Indian Institute of Technology Kharagpur
Department of Computer Science & Engineering

CS60075
Natural Language Processing
Autumn 2020

Module 7:
Machine Translation 4
Neural Machine Translation
21 October 2020

Conditional Language Modeling

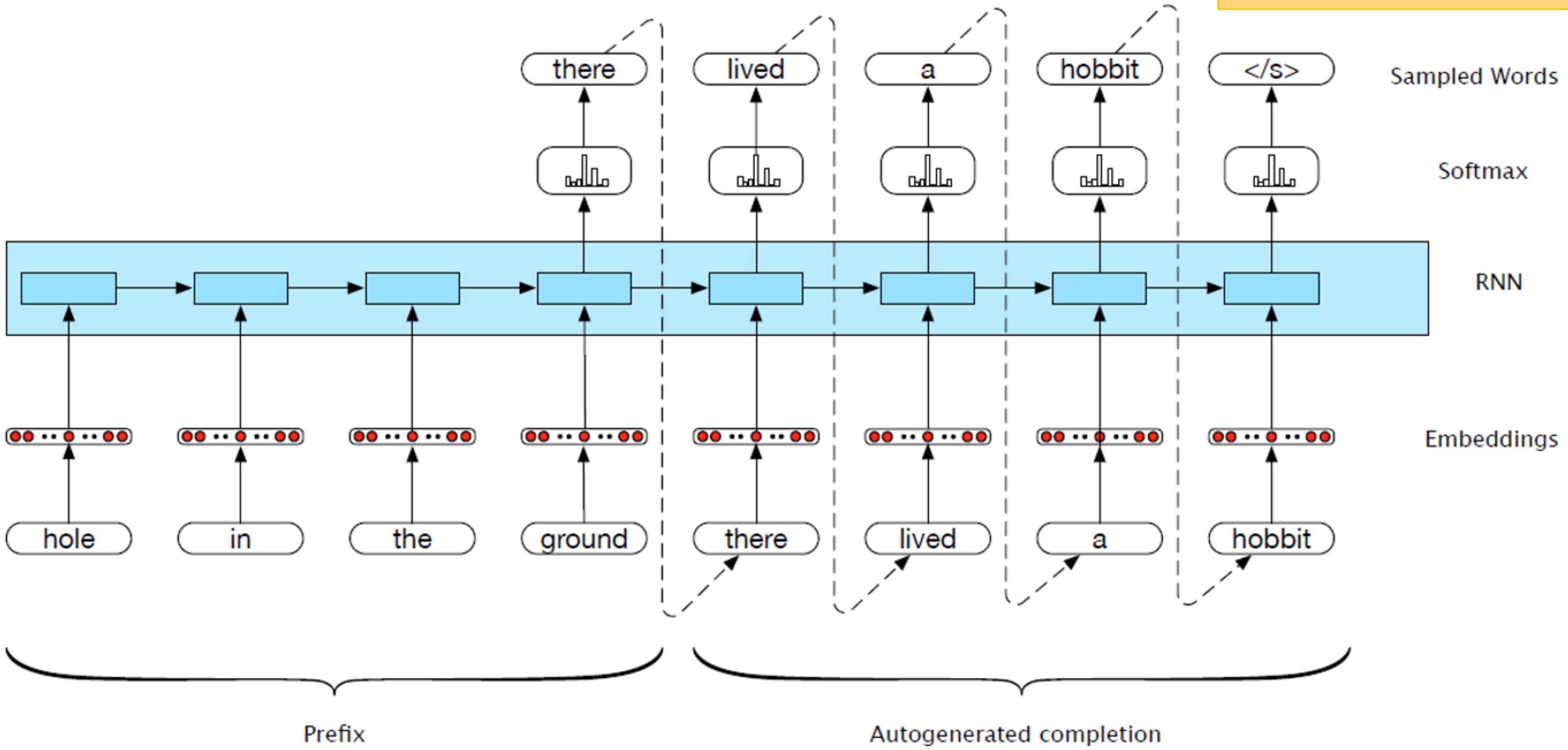
$$P(Y|X) = \prod_{j=1}^{J} P\left(y_j | \mathbf{X}, y_1, \ldots, y_{j-1}\right)$$

Generate text according to some specification

Input X	Output Y	Task
Structured data	NL Description	NL Generation
Hindi	French	Translation
Document	Short Description	Summarization
Utterance	Response	Response generation
Image	Text	Image Captioning
Speech	Transcript	Speech Recognition

Generation with prefix

$$h_t = g(h_{t-1}, x_t)$$
$$y_t = f(h_t)$$

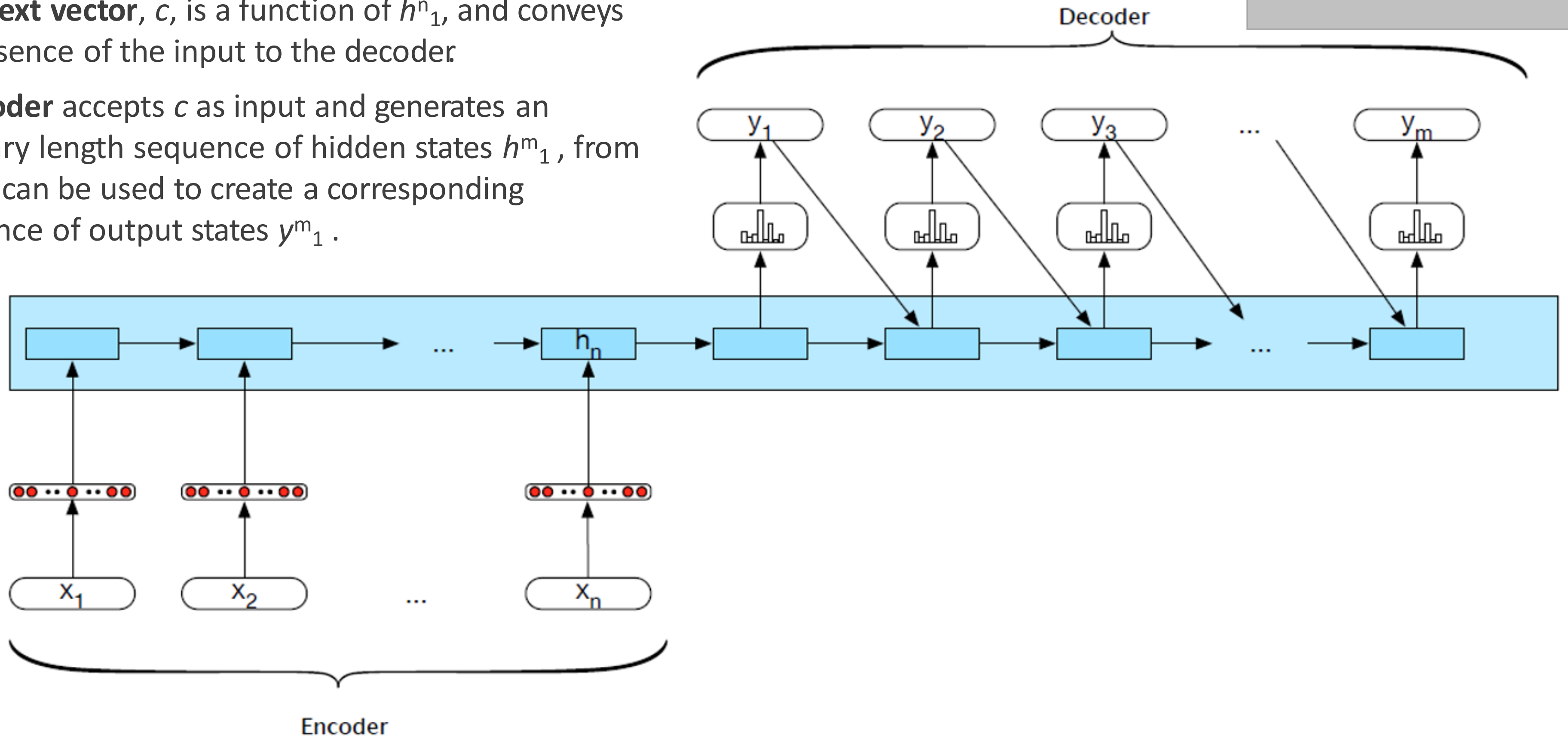


Encoder-decoder networks

$$c = h_n^e$$
$$h_0^d = c$$

$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d)$$
$$Z_t = f(h_t^d)$$
$$y_t = \text{soft max } (z_t)$$

1. An **encoder** takes an input sequence $x^n_{1,}$ and generates a corresponding sequence of contextualized representations, h^n_1.
2. A **context vector**, *c*, is a function of h^n_1, and conveys the essence of the input to the decoder.
3. A **decoder** accepts *c* as input and generates an arbitrary length sequence of hidden states h^m_1, from which can be used to create a corresponding sequence of output states y^m_1.



Encoder-decoder networks

$$c = h_n^e$$
$$h_0^d = c$$

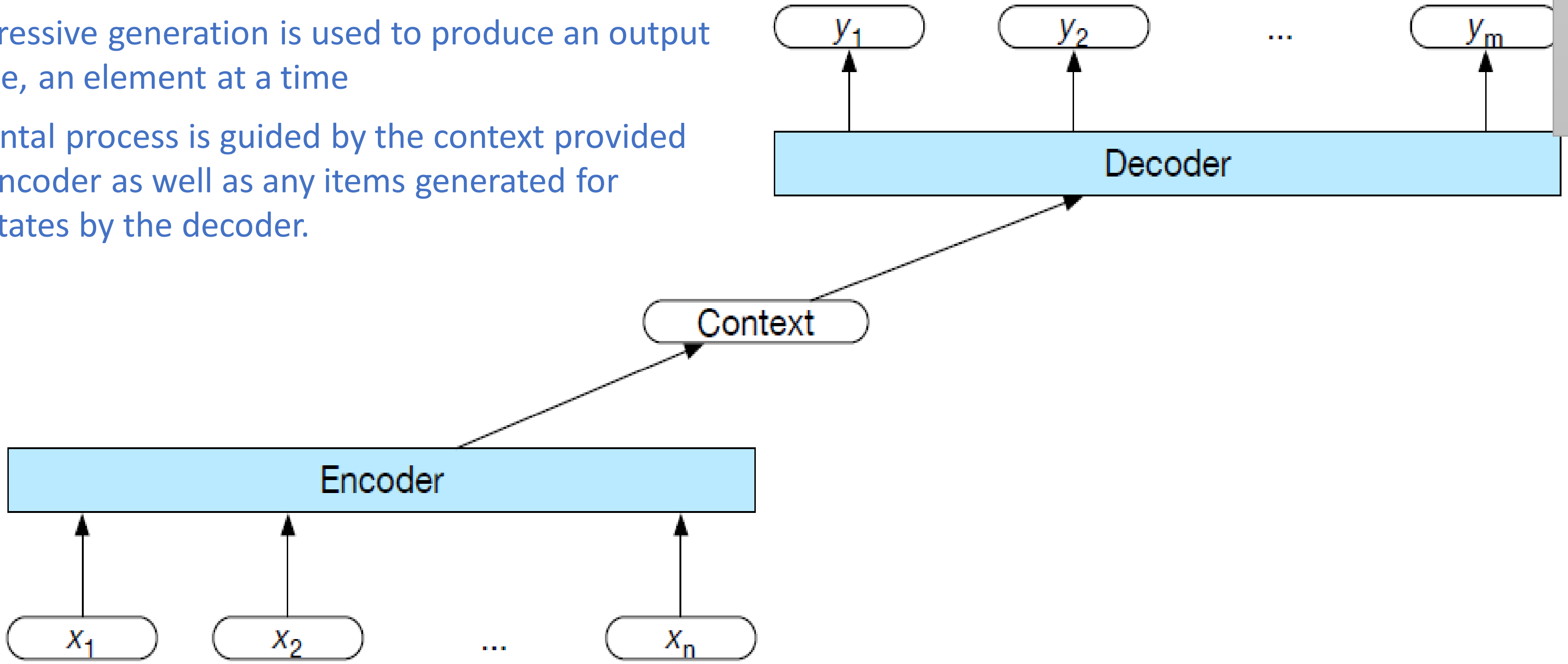
$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d)$$
$$Z_t = f(h_t^d)$$
$$y_t = \text{soft max } (z_t)$$

Encoder

- simple RNNs, LSTMs, GRUs,
- stacked Bi-LSTMs widely used

Decoder

- Autoregressive generation is used to produce an output sequence, an element at a time
- Incremental process is guided by the context provided by the encoder as well as any items generated for earlier states by the decoder.



Decoder Weaknesses

- The context vector *c* was only directly available at the beginning of the generation process.
- This meant that its influence became less-and-less important as the output sequence was generated.
- Make *c* available at each step in the decoding process,

1. when generating the hidden states in the deocoder, and
2. while producing the generated output.

$$c = h_n^e$$
$$h_0^d = c$$

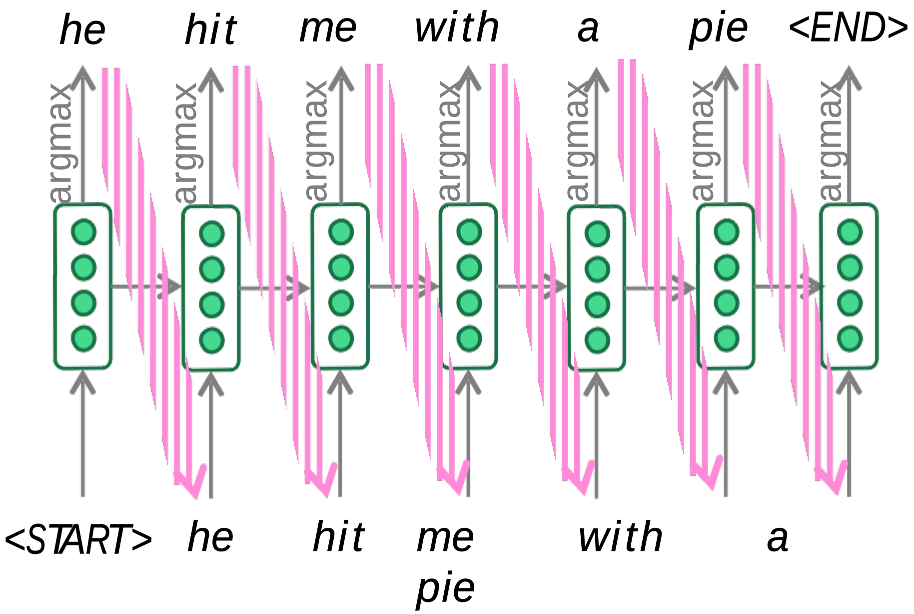
$$h_t^d = g\left(\hat{y}_{t-1}, h_{t-1}^d, c\right)$$
$$Z_t = f\left(h_t^d\right)$$
$$y_t = \operatorname{softmax}\left(\hat{y}_{t-1}, z_t, c\right)$$

Choosing the best output

- For neural generation, where we are trying to generate novel outputs, we can simply sample from the softmax distribution.
- In MT where we're looking for a specific output sequence, sampling isn't appropriate and would likely lead to some strange output.
- Instead we choose the most likely output at each time step by taking the argmax over the softmax output

$$\hat{y} = \text{argmax} P(y_i | y_< i)$$

Greedy decoding



This is greedy decoding (take most probable word on each step)

Exhaustive search decoding

Ideally we want to find a (length T) translation y that maximizes

$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots, P(y_T|y_1, \ldots, y_{T-1}, x)$$
$$= \prod_{t=1}^{T} P(y_t|y_1, \ldots, y_{t-1}, x)$$

- We could try computing all possible sequences *y*
  - On each step *t* of the decoder, we will be tracking V^t possible partial translations,
  - O(V^t) complexity

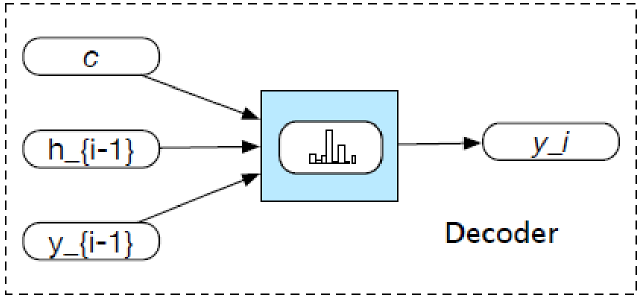
Beam search decoding

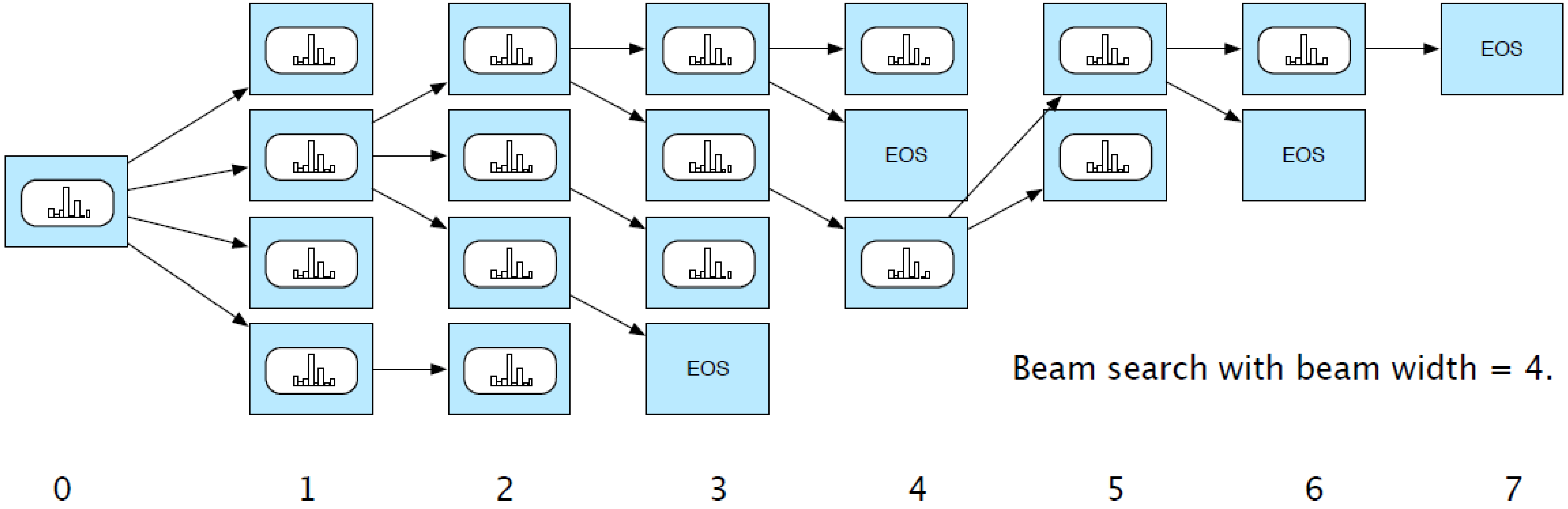
- Core idea: On each step of decoder, keep track of the *k* most probable partial translations (which we call *hypotheses*)
  - *k* is the beam size (in practice around 5 to 10)
- A hypothesis $y_1, \ldots, y_t$ has a score which is its log probability:

$$\text{score}(y_1, \ldots, y_t) = \log P_{\text{LM}}(y_1, \ldots, y_t | x) = \sum_{i=1}^{t} \log P_{\text{LM}}(y_i | y_1, \ldots, y_{i-1}, x)$$

  - Scores are negative, and higher score is better
  - We search for high-scoring hypotheses, tracking top *k* on each step
- Beam search is not guaranteed to find optimal solution
- But much more efficient than exhaustive search!

Beam search





Beam search with beam width = 4.

Attention

- Weaknesses of the context vector:

- Only directly available at the beginning of the process and its influence will wane as the output sequence is generated

- Context vector is a function (e.g. last, average, max, concatenation) of the hidden states of the encoder. This approach loses useful information about each of the individual encoder states

- Potential solution: **attention mechanism**

Attention

- Replace the static context vector with one that is dynamically derived from the encoder hidden states at each point during decoding
- This context vector is available to decoder hidden state calculations

$$h_i^d = g\left(\hat{y}_{i-1}, h_{i-1}^d, c_i\right)$$

- To calculate c_i, find relevance of each encoder hidden state j to the decoder state. Call it $score\left(h_{i-1}^d, h_j^e\right)$
- The $score$ can simply be dot product,

$$score\left(h_{i-1}^d, h_j^e\right) = h_{i-1}^d \cdot h_j^e$$

Attention

- The score can also be parameterized with weights

$$score\left(h_{i-1}^{d}, h_{j}^{e}\right) = h_{i-1}^{d} W_s \, h_{j}^{e}$$

- Normalize them with a softmax to create a vector of weights $\alpha_{i,j}$ that tells us the proportional relevance of each encoder hidden state j to the current decoder state i

$$\alpha_{i,j} = softmax\left(score\left(h_{i-1}^{d}, h_{j}^{e}\right) \forall j \in e\right)$$

- Finally, context vector is the weighted average of encoder hidden states

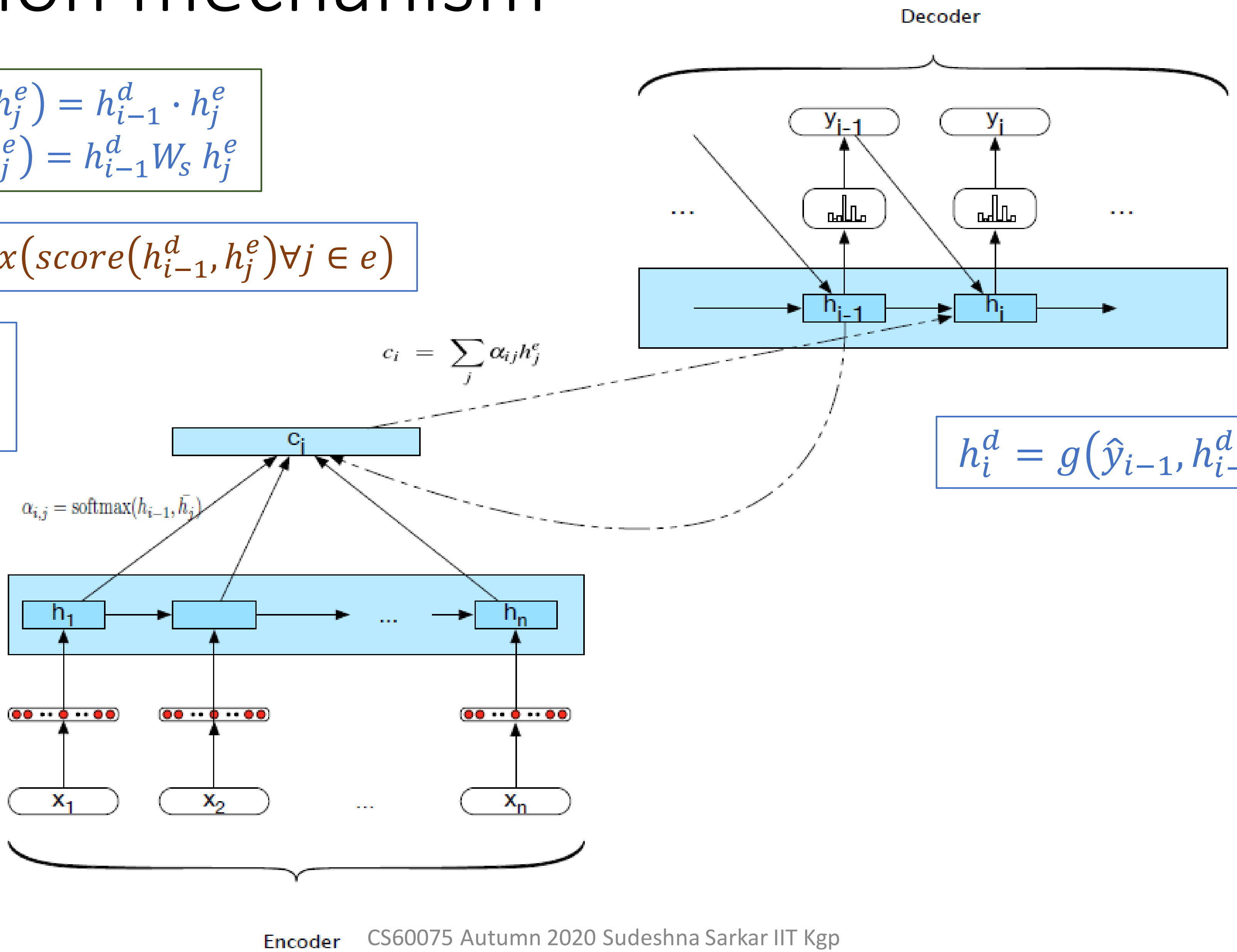
$$c_i = \sum_{j} \alpha_{i,j} h_{j}^{e}$$

Attention mechanism

$$score1\left(h_{i-1}^{d}, h_{j}^{e}\right) = h_{i-1}^{d} \cdot h_{j}^{e}$$
$$score2\left(h_{i-1}^{d}, h_{j}^{e}\right) = h_{i-1}^{d} W_s \, h_{j}^{e}$$

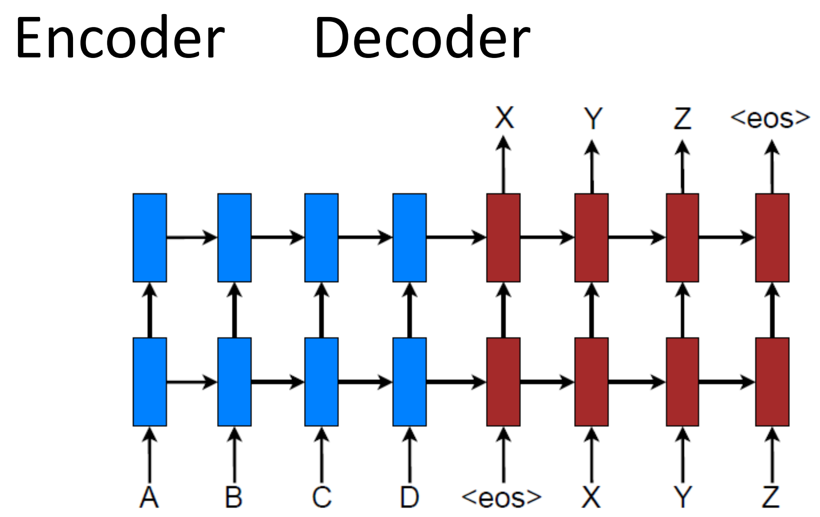
$$\alpha_{i,j} = softmax\left(score\left(h_{i-1}^{d}, h_{j}^{e}\right) \forall j \in e\right)$$

$$c_i = \sum_j \alpha_{i,j} h_j^e$$

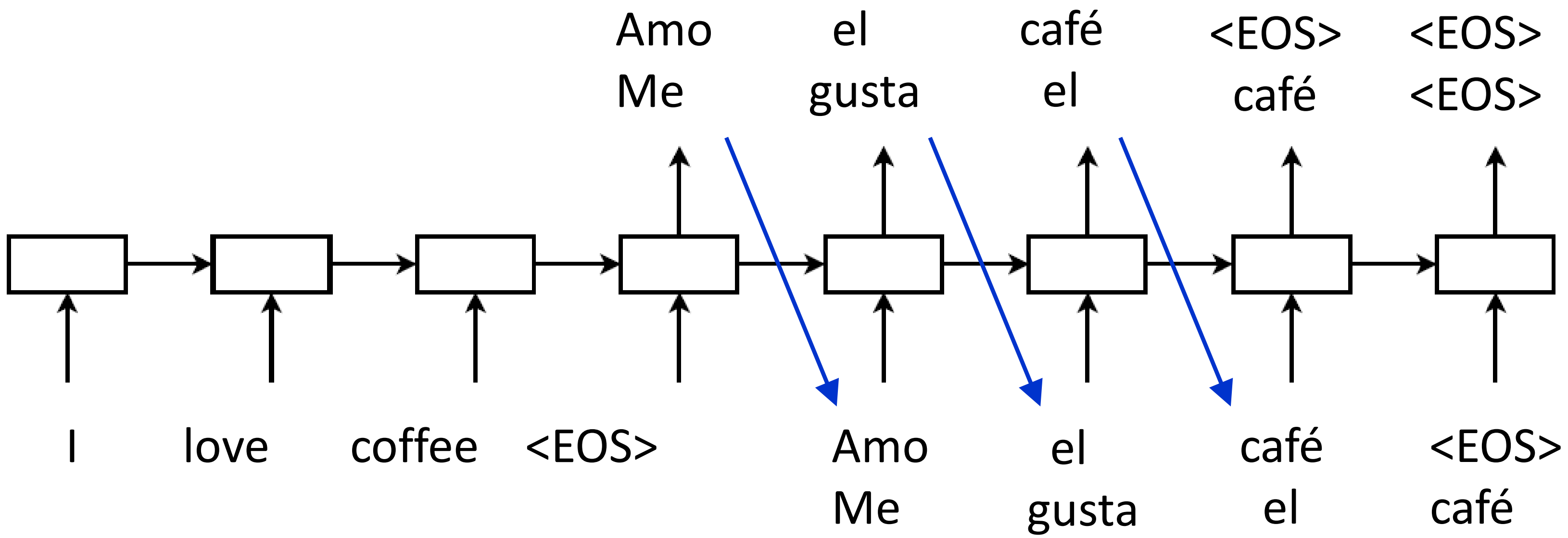


$$h_i^d = g\left(\hat{y}_{i-1}, h_{i-1}^d, c_i\right)$$

Sequence-To-Sequence Model Translation



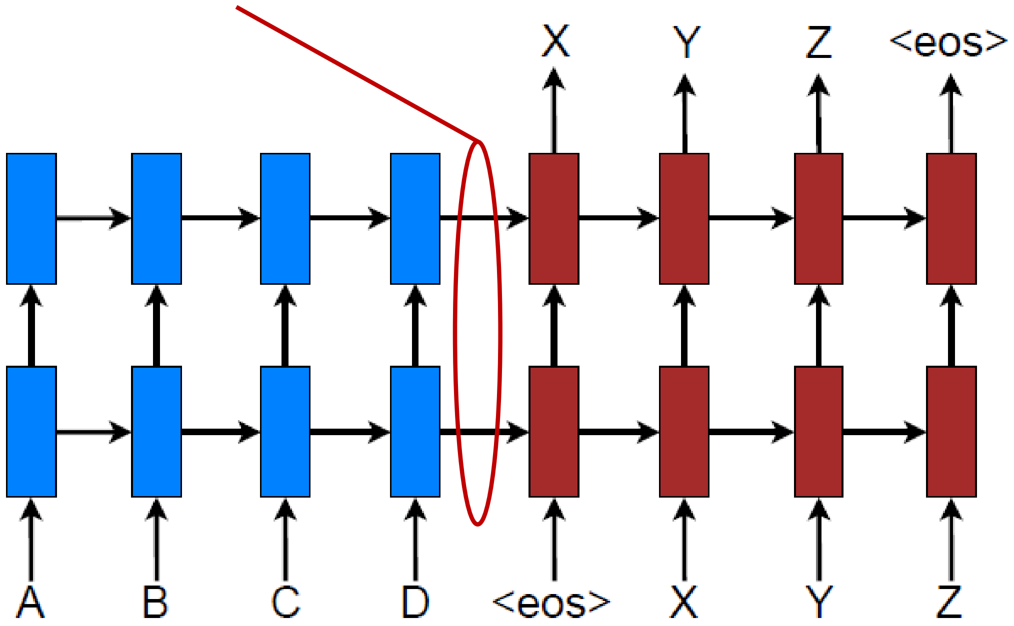
A depth-2 model



Beam search width = 2

Sequence-To-Sequence Criticisms

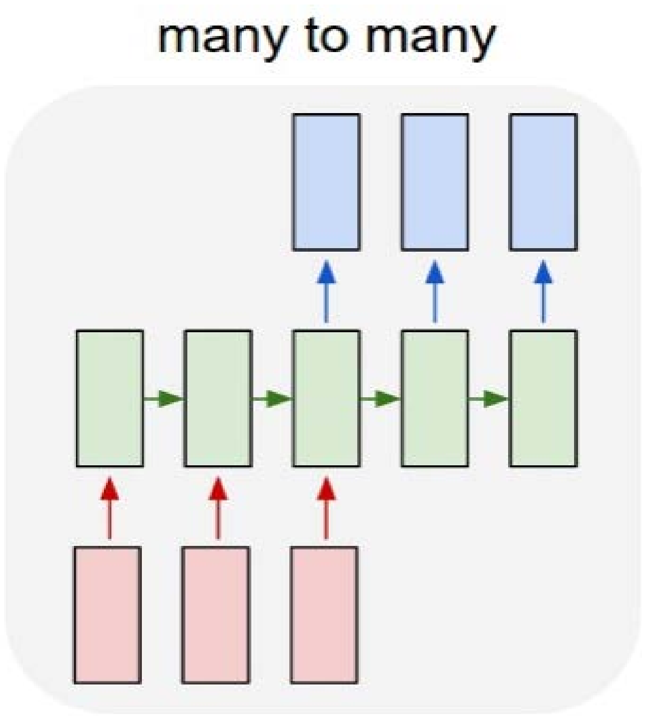
All the information from the source sentence has to pass through the bottleneck at the last unit(s) of the encoder.



Sentence length varies, but the encoding always has a fixed size.

Soft Attention for Translation

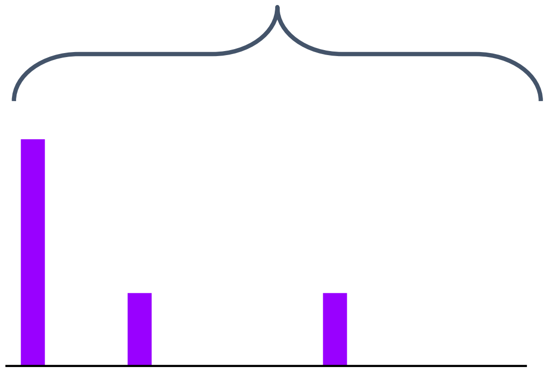
- “I love coffee” -> “Me gusta el café”



Bahdanau et al, “Neural Machine Translation by Jointly Learning to Align and Translate”, ICLR 2015

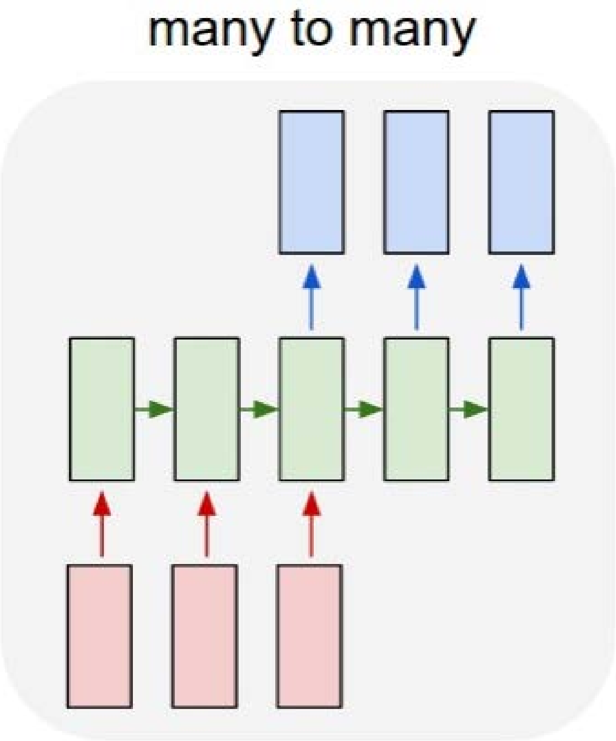
Soft Attention for Translation

Distribution over input words



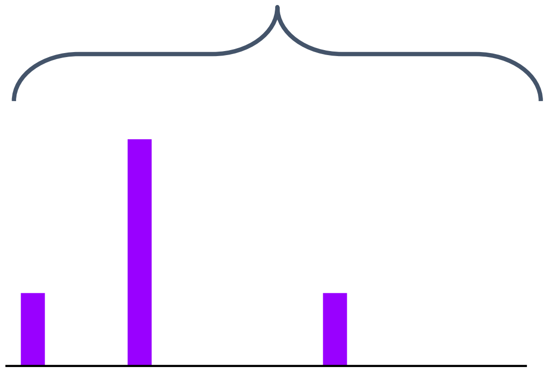
- "I love coffee" -> "Me gusta el café"

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



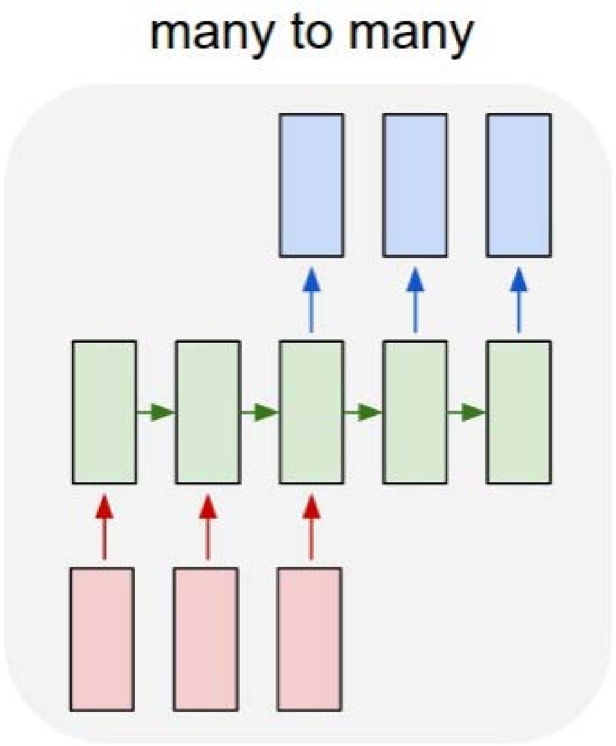
Soft Attention for Translation

Distribution over input words



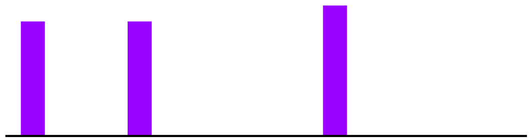
"I love coffee" -> "Me gusta el café"

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



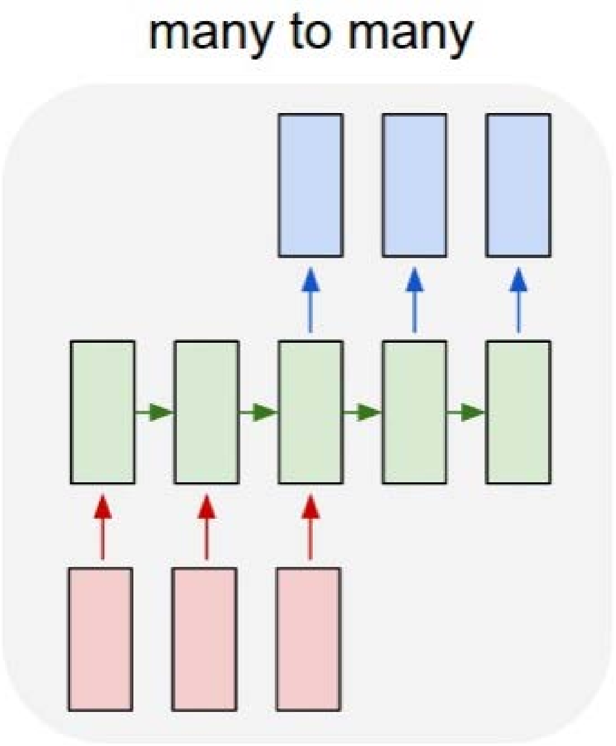
Soft Attention for Translation

Distribution over input words



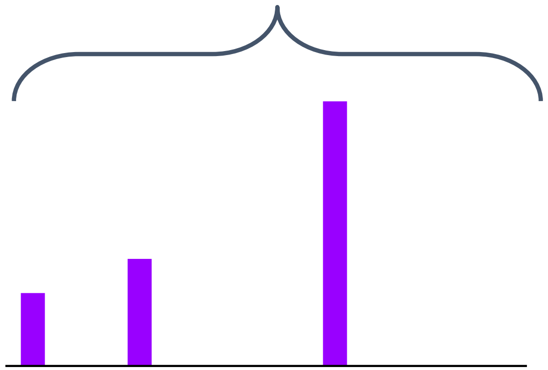
"I love coffee" -> "Me gusta el café"

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

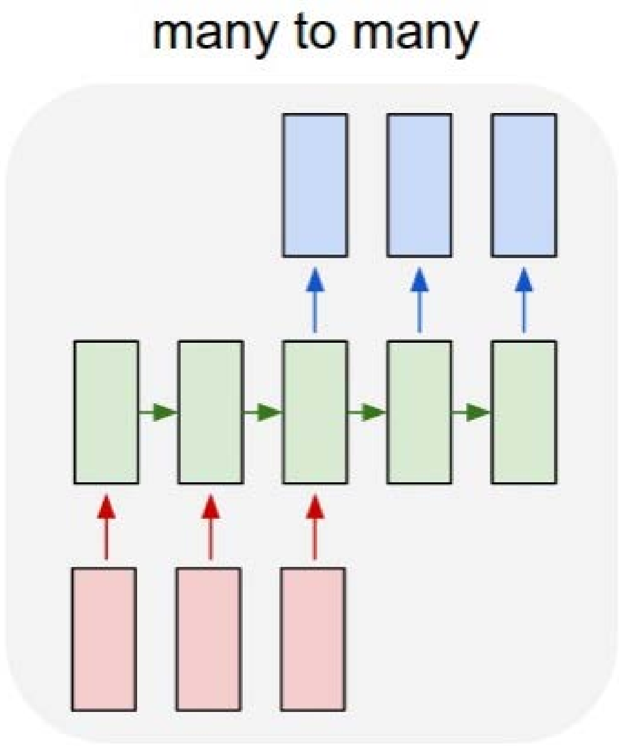


Soft Attention for Translation

Distribution over input words



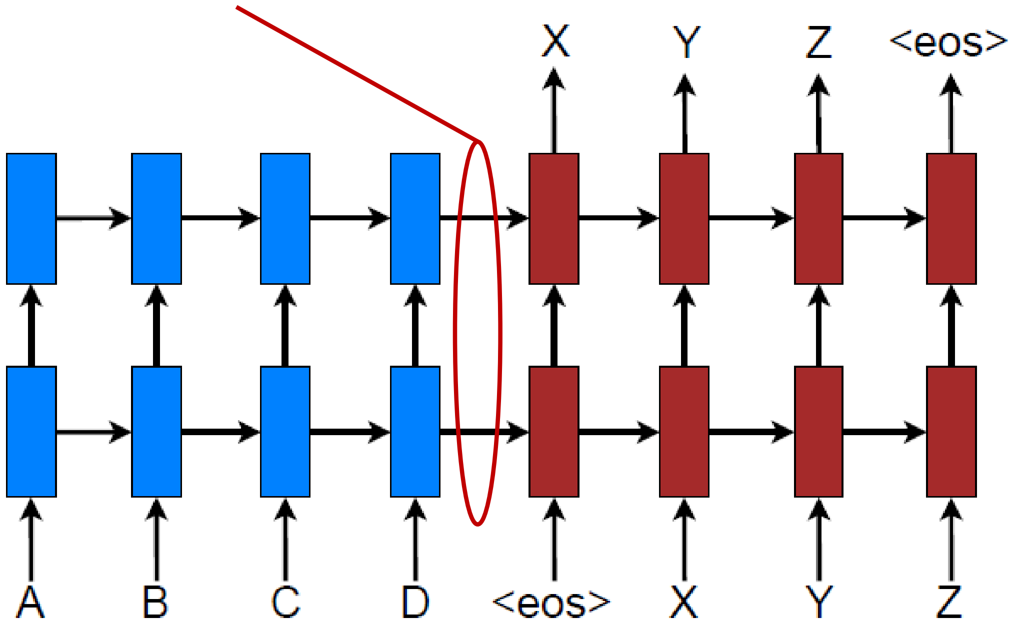
"I love coffee" -> "Me gusta el café"



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

Sequence-To-Sequence Criticisms

All the information from the source sentence has to pass through the bottleneck at the last unit(s) of the encoder.

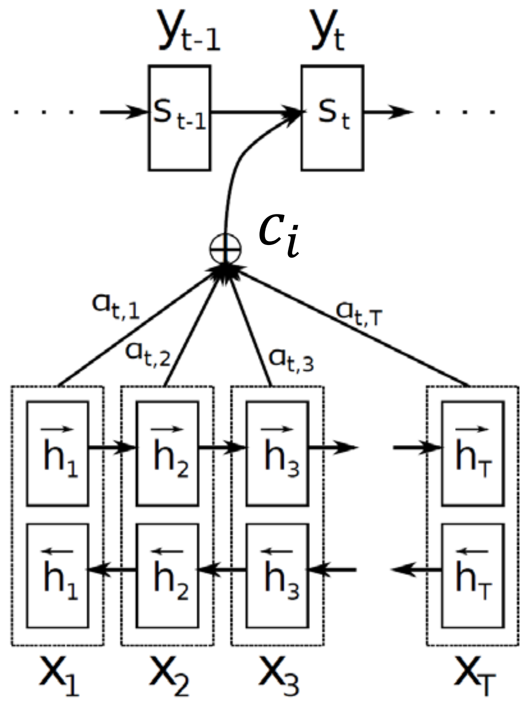


Sentence length varies, but the encoding always has a fixed size.

Soft Attention for Translation – Bahdanau et al. model

For each output word, focus attention on a subset of all input words.

Decoder



Encoder
(bidirectional RNN)

Context vector (input to decoder):

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

Mixture weights:

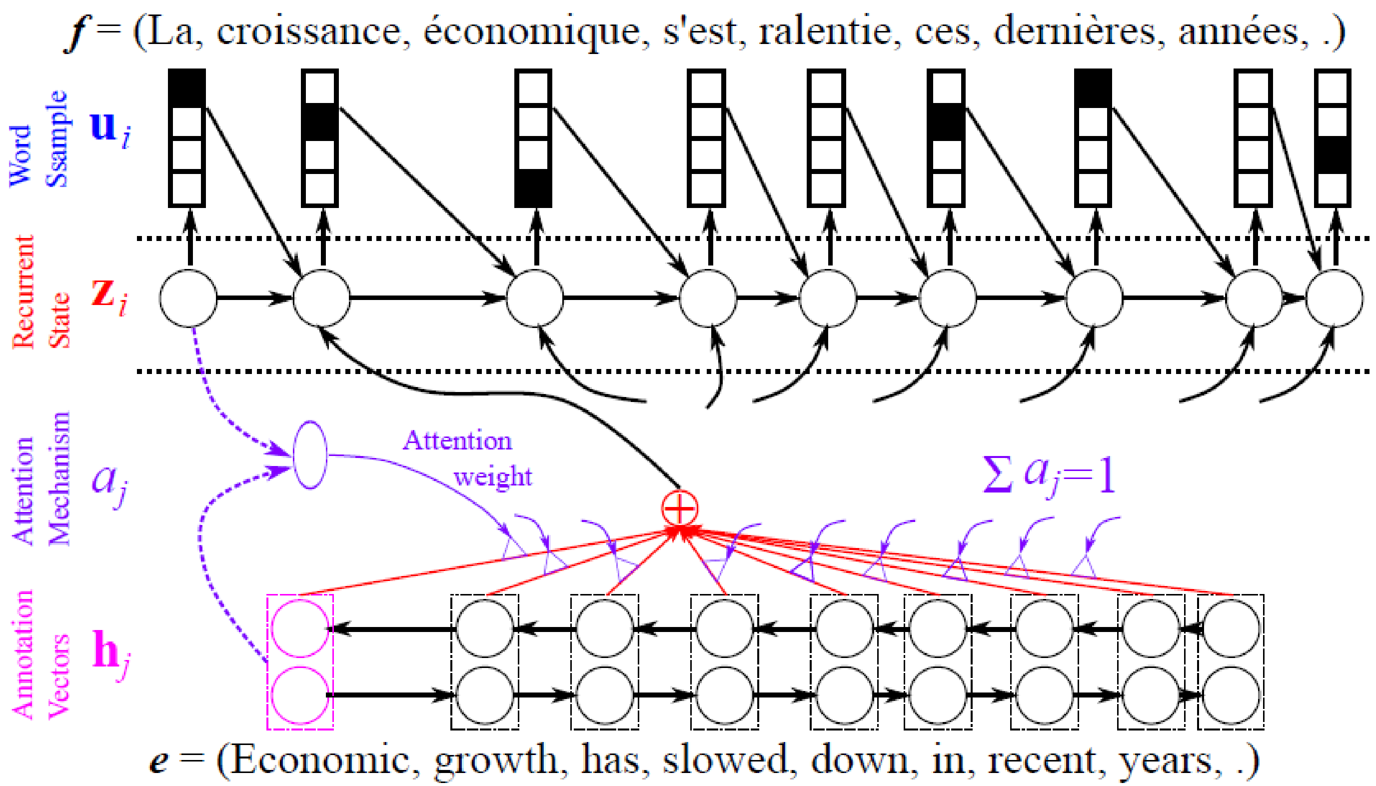
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

Alignment score (how well do input words near j match output words at position i):

$$e_{ij} = a(s_{i-1}, h_j)$$

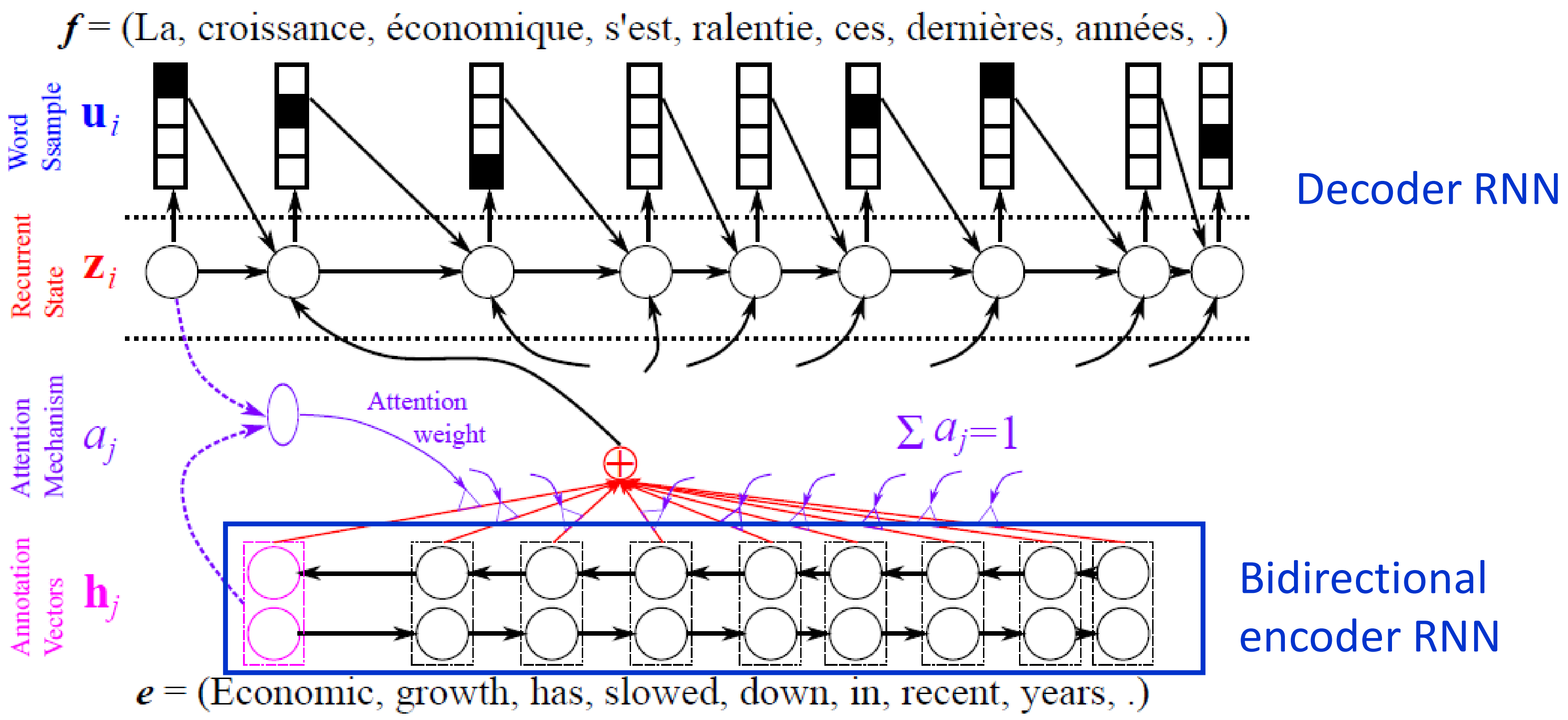
Bahdanau et al, "Neural Machine Translation by Jointly Learning to Align and Translate", ICLR 2015

Soft Attention for Translation



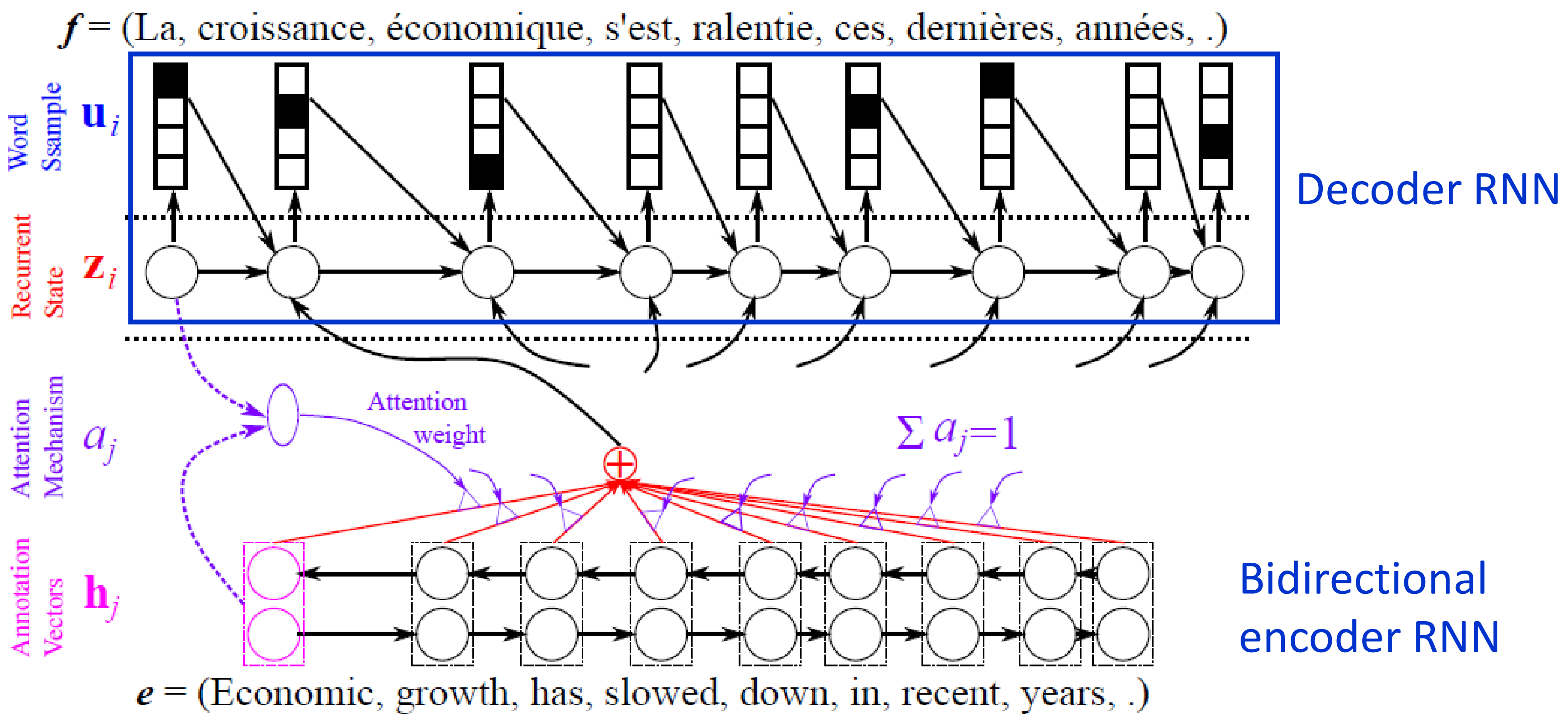
From Y. Bengio CVPR 2015 Tutorial

Soft Attention for Translation



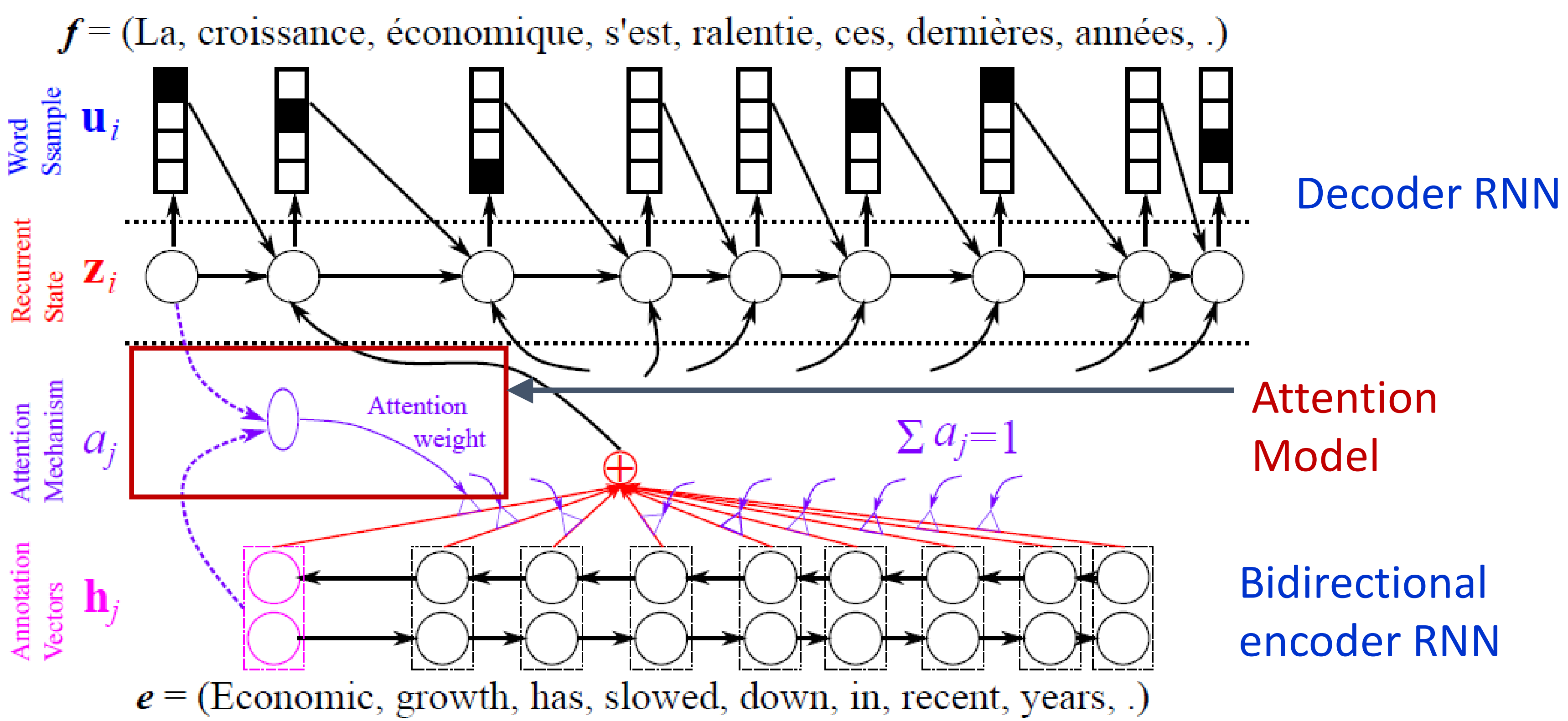
From Y. Bengio CVPR 2015 Tutorial

Soft Attention for Translation



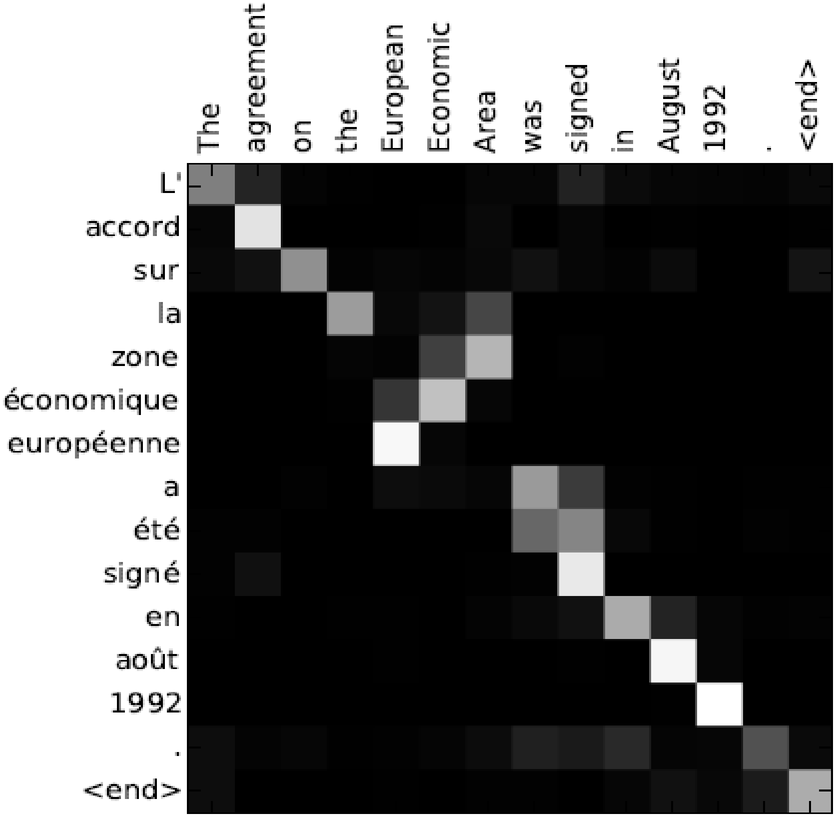
From Y. Bengio CVPR 2015 Tutorial

Soft Attention for Translation

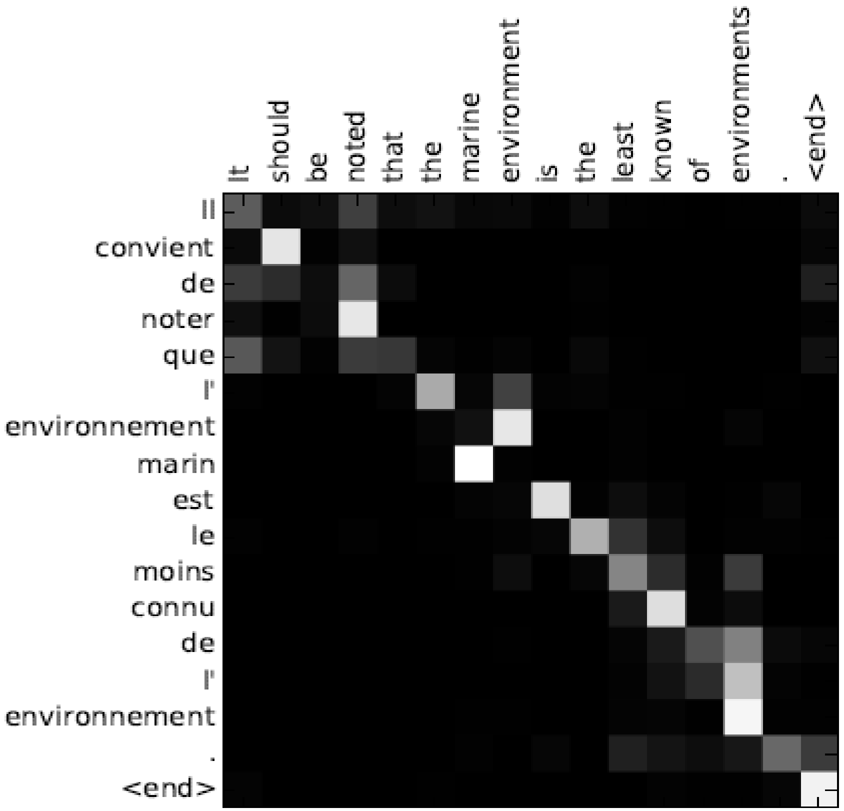


From Y. Bengio CVPR 2015 Tutorial

Soft Attention for Translation



(a)



(b)

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

Soft Attention for Translation

Reached State of the art in one year:

(a) **English→French (WMT-14)**

	NMT(A)	Google	P-SMT
NMT	32.68	30.6*	**37.03•**
+Cand	33.28	–	
+UNK	33.99	32.7°	
+Ens	36.71	**36.9°**	

(b) **English→German (WMT-15)**

Model	Note
24.8	Neural MT
24.0	U.Edinburgh, Syntactic SMT
23.6	LIMSI/KIT
22.8	U.Edinburgh, Phrase SMT
22.7	KIT, Phrase SMT

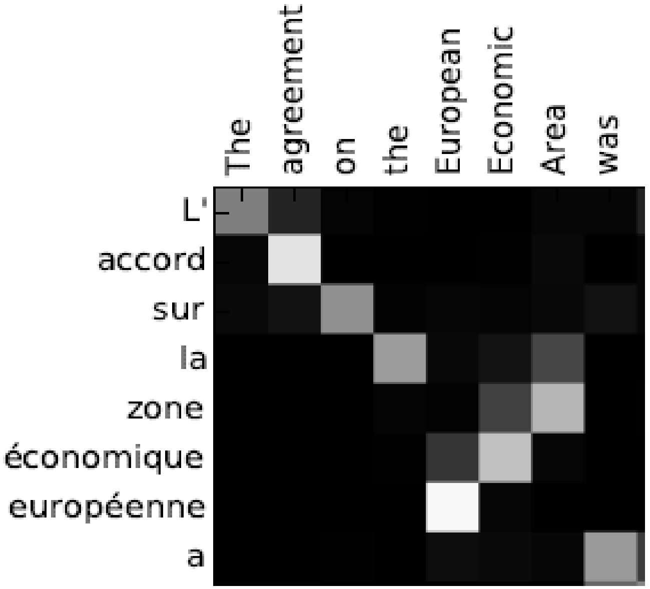
(c) **English→Czech (WMT-15)**

Model	Note
18.3	Neural MT
18.2	JHU, SMT+LM+OSM+Sparse
17.6	CU, Phrase SMT
17.4	U.Edinburgh, Phrase SMT
16.1	U.Edinburgh, Syntactic SMT

Yoshua Bengio, NIPS RAM workshop 2015

Criticism of Bahdanau et al.

The attention function $a(s_{i-1}, h_j)$ is rather complex, yet the attention often seems to be a simple heat map on word similarity:



The data path in Badanau is quite complicated: the attention path is another recurrent path between output states.

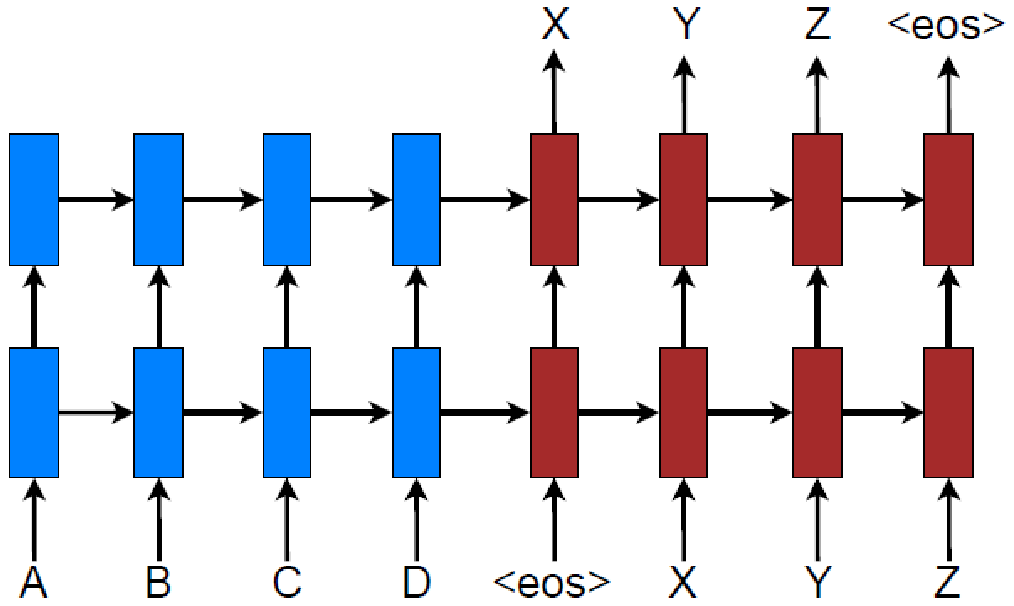
Doesn't generalize to deeper networks (shown to be Important by Sutskeyver et al.).

Luong and Manning added several architectural improvements.

Effective Approaches to Attention-based Neural Machine Translation
Minh-Thang Luong, Hieu Pham, Christopher D. Manning, EMNLP 15

Luong, Pham and Manning 2015

Stacked LSTM with arbitrary depth (c.f. bidirectional flat encoder in Bahdanau et al):

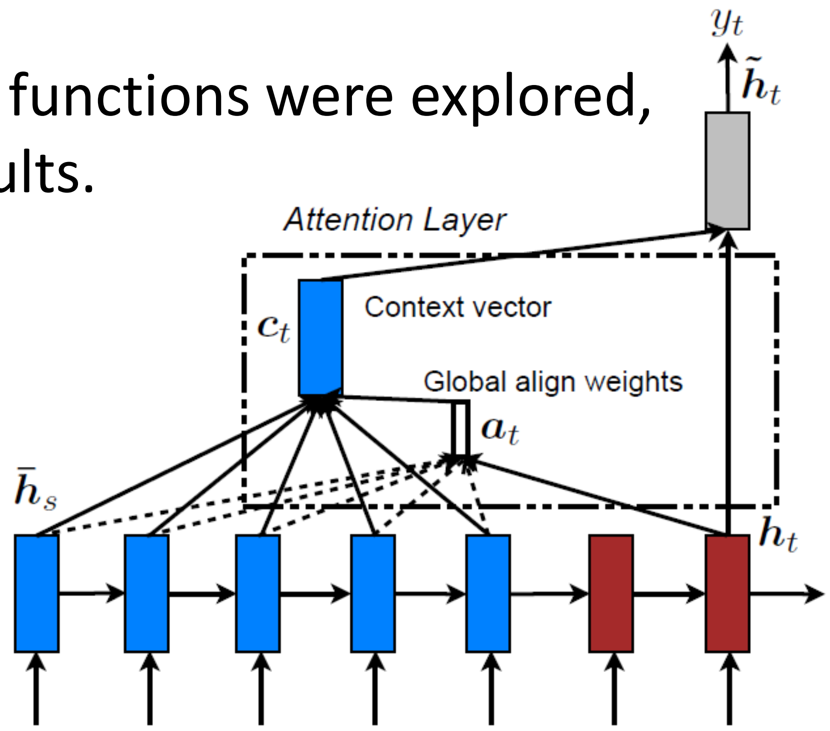


Effective Approaches to Attention-based Neural Machine Translation
Minh-Thang Luong, Hieu Pham, Christopher D. Manning, EMNLP 15

Global Attention Model

Global attention model is similar but simpler than Badanau's. It sits above the encoder/decoder and is not itself recurrent.

Different word matching functions were explored, some yielding better results.

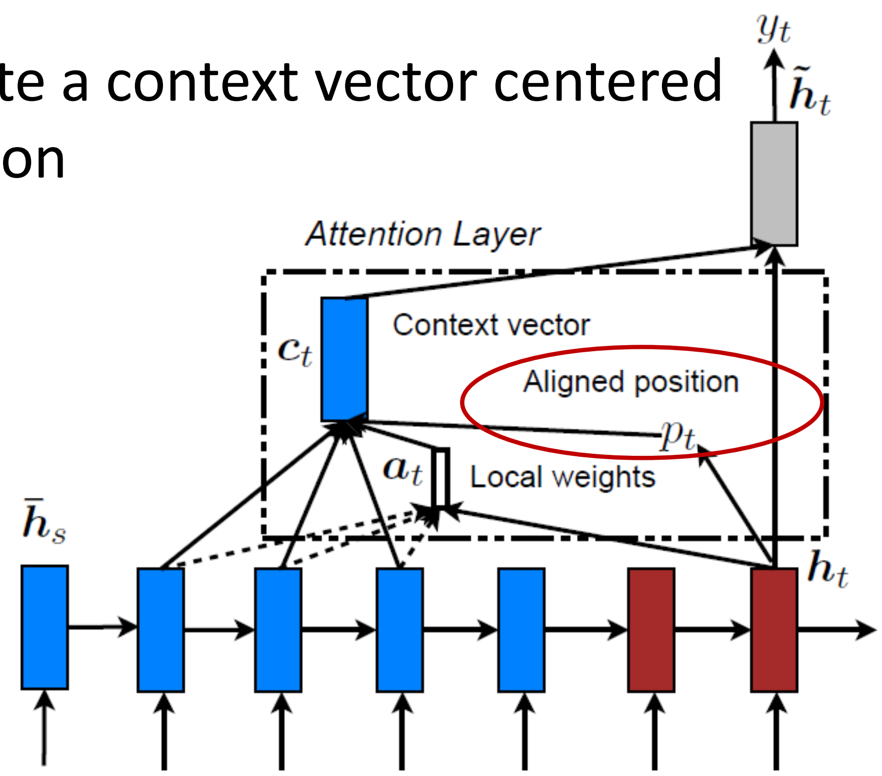


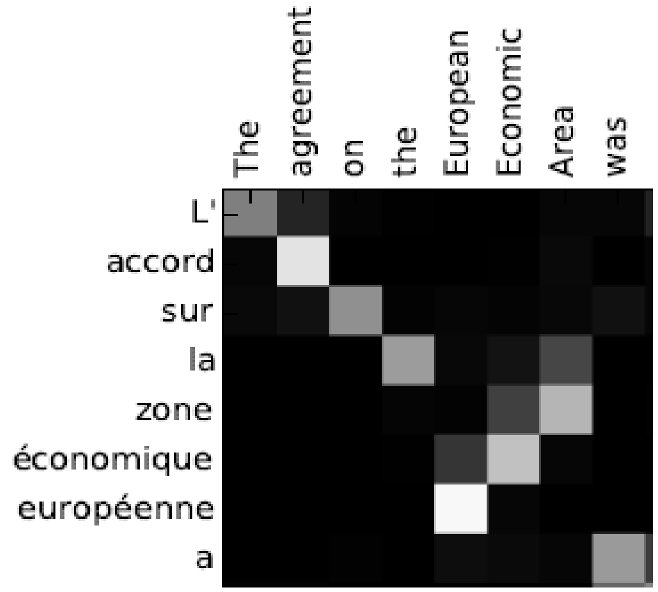
Effective Approaches to Attention-based Neural Machine Translation
Minh-Thang Luong Hieu Pham Christopher D. Manning, EMNLP 15

Local Attention Model

- Compute a best aligned position p_t first

- Then compute a context vector centered at that position





Effective Approaches to Attention-based Neural Machine Translation
Minh-Thang Luong Hieu Pham Christopher D. Manning, EMNLP 15

Luong, Pham and Manning's Translation System (2015):

System	BLEU
Top – *NMT + 5-gram rerank* (Montreal)	24.9
Our ensemble 8 models + unk replace	**25.9**

Table 2: **WMT'15 English-German results** – *NIST* BLEU scores of the winning entry in WMT'15 and our best one on newstest2015.

System	Ppl.	BLEU
WMT'15 systems		
SOTA – *phrase-based* (Edinburgh)		**29.2**
NMT + 5-gram rerank (MILA)		27.6
Our NMT systems		
Base (reverse)	14.3	16.9
+ global (*location*)	12.7	19.1 (+*2.2*)
+ global (*location*) + feed	10.9	20.1 (+*1.0*)
+ global (*dot*) + drop + feed	9.7	22.8 (+*2.7*)
+ global (*dot*) + drop + feed + unk		24.9 (+*2.1*)

Table 3: **WMT'15 German-English results** –

Attention-only Translation Models

Problems with recurrent networks:

- Sequential training and inference: time grows in proportion to sentence length. Hard to parallelize.
- Long-range dependencies have to be remembered across many single time steps.
- Tricky to learn hierarchical structures (“car”, “blue car”, “into the blue car”…)

Alternative:

- Convolution – but has other limitations.

Self-Attention

Information flows from within the same subnetwork (either encoder or decoder). Convolution applies fixed transform weights. Self-attention applies variable weights (but typically not transformations):

image from Lukas Kaiser, Stanford NLP seminar

Self-Attention “Transformers” (not spatial transformers)

- Constant path length between any two positions.
- Variable receptive field (or the whole input sequence).
- Supports hierarchical information flow by stacking self-attention layers.
- Trivial to parallelize.
- Attention weighting controls information propagation.

- **Can replace word-based recurrence entirely.**

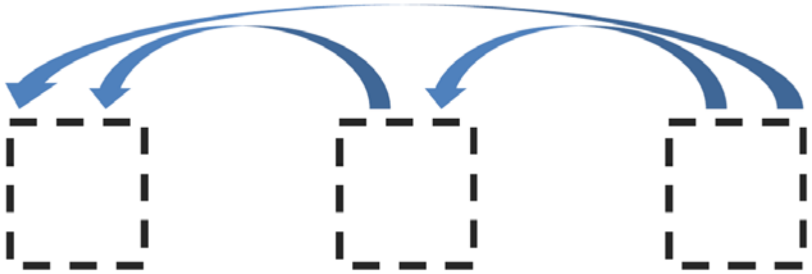
Vaswani et al. “Attention is all you need”, arXiv 2017

Attention in Transformer Networks

We saw this in Bahdanau and Luong models

Encoder-Decoder Attention

Encoder Self-Attention



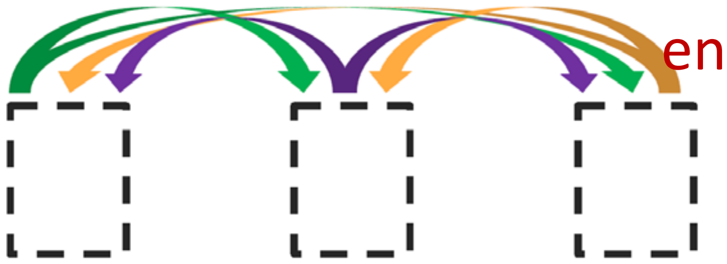
MaskedDecoder Self-Attention

image from Lukas Kaiser, Stanford NLP seminar

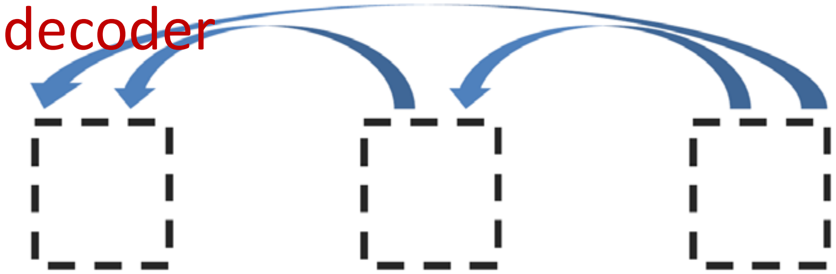
Attention in Transformer Networks

Encoder-Decoder Attention

Replaces word recurrence in encoder and decoder



Encoder Self-Attention



MaskedDecoder Self-Attention

image from Lukas Kaiser, Stanford NLP seminar

Attention in Transformer Networks

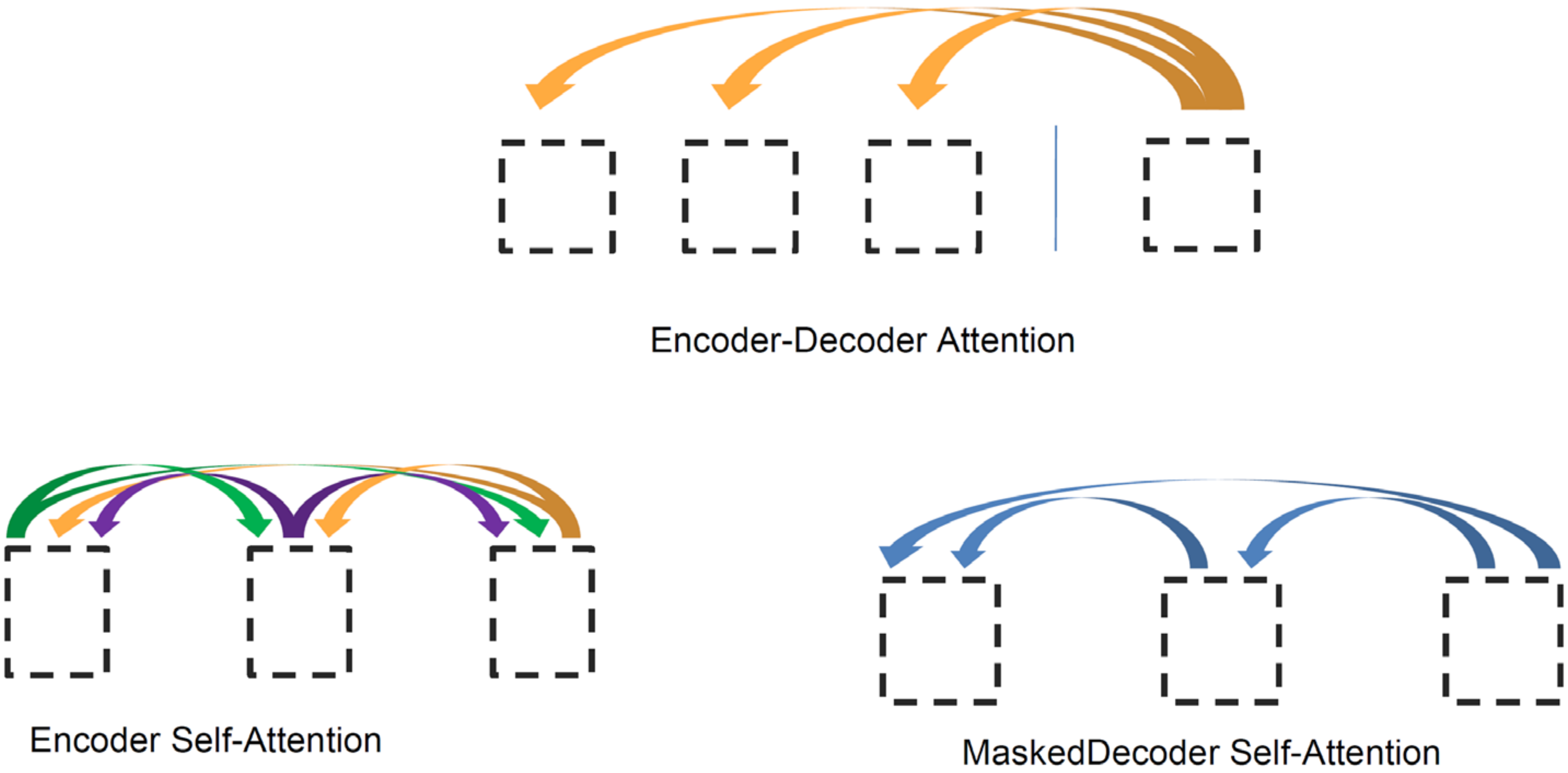
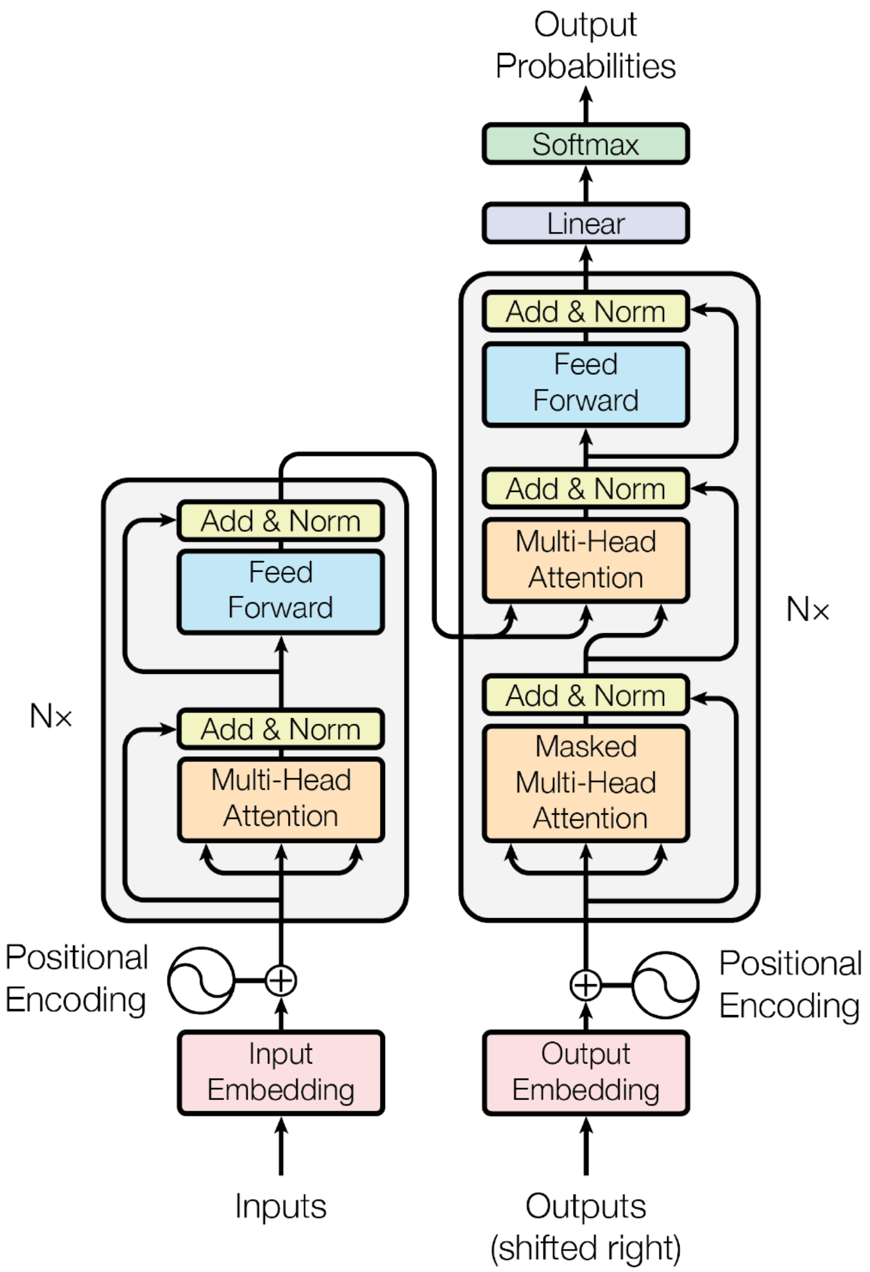


image from Lukas Kaiser, Stanford NLP seminar

Masking limits attention to earlier units: y_i depends only on y_j for $j < i$.

The Transformer

- Basic unit shown at right.
- In experiments, stacked with N=6.
- Output words fed back as input, shifted right. Can use beam search as before.
- Inputs and outputs are embedded in vector spaces of fixed dimension.
- Positional encoding: when words are combined through attention, their location is lost. Positional encoding adds it back.



Attention Implementation

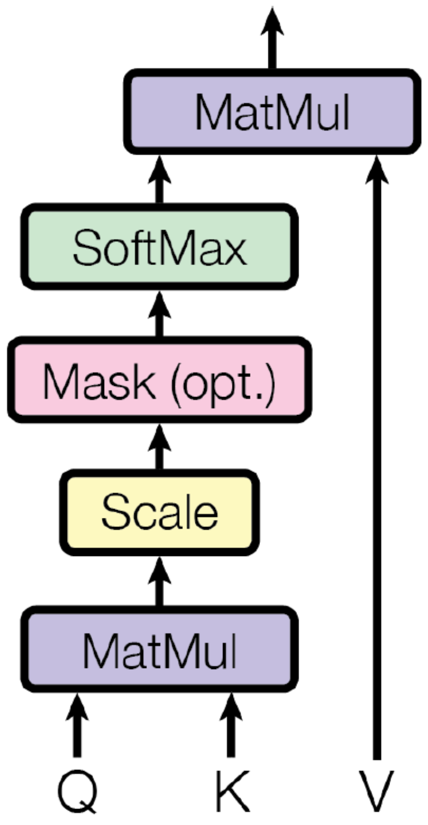
- Attention is modeled as a key-value store:
  Q = query vector
  K = key
  V = value

Encoder-decoder layer: the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. (Similar to Bahdanau).

Self-attention layer: all of the keys, values and queries come from the output of the previous layer in the encoder.

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

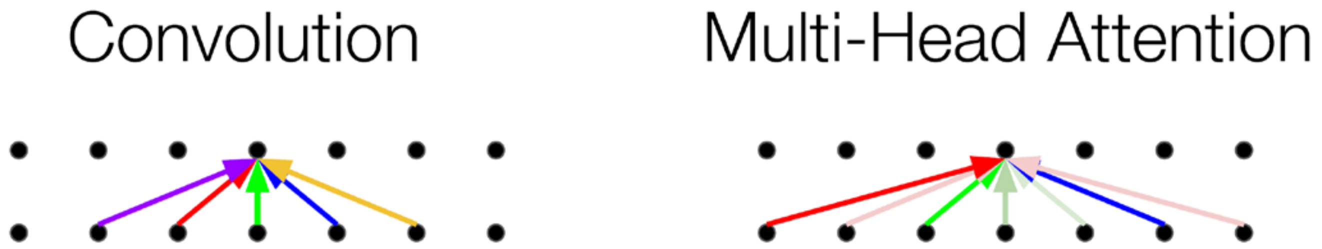
Scaled Dot-Product Attention

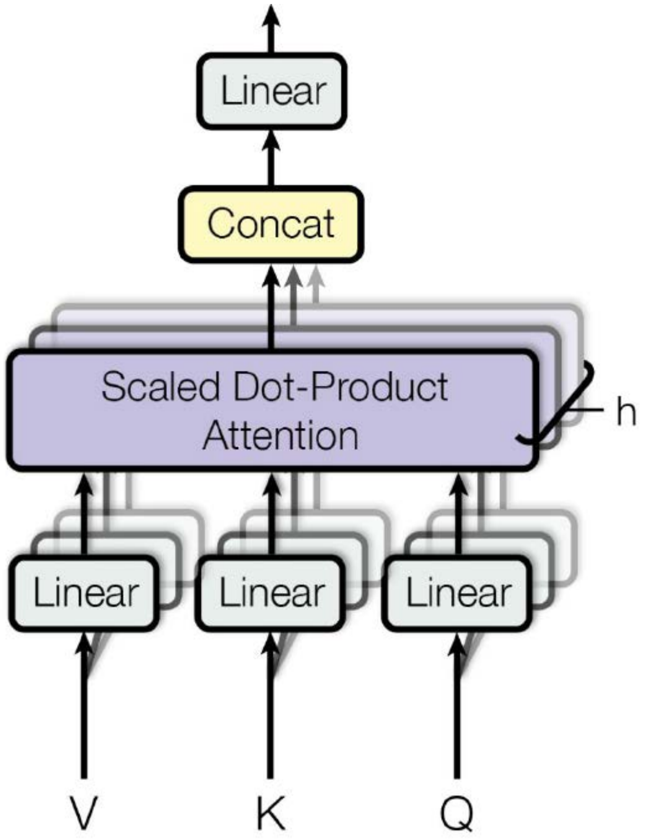


Multi-Headed Attention

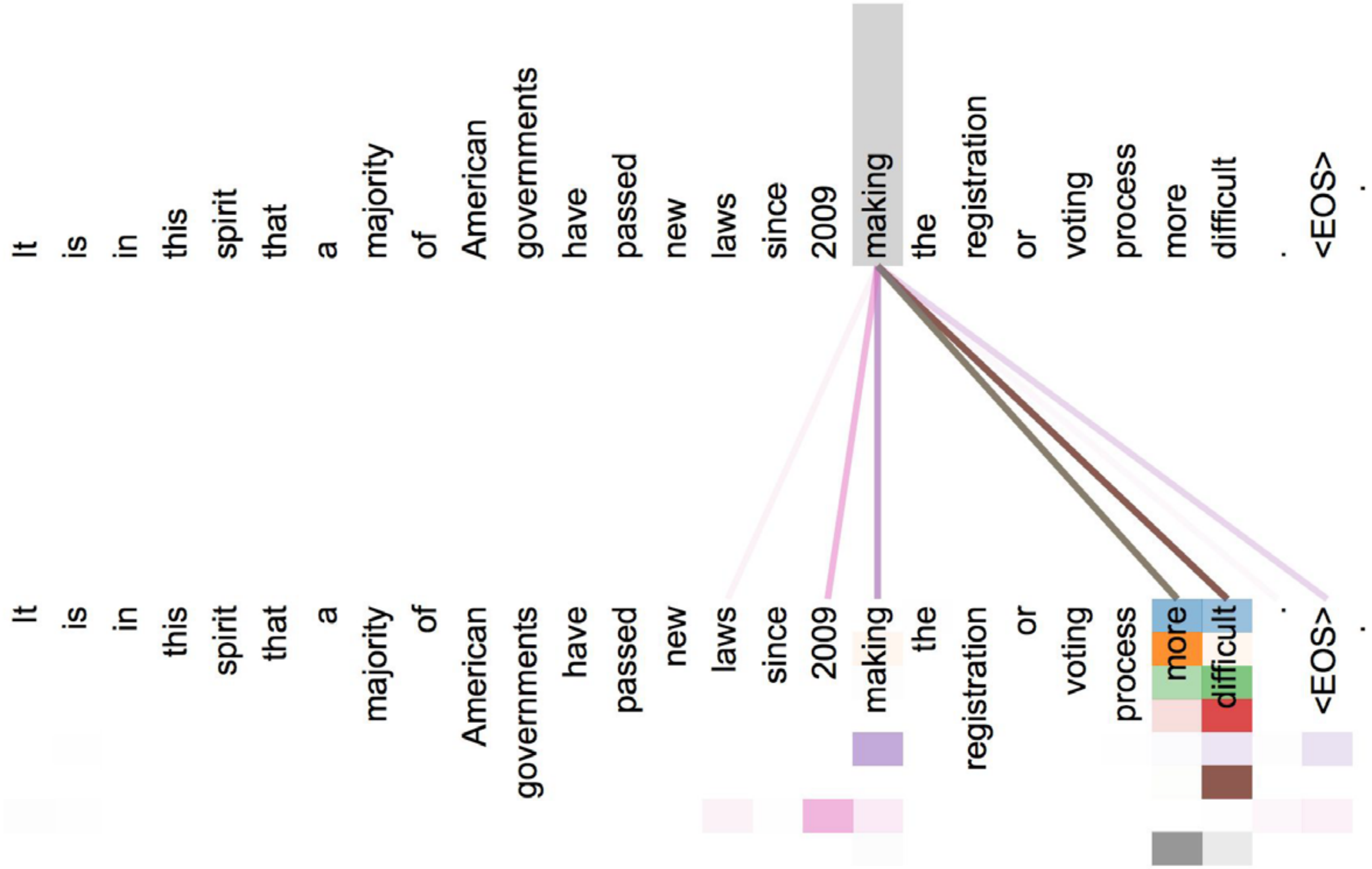
- Simple attention blends the results of all the attended-to inputs. It doesn't allow a per-input transformation, as convolution does.

- The solution is to use "multi-headed attention":

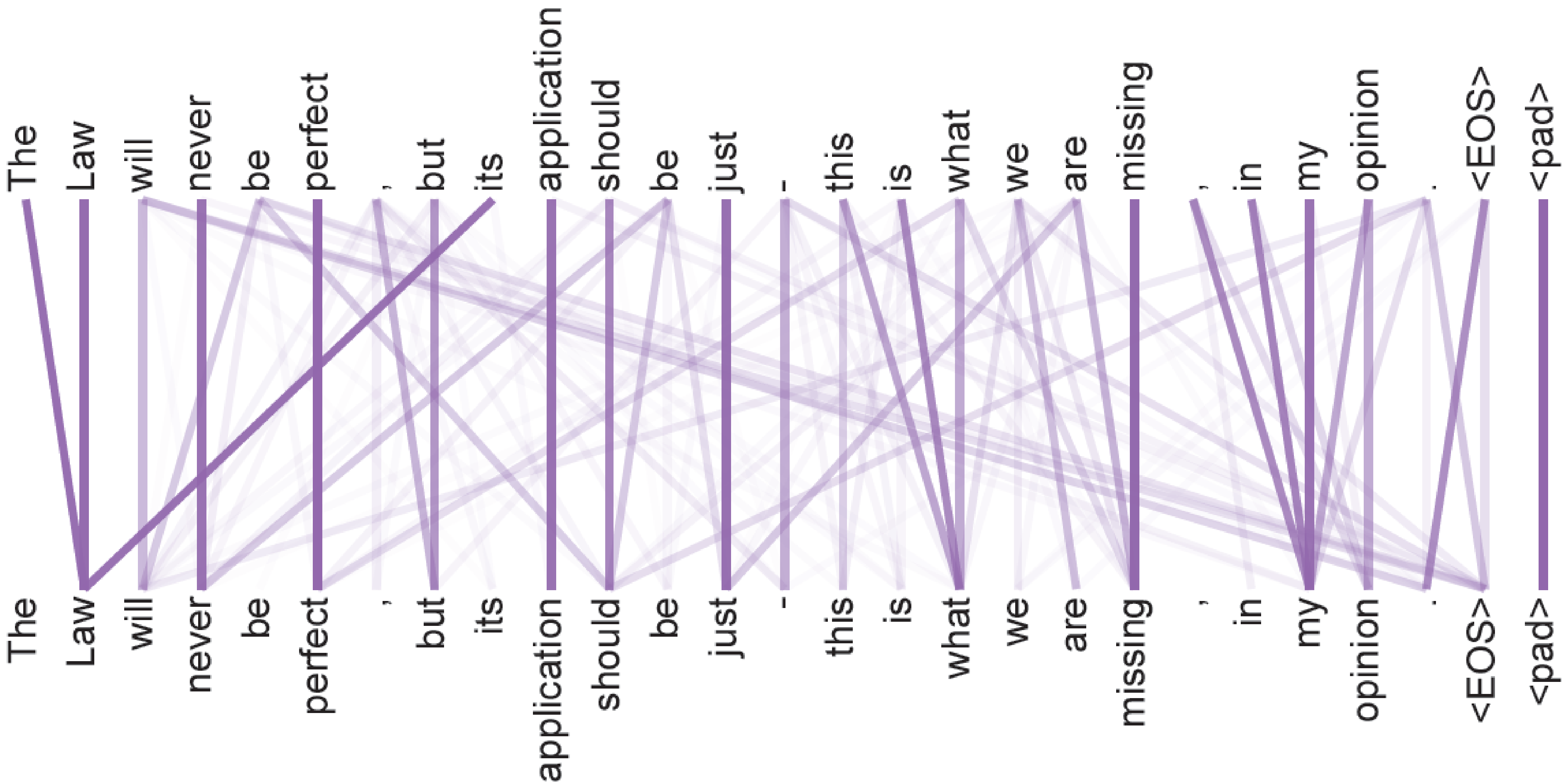




Multi-Headed Attention

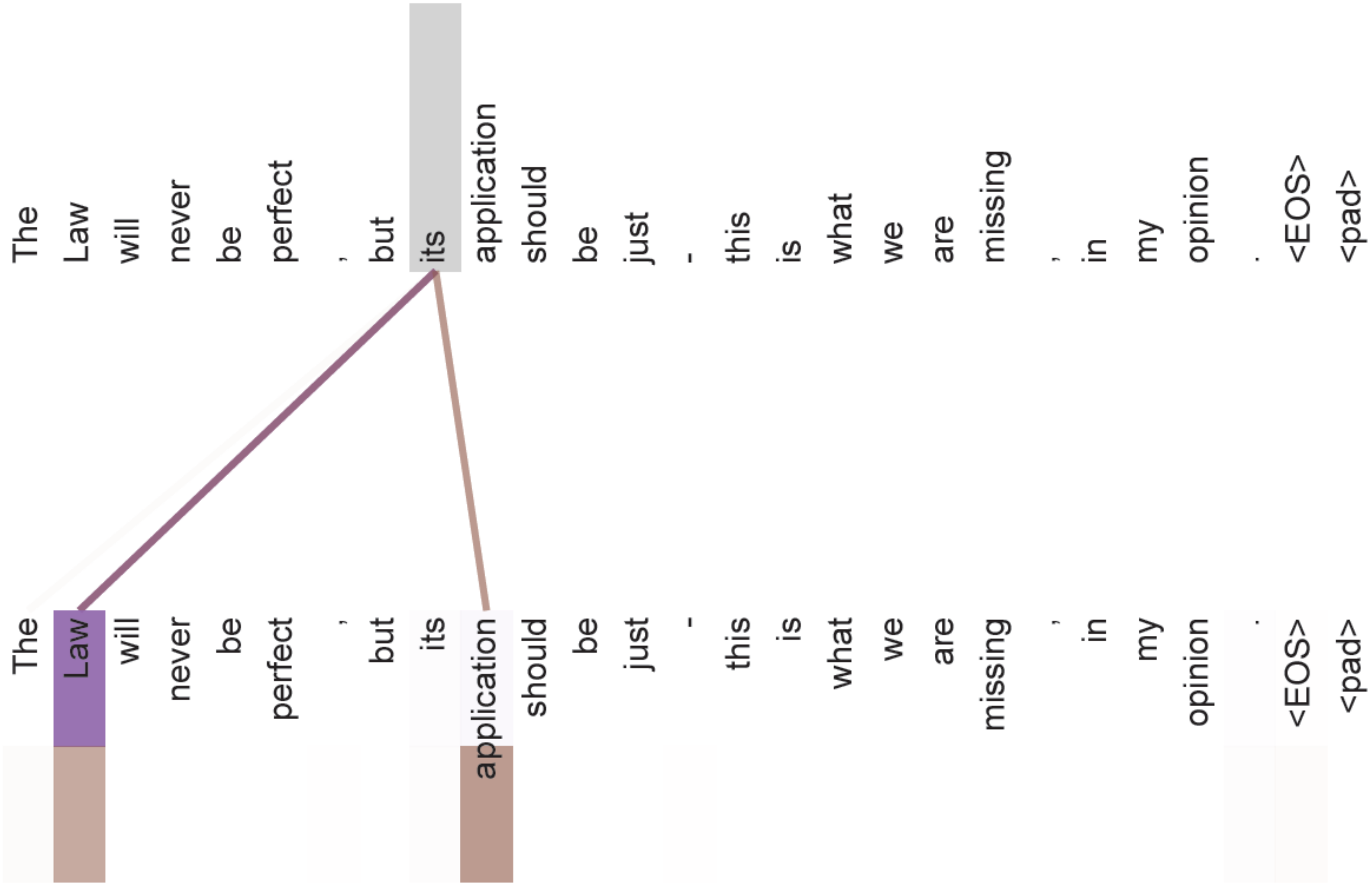


Multi-Headed Attention



Anaphora (pronoun or article) resolution

Multi-Headed Attention



Anaphora (pronoun or article) resolution

Transformer Results

Machine Translation Results: WMT-14

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [17]	23.75			
Deep-Att + PosUnk [37]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [36]	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 [31]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [37]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [36]	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	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

English-to-English Translation ?!

Yes, it does make sense. a.k.a. summarization.

Liu et al, “GENERATING WIKIPEDIA BY SUMMARIZING LONG SEQUENCES” arXiv 2018

M = input length, N = output length

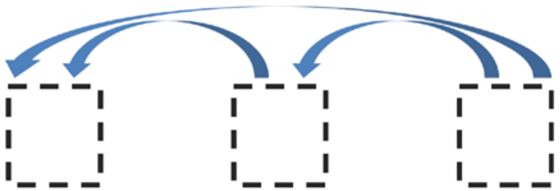
Summarization: M >> N

$O(MN)$

Encoder-Decoder Attention

$O(M^2)$

Encoder Self-Attention



$O(N^2)$

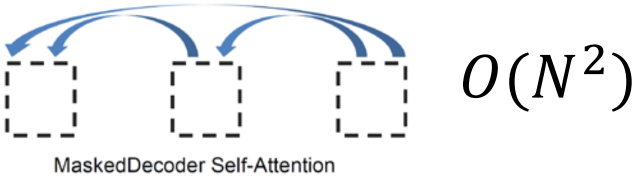
MaskedDecoder Self-Attention

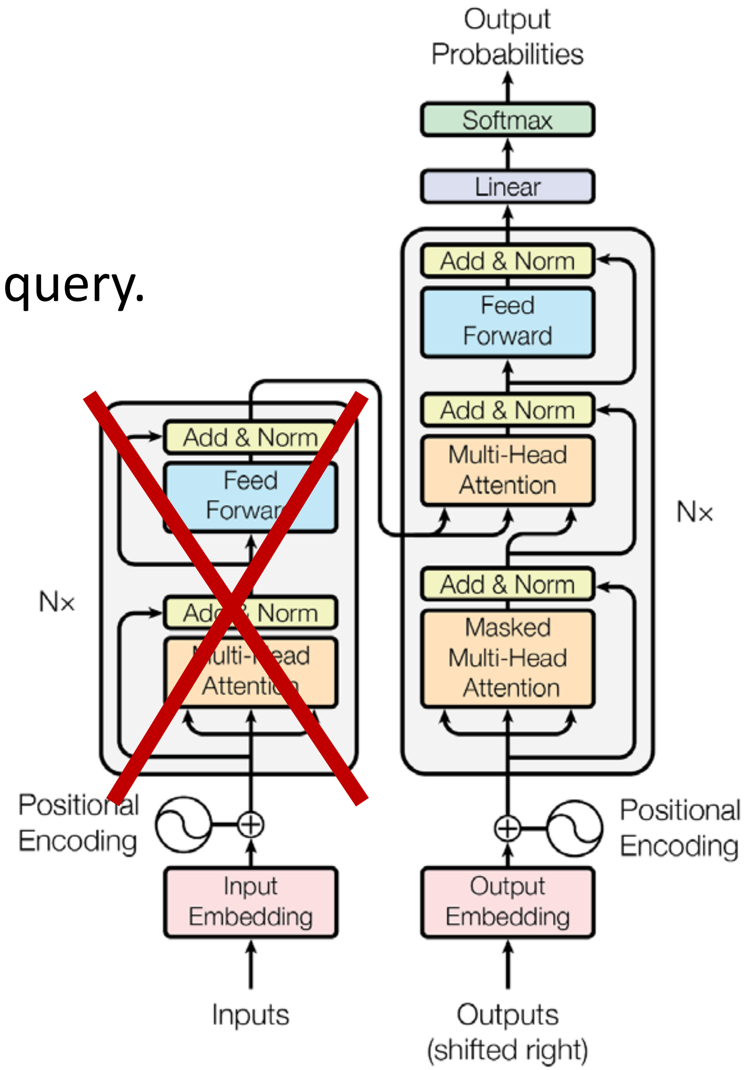
Large-scale Summarization (Wikipedia)

Like translation, but we completely remove the encoder.

Source data (large!):

- The references for a Wikipedia article.
- Web search using article section titles, ~ 10 web pages per query.





Large-scale Summarization

Results:

Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

L = input window length.
ED = encoder-decoder.
D = decoder only.
DMCA = a memory compression technique (strided convolution).
MoE = mixture of experts layer.

Liu et al, “GENERATING WIKIPEDIA BY SUMMARIZING LONG SEQUENCES” arXiv 2018

Translation Takeaways

- Sequence-to-sequence translation

    - Input reversal
    - Narrow beam search

- Adding Attention

    - Compare latent states of encoder/decoder (Bahdanau).
    - Simplify and avoid more recurrence (Luong).

Translation Takeaways

- Parsing as translation:

  - Translation models can solve many "transduction" tasks.

- Attention only models:

  - Self-attention replaces recurrence, improves performance.

  - Use depth to model hierarchical structure.

  - Multi-headed attention allows interpretation of inputs.