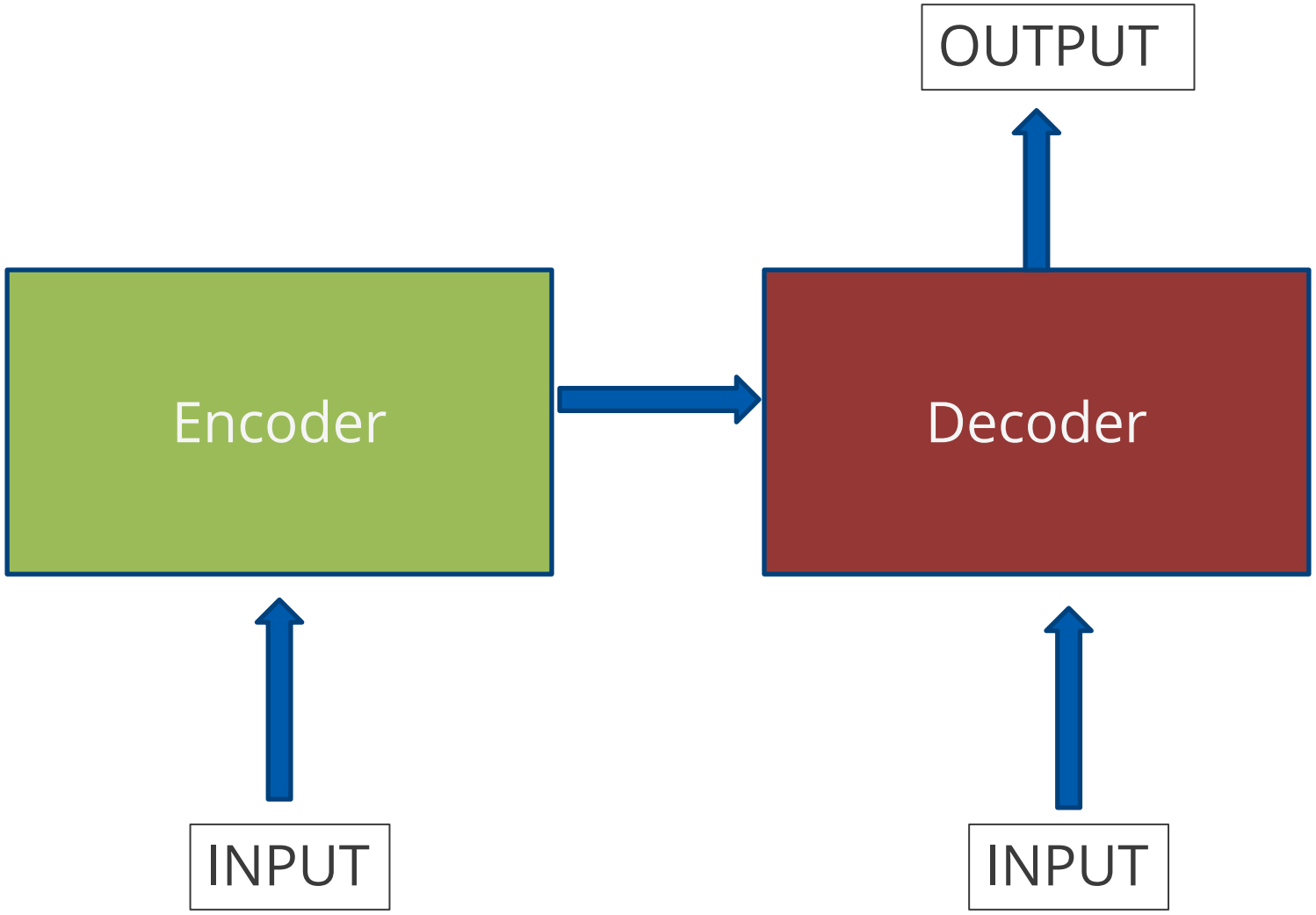
NOW: attention + attention + attention

- Number of unparallelizable operations does not increase sequence length.
- Maximum interaction distance:
  $O(1)$, since all words interact at every layer!

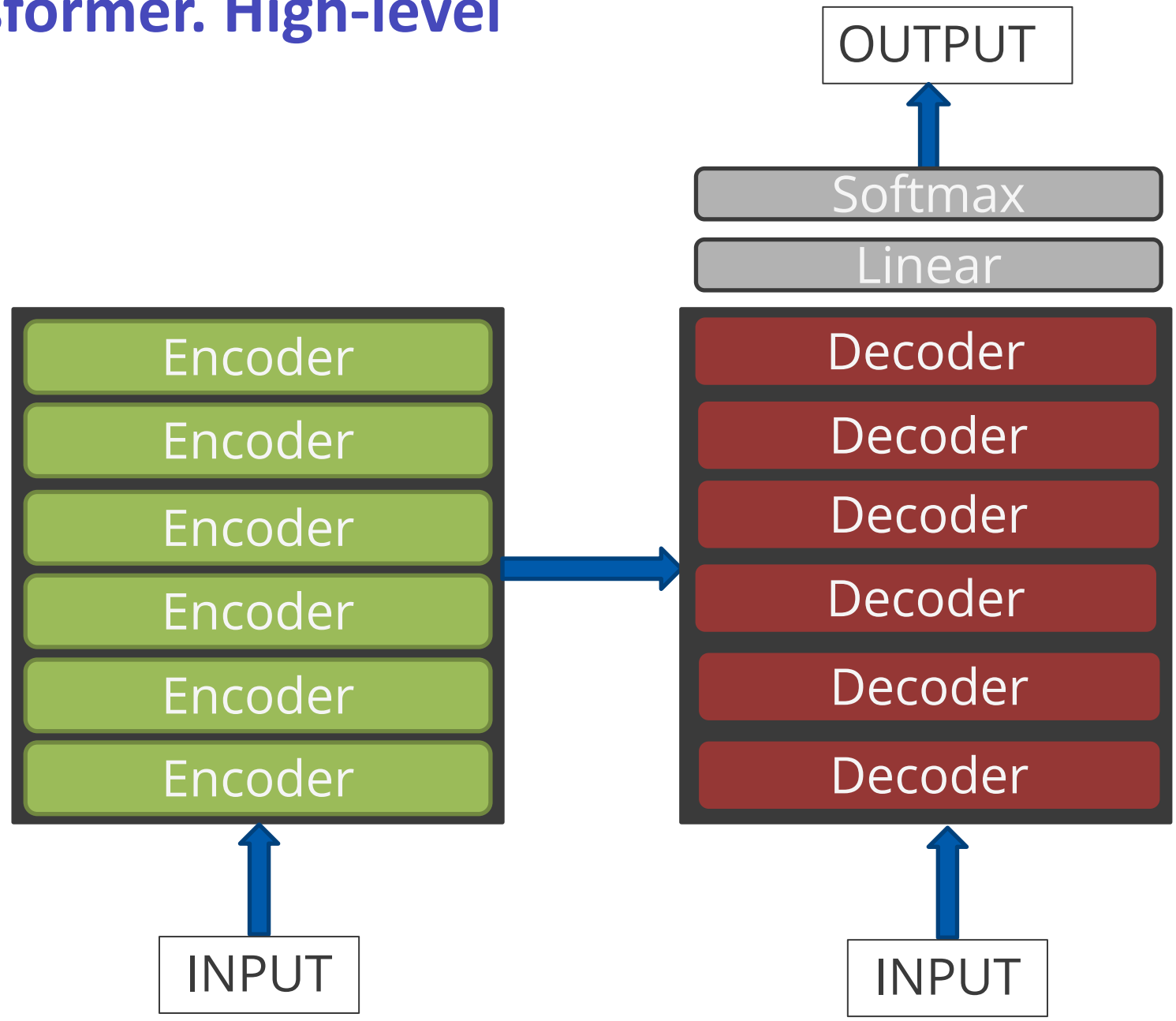
Transformer. High-level



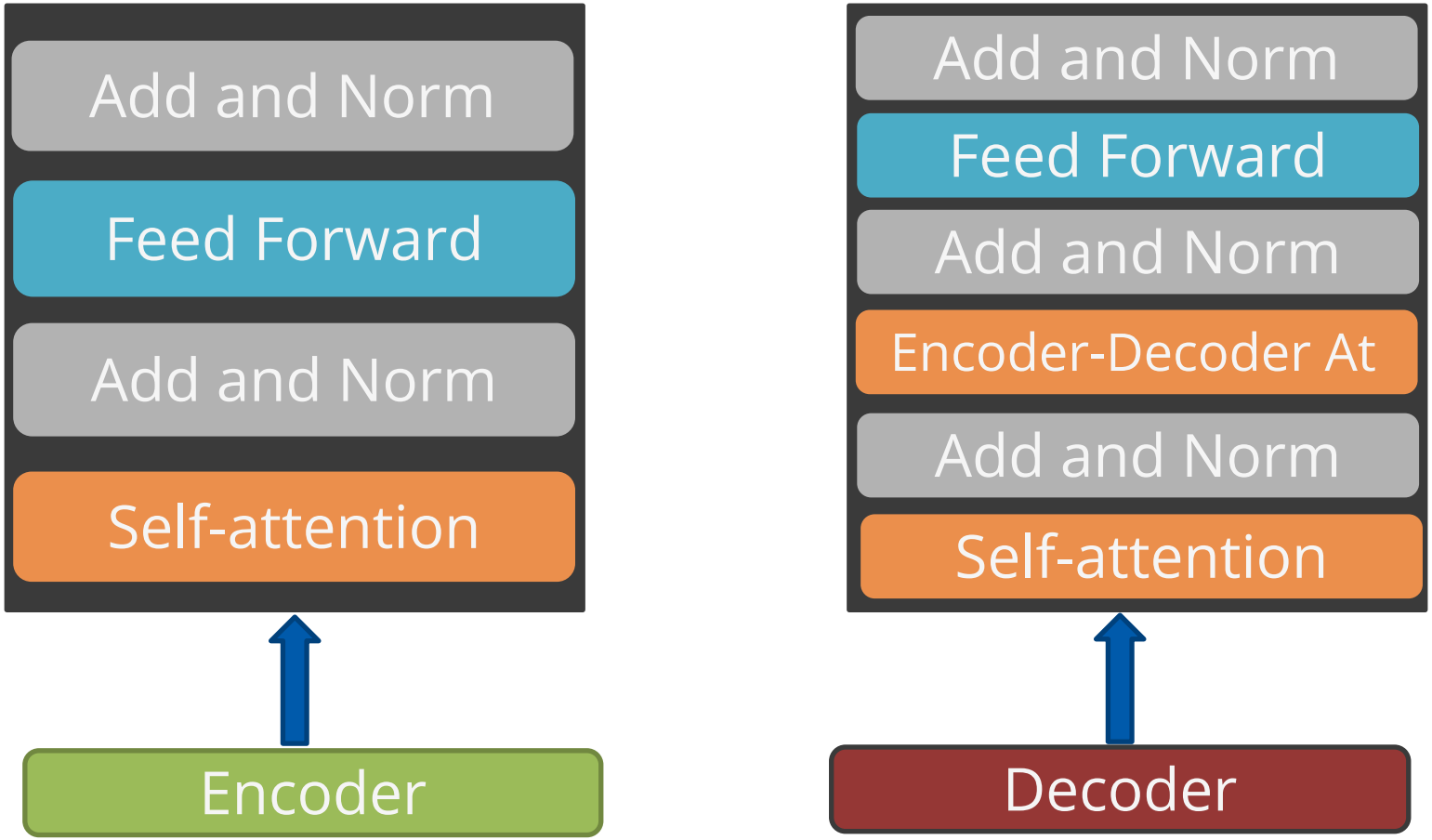
Transformer. High-level



Transformer. High-level



Transformer. High-level

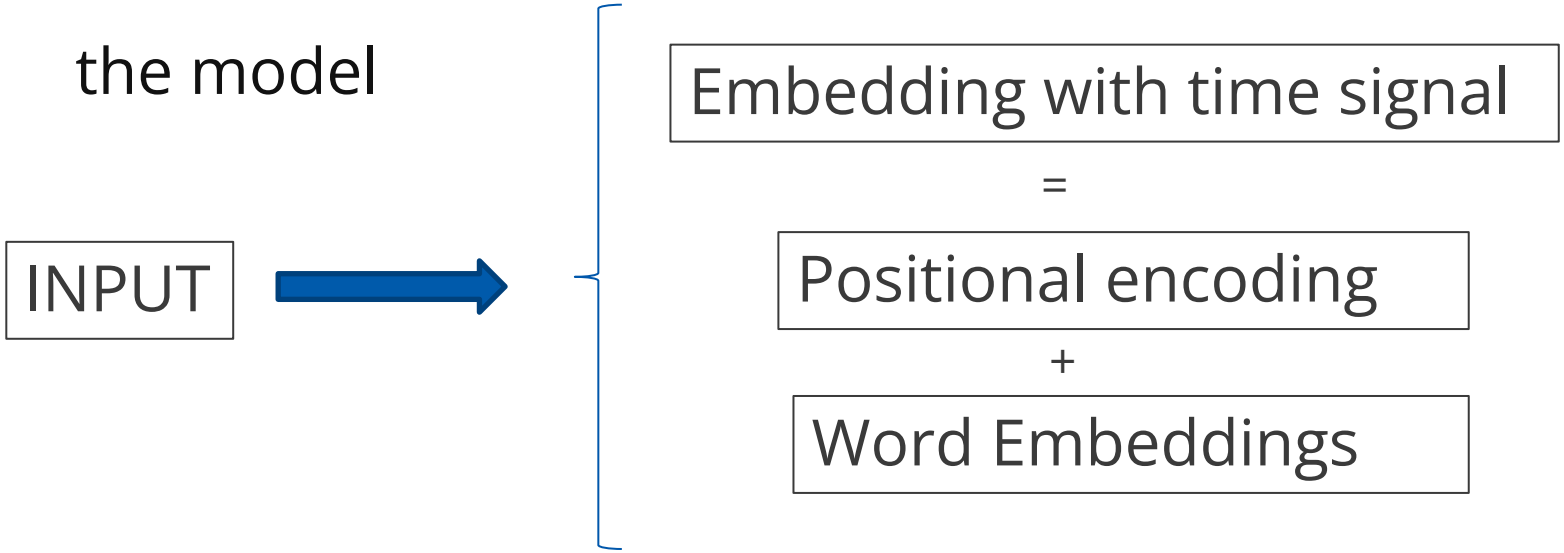


Transformer. Positional encoding

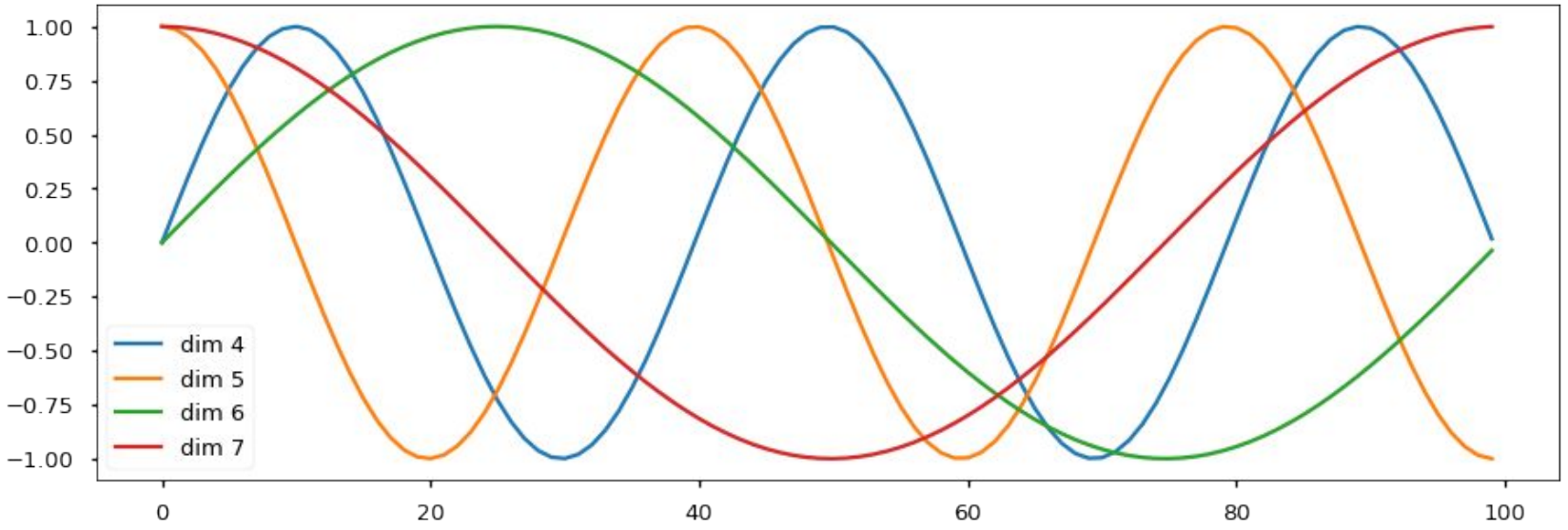
- **positional encoding**

  provides *order information* to

  the model



Transformer. Positional encoding



$$PE(i,\delta) = \begin{cases} \sin\left(\frac{i}{10000^{\frac{2\acute{\delta}}{d}}}\right), & if\,\delta = 2\acute{\delta} \\ \cos\left(\frac{i}{10000^{\frac{2\acute{\delta}}{d}}}\right), & if\,\delta = 2\acute{\delta} + 1 \end{cases}$$

Transformer. Self-attention

Decoder-Encoder - one current decoder state looked at all encoder states

Self-attention - each state looks at each other states

Self-attention:

- tokens interact with each other
- each token "looks" at other tokens
- gathers context
- updates the previous representation of "self"

Transformer. Self-attention

Self-attention is the part of the model where tokens

interact with each other. How?

Query, Key and Value vectors

Input embedding  X W_k = Key vector: where the query looks

Input embedding  X W_q = Query vector: from where attention is looking

Input embedding  X W_v = Value vector: attention output is here

Transformer. Self-attention

Each vector receives three representations:

- query - asking for information;
- key - saying that it has some information;
- value - giving the information.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

These matrices allow different aspects of the x vectors to be used/emphasized in each of the three roles

Attention matches the key and query by assigning a value to the place the key is most likely to be.

Transformer. Masked self-attention

Decoder has different self-attention

Masked self-attention

- we generate one token at a time

  (we don’t have all source tokens at once like in encoder):

  during generation, we don't know which tokens we'll generate in future.
- to enable parallelization we forbid the decoder to look ahead - future tokens are masked out (setting them to -inf) before the softmax step in the self-attention calculation

Transformer. Multi-head attention

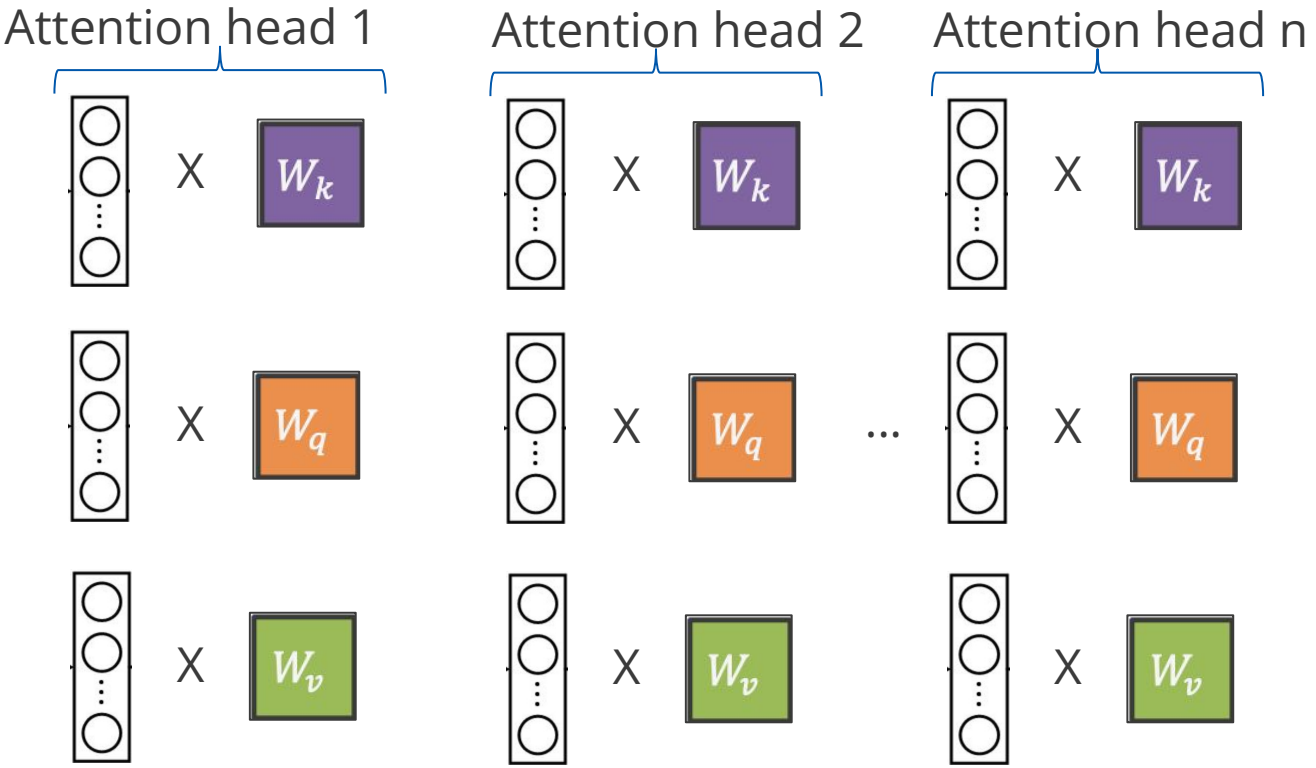
We need to know different relationships between tokens in a sentence:

- syntactic relationships,
- lexical preferences,
- order,
- grammar issues like case or gender agreement
- etc.

Instead of having one attention mechanism, multi-head attention has several "heads" which work independently and focus on different things.

Heads works independently

Transformer. Multi-head attention



With multi-headed attention, we maintain separate Q/K/V weight matrices for each head, get different Q/K/V matrices.

Multiple them by an additional weights matrix WO.

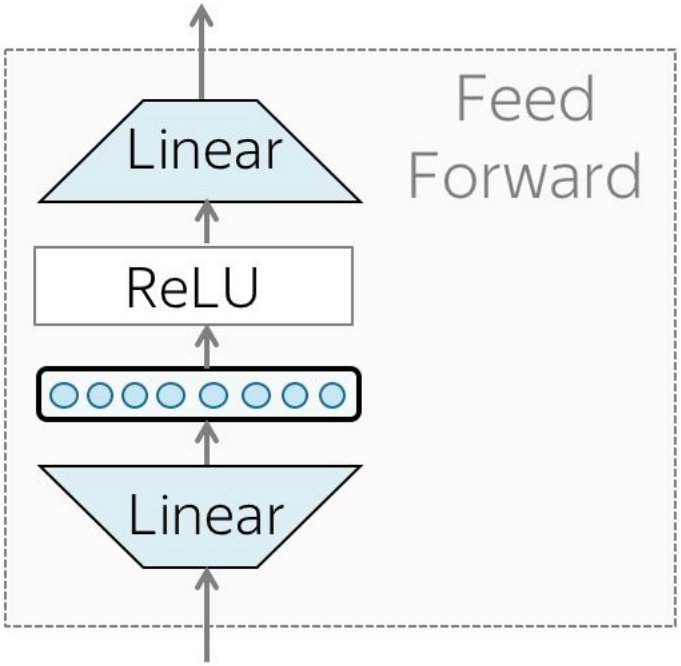
$$\text{MultiHead}\left(\mathbf{Q}, \mathbf{K}, \mathbf{V}\right) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right)\mathbf{W}^O$$

$$\text{where head}_i = \text{Attention}\left(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V\right)$$

Transformer

Feed-forward blocks

each layer has a feed-forward network block:
two linear layers with ReLU non-linearity between them



$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Transformer

Residual connection

Residual connections => add a block's input to its output

They ease the gradient flow through a network and allow stacking a lot of layers

* In the Transformer in the "Add & Norm" part, the "Add" part stands for the residual connection.

https://arxiv.org/abs/1512.03385

Transformer

Layer Normalization

Normalizes vector representation of each example in batch.

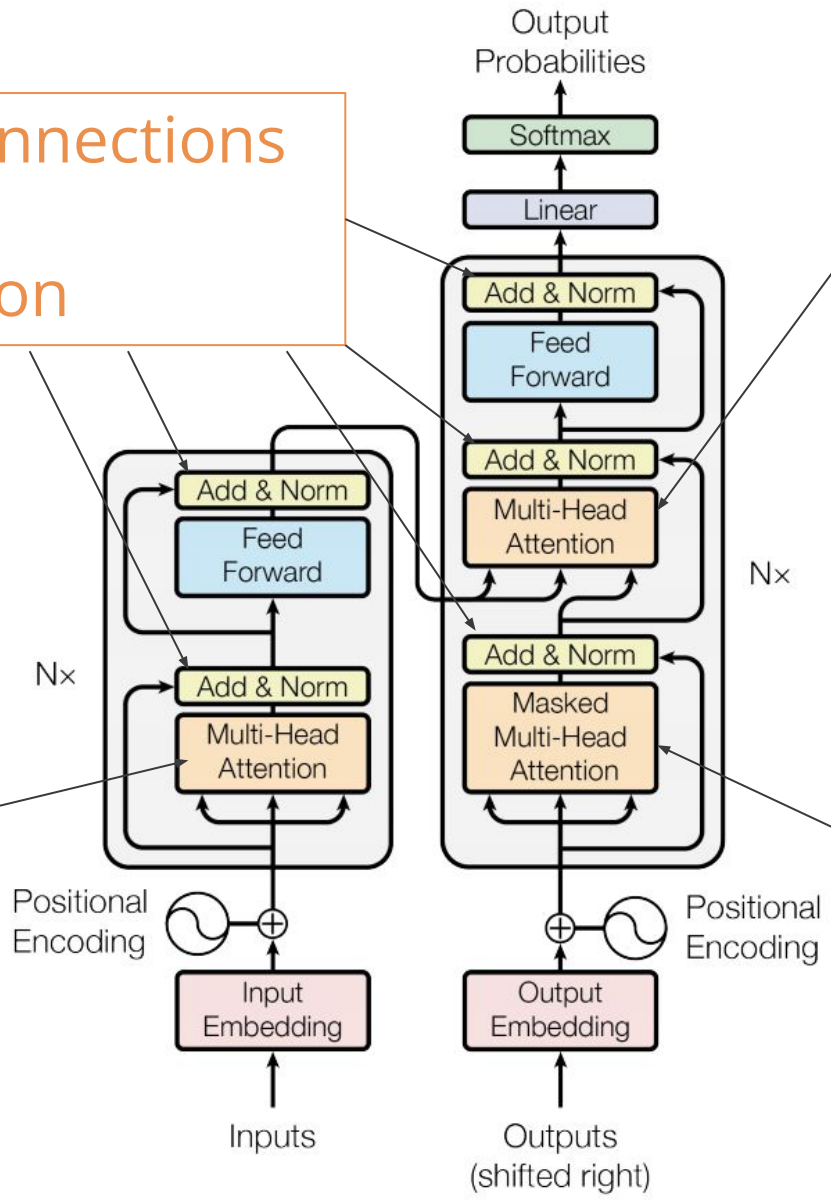
Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation within each layer

+ Train faster
+ Improves convergence stability

Transformer

Residual connections and layer normalization

Decoder-Encoder attention: Q – from decoder states, K and V from encoder



Encoder self-attention: Q,K,V are computed from encoder states

Decoder (masked) self-attention: Q,K,V are computed from decoder states

References

Attention is all you need
Transformer survey

High-level
Jay Alammar:

- Transformers - http://jalammar.github.io/illustrated-transformer/
- Seq2seq with attention - https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

AI Summer https://theaisummer.com/transformer/

Deeper

- Stanford lectures https://www.youtube.com/watch?v=IxQtK2SjWWM&ab_channel=StanfordUniversitySchoolofEngineering
- Lena Voita https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html
- https://lilianweng.github.io/lil-log/2020/04/07/the-transformer-family.html

With code:

- https://nlp.seas.harvard.edu/2018/04/03/attention.html
- https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/text/transformer.ipynb#scrollTo=1kLCla68EloE