Neural Machine Translation Model

Seq2seq Models with Attention

Some slides were adated/taken from various sources, including Andrew Ng's Coursera Lectures, CS231n: Convolutional Neural Networks for Visual Recognition lectures, Stanford University CS Waterloo Canada lectures, Aykut Erdem, et.al. tutorial on Deep Learning in Computer Vision, Jay Alamnar's BLOGs, Stanford CS224N Course, Umar Jamil's video lecture, Ismini Lourentzou's lecture slide on "Introduction to Deep Learning", Ramprasaath's lecture slides, and many more. We thankfully acknowledge them. Students are requested to use this material for their study only and NOT to distribute it.

Outlines

- Sequence to sequence model
- Neural Machine Translation
- Attention
- Sequence to sequence model with Attention

Seq2seq Models

- Sequence-to-sequence models are deep learning models that have achieved a lot of success in tasks like machine translation, text summarization, and image captioning.

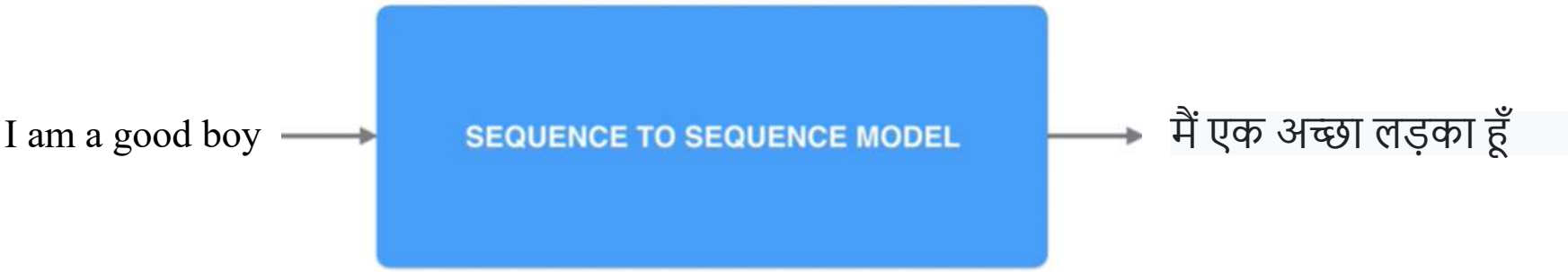
- Google Translate started using such a model in production in late 2016.

- These models are explained in the two pioneering papers (Sutskever et al., 2014, Cho et al., 2014).

Sutskever et al., 2014: https://proceedings.neurips.cc/paper_files/paper/2014/file/a14ac55a4f27472c5d894ec1c3c743d2-Paper.pdf
Cho et al., 2014: https://emnlp2014.org/papers/pdf/EMNLP2014179.pdf

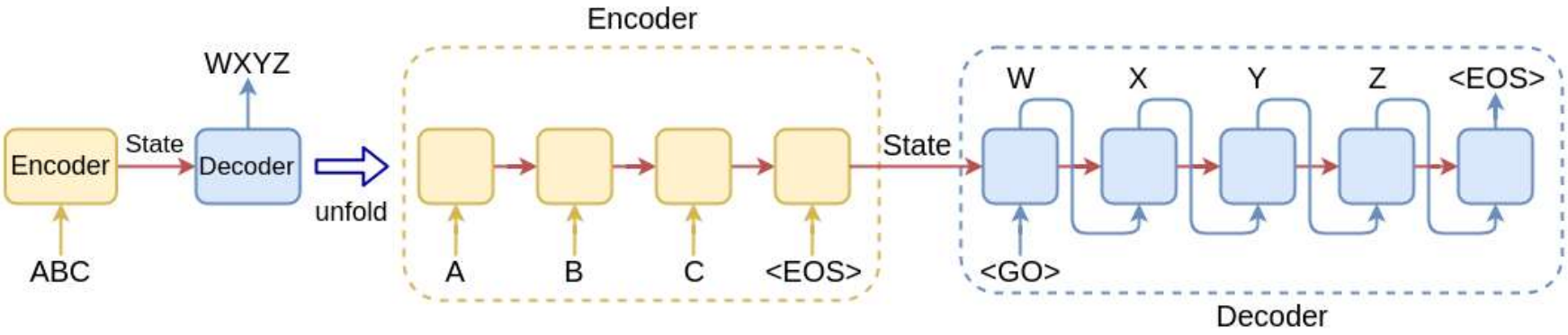
Seq2seq Models

A sequence-to-sequence model is a model that takes a sequence of items (words, letters, features of an images…etc) and outputs another sequence of items. A trained model would work like this:



Model Architecture

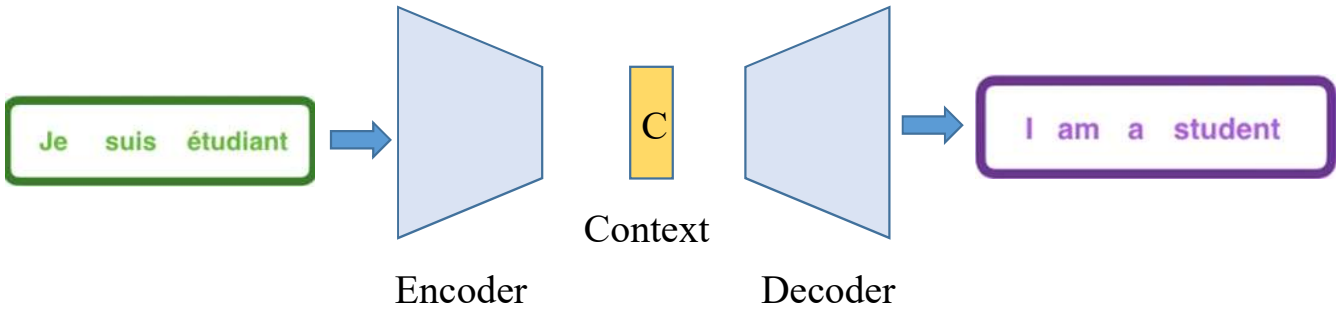
- An encoder-decoder model



Seq2seq Models

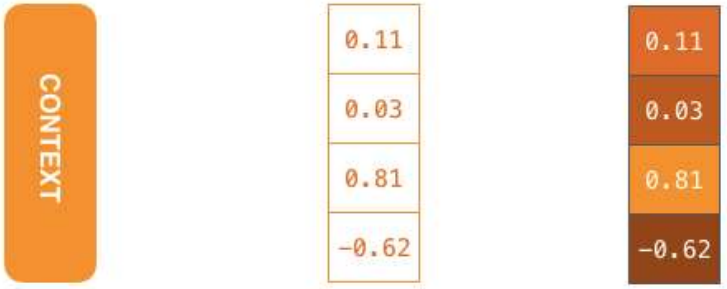
In neural machine translation, a sequence is a series of words, processed one after another. The output is, likewise, a series of words

The encoder processes each item in the input sequence, it compiles the information it captures into a vector (called the context). After processing the entire input sequence, the encoder sends the context over to the decoder, which begins producing the output sequence item by item.



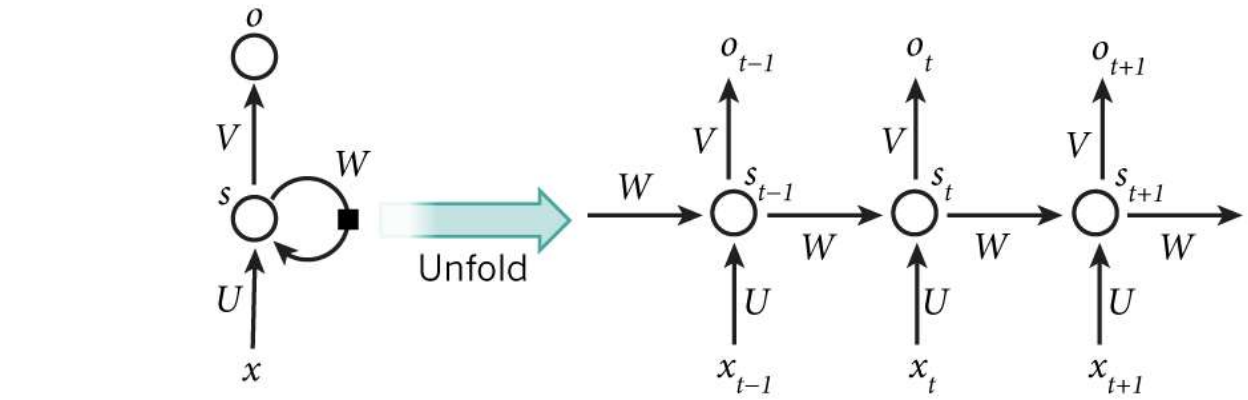
Context

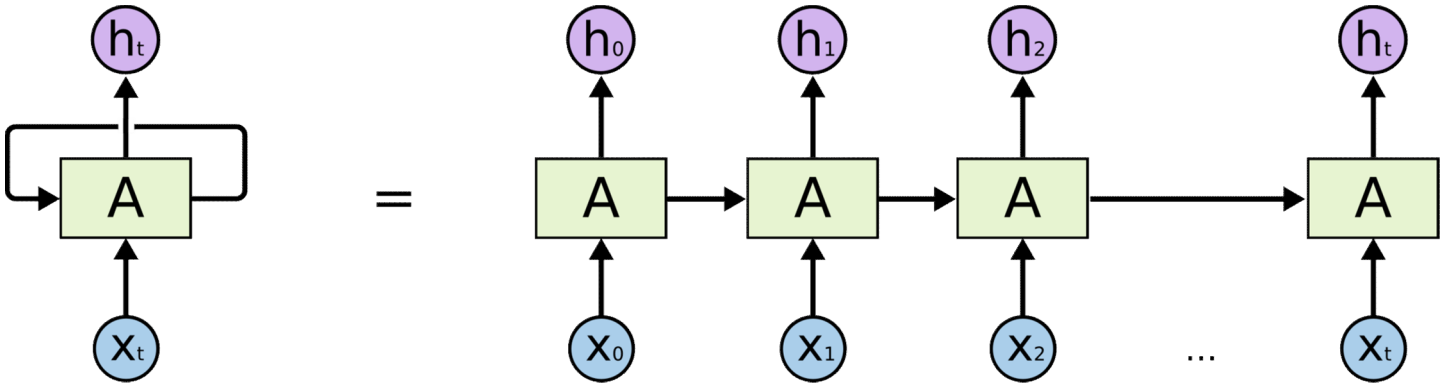
The context is a vector (an array of numbers, basically) in the case of machine translation. The encoder and decoder tend to both be recurrent neural networks.



You can set the size of the context vector when you set up your model. It is basically the number of hidden units in the encoder RNN. In our example, the vector is of size 4, but in real world applications the context vector would be of a size like 256, 512, or 1024.

Recurrent Neural Network





Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

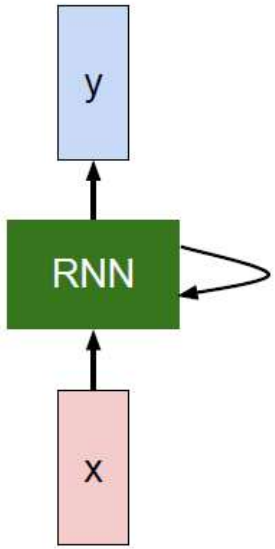
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

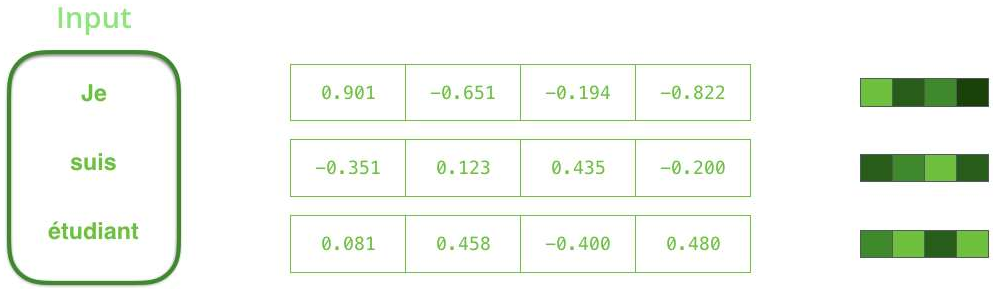
input vector at some time step



Slide Credit: Fei-Fei Li & Justin Johnson & Serena Yeung

When the Encoder is a RNN

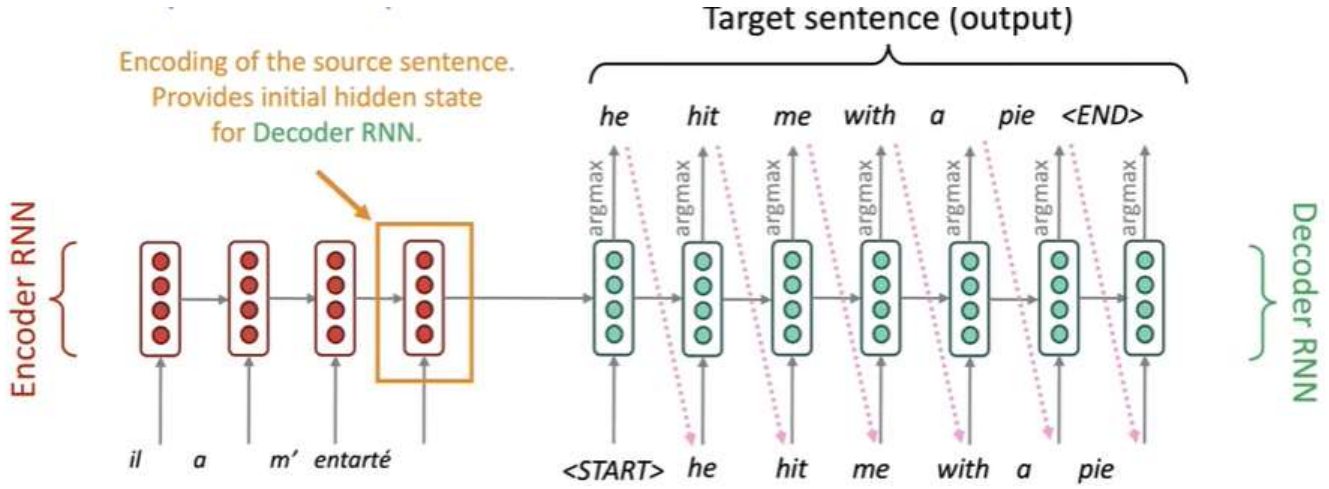
- By design, a RNN takes two inputs at each time step: an input (in the case of the encoder, one word from the input sentence), and a hidden state.
- The word, however, needs to be represented by a vector. To transform a word into a vector, we turn to the class of methods called "word embedding" algorithms.
- These turn words into vector spaces that capture a lot of the meaning/semantic information of the words (e.g. king - man + woman = queen).



We need to turn the input words into vectors before processing them. That transformation is done using a word embedding algorithm. We can use pre-trained embeddings or train our own embedding on our dataset. Embedding vectors of size 200 or 300 are typical, we're showing a vector of size four for simplicity.

Seq2seq Models

- In the following figure, each pulse for the encoder or decoder is that RNN processing its inputs and generating an output for that time step.
- Since the encoder and decoder are both RNNs, each time step one of the RNNs does some processing, it updates its hidden state based on its inputs and previous inputs it has seen.
- The last hidden state is actually the context, we pass along to the decoder.



The Encoder RNN produces an encoding of the source sentence

The Decoder RNN is a language model that generates target sentence, conditioned on Encoding

Neural Machine Translation (NMT)

- Sequence-to-sequence is useful for *more than just MT*

- Many NLP tasks can be phrased as sequence-to-sequence:
  - Summarization (long text → short text)
  - Dialogue (previous utterances → next utterance)
  - Parsing (input text → output parse as sequence)
  - Code generation (natural language → Python code)

Neural Machine Translation (NMT)

- The sequence-to-sequence model is an example of a **Conditional Language Model**
  - **Language Model** because the decoder is predicting the next word of the target sentence *y*
  - **Conditional** because its predictions are *also* conditioned on the source sentence *x*

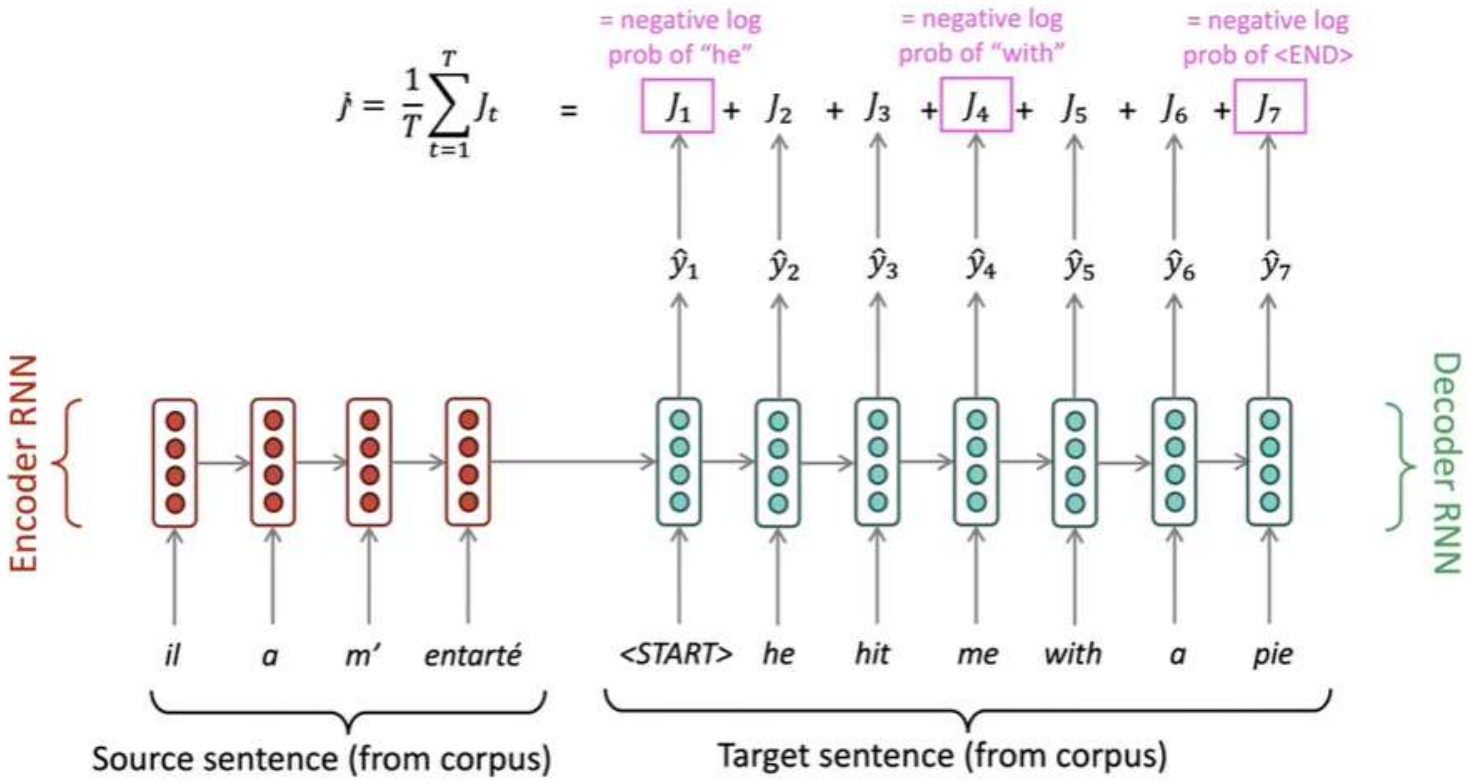
- NMT directly calculates $P(y|x)$:

$$P(y|x) = P(y_1|x)\, P(y_2|y_1, x)\, P(y_3|y_1, y_2, x) \ldots \underbrace{P(y_T|y_1, \ldots, y_{T-1}, x)}$$

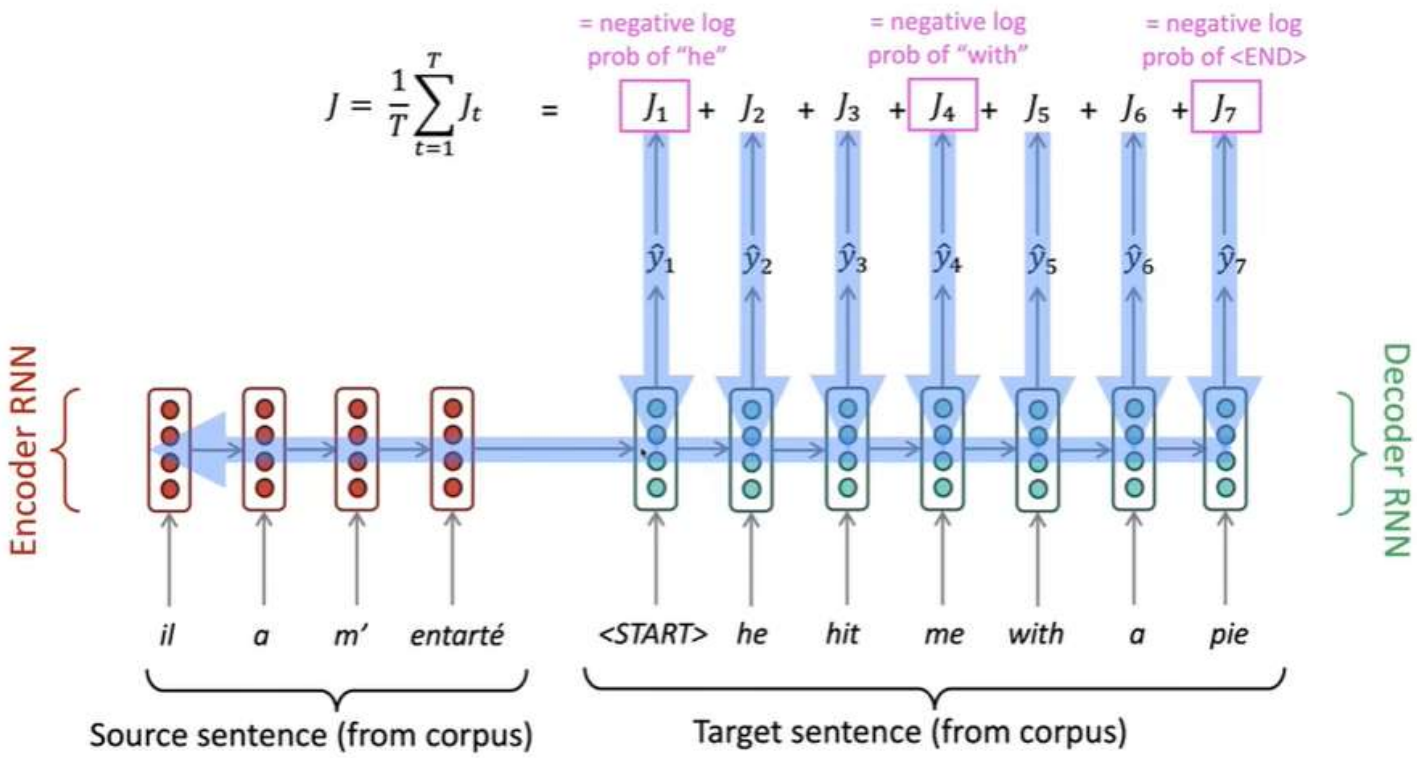
Probability of next target word, given target words so far and source sentence *x*

- **Question**: How to train a NMT system?
- **Answer**: Get a big parallel corpus…

Neural Machine Translation (NMT)



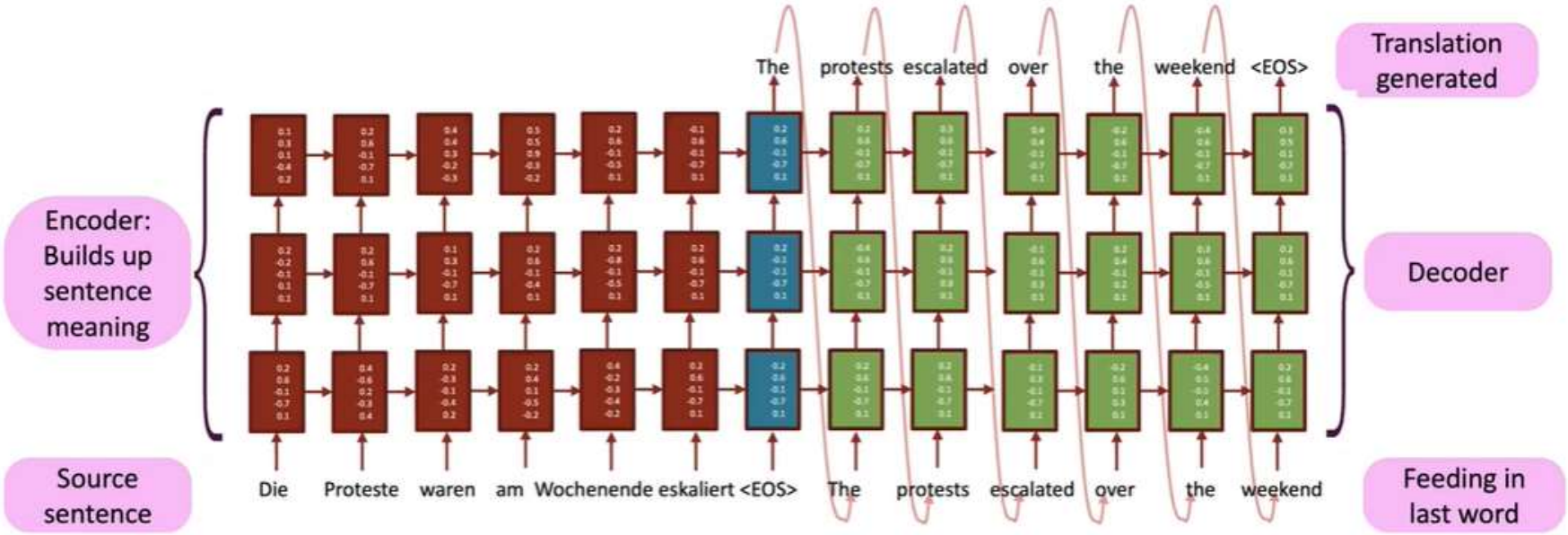
Neural Machine Translation (NMT)



Multi-layer deep encoder-decoder machine translation net

[Sutskever et al. 2014; Luong et al. 2015]

The hidden states from RNN layer *i* are the inputs to RNN layer *i*+1



Exhaustive Search Decoding

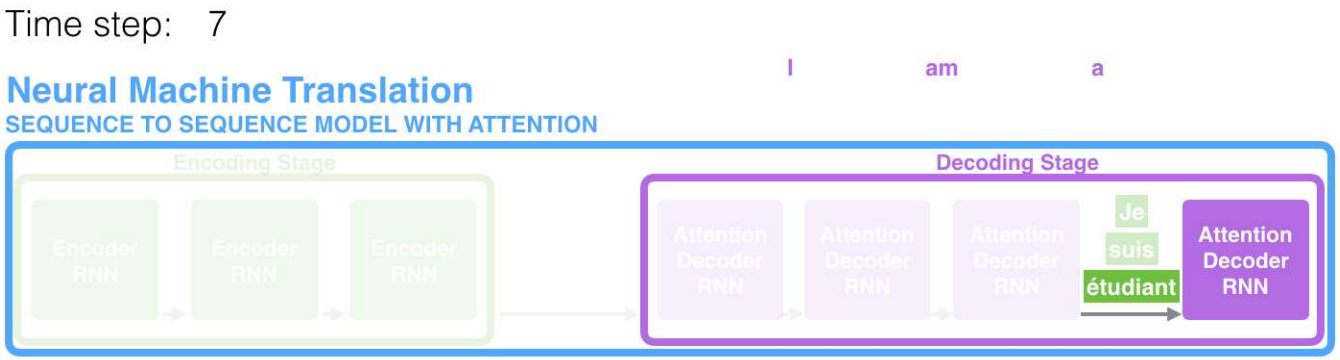
- Ideally, we want to find a (length *T*) translation *y* that maximizes

$$P(y|x) = P(y_1|x)\,P(y_2|y_1,x)\,P(y_3|y_1,y_2,x)\ldots,P(y_T|y_1,\ldots,y_{T-1},x)$$
$$= \prod_{t=1}^{T} P(y_t|y_1,\ldots,y_{t-1},x)$$

- We could try computing all possible sequences *y*
  - This means that on each step *t* of the decoder, we're tracking V^t possible partial translations, where *V* is vocab size
  - This $O(V^T)$ complexity is far too expensive!

Attention

- The context vector turned out to be a bottleneck for these types of models. It made it challenging for the models to deal with long sentences.
- A solution was proposed in Bahdanau et al., 2014 and Luong et al., 2015.
- These papers introduced and refined a technique called “Attention”, which highly improved the quality of machine translation systems.
- Attention allows the model to focus on the relevant parts of the input sequence as needed.

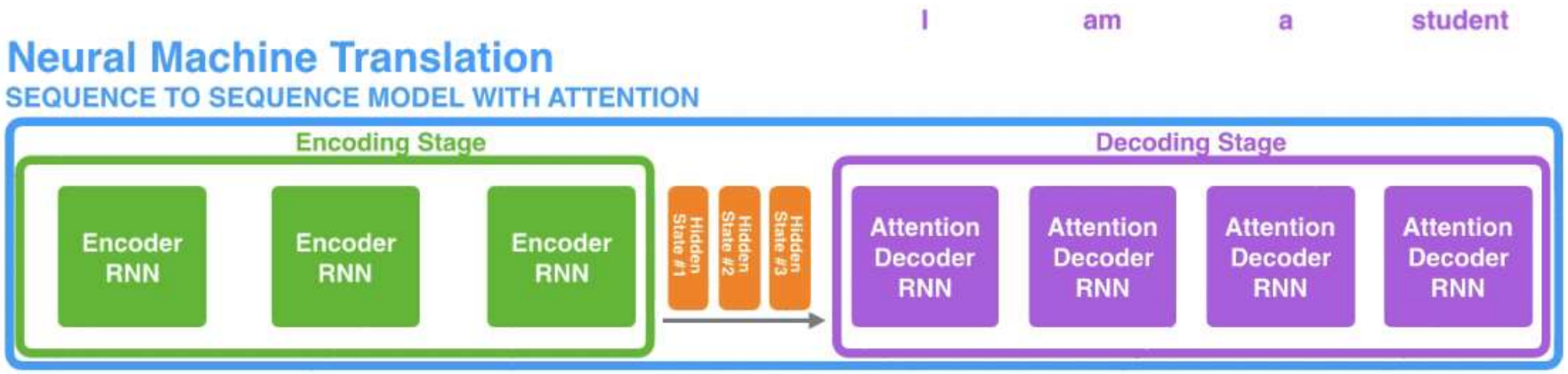


At time step 7, the attention mechanism enables the decoder to focus on the word "étudiant" ("student" in french) before it generates the English translation. This ability to amplify the signal from the relevant part of the input sequence makes attention models produce better results than models without attention.

Bahdanau et al., 2014: https://arxiv.org/abs/1409.0473 #### Luong et al., 2015: https://arxiv.org/abs/1508.04025

Attention

- Let's continue looking at attention models at this high level of abstraction. An attention model differs from a classic sequence-to-sequence model in two main ways:
- First, the encoder passes a lot more data to the decoder. Instead of passing the last hidden state of the encoding stage, the encoder passes *all* the hidden states to the decoder:



Attention

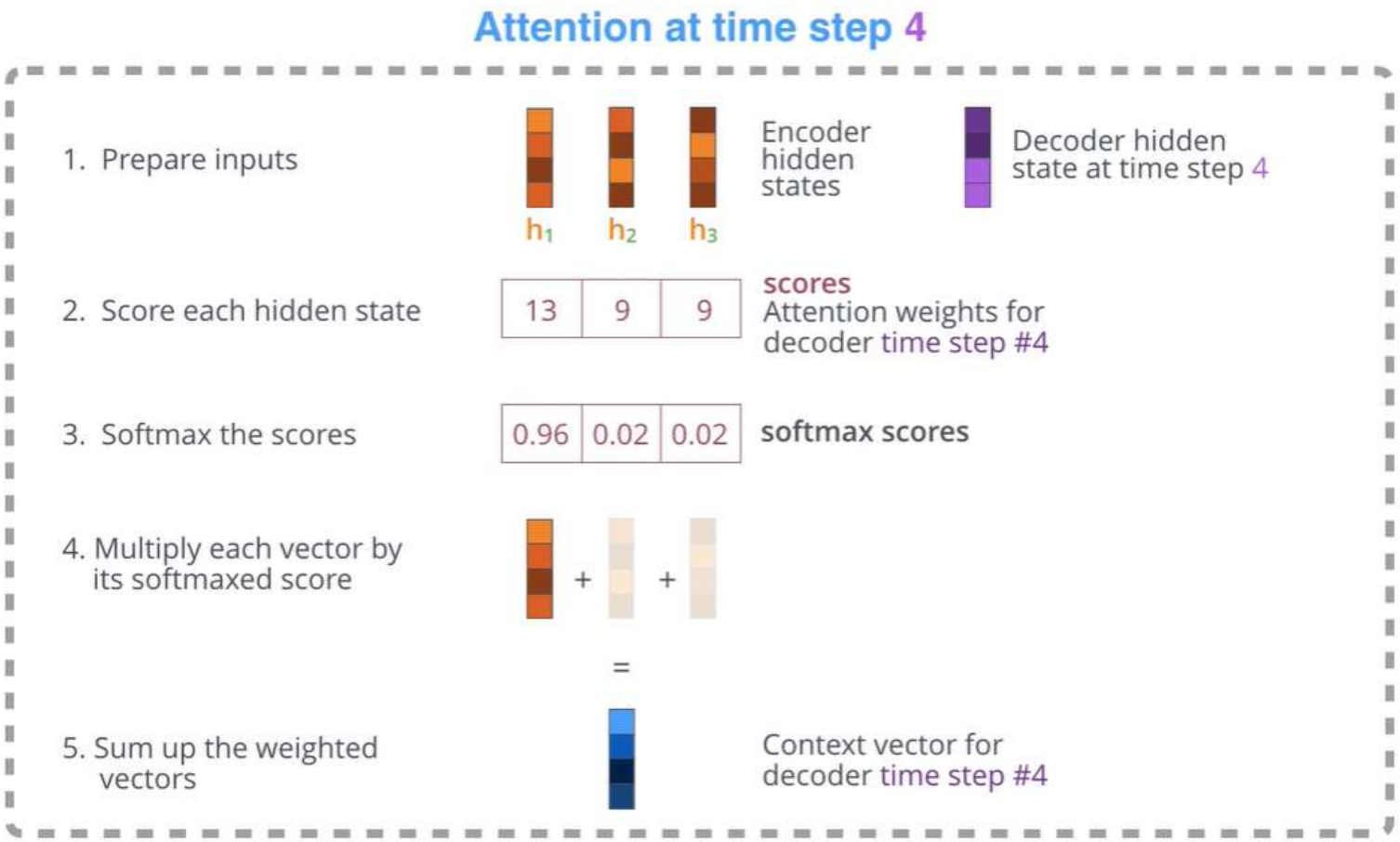
- Second, an attention decoder does an extra step before producing its output. In order to focus on the parts of the input that are relevant to this decoding time step, the decoder does the following:

1. Look at the set of encoder hidden states it received – each encoder hidden state is most associated with a certain word in the input sentence

2. Give each hidden state a score (let's ignore how the scoring is done for now)

3. Multiply each hidden state by its softmaxed score, thus amplifying hidden states with high scores, and drowning out hidden states with low scores

Attention


Attention at time step 4
1. Prepare inputs
h1
h2
h3
Encoder hidden states
Decoder hidden state at time step 4
2. Score each hidden state
13
9
9
scores
Attention weights for decoder time step #4
3. Softmax the scores
0.96
0.02
0.02
softmax scores
4. Multiply each vector by its softmaxed score
+
+
=
5. Sum up the weighted vectors
Context vector for decoder time step #4

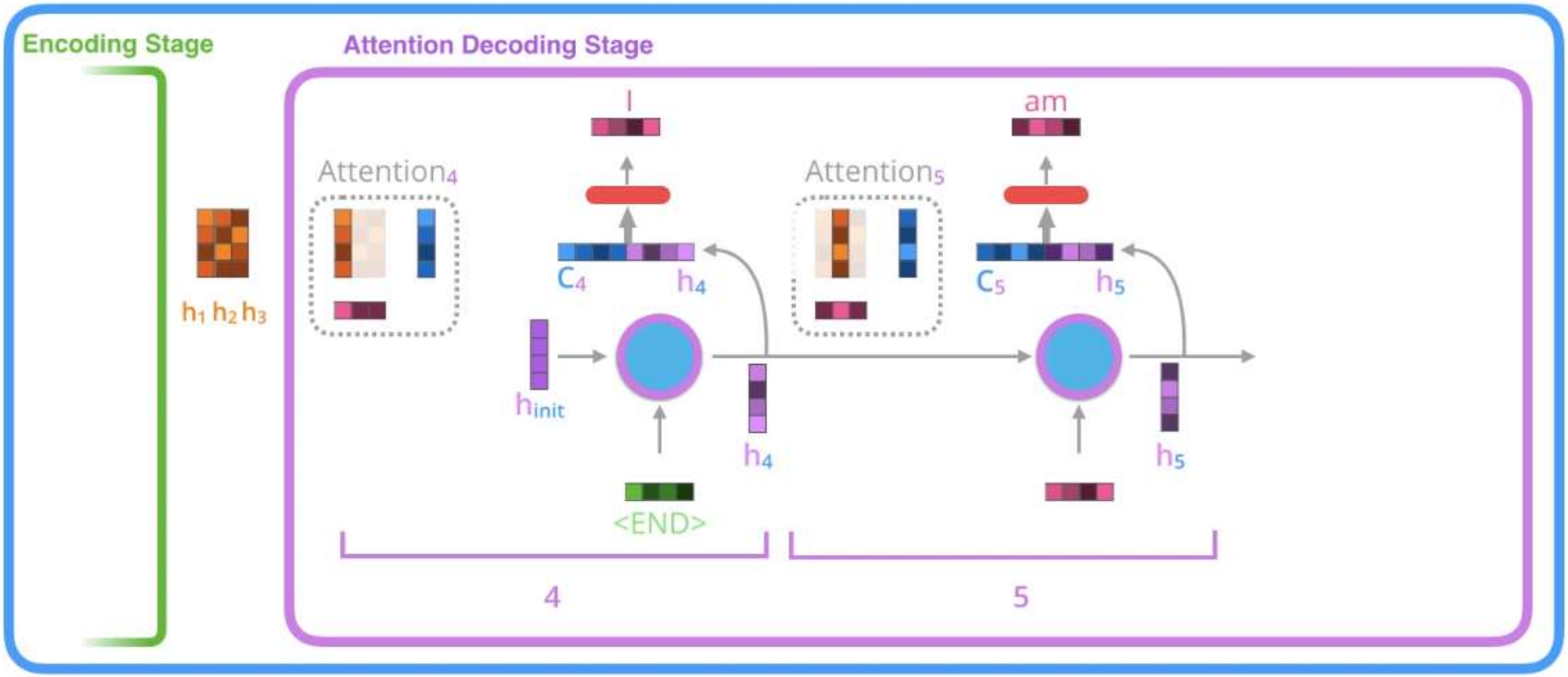
Attention

This scoring exercise is done at each time step on the decoder side.

1. The attention decoder RNN takes in the embedding of the <END> token, and an initial decoder hidden state.
2. The RNN processes its inputs, producing an output and a new hidden state vector (h4). The output is discarded.
3. Attention Step: We use the encoder hidden states and the h4 vector to calculate a context vector (C4) for this time step.
4. We concatenate h4 and C4 into one vector.
5. We pass this vector through a feedforward neural network (one trained jointly with the model).
6. The output of the feedforward neural networks indicates the output word of this time step.
7. Repeat for the next time steps

Neural Machine Translation

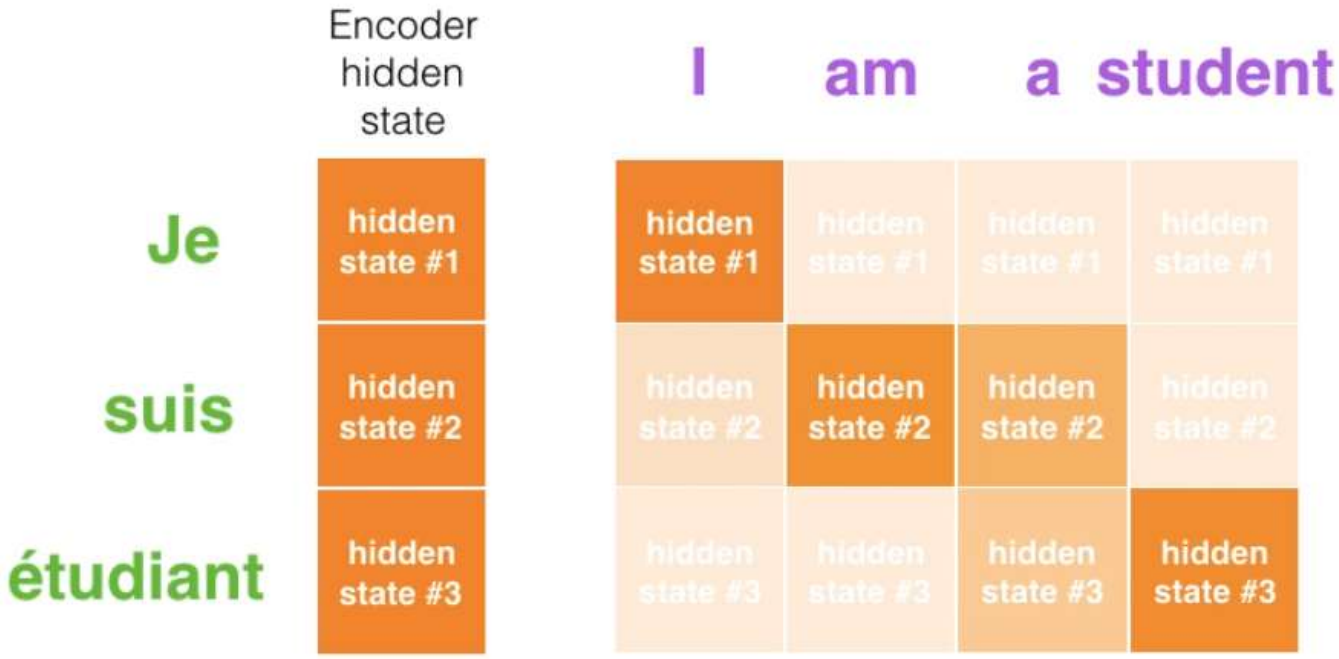
Sequence to sequence model with Attention



Neural Machine Translation

Sequence to sequence model with Attention

Another way to look at which part of the input sentence we're paying attention to at each decoding step



Neural Machine Translation

Sequence to sequence model with Attention

Note that the model isn't just mindless aligning the first word at the output with the first word from the input.

It actually learned from the training phase how to align words in that language pair (French and English in our example).

You can see how the model paid attention correctly when outputing "European Economic Area".

In French, the order of these words is reversed ("européenne économique zone") as compared to English.

Every other word in the sentence is in similar order.