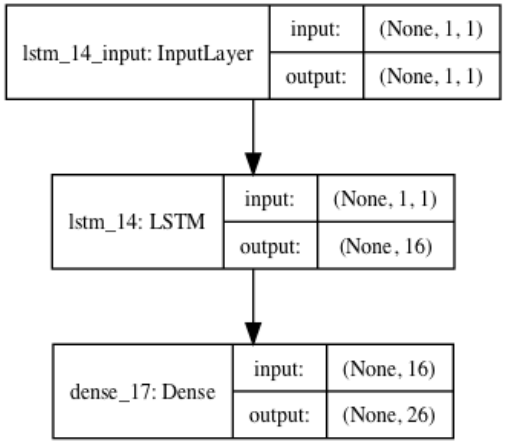
More on RNN

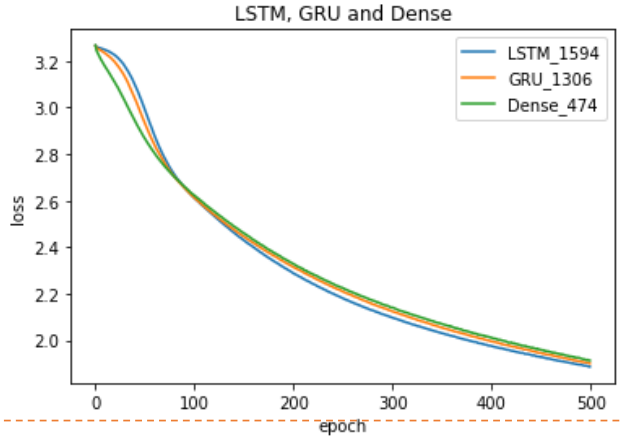

Stateless and stateful
Recurrent Neural Network

Try to predict a series

- Approach 1: without context
  - With shuffle the input
    - ['A'] -> B
    - ['B'] -> C
    - ...
    - ['Z'] -> A

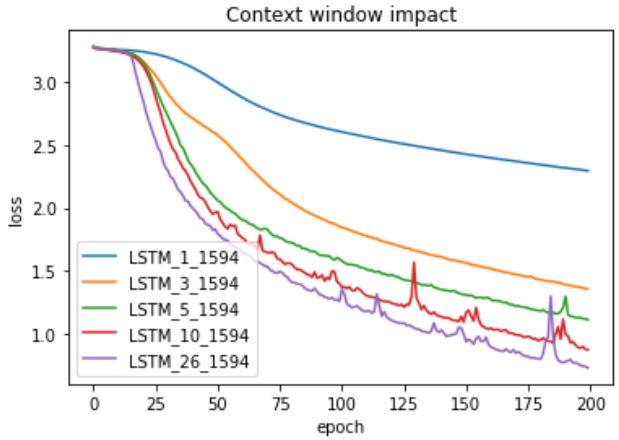


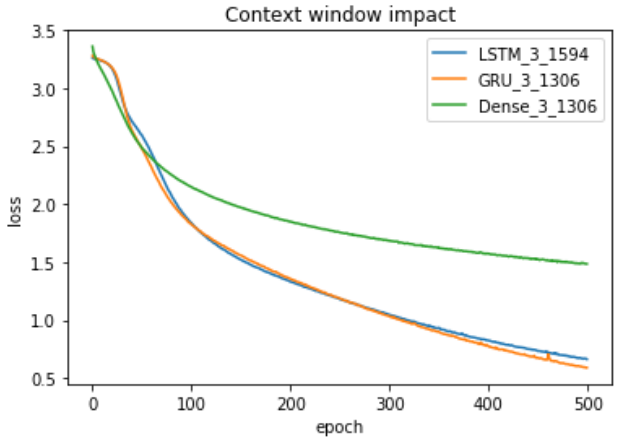
- No difference between
  - LSTM
  - GRU
  - And Dense



Try to predict a series

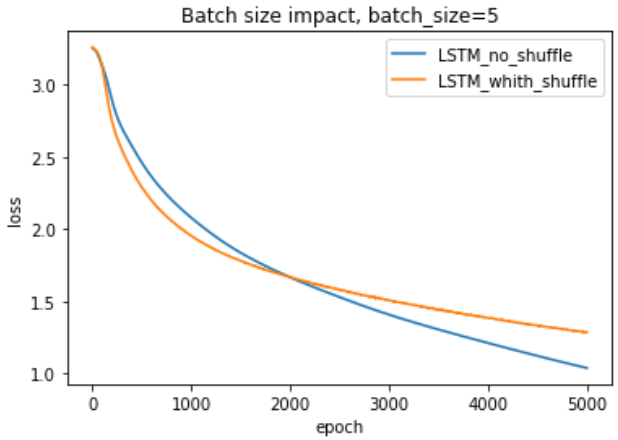
- Approach 2: with a context
  - With shuffle the input
    - ['A', 'B', 'C'] -> D
    - ['B', 'C', 'D'] -> E
    - ...
    - ['Z', 'A', 'B'] -> C
- For LSTM and GRU
  - When a context increase
    - Performance is better
    - Constant number of parameters
- For Dense
  - When a context increase
    - Performance is better
    - But number of parameters increase
      - Ctx=3 = 1306 parameters
      - Ctx 10 = 4218 parameters
      - Ctx 26 = 10_874 parameters

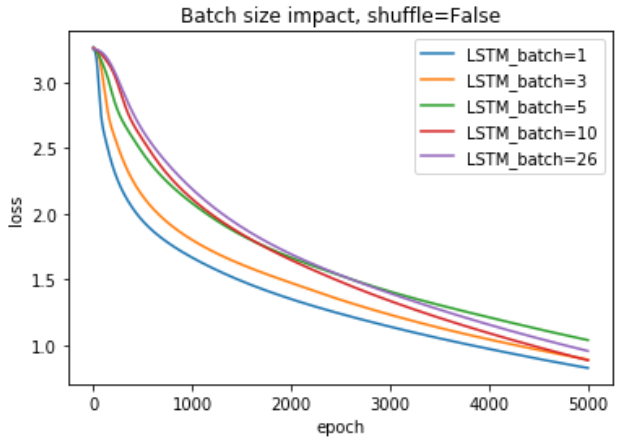




Try to predict a series

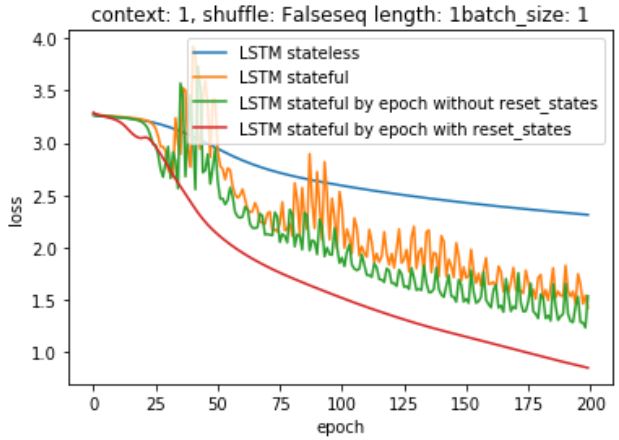
- Approach 3: no context but use a batch as context information
  - It's not really conclusive
    - Batch size = 1 better than the full dataset
    - Small impact with shuffle

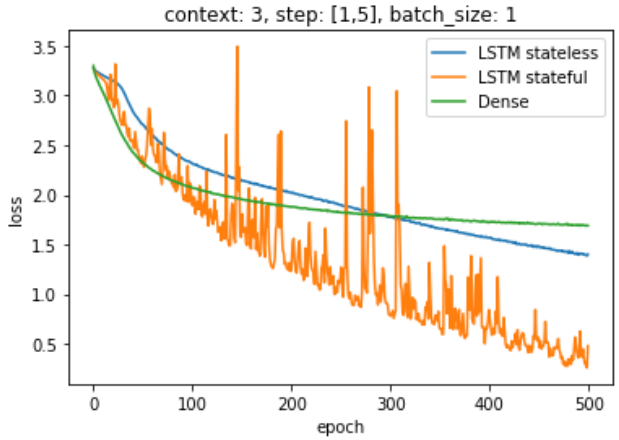




Try to predict a series

- Approach 4: use statefull LSTM
  - No context
  - Batch_size = 1
  - Shuffle = False





What have we learned

- For predicting a serie
  - a context reduces learning time
  - LSTM or GRU reduce the number of parameters
- if the entries are independent:
  - Use stateless LSTM or GRU
  - Shuffle = True
  - Batch size = ?
- if the series is described by several inputs
  - Use stateful LSTMs
  - Shuffle = False
  - reset hidden state when changing context (new documents for example for NER)

Seq2seq model

Seq2Seq model

- Goal:
  - Develop an architecture capable of generating contextually appropriate, arbitrary length, output sequences

- Applications:
  - Machine translation,
  - Summarization,
  - Question answering,
  - Dialogue modeling.

Simple recurrent neural network illustrated as a feed-forward network

- **Most significant change: new set of weights, U**
  - connect the hidden layer from the previous time step to the current hidden layer.
  - determine how the network should make use of past context in calculating the output for the current input.

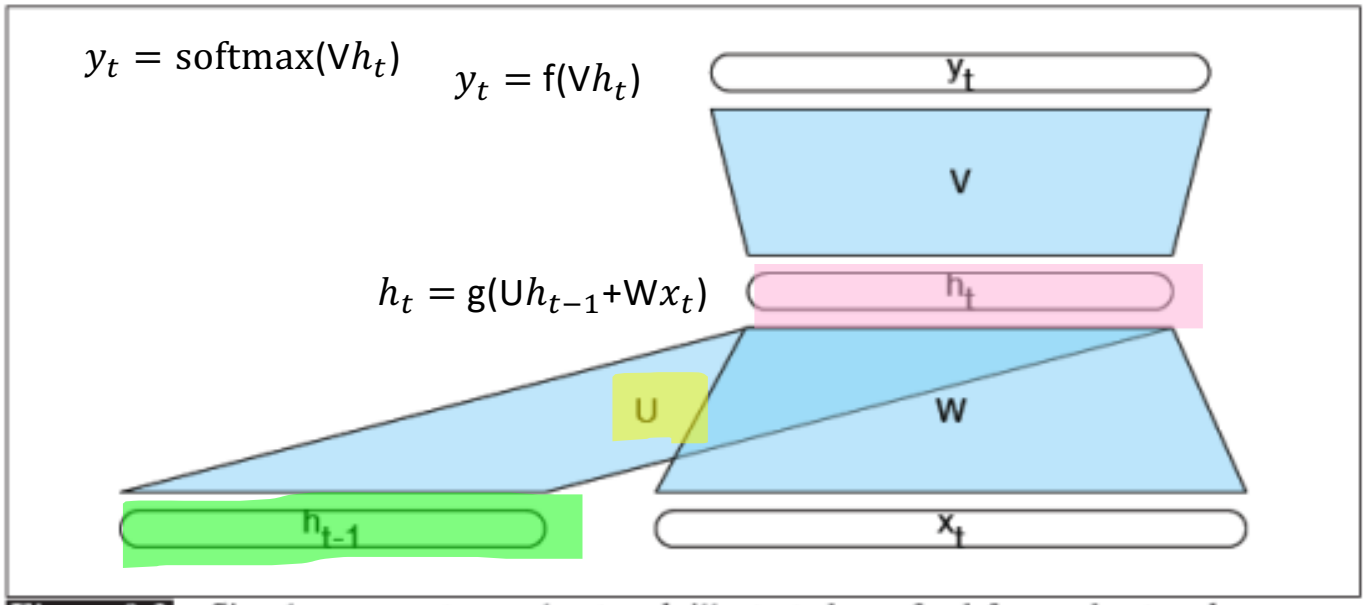
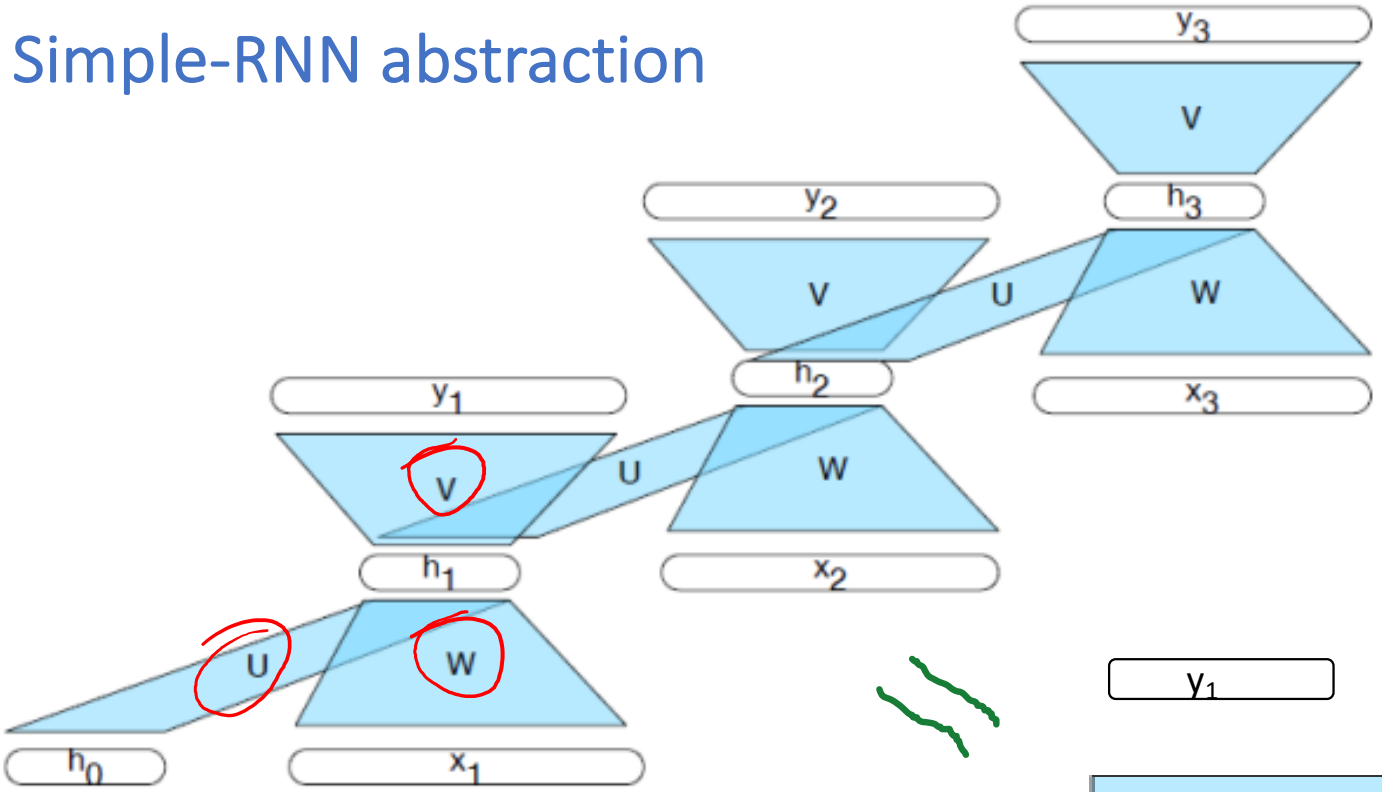
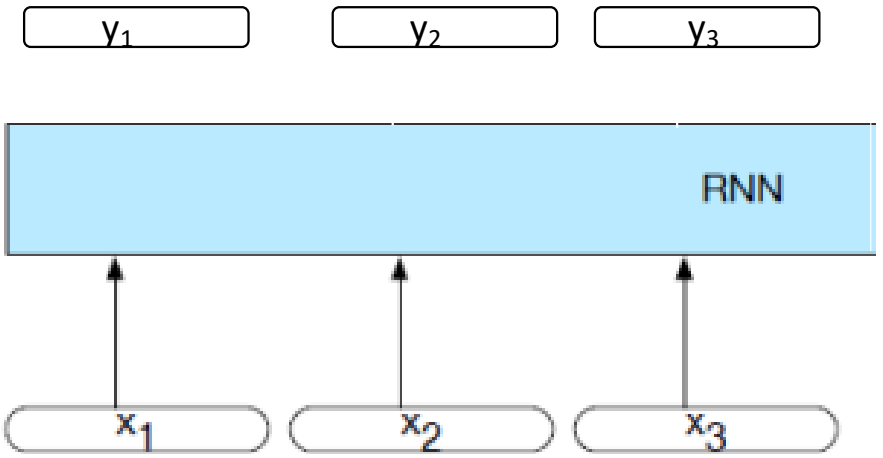


Figure 9.3 Simple recurrent neural network illustrated as a feed-forward network.

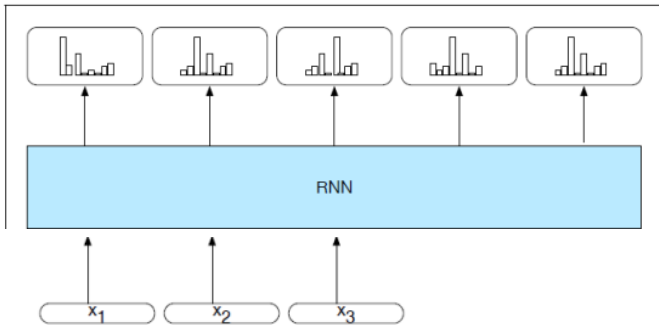
Simple-RNN abstraction



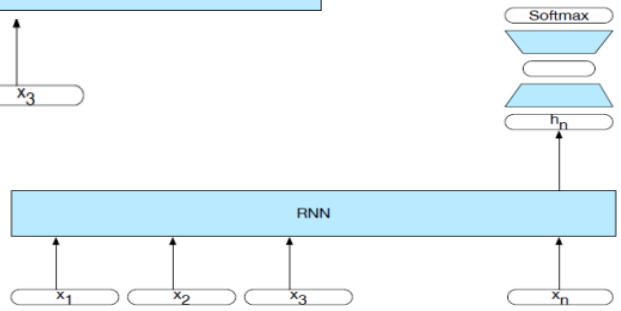


RNN Applications

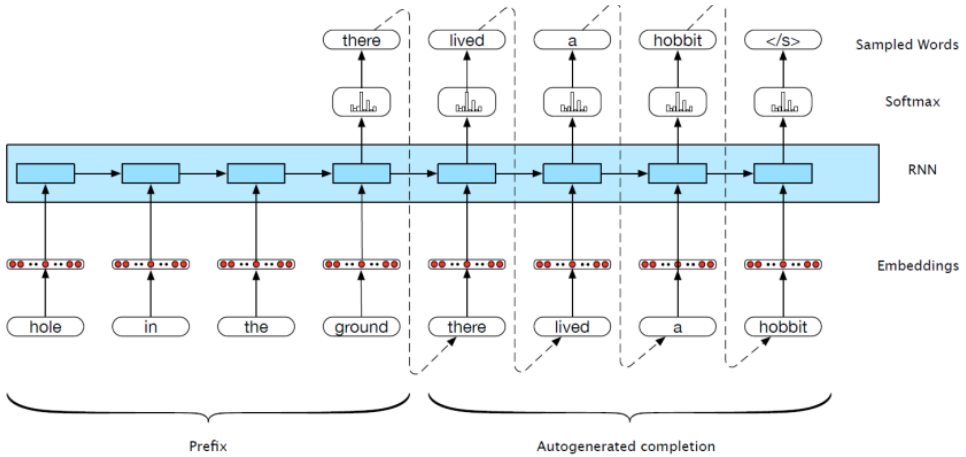
- Language Modeling



- Sequence Classification (Sentiment, Topic)

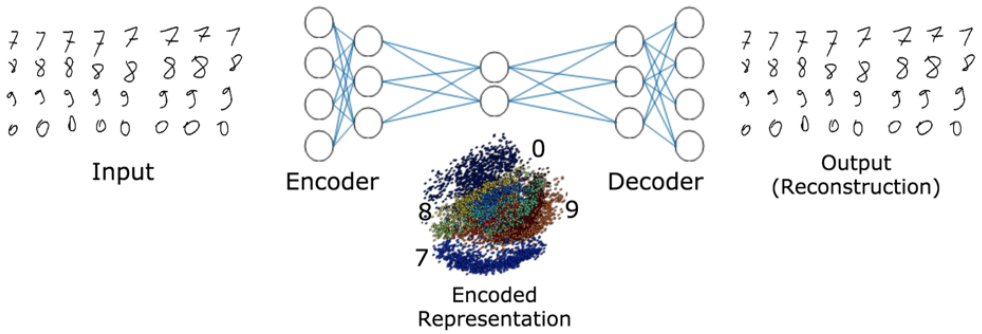


- Sequence to Sequence

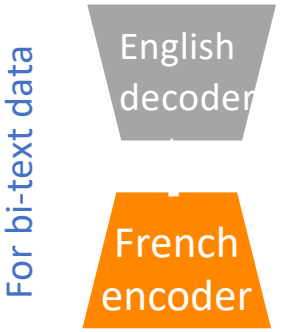


Seq2seq model

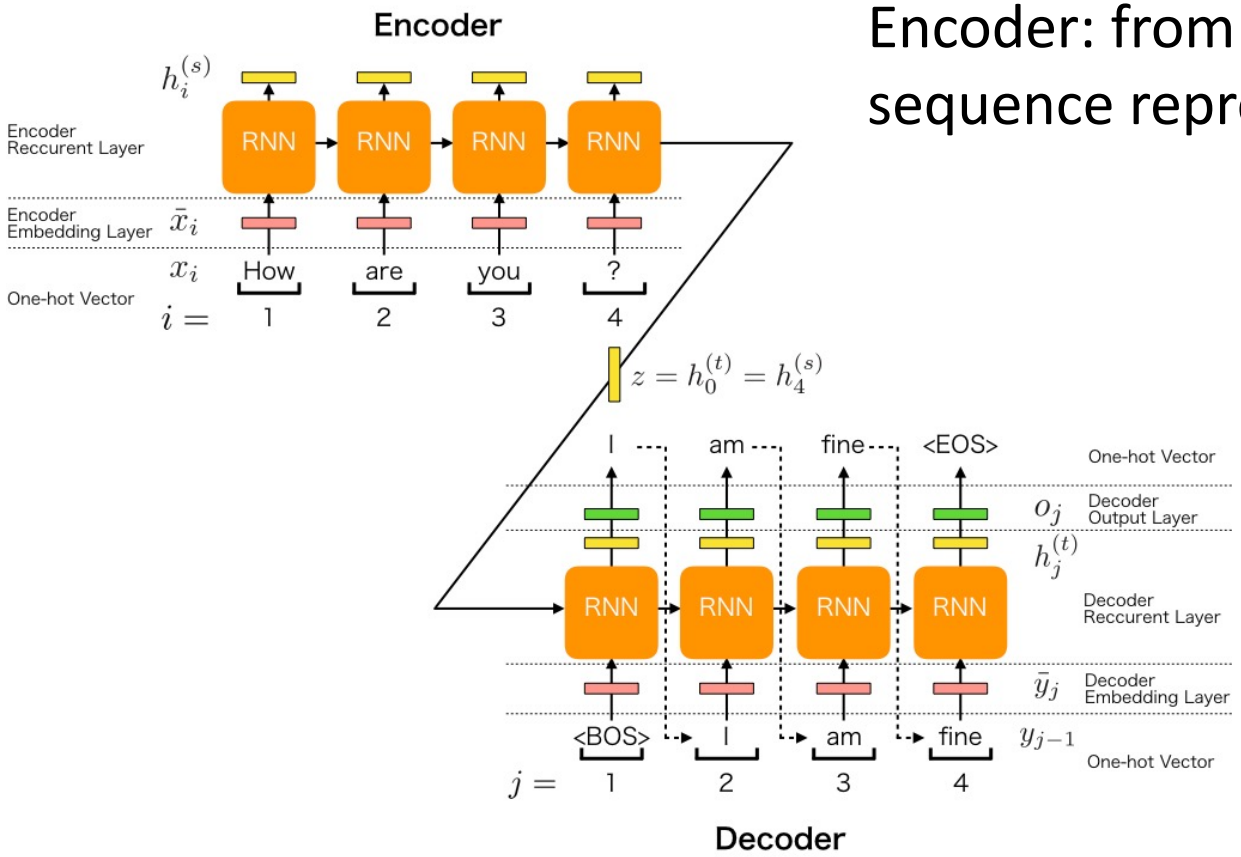
- A generalisation of auto-encoder to recurrent network
  - Traditionnal auto-encoder



  - Seq2Seq model
    - Encoder: from word sequence to sequence representation



Seq2Seq model



Encoder: from word sequence to sequence representation

Decoder: from sequence representation to text generation

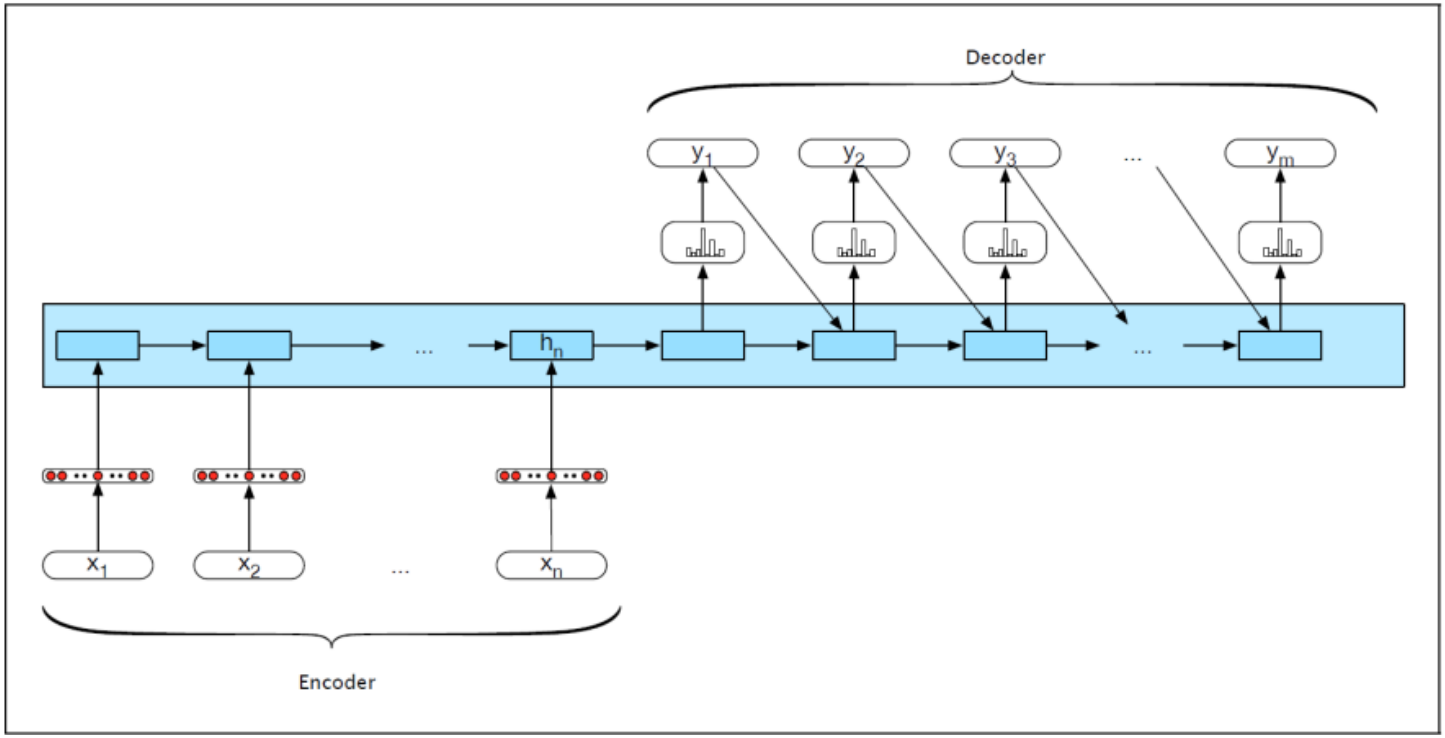
Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. http://emnlp2014.org/papers/pdf/EMNLP2014179.pdf

Seq2seq model

- Limited representation: Encoder and Decoder assumed to have the same internal structure (here RNNs)
- Long distance constrained: Final state of the Encoder is the only context available to Decoder
- Limited context: this context is only available to Decoder as its initial hidden state.

Encoder generates a contextualized representation of the input (last state).

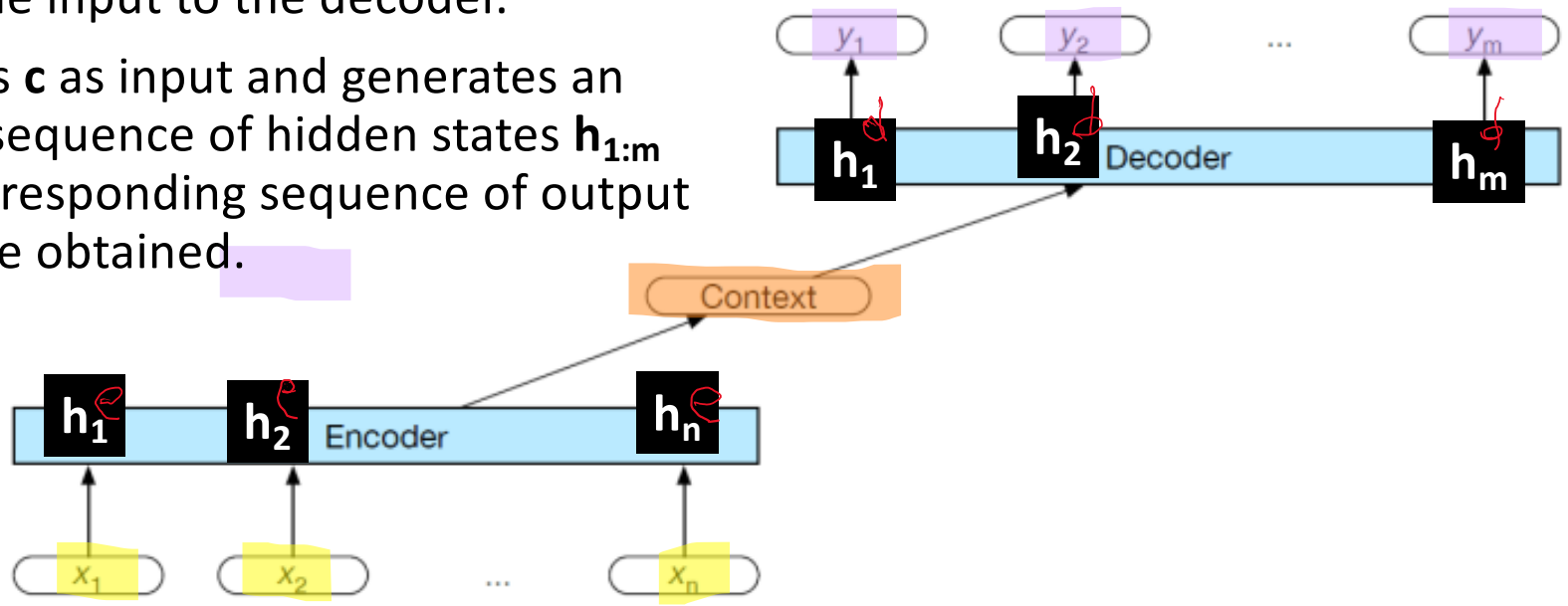
Decoder takes that state and autoregressively generates a sequence of outputs



General Seq2Seq model

Abstracting away from these choices

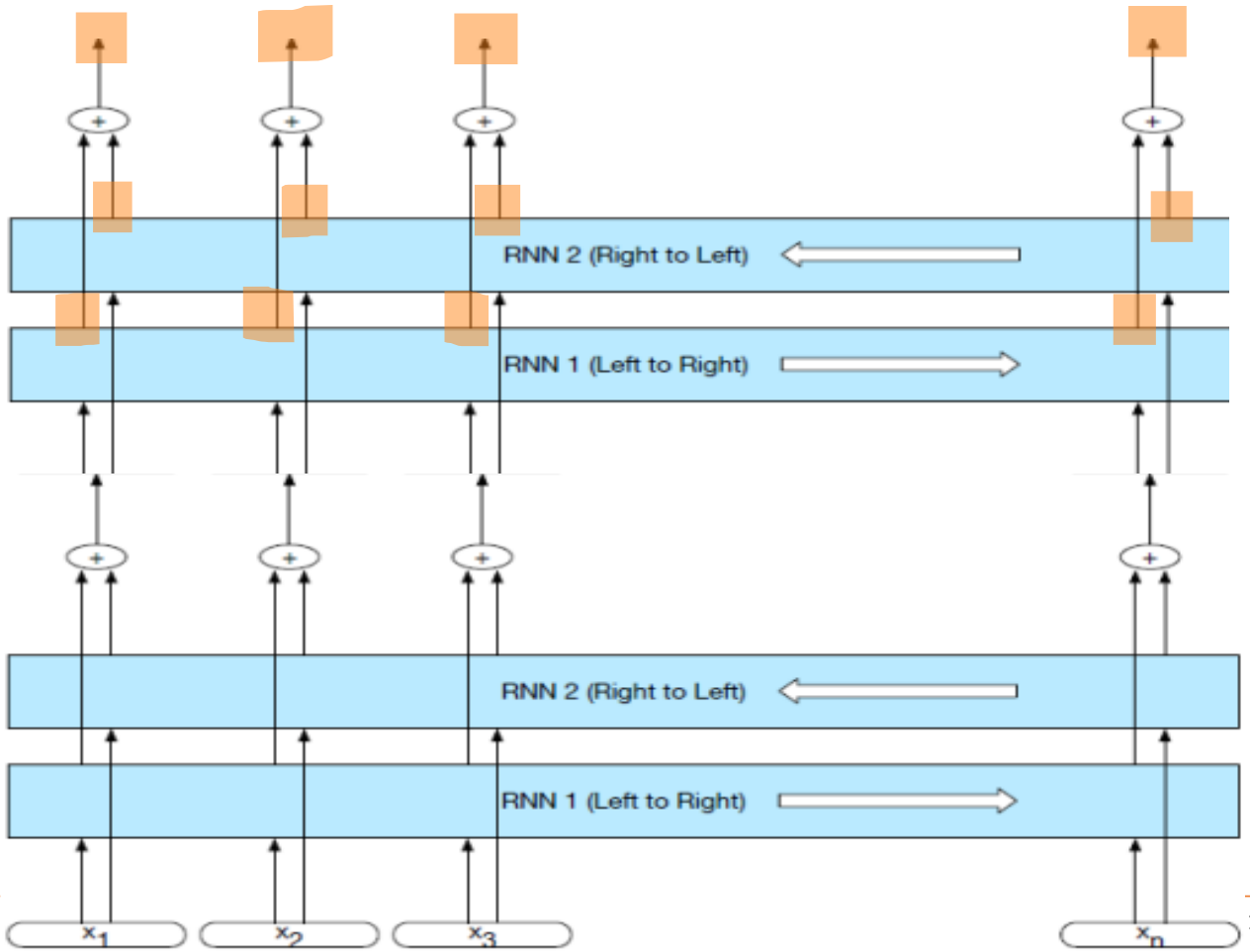
1. **Encoder**: accepts an input sequence, $\mathbf{x}_{1:n}$ and generates a corresponding sequence of contextualized representations, $\mathbf{h}_{1:n}$
2. **Context vector c**: function of $\mathbf{h}_{1:n}$ and conveys the essence of the input to the decoder.
3. **Decoder**: accepts **c** as input and generates an arbitrary length sequence of hidden states $\mathbf{h}_{1:m}$ from which a corresponding sequence of output states $\mathbf{y}_{1:m}$ can be obtained.



Popular architectural choices: Encoder

Widely used encoder design: **stacked Bi-LSTMs**

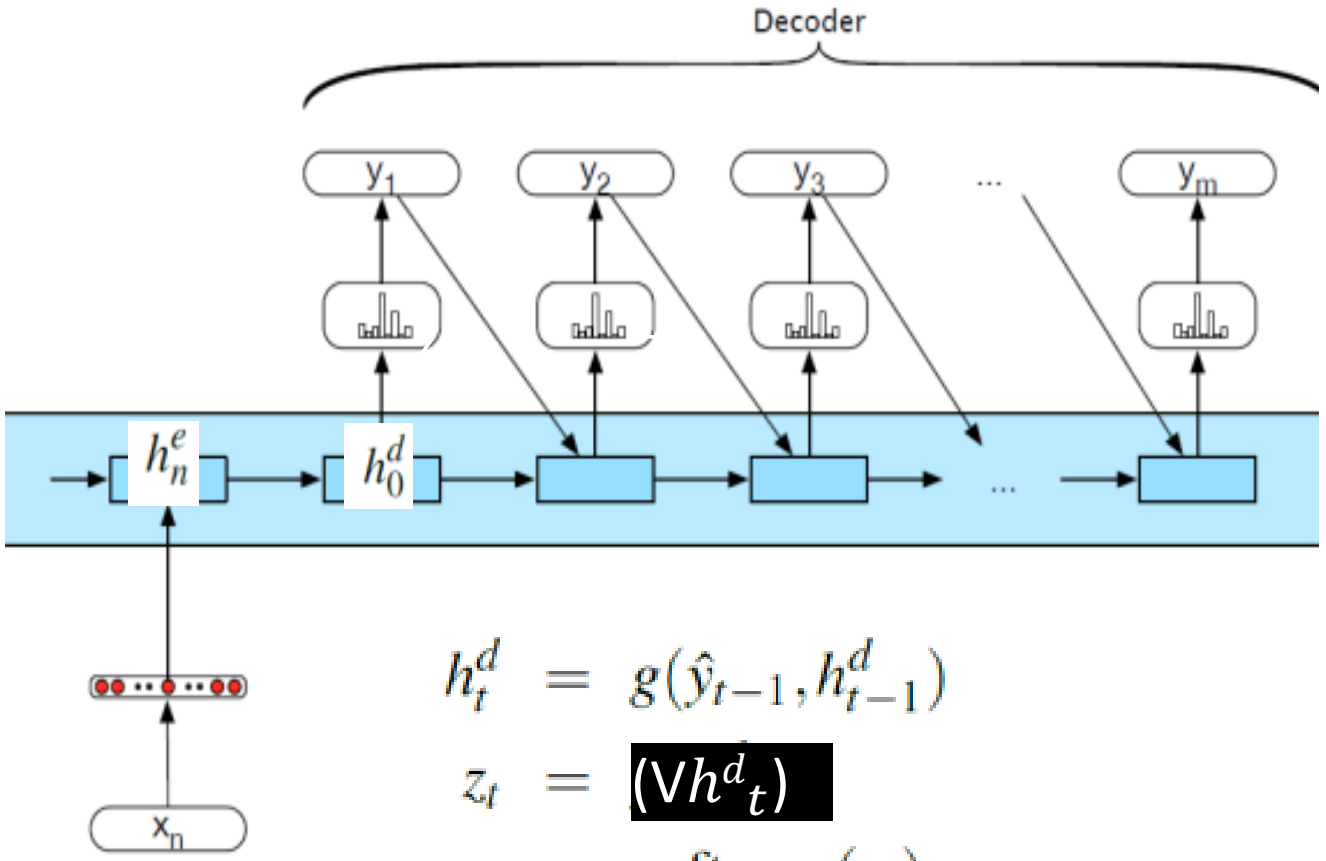
- Contextualized representations for each time step**:** **hidden states from top layers** from the forward and backward passes



Decoder Basic Design

- produce an output sequence an element at a time

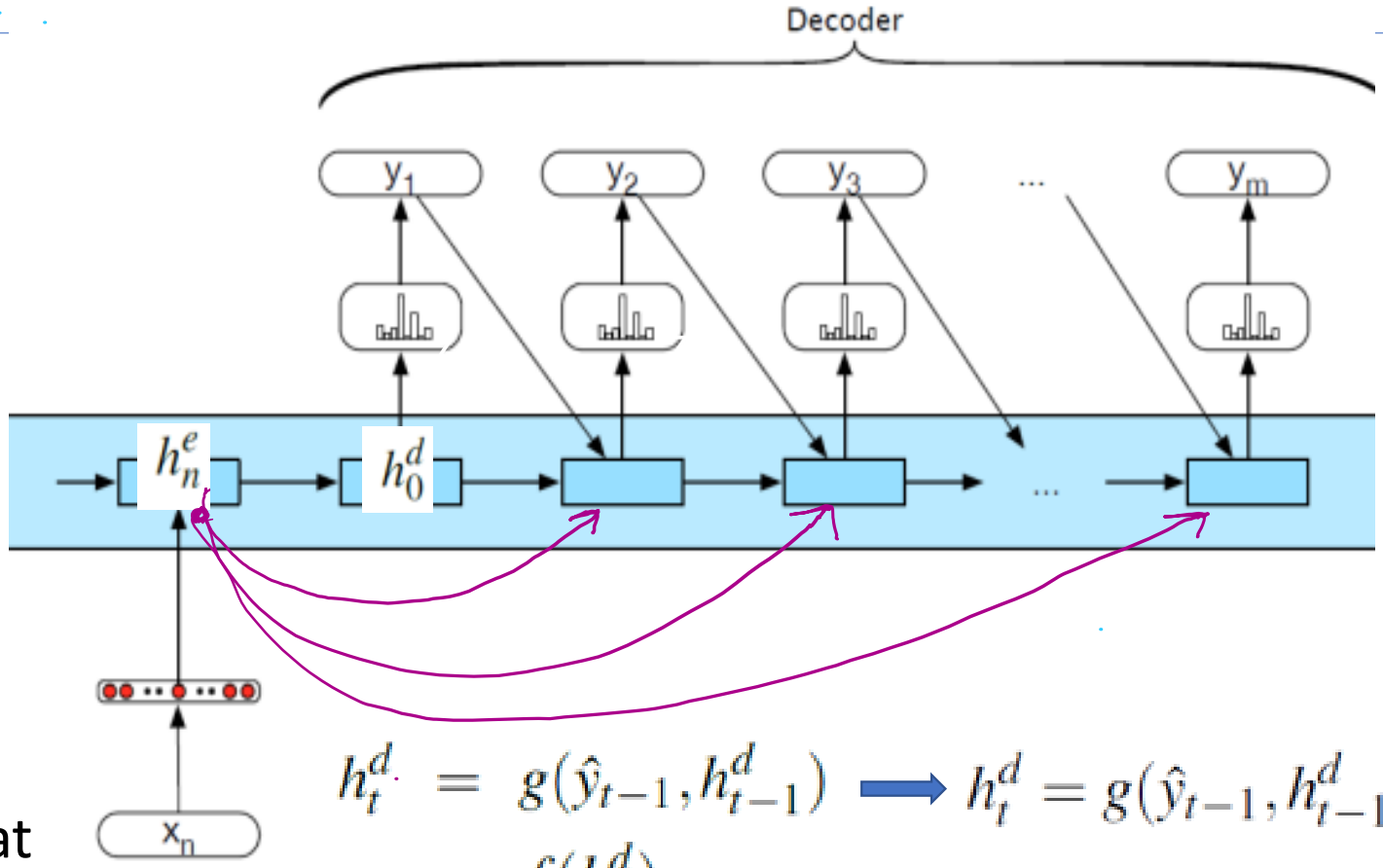
$$c = h_n^e$$
$$h_0^d = c$$



$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d)$$
$$z_t = (Vh^d{}_t)$$
$$y_t = \text{softmax}(z_t)$$

Decoder Design Enhancement

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

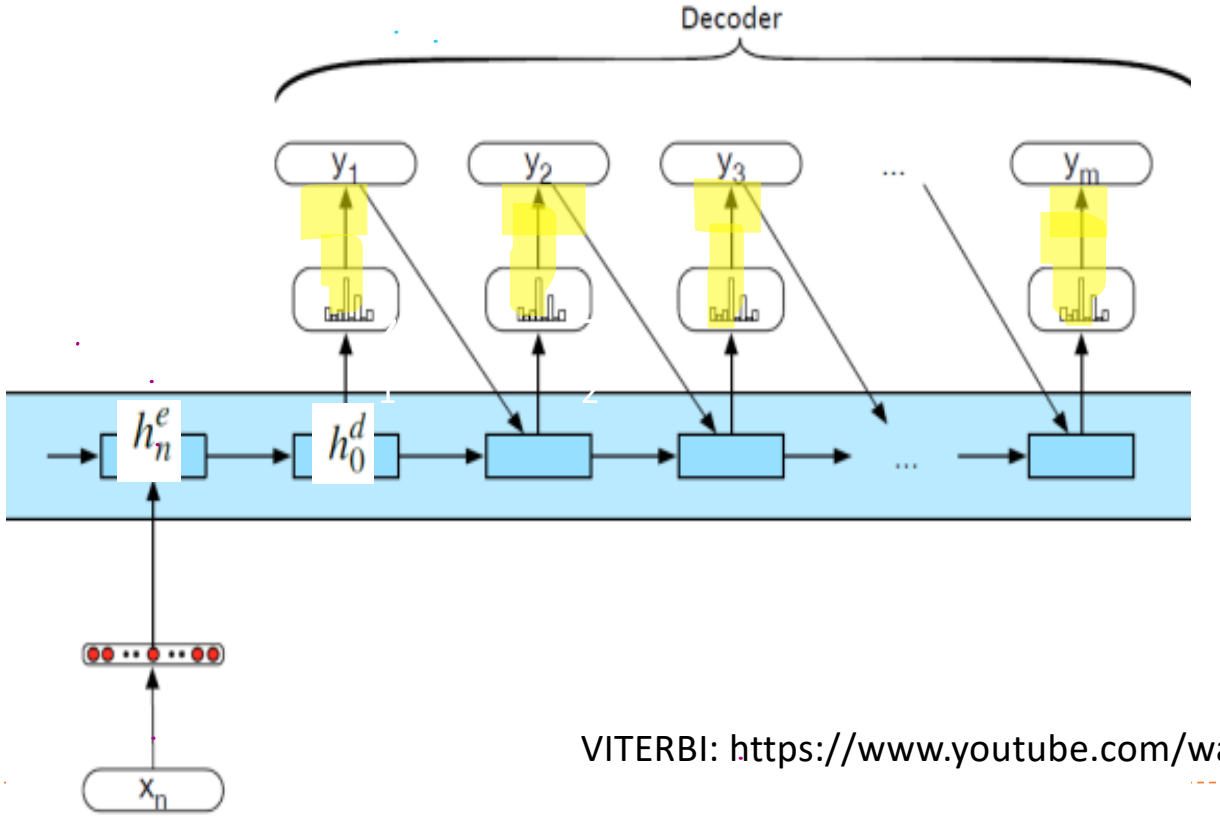


$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d) \rightarrow h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c)$$
$$z_t = f(h_t^d)$$
$$y_t = \text{softmax}(z_t)$$

Context available at each step of decoding

Decoder: How output y is chosen

- **softmax function:** most likely output (does not guarantee that the individual choices made make sense together).



VITERBI: https://www.youtube.com/watch?v=6JVqutwtzmo

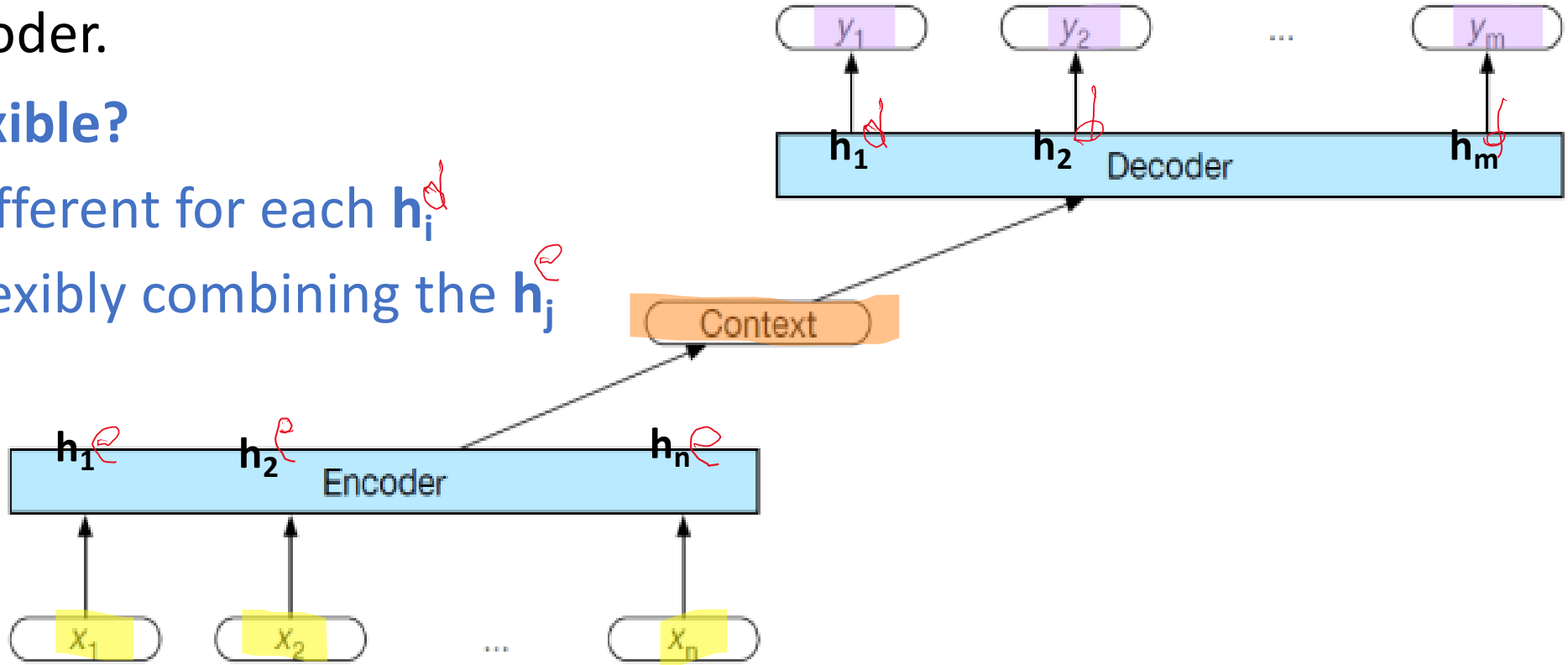
Attentionnel model

Motivation

In Seq2Seq approach we need a flexible context vector c: function of $\mathbf{h}_{1:n}$ and conveys the essence of the input to the decoder.

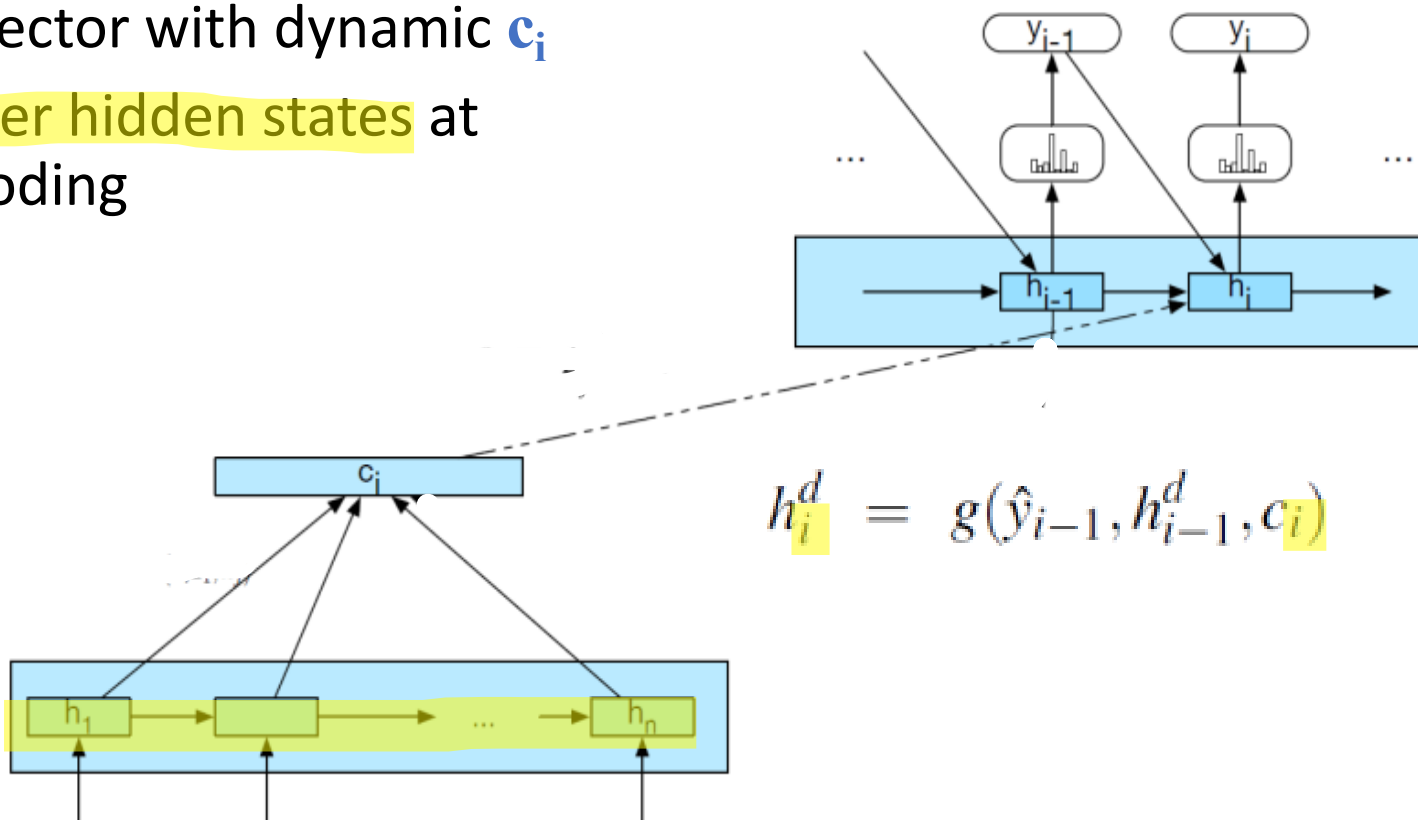
Flexible?

- Different for each $\mathbf{h}_i$
- Flexibly combining the $\mathbf{h}_j$



Attention (1): dynamically derived context

- Replace static context vector with dynamic $\mathbf{c_i}$
- derived from the encoder hidden states at each point i during decoding



Ideas:

- should be a linear combination of those states

$$c_i = \sum_j \alpha_{ij} h_j^e$$

- α_{ij} should depend on ?

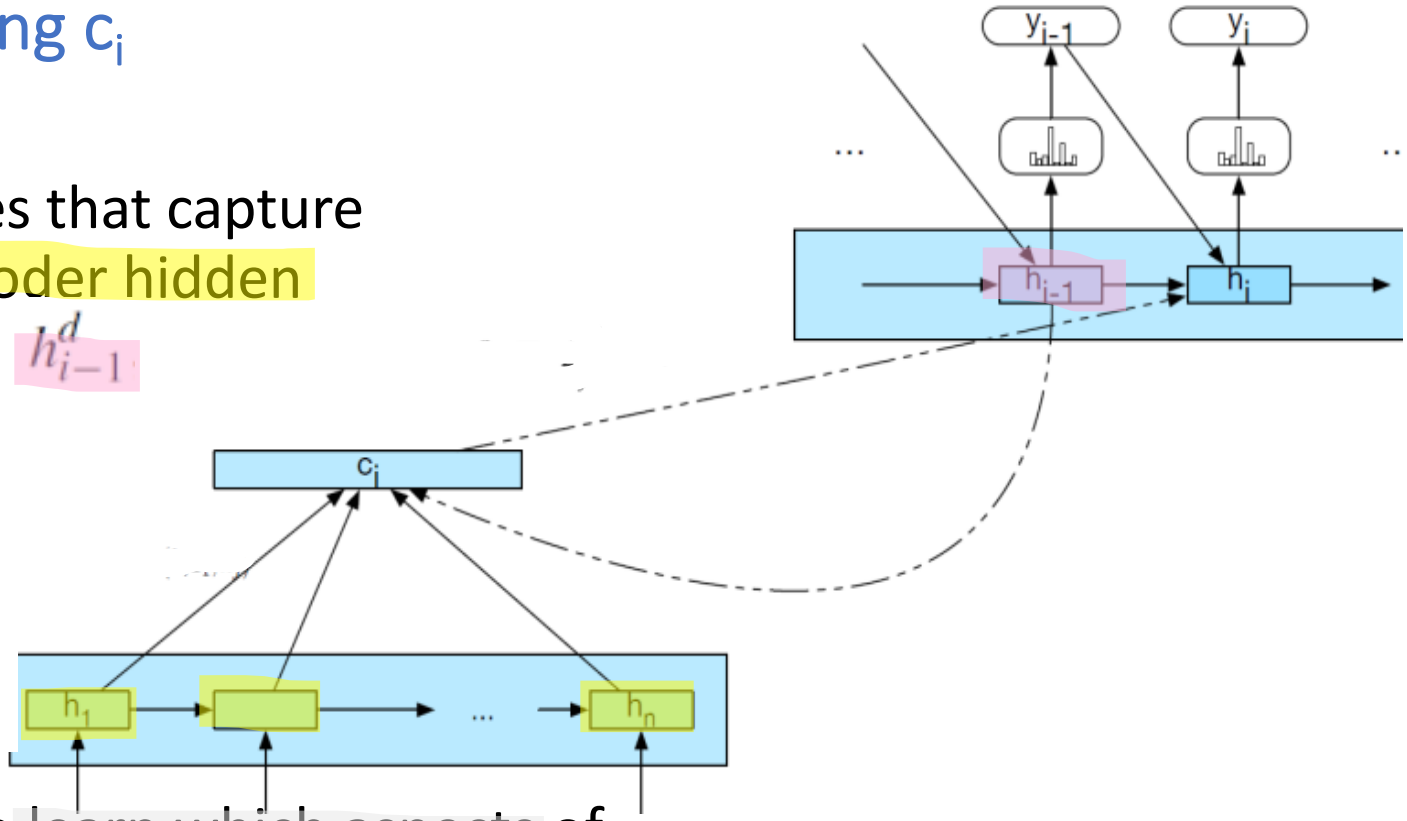
Attention (2): computing c_i

- Compute a vector of scores that capture the relevance of each encoder hidden state to the decoder state h^d_{i-1}

$$score(h^d_{i-1}, h^e_j)$$

- Just the similarity

$$score(h^d_{i-1}, h^e_j) = h^d_{i-1} \cdot h^e_j$$



- Give network the ability to learn which aspects of similarity between the decoder and encoder states are important to the current application.

$$score(h^d_{i-1}, h^e_j) = h^d_{i-1} W_s h^e_j$$

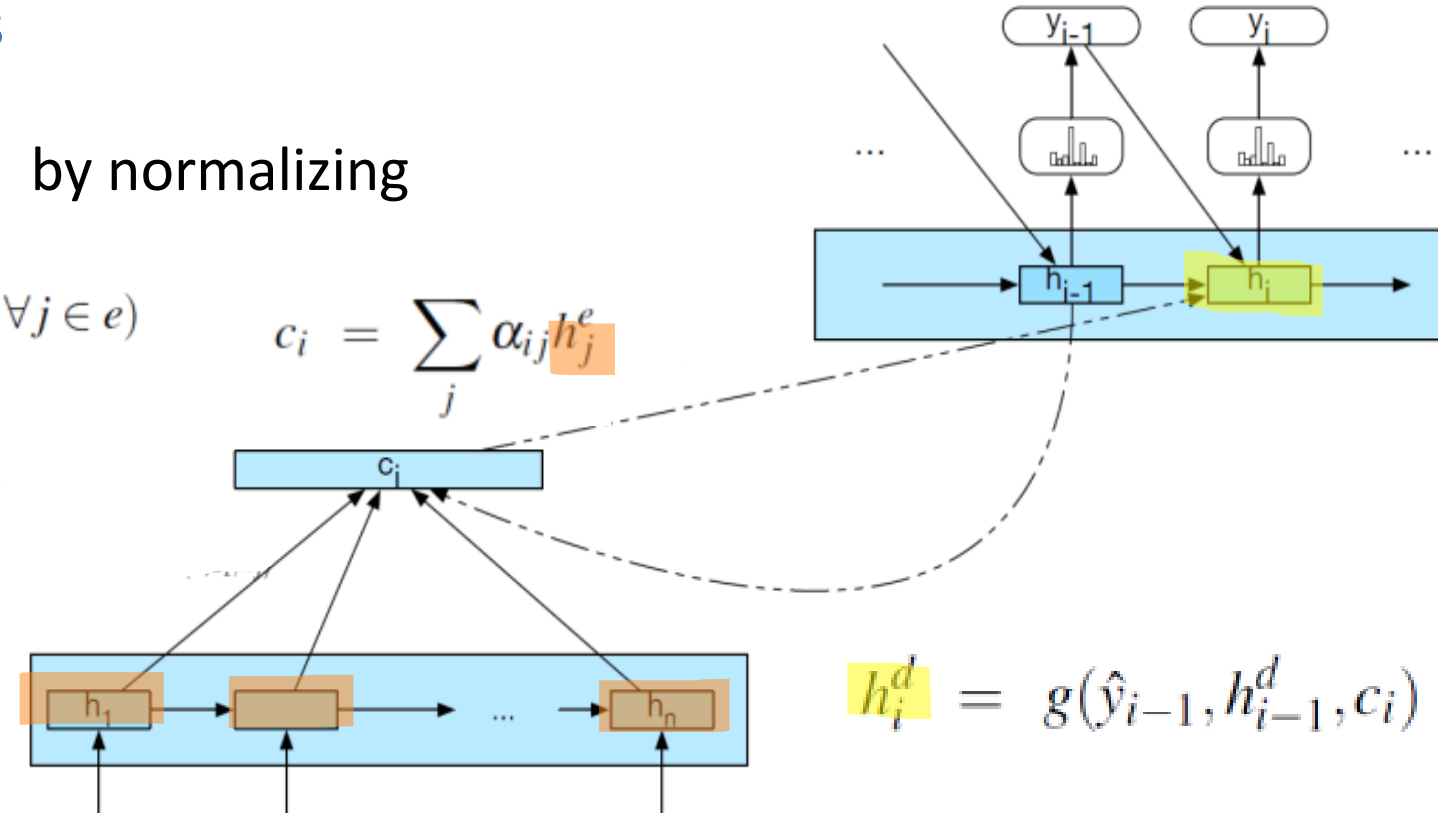
Attention (3): computing c_i
From scores to weights

- Create vector of weights by normalizing scores

$$\alpha_{ij} = \text{softmax}(score(h^d_{i-1}, h^e_j) \ \forall j \in e)$$

$$= \frac{exp(score(h^d_{i-1}, h^e_j)}{\sum_k exp(score(h^d_{i-1}, h^e_k))}$$

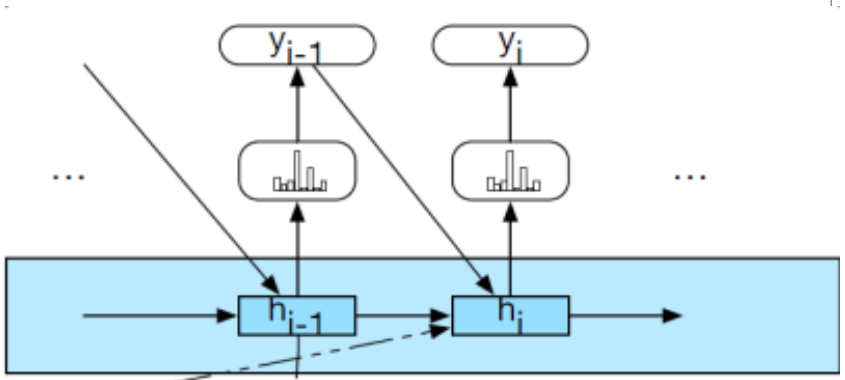
$$c_i = \sum_j \alpha_{ij} h^e_j$$



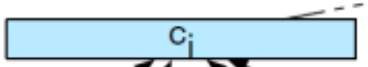
$$h^d_i = g(\hat{y}_{i-1}, h^d_{i-1}, c_i)$$

- **Goal achieved**: compute a fixed-length context vector for the current decoder state by taking a weighted average over all the encoder hidden states.

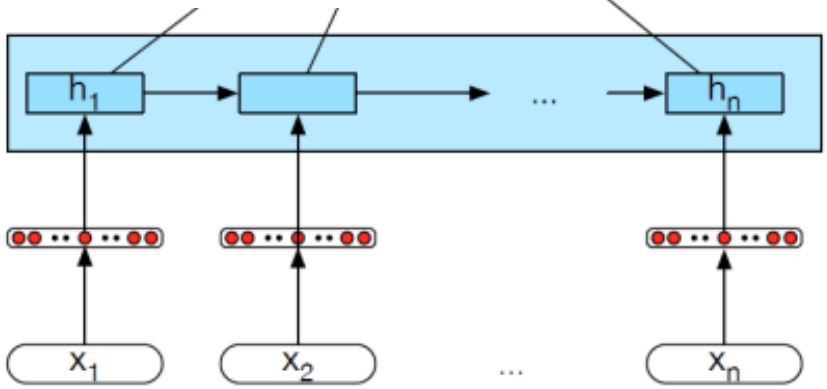
Attention: Summary



$$c_i = \sum_j \alpha_{ij} h_j^e$$



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



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

$$c_i = \sum_j \alpha_{ij} h_j^e$$

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

Intro to Encoder-Decoder and Attention
3 LSTM Seq2seq layers

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

$$c_i = \sum_j \alpha_{ij} h_j^e$$

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

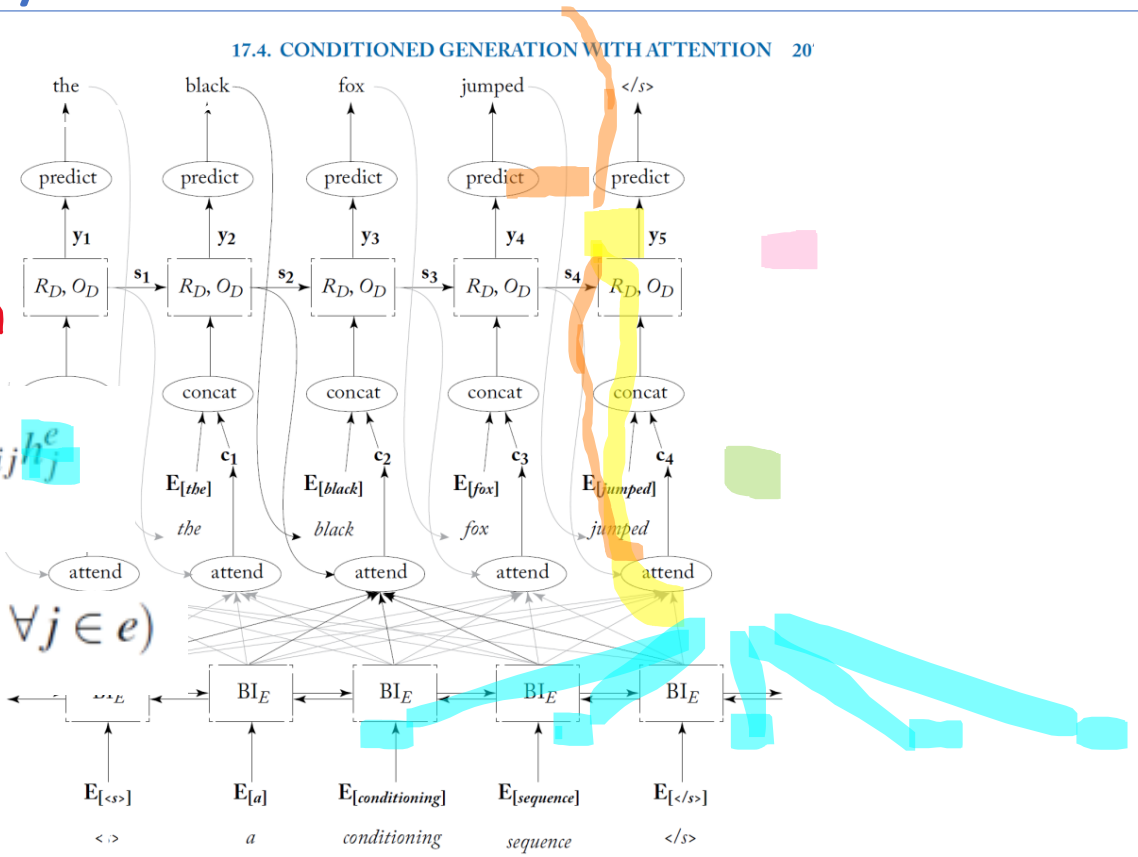
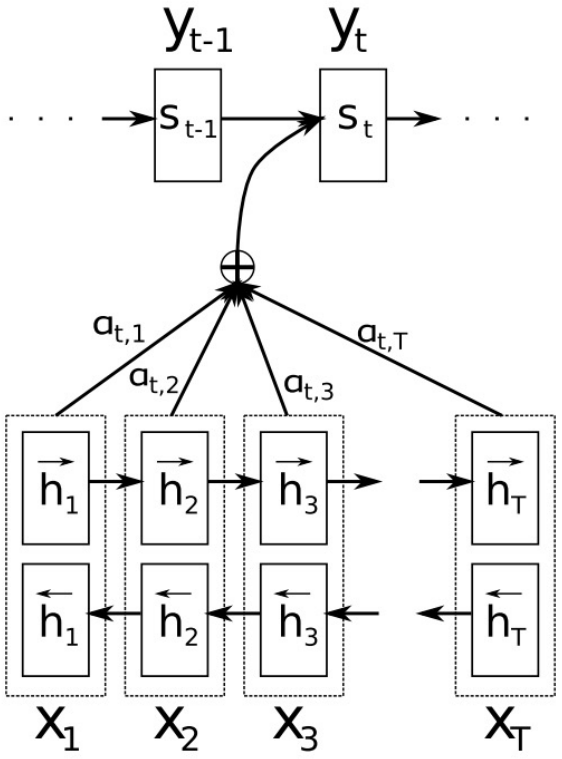
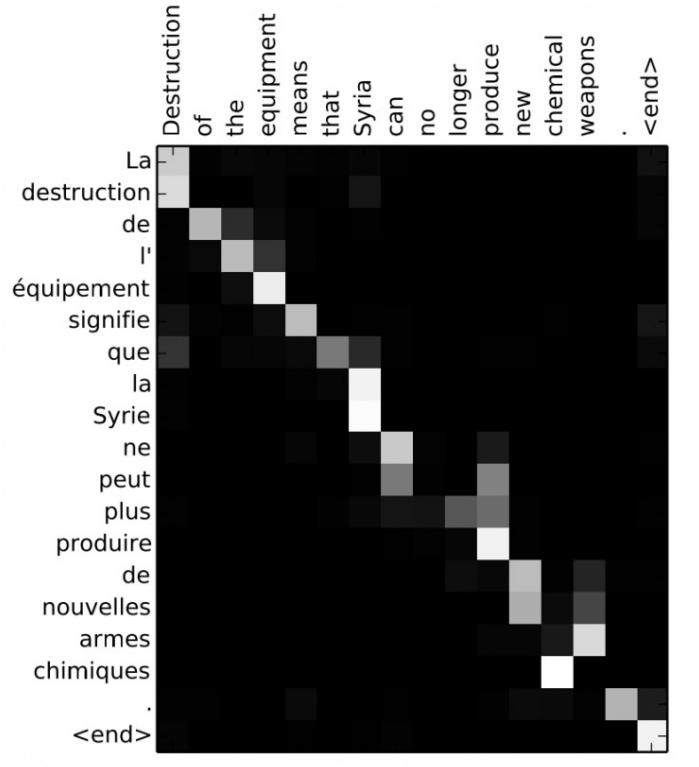


Figure 17.5: Sequence-to-sequence RNN generator with attention.

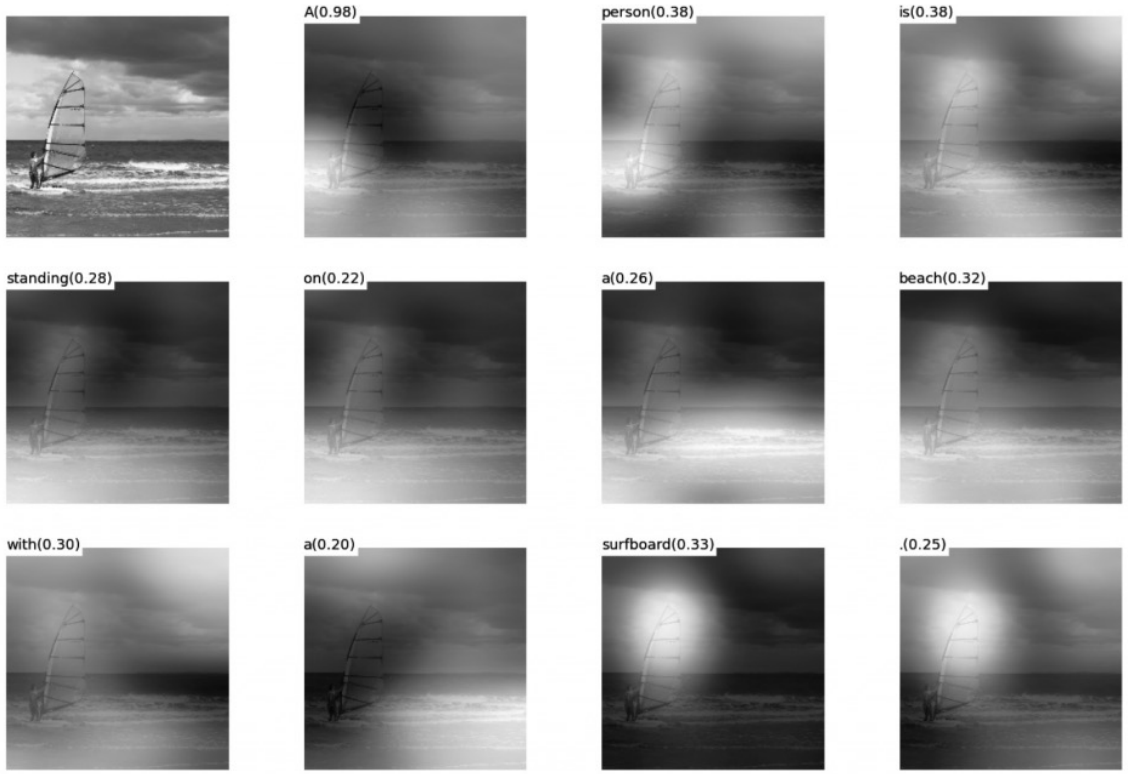
Neural Machine Translation (NMT) with Recurrent Nets and Attention Mechanism



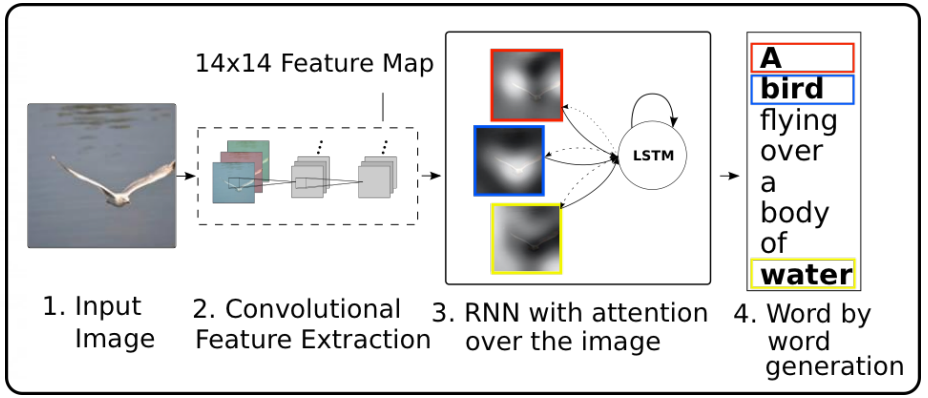


D. Bahdanau, K. H. Cho, Y. Bengio, **Neural machine translation by jointly learning to align and translate,** ICLR 2015

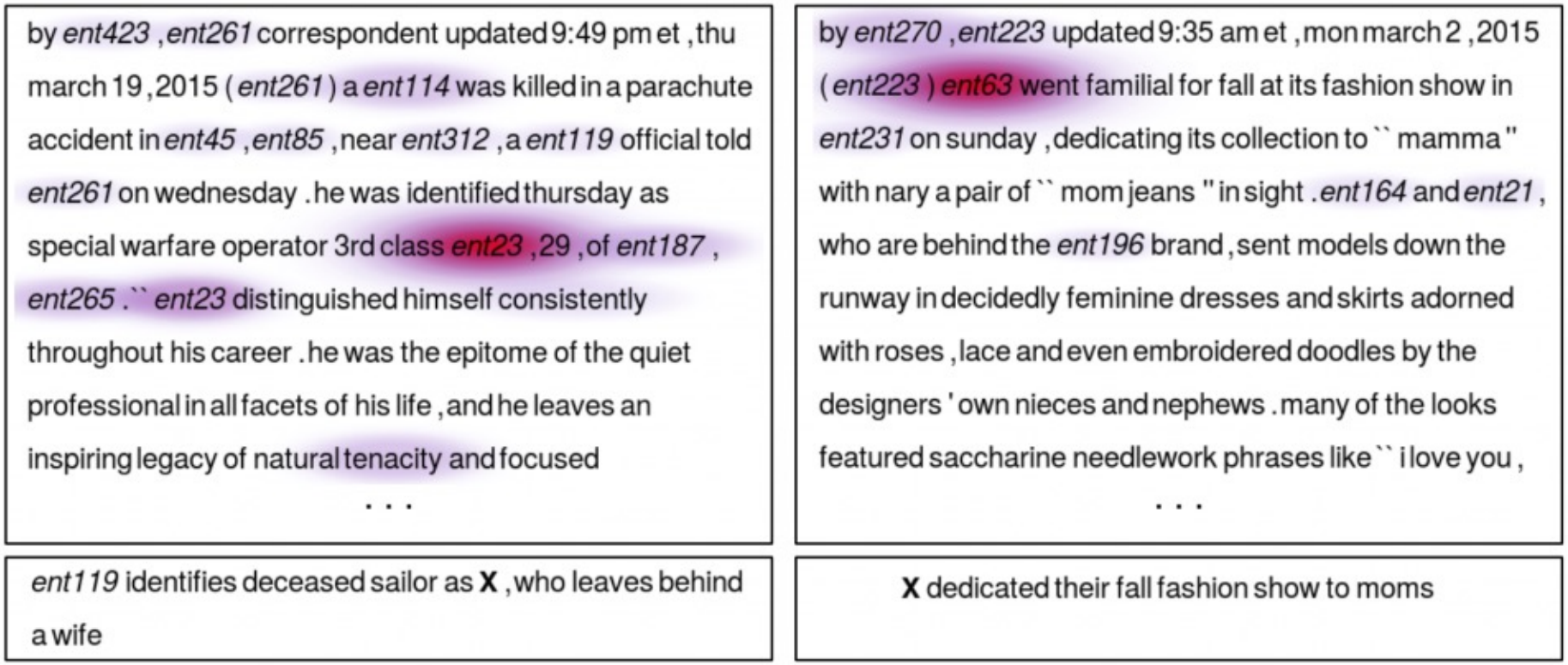
Image-to-Text: Caption Generation with Attention



(b) A person is standing on a beach with a surfboard.



Teaching Machines to Read and Comprehend



Neural Attention Model for Sentence Summarization

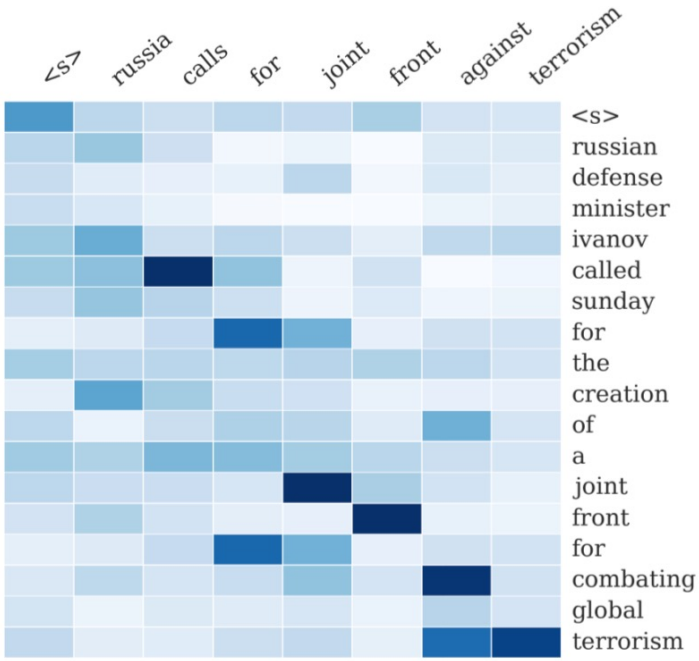
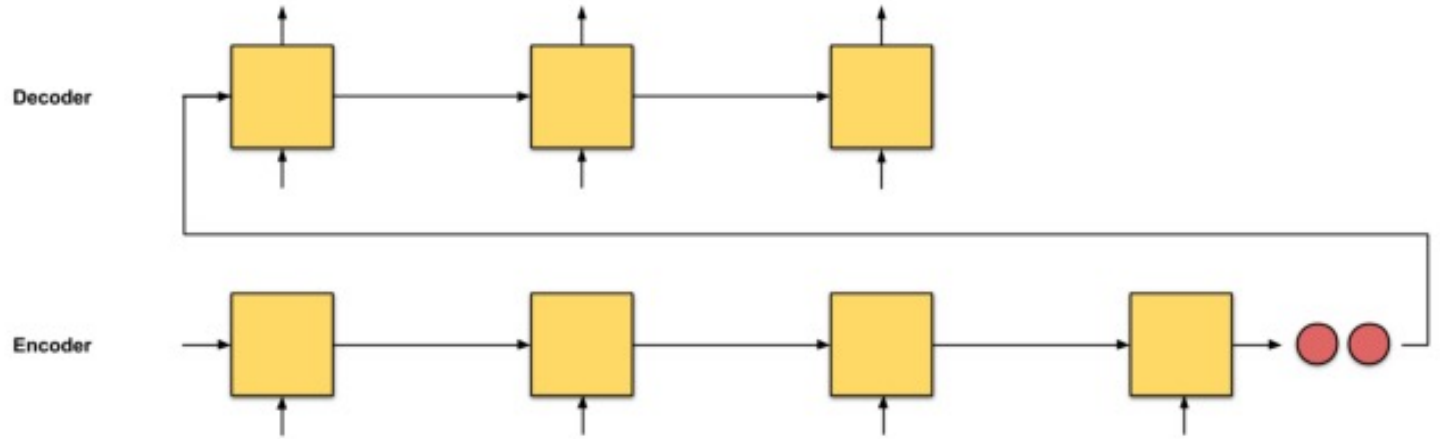


Figure 1: Example output of the attention-based summarization (ABS) system. The heatmap represents a soft alignment between the input (right) and the generated summary (top). The columns represent the distribution over the input after generating each word.

Summary

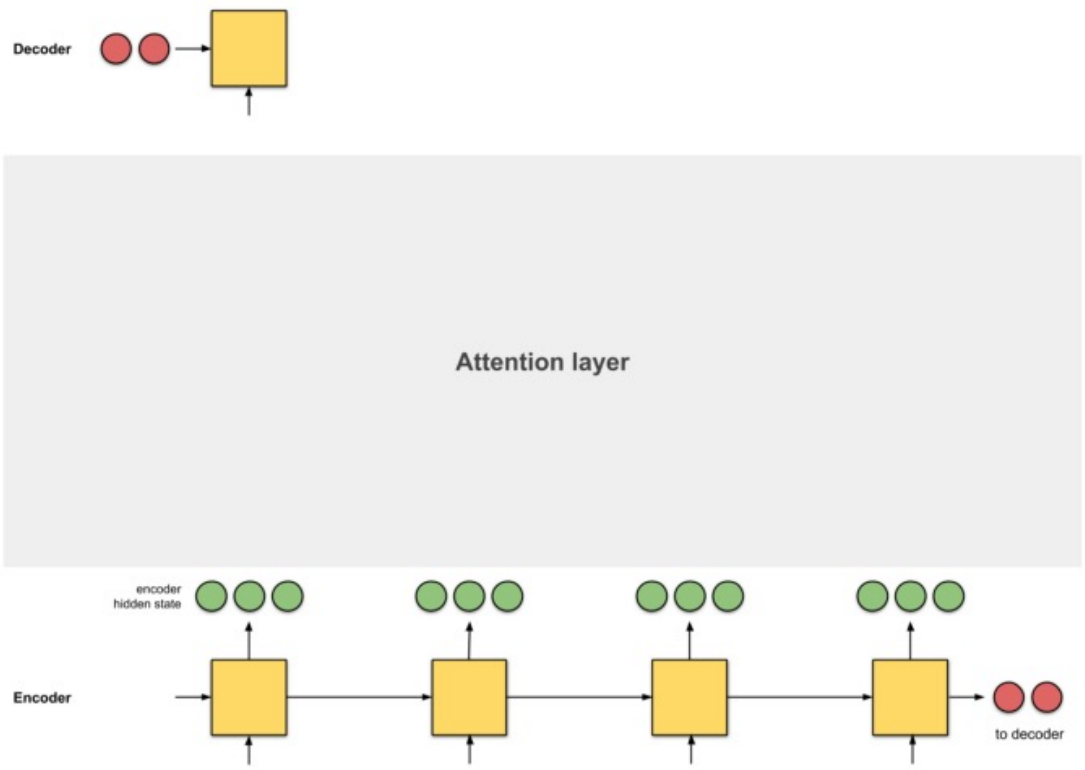
Traditionnal Seq2Seq architecture



: https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3#0458

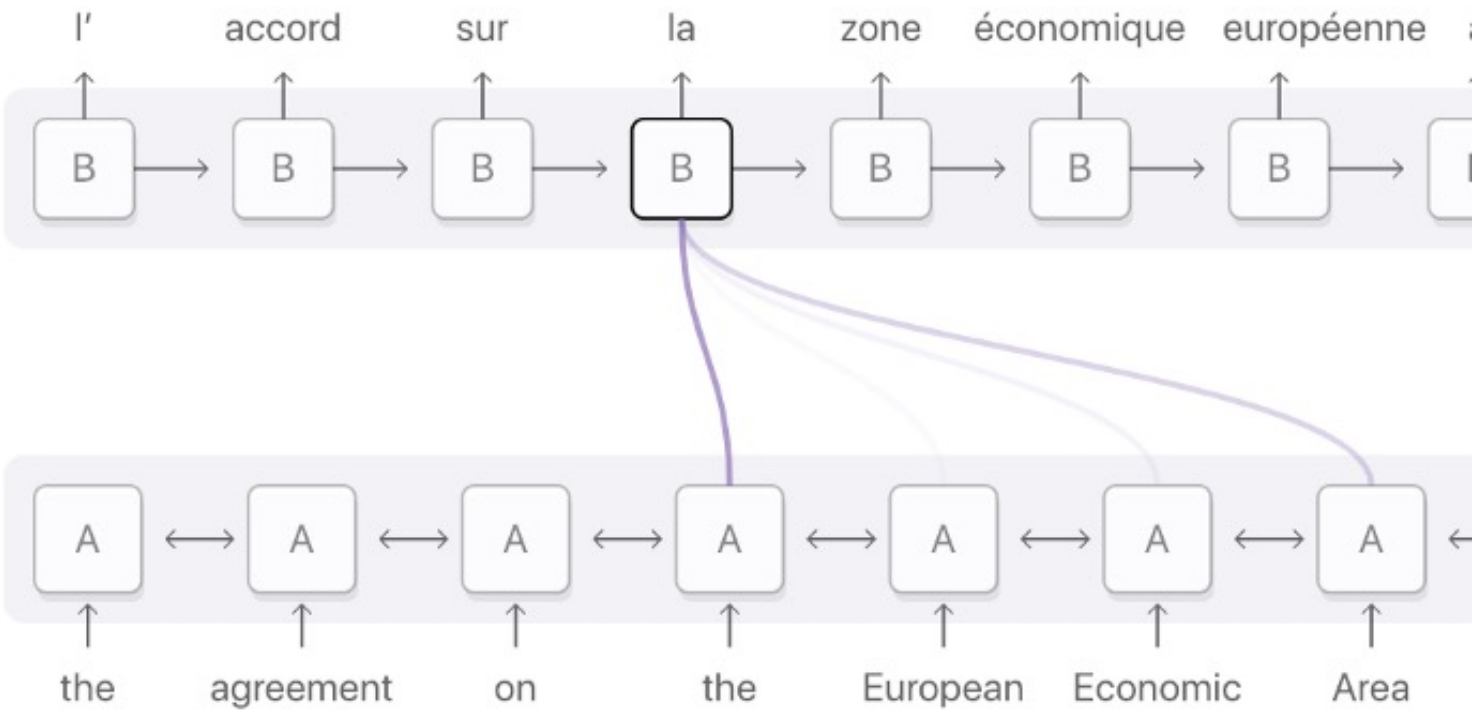
Seq2seq with an input of sequence length 64 and output length 53

Seq2seq with attention layer



Alignment

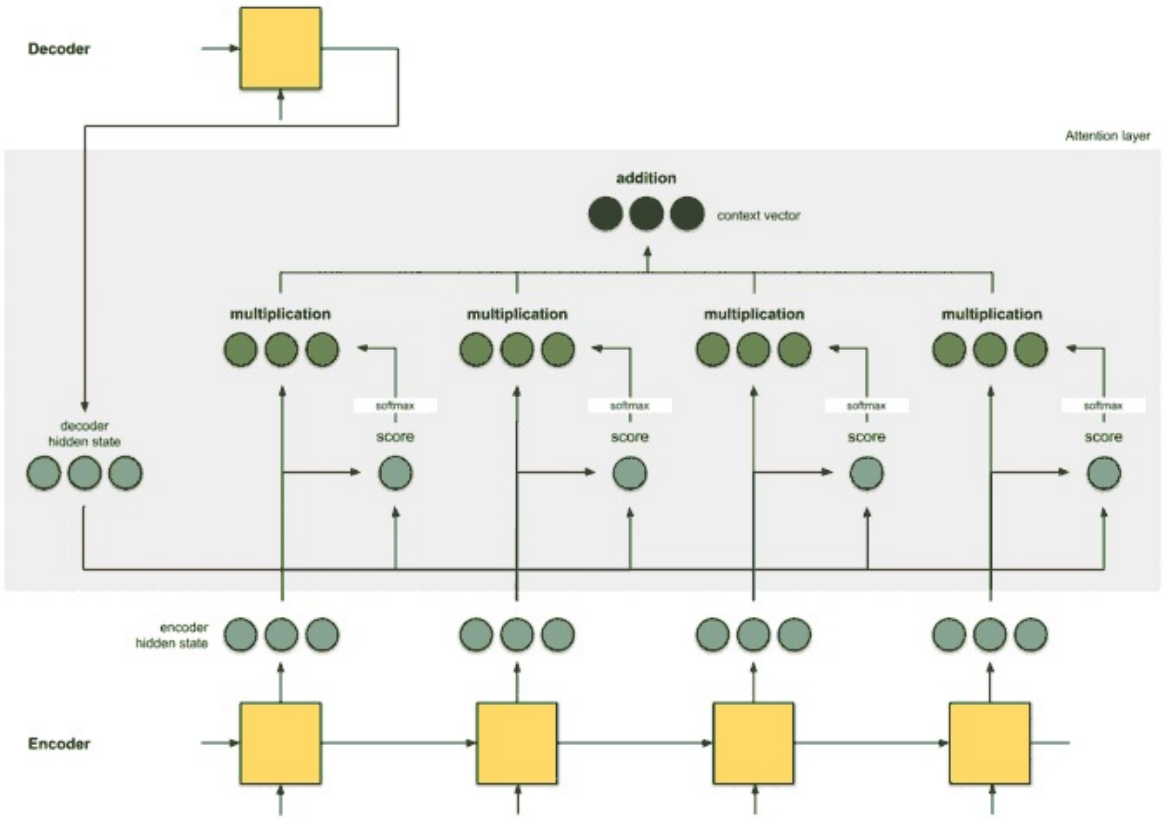
- Alignment means matching segments of an original text with their corresponding segments of the translation.



Be carefull: 3 types of attention

- ***global attention**:* uses all the encoder hidden
  - Presented in this lecture
- ***local attention**:* uses only a subset of the encoder hidden states
- ***self attention**:* uses with transformer architecture
  - Could be parallized

Global attention



At each output prediction step

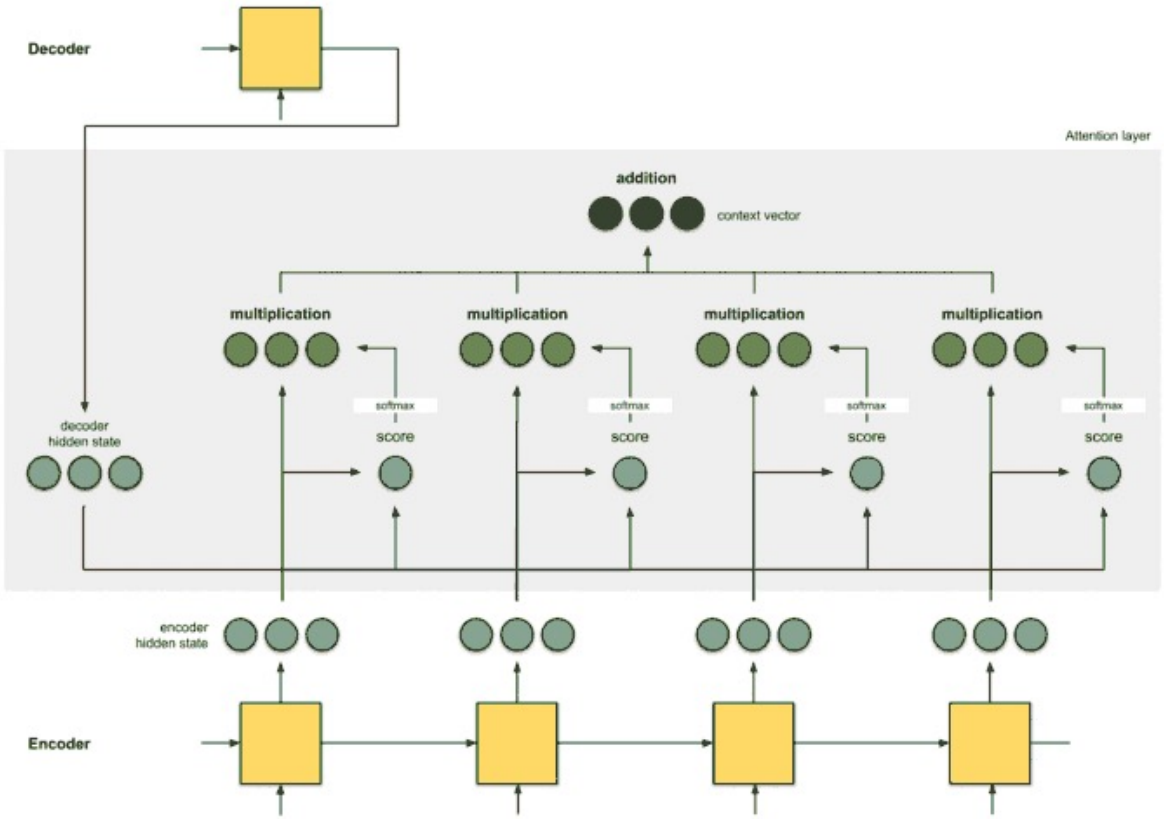
calculate a score for each input : for example dot product between previous decoder hidden state and encoder hidden state.

Normalized score with softmax

Multiply each hidden state by its normalized score

Sum the result, in order to obtain the context vector

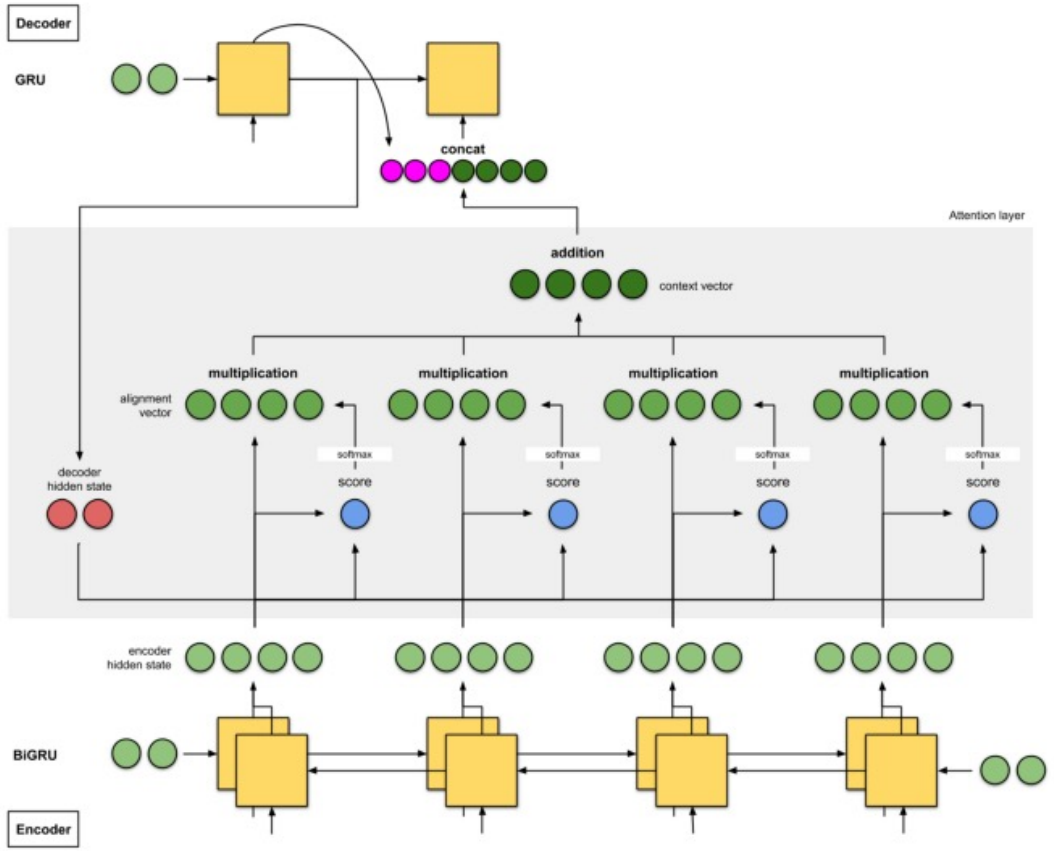
Global attention



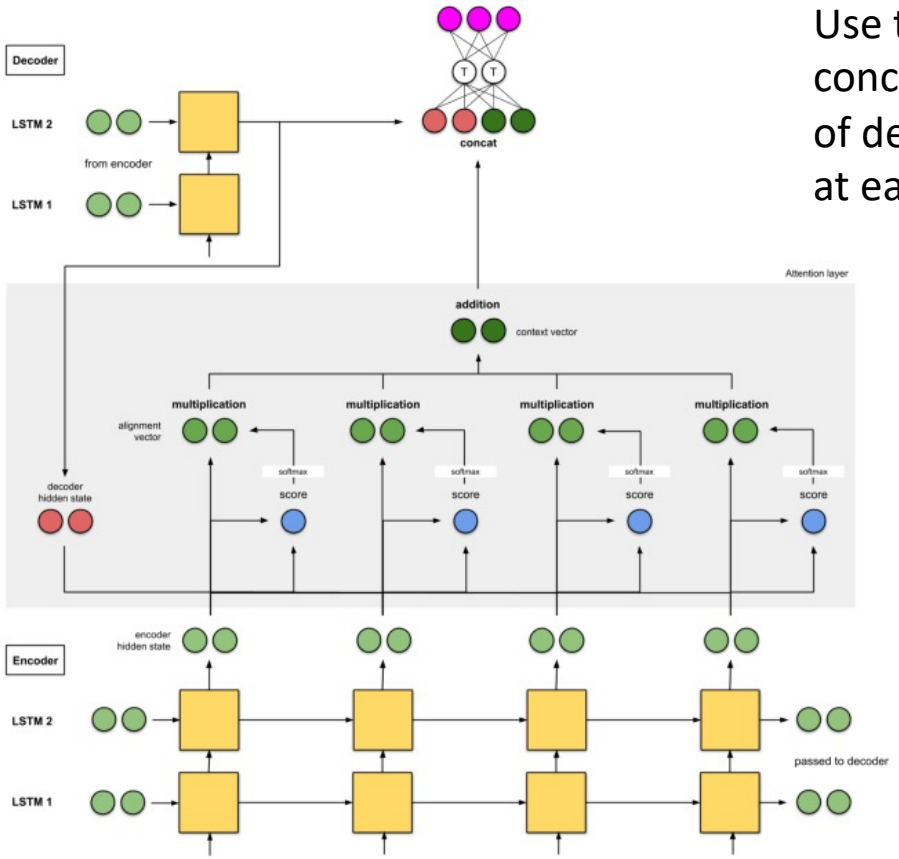
During training, the input to each decoder time step t is our **ground truth output** from decoder time step *t-1*.

During inference, the input to each decoder time step t is the **predicted output** from decoder time step *t-1*.

Bahdanau et. al (2015)
Encoder use BI-LSTM / Decoder use LSTM

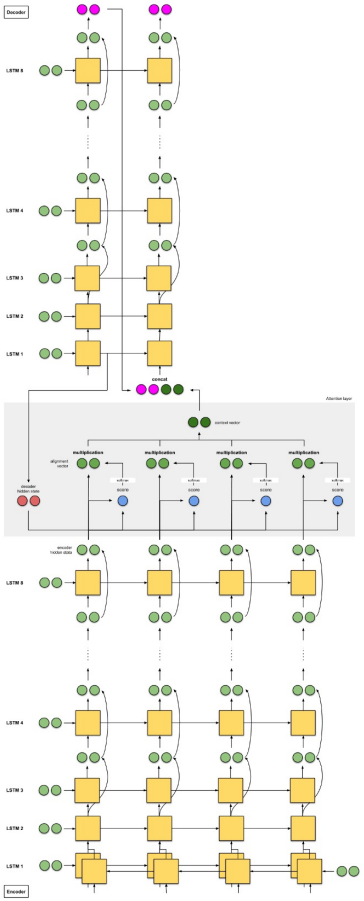


Luong et. al (2015)
Encoder 2 LSTM layers / Decoder 2 LSTM layers



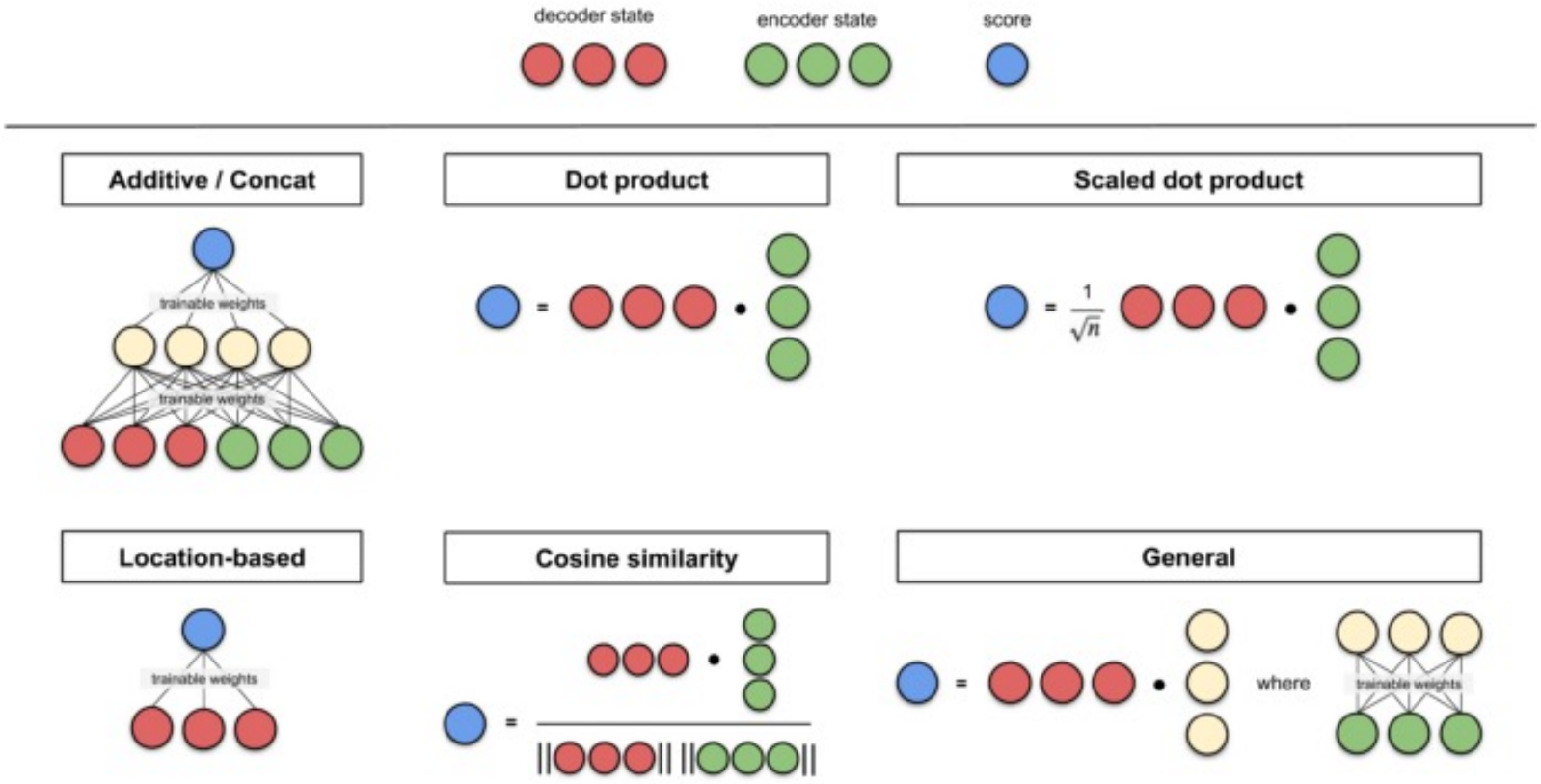
Use the same MLP with the concatenation of hidden state of decoder and context vector at each step

Google's Neural Machine Translation (GNMT) in 2016



- The encoder consists of a stack of 8 LSTMs, where the first is bidirectional (whose outputs are concatenated), and a residual connection exists between outputs from consecutive layers (starting from the 3rd layer). The decoder is a *separate* stack of 8 unidirectional LSTMs.
- The score function used is the **additive/concat**
- The input to the next decoder step is the concatenation between the output from the previous decoder time step (pink) and the context vector from the current time step (dark green).

Different score function



Lab work

Lab work

1. Understand the code
2. Play with LSTM model for sentiment analysis
   - Use BI-LSTM
   - Use stacked LSTM
   - Use all hidden state and average it
3. Play with LSTM model with attention for sentiment analysis
   - Take inspiration from the course slides to build an original architecture that you will describe
4. Upload on moodle a **clean**, **documented** notebook containing your **best LSTM attentional** model.
   - The evaluation metric is the f1 score (macro avg)
   - Exploit the attention in order to explain the decision of the network
   - This notebook will be evaluated, and the grade will take into account the editorial quality of your text.
   - Any notebook containing more than 1 model will not be evaluated (score = 0 -> **You have to choose the best one**).
     - Best: original architecture, very good performance… argument on it.