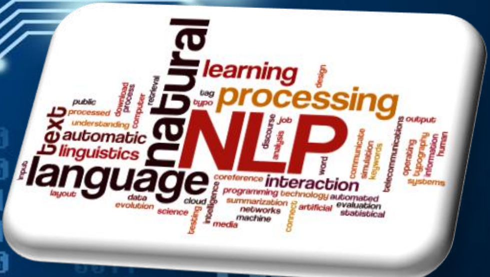
Cairo University
Faculty of Engineering
Computer Engineering Department

Natural Language Processing

Dr. Sandra Wahid



Machine Translation

- It is the use of computers to translate from one language to another.

- Machine translation (MT) focuses on a number of very **practical tasks**:
  - **Information access:** translate information on the web such as articles, reviews, …
  - **Computer-aided translation (CAT):** produce a draft translation that is fixed up in a post-editing phase by a human translator.
  - **In-the-moment human communication needs:** incremental translation, translating speech on-the-fly before the entire sentence is complete.
  - **Image-centric translation:** use OCR of the text on a phone camera image as input to an MT system to translate menus or street signs.

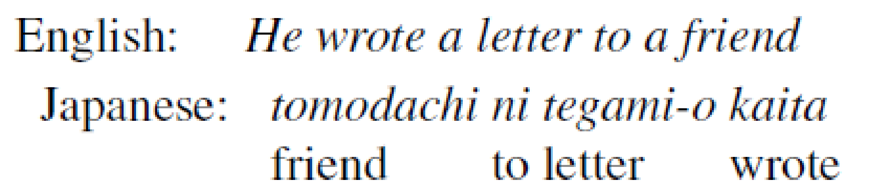
- The standard algorithm for MT is the **encoder-decoder network**, also called the **sequence to sequence network**, an architecture that can be implemented with **RNNs** or with **Transformers**.

Machine Translation

- Encoder-decoder or sequence-to-sequence models are used for a different kind of sequence modeling (either than POS-tagging/NER) in which the output sequence is a complex function of the entire input sequencer

  →mapping from a sequence of input words or tokens to a sequence of tags that are **not merely direct mappings from individual words**.

- Example:



English: *He wrote a letter to a friend*

Japanese: *tomodachi ni tegami-o kaita*

friend to letter wrote

  - In English, the verb is in the middle of the sentence, while in Japanese, the verb *kaita* comes at the end. The Japanese sentence doesn't require the pronoun *he*, while English does.

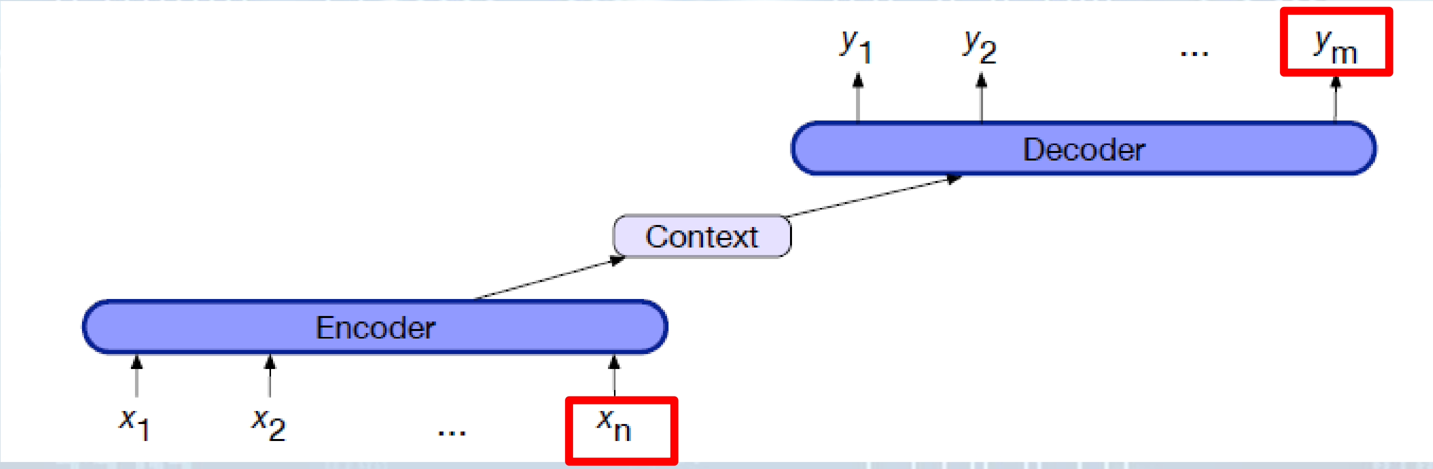
- Encoder-decoder networks are very successful at handling these sorts of complicated cases of sequence mappings.
- The encoder-decoder algorithm is not just for MT, it's the state of the art for many other tasks where complex mappings between two sequences are involved:
  - **summarization** (where we map from a long text to its summary, like a title or an abstract)
  - **dialogue** (where we map from what the user said to what our dialogue system should respond)
  - **semantic parsing** (where we map from a string of words to a semantic representation like logic or SQL).

Encoder-Decoder Model

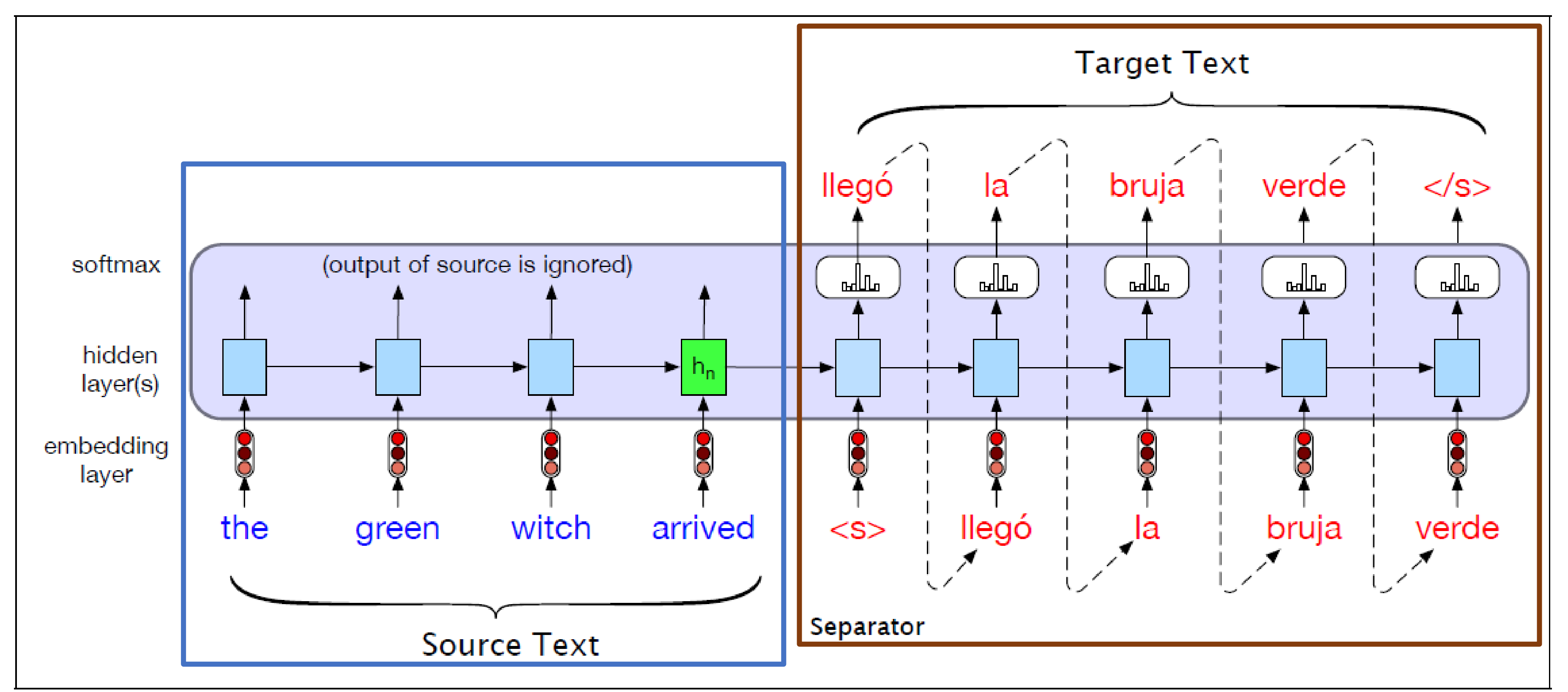
- Are models capable of generating contextually appropriate, **arbitrary length**, output sequences.

- Applied to a very wide range of applications including machine translation, summarization, question answering, and dialogue.

- The key idea: is the use of an **encoder** network that takes an input sequence and creates a contextualized representation of it, often called the **context**. This representation is then passed to a **decoder** which generates a task-specific output sequence.



Encoder-Decoder with RNNs



$$\mathbf{h}_t = g(\mathbf{h}_{t-1}, \mathbf{x}_t)$$
$$\mathbf{y}_t = f(\mathbf{h}_t)$$

- g is an activation function like **tanh** or **ReLU**, a function of the input at time t and the hidden state at time t-1.

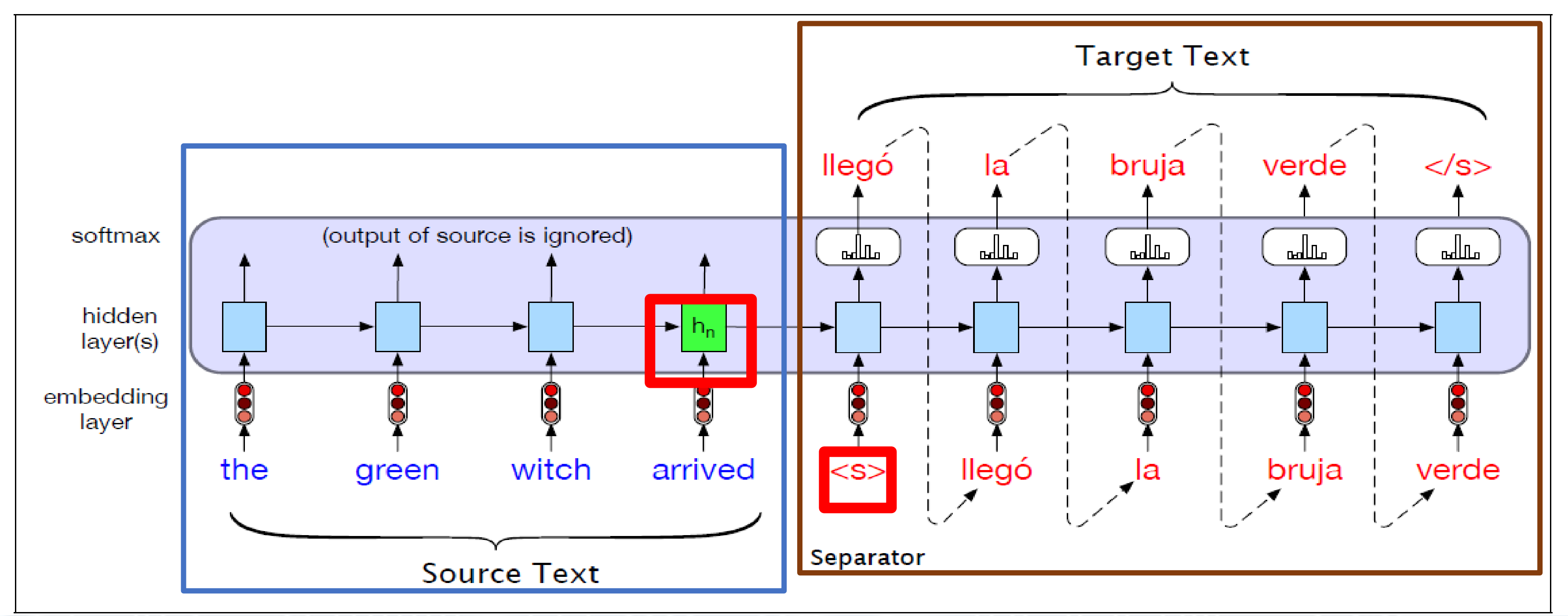
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$f(x) = \max(0, x)$$

- f is a softmax over the set of possible vocabulary items.

Encoder-Decoder with RNNs

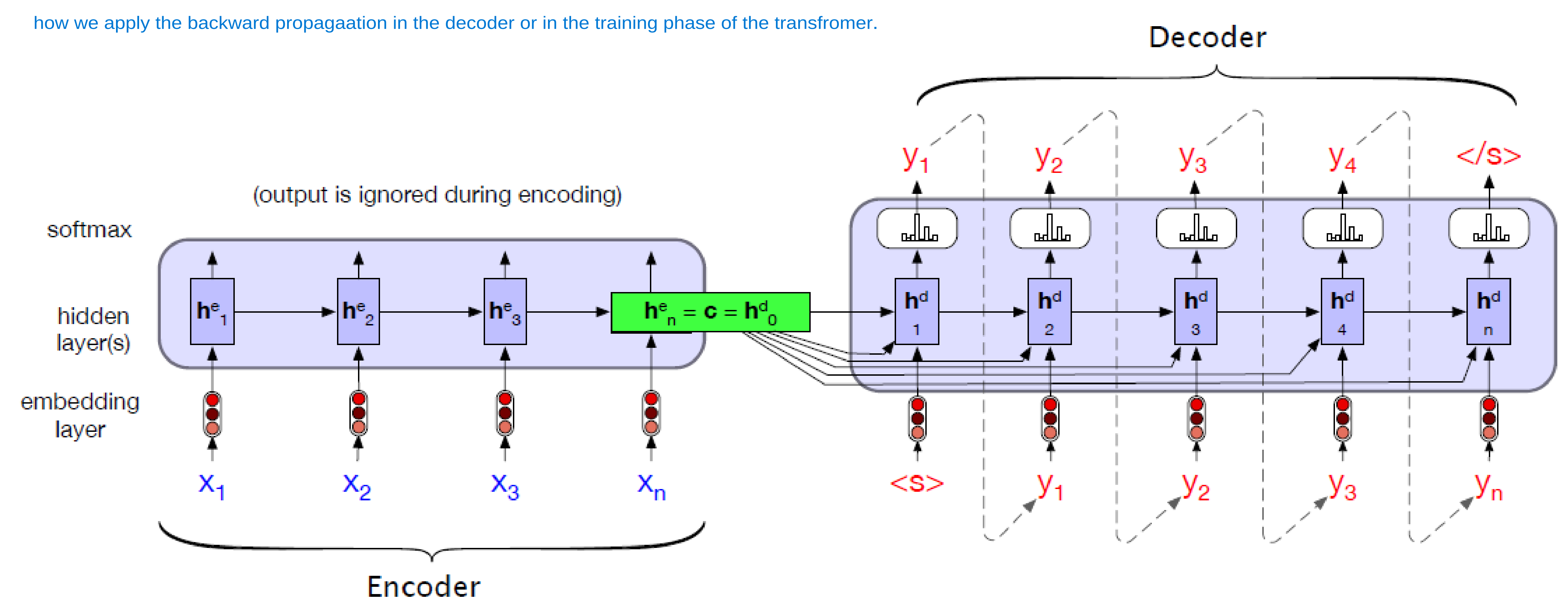
- In MT, note the addition of a **sentence separation marker** at the end of the source text, and then simply concatenate the target text.



- An English source text (“the green witch arrived”), a sentence separator token <s>, and a Spanish target text (“llego´ la bruja verde”).
- To translate a source text, we run it through the network performing forward inference to generate hidden states until we get to the end of the source “hn”. Then we begin **autoregressive** generation (since the word generated at each time step is conditioned on the word selected by the network from the previous step), asking for a word in the context of the hidden layer from the end of the source input.
- Subsequent words are conditioned on the previous hidden state and the embedding for the last word generated.

Encoder-Decoder with RNNs

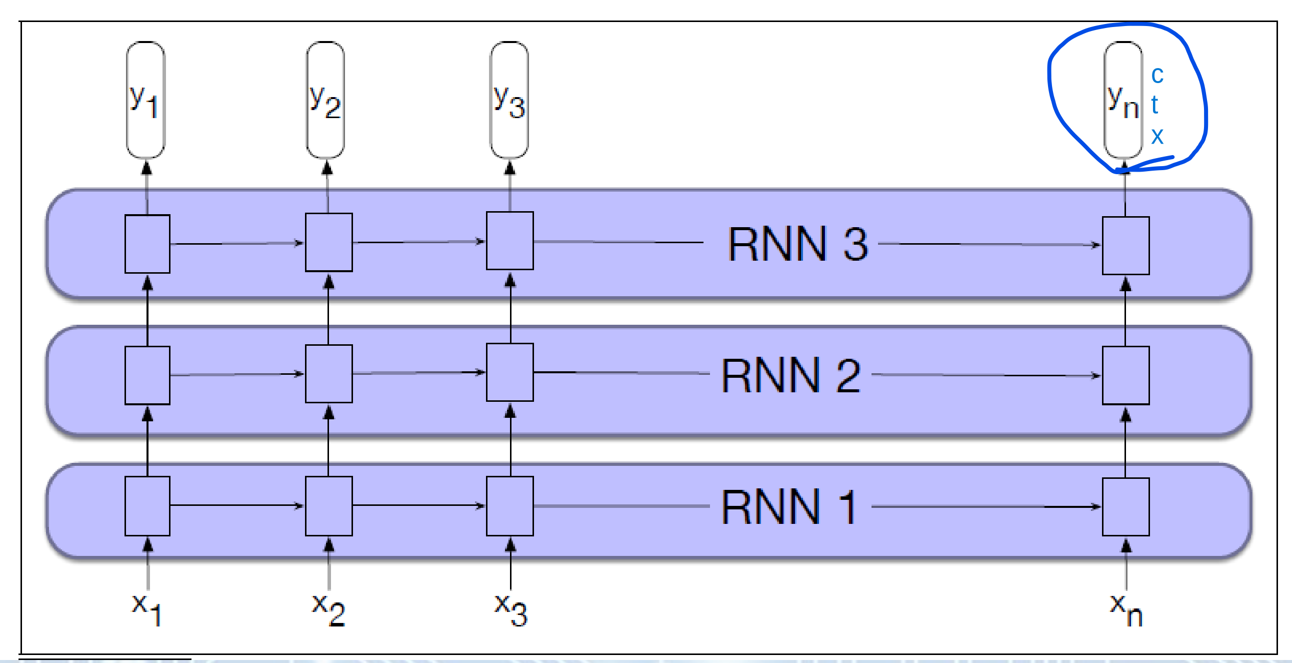
how we apply the backward propagaation in the decoder or in the training phase of the transfromer.



- The elements of the network on the left process the input sequence x and comprise the **encoder**.
- While our simplified figure shows only a single network layer for the encoder, stacked architectures are the norm, where the output states from the top layer of the stack are taken as the final representation.

Stacked RNNs

- Stacked RNNs consist of multiple networks where the **output of a lower level** serves as the **input to higher levels** with the output of the last network serving as the final output.



bn

assume we insert the following abdelaziz salah mohammed => v1

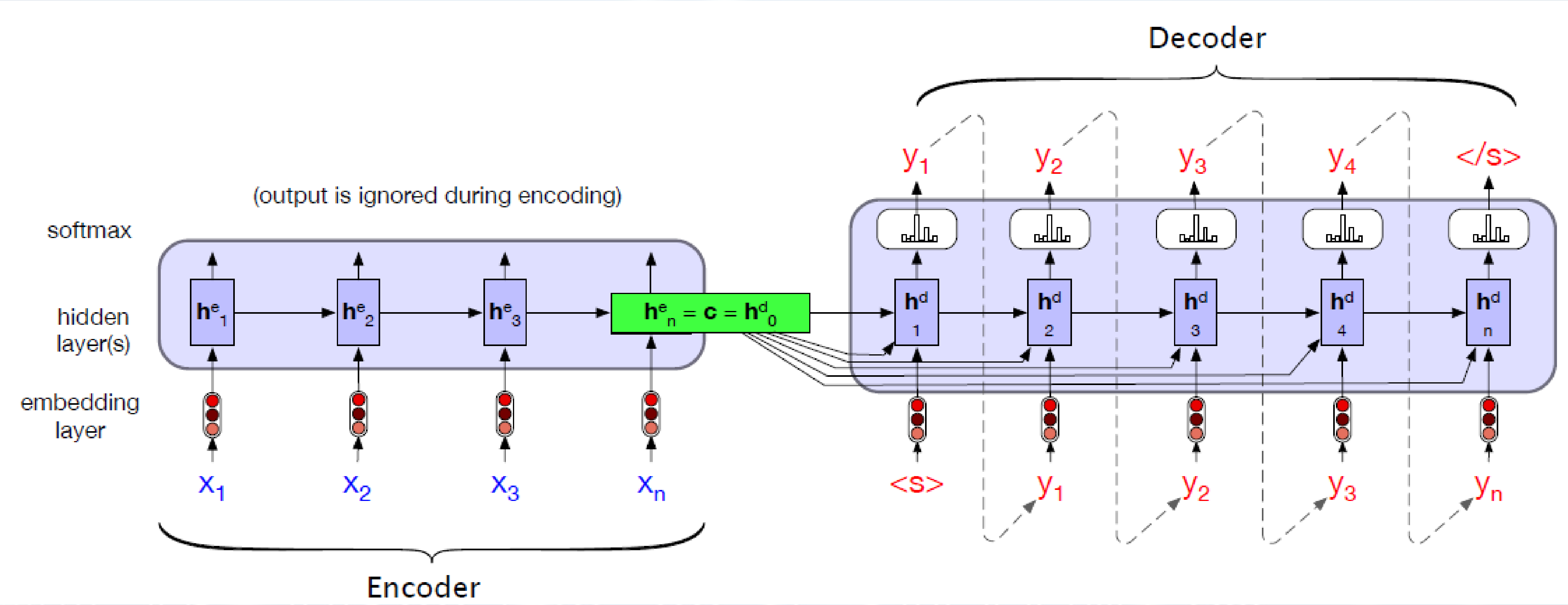
we insert v1 on the first RNN encoder. it generates v2 of same size as v1, after applying some math.

we take v2 and insert it into the second RNN, and so on.

our context will be the result of the top most layer at the last layer.

- Stacked RNNs generally outperform single-layer networks.
  - One reason for this success seems to be that the network induces representations at differing levels of abstraction across layers. Just as the early stages of the human visual system detect edges that are then used for finding larger regions and shapes, the initial layers of stacked networks can induce representations that serve as useful abstractions for further layers—representations that might prove difficult to induce in a single RNN.
- The optimal number of stacked RNNs is specific to each application and to each training set.
- However, as the number of stacks is increased the training costs rise quickly.

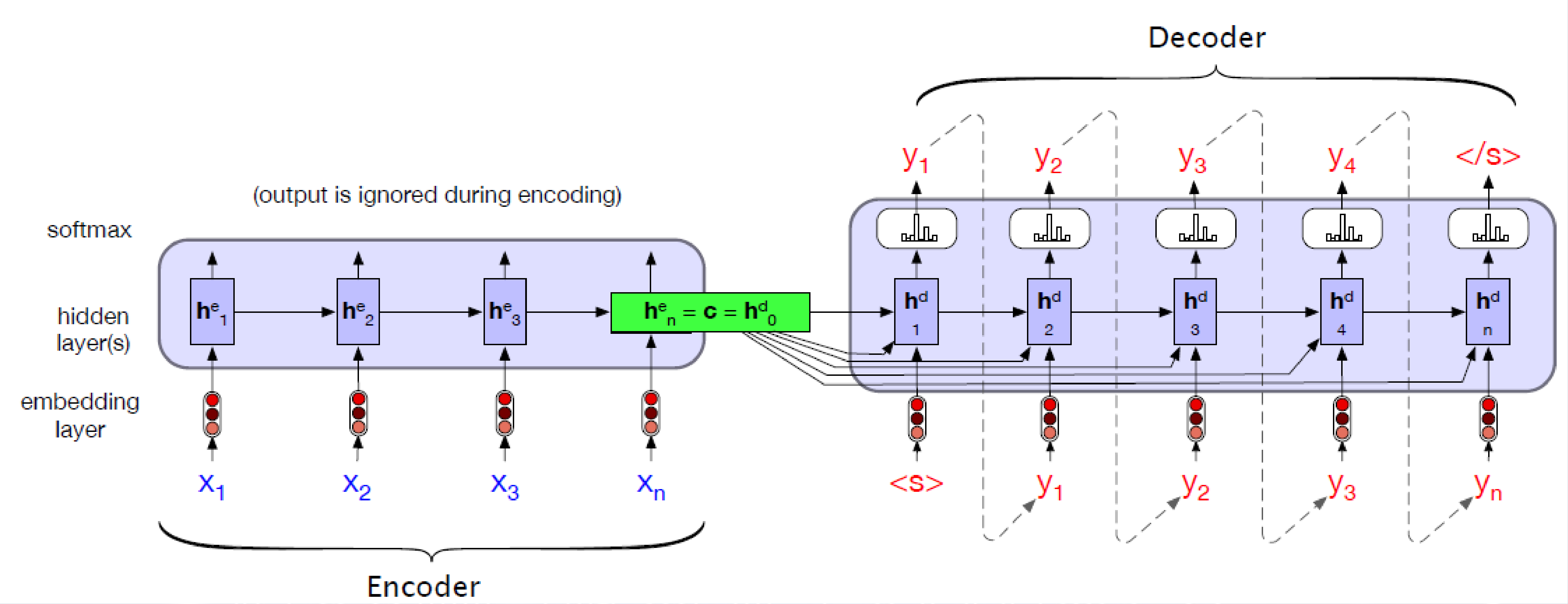
Encoder-Decoder with RNNs



- The entire purpose of the encoder is to generate a contextualized representation of the input. This representation is embodied in the final hidden state of the encoder $\boldsymbol{h_n^e}$ called ***c*** for **context**, is then passed to the decoder.

- The **decoder** network on the right takes this state and uses it to initialize the first hidden state of the decoder. That is, the first decoder RNN cell uses c as its prior hidden state $\boldsymbol{h_0^d}$.

Encoder-Decoder with RNNs



- The decoder autoregressively generates a sequence of outputs, an element at a time, until an **end-of-sequence marker** is generated.

- Each hidden state is conditioned on the previous hidden state and the output generated in the previous state.

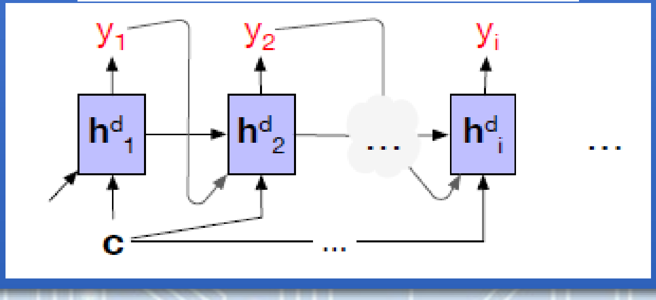
Encoder-Decoder with RNNs

- One weakness of this approach as described so far is that the influence of the context vector c will decrease as the output sequence is generated.

Solution:

Make the context vector **c** available at each step in the decoding process by adding it as a parameter to the computation of the current hidden state:

$$\mathbf{h}_t^d = g(\hat{y}_{t-1}, \mathbf{h}_{t-1}^d, \mathbf{c})$$



- Full equations for this version of the decoder in the basic encoder-decoder model:

$$\begin{aligned}
\mathbf{c} &= \mathbf{h}_n^e \\
\mathbf{h}_0^d &= \mathbf{c} \\
\mathbf{h}_t^d &= g(\boxed{\hat{y}_{t-1},}\mathbf{h}_{t-1}^d, \mathbf{c}) \\
\mathbf{z}_t &= f(\mathbf{h}_t^d) \\
y_t &= \text{softmax}(\mathbf{z}_t)
\end{aligned}$$

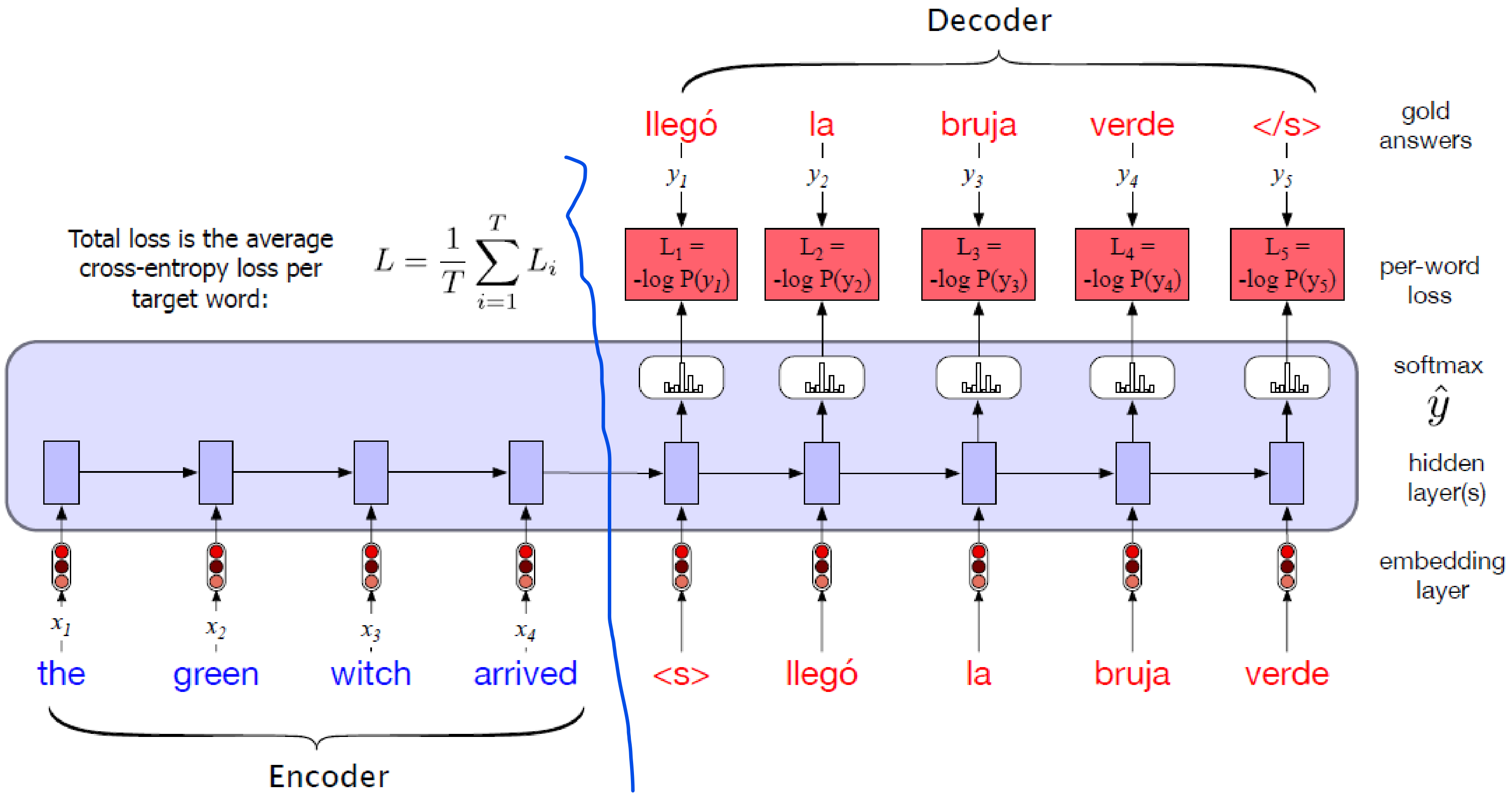
is the embedding for the output sampled from the softmax at the previous step

Training the Encoder-Decoder Model

- Each training example is a tuple of paired strings, a source and a target.
- Concatenated with a separator token, these source-target pairs serve as training data.
- The training itself proceeds as with any RNN-based language model.
- The network is given the source text and then starting with the separator token is trained autoregressively to predict the next word.

In the decoder:

- **During inference:** it uses its own estimated output $\hat{y}_t$ as the input for the next time step x_{t+1}.
- **During Training:** we usually don't propagate the model's softmax outputs, but use **teacher forcing** to force each input to the correct gold value→ this speeds up training. We compute the softmax output distribution over $\hat{y}_t$ to compute the loss at each token.



Attention

- The simplicity of the encoder-decoder model is its clean separation of the encoder—which builds a representation of the source text—from the decoder, which uses this context to generate a target text.

  →this context vector is ***hn***, the hidden state of the last (nth) time step of the source text.

- This final hidden state is thus acting as a **bottleneck**: it must represent absolutely everything about the meaning of the source text, since the only thing the decoder knows about the source text is what's in this context vector.

- Information at the beginning of the sentence, especially for long sentences, may not be equally well represented in the context vector.

Solution:

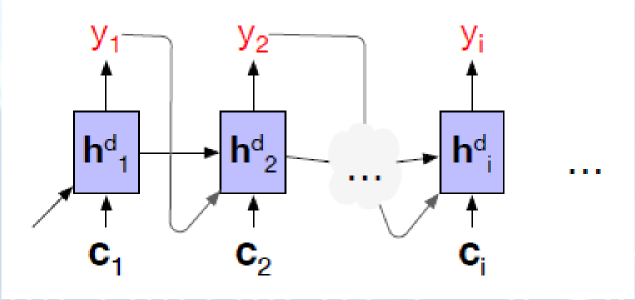
attention mechanism a way of mechanism allowing the decoder to get information from all the hidden states of the encoder, not just the last hidden state.

Attention

- Create a **fixed-length** vector c by taking a **weighted sum** of all the encoder hidden states.
- The weights focus on ('attend to') a particular part of the source text that is relevant for the token the decoder is currently producing.

Attention thus replaces the static context vector with one that is **dynamic**, different for each token in decoding

$$\mathbf{h}_i^d = g(\hat{y}_{i-1}, \mathbf{h}_{i-1}^d, \mathbf{c}_i)$$



- The first step in computing c_i is to compute how much to focus on each encoder state →how relevant each encoder state is to the decoder state captured in h_{i-1}^d

Attention

- To capture relevance: at each state *i* during decoding, a score for each encoder state *j* is computed $\boldsymbol{score(h_{i-1}^{d}, h_j^{e})}$
- The simplest such score **"dot-product attention"**

$$score(\mathbf{h}_{i-1}^{d}, \mathbf{h}_j^{e}) = \mathbf{h}_{i-1}^{d} \cdot \mathbf{h}_j^{e}$$

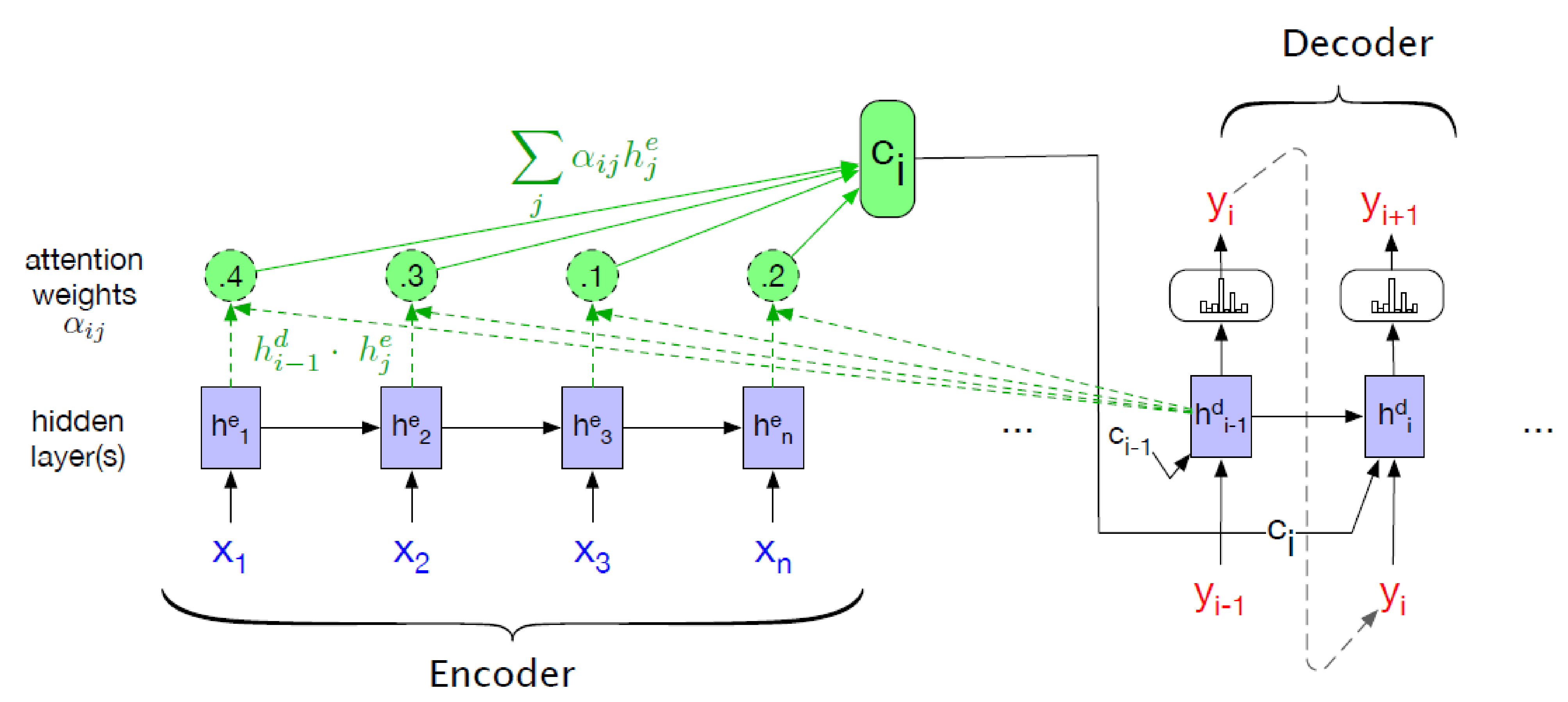
- The score that results from this dot product is a scalar that reflects the degree of similarity between the two vectors.
- The vector of these scores across all the encoder hidden states gives us the relevance of each encoder state to the current step of the decoder.
- We normalize these scores with a softmax to create a vector of weights α_{ij}, that tells us the **proportional relevance** of each encoder hidden state *j* to the prior hidden decoder state h_{i-1}^{d} .

$$\begin{aligned}\alpha_{ij} &= \mathrm{softmax}(score(\mathbf{h}_{i-1}^{d}, \mathbf{h}_j^{e})\ \forall j \in e) \\ &= \frac{\exp(score(\mathbf{h}_{i-1}^{d}, \mathbf{h}_j^{e}))}{\sum_k \exp(score(\mathbf{h}_{i-1}^{d}, \mathbf{h}_k^{e}))}\end{aligned}$$

Attention

i is the timestep of the decoder.

- The context vector for the current decoder state: $\mathbf{c}_i = \sum_j \alpha_{ij}\,\mathbf{h}^e_j$



Note: the encoder and decoder hidden states must have the same dimensionality.

Note

- The described decoding is **greedy:** at each time step in decoding, the output y_t is chosen by computing a softmax over the set of possible outputs (the vocabulary, in the case of language modeling or MT), and then choosing the **highest probability token** → the choice is **locally optimal**

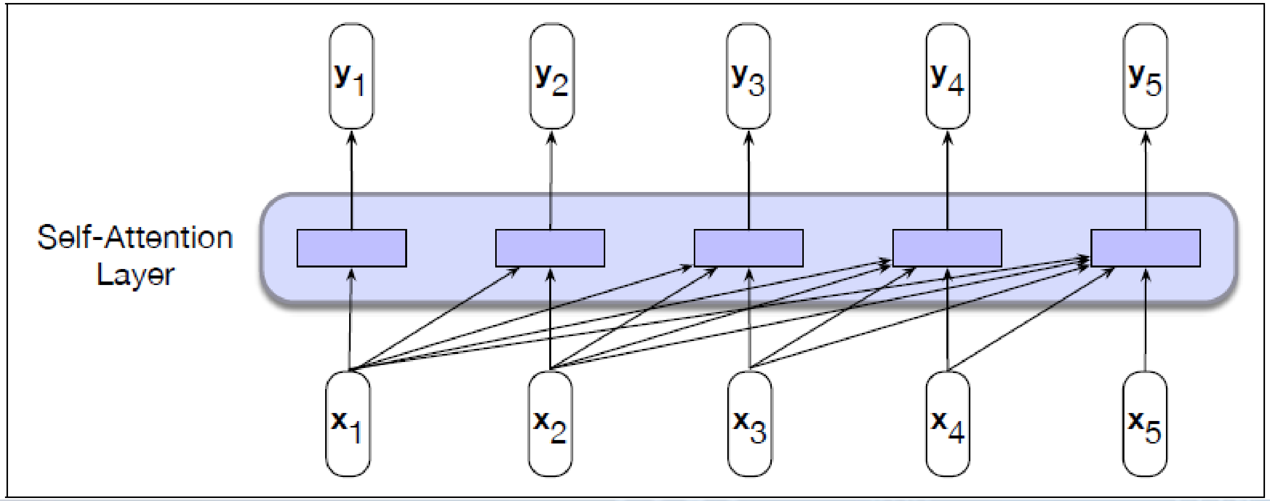
- Greedy search is not optimal, and may not find the highest probability translation →the problem is that the token that looks good to the decoder now might turn out later to have been the wrong choice!

- Another generally used decoding method in MT is called **beam search**. In beam search, instead of choosing the best token to generate at each timestep, we keep *k* possible tokens at each step.

Transformers

BERT alak baa en msh bs el mady hwa el byfr2 m3aya, laa kman el most2bl mohem, fa hnakhdu m3aana.

- Transformers are **Self-Attention Networks.**
- Can handle distant information without using recurrent connections (which can be hard to parallelize), which means that transformers can be **more efficient to implement at scale**.
- Map sequences of input vectors (x1,…,xn) to sequences of output vectors (y1,…,yn) of the same length.
- Transformers are made up of stacks of **transformer blocks**, each of which is a multilayer network made by combining: simple linear layers, feedforward networks, and self-attention layers →the key innovation of transformers.



In processing each element of the sequence, **the model attends to all the inputs up to, and including, the current one**.
Unlike RNNs, the computations at each time step are independent of all the other steps and therefore can be performed in parallel.

Thank You