Quebec
Artificial
Intelligence
Institute

Mila

Sequence to Sequence Models

Mirko Bronzi
Applied Research Scientist, Mila
mirko.bronzi@mila.quebec

Plan

- RNN Recap
- Sequence to Sequence Models
- Attention Mechanism
- Transformer
- Libraries and References

Plan

- **RNN Recap**
- Sequence to Sequence
- Attention Mechanism
- Transformer
- Libraries and References

Recurrent Neural Networks

- The parameters of the model are **shared** over time.
- The internal state (h_t) is updated at each time step.

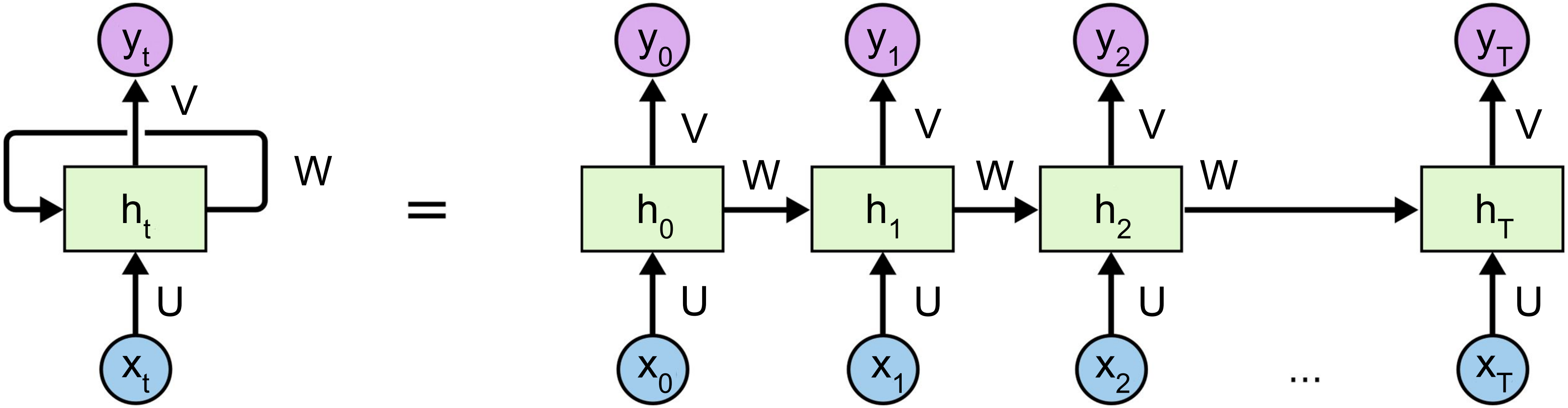


Image from Christopher Olah's blog

The initial internal state (h_{-1}) is dropped for simplicity

Backpropagation Through Time

- The global error is:

$$E = \sum_{t=0}^{T} E_t$$

- To compute the gradient of the global error with respect to a parameter, we compute the gradient of the individual error at each time step, and then sum all those values. For example:

$$\frac{\partial E}{\partial U} = \sum_{t=0}^{T} \frac{\partial E_t}{\partial U}$$

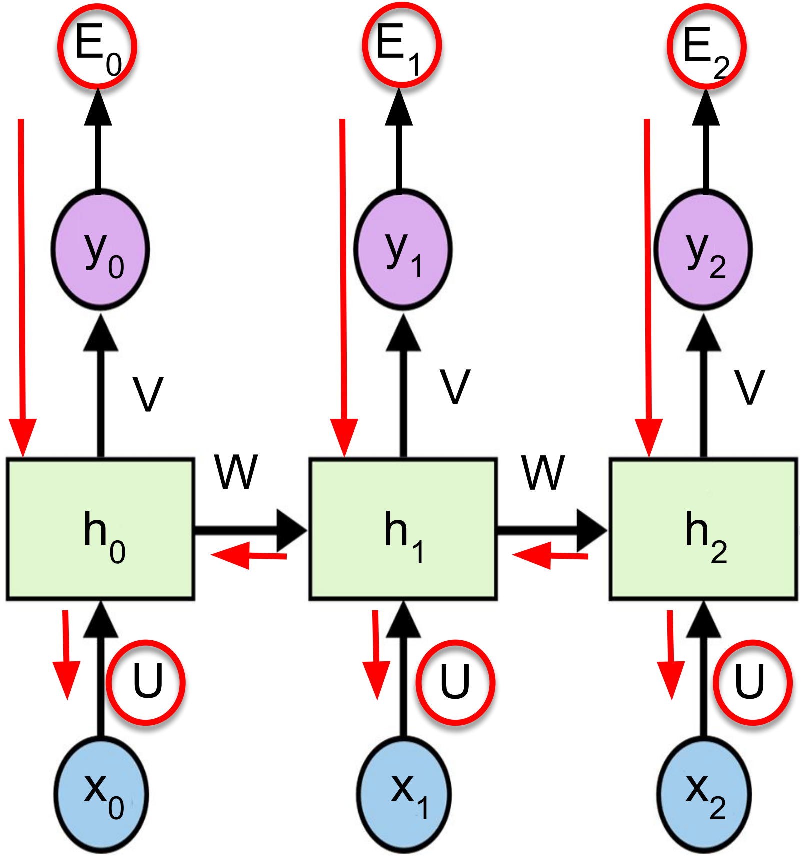


Image from Christopher Olah's blog

Mila

Long-Term Dependencies

- Long-term dependencies are difficult to learn due to the long chain of gradients $\frac{\partial h_T}{\partial h_{T-1}} \cdots \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial h_0}$ that can lead to vanishing gradients

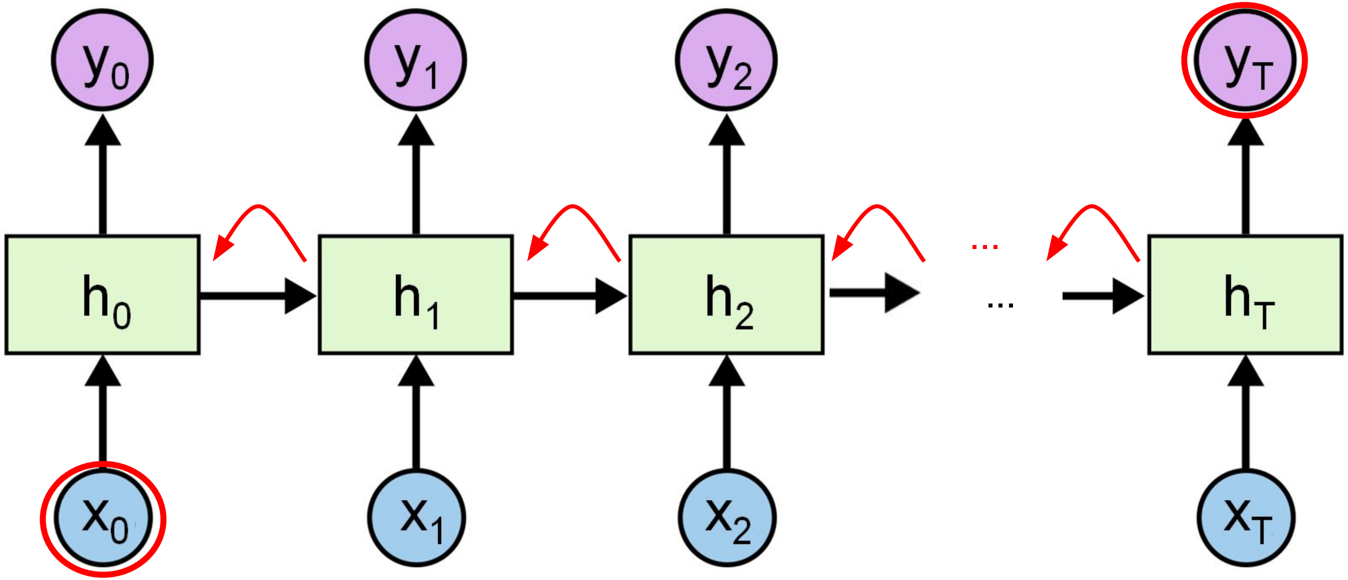


Image from Christopher Olah's blog

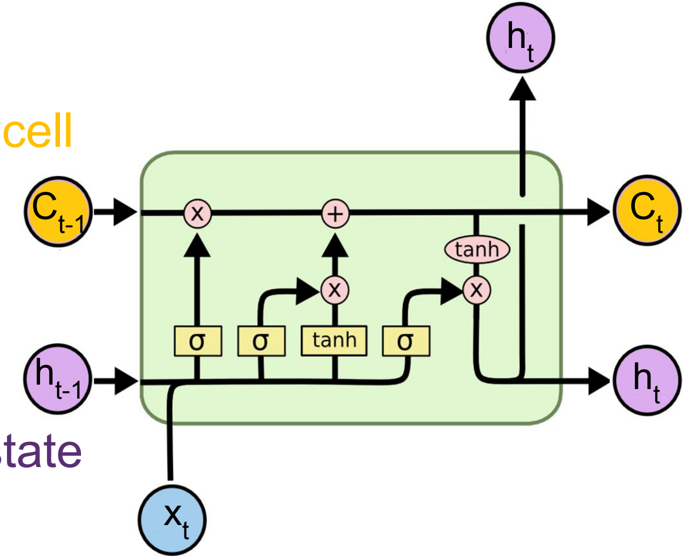
Mila

Long Short-Term Memory (LSTM)

- Reduce the vanishing gradient problem using a gate mechanism and adding a memory cell.

Memory cell

Internal state



$$i_t = \sigma(U_i x_t + W_i h_{t-1} + b_i)$$

$$f_t = \sigma(U_f x_t + W_f h_{t-1} + b_f)$$

$$o_t = \sigma(U_o x_t + W_o h_{t-1} + b_o)$$

$$g_t = tanh(U_g x_t + W_g h_{t-1} + b_g)$$

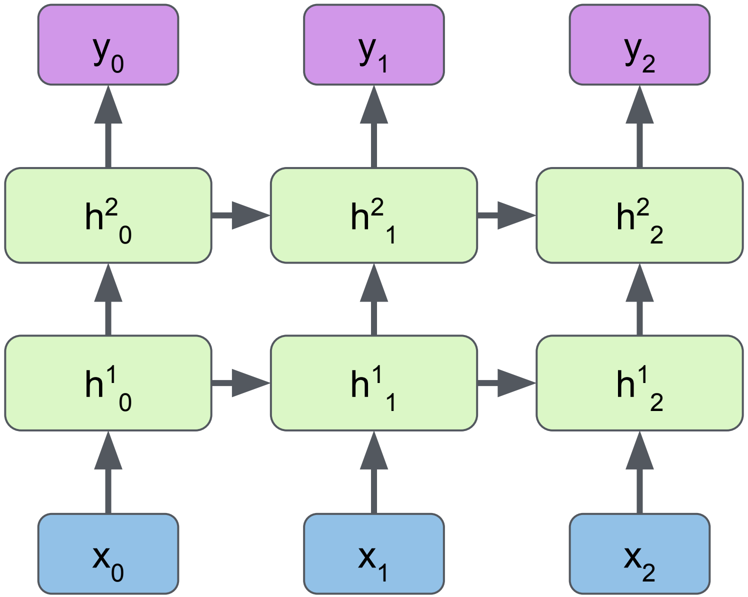
$$C_t = i_t \times g_t + f_t \times C_{t-1}$$

$$h_t = o_t \times tanh(C_t)$$

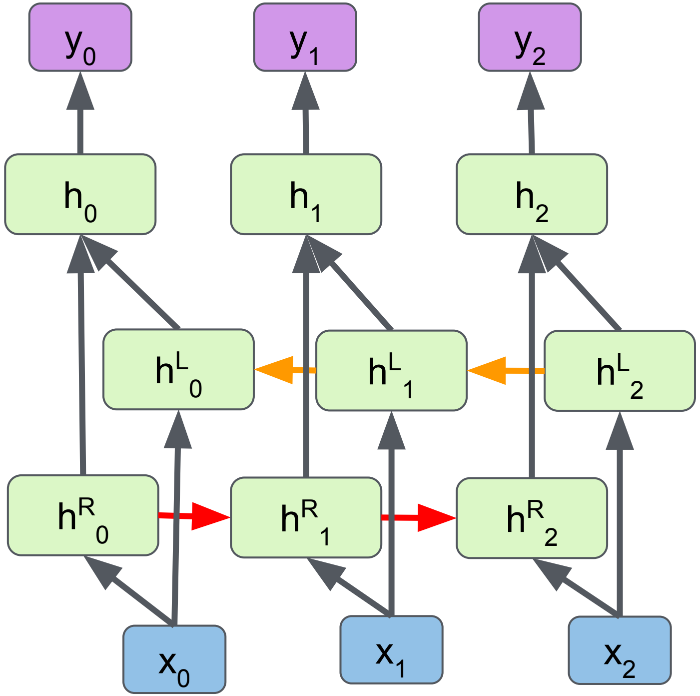
Image from Christopher Olah's blog

Hochreiter et al., Long short-term memory, Neural Computation 1997

Multi-Layer and Bidirectional RNNs



Layers of RNNs

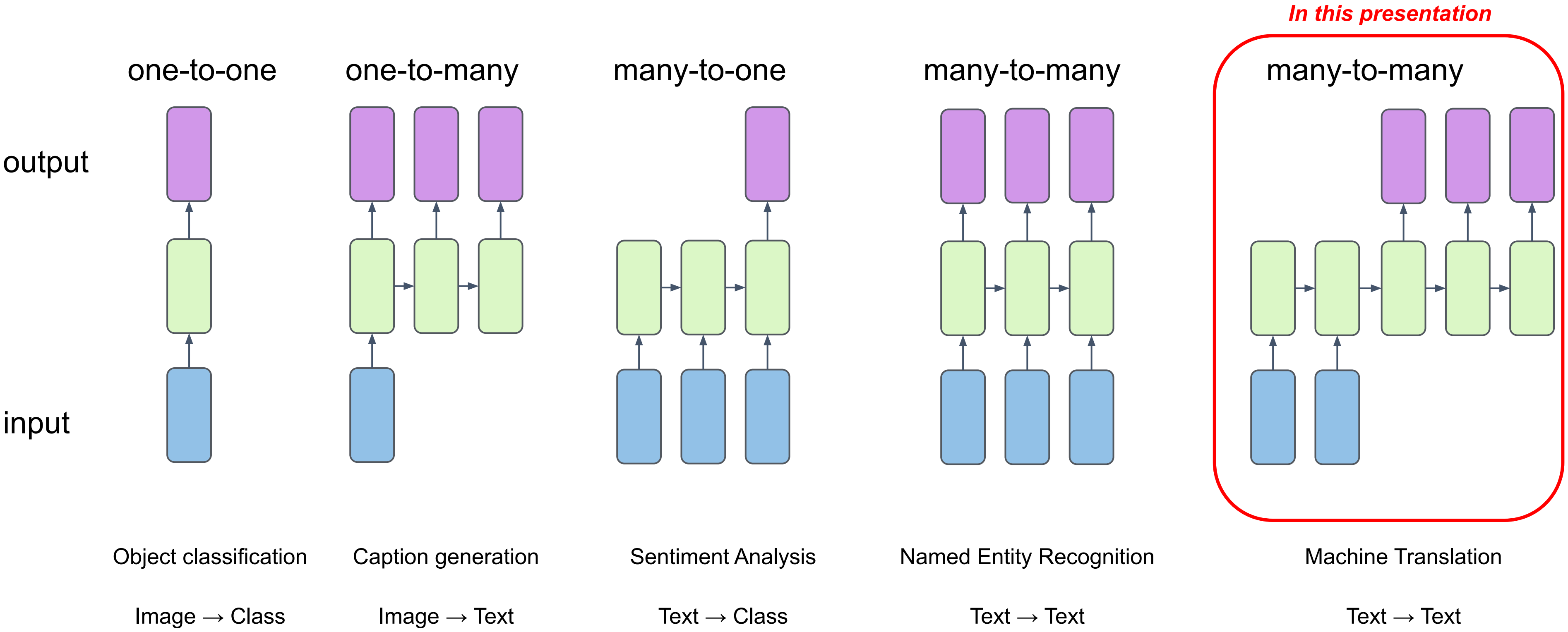


Bidirectional RNNs

Plan

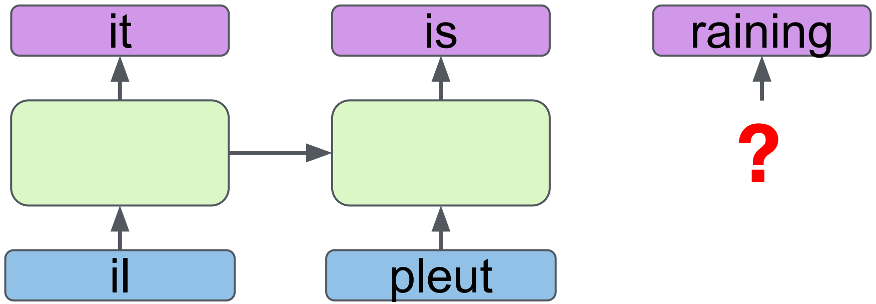
- RNN Recap
- **Sequence to Sequence Models**
- Attention Mechanism
- Transformer
- Libraries and References

Modeling Sequences



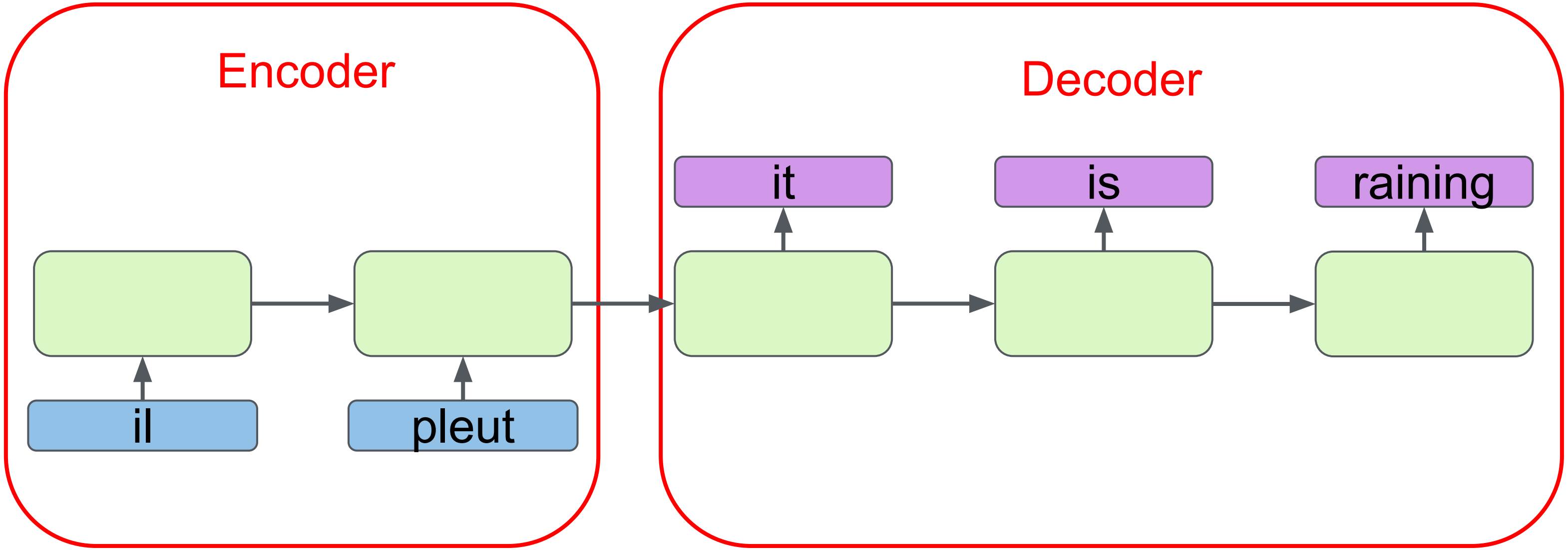
Modeling Sequences

- How to handle input and output sequences of different lengths?
  - Machine translation.
  - Text summarization.
  - …



Different input-output sequence sizes

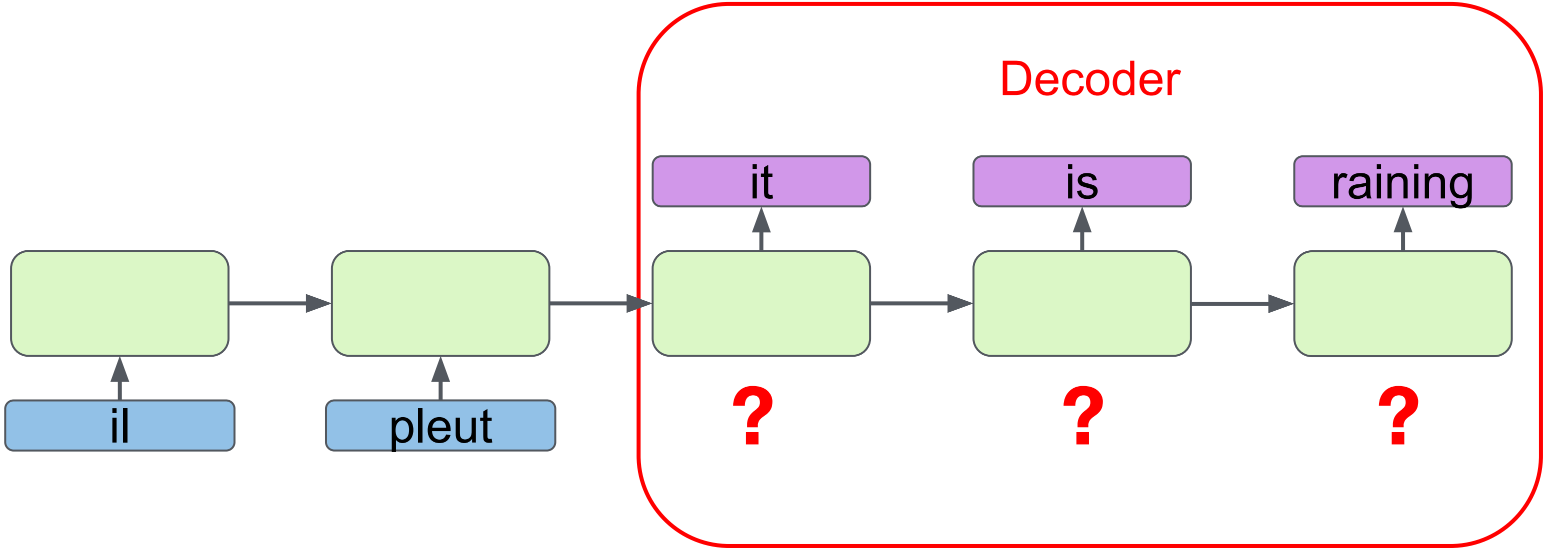
- Create an architecture composed of two components (e.g. two different RNNs):
    - Encoder that processes the input sequence.
    - Decoder that generates the output sequence based on the encoded input.



Mila

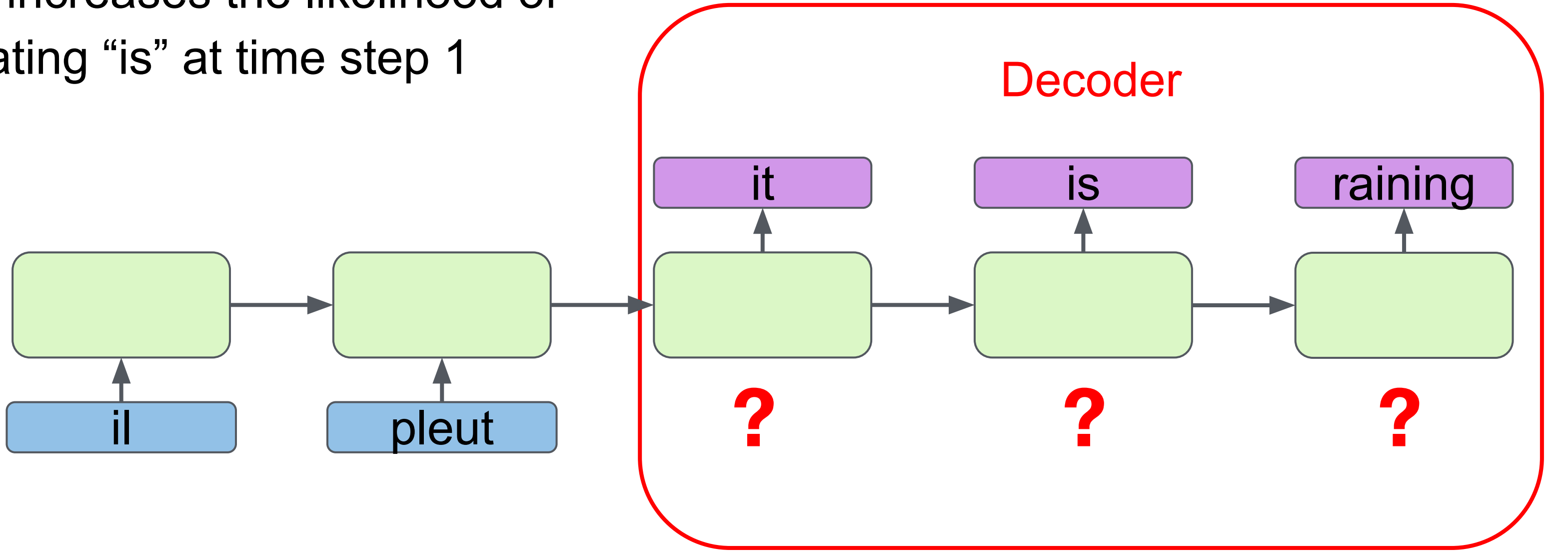
Different input-output sequence sizes

- How to implement the decoder?
- Note the missing input. We need a mechanism that will allow the decoder to generate consistent outputs across time.



Different input-output sequence sizes

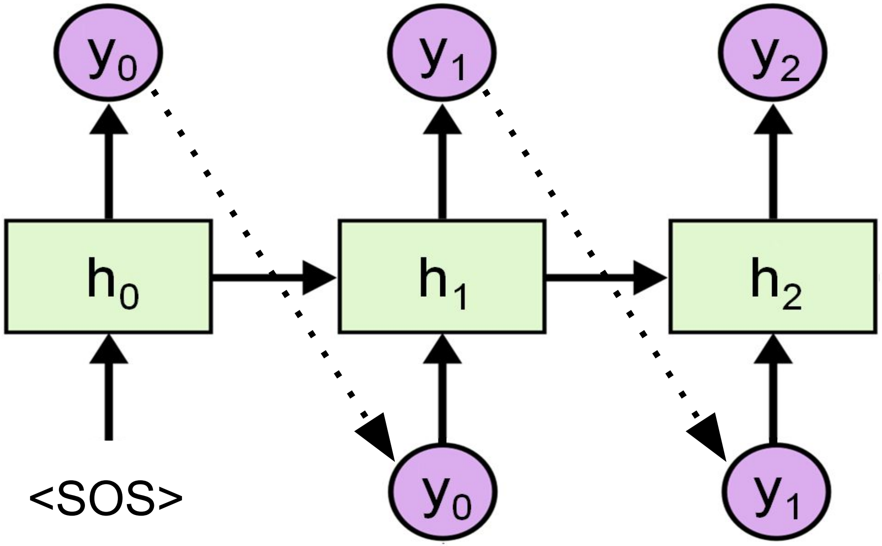
Example: knowing that the decoder generated “it” at time step 0 increases the likelihood of generating “is” at time step 1



Autoregressive RNNs

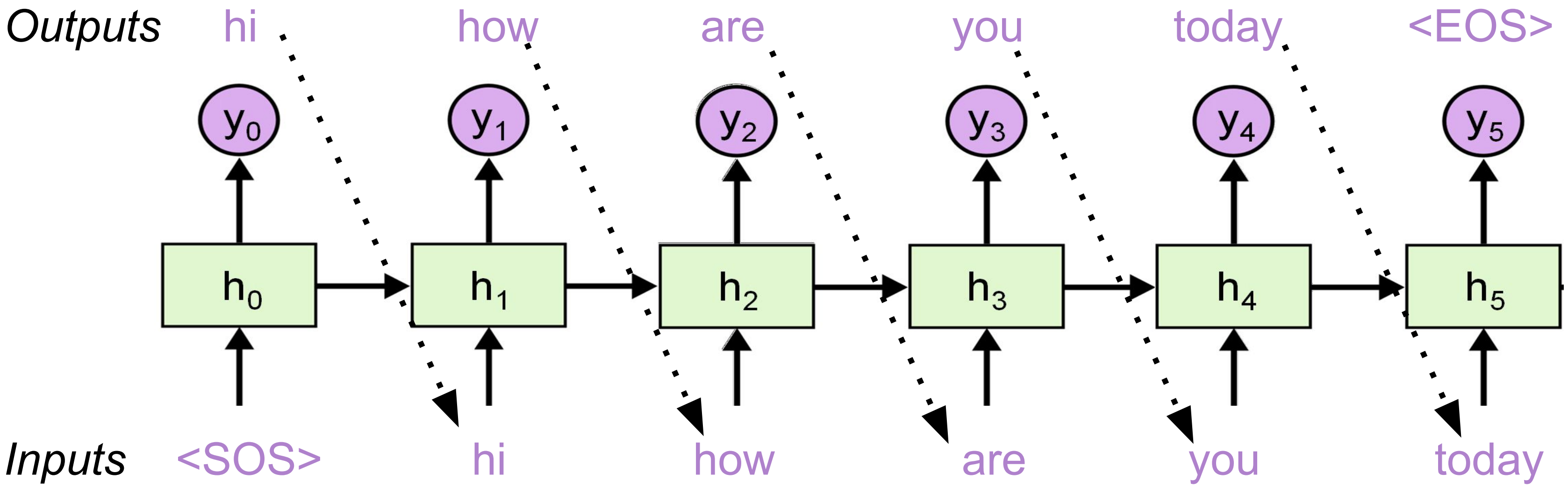
<SOS> *Start of sequence*

- We can use a RNN to **generate** a sequence.
- In order to generate consistent outputs across time, we can condition each output on previously generated outputs.
- Such a model is called **autoregressive**.



Autoregressive RNNs

<SOS> Start of sequence
<EOS> End of sequence

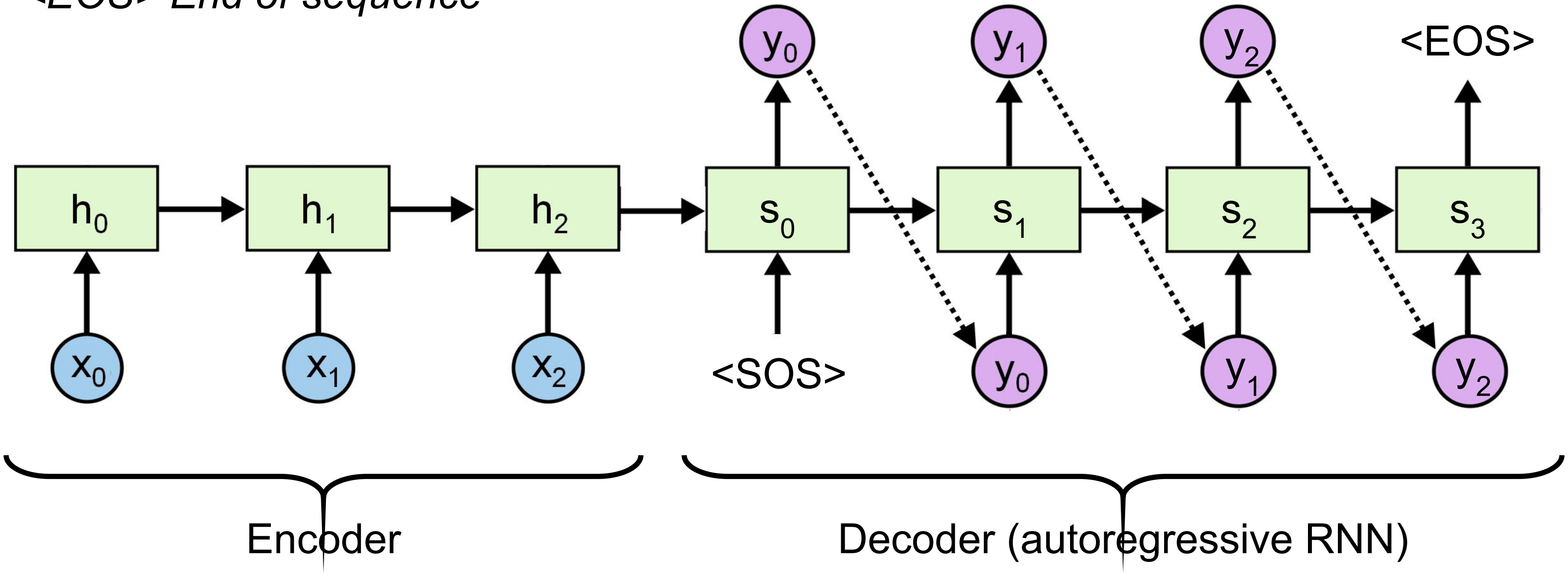


Mila

Sequence-to-Sequence Models

<SOS> Start of sequence

<EOS> End of sequence



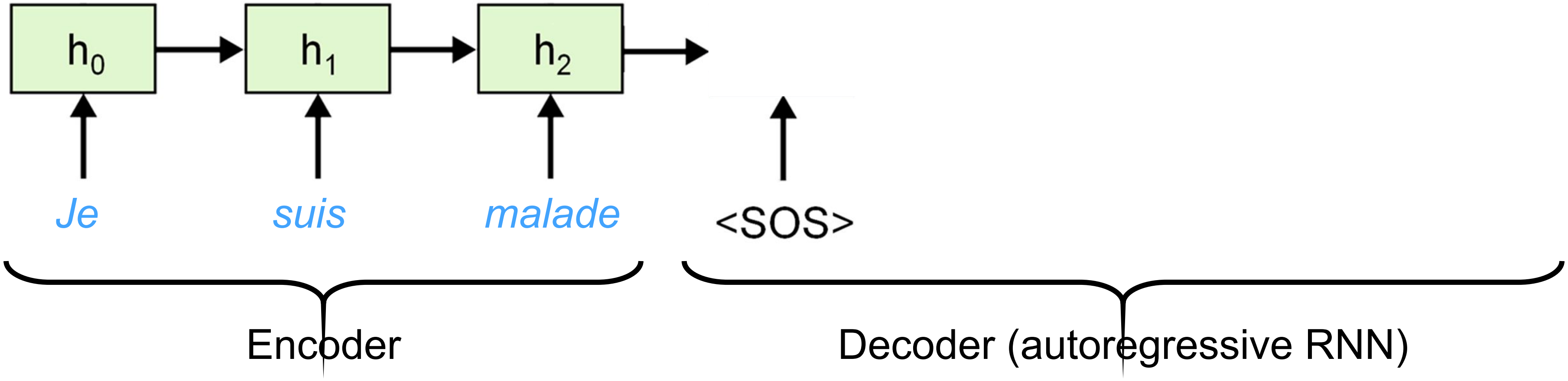
Sutskever et al., Sequence to Sequence Learning with Neural Networks

Cho et al., Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation

Sequence-to-Sequence Models - Example

<SOS> Start of sequence

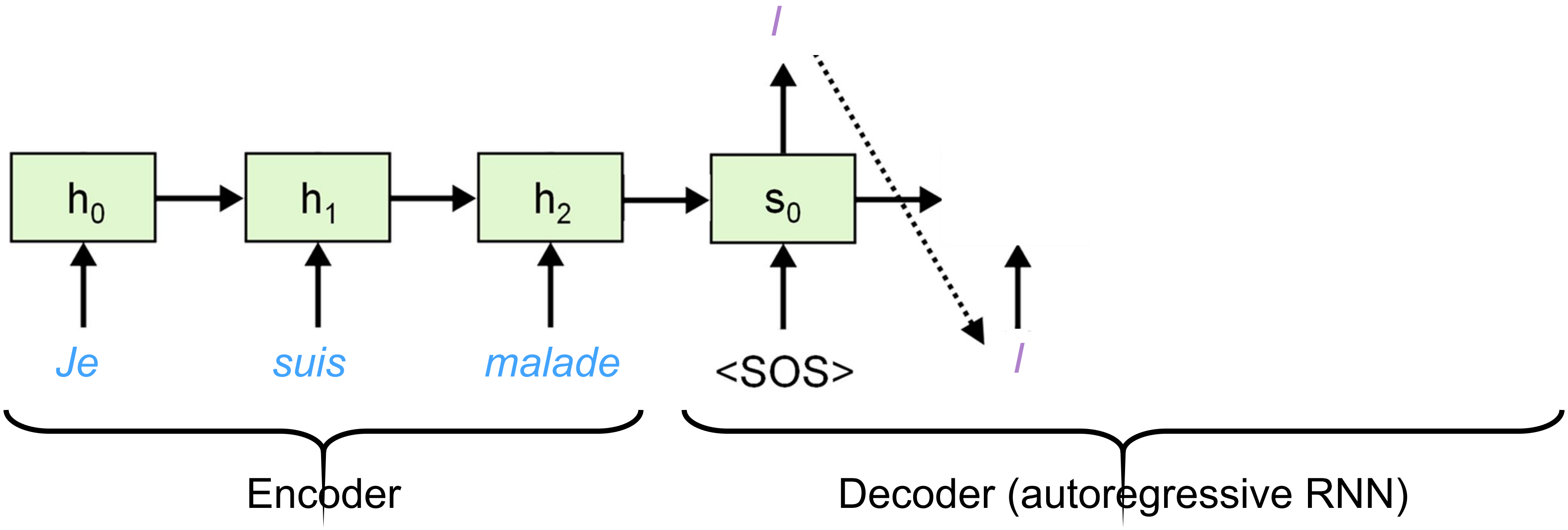
<EOS> End of sequence



Sequence-to-Sequence Models - Example

<SOS> Start of sequence

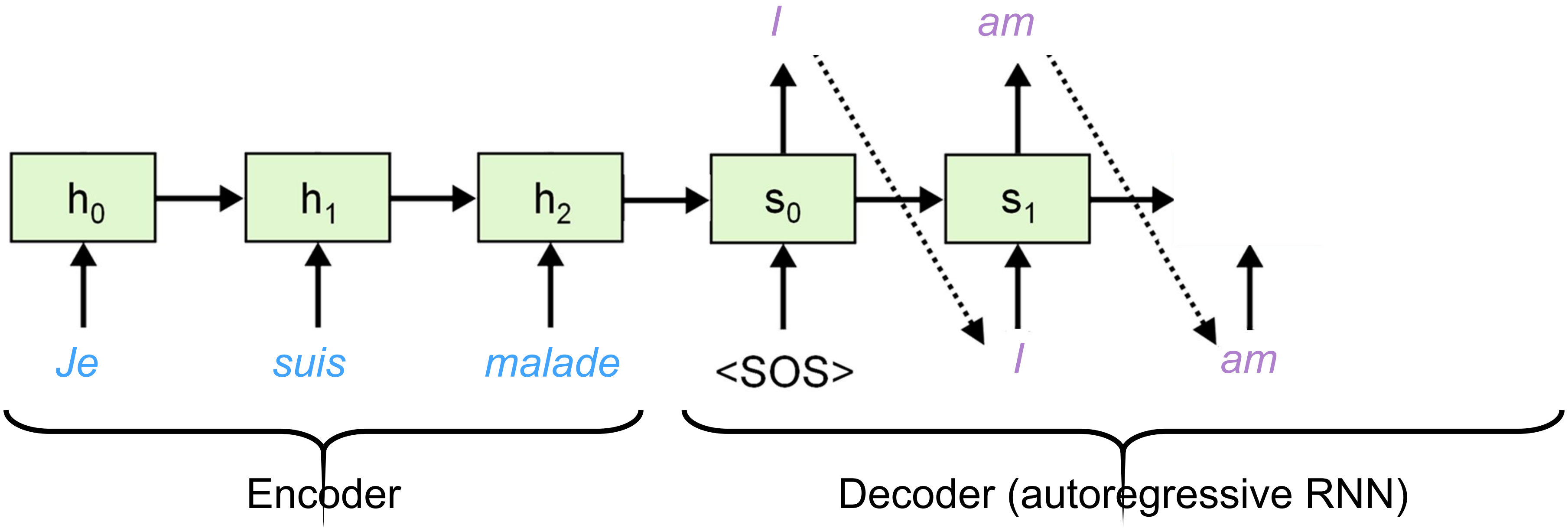
<EOS> End of sequence



Sequence-to-Sequence Models - Example

<SOS> Start of sequence

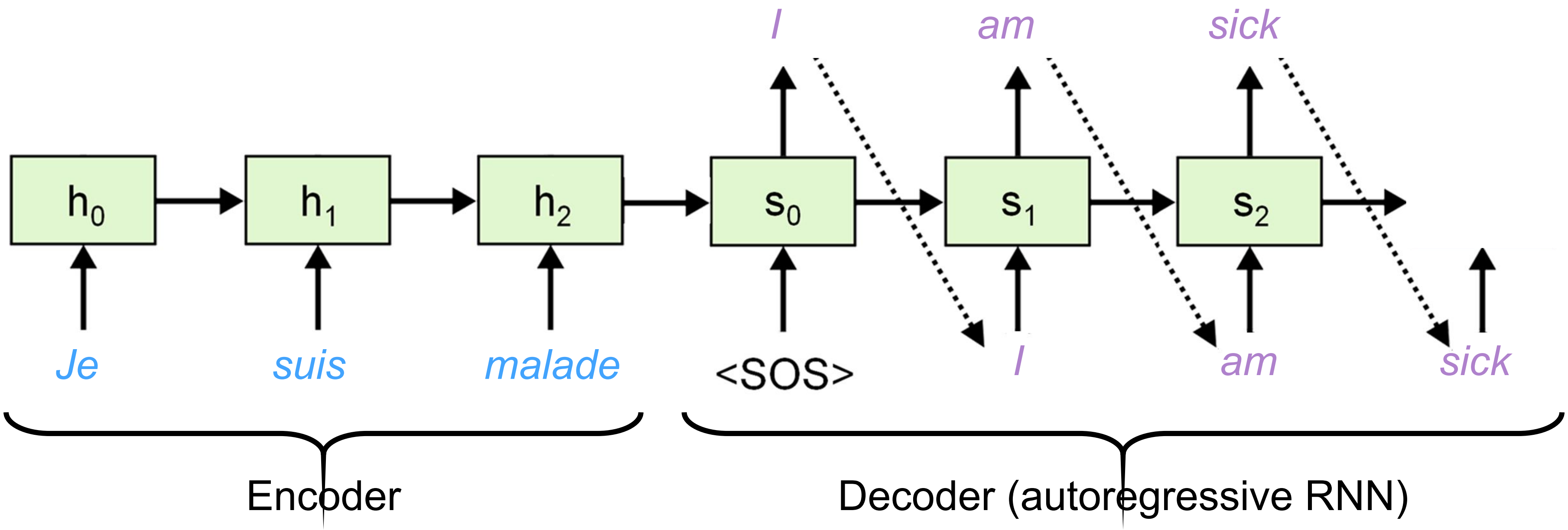
<EOS> End of sequence



Sequence-to-Sequence Models - Example

<SOS> Start of sequence

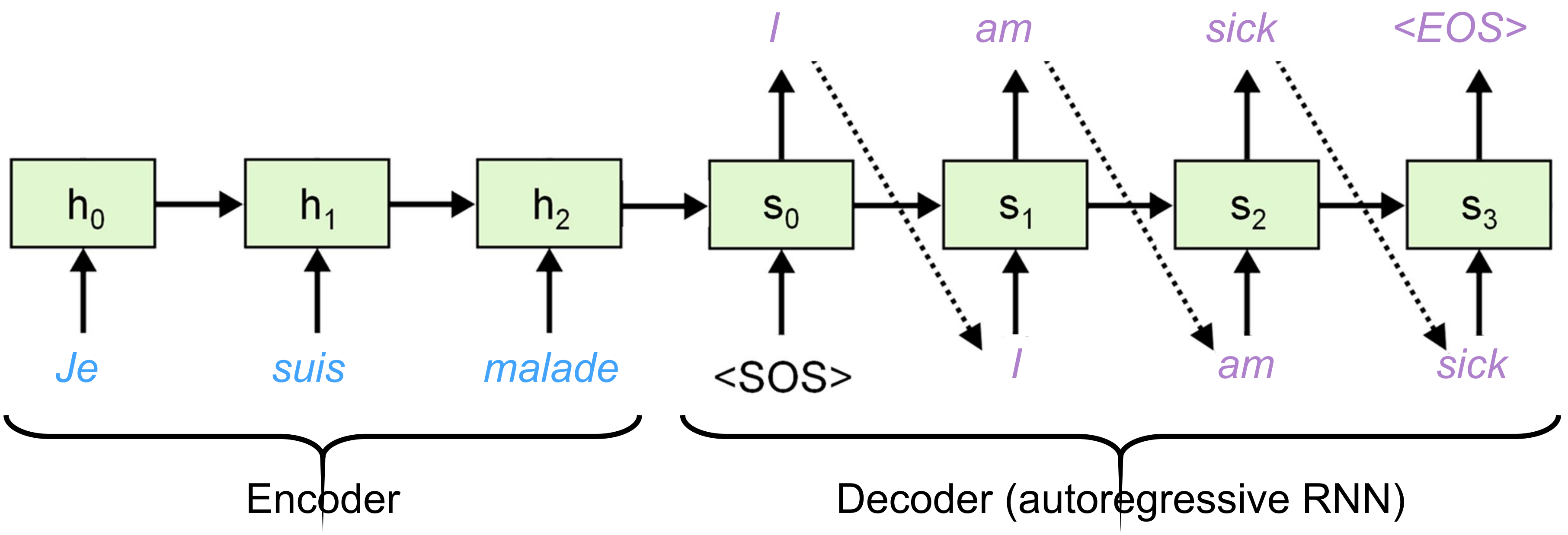
<EOS> End of sequence



Sequence-to-Sequence Models - Example

<SOS> Start of sequence

<EOS> End of sequence

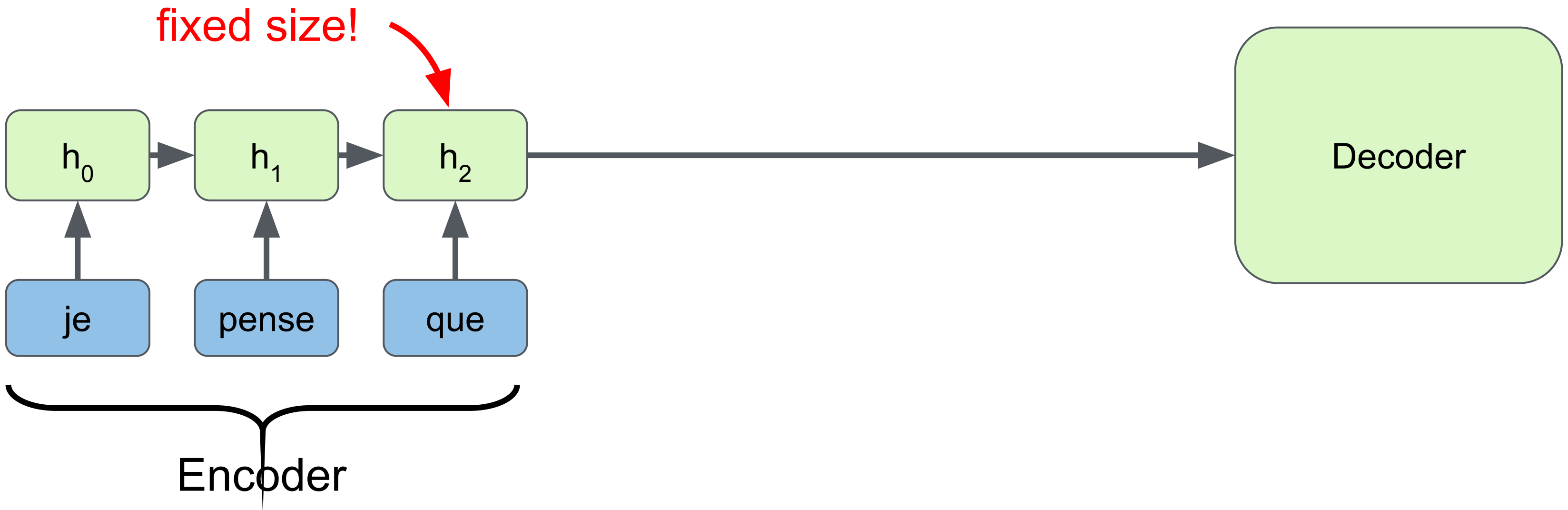


Plan

- RNN Recap
- Sequence to Sequence Models
- **Attention Mechanism**
- Transformer
- Libraries and References

Sequence-to-Sequence Models - Bottleneck

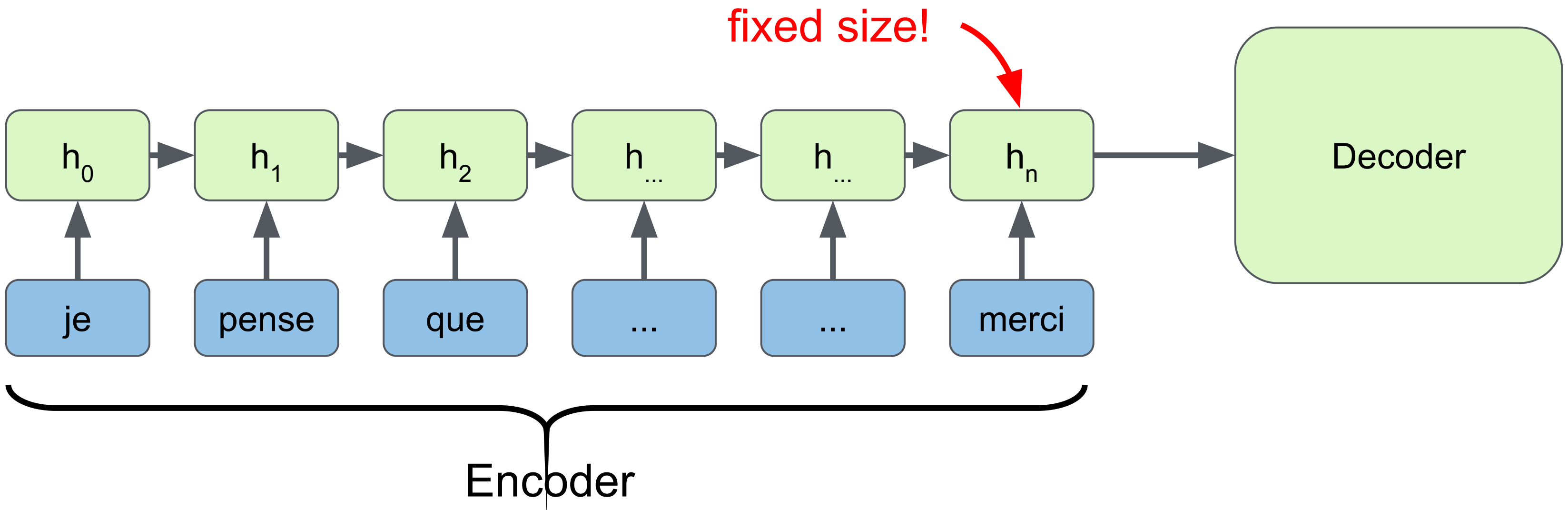
- The encoder has to store/compress all the information from the input into a **fixed size** vector (**h_2** in this example).



Mila

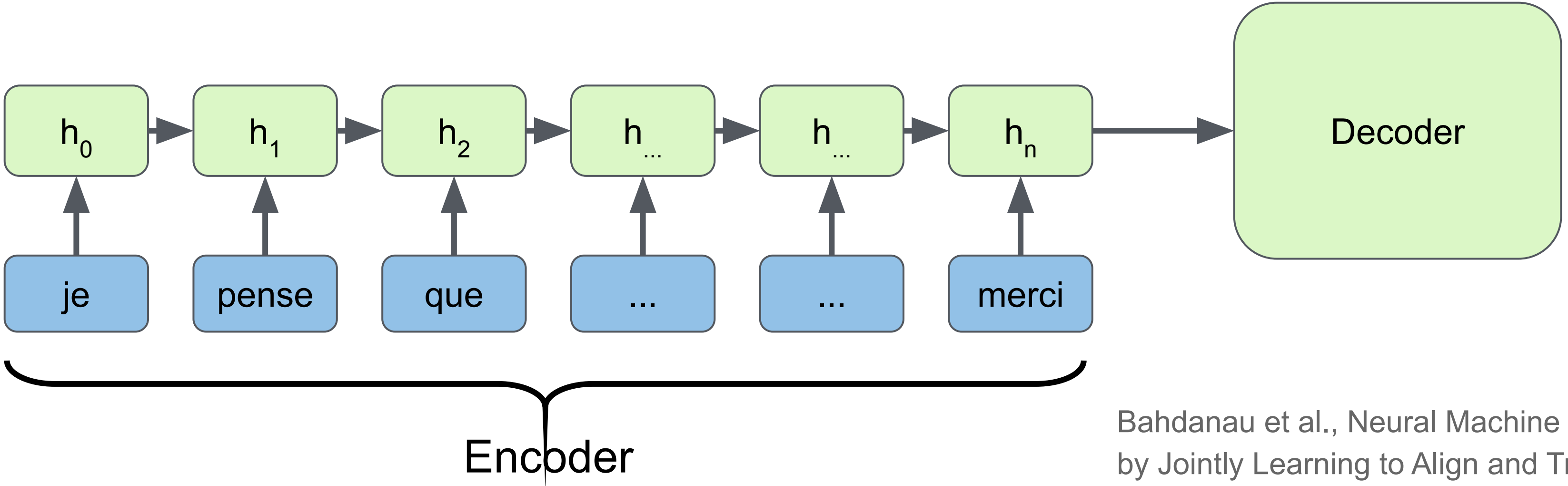
Sequence-to-Sequence Models - Bottleneck

- This is not easy to do with very long input sequences.
- $\mathbf{h_n}$ is a bottleneck.



Attention Mechanism

- Problem: it is not easy to store all the necessary information from an **arbitrary long** sequence into a **fixed-size** vector.

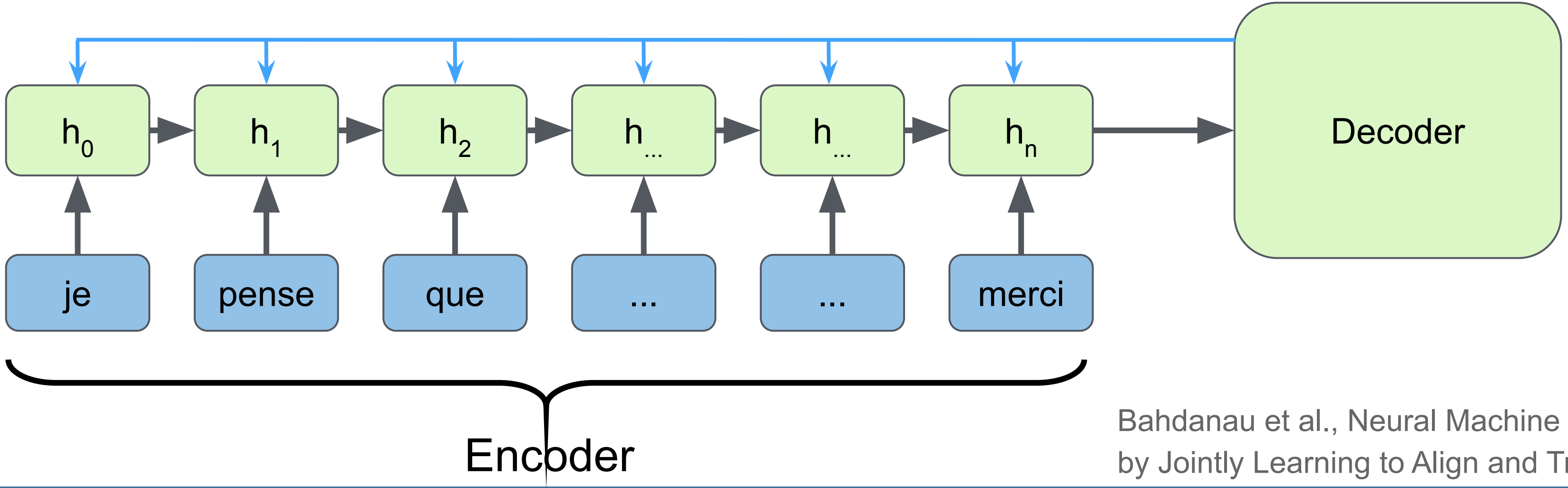


Bahdanau et al., Neural Machine Translation by Jointly Learning to Align and Translate

Mila

Attention Mechanism

- Problem: it is not easy to store all the necessary information from an **arbitrary long** sequence into a **fixed-size** vector.
- A possible solution can be to allow the decoder to "selectively look back" at the encoded input sequence.

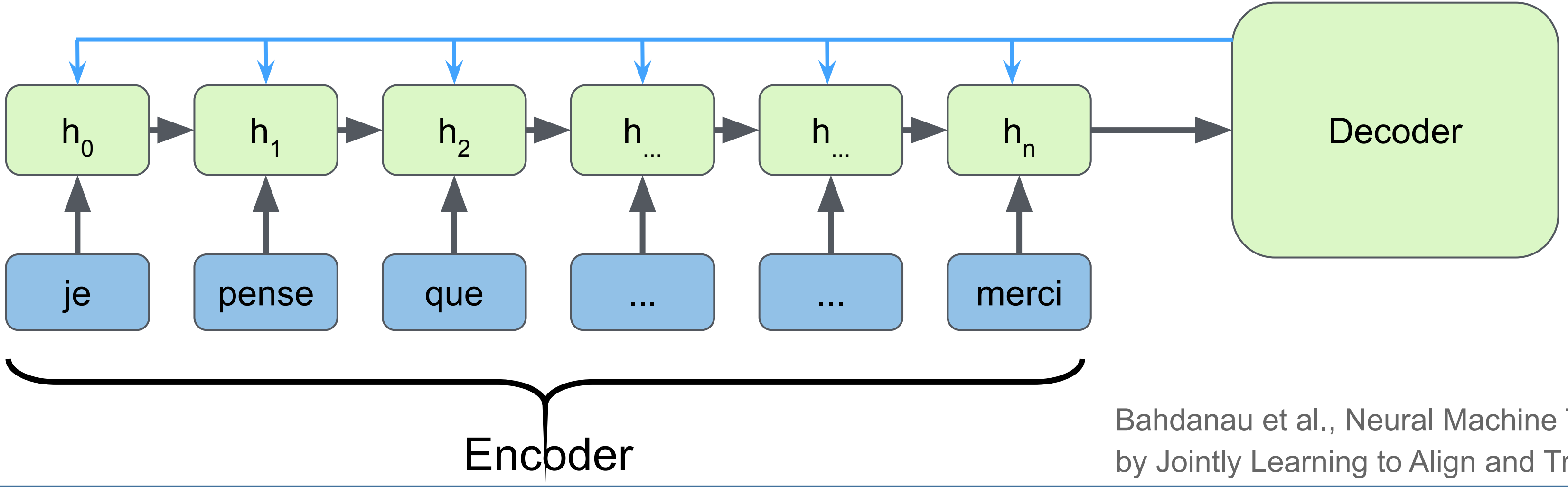


Bahdanau et al., Neural Machine Translation by Jointly Learning to Align and Translate

Mila

Attention Mechanism

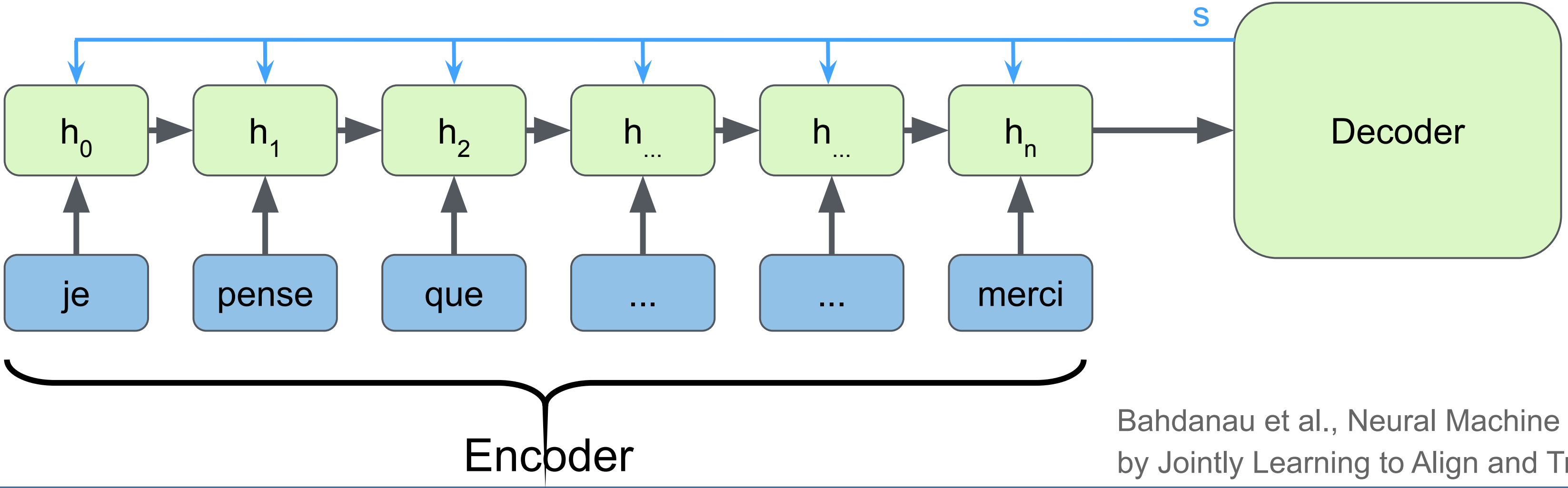
- This can be done with **attention**:
  - At any decoding time step, the decoder can use attention to fetch the relevant information for that step from the encoded input sequence.
- E.g., when producing the output word “think” (in a machine translation task), the decoder can focus on the encoding of the input word “pense”.



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

Attention Mechanism - Formalization

- Attention is a function **A** that, given a decoder state **s** and an encoded input sequence **h**, identifies the elements in **h** that are important at the current decoding time step.

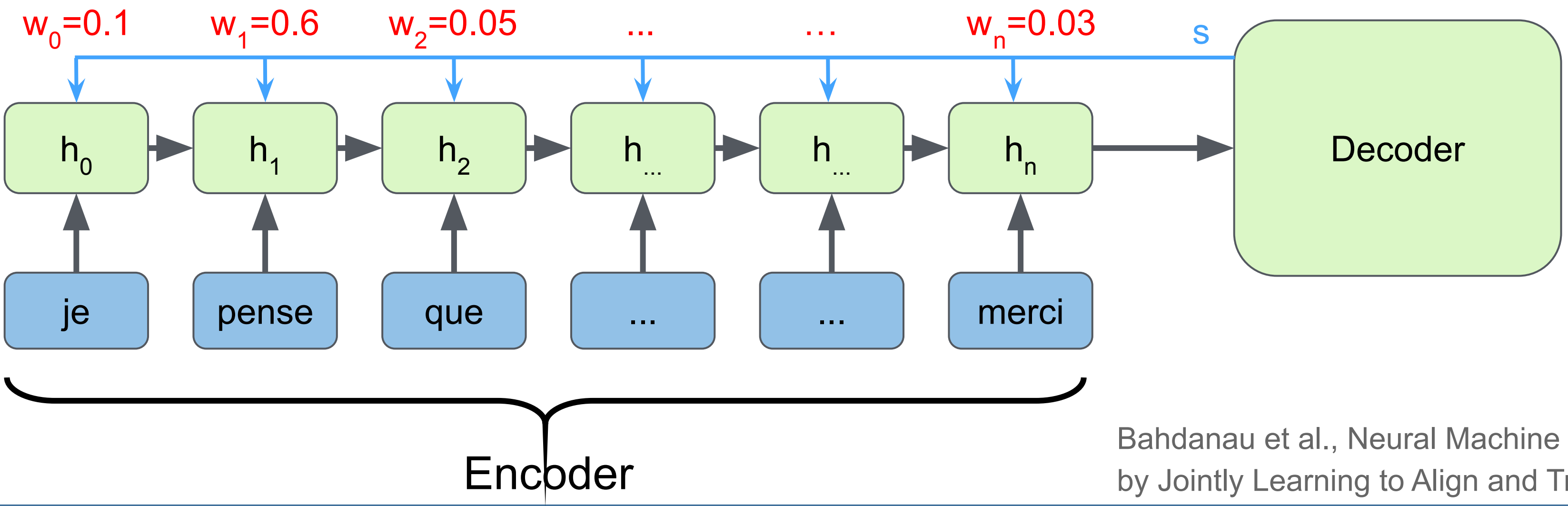


Bahdanau et al., Neural Machine Translation by Jointly Learning to Align and Translate

Mila

Attention Mechanism - Formalization

- Attention is a function A that, given a decoder state s and an encoded input sequence h, identifies the elements in h that are important at the current decoding time step.
  - **A** assigns weights **w** to the elements in **h**.
  - Those weights are normalized (to sum to 1).

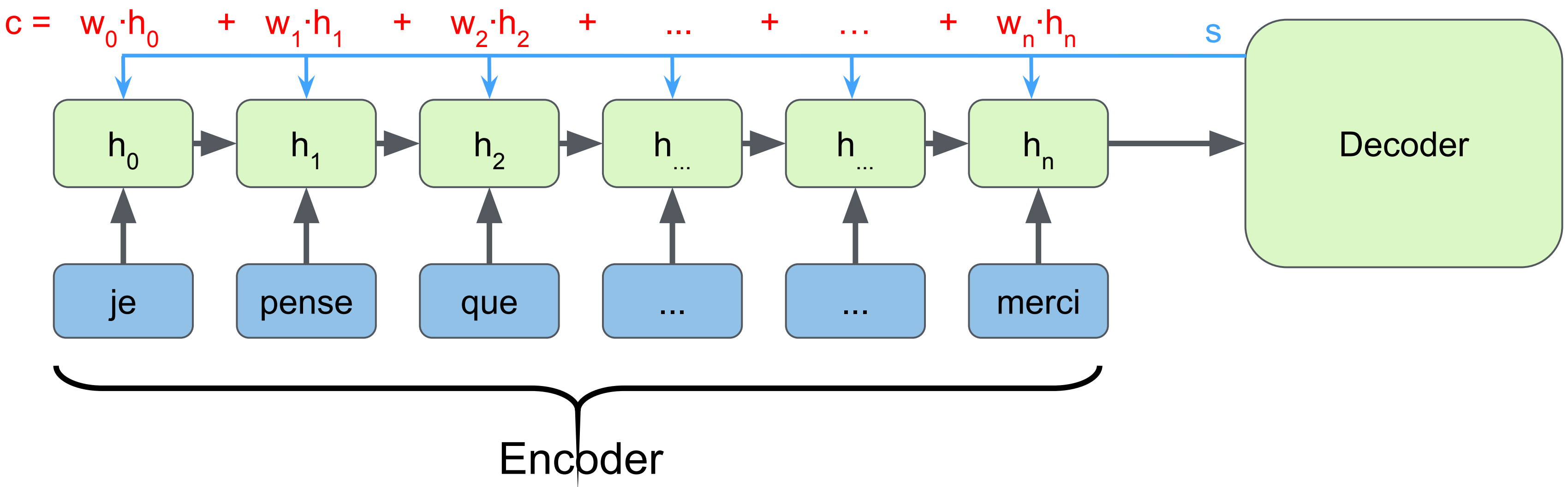


Bahdanau et al., Neural Machine Translation by Jointly Learning to Align and Translate

Attention Mechanism - Formalization

- The weights **w** are used to compute a weighted sum **c** of the elements in the sequence **h**. **c** is called **context vector**.

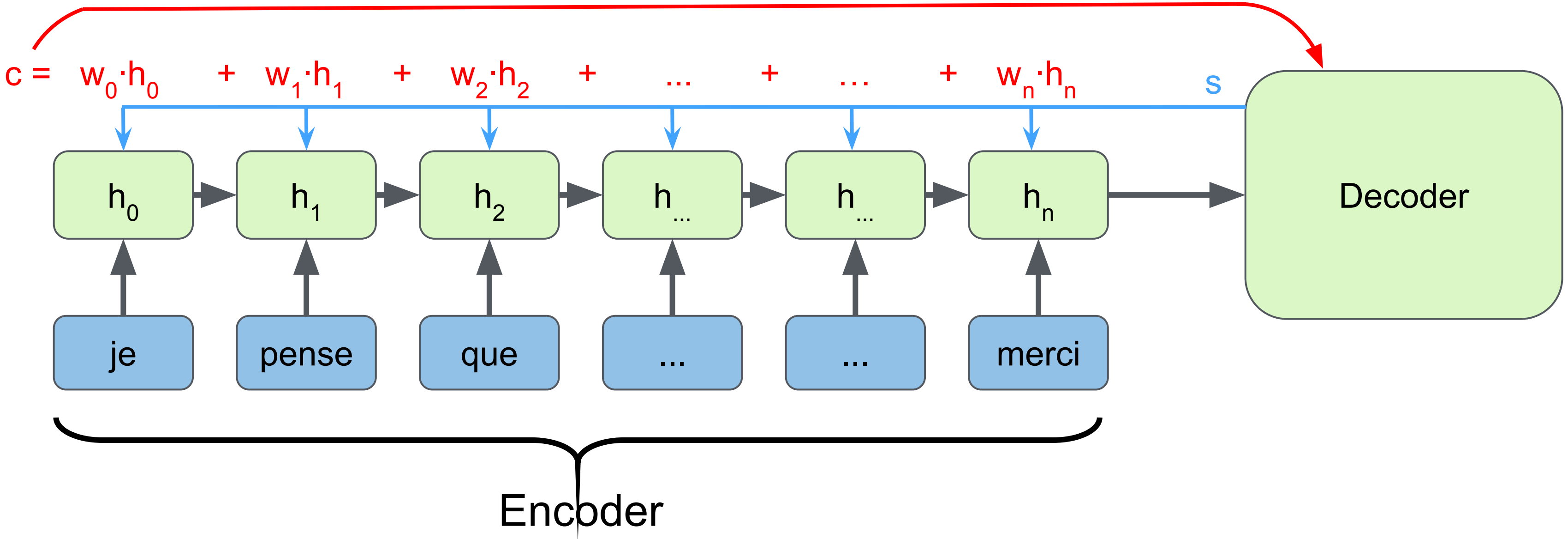
$$c = \sum_{i=0}^{n} w_i \cdot h_i$$



Mila

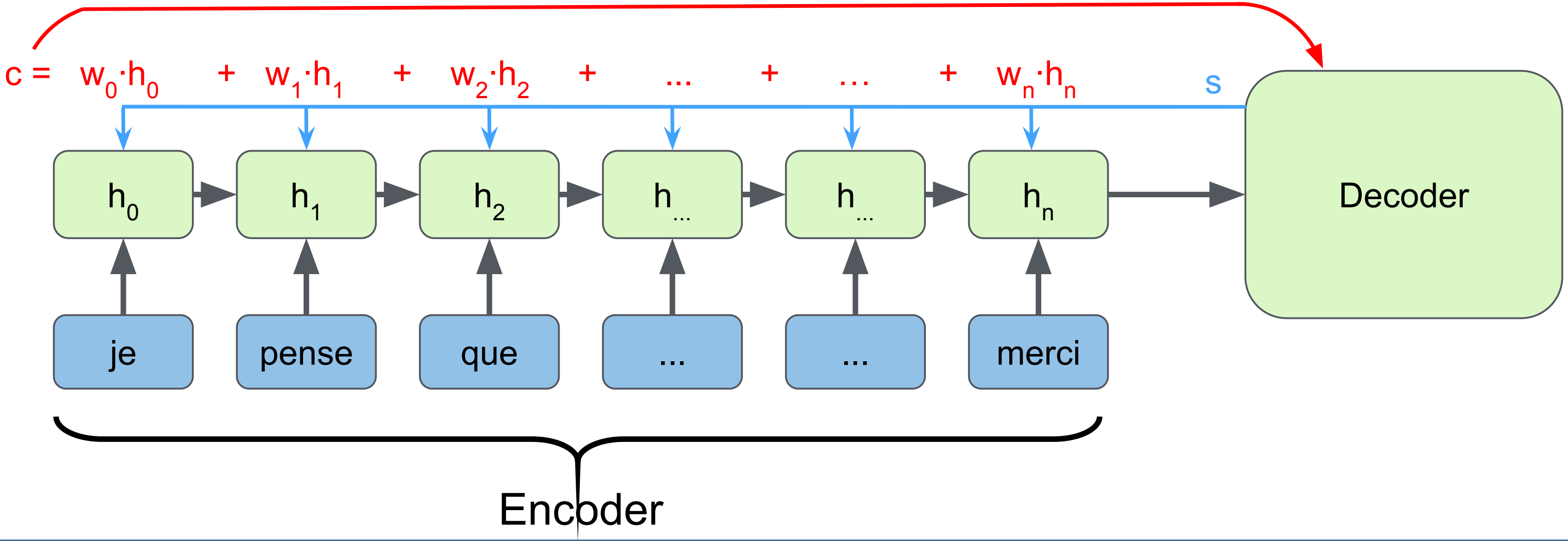
Attention Mechanism - Formalization

- The context vector **c** is then fed to the decoder.



Attention Mechanism - Formalization

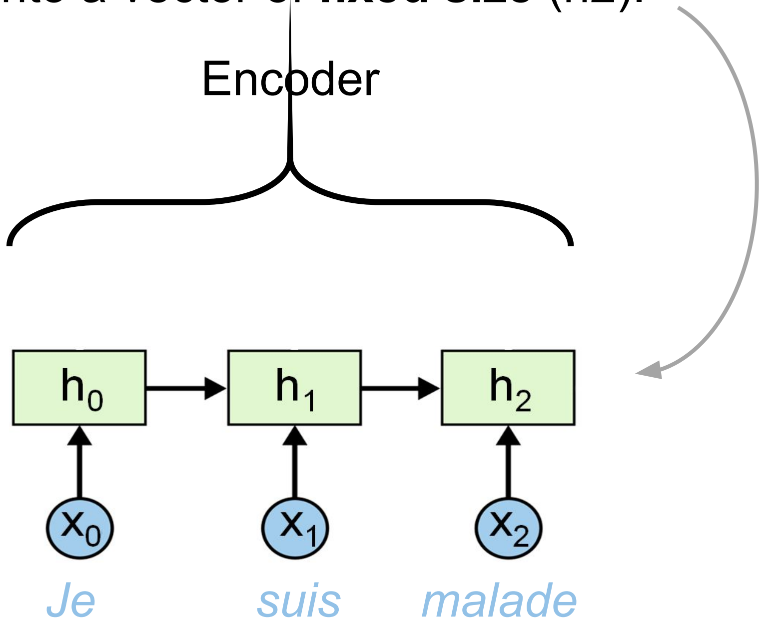
- Let's see a full step-by-step example of the attention mechanism.
- We will then look at how to implement the function **A**.



Example: Sequence-to-Sequence + Attention

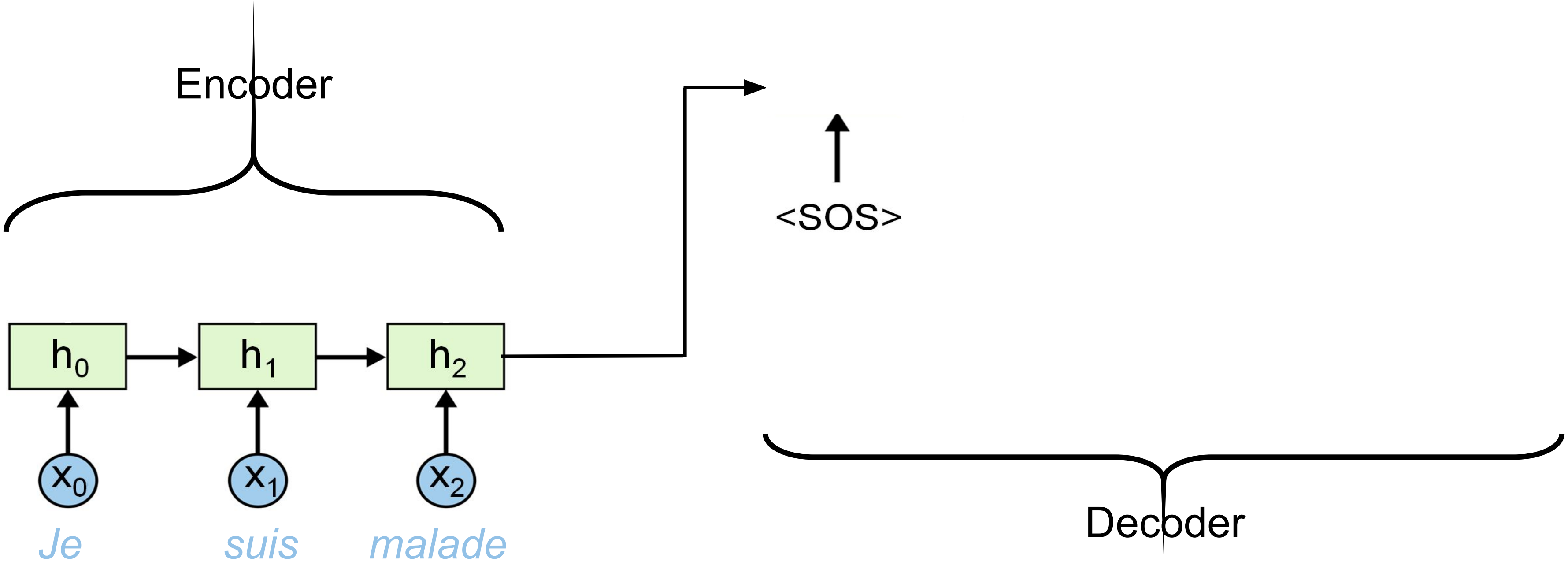
Example: Sequence-to-Sequence + Attention

All the x sequence is encoded into a vector of **fixed size** (h2).



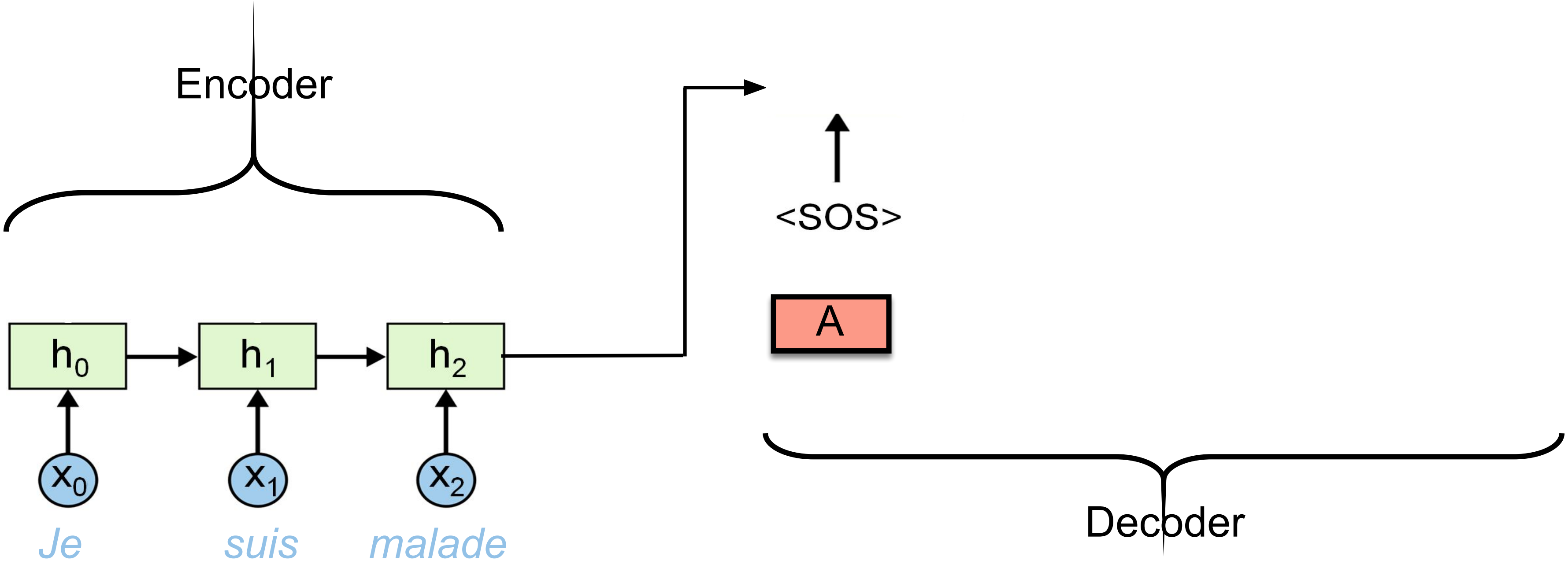
Example: Sequence-to-Sequence + Attention

The decoder starts with the **<SOS>** symbol.



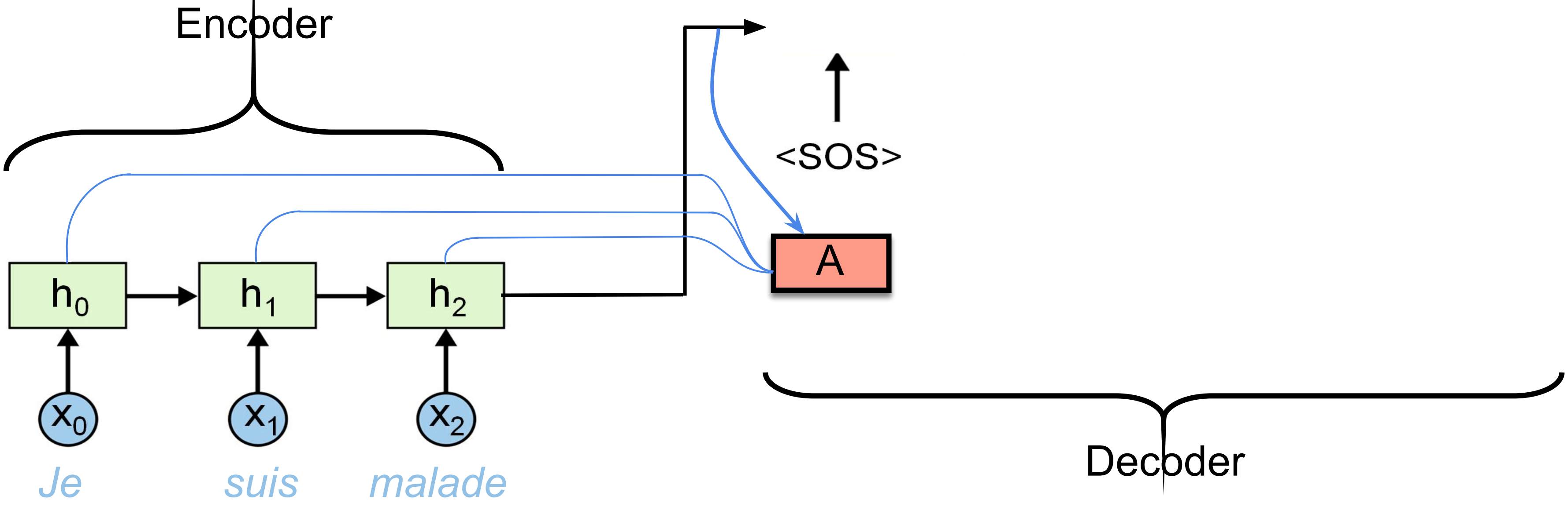
Example: Sequence-to-Sequence + Attention

The attention model **A** is added to the decoder.



Example: Sequence-to-Sequence + Attention

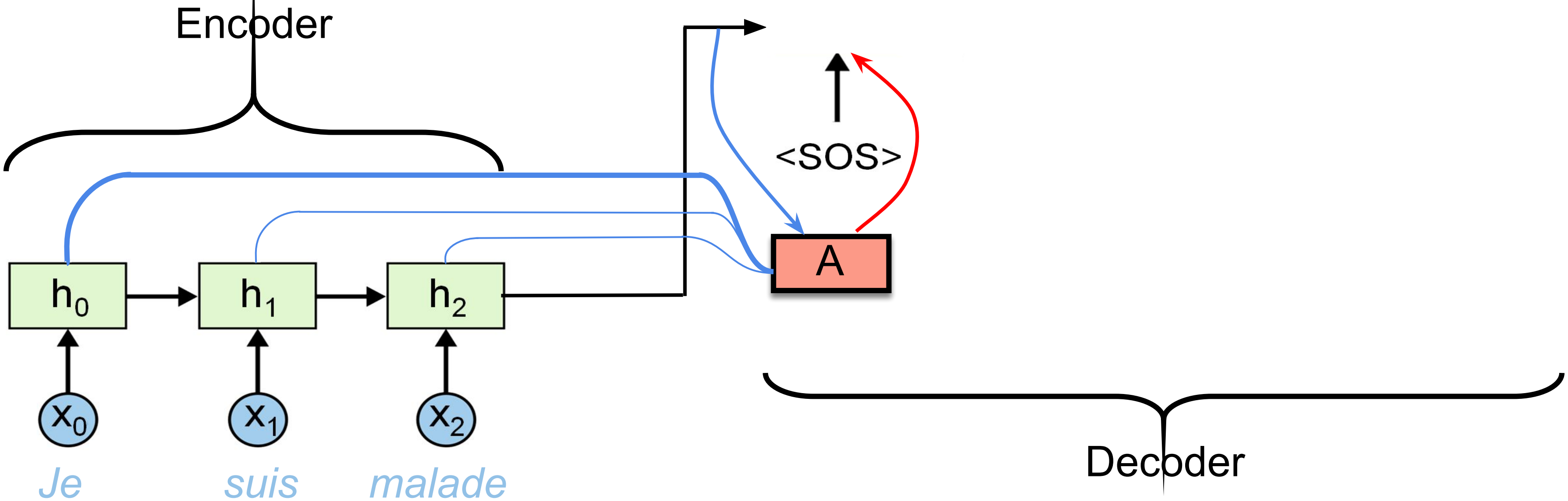
The decoder's previous state and the encoded input sequence **h** are fed as inputs to the attention model (**A**).



Mila

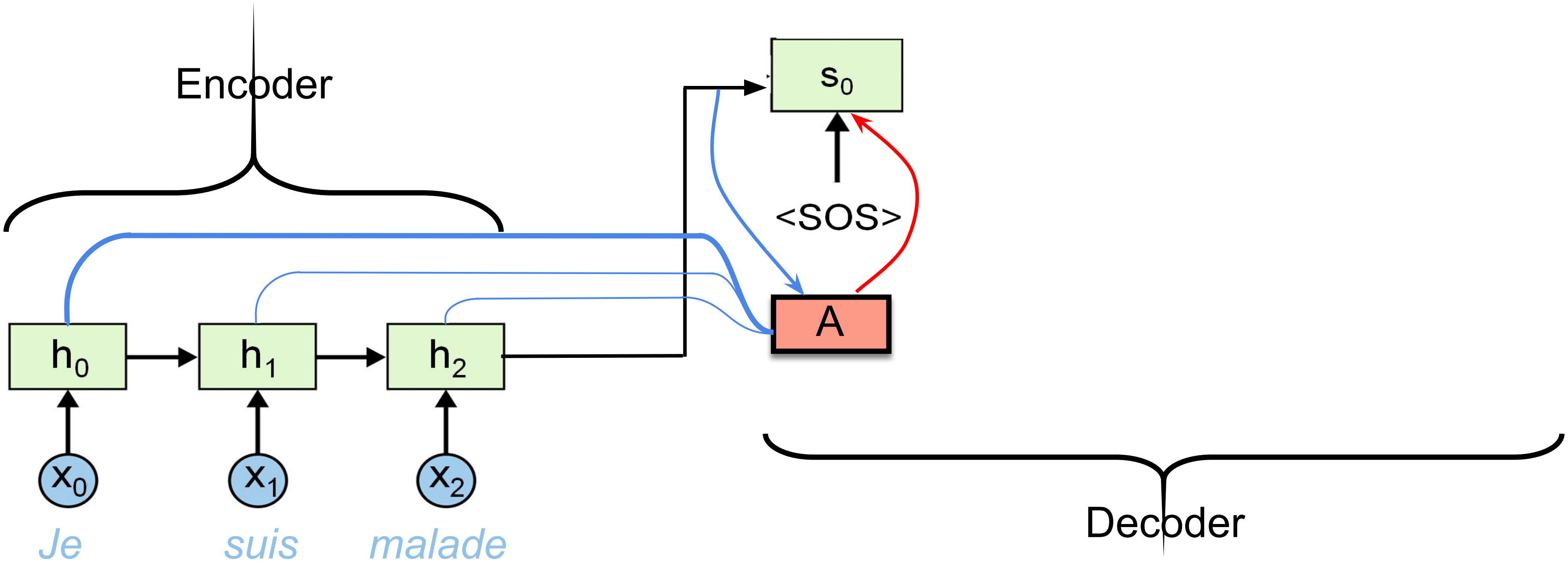
Example: Sequence-to-Sequence + Attention

The context vector (output of the attention) is fed as an input to the decoder.



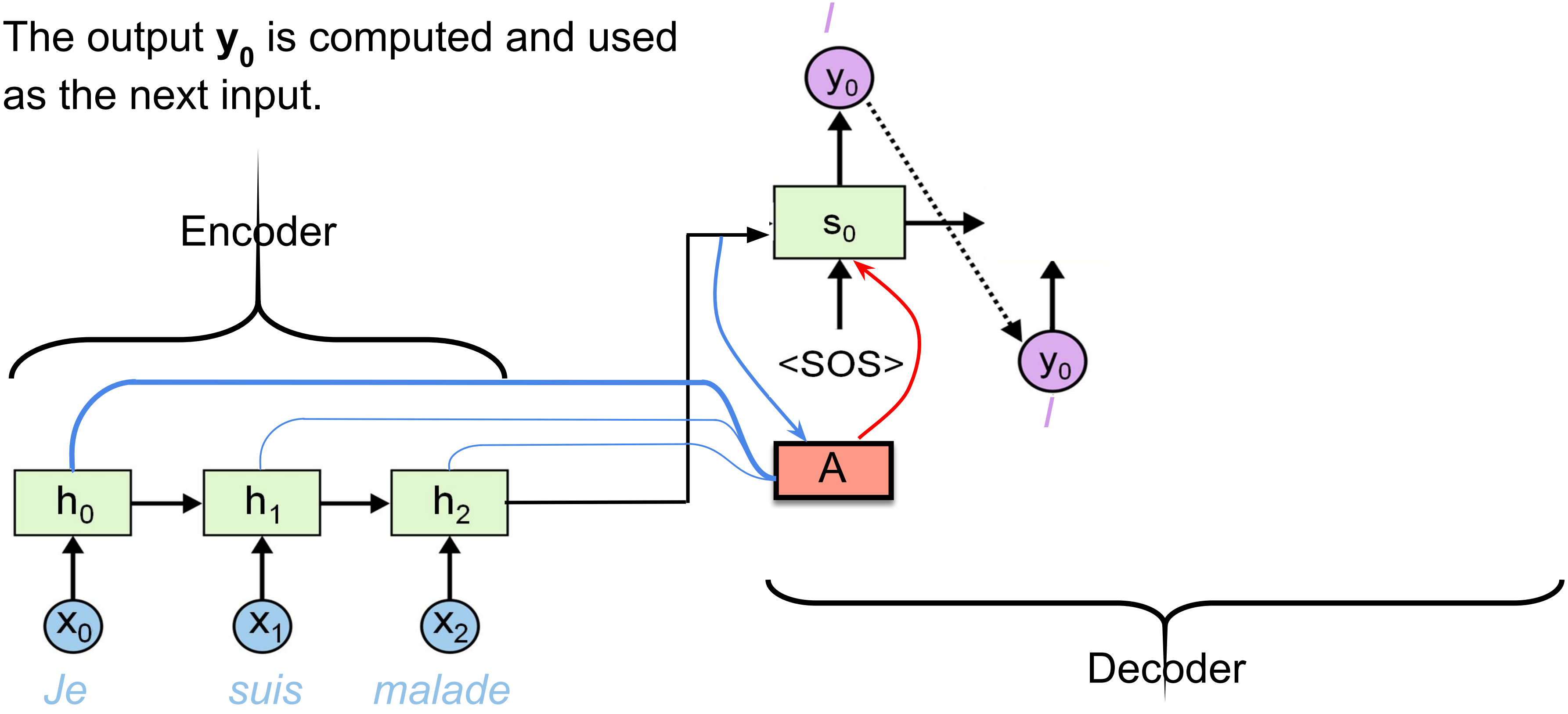
Example: Sequence-to-Sequence + Attention

The internal state $\mathbf{s}_0$ is computed.

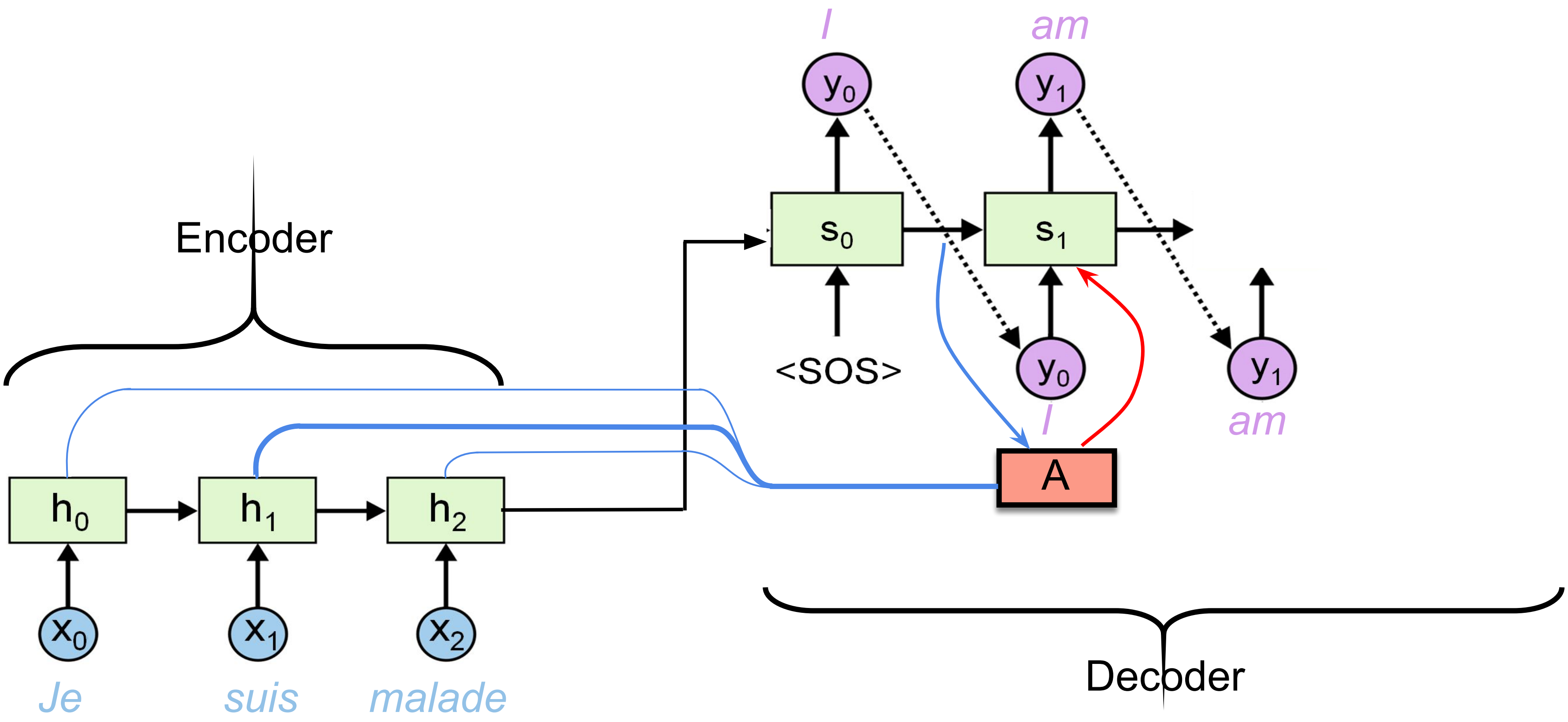


Example: Sequence-to-Sequence + Attention

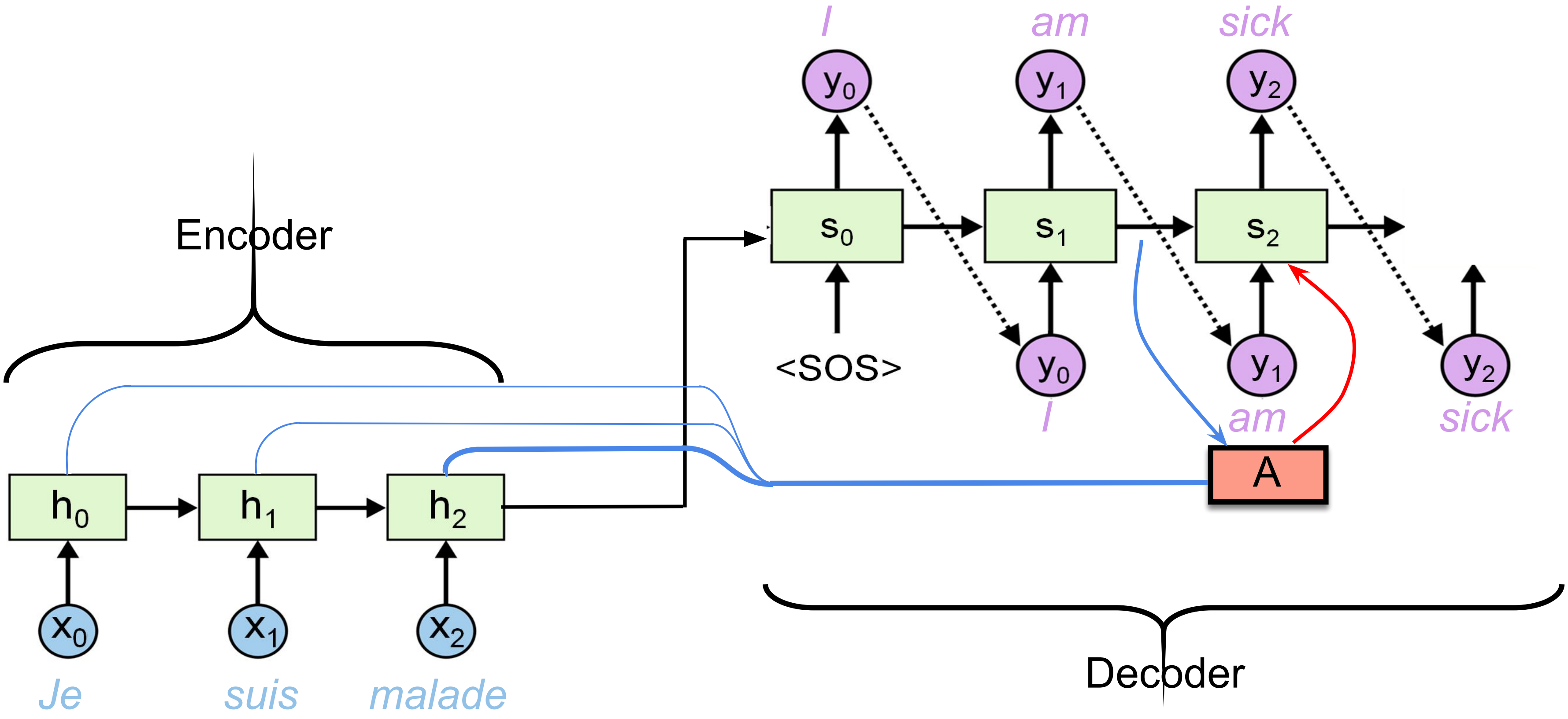
The output $\mathbf{y_0}$ is computed and used as the next input.



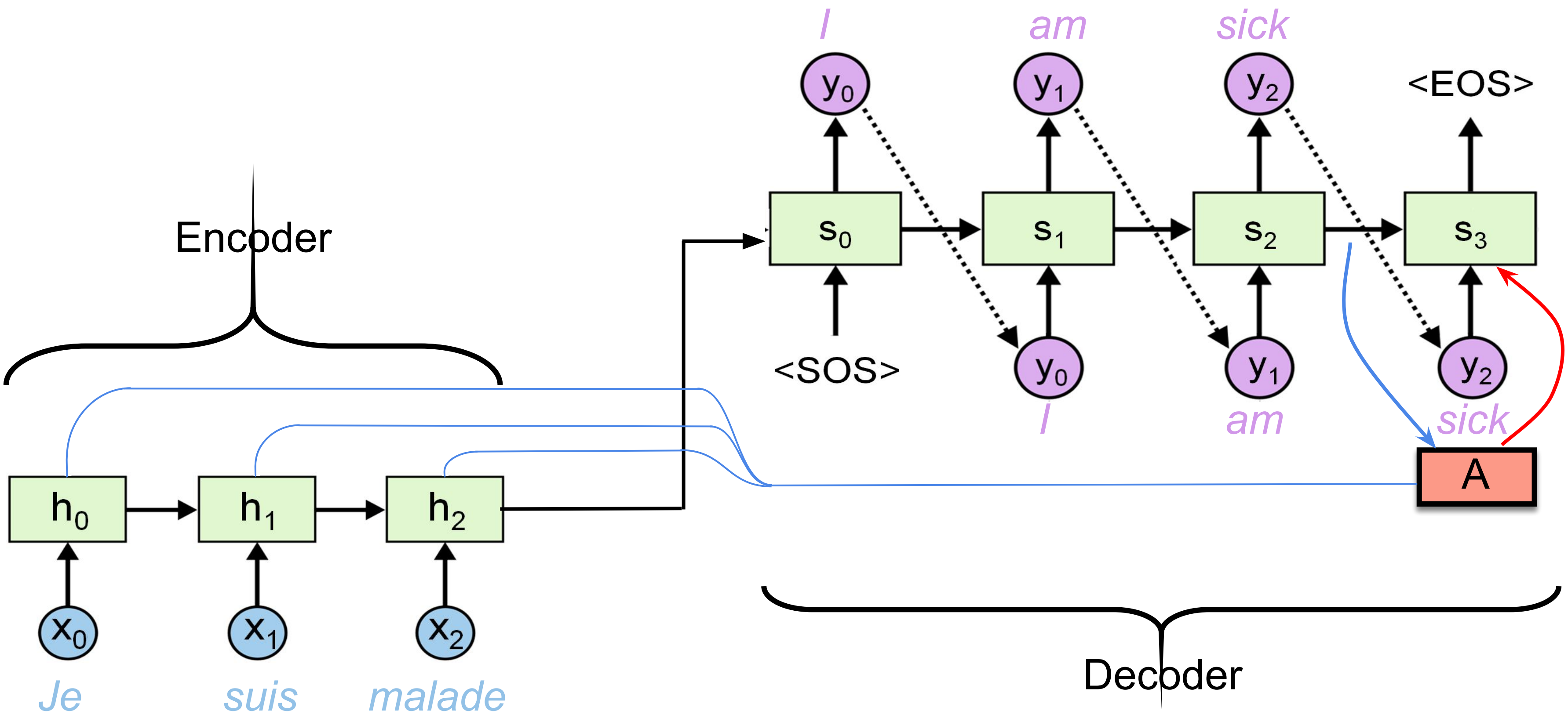
Example: Sequence-to-Sequence + Attention



Example: Sequence-to-Sequence + Attention



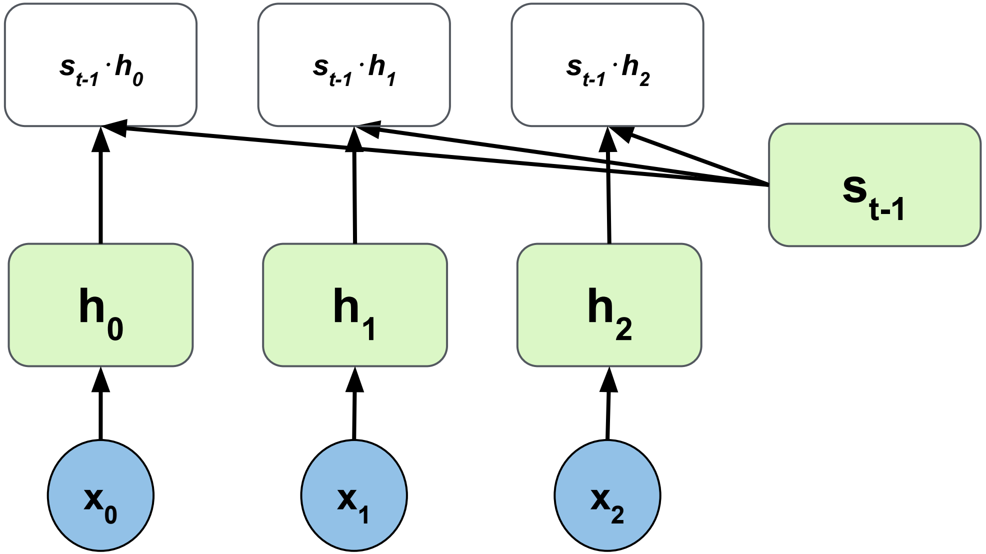
Example: Sequence-to-Sequence + Attention



Mila

Attention Function

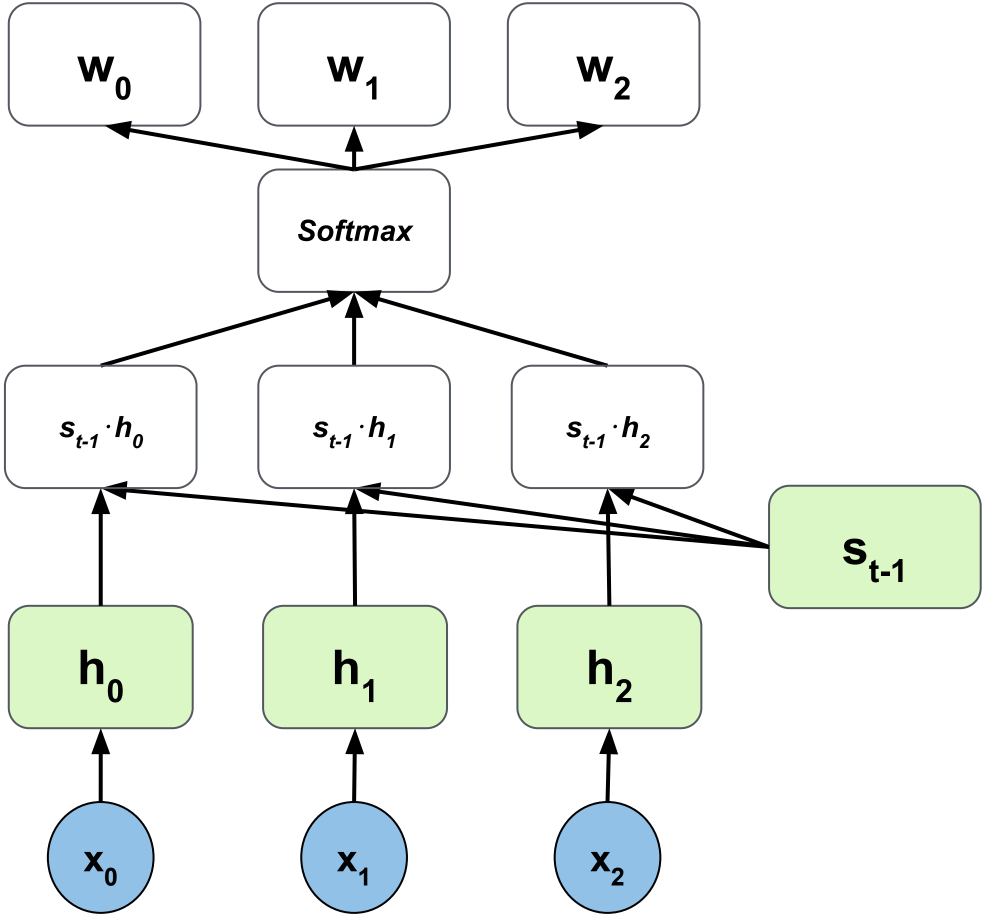
- There are several possible implementations for **A**.
- The most simple version is based on a dot product, i.e., $\mathbf{e_i = s_{t-1} \cdot h_i}$.



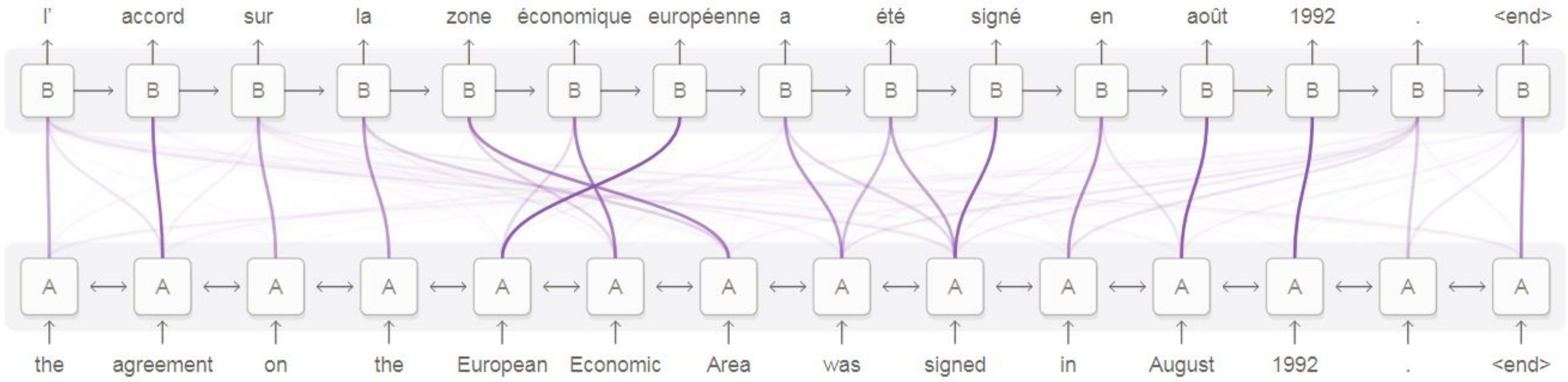
Attention Function

- The dot product results are passed through a Softmax to get normalized weights **w=[w$_0$, ..., w$_n$]**, which indicate how "important" the various elements are.
- The final result is the weighted sum of **h**.

$$c = \sum_{i=0}^{n} w_i \cdot h_i$$



Visualizing Attention

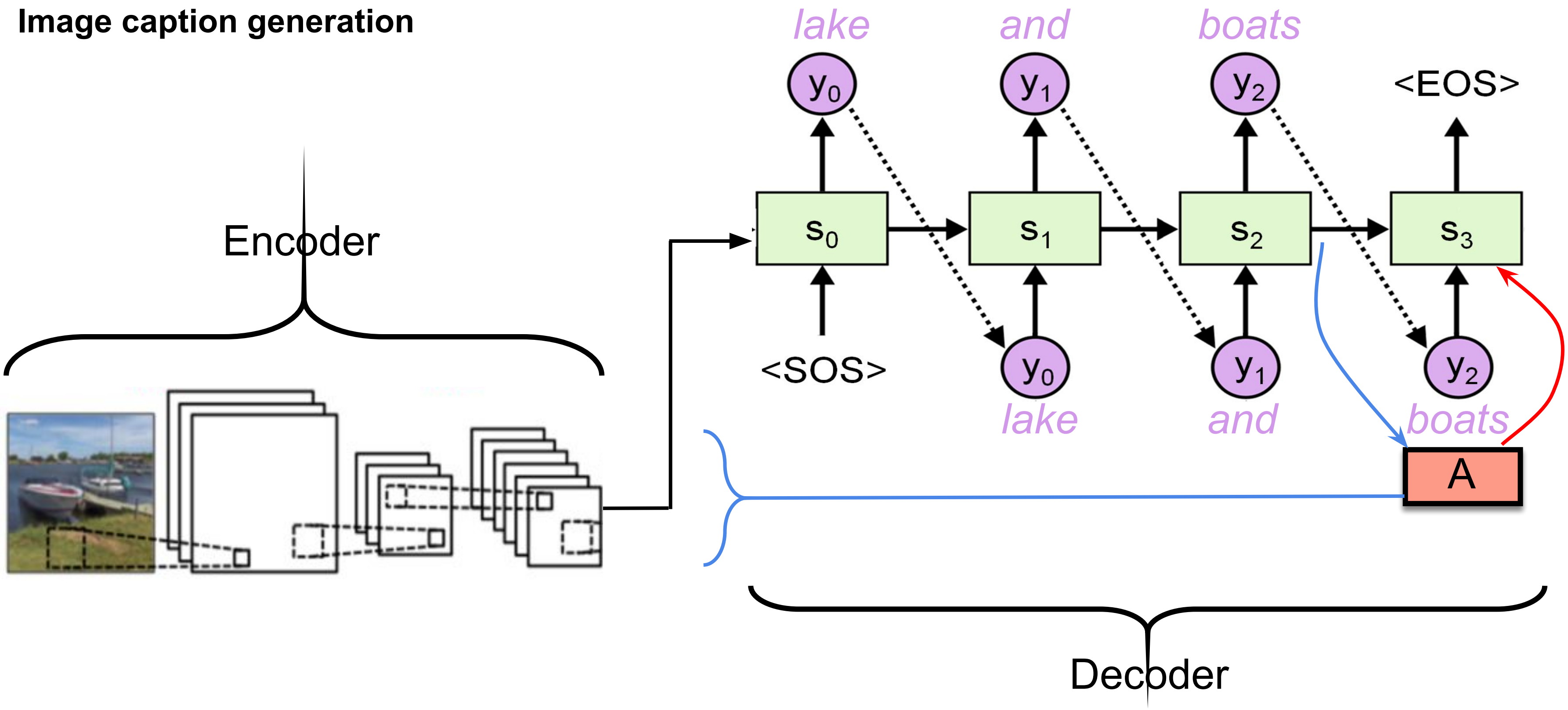


- The thick lines show where the decoder is focusing its attention when analyzing the encoded input sequence.

Image from Christopher Olah's blog

Other Examples

Image caption generation



Other Examples

A woman is throwing a frisbee in a park .

Xu et al., Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, 2015

Plan

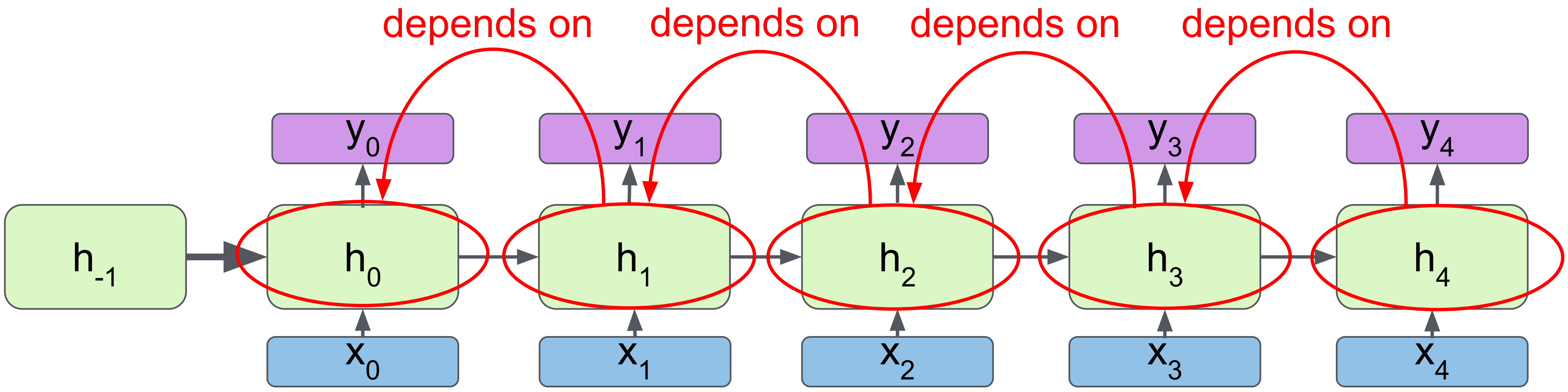
- RNN Recap
- Sequence to Sequence Models
- Attention Mechanism
- **Transformer**
- Libraries and References

Beyond RNNs

- Sequence-to-Sequence + Attention systems perform well, but they are based on RNNs (simple RNNs / LSTMs / GRUs).
- RNNs suffer from two problems:
  - Not easy to parallelize.
  - Even in the more “complex” implementations (e.g. LSTMs), they struggle to capture (very) long-term dependencies.

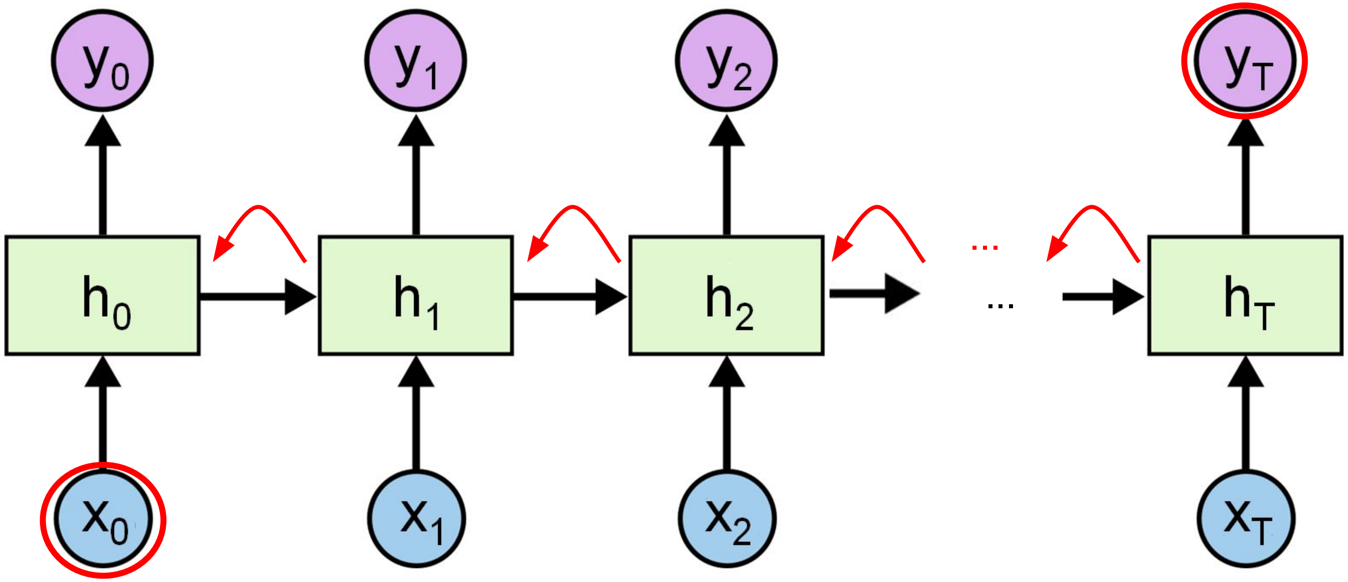
Beyond RNNs

- Every state in an RNN depends on the previous internal state.
- This creates a chain of computation which prevents parallelization.



Beyond RNNs

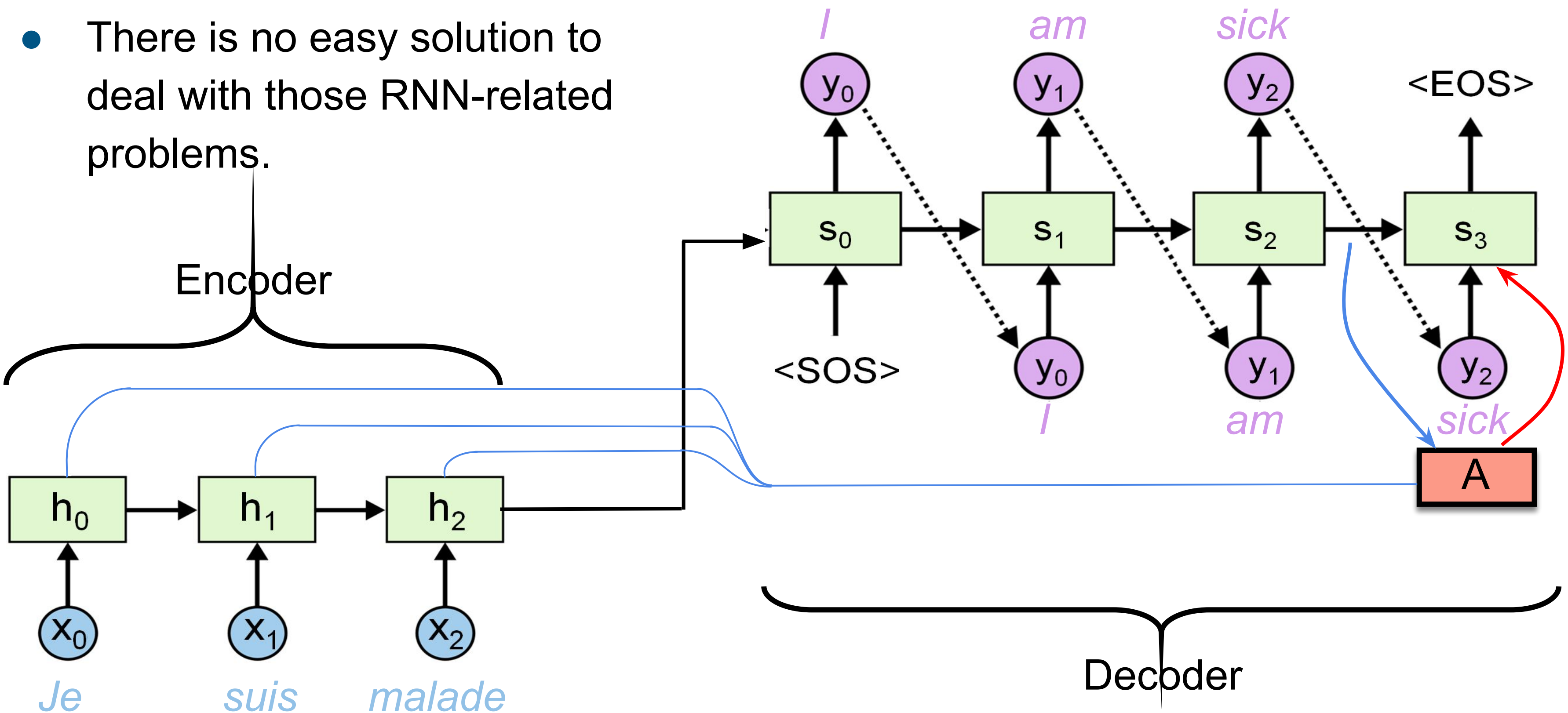
- Long-term dependencies are hard to capture with RNNs.
- This problem is strongly mitigated using LSTMs / GRUs, but it's still there for very long sequences.



Mila

Beyond RNNs

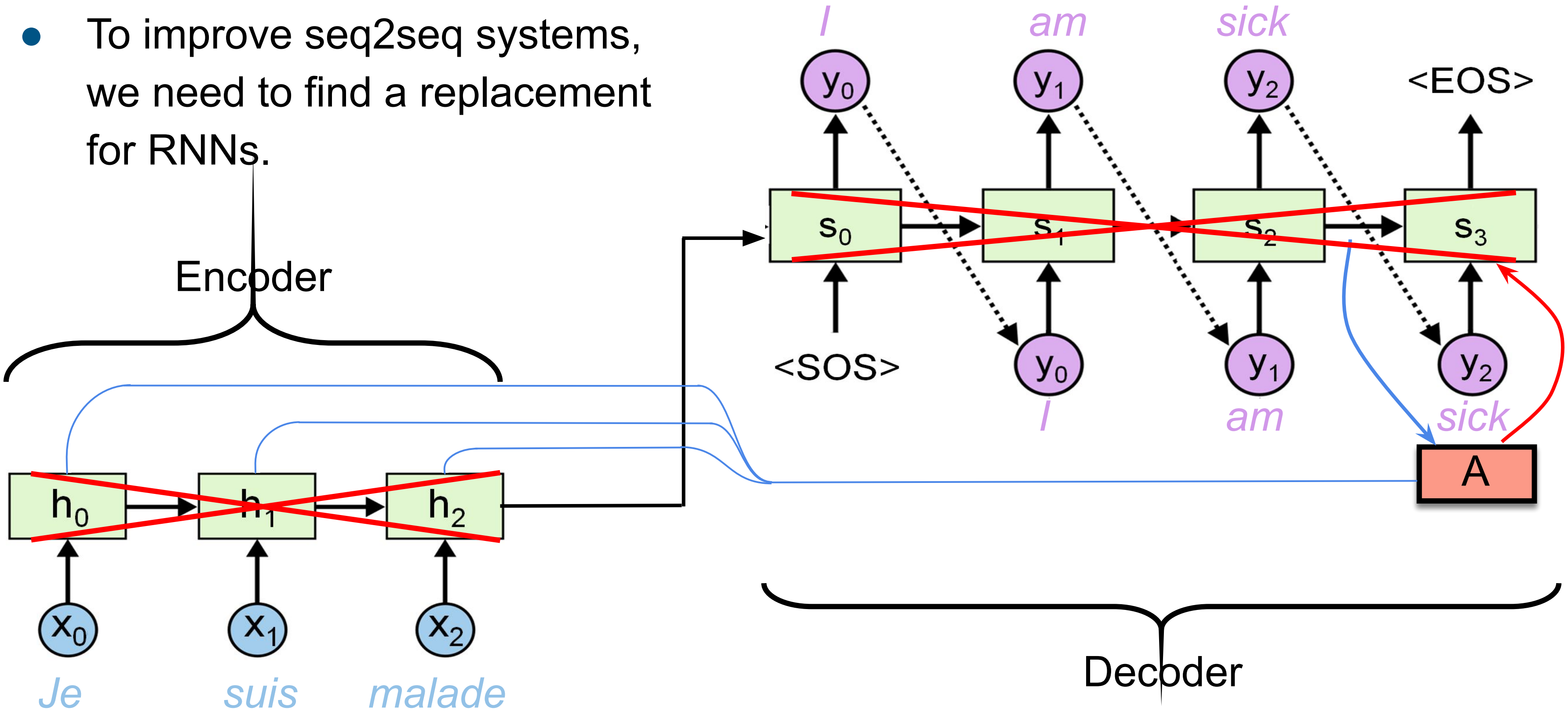
- There is no easy solution to deal with those RNN-related problems.



Mila

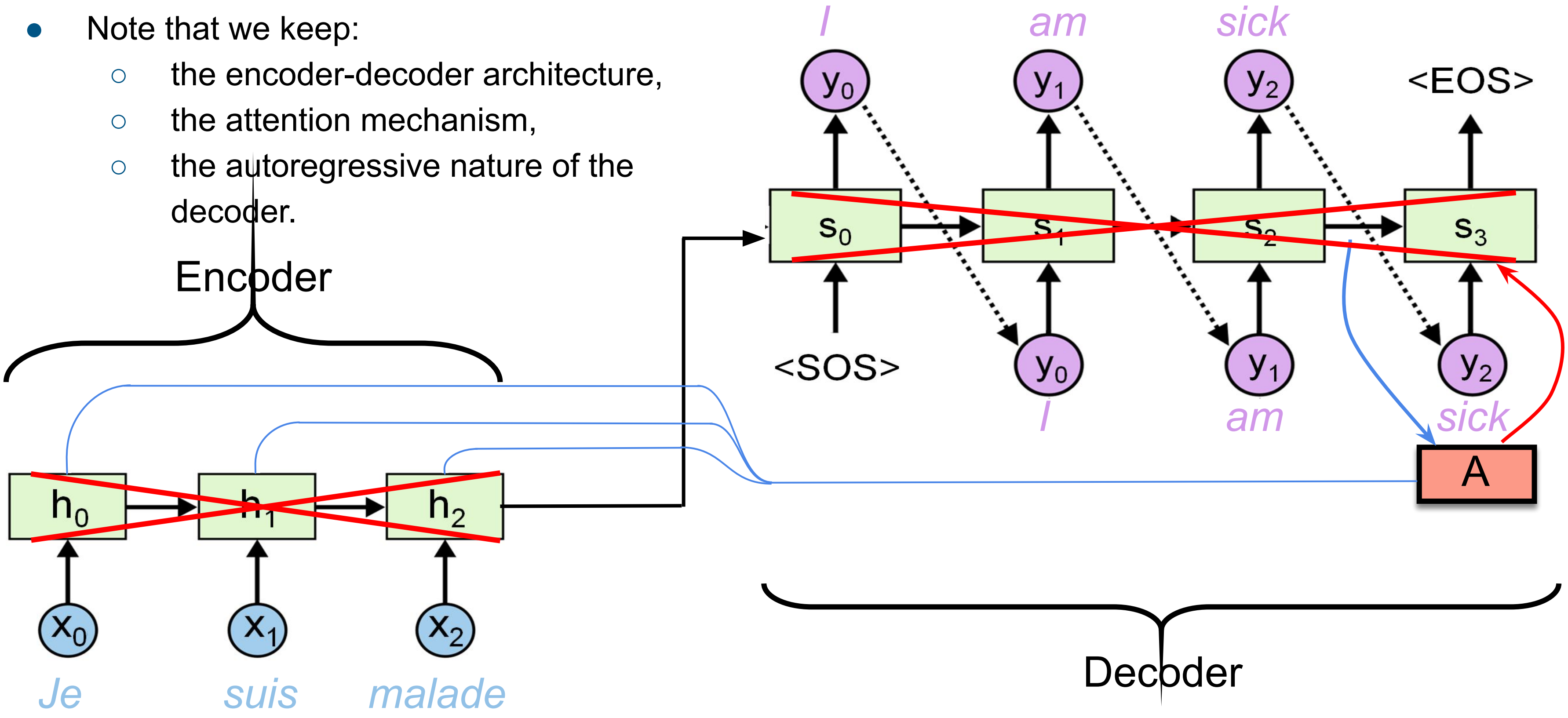
Beyond RNNs

- To improve seq2seq systems, we need to find a replacement for RNNs.



Beyond RNNs

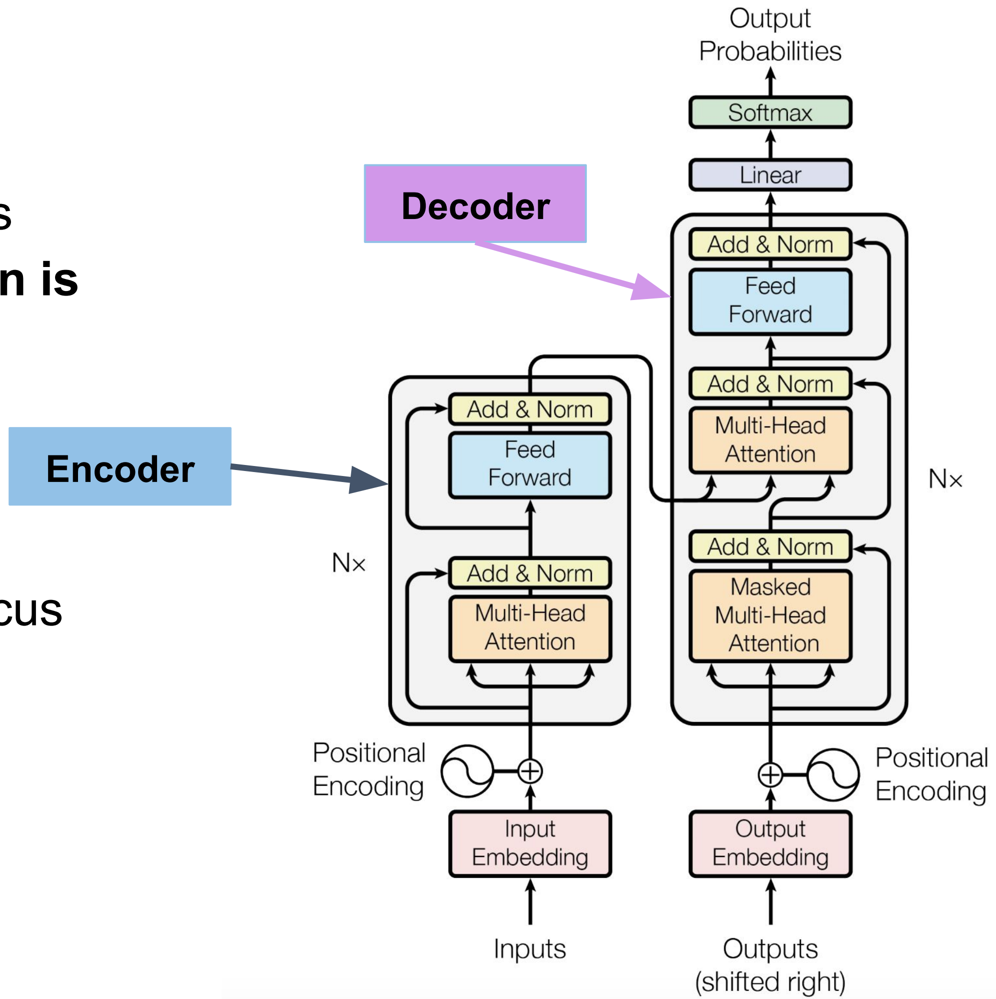
- Note that we keep:
  - the encoder-decoder architecture,
  - the attention mechanism,
  - the autoregressive nature of the decoder.



Transformer

- The Transformer architecture was introduced in the paper "**Attention is all you need**".

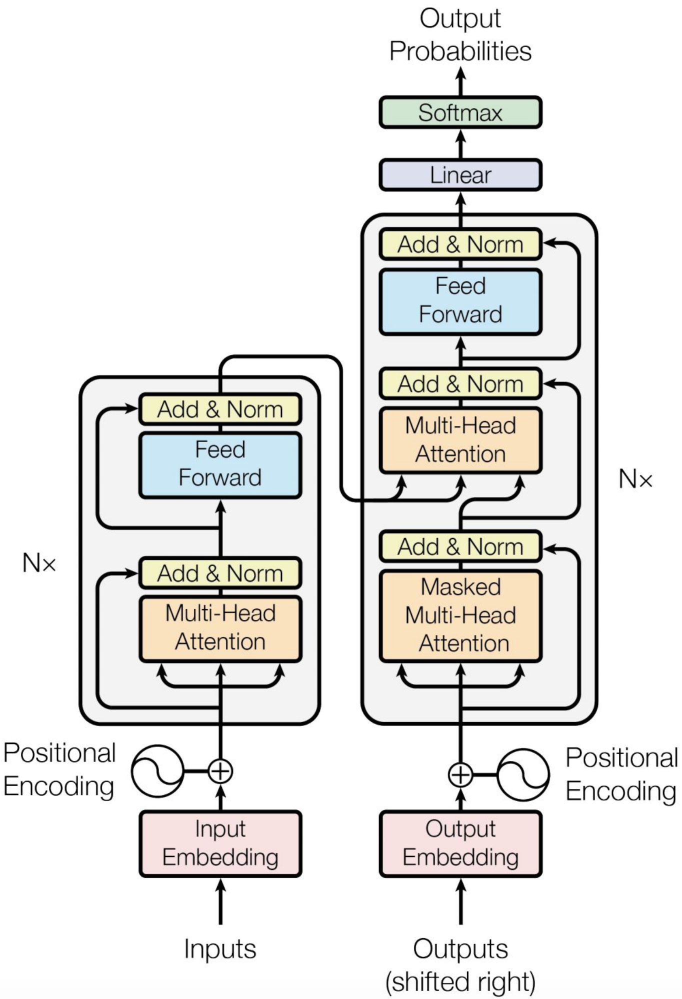
- Note: in the next slides we will focus on providing the intuition, thus simplifying some aspects of the architecture.



Vaswani et al, "Attention Is All You Need", 2017

Transformer

- Several key points:
  - Recurrence replaced with self-attention and multi-head attention.
  - Positional encodings.
  - Residual connections.
  - Layer normalization.
  - Position-wise feed-forward networks.
- We will focus on the self-attention and the multi-head attention.

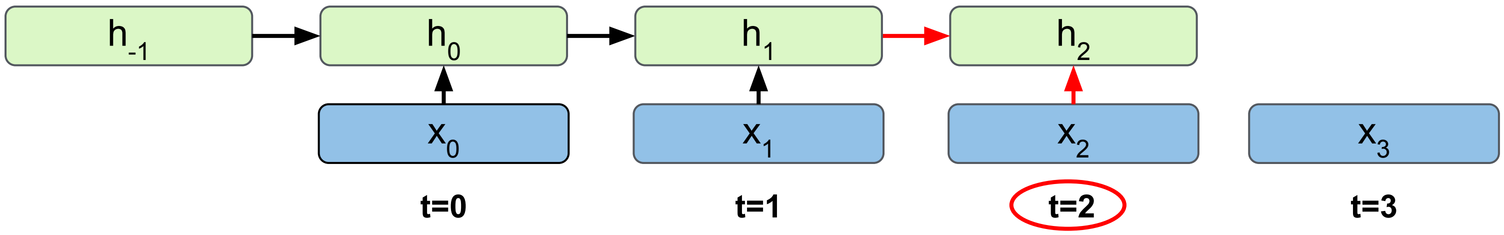


Vaswani et al, “Attention Is All You Need”, 2017

Self-Attention

- Before introducing Self-Attention, let's recap how a RNN works.
- At each time step, a RNN encodes the current input taking into consideration the past context (or the future context for right-to-left models).

- Example: the hidden state h_2 is encoding the information from the current input x_2 as well as the previous context.

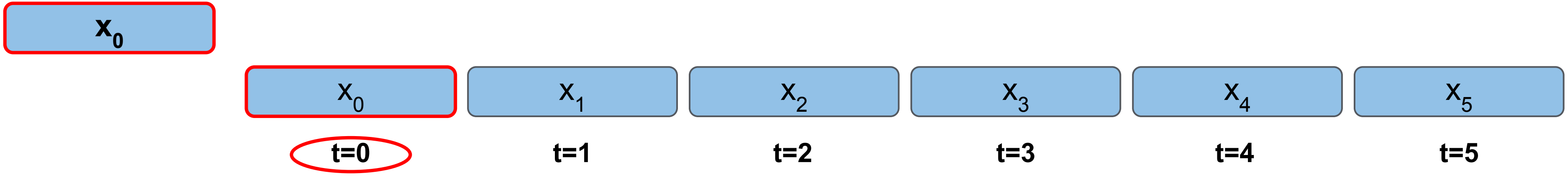


Self-Attention

- We want to do something similar with self attention:
  - encode the current input taking into consideration the surrounding context.
- The attention mechanism is used to identify the elements in a sequence which are “relevant” to encode the current one.

Self-Attention

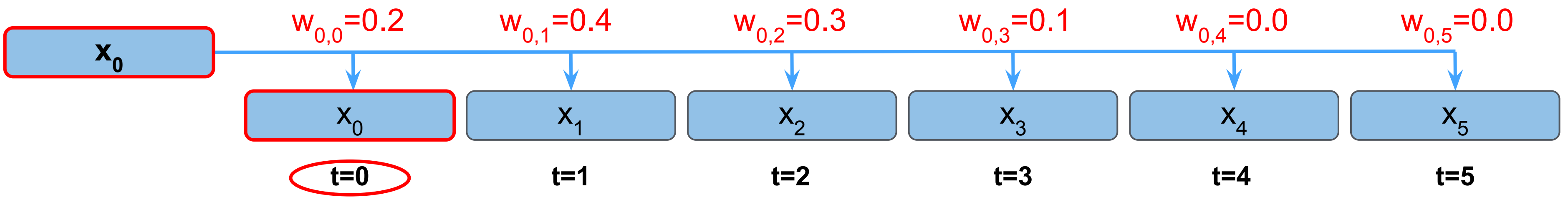
- Let's consider time step 0 with its element x_0.
- We will identify all elements in the sequence which are "relevant" to encode x_0.



Mila

Self-Attention

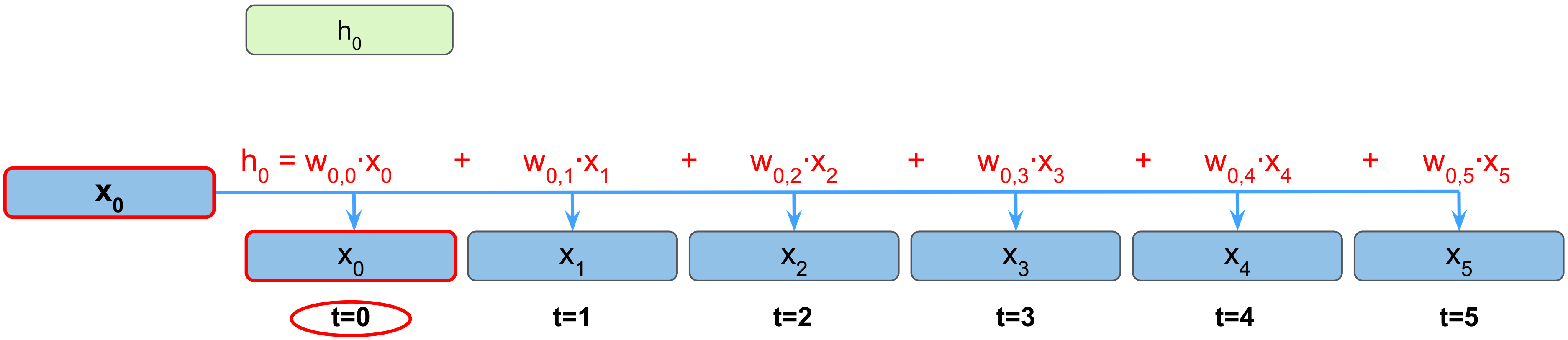
- This is done by assigning a weight to each element (by computing a dot product between the element and x_0)…



Mila

Self-Attention

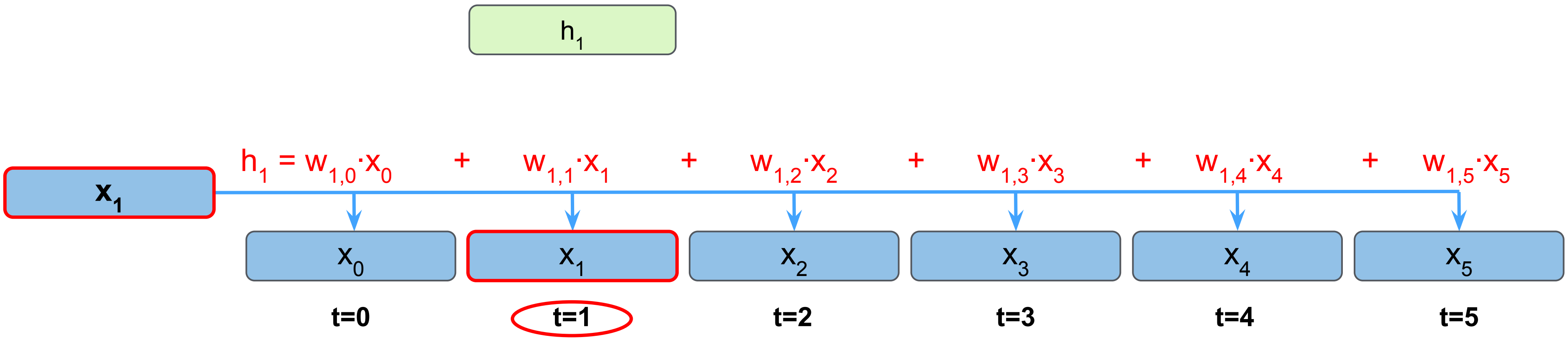
- This is done by assigning a weight to each element (by computing a dot product between the element and x0)…
- ... and then computing a normalized weighted sum.



Mila

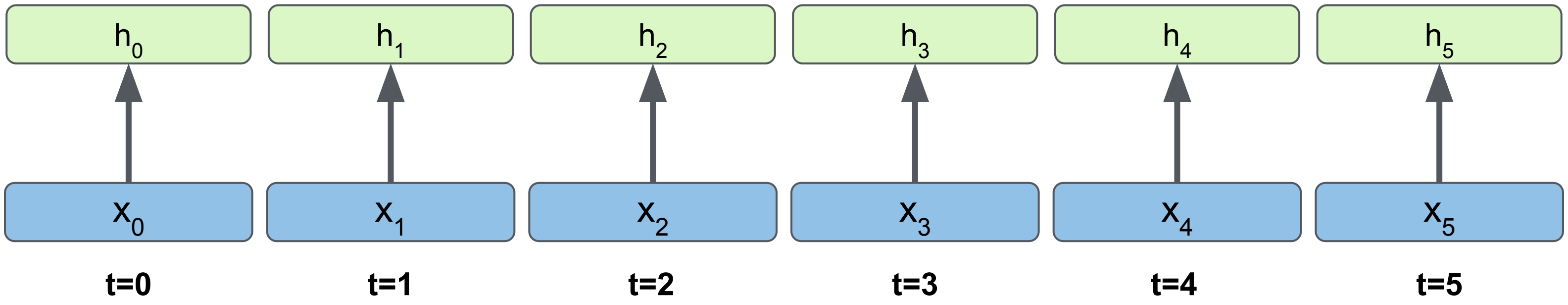
Self-Attention

- This is repeated for every step… (note that the weights vary across steps)



Self-Attention

- This is repeated for every step… (note that the weights vary across steps)
- … until all the steps are completed.



Self-Attention - Multiple Heads

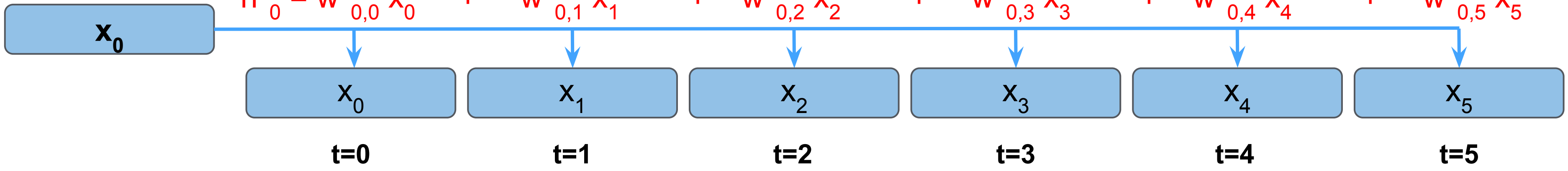
- The self-attention is meant to identify all elements in a sequence which are "relevant" to encode x_t.
- Given that there can be several types of relevant information, we can have several attention mechanisms.

Self-Attention - Multiple Heads

- The self-attention is meant to identify all elements in a sequence which are “relevant” to encode x_t.
- Given that there can be several types of relevant information, we can have several attention mechanisms.
- Each attention mechanism is called a **head**, leading to a **multi-head self-attention**.
    - In this example, there are head#0 and head#1, each with different weights.

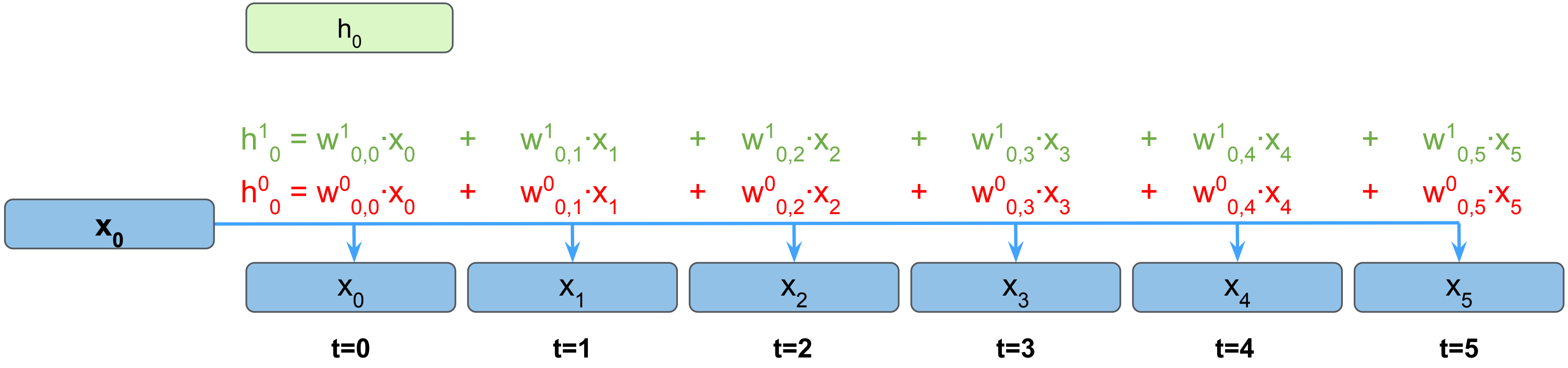
$$h^1_0 = w^1_{0,0} \cdot x_0 + w^1_{0,1} \cdot x_1 + w^1_{0,2} \cdot x_2 + w^1_{0,3} \cdot x_3 + w^1_{0,4} \cdot x_4 + w^1_{0,5} \cdot x_5$$

$$h^0_0 = w^0_{0,0} \cdot x_0 + w^0_{0,1} \cdot x_1 + w^0_{0,2} \cdot x_2 + w^0_{0,3} \cdot x_3 + w^0_{0,4} \cdot x_4 + w^0_{0,5} \cdot x_5$$



Self-Attention - Multiple Heads

- The various heads are then merged together.
  - E.g., they are concatenated, $\mathbf{h_0 = [h^0_0, h^1_0]}$.

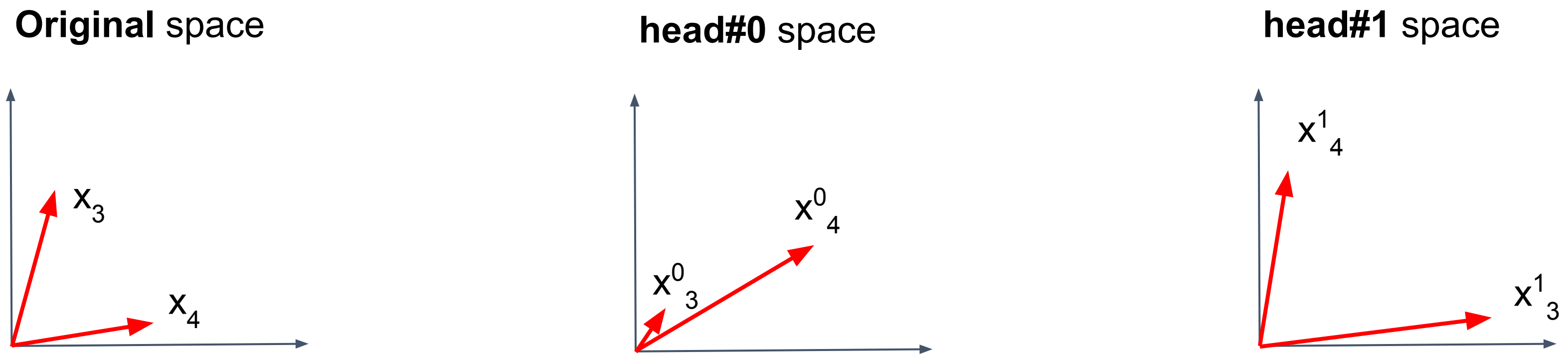


Mila

Self-Attention - Multiple Heads

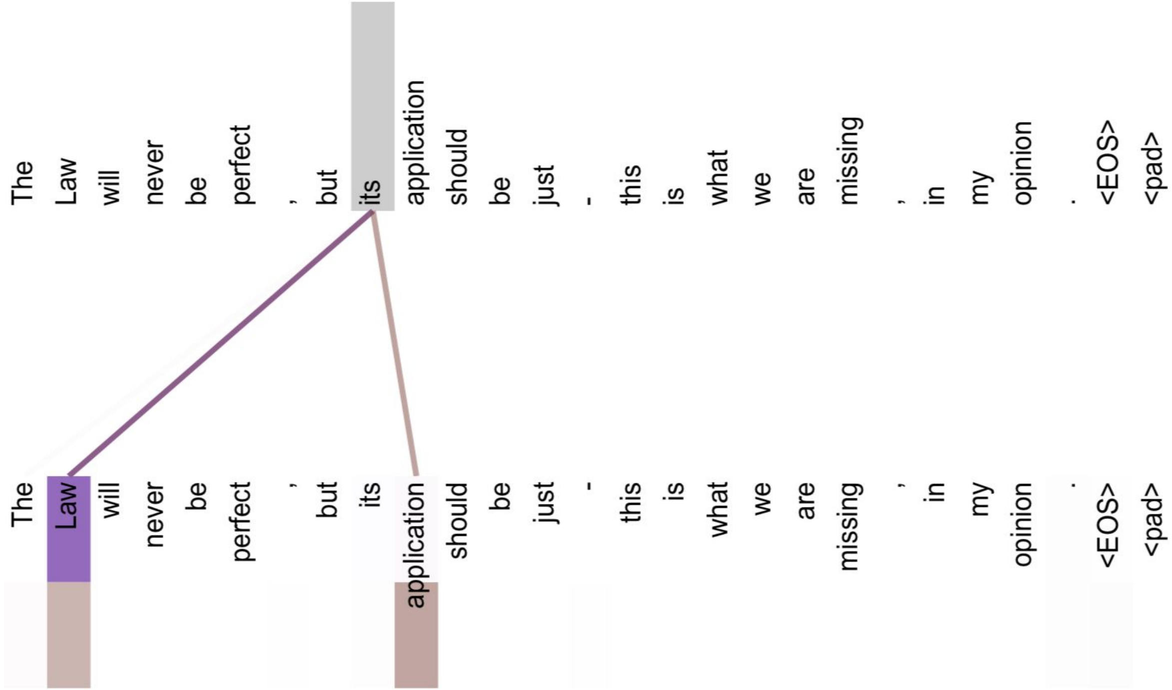
- The weights for a given head are based on a dot-product.
  - E.g., $\mathbf{w^0_{3,4} = x^0_3 \cdot x^0_4}$

- The dot-product is computed in a different space for each attention head. This space is obtained by learning a projection from the original space to the one dedicated to a particular head.



Mila

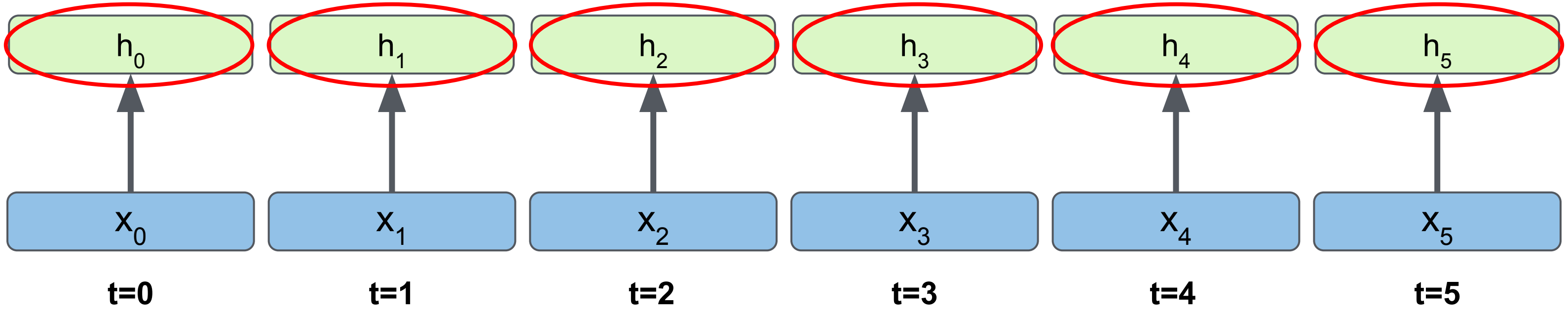
Self-Attention - Visualization



Vaswani et al, “Attention Is All You Need”, 2017

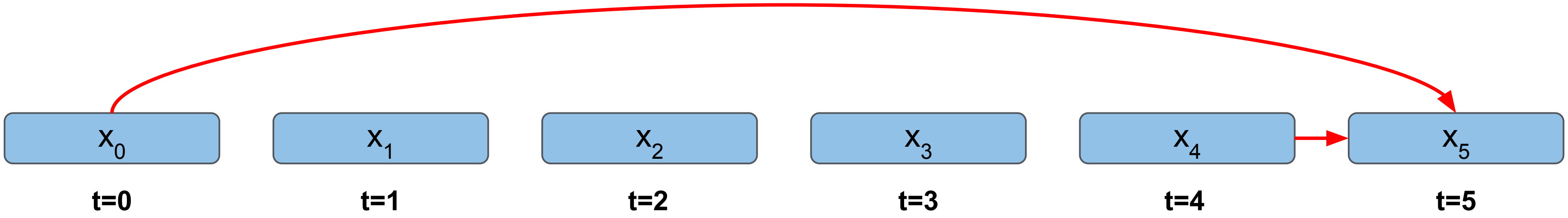
Self-Attention - Advantages

- The multi-head self-attention can be computed in parallel at all time steps.
- There are no dependencies between time steps.



Self-Attention - Advantages

- The RNN chain of computation is **not** there anymore.
- The information does **not** need to flow over a long chain of elements.
- E.g., $\mathbf{x_5}$ has direct access to both $\mathbf{x_0}$ and $\mathbf{x_4}$.



Plan

- RNN Recap
- Sequence to Sequence models
- Attention Mechanism
- Transformer
- **Libraries and References**

Libraries

- RNNs are included in the main DL frameworks:
  - PyTorch : https://pytorch.org/docs/stable/nn.html#recurrent-layers
  - Tensorflow: https://www.tensorflow.org/tutorials/recurrent
- There are several Transformer implementations:
  - in Tensorflow: https://github.com/tensorflow/tensor2tensor
  - in PyTorch: https://github.com/huggingface/pytorch-transformers

References

- Christopher Olah's blog about LSTMs: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Christopher Olah's publications on the attention mechanism: https://distill.pub/2016/augmented-rnns/
- Andrej Karpathy's blog about RNNs: http://karpathy.github.io/2015/05/21/rnn-effectiveness/
- The Deep Learning Book (Goodfellow et al.): http://www.deeplearningbook.org/

Quebec
Artificial
Intelligence
Institute

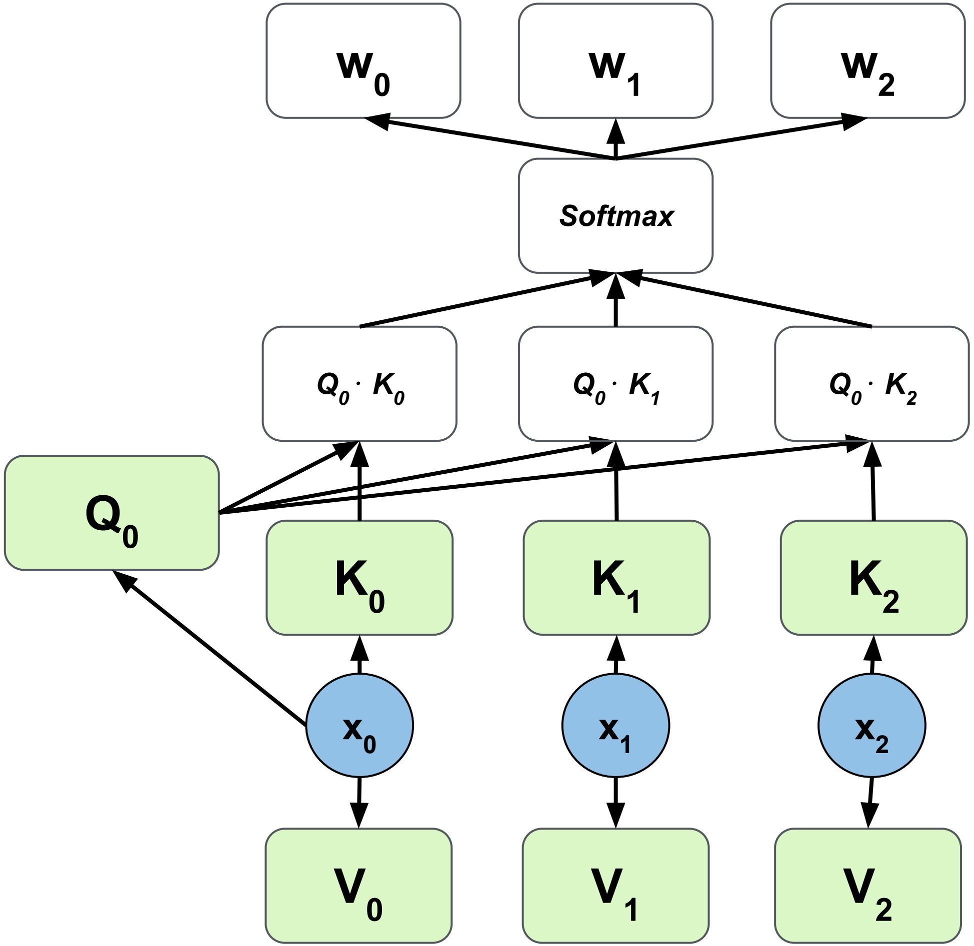
Mila

Questions?

Self-Attention

- In reality, every input (x) has 3 different “views”: Q, K, V.
  - Q: query - used to represent the current state (called **s** before).
  - K: keys - used for the dot-product (to compute the weights - called **h** before).
  - V: values - used in the final weighted sum.
- In this example, we are focusing on x_0 .

$$x'_0 = \sum_{i=0}^{n} w_i \cdot V_i$$



Mila

Decoder Probabilities

- Want to maximize:

$$p(y_1, y_2, \ldots, y_n | x)$$

- We can decompose as:

$$p(y_1, y_2, \ldots, y_n | x) = p(y_1 | x)p(y_2, \ldots, y_n | x, y_1)$$

and again:

$$p(y_1, y_2, \ldots, y_n | x) = p(y_1 | x)p(y_2 | x, y_1)p(y_3, \ldots, y_n | x, y_1, y_2)$$

until:

$$p(y_1, y_2, \ldots, y_n | x) = p(y_1 | x)p(y_2 | x, y_1) \ldots p(y_n | x, y_1, \ldots y_{n-1})$$

Results - Seq2Seq W/ and W/O Attention

Model	Max sequence length	BLEU score
Seq2seq	30	13.9
Seq2seq + attention	30	**21.5**
Seq2seq	50	17.8
Seq2seq + attention	50	**26.7**

- The BLEU score (BiLingual Evaluation Understudy) measures the quality of the translation (the higher the score the better).
  - BLEU is a modified form of the Precision metric based on the overlap between output and target.
- Attention improves the score significantly.

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

Results - Transformer

- Better results than previous systems (left column)...
- … with a fraction of the training cost (right column).

Model	BLEU		Training cost (FLOP)	
	EN-DE	EN_FR	EN-DE	EN_FR
GNMT (RNNs)	24.6	39.92	$2.3*10^{19}$	$1.4*10^{20}$
Transformer (base)	27.3	38.1	$3.3*10^{18}$	
Transformer (big)	28.4	41.0	$2.3*10^{19}$	

- BLEU score (BiLingual Evaluation Understudy): the higher the score the better.

Self-Attention - Word Order

- The (weighted) sum leads to the loss of the elements' order.
  - E.g., $w_0 \cdot x_0 + w_1 \cdot x_1 = w_1 \cdot x_1 + w_0 \cdot x_0$
- Solution: attach the position information to every word.
  - E.g., $x'_0 = [x_0 + p_0]$, $x'_1 = [x_1 + p_1]$
- Now the order of the elements has an impact on the results:

$$w_0 \cdot [x_0 + p_0] + w_1 \cdot [x_1 + p_1] \neq w_1 \cdot [x_1 + p_0] + w_0 \cdot [x_0 + p_1]$$

Mila