# Machine Learning for Natural Language Processing

*Guillaume Gravier*

guillaume.gravier@irisa.fr

# Outline of the course

**Lectures** (6 x 3h)

- Lecture #1: Introduction and representation of words

  Notions: morphology, tokens, lemmas, POS, word net, word embedding

  Hands-on: manipulate basic pipelines and visualize word embeddings

- Lecture #2: Representation of documents

  Notions: vocabulary, Zipf's curse, bag of words, Bayes, RNN, BERT

  Hands-on: basic neural network classifiers

- Lecture #3: Language models

  Notions: ngrams, LSTM, bi-LSTM, language generation

  Hands-on: train a small LM and generate text

- Lecture #4: Transformers and large language models

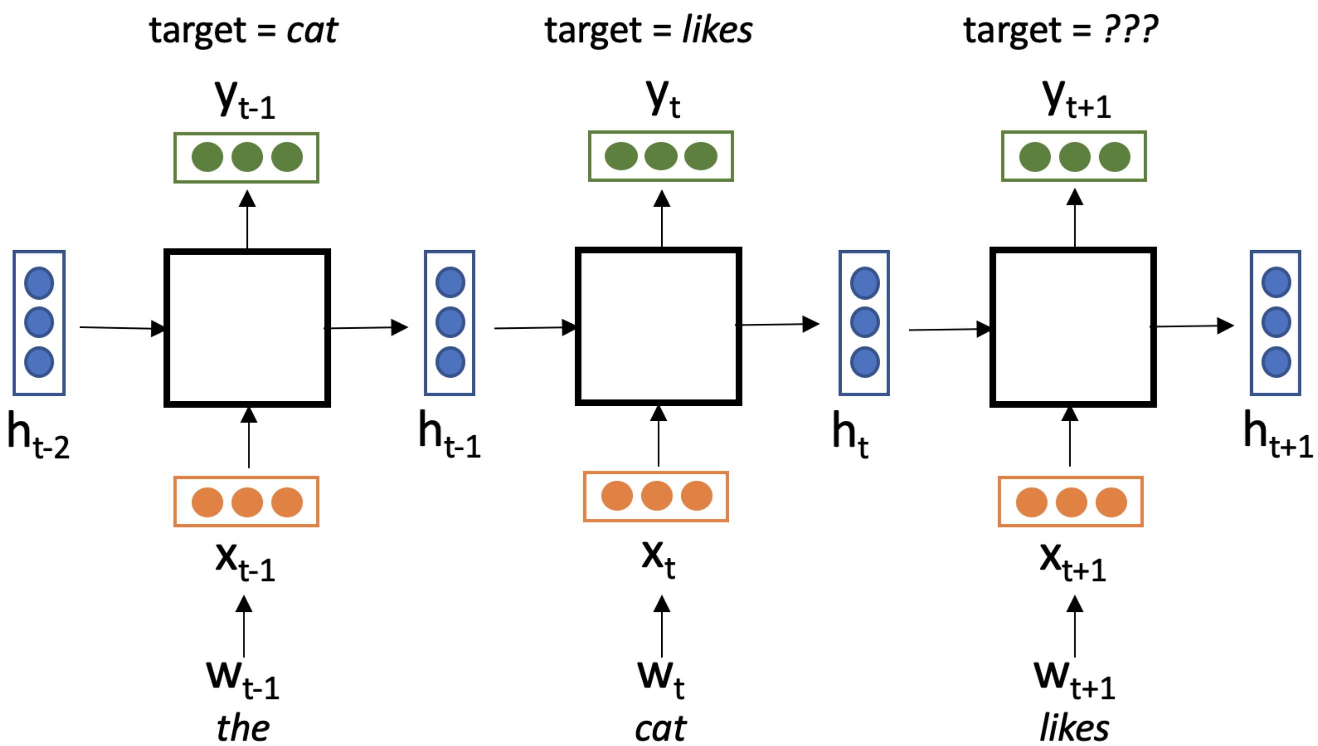
  Notions: encoder/decoder, transformers, fine-tuning

  Hands-on: visualize embeddings, fine-tune a LLM

# Language modeling with recurrent networks

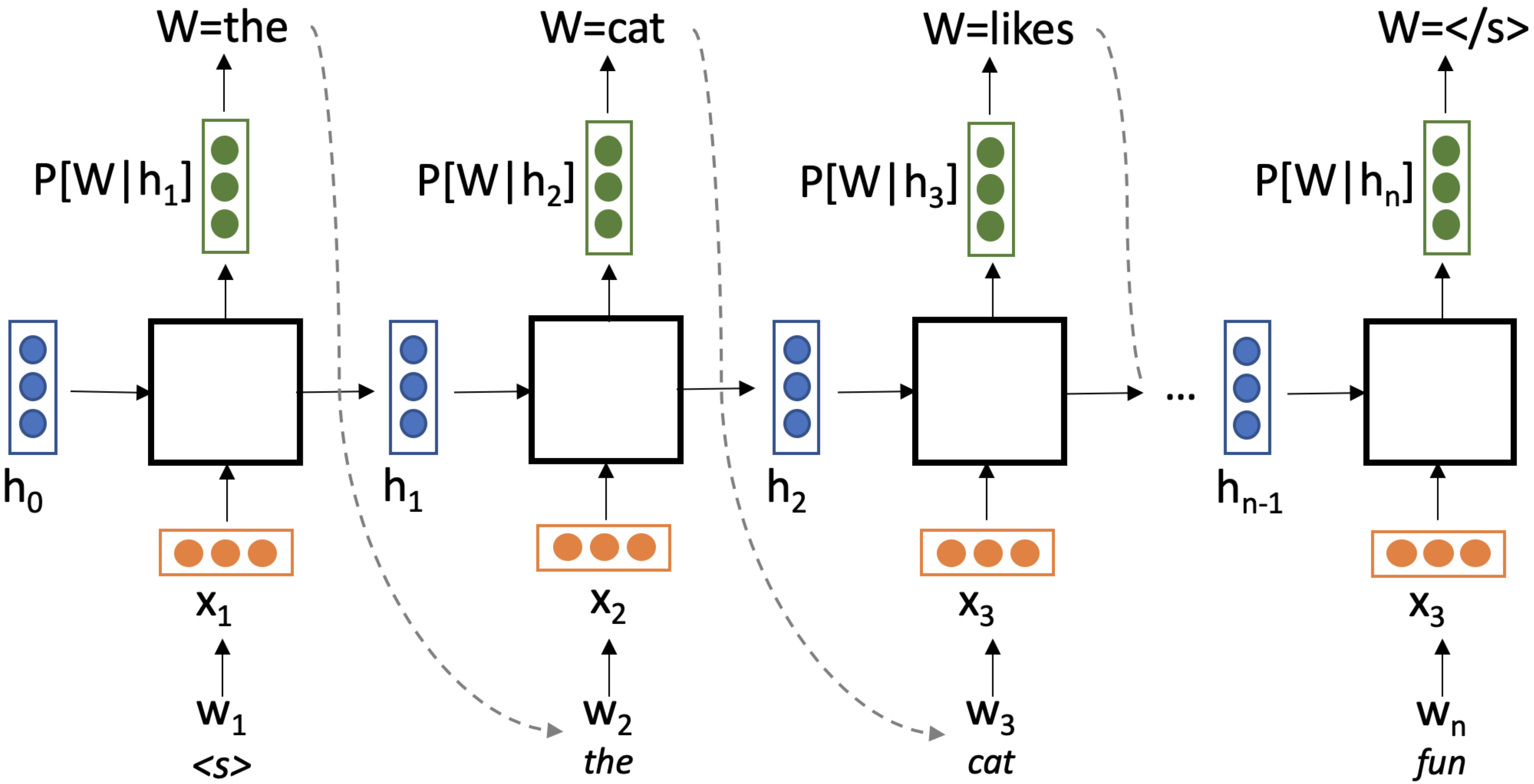
Use the hidden state vector $h_{i-1}$ as (a summary of) the history of word $w_i$ to predict

$$P[w_i|\underbrace{w_{i-1}\dots w_1}_{\text{full history}}] \doteq f(w_i, h_{i-1})$$




Much less parameters than in the feed-forward approach!

# Sampling from RNNs is trivial



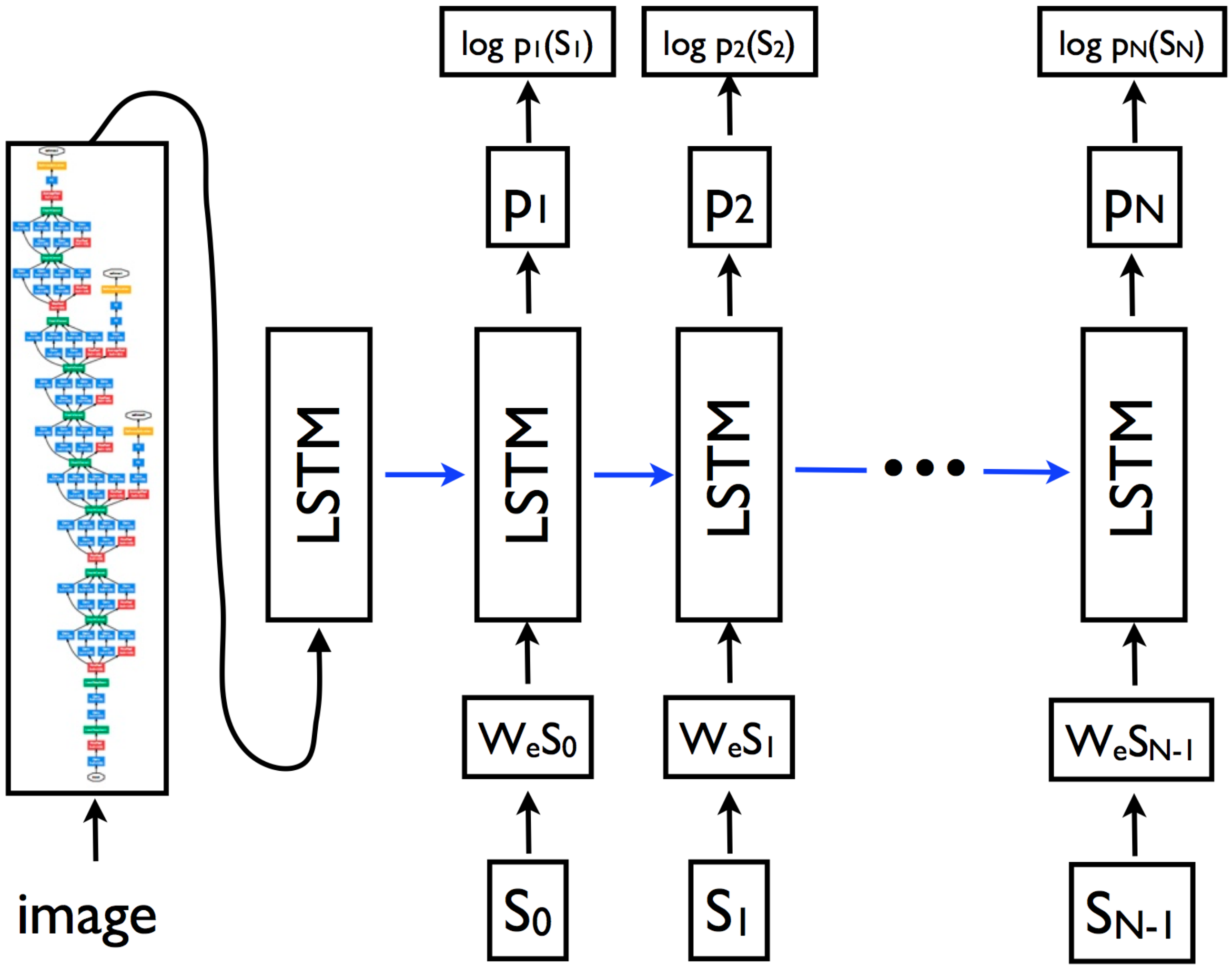

**Yet no control on semantics, cannot draw from** $P[W_1, \ldots, W_n|X]$

# Sampling algorithm

- $h_0 = \text{random}(\text{dim})$ $\leftarrow$ random init
- $w_1 = " < s > "$ $\leftarrow$ start of sentence
- for $t = 1, \ldots, T'$
  - $x_t = \text{embed}(w_t)$
  - $h_t = \text{RNNCell}(h_{t-1}, x_t)$
  - $y_t = \text{softmax}(Ah_t + b)$ $\leftarrow$ distrib. over vocab
  - $w_{t+1} = \text{choose}(y_t)$ $\leftarrow$ choose best predicted word

# Conditioning language model on images

# Conditioning language model on images




Training parameters of the RNN on images with human-generated captions so as to maximize the likelihood of the (training) caption texts (image encoding model frozen)

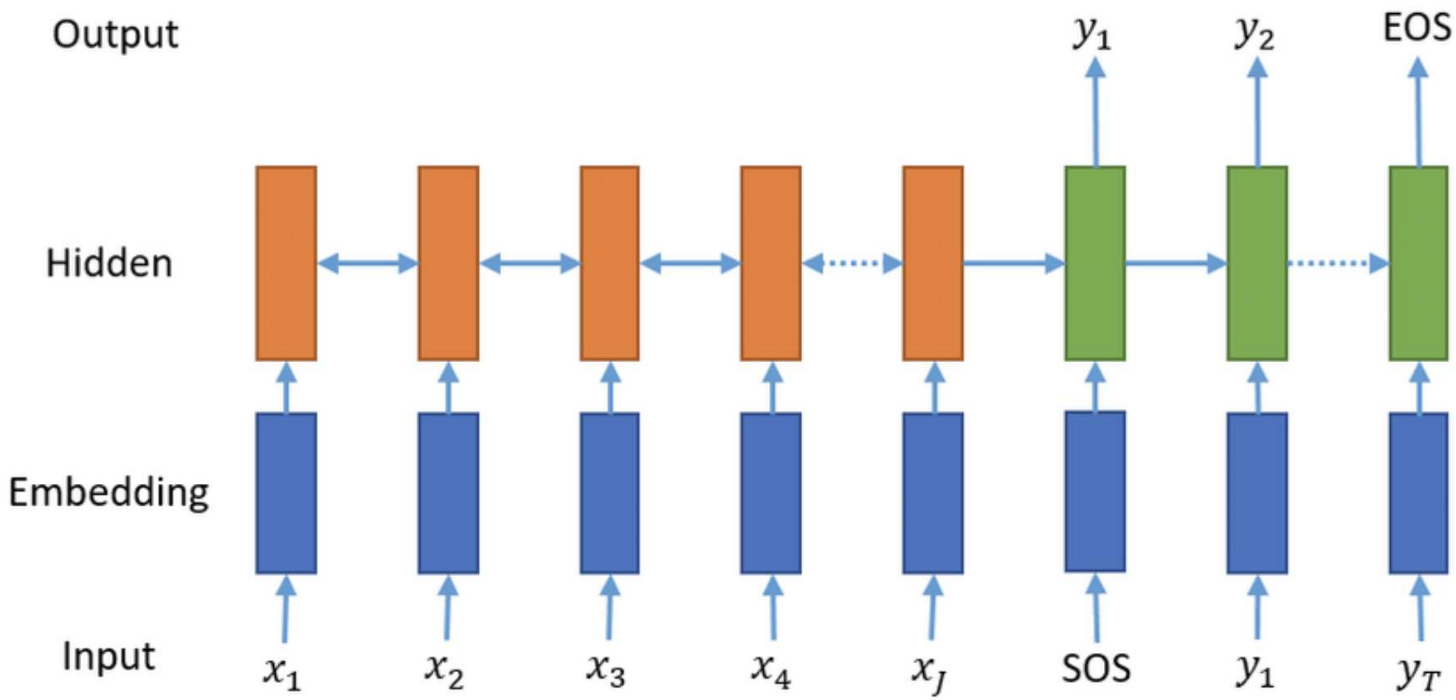
# Caption generation algorithm

- $h_0 = \text{encode}(\text{image})$ $\leftarrow$ encode image
- $w_1 = "<s>"$ $\leftarrow$ start of sentence
- for $t = 1, \ldots, T'$
  - $x_t = \text{embed}(w_t)$
  - $h_t = \text{RNNCell}(h_{t-1}, x_t)$
  - $y_t = \text{softmax}(Ah_t + b)$ $\leftarrow$ distrib. over vocab
  - $w_{t+1} = \text{choose}(y_t)$ $\leftarrow$ choose (best) predicted word

# Conditioning LMs on text (aka seq2seq)

Sequence to sequence encoder/decoder systems combine

- a RNN to encode a message from a prompt/text, i.e., $h_0 = \text{RNN}_e(x_1, \ldots, x_n)$
- a RNN to generate a message conditioned on $h_0$, i.e., $w_1, \ldots, w_n = \text{RNN}_d(h_0)$




borrowed from Tian Shi et al., 2018. Neural Abstractive Text Summarization with Sequence-to-Sequence Models.

# The seq2seq (sort of) maths

**Encoder**

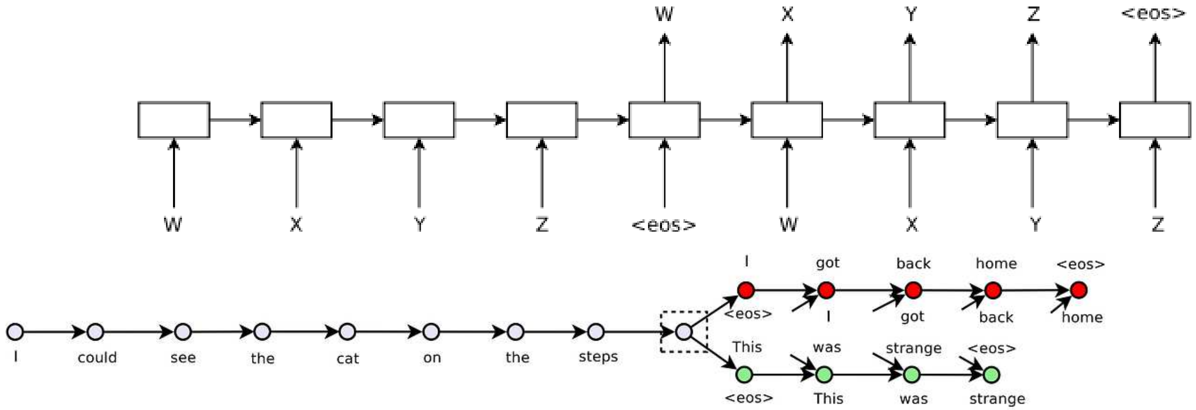
- $h_0 = \text{zeros}(\text{hidden\_dim})$
- for $t = 1, \ldots, T$
  - $x_t = \text{embed}(w_t)$
  - $h_t = \text{RNNCell1}(h_{t-1}, x_t)$

**Decoder**

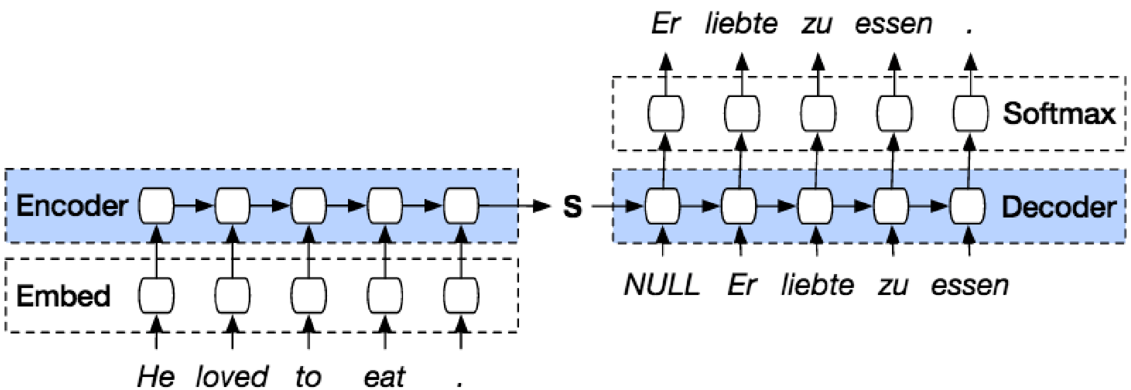
- $h'_0 = h_T$
- $w'_1 = " < s > "$ $\leftarrow$ start of sentence
- for $t = 1, \ldots, T'$
  - $x'_t = \text{embed}(w'_t)$
  - $h'_t = \text{RNNCell2}(h'_{t-1}, x'_t)$
  - $y_t = \text{softmax}(Ah'_t + b)$ $\leftarrow$ distrib. over vocab
  - $w'_{t+1} = \text{choose}(y_t)$ $\leftarrow$ choose best predicted word

# Applications of seq2seq models

- sentence embedding, e.g., auto-encoders, denoising auto-encoders




- machine translation




- abstractive summarization, question answering, etc.

# On practical aspects and use of encoder/decoder

- often convenient to also consider input sequence backward
  - process from $x_n$ to $x_1$
  - use a bidirectional encoder
- can layer RNNs, both in the encoder and decoder
- better use ground truth in decoder at training time (or alternate) – aka teacher forcing

**Yet not always good enough for two main reasons**

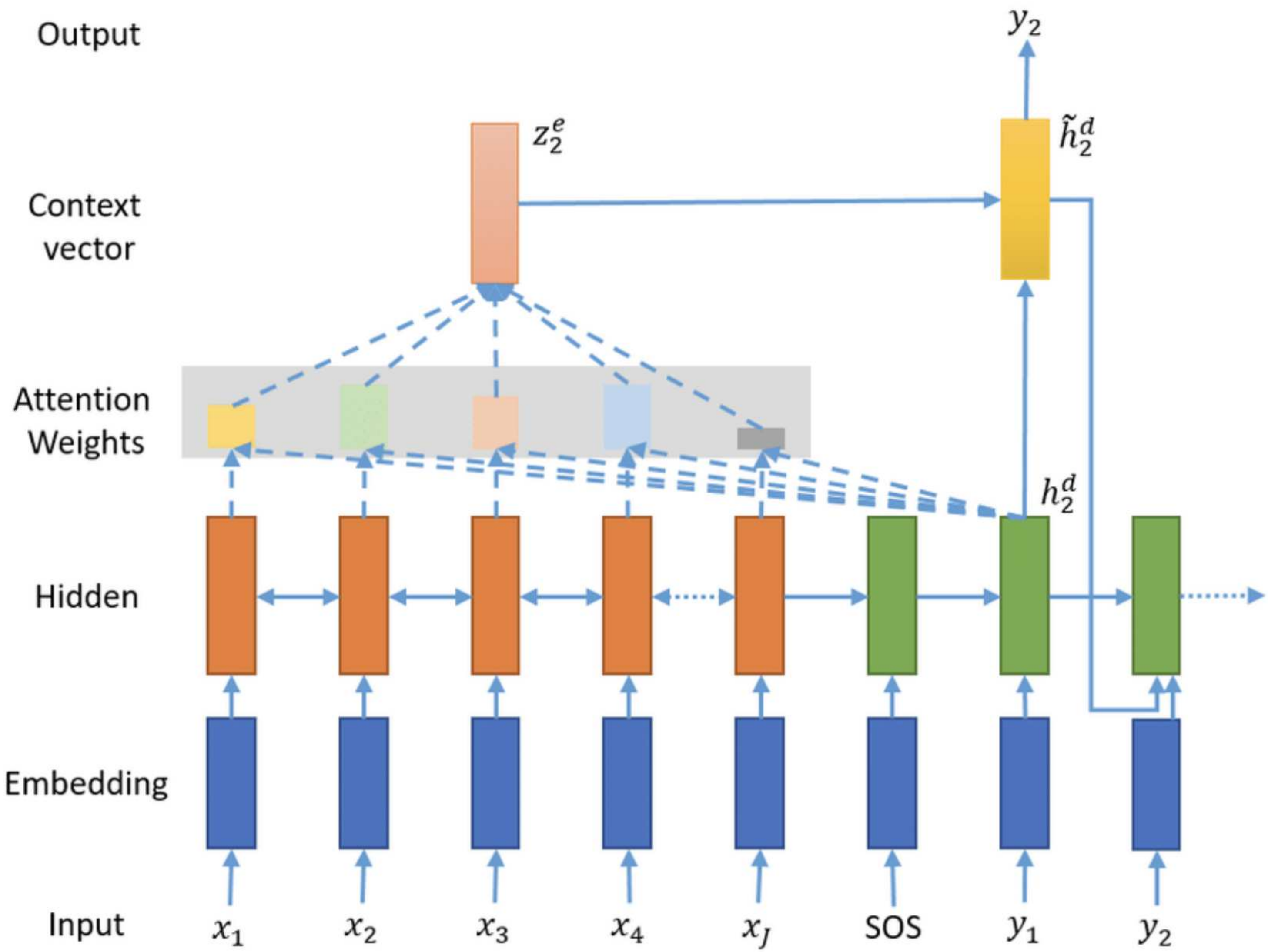
1. the input message needs be fully summarized in a single embedding $h_0$ (hence only rather simple inputs work in practice)
2. (almost) independent choice of words might lead to poor language
   - might not respect syntax
   - short or truncated outputs
   - repeats

See Ziang Xie's 2018 practical guide on neural text generation for further details,
`https://cs.stanford.edu/~zxie/textgen.pdf`

# Attention and transformers

ENSAI École nationale de la statistique et de l'analyse de l'information

# Attention mechanisms in seq2seq




borrowed from Tian Shi et al., 2018. Neural Abstractive Text Summarization with Sequence-to-Sequence Models.

# Attention mechanisms in seq2seq: the maths

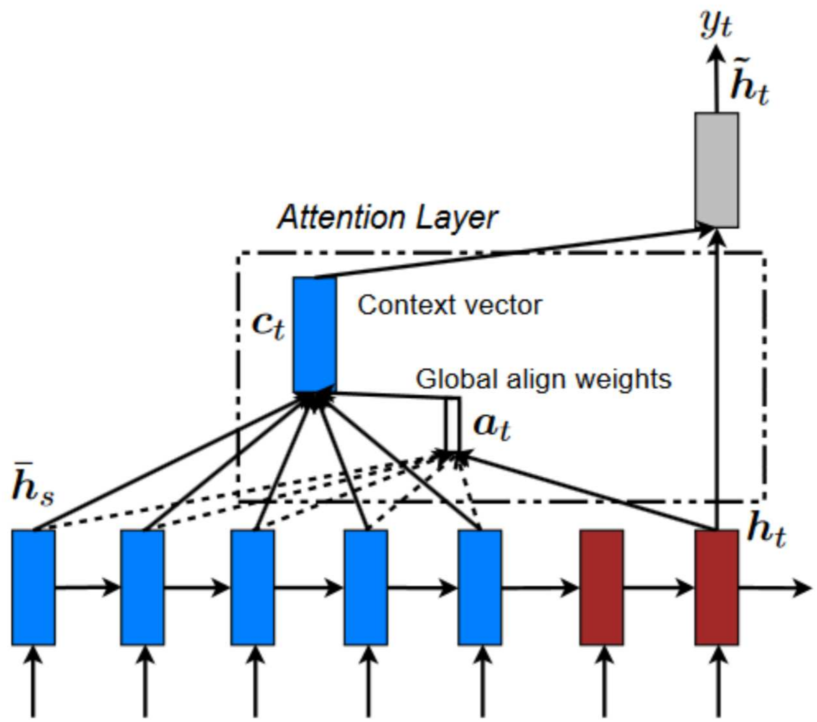
input $x = \{x_1, \ldots, x_n\}$, output $y$

$\overline{h}_s$: state in encoder at time $s \in [1, n]$

$h_t$: state in decoder at time $t$

$\tilde{h}_t$: modified state in decoder at time $t$

$$
\begin{aligned}
a_{ts} &= \text{softmax}(h_t^t W_a \overline{h}_s) \\
c_t &= \sum_{s=1}^{n} a_{ts} \overline{h}_s \\
\tilde{h}_t &= \tanh(W_c[c_t; h_t]) \\
p(\cdot | y_{<t}, x) &= \text{softmax}(W_s \tilde{h}_t)
\end{aligned}
$$




from Luong et al. 2015

**$a_{ij}$ = how much input $j$ matters for output $i$**

Luong et al. 2015. Effective approaches to attention-based neural machine translation.

# Attention mechanisms in seq2seq: the algorithm

**Encoder**

- $h_0 =$ zeros(hidden_dim)
- for $t = 1, \ldots, T$
  - $x_t =$ embed$(w_t)$
  - $h_t =$ RNNCell1$(h_{t-1}, x_t)$

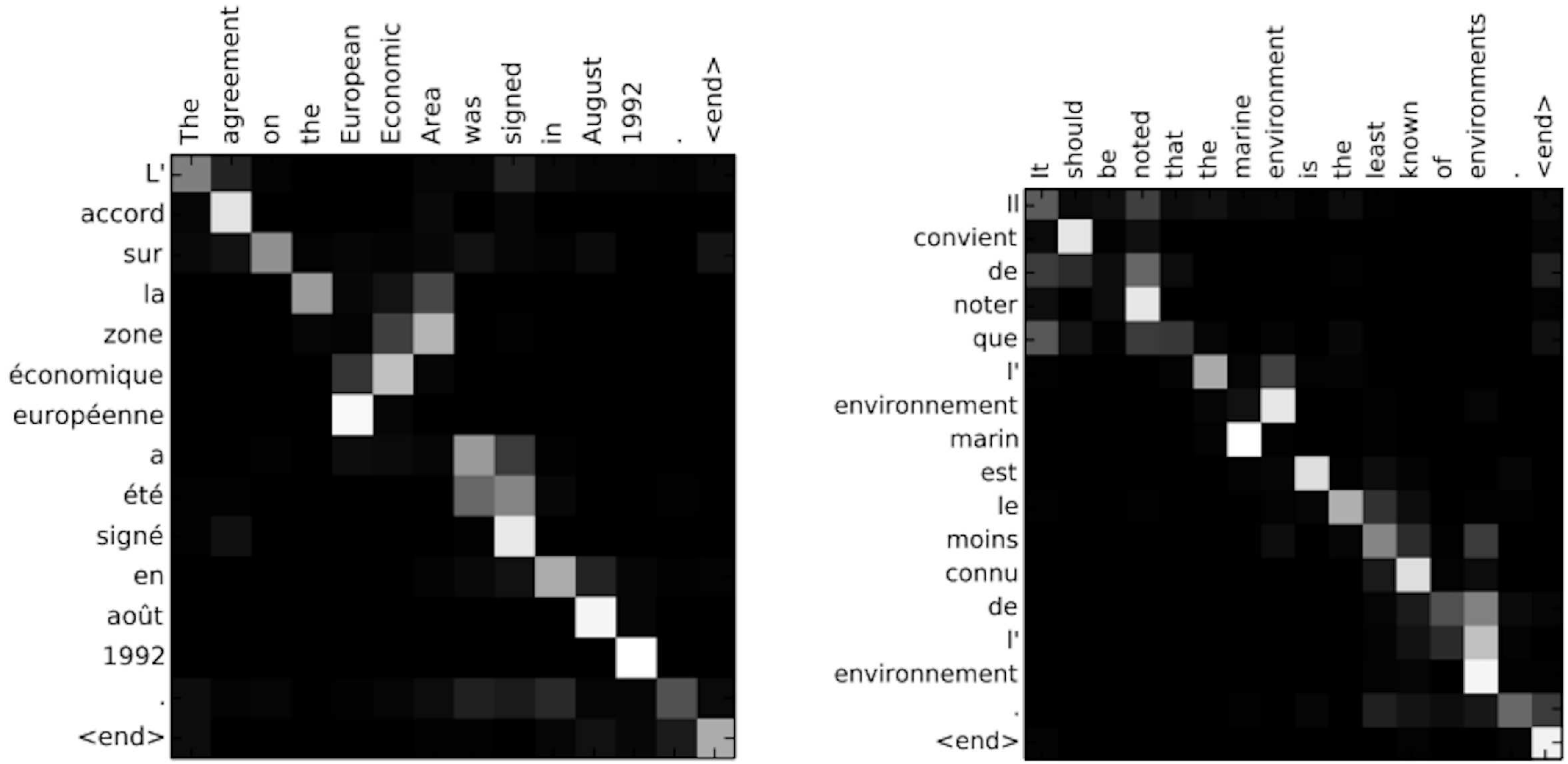
**Decoder**

- $h'_0 = h_T; w'_1 = " < s > "$ $\leftarrow$ start of sentence
- for $t = 1, \ldots, T'$
  - $x'_t =$ embed$(w'_t)$
  - $h'_t =$ RNNCell2$(h'_{t-1}, x'_t)$
  - $a_{ts} \propto h_t'^{t} W_a h_s$ $\leftarrow$ **how input s matters for output t**
  - $c_t = \sum_s a_{ts} h_s$ $\leftarrow$ "context" representation at t
  - $\tilde{h}_t =$ tanh$(W_c[c_t; h_t])$ $\leftarrow$ "modified" decoder state
  - $y_t =$ softmax$(A\tilde{h}_t + b)$ $\leftarrow$ predict from $\tilde{h}_t$
  - $w'_{t+1} =$ choose$(y_t)$

ENSAI École nationale de la statistique et de l'analyse de l'information

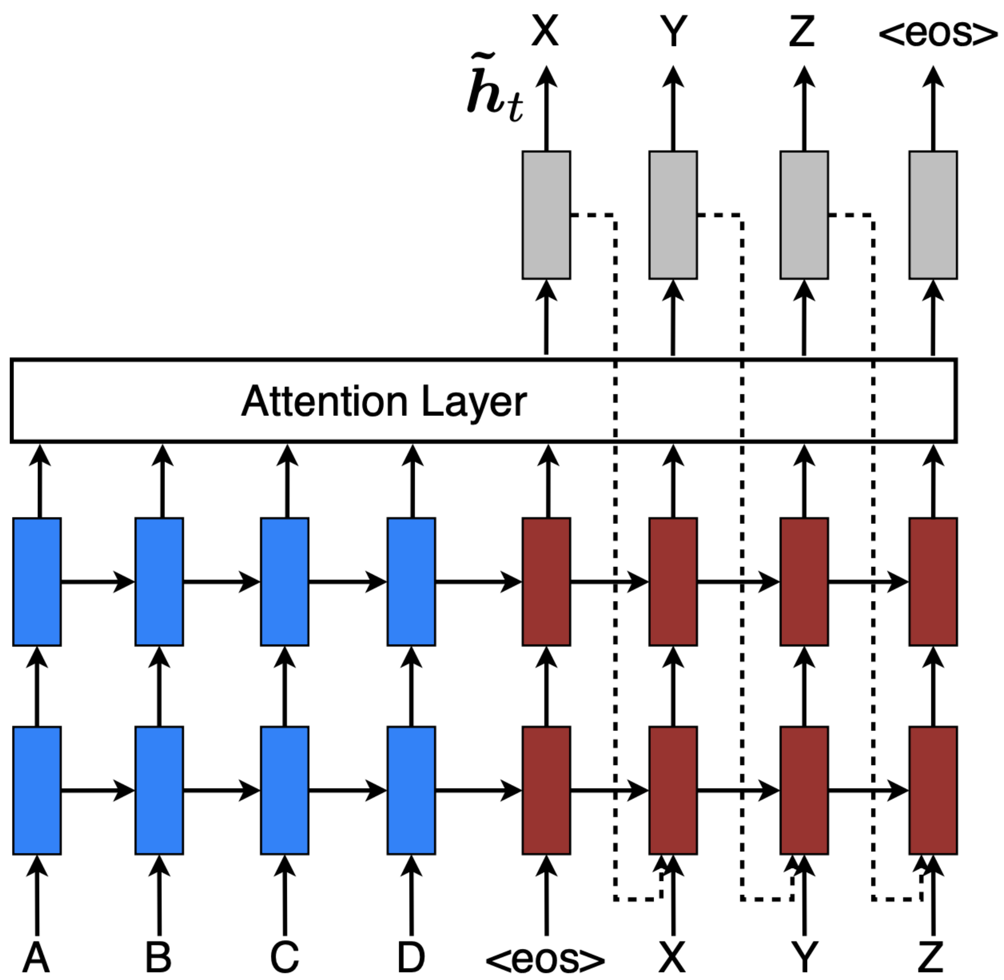
# Attention maps in seq2seq

The **attention matrix** $A$ gathers for all output tokens (rows) the attention w.r.t. to the input tokens (columns) in an English to French translation task here.



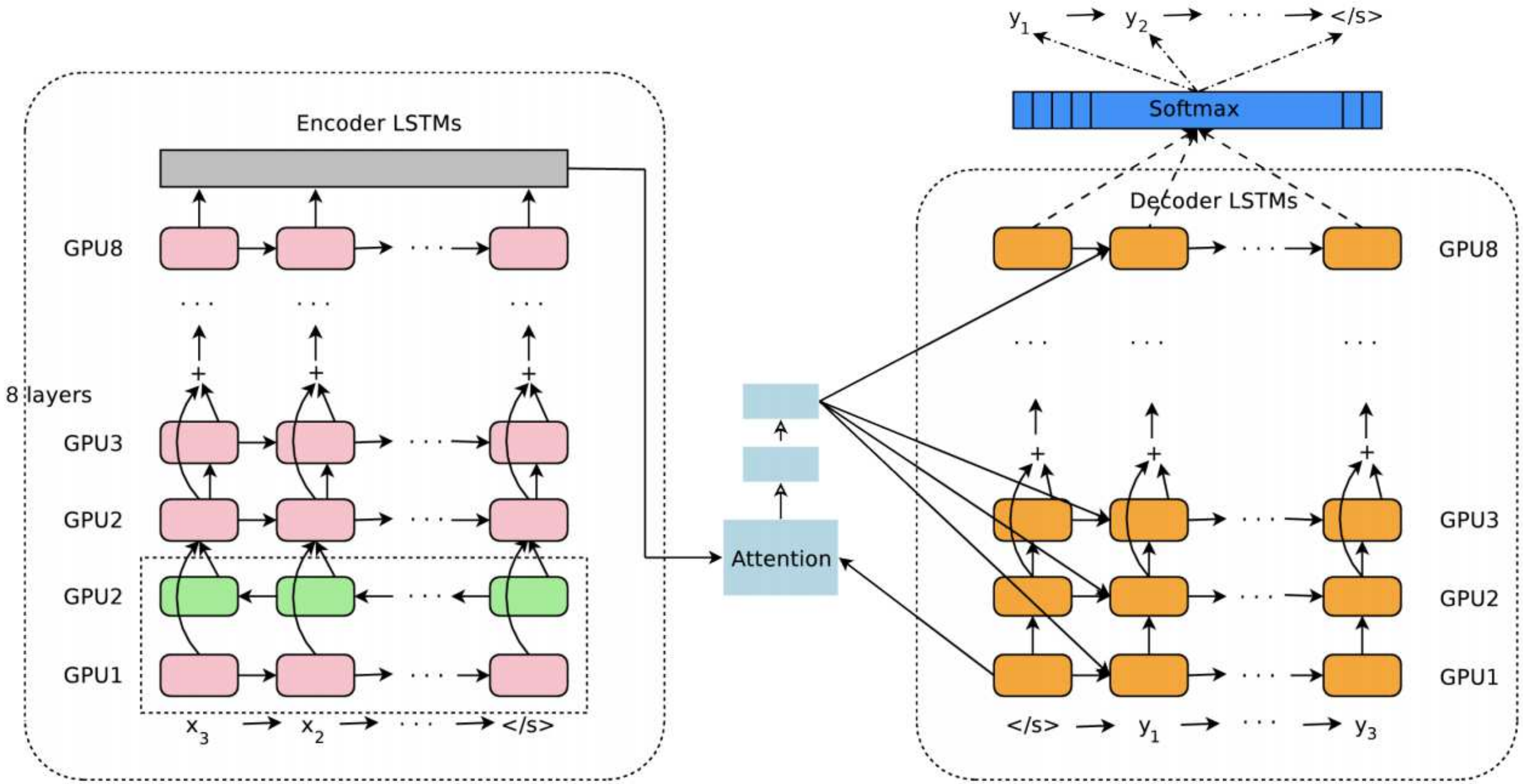

Dzmitry Bahdanau et al. 2015. Neural machine translation by jointly learning to align and translate.

# Complexifying things with layers



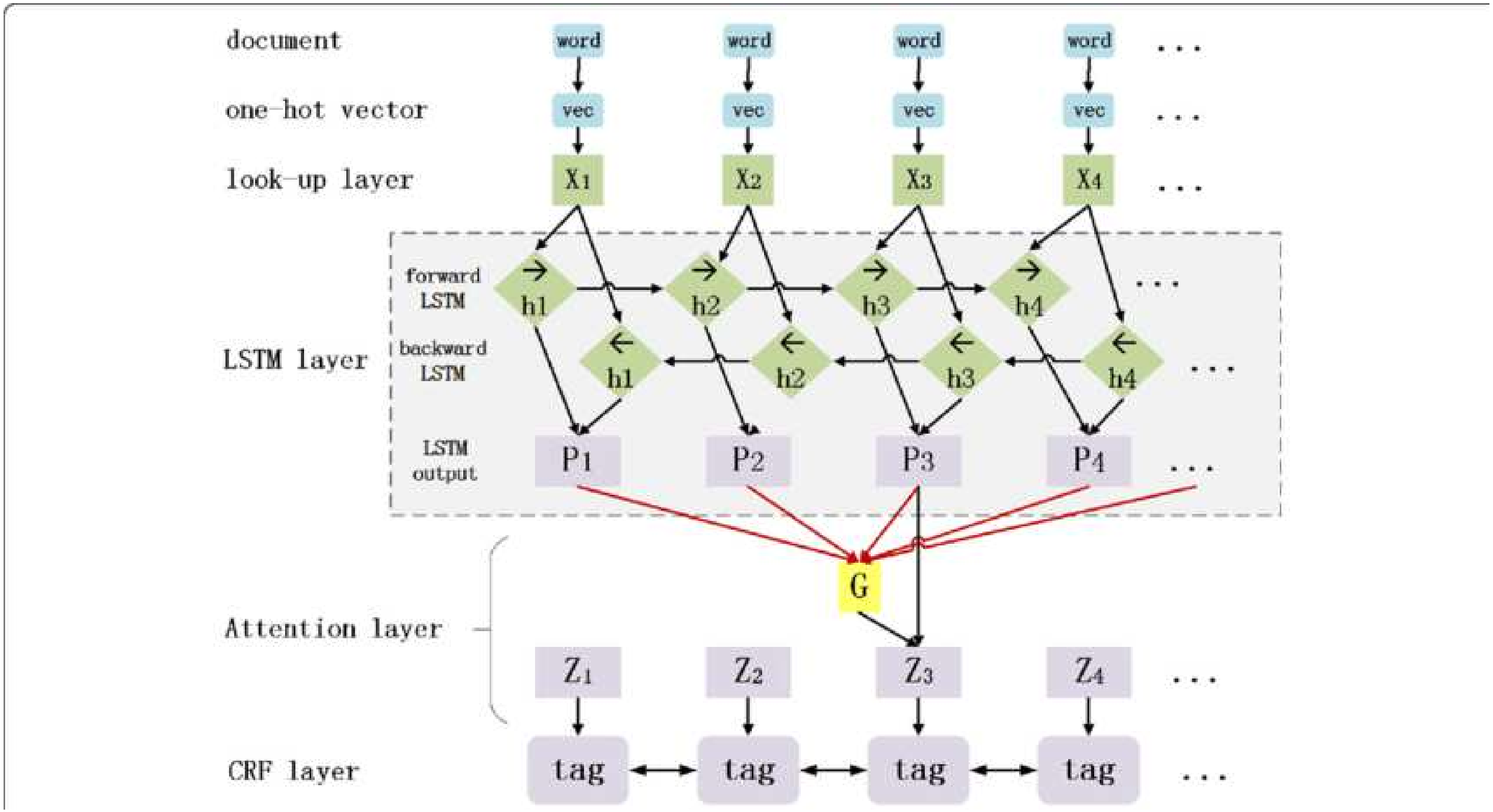

Luong et al. 2015. Effective approaches to attention-based neural machine translation.

# Google neural machine translation: a complex seq2seq




Yonghui Wu et al. 2016 Google's neural machine translation system: Bridging the gap between human and machine translation

# Attention mechanisms are not limited to seq2seq




Bin Ji et al. 2019. A hybrid approach for named entity recognition in Chinese electronic medical record

# Abstracting the attention mechanism

$$a_{ts} = \text{softmax}(h_t^t W_a \overline{h}_s) \quad \text{and} \quad c_t = \sum_{s=1}^{n} a_{ts} \overline{h}_s$$

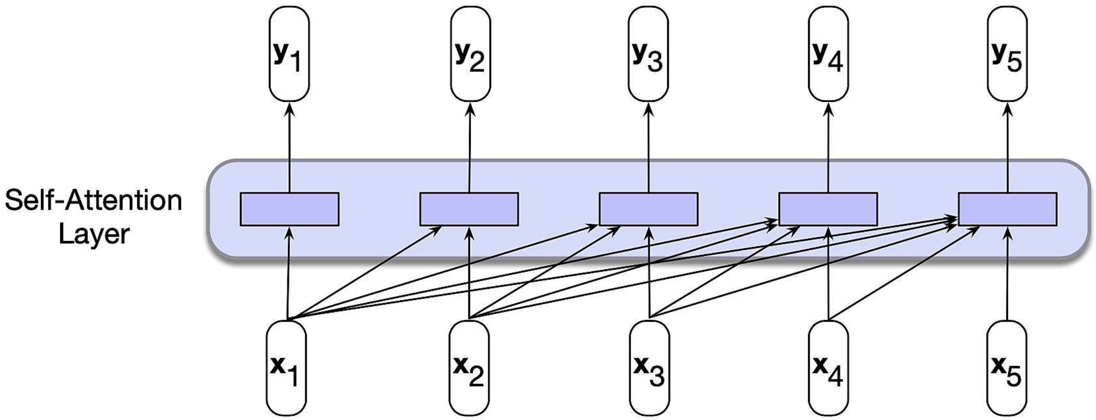
Can be abstracted through three key elements

- **Query**: a representation of the element that you are considering as your focus point (the point you attend from)
  $\rightarrow$ *the decoder state* $h_t$ *at time* $t$
- **Keys**: a representation of the elements you want to consider to change your query element (the points you attend to)
  $\rightarrow$ *the encoder state variables* $\overline{h}_i$
- **Values**: a representation of the elements you attended to to compute the average and get a new representation related to the query
  $\rightarrow$ *keys and values are alike here in Luong's attention*

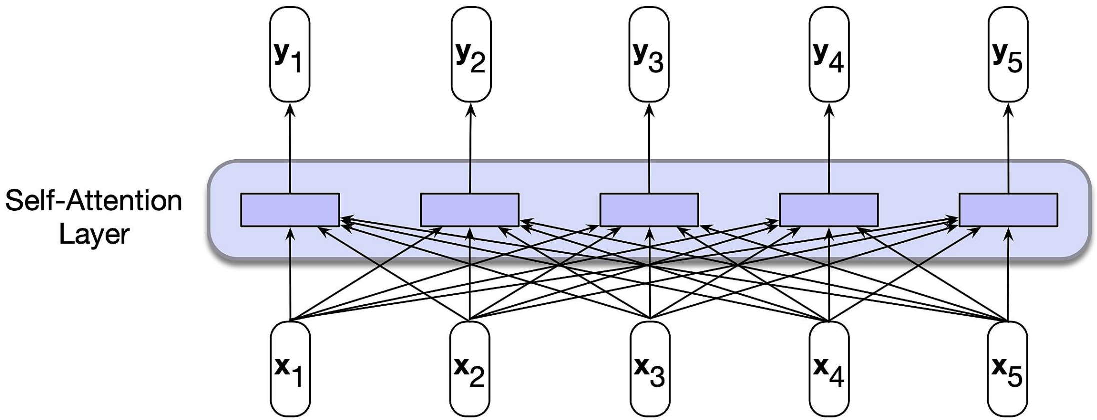
$$a_t = q_t K^t = \text{sofmax}\left( \begin{pmatrix} q_{t1} & \dots & q_{td} \end{pmatrix} \begin{pmatrix} k_{11} & \dots & k_{n1} \\ \vdots & \ddots & \vdots \\ k_{1d} & \dots & k_{nd} \end{pmatrix} \right)$$

# The self-attention principle

The key idea is to *transform* an entry vector (word embedding) based on its relations/attentions with respect to other entry vectors, either in a causal manner (language generation) – $y_i = f(x_i, x_{i-1}, \ldots, x_1)$




or in a bidirectional manner – $y_i = f(x_n, \ldots, x_{+1}, x_i, x_{i-1}, \ldots, x_1)$




borrowed from Jurafsky and Martin's book

# Self-attention basic layer (the maths)

Given an input sequence $x = x_1, \ldots, x_n$, with $x_i \in \mathbb{R}^d$

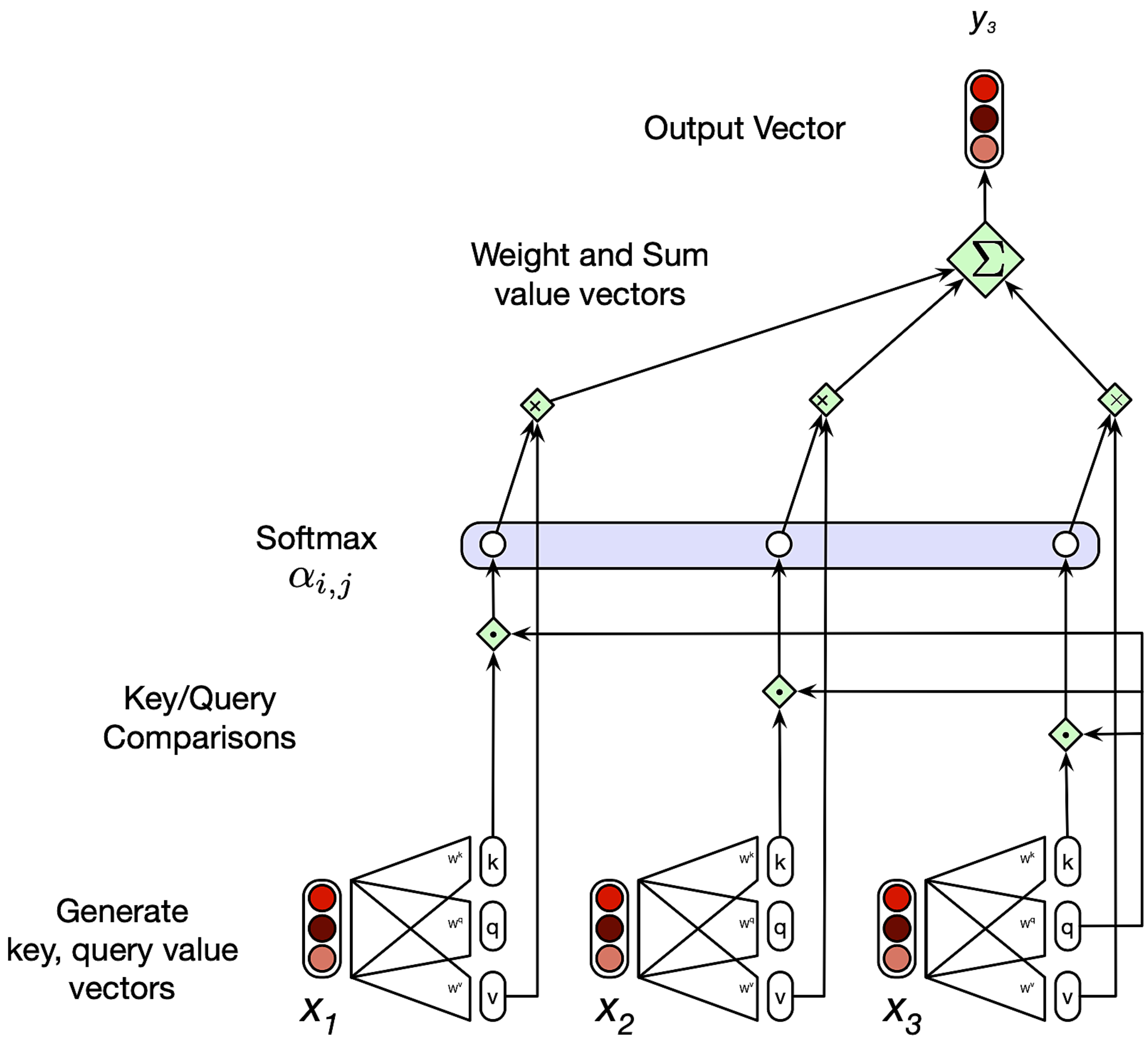
1. project each $x_i$ to its corresponding query $q_i$, key $k_i$ and value $v_i$ (all in $\mathbb{R}^d$)
   - $q_i = \mathbf{W}_q x_i$: query to measure how $x_i$ relates to $x_j$ for all $j$
   - $k_i = \mathbf{W}_k x_i$: key to measure how $x_j$ is related to $x_i$
   - $v_i = \mathbf{W}_v x_i$: value to compute new output from $x_i$
2. compute attention distribution at all positions $i$

$$a_{ij} = \frac{\exp(q_i \cdot k_j / \sqrt{d})}{\sum_{l=1}^{n} \exp(q_i \cdot k_l / \sqrt{d})}$$

3. compute output values

$$y_i = \sum_{j=1}^{n} a_{ij} v_j$$

# Self-attention basic layer illustrated




borrowed again from Jurafsky and Martin's book

# Self-attention basic layer (the matrix maths)

All these operations can be efficiently performed with matrices, starting from $\mathbf{X} \in \mathbb{R}^{n \times d}$ gathering all input embeddings $x_i$:

1. compute the queries, keys, values for all tokens

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q \quad \mathbf{K} = \mathbf{X}\mathbf{W}_k \quad \mathbf{V} = \mathbf{X}\mathbf{W}_v$$

2. compute the attention matrix

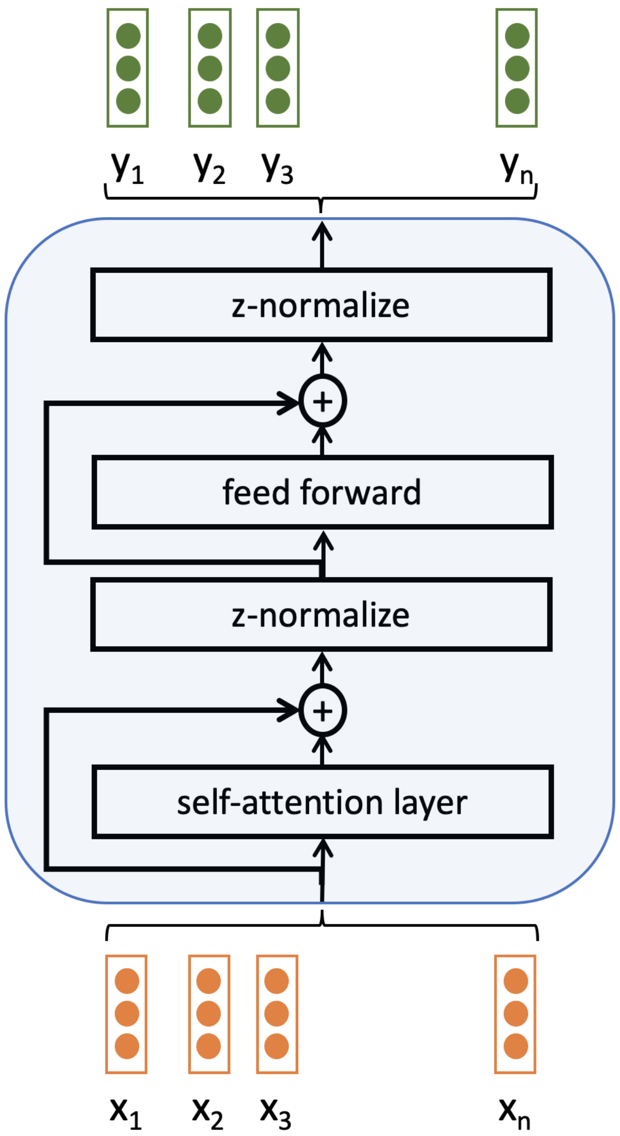
$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^t}{\sqrt{d}}\right)$$

3. compute new representations

$$\mathbf{Y} = \mathbf{A}\mathbf{V}$$

For causal models, simply force the upper triangular part of $\mathbf{A}$ to be zero

# From self-attention to transformer block




Add a few things on top of the self-attention layer:

- residual connections

$$\mathbf{X}' = \mathbf{X} + \text{SelfAttention}(\mathbf{X})$$

- layer normalization

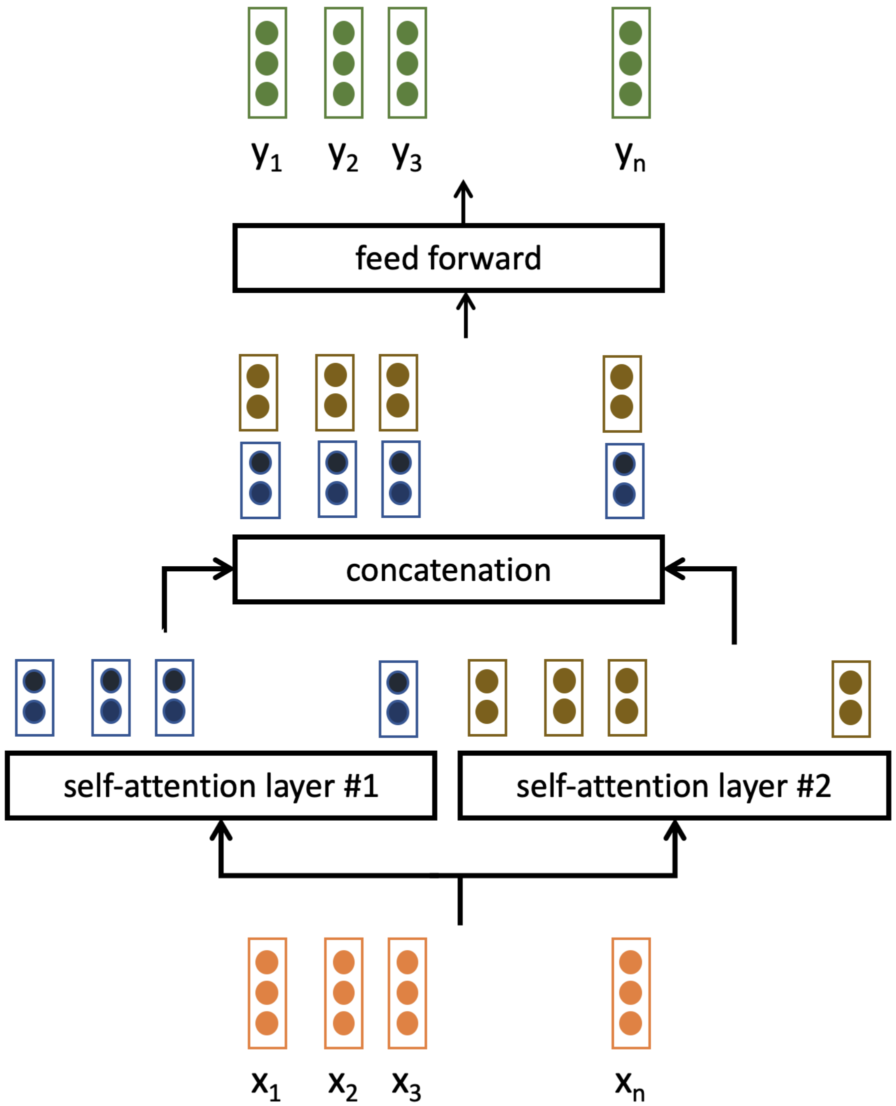
$$\mathbf{X}'(i,j) = \gamma \frac{\mathbf{X}(i,j) - \mu_i}{\sigma_i} + \beta$$

- pointwise feed-forward

$$\mathbf{X}'(i,:) = \mathbf{A}_2 \text{ReLU}\left(\mathbf{A}_1 \mathbf{X}(i,:) + b_1\right) + b_2$$

# Always more: multi-head attention

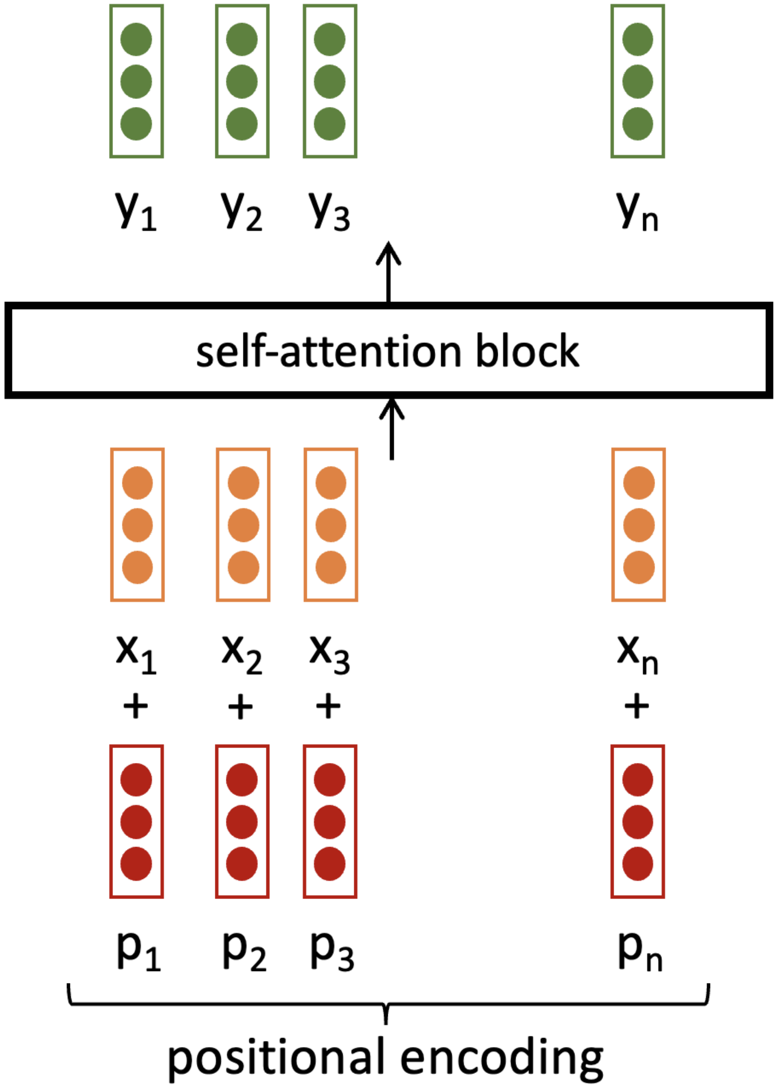
Use multiple self-attention layers (aka *heads*) in parallel, each with its own set of parameters $\mathbf{W}_q^i, \mathbf{W}_k^i, \mathbf{W}_v^i$ and combine the result.

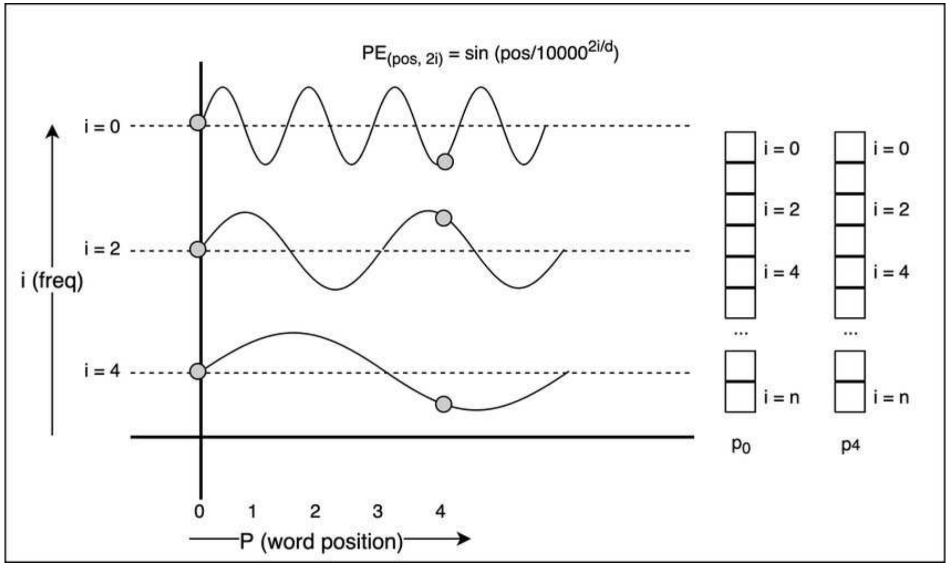



In practice, each self-attention operates on a subset of dimensions of the input space $\mathbb{R}^d$ and the final projection is omitted.

If $d = 64$ and considering $2$ heads, each head operates on $64/2 = 32$ dimensions, the first one being $X(:, [0 : 31])$, the second one $X(:, [32 : 63])$.

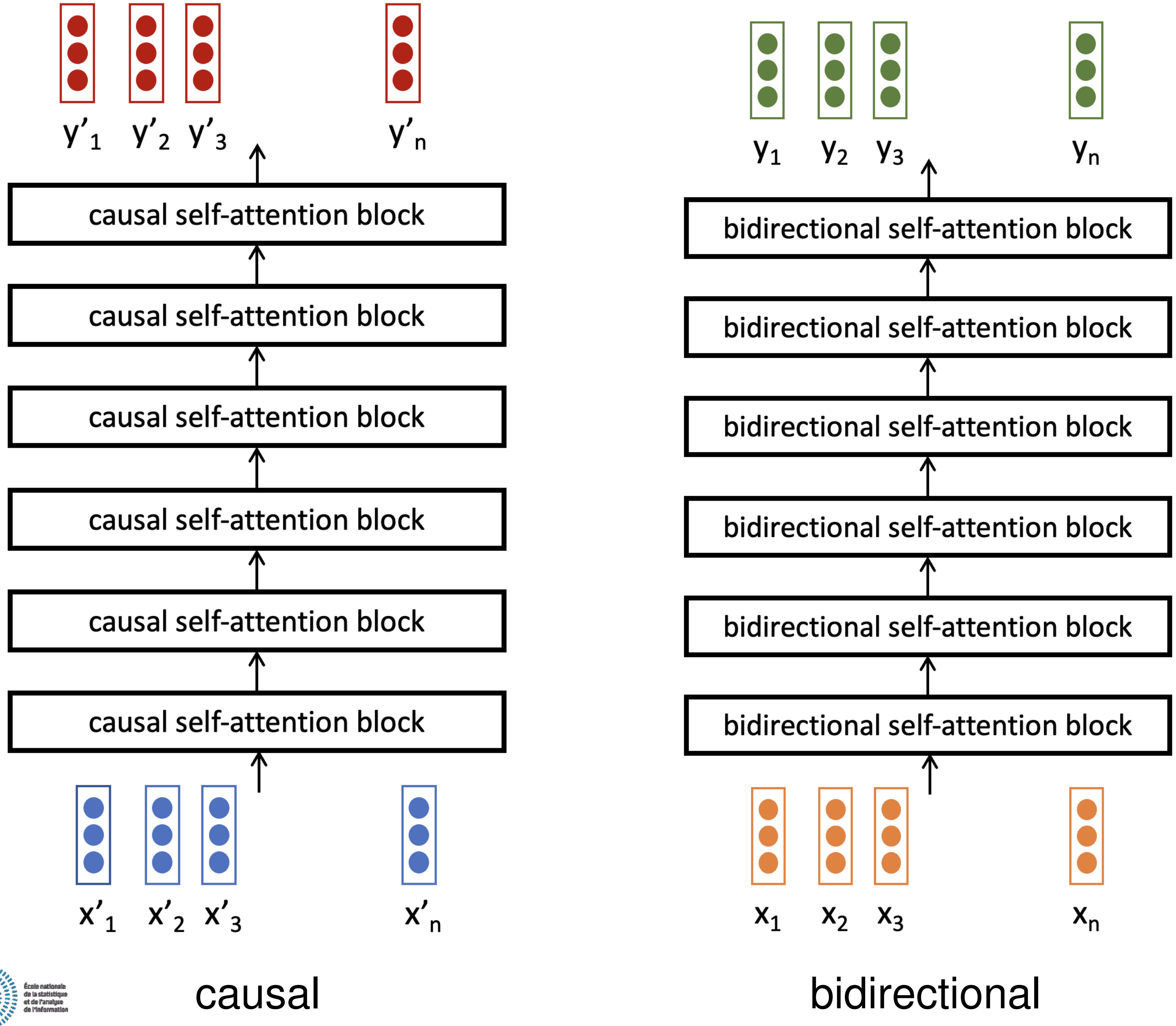
# And the cherry on the cake: Positional encodings



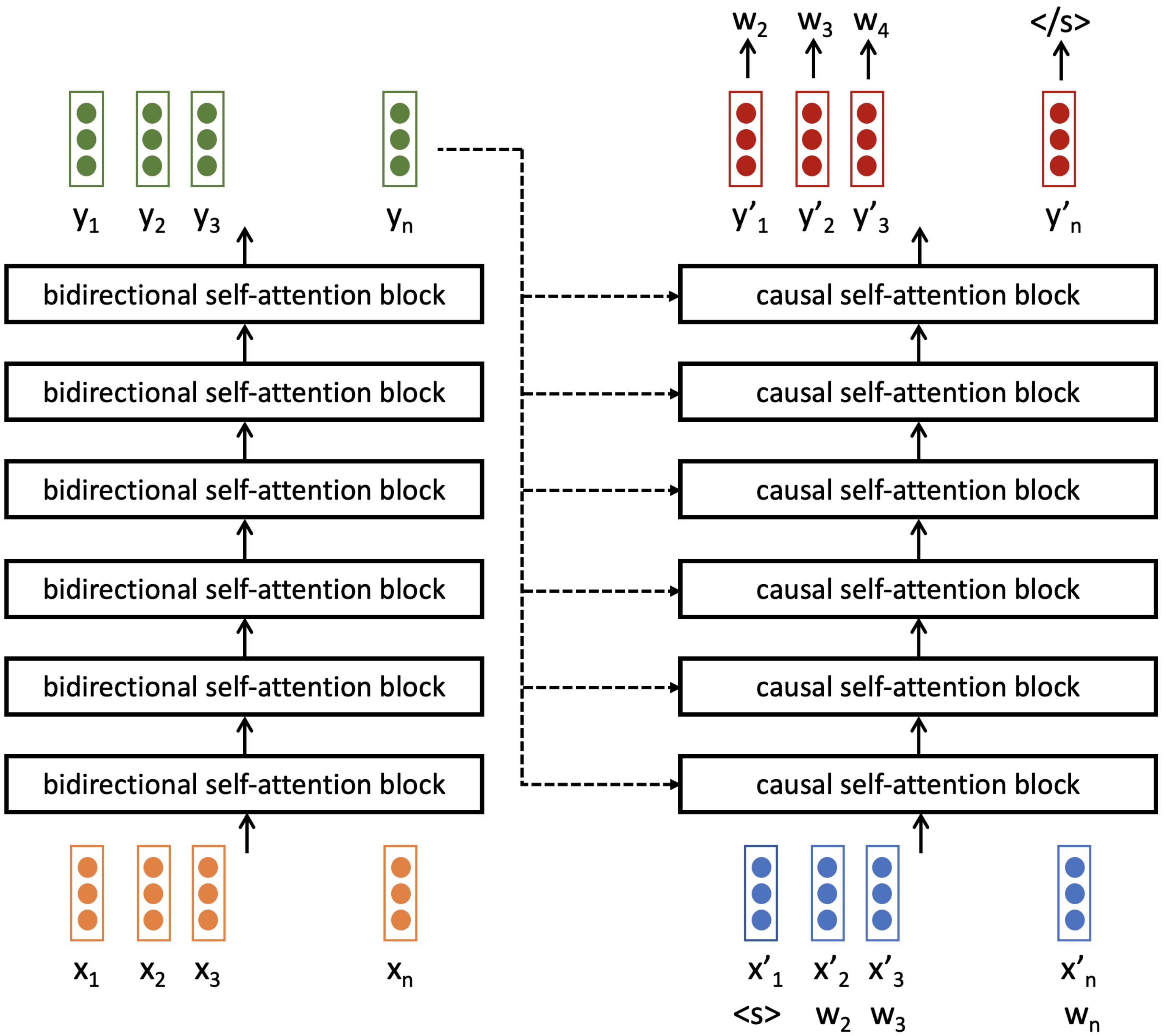




borrowed from Haque & Ghani (2022). The Storyteller: Computer Vision Driven Context and Content Generation System.

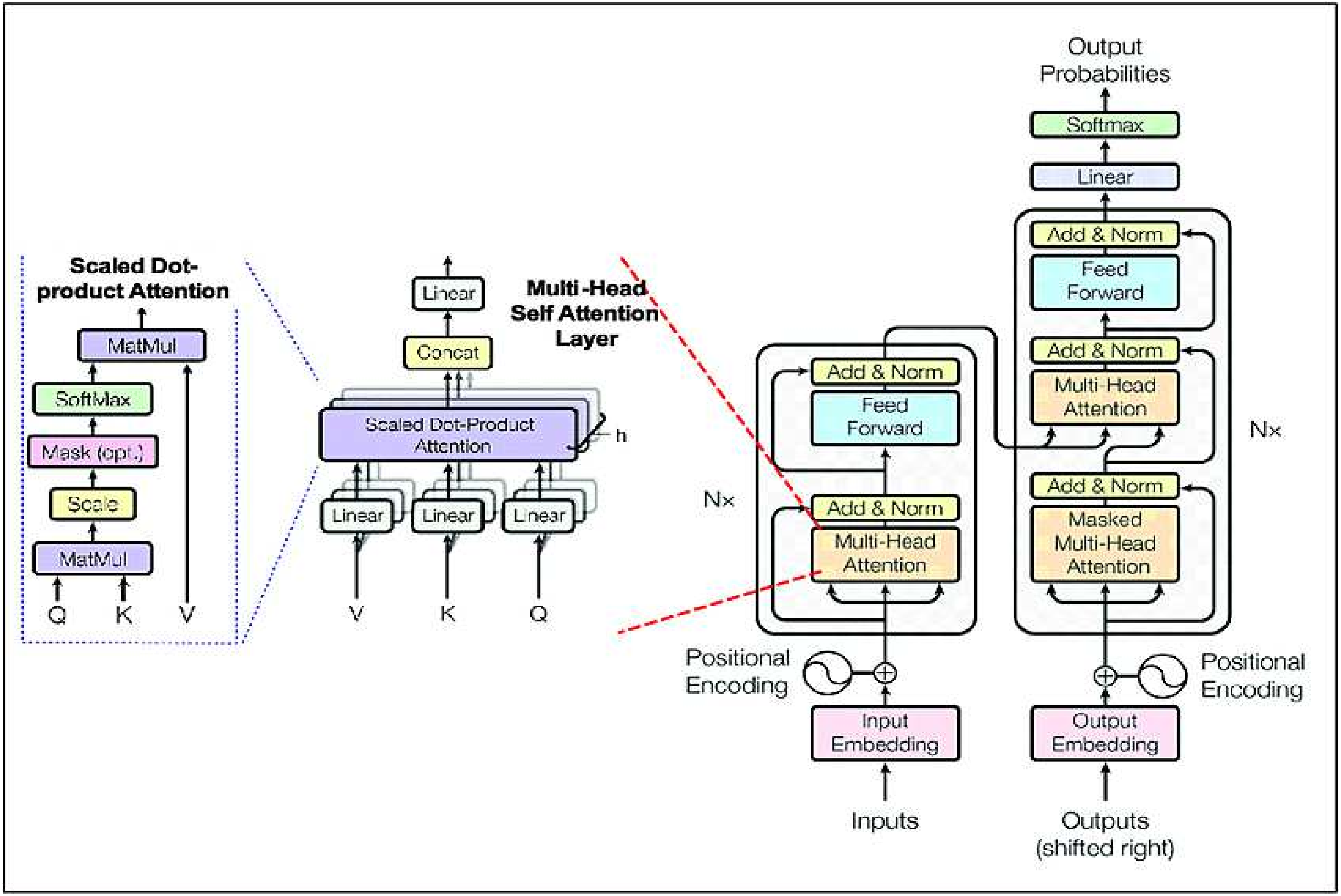
# Causal and bidirectional transformers...



# Putting it all together in a *transformer*



# Putting it all together in a *transformer* (zoom)




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


That's all Folks!