E9: 309 Advanced Deep Learning
21-10-2020

Instructor: Sriram Ganapathy
sriramg@iisc.ac.in

Teaching Assistant : Akshara Soman, Prachi Singh, Jaswanth Reddy
aksharas@iisc.ac.in, prachis@iisc.ac.in, jaswanthk@iisc.ac.in

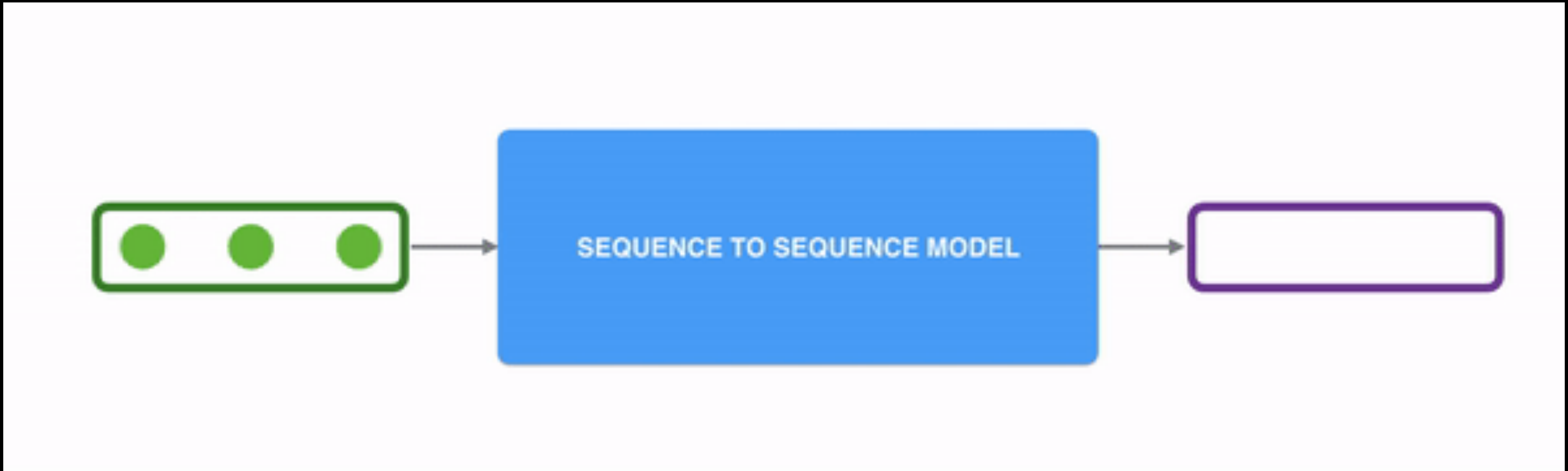
http://leap.ee.iisc.ac.in/sriram/teaching/ADL2020/

Recap of previous class

State of affairs

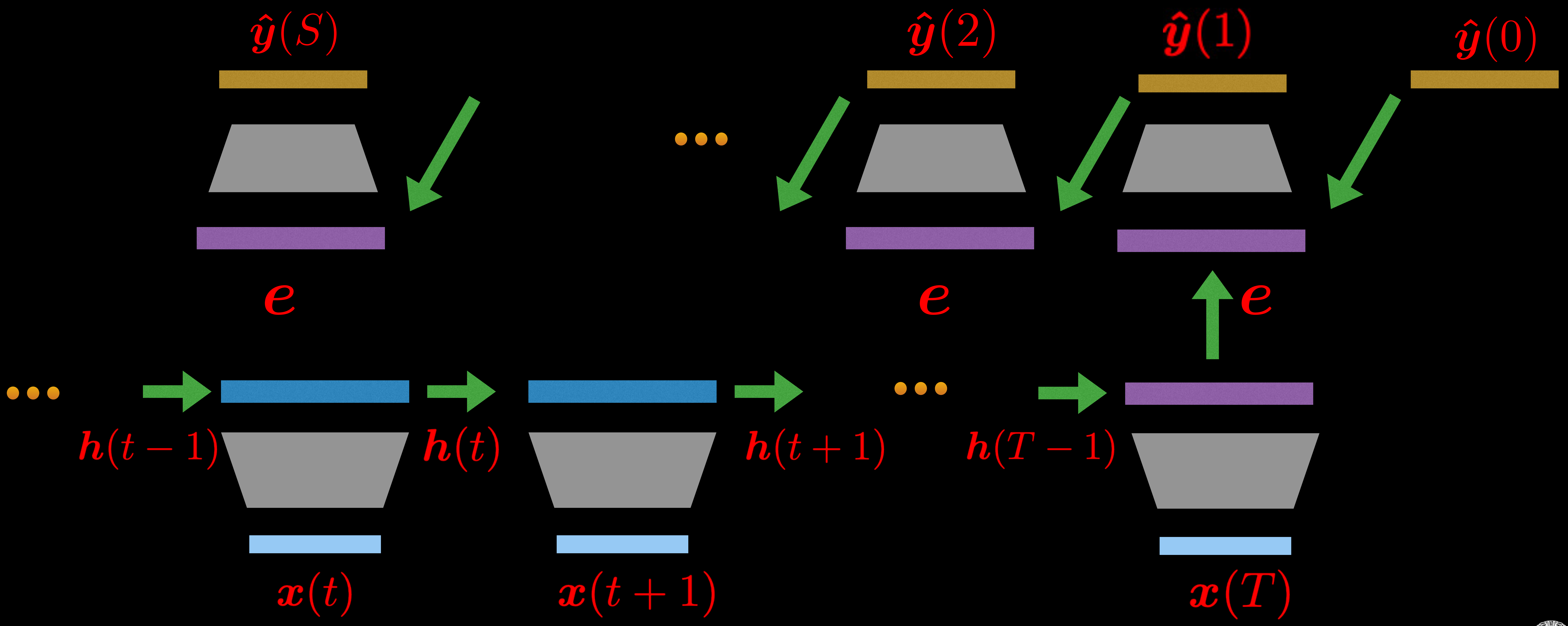
* LSTM models
* Bidirectional RNNs
* Seq2vec and vec2Seq models
* Encoder-decoder models.

Encoder-decoder models



Encoder-decoder models

* Multiple input multiple output (with different label index) - Seq2seq



Encoder-decoder models

* Encoder — convert sequence $\mathbf{x} = \{\mathbf{x}(1), .., \mathbf{x}(T)\}$ to vector

$$\mathbf{h}(t) = f(\mathbf{h}(t-1), \mathbf{x}(t))$$
$$\mathbf{e} = f'(\mathbf{h}_1, ..., \mathbf{h}_T)$$

* The encoder can have multiple deep RNN layers.

* For simplicity

$$\mathbf{e} = \mathbf{h}_T$$

Encoder-decoder models

* Encoder — convert sequence $\mathbf{x} = \{\mathbf{x}(1), .., \mathbf{x}(T)\}$ to vector $\mathbf{e}$

* Decoder — converts the vector embedding from the encoder to the output sequence $\hat{\mathbf{y}} = \{\hat{\mathbf{y}}(1), ..., \hat{\mathbf{y}}(S)\}$ with different label index.

$$p(\hat{\mathbf{y}}) = \prod_{s=1}^{S} p(\hat{\mathbf{y}}(s)|\hat{\mathbf{y}}(1), ..., \hat{\mathbf{y}}(s-1))$$

* RNN decoder assumption

$$p(\hat{\mathbf{y}}(s)|\hat{\mathbf{y}}(1), ..., \hat{\mathbf{y}}(s-1)) = p(\hat{\mathbf{y}}(s)|\hat{\mathbf{y}}(s-1), \mathbf{e}) = softmax(\mathbf{V}\hat{\mathbf{y}}(s-1) + \mathbf{R}\mathbf{c}(s-1) + \mathbf{T}\mathbf{e} + \mathbf{d})$$

$$\mathbf{c}(s) = f(\mathbf{c}(s-1), \mathbf{e})$$

* The decoder can also have multiple layers of deep RNNs before softmax.

Encoder-decoder models

- Encoder — convert sequences to vectors
- Decoder — converts the vector embedding from the encoder to the output sequence with different label index.
  - ✓ Start and end label are also encoded as output vector indices.
    - ⋆ Enable the starting and ending of the output sequence.
- Assumption
  - ✓ The entire input sequence can be represented as a single vector $\mathbf{e}$
    - ⋆ May not be able to perform this efficiently for long sequences.

Encoder-decoder models

* Modification of encoder-decoder model

$$p(\hat{\mathbf{y}}(s)|\hat{\mathbf{y}}(1), ..., \hat{\mathbf{y}}(s-1)) = p(\hat{\mathbf{y}}(s)|\hat{\mathbf{y}}(s-1), \mathbf{e}) = softmax(\mathbf{V}\hat{\mathbf{y}}(s-1) + \mathbf{R}\mathbf{c}(s-1) + \mathbf{T}\mathbf{e} + \mathbf{d})$$

$$p(\hat{\mathbf{y}}(s)|\hat{\mathbf{y}}(1), ..., \hat{\mathbf{y}}(s-1)) = p(\hat{\mathbf{y}}(s)|\hat{\mathbf{y}}(s-1), \mathbf{e}(s)) = softmax(\mathbf{V}\hat{\mathbf{y}}(s-1) + \mathbf{R}\mathbf{c}(s-1) + \mathbf{T}\mathbf{e}(s) + \mathbf{d})$$

* where

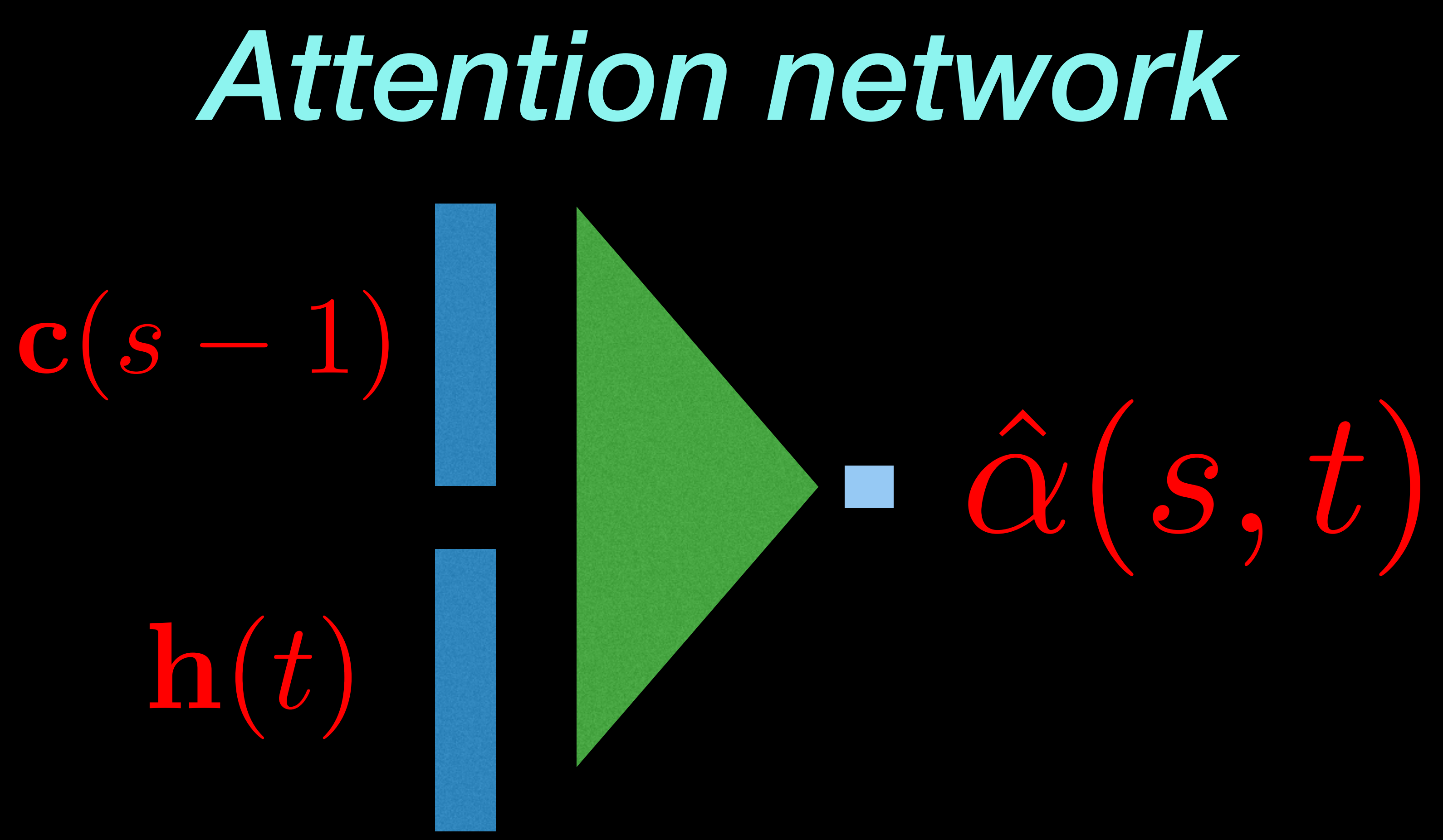
$$\mathbf{e}(s) = \sum_{t=1}^{T} \alpha(s,t)\mathbf{h}(t)$$

* Here $\alpha(s,t)$ captures the contribution of input at time t with output at time s

Encoder-decoder models

* Obtaining the relative contribution $\alpha(s,t)$

  ✓ Implementing this automatically using network-in-network



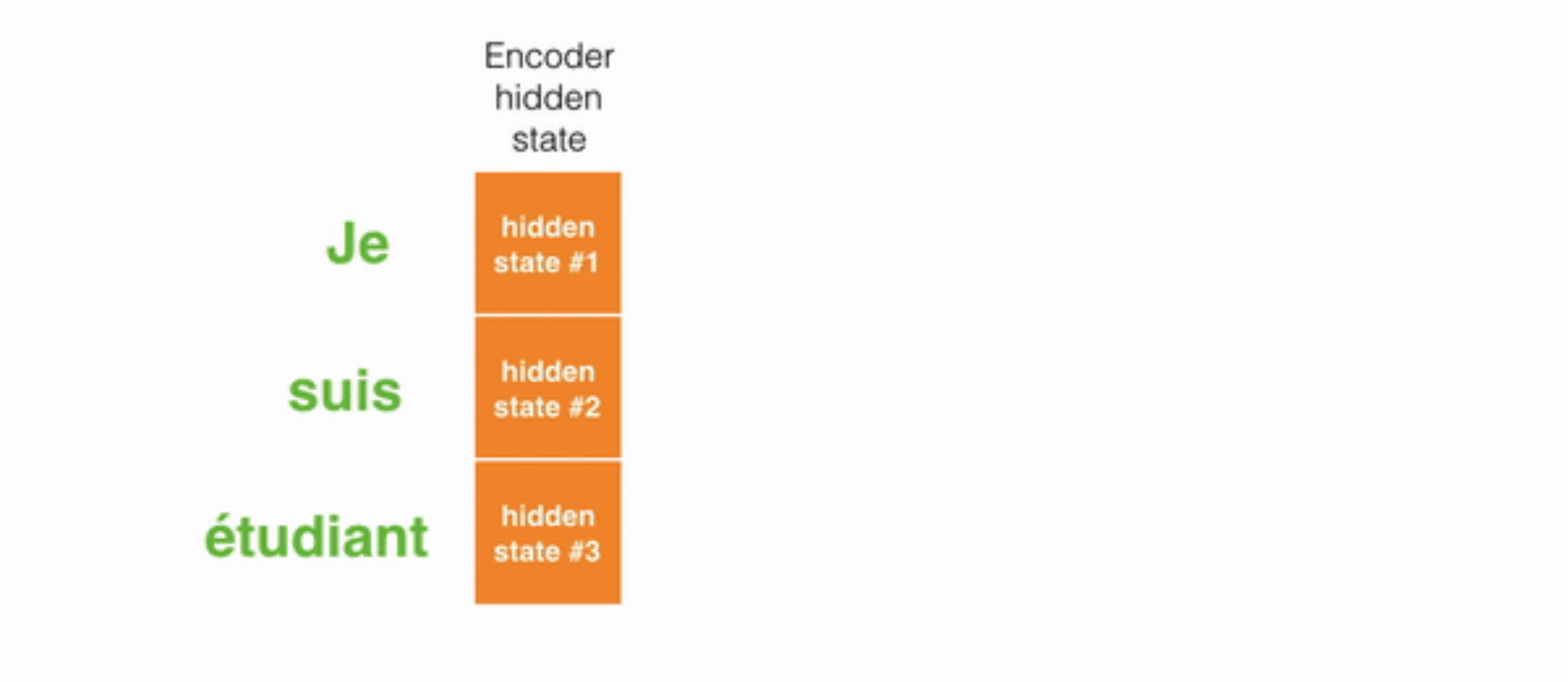
$$\hat{\alpha}(s,t) = \mathbf{A}[\mathbf{c}(s-1); \mathbf{h}(t)]$$

$$\alpha(s,t) = S(\hat{\alpha}(s,t)) = \frac{exp(\hat{\alpha}(s,t))}{\sum_{t'} exp(\hat{\alpha}(s,t'))}$$

  ✓ The values $\alpha(s,t)$ are called attention weights.

Visualizing attention

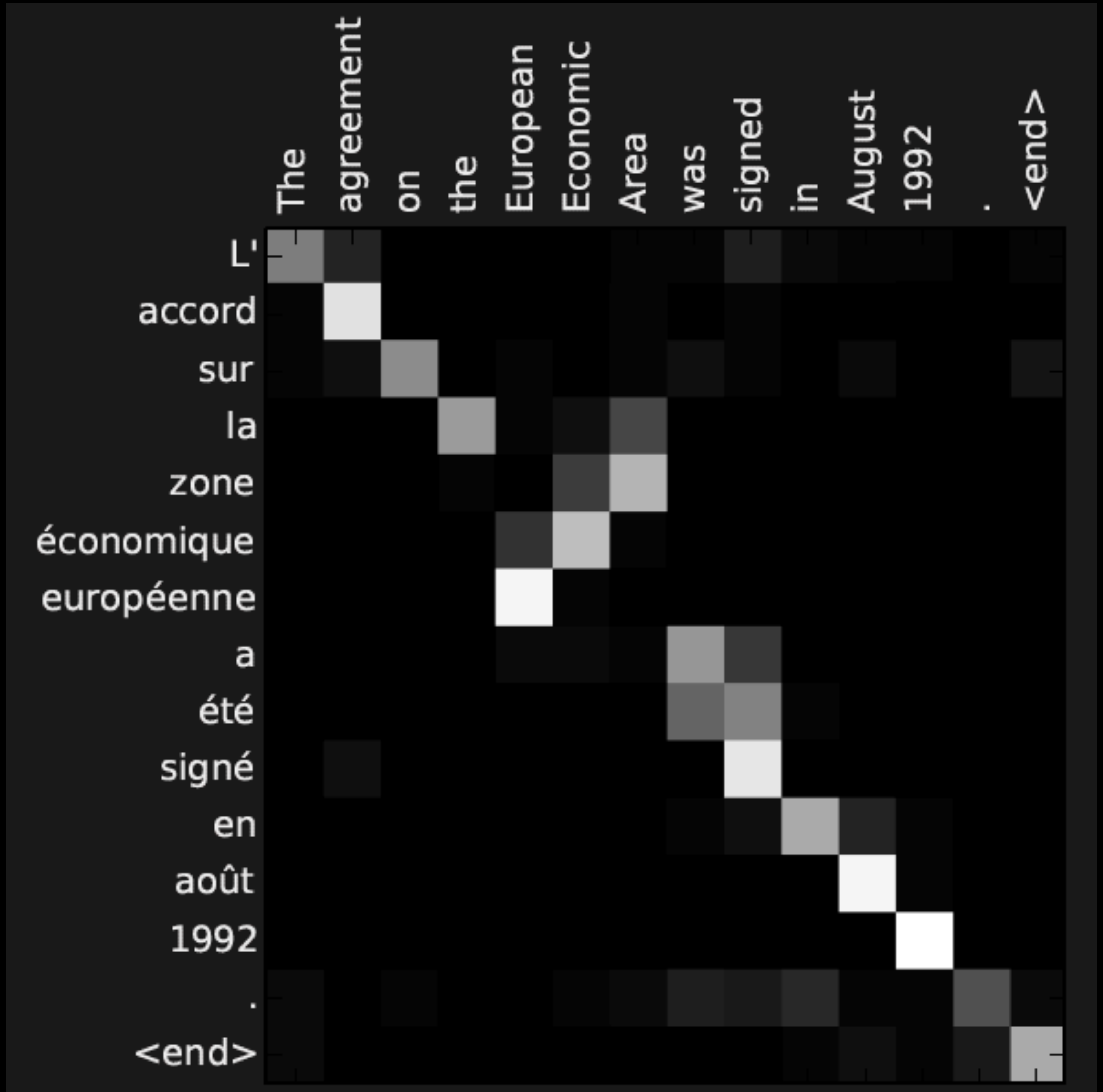
Visualizing attention



Analysis of attention networks

- Attention weights $\alpha(s, t)$
  - ✓ Probability of linking (attending) to input at t for generating output at s
  - ✓ Useful in analyzing the internal structure of the encoder-decoder model

Visualizing the attention weights



Reading Assignment - "Neural Machine Translation by Jointly Learning to Align and Translate"
https://arxiv.org/pdf/1409.0473.pdf

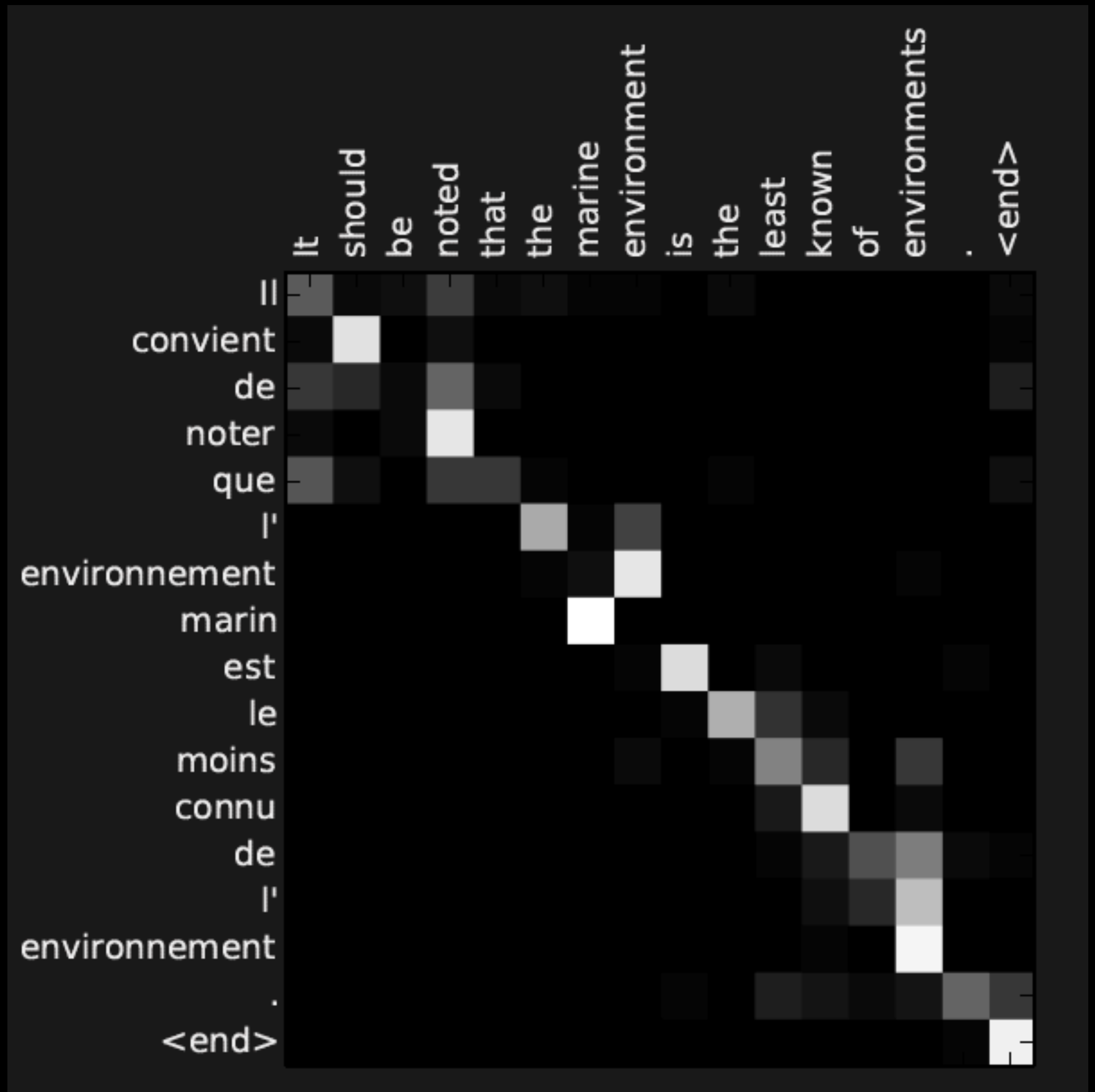
Analysis of attention networks

* Attention weights $\alpha(s, t)$

  ✓ Probability of linking (attending) to input at t for generating output at s

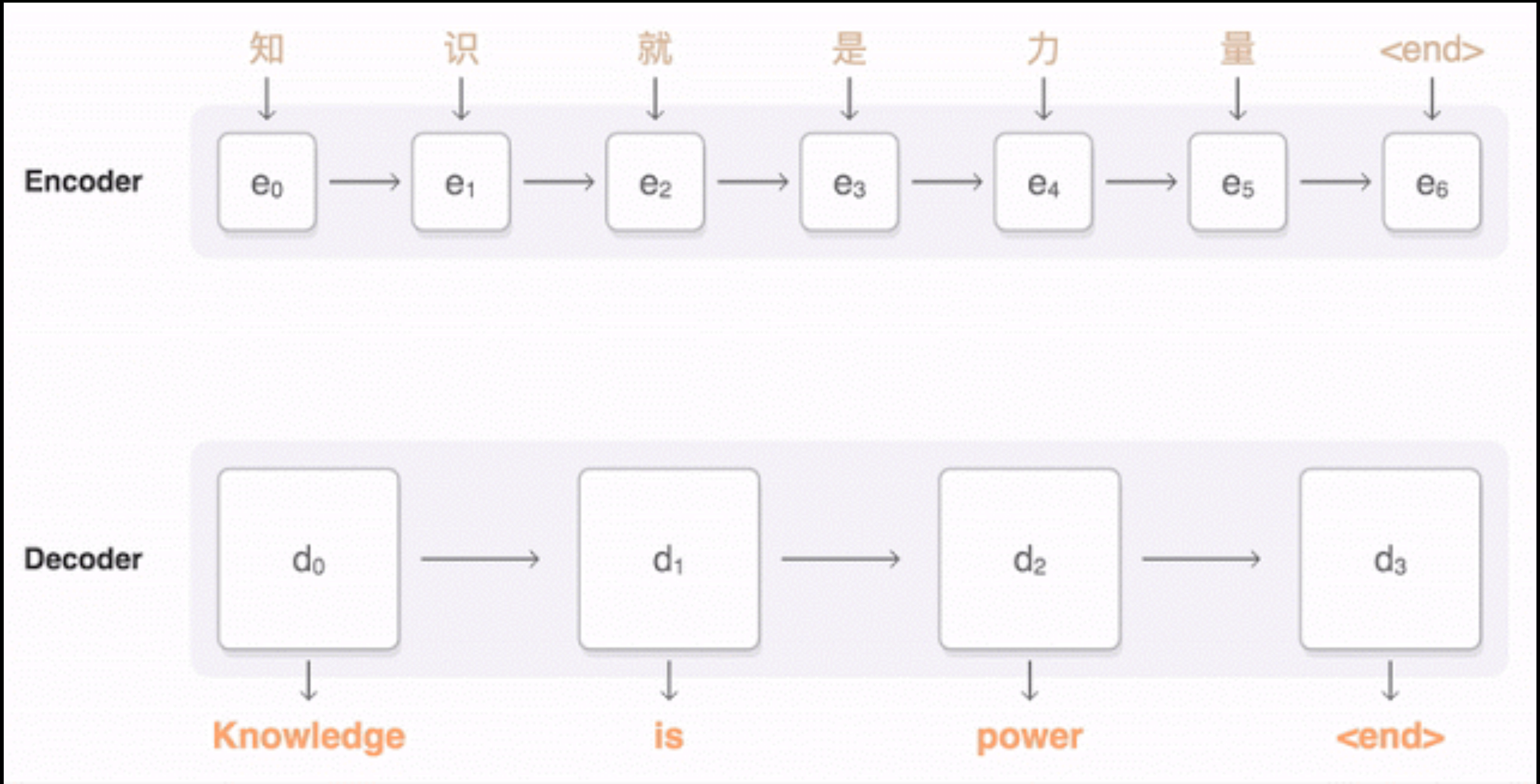
  ✓ Useful in analyzing the internal structure of the encoder-decoder model

Visualizing the attention weights



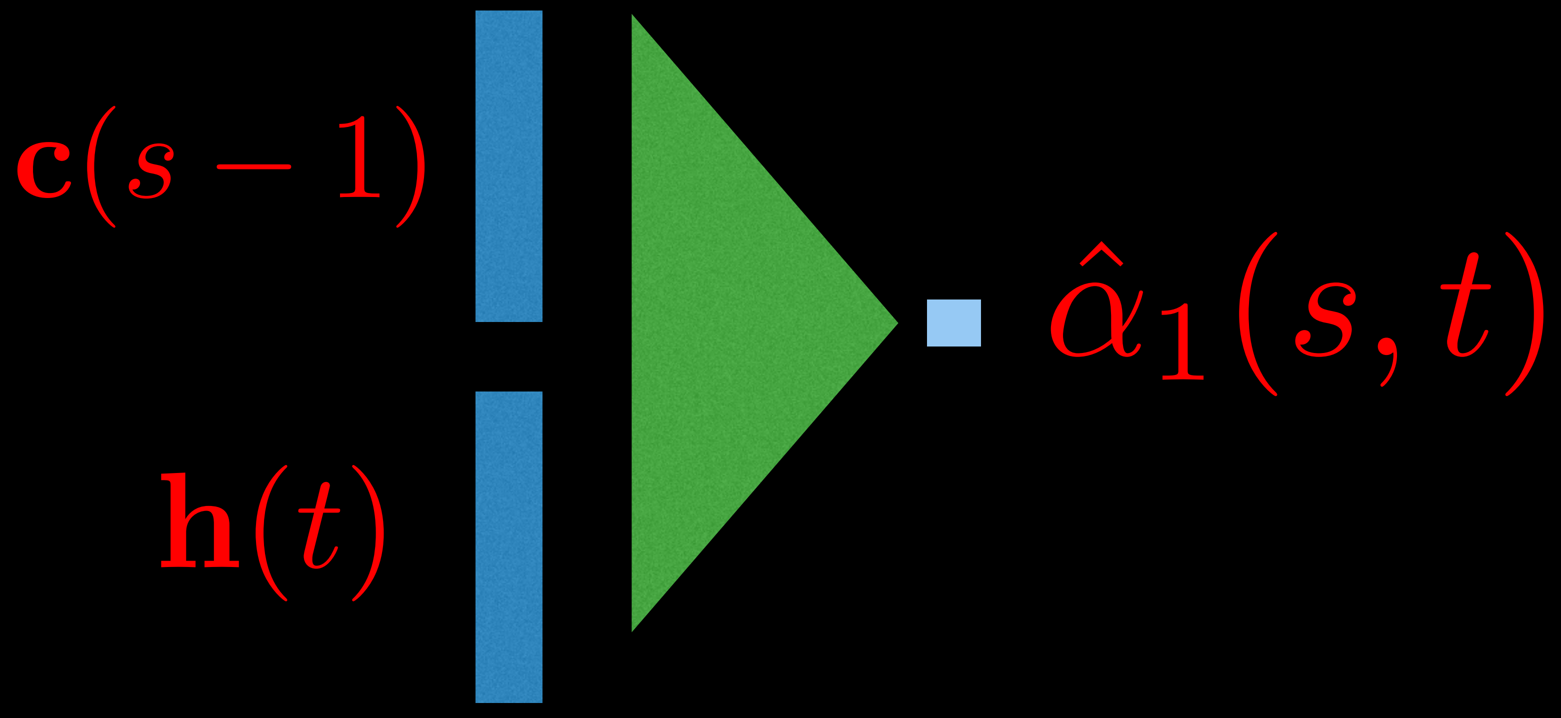
Reading Assignment - "Neural Machine Translation by Jointly Learning to Align and Translate"
https://arxiv.org/pdf/1409.0473.pdf

Visualizing attention

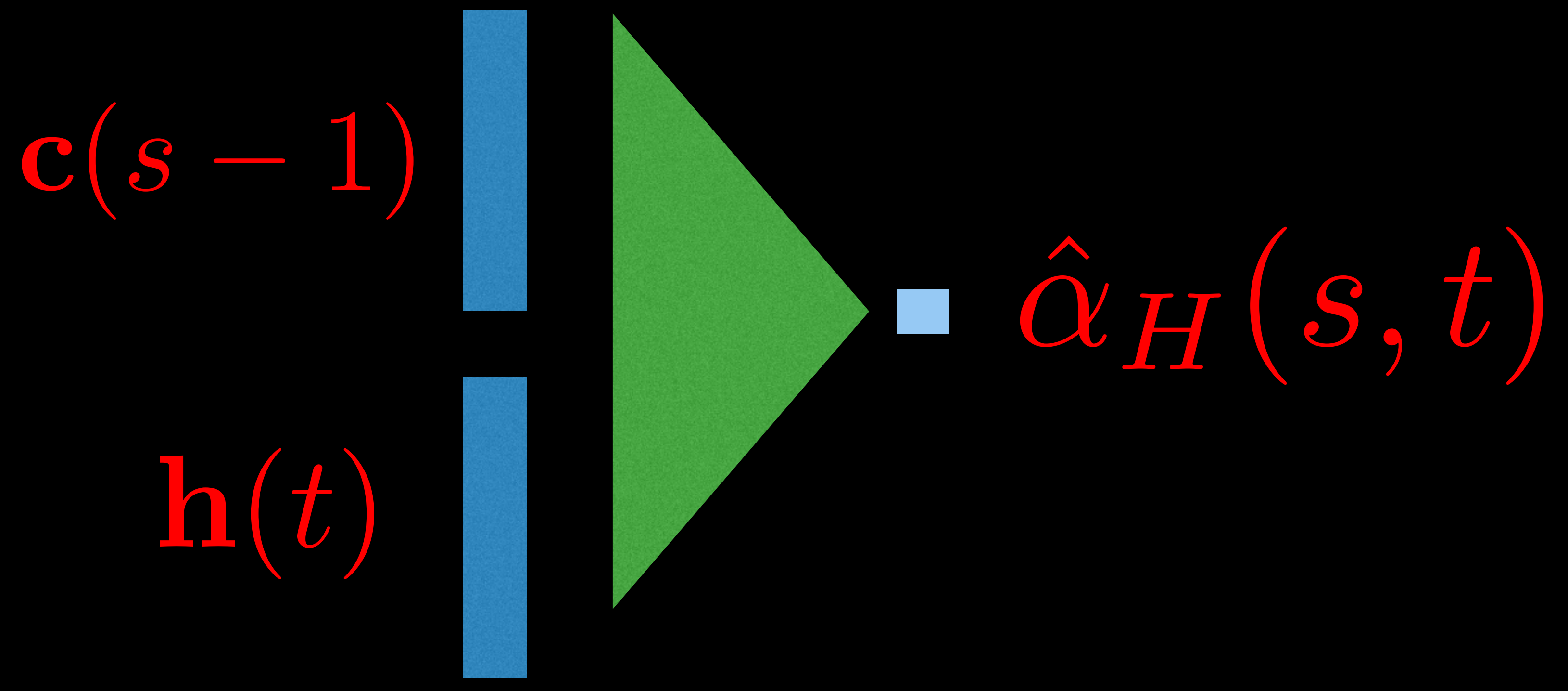


Multi-head attention

- Having more than one attention heads



…



$$\hat{\alpha}_1(s,t) = \mathbf{A}_1[\mathbf{c}(s-1);\mathbf{h}(t)]$$

$$\mathbf{e}_1(s) = \sum_t \alpha_1(s,t)\mathbf{h}(t)$$

…

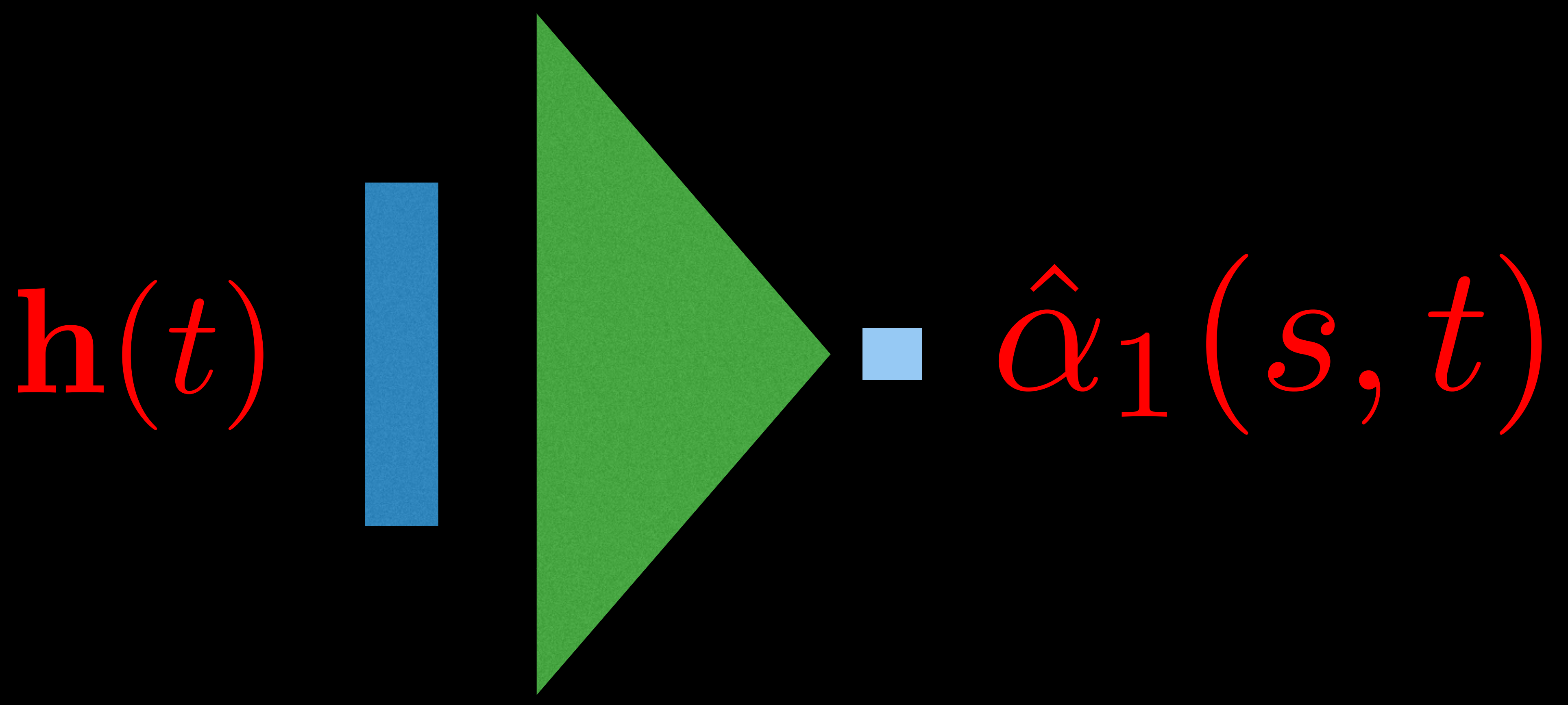
$$\hat{\alpha}_H(s,t) = \mathbf{A}_H[\mathbf{c}(s-1);\mathbf{h}(t)]$$

$$\mathbf{e}_H(s) = \sum_t \alpha_H(s,t)\mathbf{h}(t)$$

$$\mathbf{e}(s) = [\mathbf{e}_1(s); \ldots; \mathbf{e}_H(s)]$$

Self-attention

✳ Using attention layers without feedback from decoder.



✳ Without feedback the attention performs,

➡ temporal relevance weighting of the input time-series (hidden layer representations).

Issues in RNNs/LSTMs

- Issues of long-term dependency
  - ➡ LSTMs have partial solutions
- Back propagation through time
  - ➡ Does not allow parallelism in forward pass or backward pass.
  - ➡ Significant increase in training time as well as in forward propagation.
- Question - can we use attention mechanism itself to build temporal dependencies without recurrence.

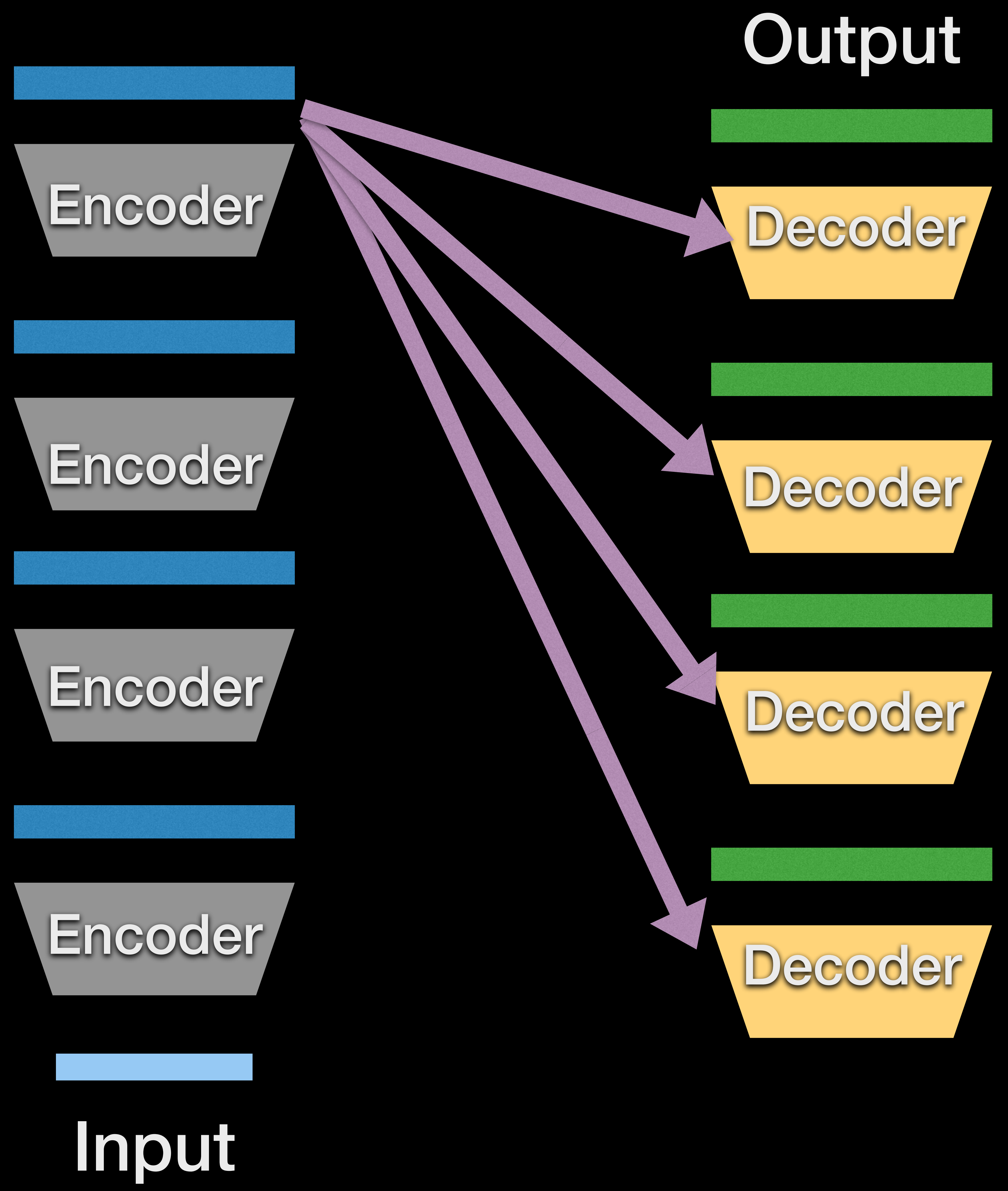
Transformers

- Encoder Decoder architecture based models.

- Uses only feed forward architectures with self-attention.

  ➡ Multi-head self attention.

- All the encoder layers and the decoder layers have the same set of operations.



Transformers - encoder

- Let $\mathbf{x}(1)...\mathbf{x}(T)$ denote the input and let $\mathbf{e}^l(1)...\mathbf{e}^l(T)$ denote encoder outputs at layer l.

$$\tilde{\mathbf{E}}^{p-1} = Layernorm\left([\mathbf{e}^{p-1}(1)...\mathbf{e}^{p-1}(T)]\right)^T \in \mathcal{R}^{T\times D}$$

$$\mathbf{Q}_h^{(p)} = \bar{\mathbf{E}}^{(p-1)}\mathbf{W}_h^{(p,Q)} + \mathbf{1}\mathbf{b}_h^{(p,Q)\top} \in \mathbb{R}^{T\times d},$$
$$\mathbf{K}_h^{(p)} = \bar{\mathbf{E}}^{(p-1)}\mathbf{W}_h^{(p,K)} + \mathbf{1}\mathbf{b}_h^{(p,K)\top} \in \mathbb{R}^{T\times d},$$
$$\mathbf{V}_h^{(p)} = \bar{\mathbf{E}}^{(p-1)}\mathbf{W}_h^{(p,V)} + \mathbf{1}\mathbf{b}_h^{(p,V)\top} \in \mathbb{R}^{T\times d},$$

$$\mathbf{A}_h^{(p)} = \mathbf{Q}_h^{(p)}\mathbf{K}_h^{(p)\top} \in \mathbb{R}^{T\times T}$$

Transformers - encoder

* Let $\mathbf{x}(1) \ldots \mathbf{x}(T)$ denote the input and let $\mathbf{e}^l(1) \ldots \mathbf{e}^l(T)$ denote encoder outputs at layer l.

$$\hat{\mathbf{A}}_h^{(p)} = \mathrm{Softmax}\left(\frac{\mathbf{A}_h^{(p)}}{\sqrt{d}}\right) \in \mathbb{R}^{T \times T}.$$

$$\mathbf{C}_h^{(p)} = \hat{\mathbf{A}}_h^{(p)} \mathbf{V}_h^{(p)} \in \mathbb{R}^{T \times d}.$$

$$\mathbf{E}^{(p,\mathrm{SA})} = [\mathbf{C}_1^{(p)} \cdots \mathbf{C}_H^{(p)}] \mathbf{W}^{(p,O)} + \mathbf{1}\mathbf{b}^{(p,O)\top} \in \mathbb{R}^{T \times D}.$$

$$\bar{\mathbf{E}}^{(p,\mathrm{SA})} = \mathrm{LayerNorm}(\bar{\mathbf{E}}^{(p-1)} + \mathbf{E}^{(p,\mathrm{SA})}) \in \mathbb{R}^{T \times D}.$$