The Transformer: motivation, original architecture and attention mechanism

Outline

- Motivation for Transformer
- Attention (from RNNs to Attention is all you need)
- The original Transformer architecture
- Do we need a transformer for our problem?

Short history of machine translation.
Motivation for transformers

History of machine translation

- 1950s: Russian→English MT (Cold War)
  - *Rule-based systems using bilingual dictionaries*

History of machine translation

- 1950s: Russian→English MT (Cold War)
  - *Rule-based systems using bilingual dictionaries*
- 1990s-2010s: **Statistical** MT (SMT)
  - *Mainly **Phrase-Based** MT (PBMT)*
  - *Learns from (large) **sentence-aligned** bilingual corpora*
  - *Consists of separate very complex components, learnt separately*
  - *Developed by large groups for decades, doesn't generalize to new language pairs*

History of machine translation

- 1950s: Russian→English MT (Cold War)
  - *Rule-based systems using bilingual dictionaries*
- 1990s-2010s: **Statistical** MT (SMT)
  - *Mainly **Phrase-Based** MT (PBMT)*
  - *Learns from (large) **sentence-aligned** bilingual corpora*
  - *Consists of separate very complex components, learnt separately*
  - *Developed by large groups for decades, doesn't generalize to new language pairs*
- Since 2014: **Neural** MT (NMT)
  - *Single NN learnt **end-to-end***
  - *Learns from (large) **sentence-aligned** bilingual corpora*
  - *A few (good) student / month to implement*
  - *better quality of translation than SMT systems developed for decades*
- Google Translate: NMT, Yandex Translate: NMT+PBMT

History of machine translation: IBM models

Original French sentence

English translation

$$\Pr(\mathbf{e}|\mathbf{f}) = \frac{\Pr(\mathbf{e})\Pr(\mathbf{f}|\mathbf{e})}{\Pr(\mathbf{f})}$$

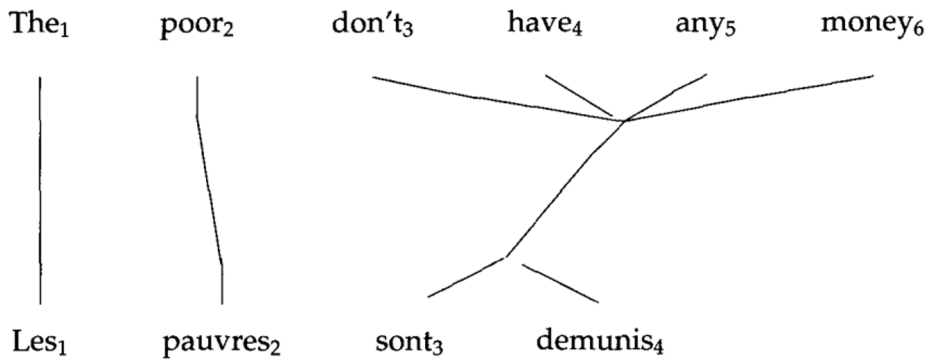
English N-gram language model

Translation model

This probability can be discarded

$$\hat{\mathbf{e}} = \underset{\mathbf{e}}{\operatorname{argmax}} \Pr(\mathbf{e})\Pr(\mathbf{f}|\mathbf{e})$$

Key model component: learned word alignment



$$\Pr(\mathbf{f}|\mathbf{e}) = \sum_{\mathbf{a}} \Pr(\mathbf{f}, \mathbf{a}|\mathbf{e})$$

$$\Pr(f, a|e) = \Pr(f|a, e)\Pr(a|e)$$

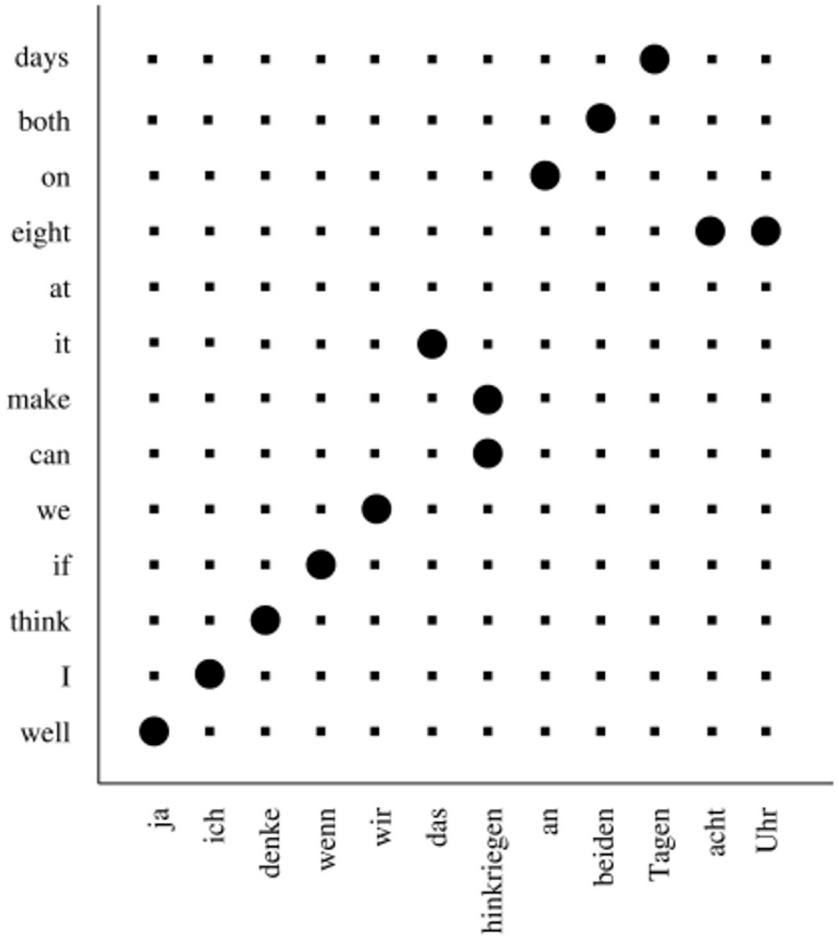
Phrase-by-phrase translation

Alignment model

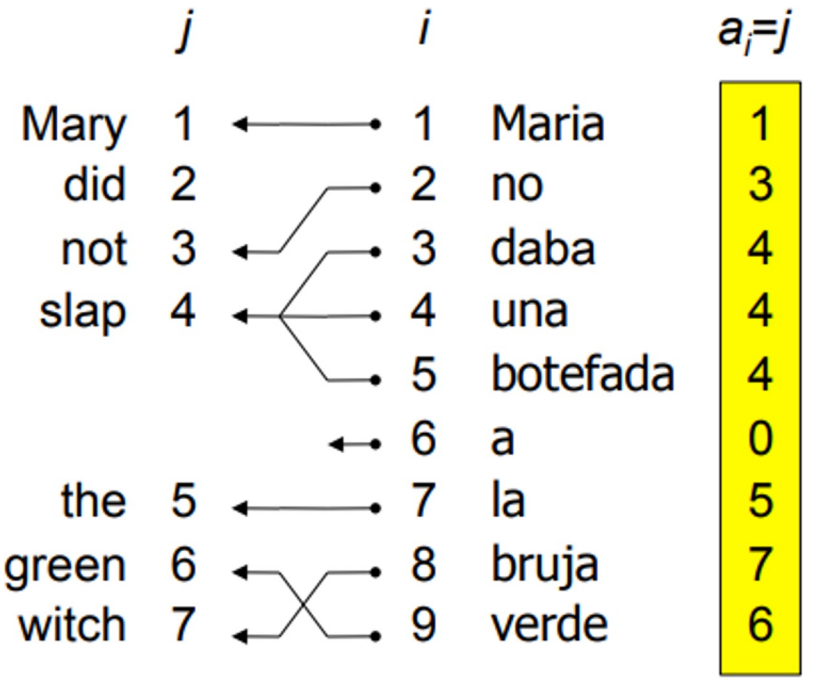
Source: The Mathematics of Statistical Machine Translation: Parameter Estimation, Brown et al, 1993

What is an alignment?

A matrix of matches between words



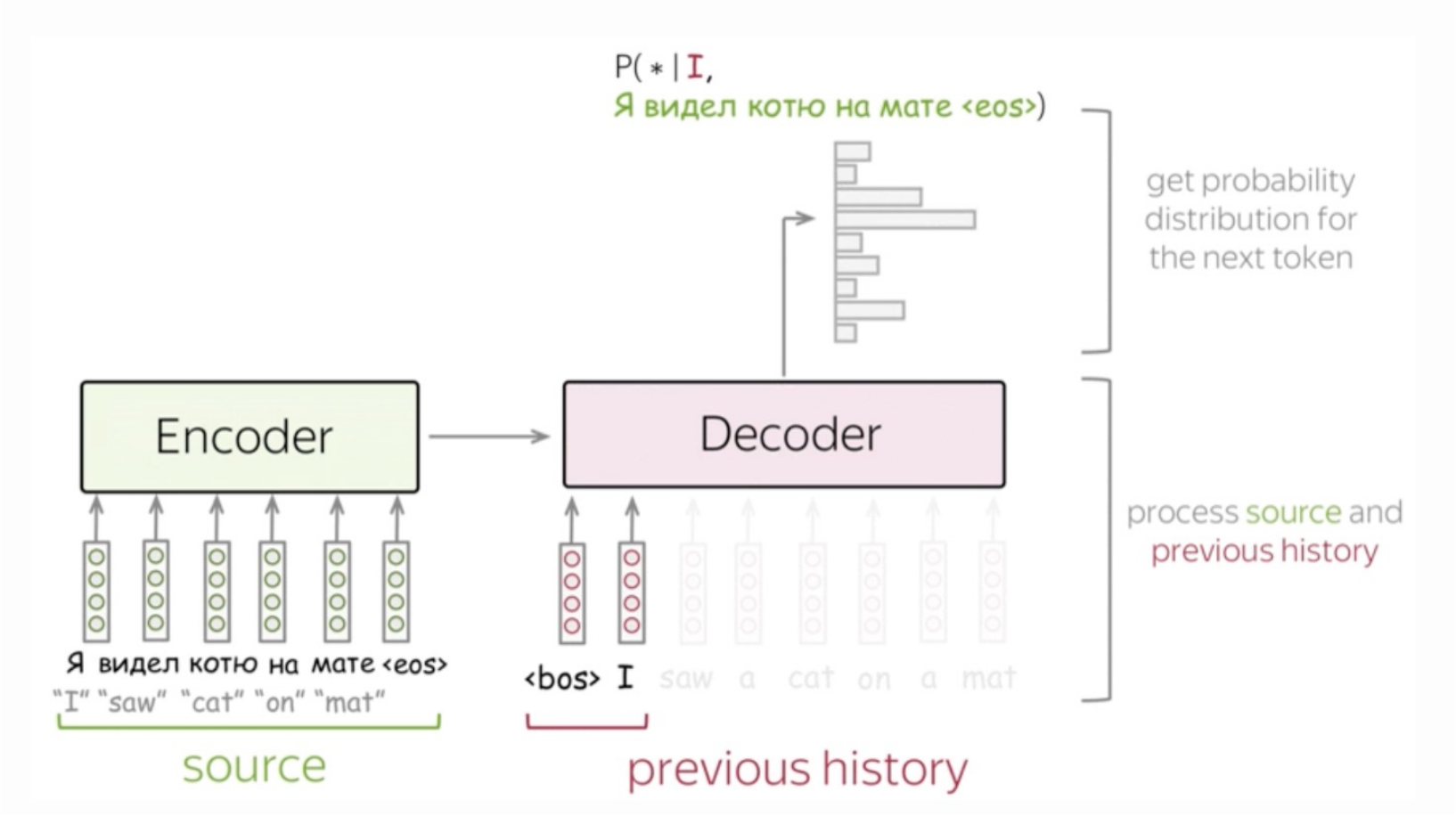
A vector of matched word*



*This approach is simpler, but it cannot describe one-to-many or many-to-many alignments

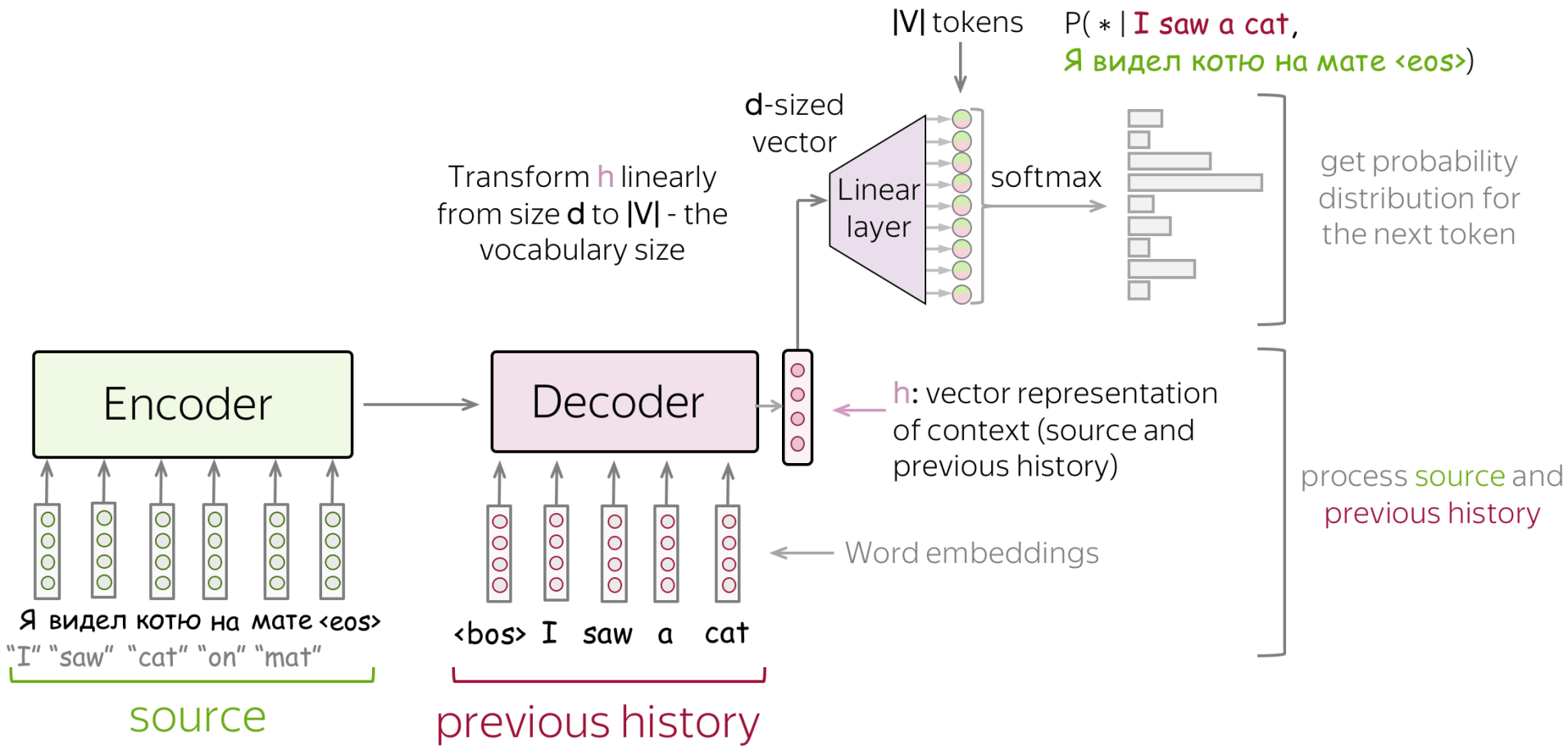
Images source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1162/syllabus.shtml

Neural machine translation: seq2seq



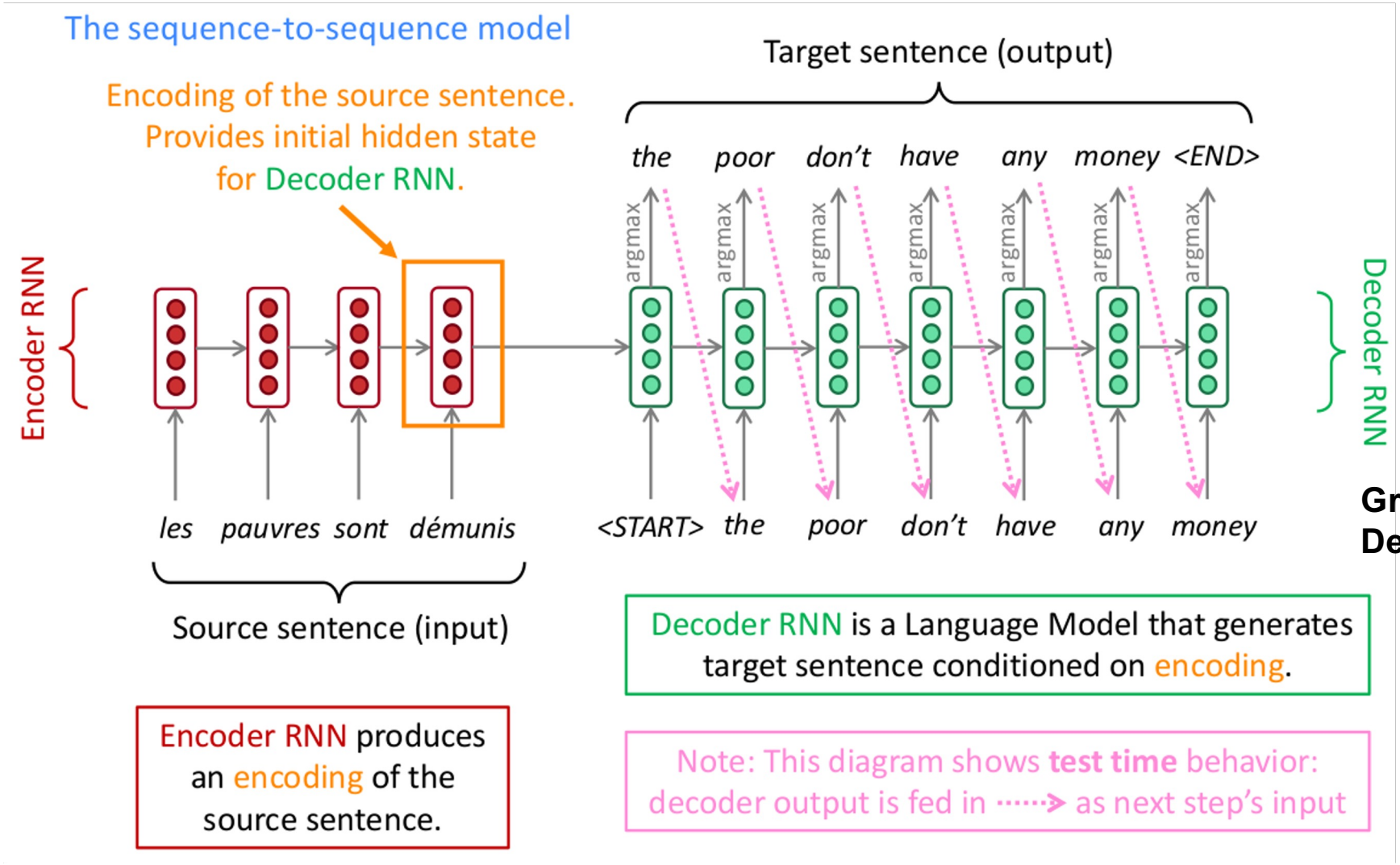
https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Neural machine translation: seq2seq



https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Neural machine translation: seq2seq



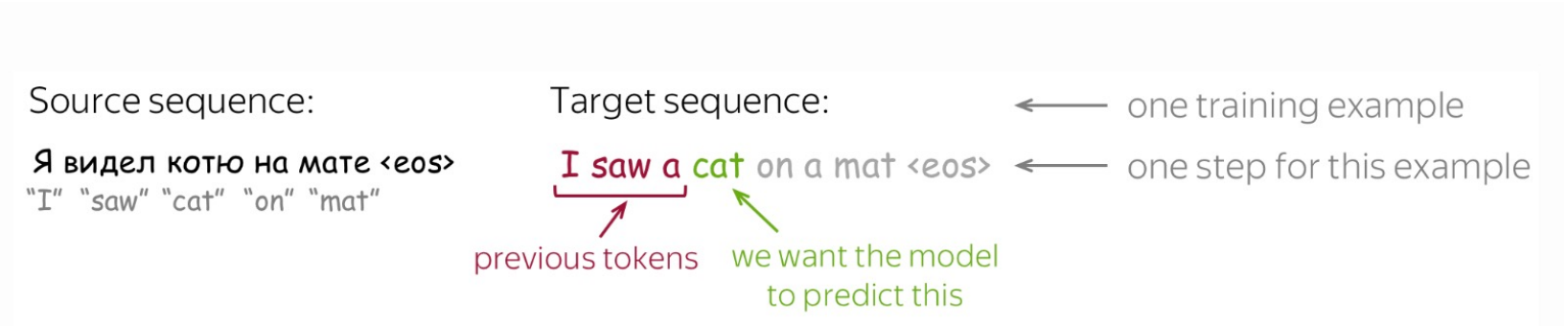
Sequence to Sequence Learning with Neural Networks, Sutskever et al, 2014
Socher, Manning. Cs224n, 2017

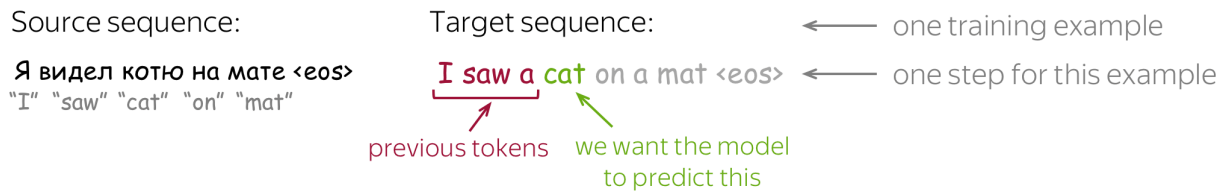
Cross-entropy loss

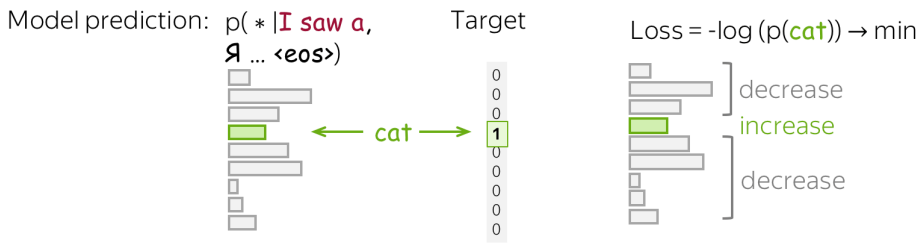
The standard loss function is the cross-entropy loss:

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

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

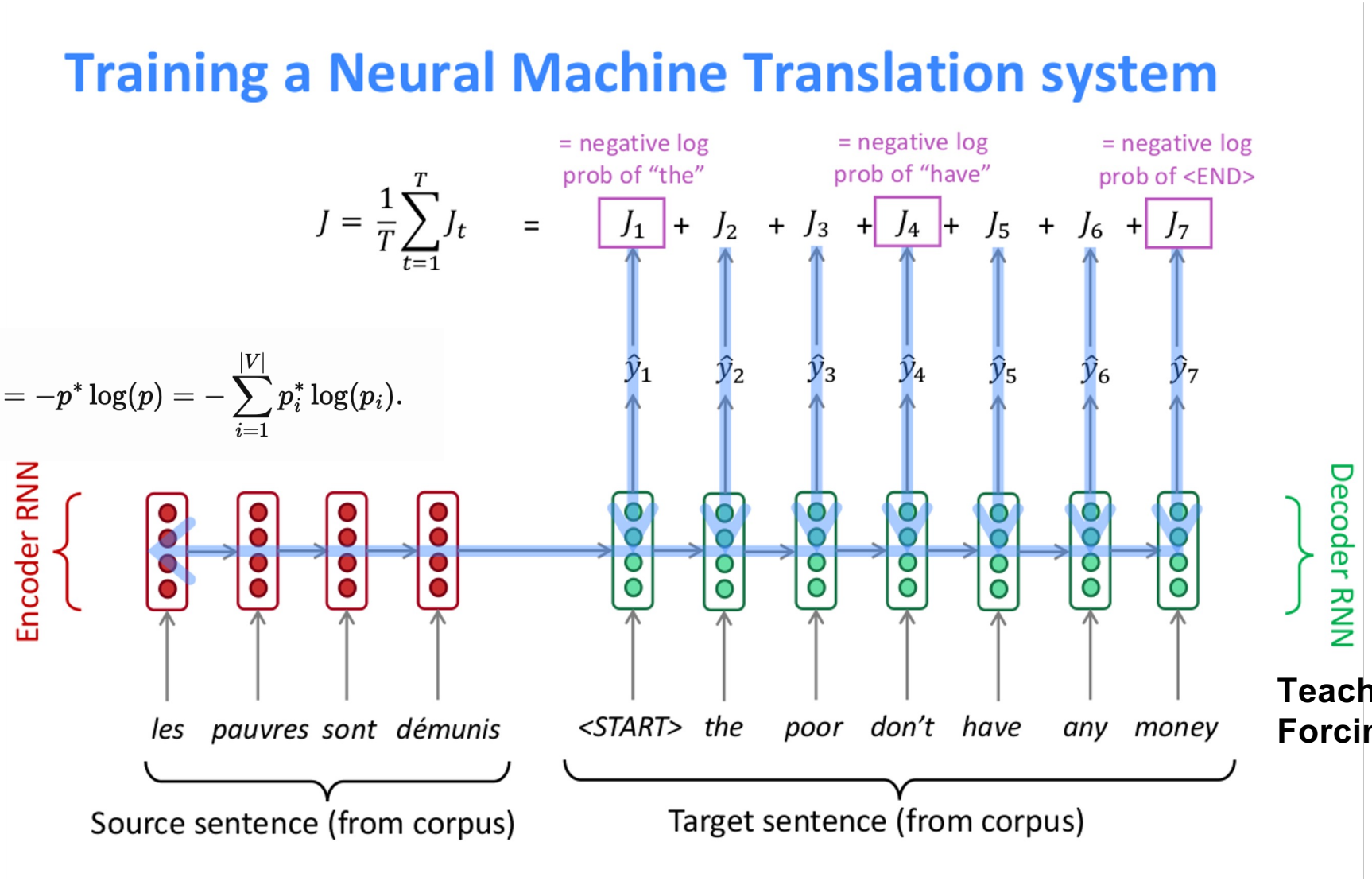






https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

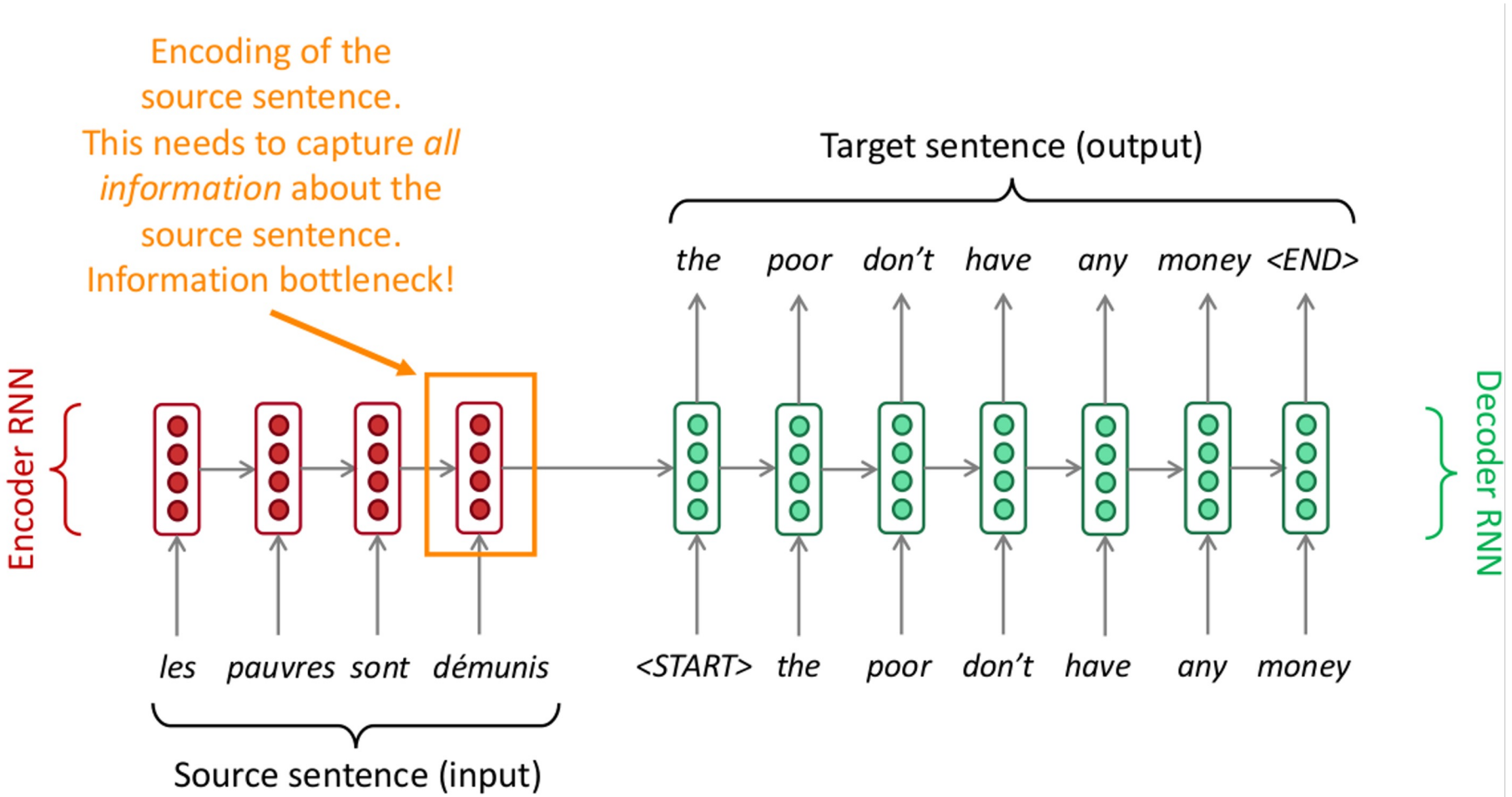
Neural machine translation: seq2seq loss



Sequence to Sequence Learning with Neural Networks, Sutskever et al, 2014

Socher, Manning. Cs224n, 2017

Seq2seq bottleneck problem



Socher, Manning. Cs224n, 2017

Attention

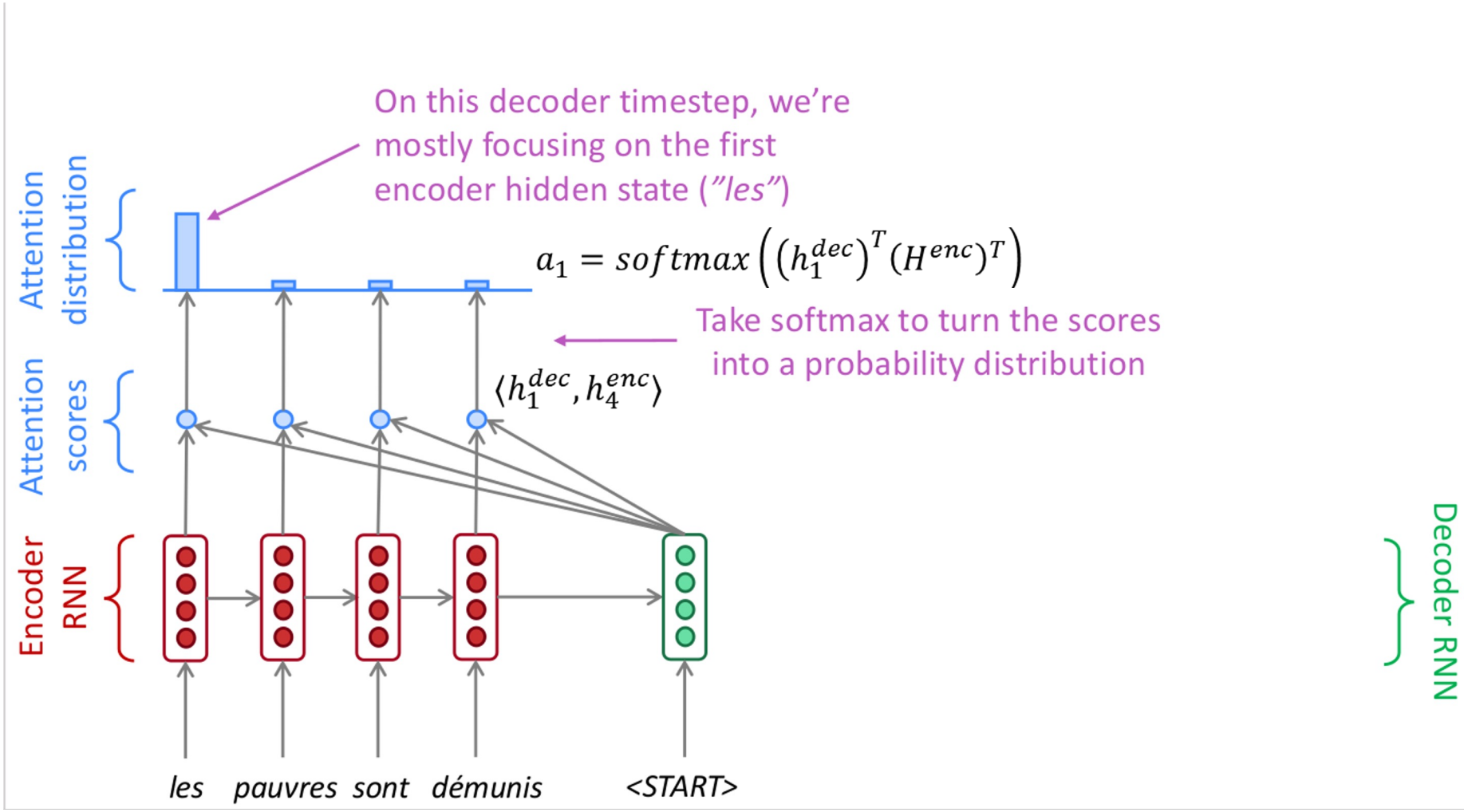
Attention:

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

Core idea:

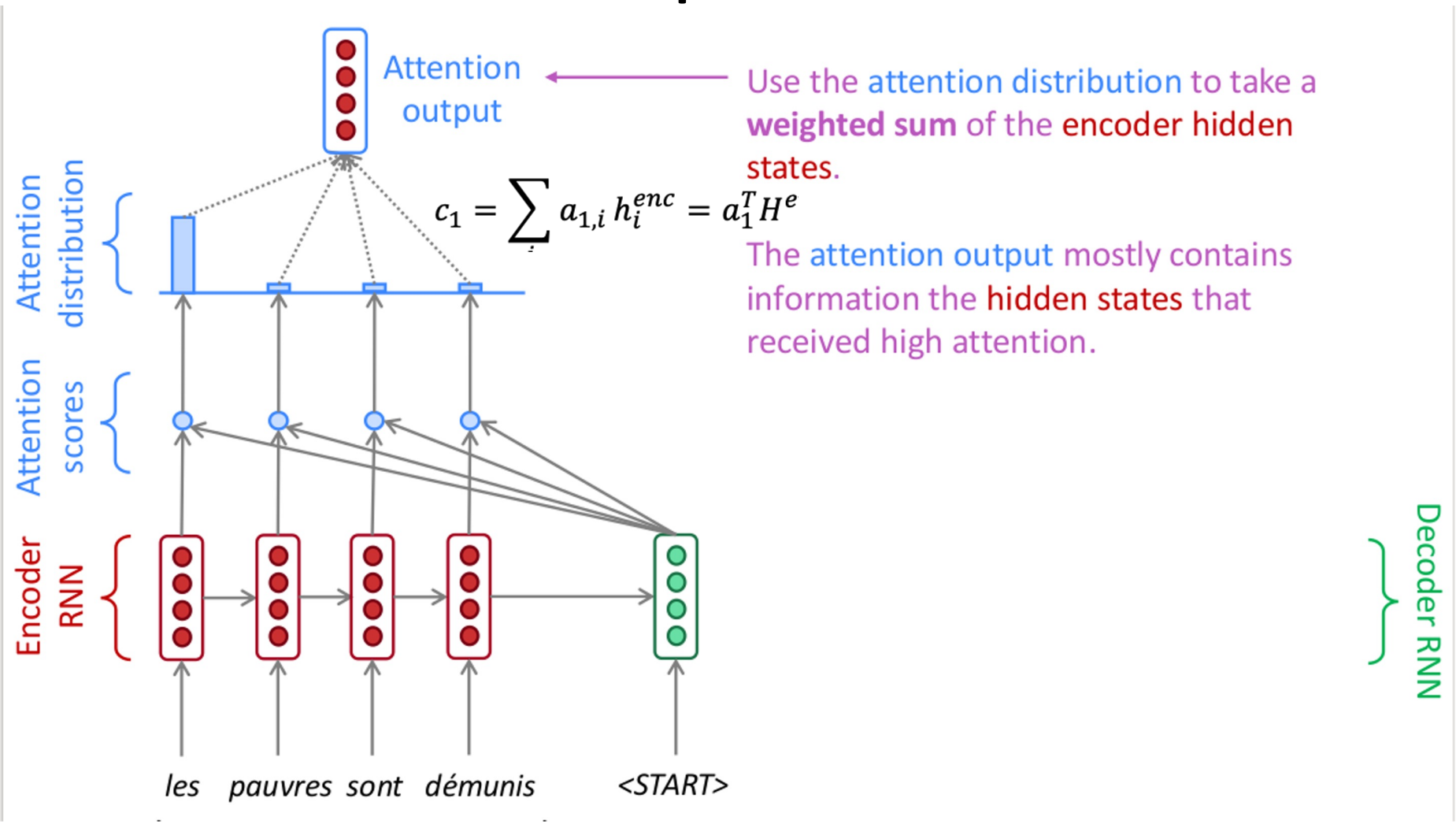
On each step of the decoder, use direct connection to the encoder to focus on a particular part of the source sequence

Attention: Example



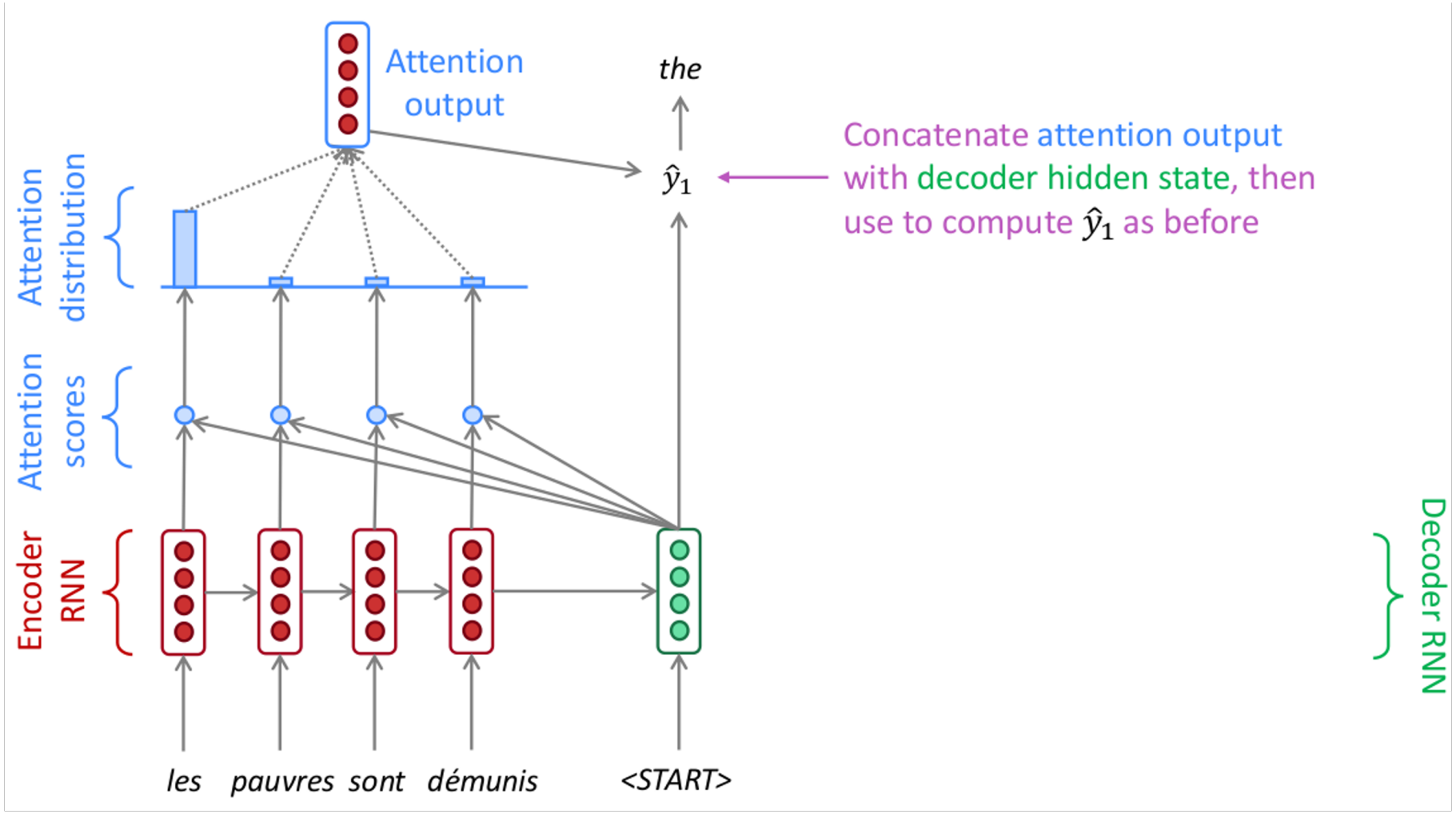
Socher, Manning. Cs224n, 2017

Attention: Example



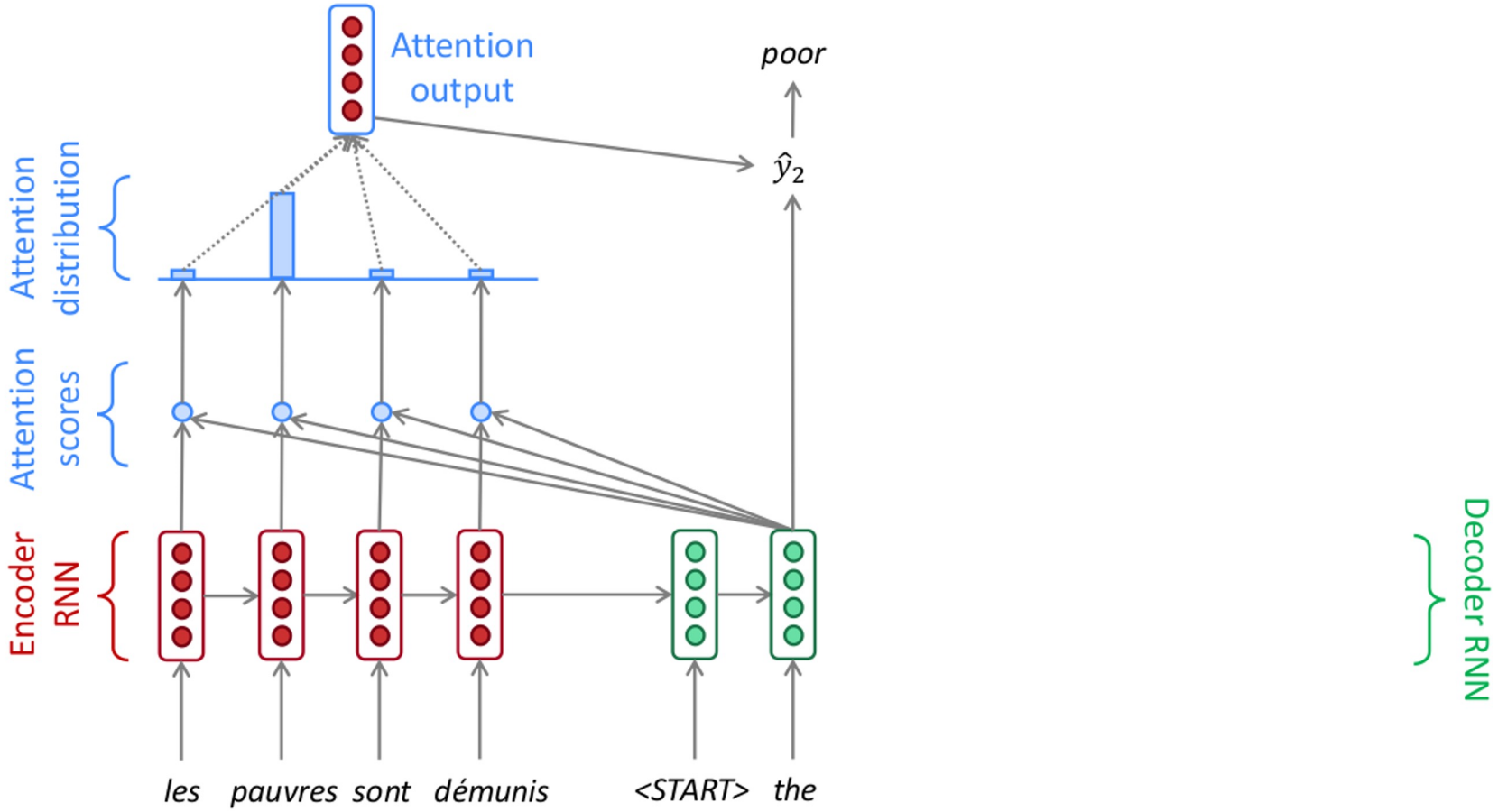
Socher, Manning. Cs224n, 2017

Attention: Example



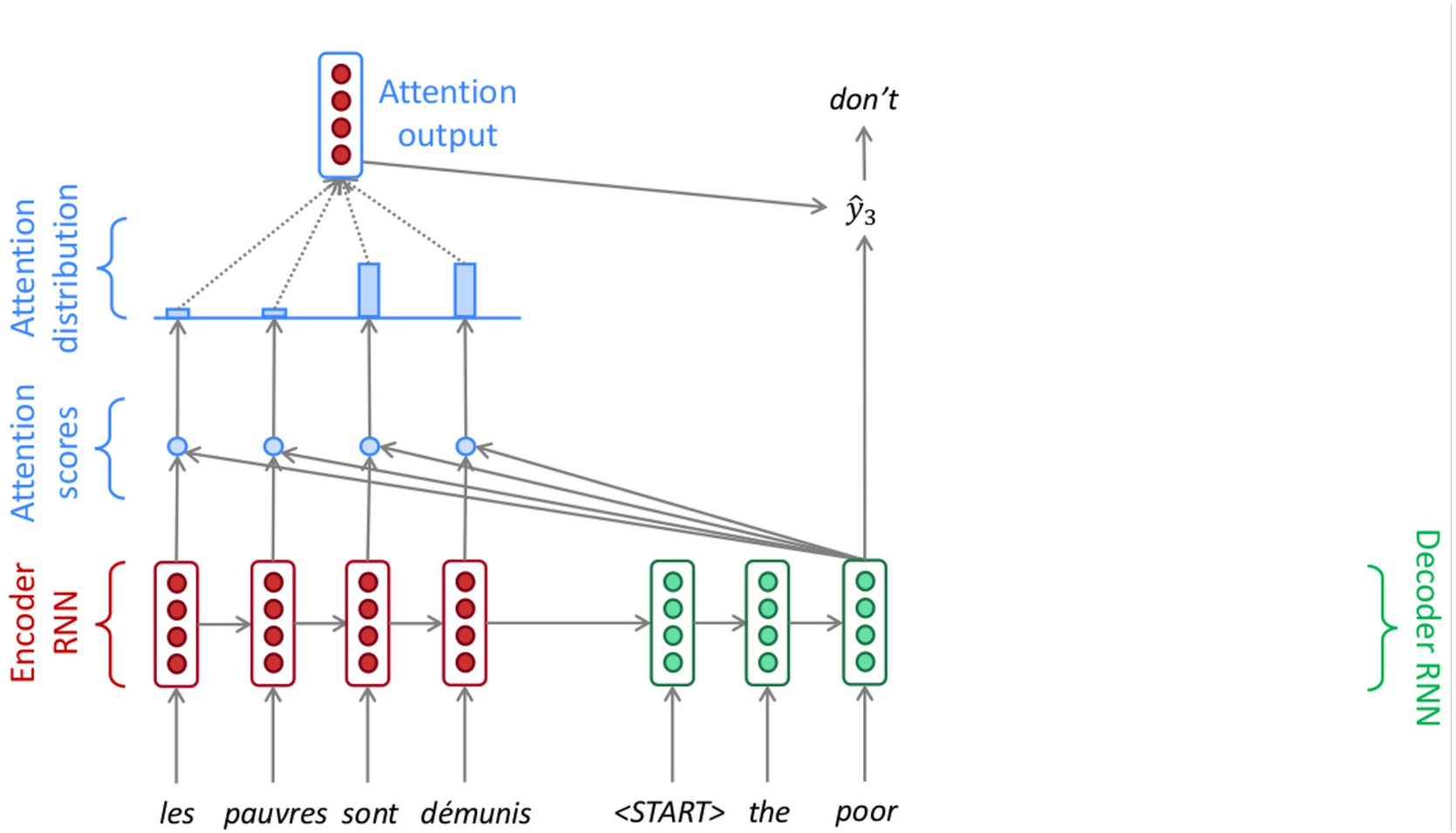
Socher, Manning. Cs224n, 2017

Attention: Example



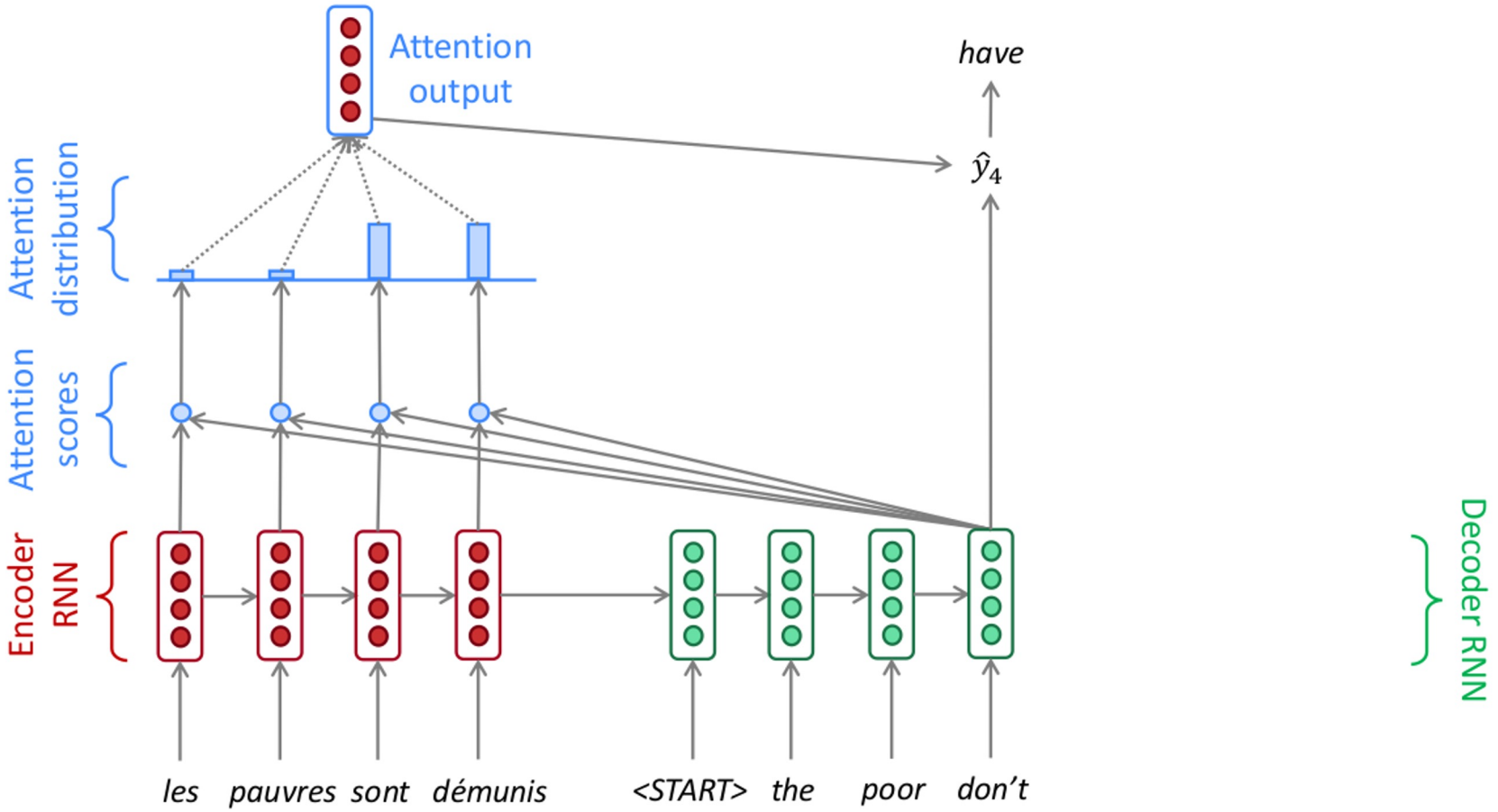
Socher, Manning. Cs224n, 2017

Attention: Example



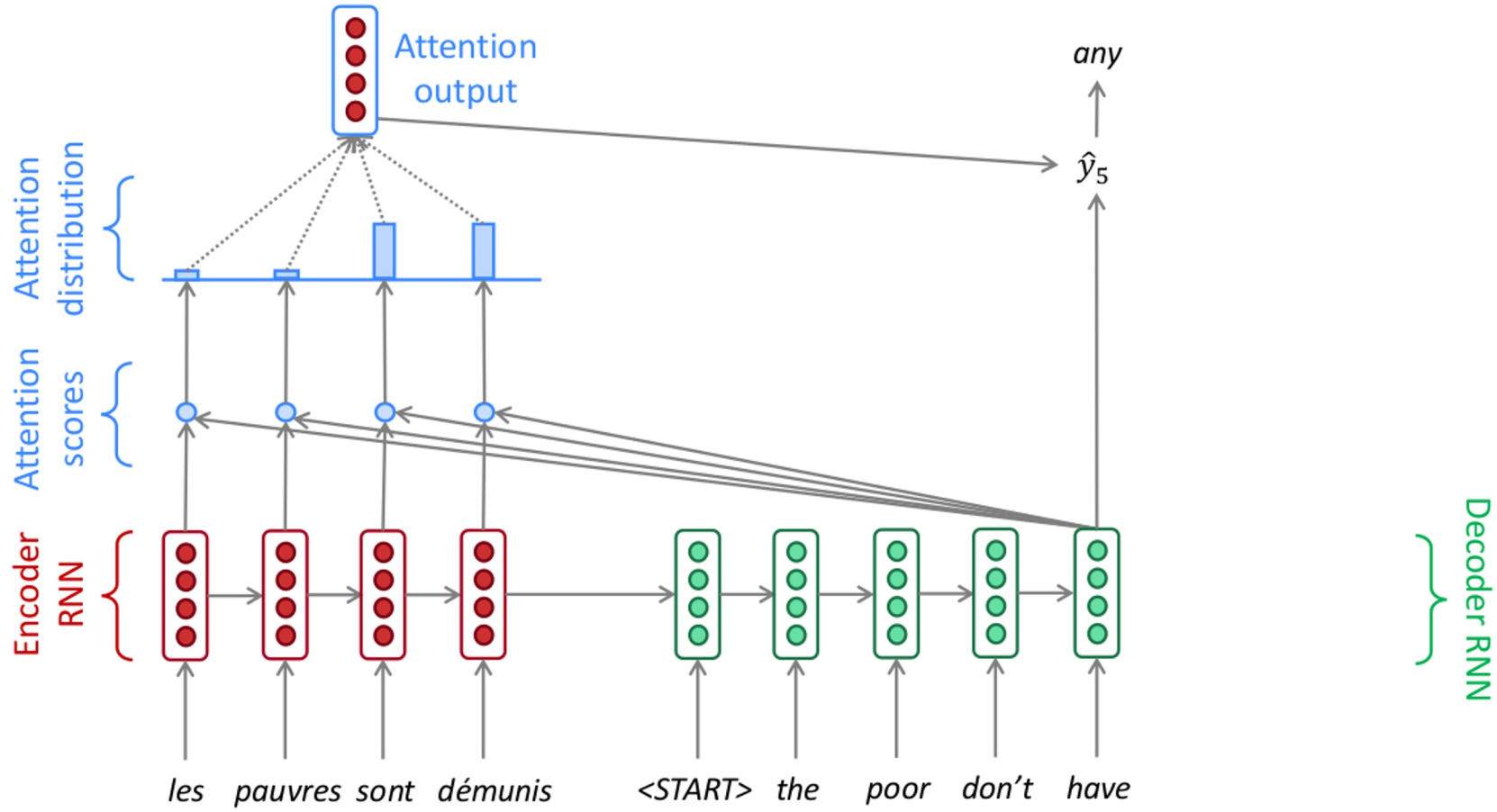
Socher, Manning. Cs224n, 2017

Attention: Example



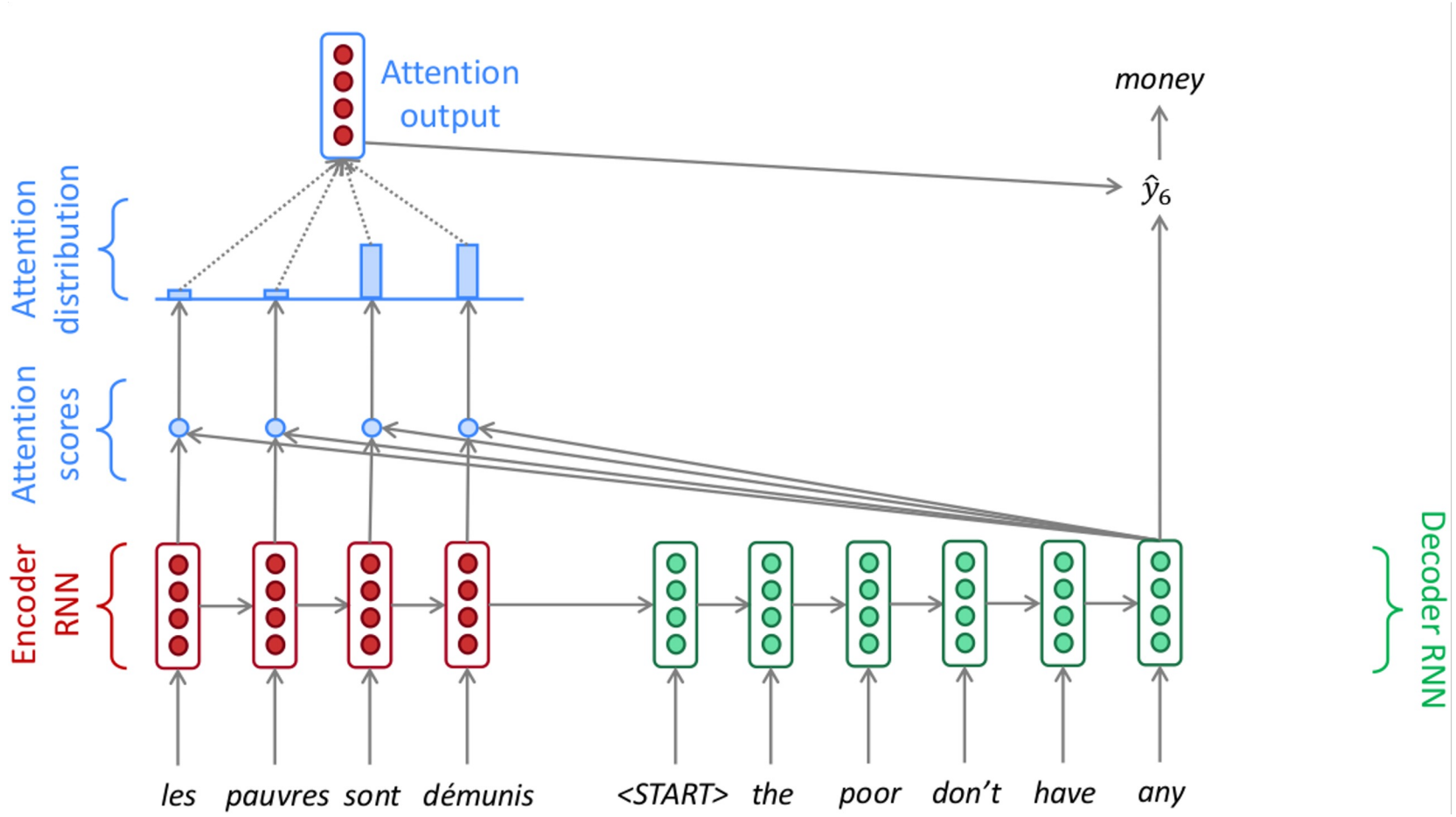
Socher, Manning. Cs224n, 2017

Attention: Example



Socher, Manning. Cs224n, 2017

Attention: Example



Socher, Manning. Cs224n, 2017

Attention

Results:

- Solves **information bottleneck** problem, acts like an additional memory
- Helps **gradient propagation** from encoder to decoder, especially in long sequences
- Better interpretability

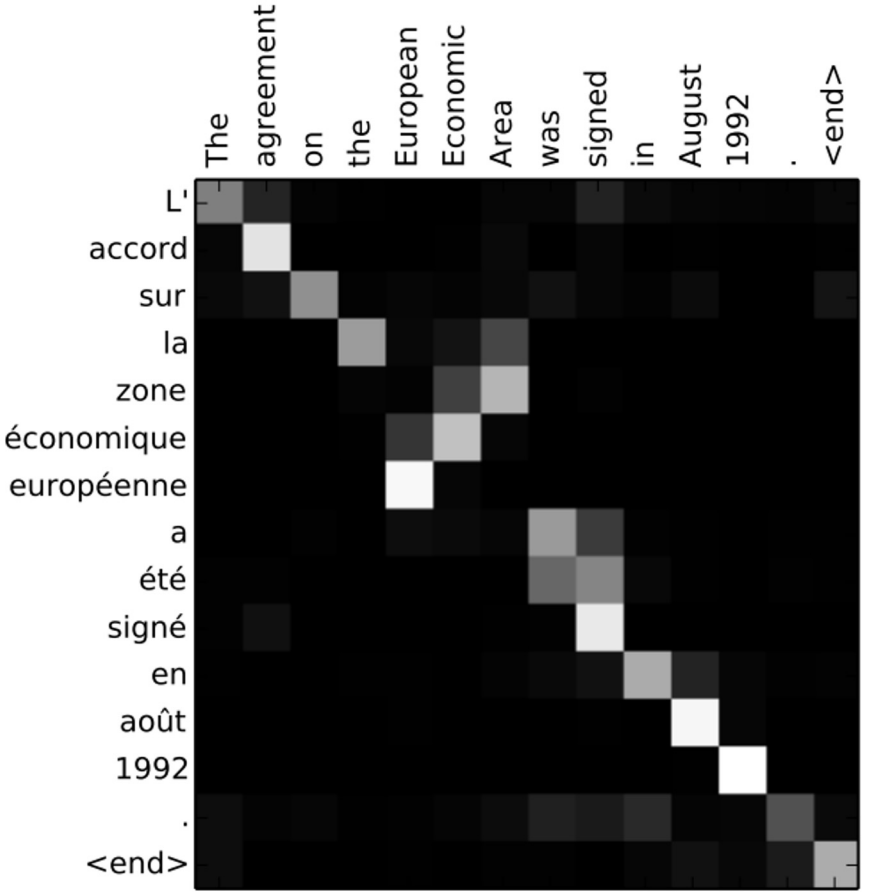
Britz et al., 2017. Massive Exploration of Neural Machine Translation Architectures:

- “we found that the attention-based models exhibited significantly larger gradient updates to decoder states throughout training. This suggests that the attention mechanism acts more like a ”weighted skip connection” that optimizes gradient flow than like a ”memory” that allows the encoder to access source states, as is commonly stated in the literature”

Attention

Better interpretability

- Networks learn alignment as a byproduct of translation
- We can look at which fragments were translated to which ones



Bahdanau et al. Neural Machine Translation by Jointly Learning to Align and Translate, 2015

Attention

- Attention is a new basic layer type (along with feed-forward, convolutional and recurrent)
- Works with variable length inputs (texts)
    - *like RNNs, CNNs*
    - *unlike feed-forward*
    - Used in SOTA models in lots of tasks (question answering, image captioning, ...)

The Transformer

Get rid of RNNs in MT?

- RNNs are slow, because not parallelizable over timesteps

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.

Vaswani et al, 2017. Attention is all you need.

The Transformer

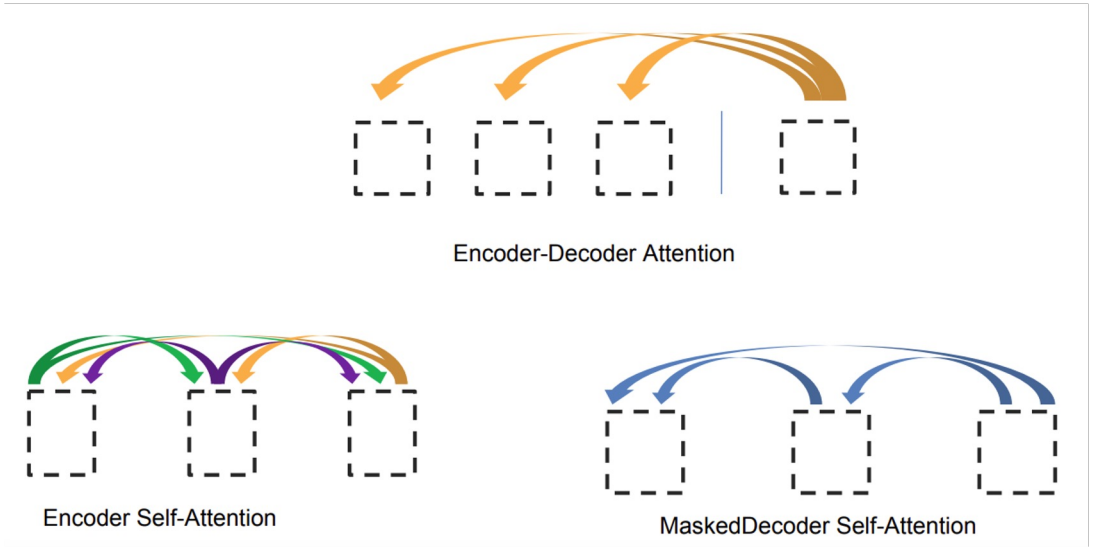
Attention is all you need =) 2017

Previously:

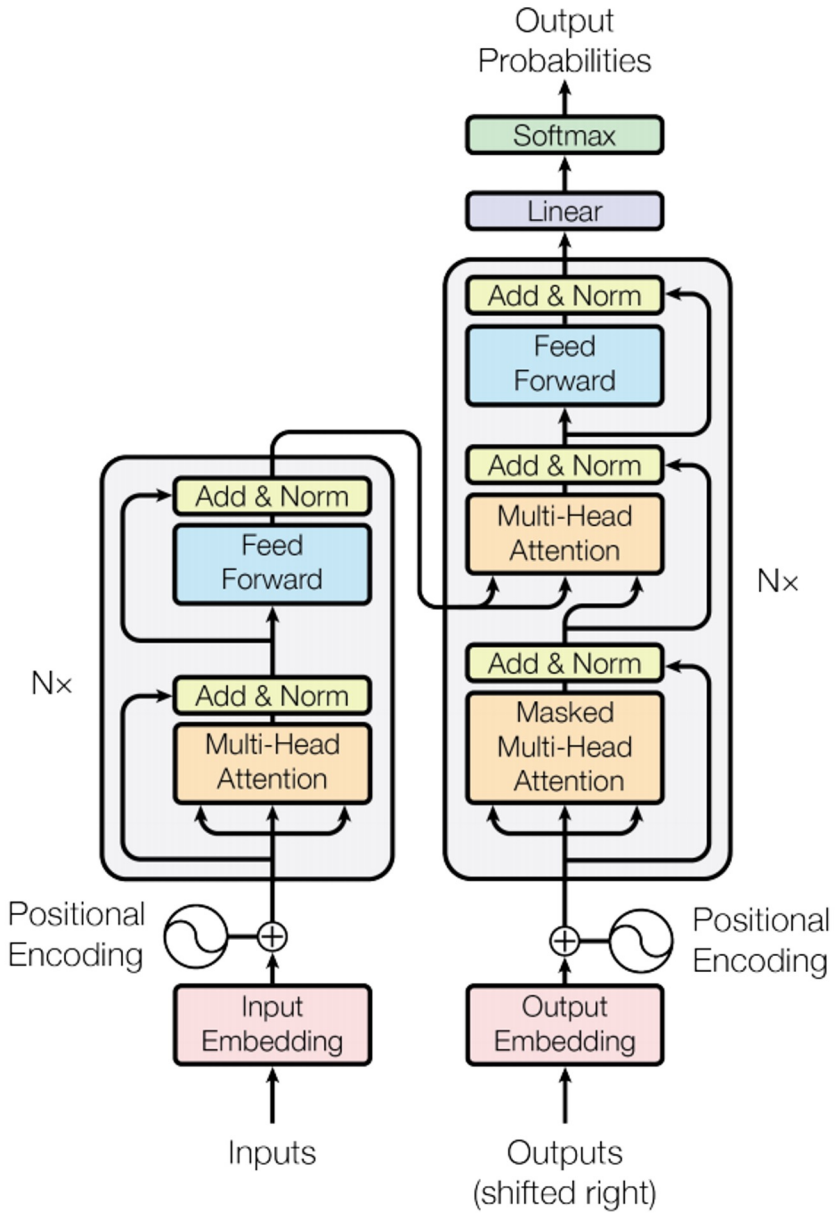
- RNN encoder + RNN decoder, interaction via fix-sized vector
- RNN encoder + RNN decoder, interaction via attention

NOW:

- **attention + attention + attention**



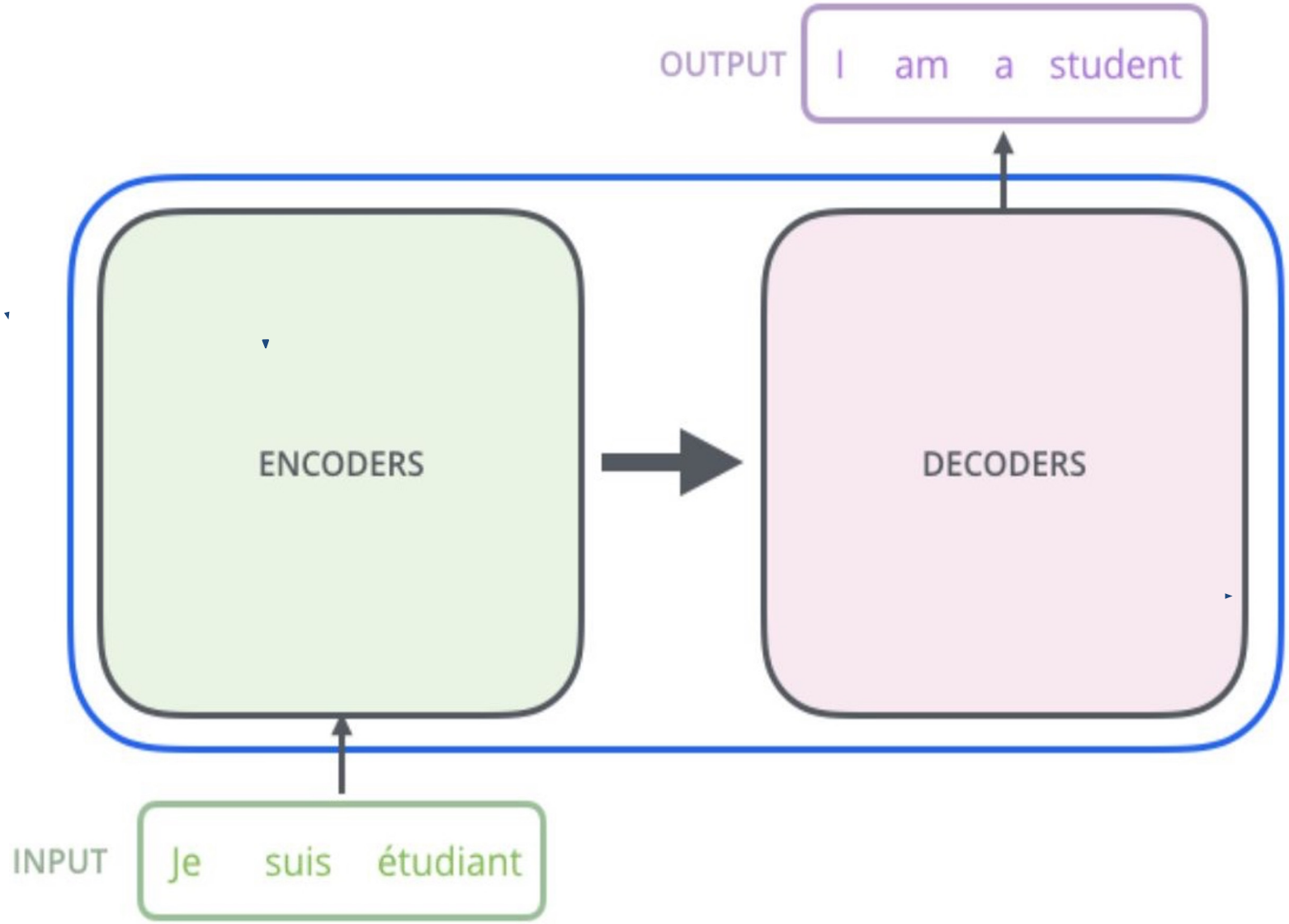
attention + attention + attention



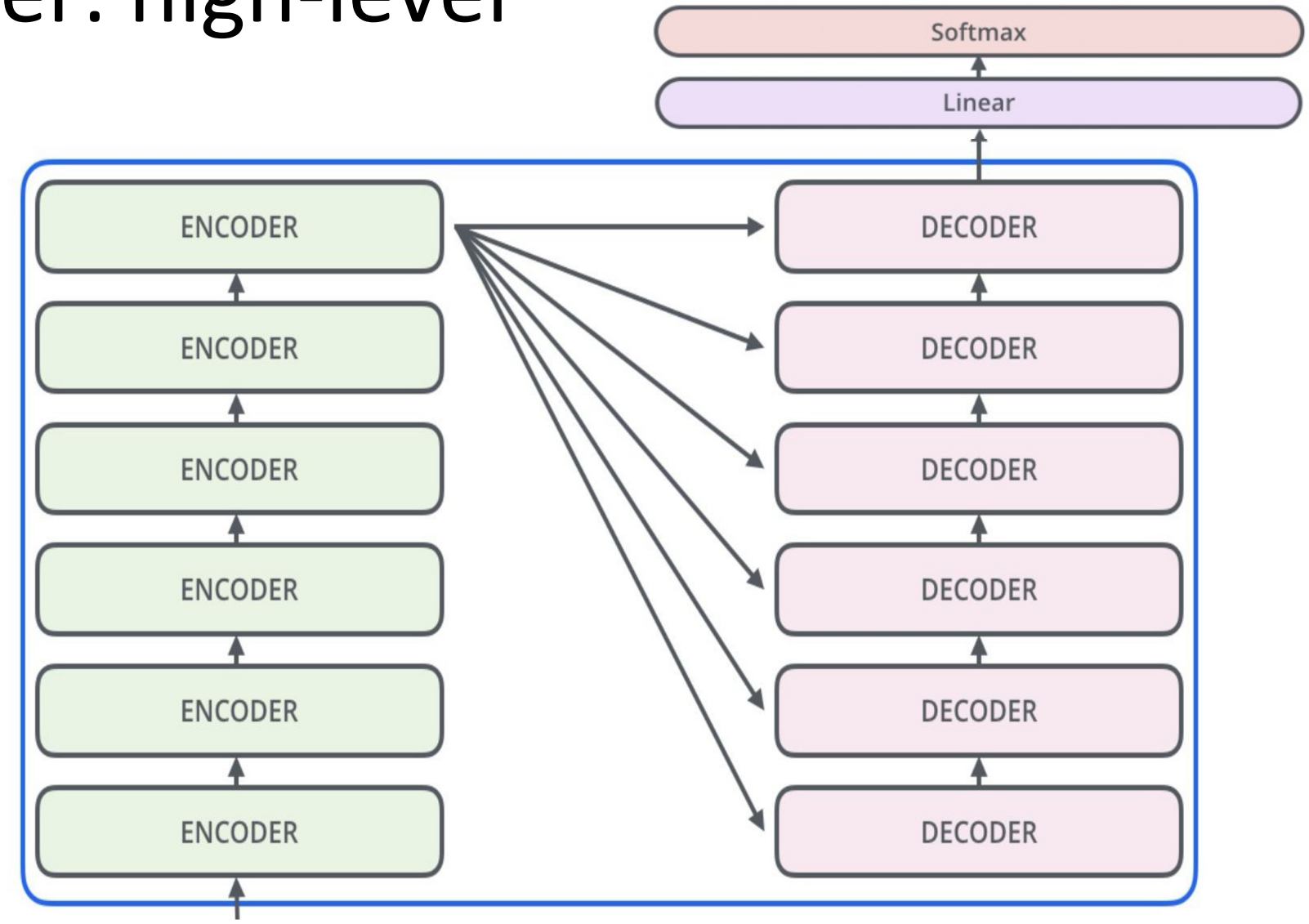
Lukasz Kaiser. 2017. Tensor2Tensor Transformers: New Deep Models for NLP. Lecture in Stanford University

Vaswani et al, 2017. Attention is all you need.

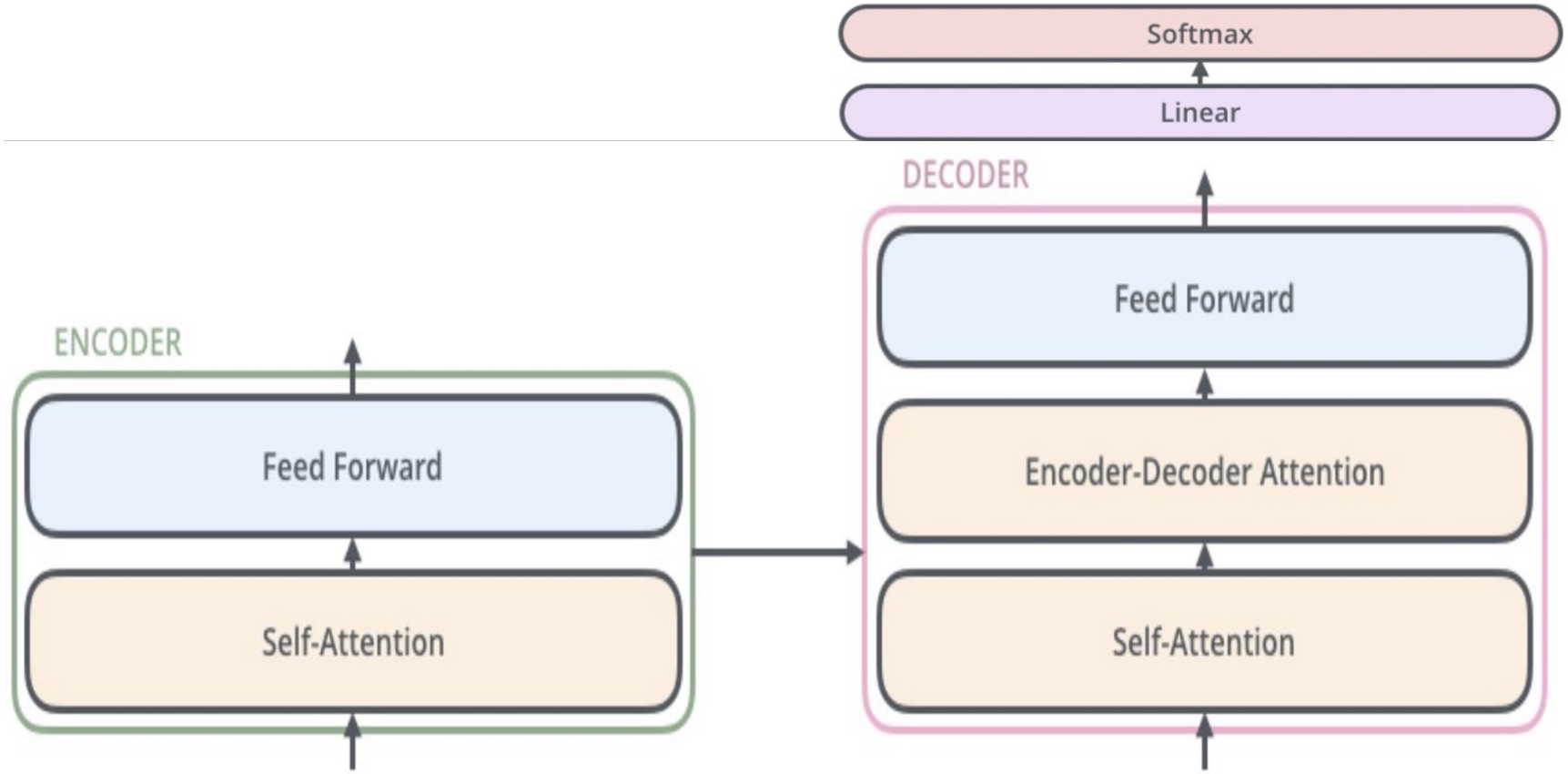
Transformer: high-level



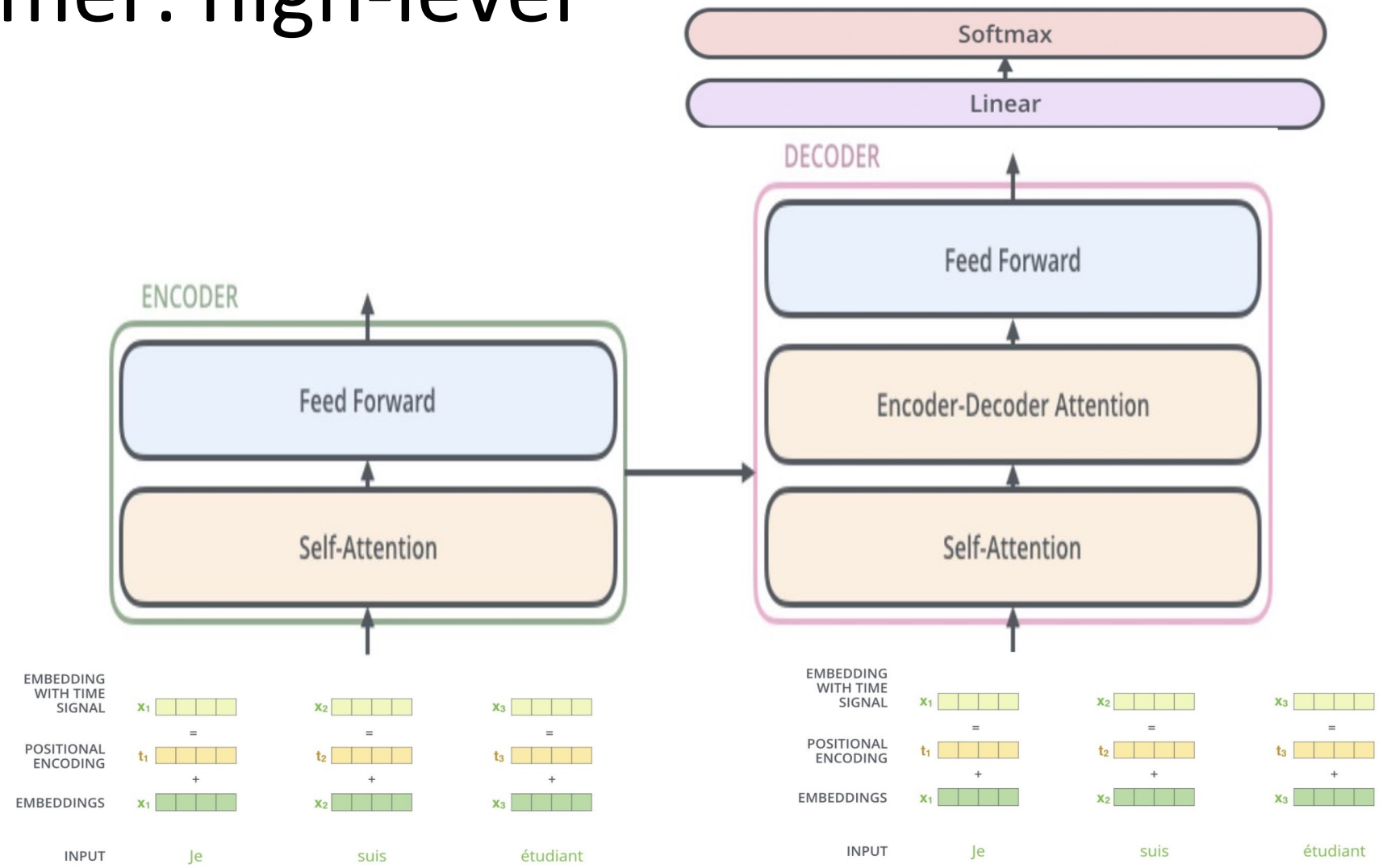
Transformer: high-level



Transformer: high-level

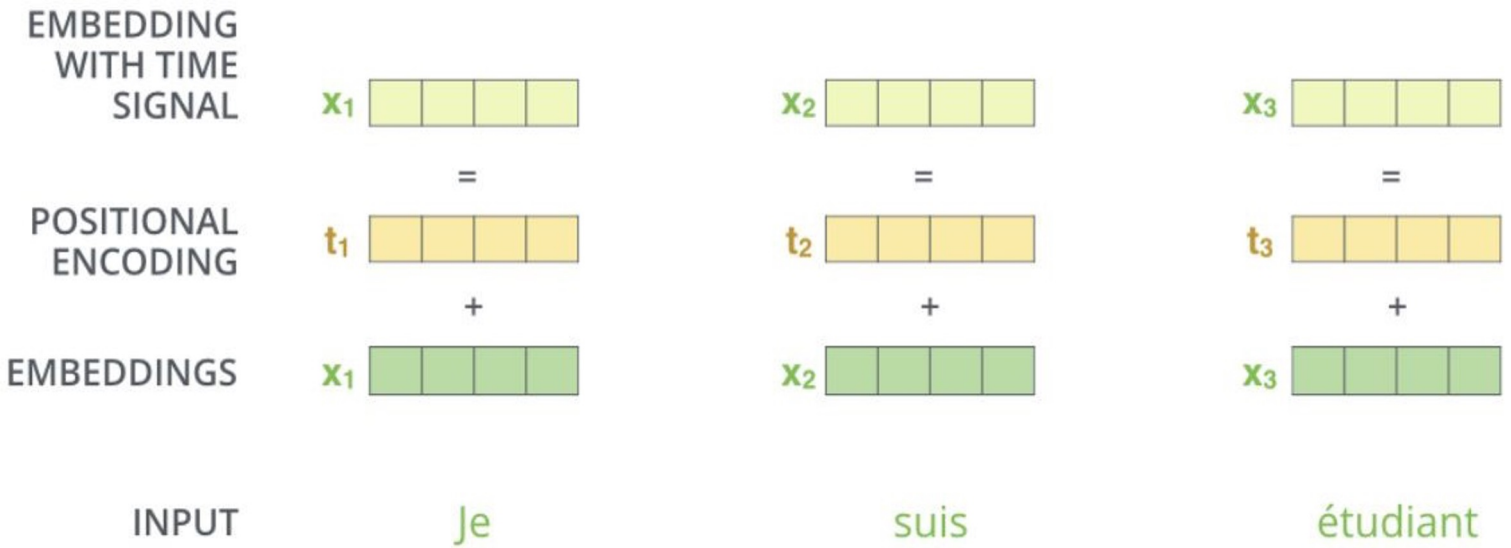


Transformer: high-level



Positional Encoding

positional encoding provides *order information* to the model

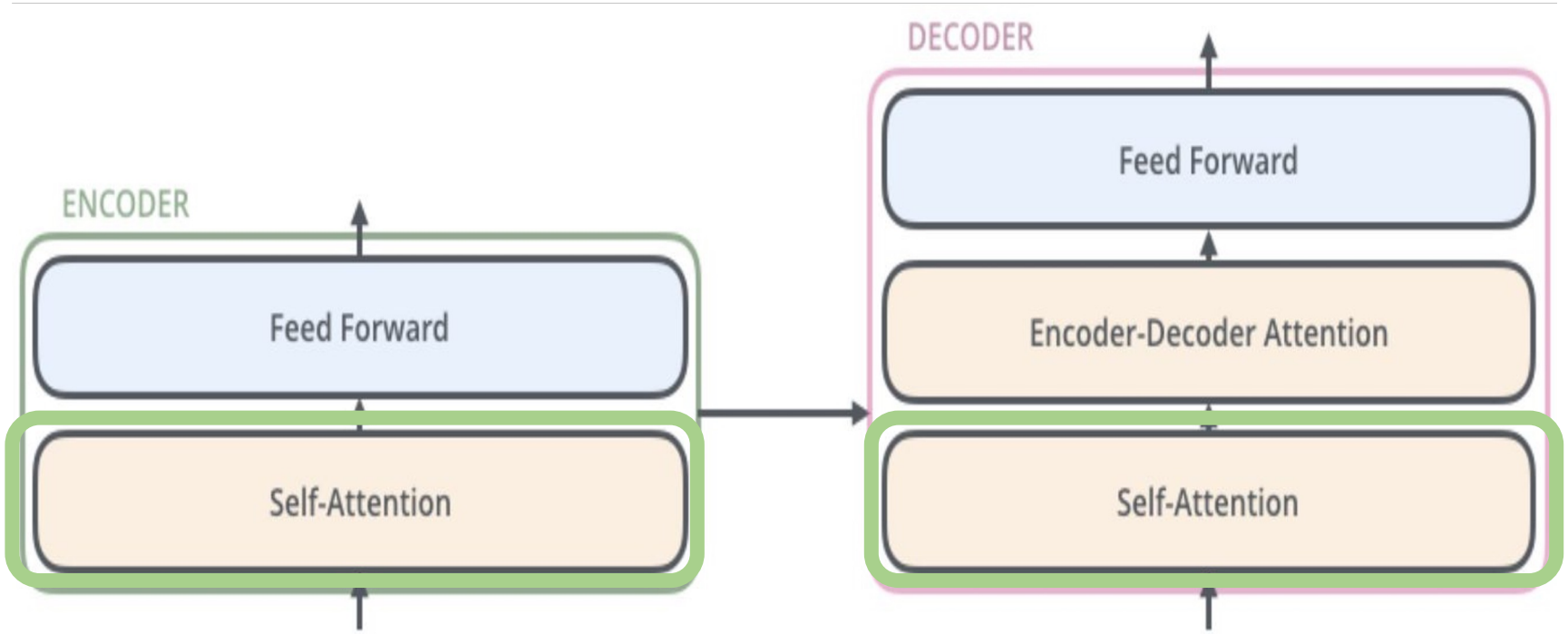


The fixed positional encodings used in the Transformer

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

https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Transformer: high-level

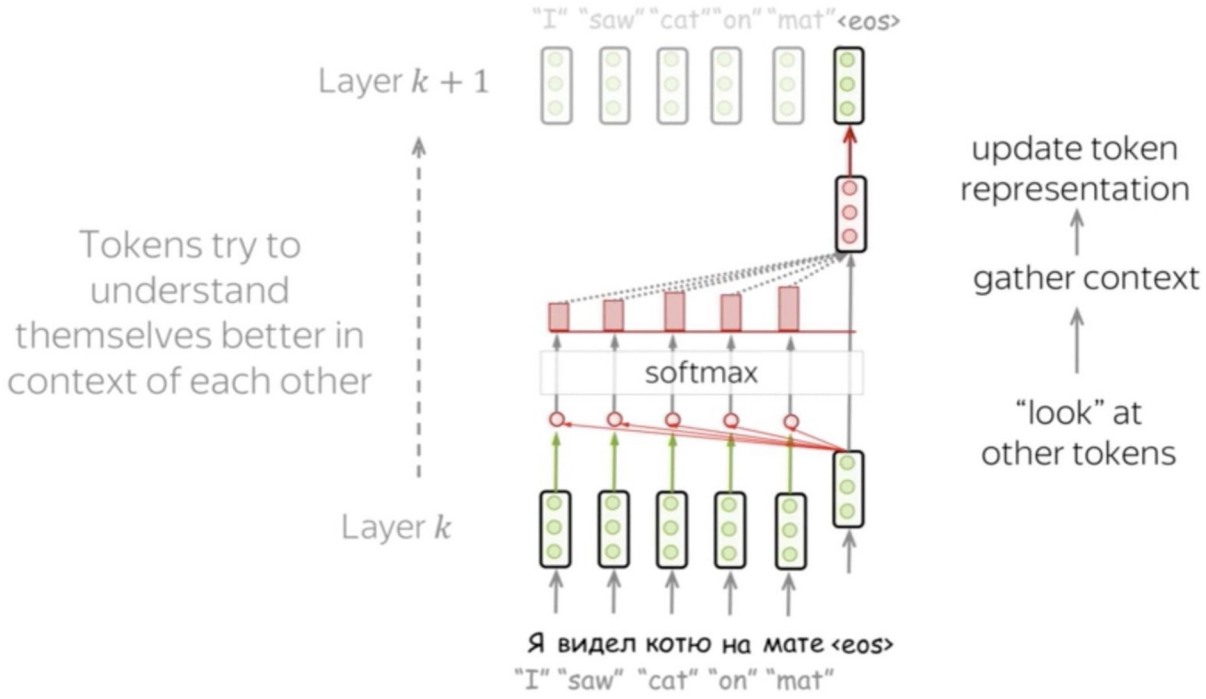


Transformers. Self-attention

Previously: one decoder state looked at all encoder states **NOW:** 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"



In Parallel!

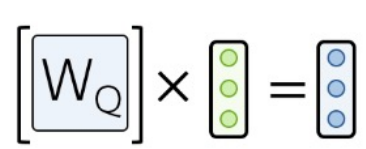
https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Query, Key and Value

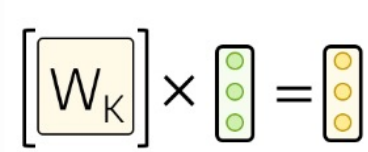
Query, Key and Value vectors

Each vector gets **three representations**:

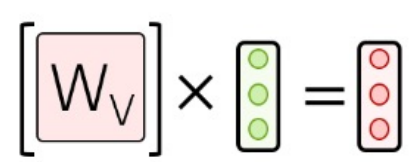
- **query** - asking for information;
- **key** - saying that it has some information;
- **value** - giving the information.



Query: vector **from** which the attention is looking



Key: vector **at** which the query looks to compute weights



Value: their weighted sum is attention output

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.

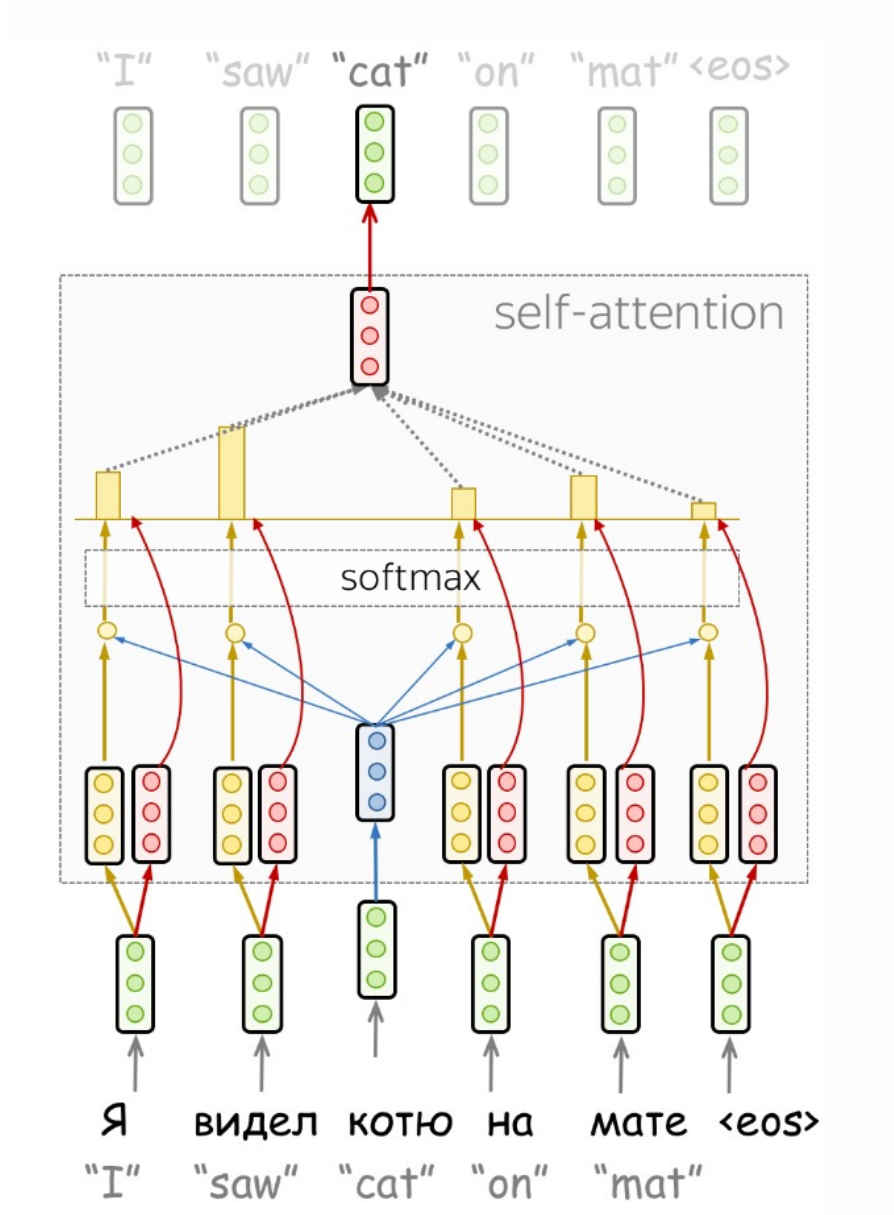
$$Attention(q, k, v) = \overbrace{softmax\left(\frac{qk^T}{\sqrt{d_k}}\right)v}^{\text{Attention weights}}$$

from (q) to (k)

vector dimensionality of K, V

https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Query, Key and Value



$$\left[W_Q\right] \times \square = \square$$

Query: vector **from** which the attention is looking

$$\left[W_K\right] \times \square = \square$$

Key: vector **at** which the query looks to compute weights

$$\left[W_V\right] \times \square = \square$$

Value: their weighted sum is attention output

$$Attention(q, k, v) = \overbrace{softmax\left(\frac{qk^T}{\sqrt{d_k}}\right)}^{\text{Attention weights}} v$$

from (q) to (k, v)

vector dimensionality of K, V

https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

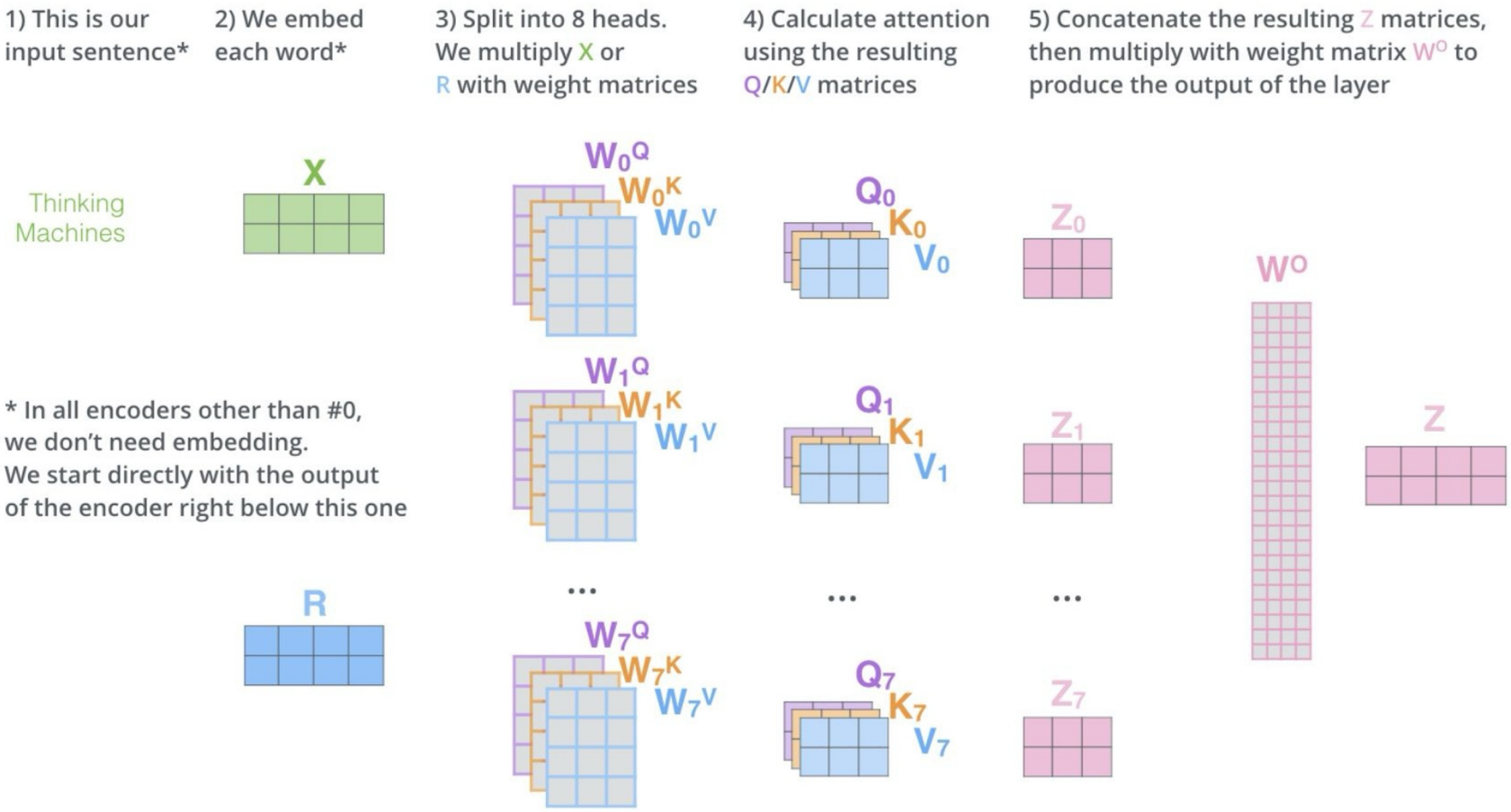
Masked self-attention

Decoder has different self-attention => **Masked self-attention**

- *we generate one token at a time => 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*

Multi-head attention

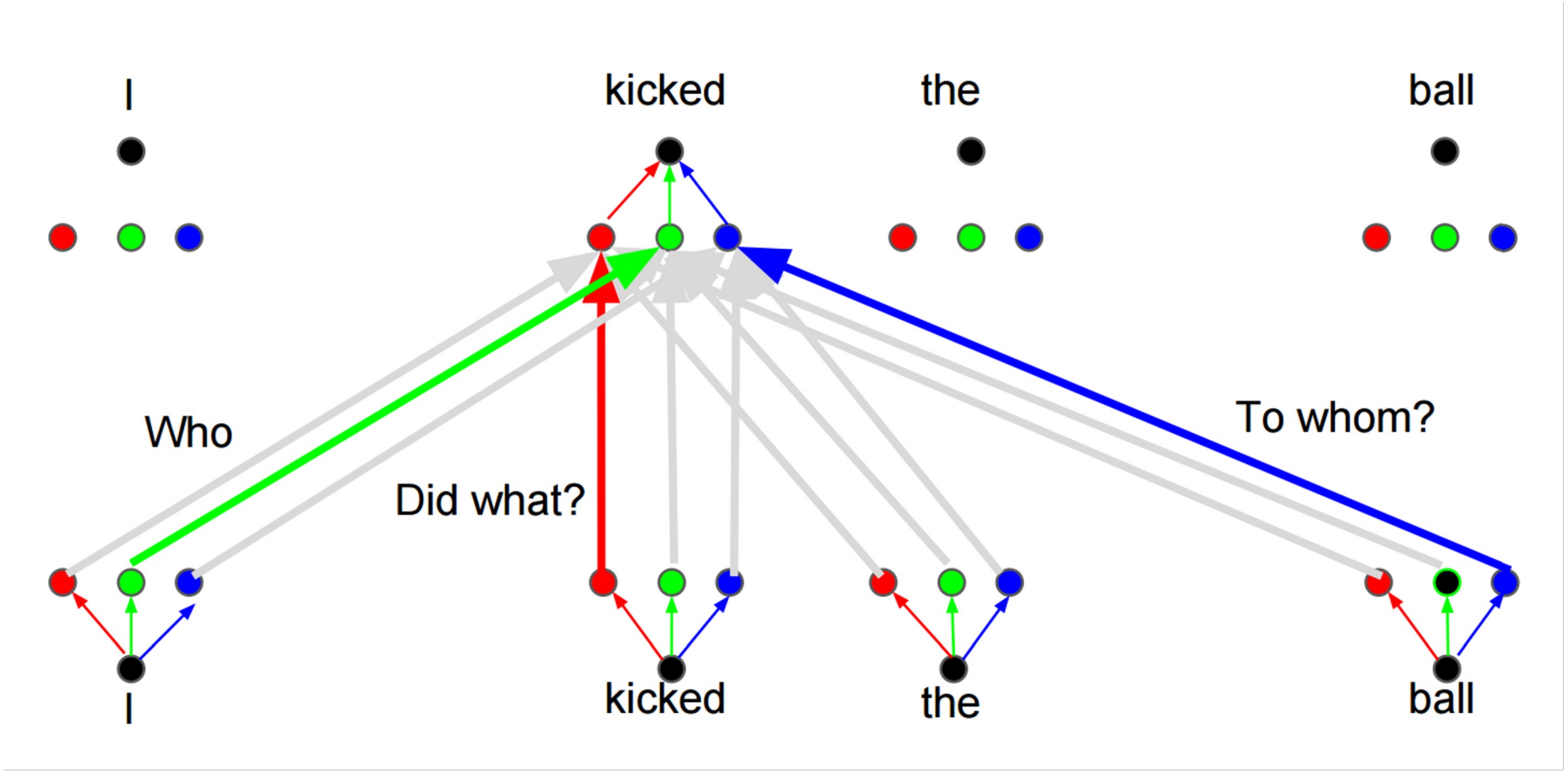
- We need to know relationships between in a sentence: relationships, preferences, order, issues like case or agreement
.
- Instead of having one attention **multi-head attention** several "heads" which **independently** and focus different things.



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

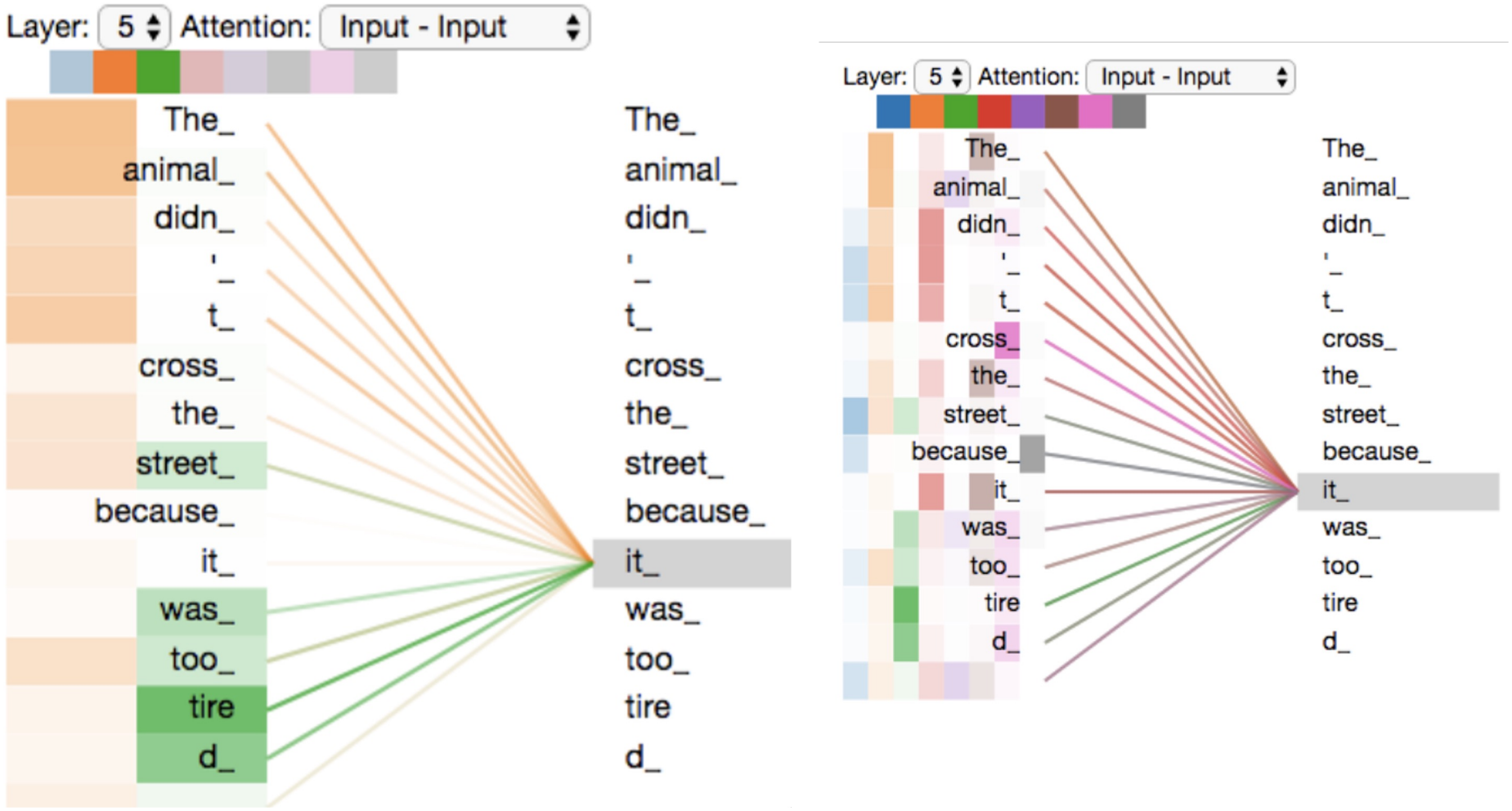
$$\text{where head}_i = \text{Attention}\left(\mathbf{Q}\mathbf{W}_i^Q,\mathbf{K}\mathbf{W}_i^K,\mathbf{V}\mathbf{W}_i^V\right)$$

Multihead attention



Ashish Vaswani and Anna Huang. Self-Attention For Generative Models

Multihead self-attention in encoder



Jay Alammar. The Illustrated Transformer. http://jalammar.github.io/illustrated-transformer/

Extra

Feed-forward blocks

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

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

Positionwise FFNN

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

- Linear→ReLU→Linear
  - Base: 512→2048→512
  - Large: 1024→4096→1024
- Equal to 2 conv layers with kernel size 1

```
class PositionwiseFeedForward(nn.Module):
    "Implements FFN equation."
    def __init__(self, d_model, d_ff, dropout=0.1):
        super(PositionwiseFeedForward, self).__init__()
        self.w_1 = nn.Linear(d_model, d_ff)
        self.w_2 = nn.Linear(d_ff, d_model)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        return self.w_2(self.dropout(F.relu(self.w_1(x))))
```


Rush. The Annotated Transformer. https://nlp.seas.harvard.edu/2018/04/03/attention.html

Extra

Feed-forward blocks

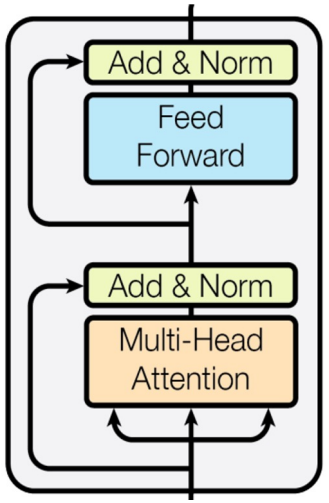
each layer has a feed-forward network block: two linear layers with ReLU non-linearity between them

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

Residual connection (train better)

Residual connections => add an input of the block to its output

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



Extra

Feed-forward blocks

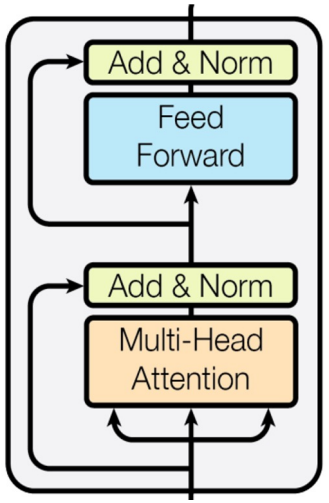
each layer has a feed-forward network block: two linear layers with ReLU non-linearity between them

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

Residual connection (train better)

Residual connections => add an input of the block to its output

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

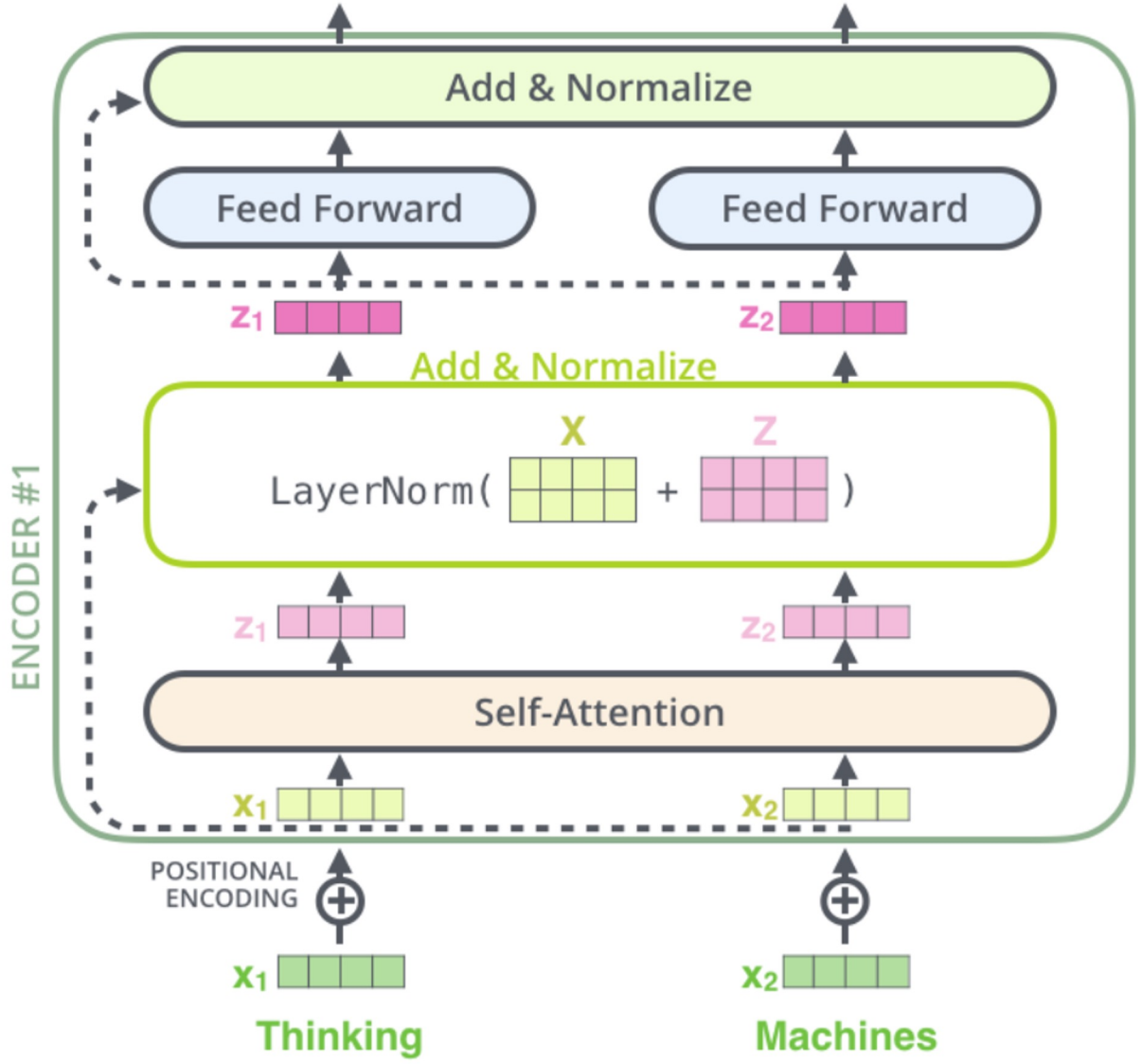


Layer Normalization (train faster)

Improves convergence

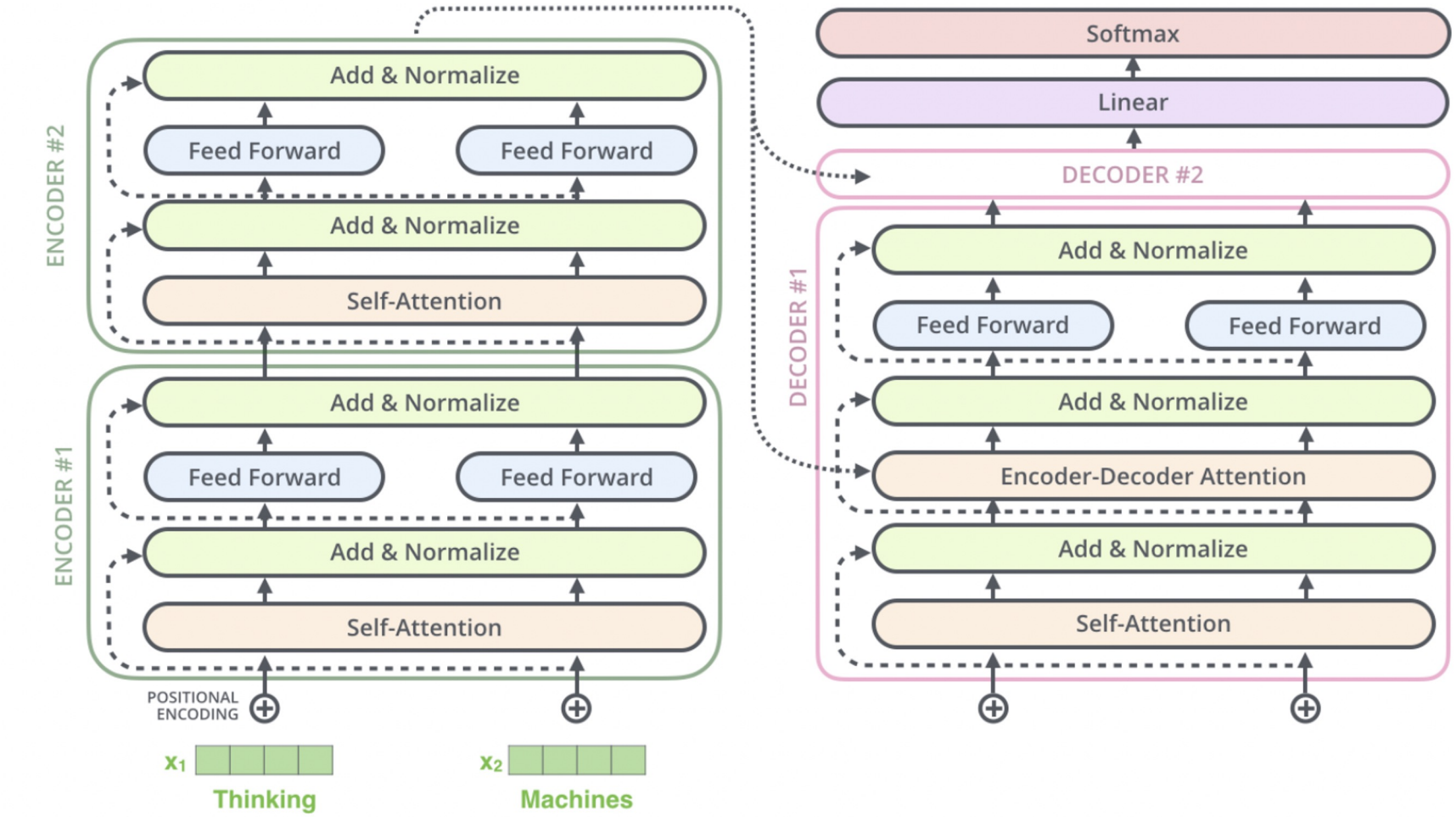
Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation within each layer

Transformer layer (enc) unrolled

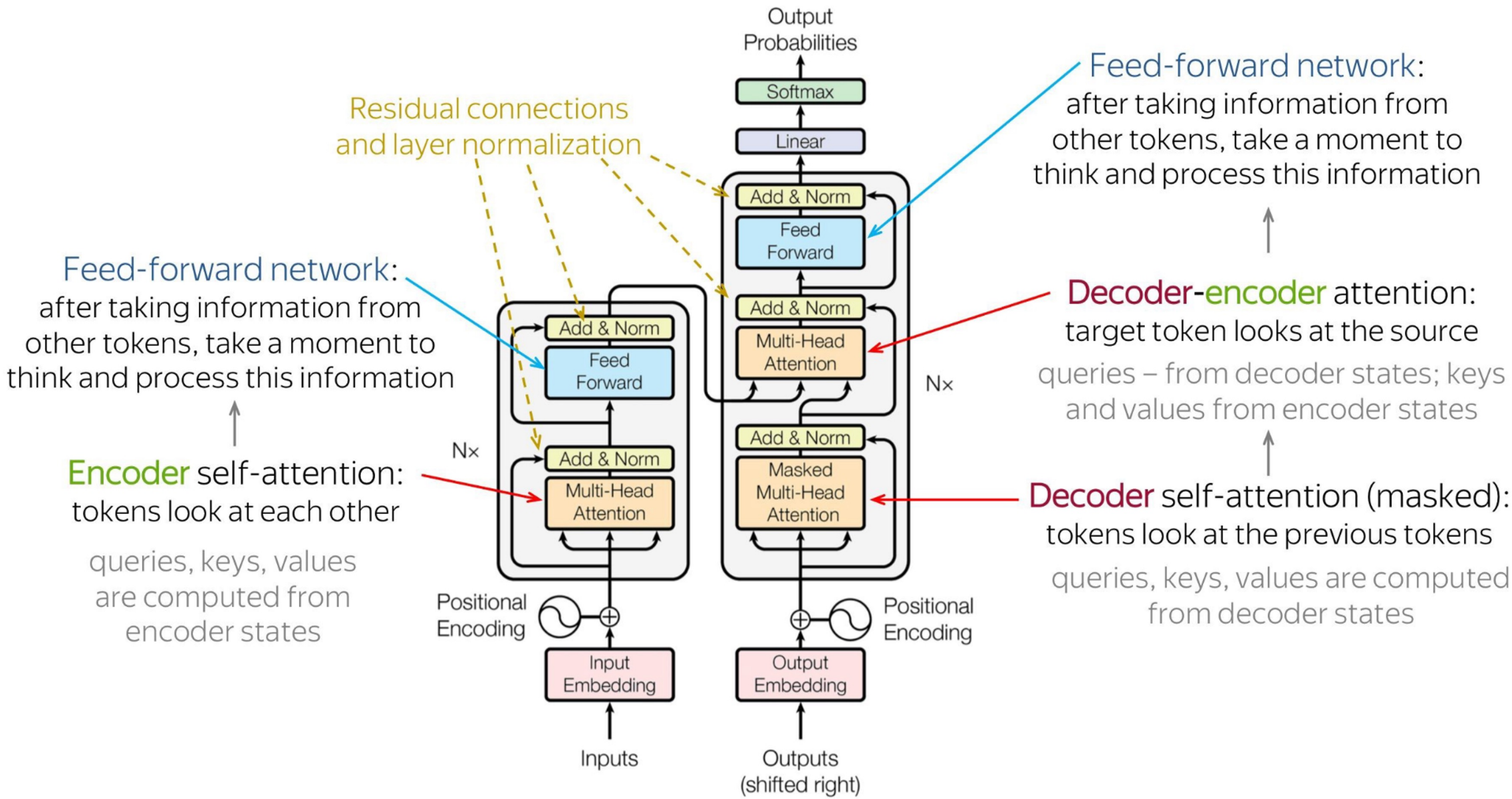


Jay Alammar. The Illustrated Transformer. http://jalammar.github.io/illustrated-transformer/

Transformer layer (dec) unrolled



Jay Alammar. The Illustrated Transformer. http://jalammar.github.io/illustrated-transformer/


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)
Residual connections and layer normalization
Feed-forward network: after taking information from other tokens, take a moment to think and process this information
Encoder self-attention: tokens look at each other
queries, keys, values are computed from encoder states
Feed-forward network: after taking information from other tokens, take a moment to think and process this information
Decoder-encoder attention: target token looks at the source
queries – from decoder states; keys and values from encoder states
Decoder self-attention (masked): tokens look at the previous tokens
queries, keys, values are computed from decoder states

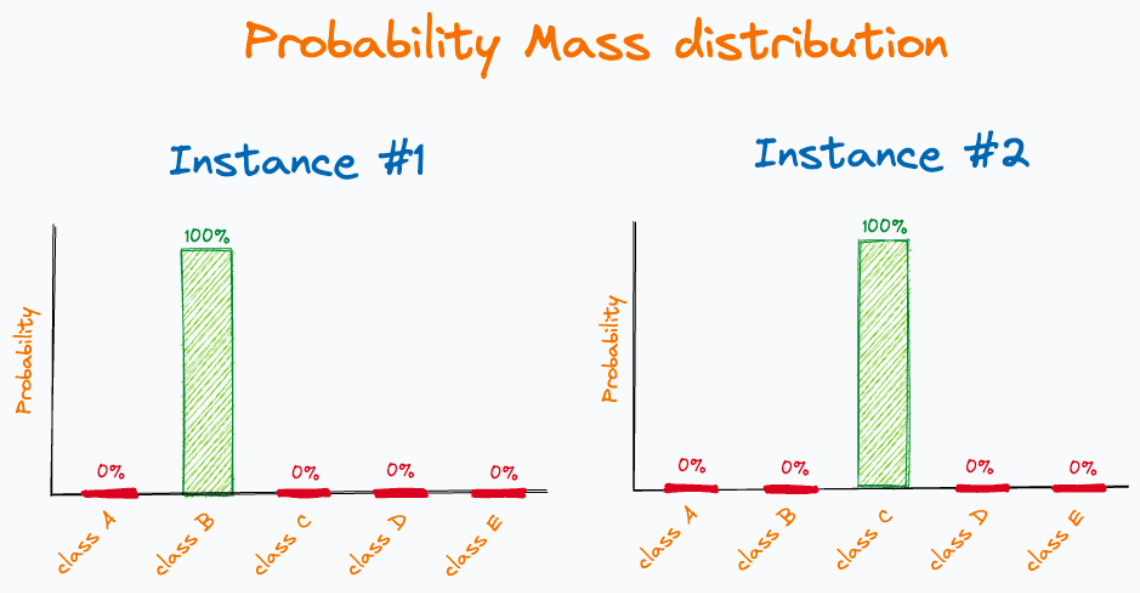
Extra: label smoothing

Problem:

For every instance in single-label classification datasets, the entire probability mass belongs to a single class, and the rest are zero.

Such label distributions **excessively motivate** the model to learn the true class for every sample with pretty high confidence.

This can impact its generalization capabilities.



https://blog.dailydoseofds.com/p/label-smoothing-the-overlooked-and

Extra: label smoothing

Solution:

- Reduce the probability mass of the true class slightly.
- The reduced probability mass is uniformly distributed to all other classes.

Result: the model to be “less overconfident” during training and prediction while still attempting to make accurate predictions.

https://blog.dailydoseofds.com/p/label-smoothing-the-overlooked-and

Extra: regularization

- **Dropout:**
  - *residual*
  - *ReLU*
  - *Input*
- **Attention dropout (only for some experiments)**
  - *Dropout on attention weights (after softmax)*
- **Label smoothing**

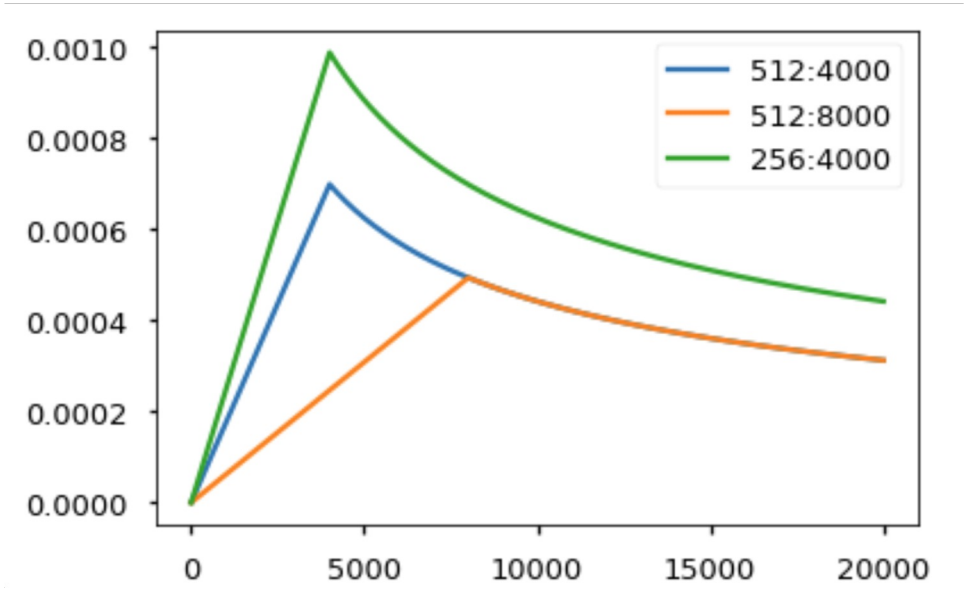
Label smoothing from:
Szegedy. Rethinking the Inception Architecture for Computer Vision, 2015

Training

- Adam, betas=0.9,0.98, eps=1e-9
- Learning rate: linear warmup: 4K-8K steps (3-10% is common) + square root decay

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

Noam Optimizer:
Adam+this lr schedule



Rush. The Annotated Transformer. https://nlp.seas.harvard.edu/2018/04/03/attention.html

Training

- WMT2014 En→De / Fr: 4.5M / 36M sent.pairs
  - *word-pieces vocab: 37K shared / 32K x2 separate*
  - *Batches: sequences of approx. same length, dynamic batch size: 25K src & 25K tgt tokens*
  - *On 8 P100 GPU (16GB), base/big: 0.5/3.5 days, 100k/300k steps 0.4/1.0s per step*
  - *Average weights from last 5/20 checkpoints*
  - *Beam search with size 4, length penalty 0.6*

Dev: newstest2013 en→de

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Vaswani et al, 2017. Attention is all you need.

Results

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Vaswani et al, 2017. Attention is all you need.

Recap on language modeling

Language Modelling

- **Language modeling** is the task of predicting what word comes next
  - *The student opened their ___ (books | laptops | exams | minds |…)*
  - *Что дальше будет неизвестно ___ (никому | заранее | …)*

Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n

Language Modelling

- **Language modeling** is the task of predicting what word comes next
  - *The student opened their ___ (books | laptops | exams | minds |…)*
  - *Что дальше будет неизвестно ___ (никому | заранее | …)*
- **Formally:** given a sequence of words $x_1, x_2, \ldots, x_t$, compute the probability distribution of the next word x_{t+1}
- A system that does it is called a **language model (LM)**
  - A probabilistic multi-class classifier: $P(x_{t+1} \mid x_1, x_2, \ldots, x_t)$

Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n

Language Modelling

- **Language modeling** is the task of predicting what word comes next
  - *The student opened their ___ (books | laptops | exams | minds |…)*
  - *Что дальше будет неизвестно ___ (никому | заранее | …)*
- **Formally:** given a sequence of words $x_1, x_2, \ldots, x_t$, compute the probability distribution of the next word x_{t+1}
- A system that does it is called a **language model (LM)**
  - A probabilistic multi-class classifier: $P(x_{t+1} \mid x_1, x_2, \ldots, x_t)$
- **Usages:**
  - Reranking hypotheses of a translation model (e.g. in IBM models)
  - Generating a translation (if the model is conditional on the input text)

Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n

n-gram Language Modeling

- First we make a simplifying assumption: $\boldsymbol{x}^{(t+1)}$ depends only on the preceding *n-1* words.

$$P(\boldsymbol{x}^{(t+1)}|\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(1)}) = P(\boldsymbol{x}^{(t+1)}|\overbrace{\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)}}^{n\text{-1 words}}) \quad \text{(assumption)}$$

prob of a n-gram

prob of a (n-1)-gram

$$= \frac{P(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{P(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(definition of conditional prob)}$$

- **Question:** How do we get these *n*-gram and (*n*-1)-gram probabilities?
- **Answer:** By counting them in some large corpus of text!

$$\approx \frac{\text{count}(\boldsymbol{x}^{(t+1)},\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})}{\text{count}(\boldsymbol{x}^{(t)},\ldots,\boldsymbol{x}^{(t-n+2)})} \quad \text{(statistical approximation)}$$

Source: Abigail See. CS224N/Ling284 slides: http://web.stanford.edu/class/cs224n/

Disadvantages of n-gram LMs

- needs smoothing
- can't handle long history
- they cannot generalize over contexts of similar words well

Source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

Neural Language Models: Motivation

(+) a neural language model has much **higher predictive accuracy** than an *n*-gram language model!

(–) neural net language models are **strikingly slower to train** than traditional language models

Source: https://web.stanford.edu/~jurafsky/slp3/7.pdf

Basic types of transformer architectures

BERT-like models: encoder only

			мы	##ла					
…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
[CLS]	М	##ама	[MASK]	[MASK]	р	##ам	##у	.	[SEP]

In encoders, each token attends **to each other token**
They are good for understanding texts.

GPT-like models: decoder only

М	##ама	мы	##ла	р	##ам	##у	.	<e>
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
<s>	М	##ама	мы	##ла	р	##ам	##у	.

In decoders, each token attends **only to the tokens on the left** (and to all tokens of the encoder, if it exists)
They are good for generating texts.

The classical transformer (also T5, BART, etc.): encoder and decoder

…	…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…	…
<s>	М	##ама	мы	##ла	р	##ам	##у	.	<e>

Mo	##m	was	was	##hing	the	frame	Mo	<e>
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…
<s>	Mo	##m	was	was	##hing	the	frame	.

Do you really want a transformer?

- Probably yes, if:
  - There is a pretrained transformer for your language and task (or a similar one)
  - You have a small training set and want to generalize from it
  - Your task is natural language generation and you have no templates
  - You have a lot of GPUs and you may run an LLM
- Probably no, if:
  - Your problem can be solved with rules or keywords
  - You need fast (around 1ms) inference without GPU
  - You need fast training without GPU
  - You have to process very long texts without splitting them
  - Your task needs some specific architecture (e.g. memory or recursion)