Neural Machine Translation Model (Seq2seq Models With Attention)

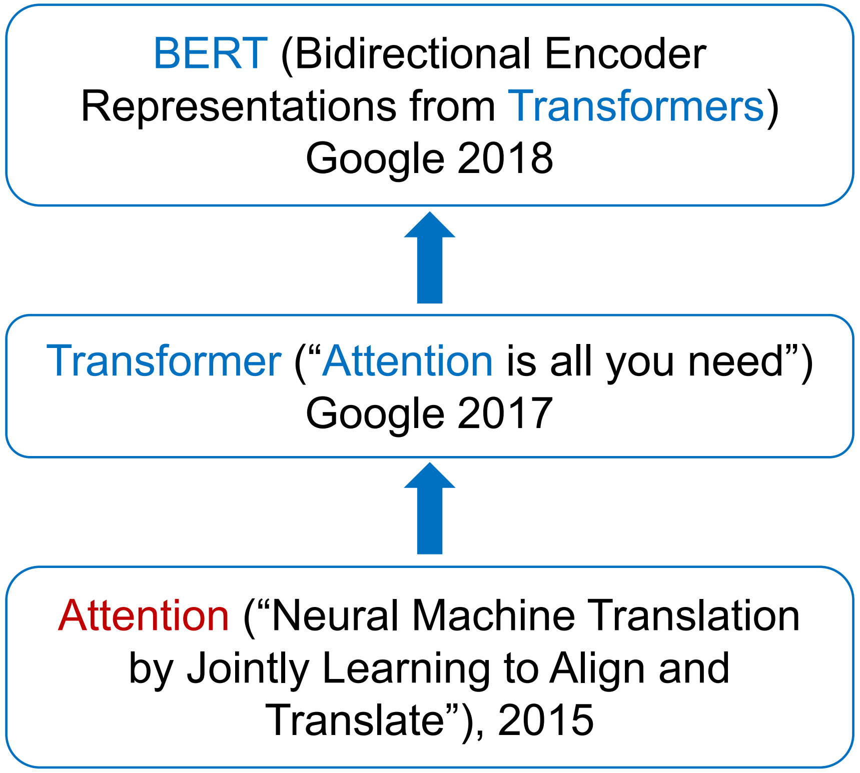
谢海华 助理研究员

北京雁栖湖应用数学研究院

大数据及人工智能

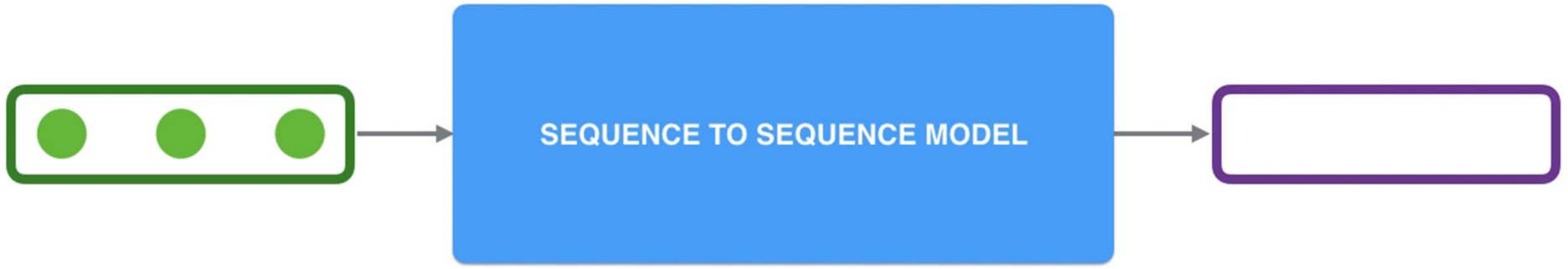
2022.04

The Attention Mechanism


BERT (Bidirectional Encoder Representations from Transformers) Google 2018
Transformer ("Attention is all you need") Google 2017
Attention ("Neural Machine Translation by Jointly Learning to Align and Translate"), 2015

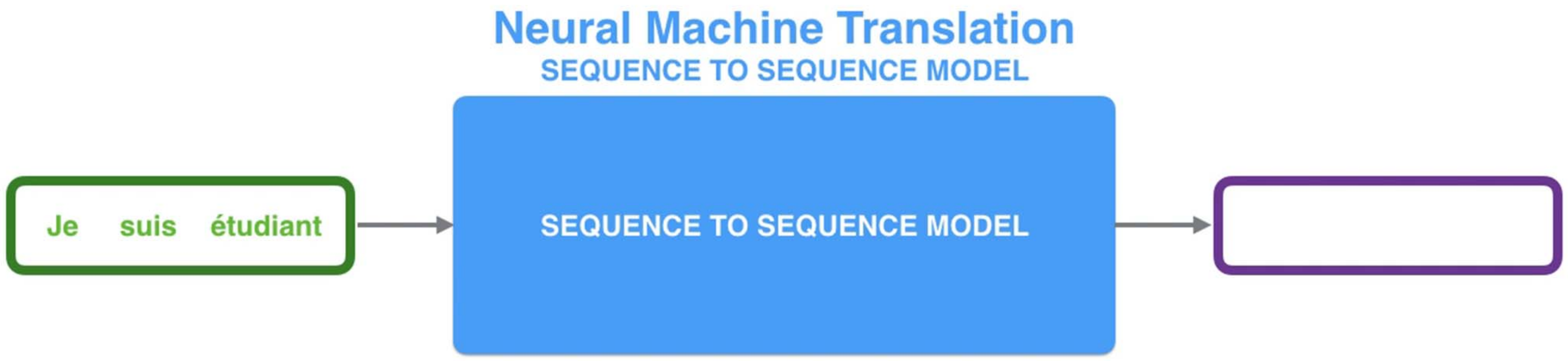
Sequence-to-sequence models

A sequence-to-sequence (seq2seq) 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:



Sequence-to-sequence models

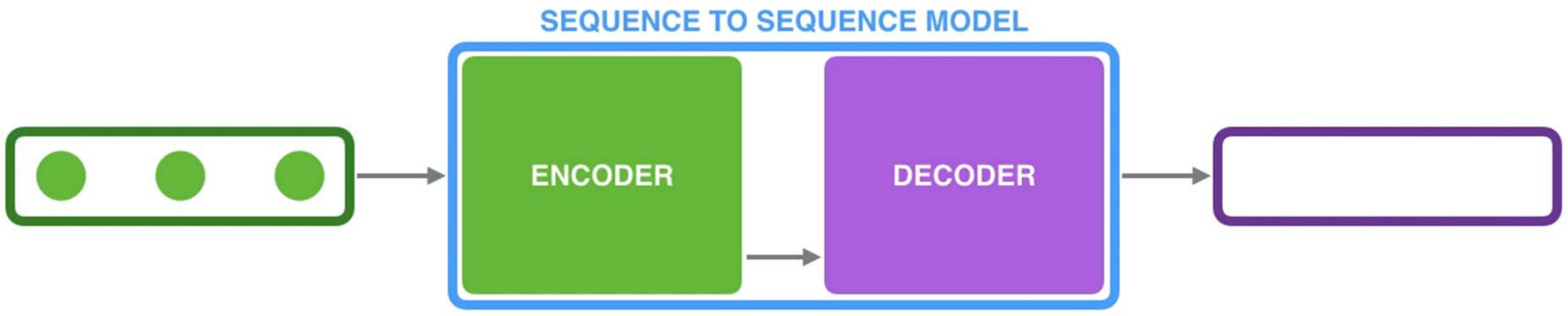
In neural machine translation, a sequence is a series of words, processed one after another. The output is, likewise, a series of words:


Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL
Je suis étudiant
SEQUENCE TO SEQUENCE MODEL

Looking under the hood

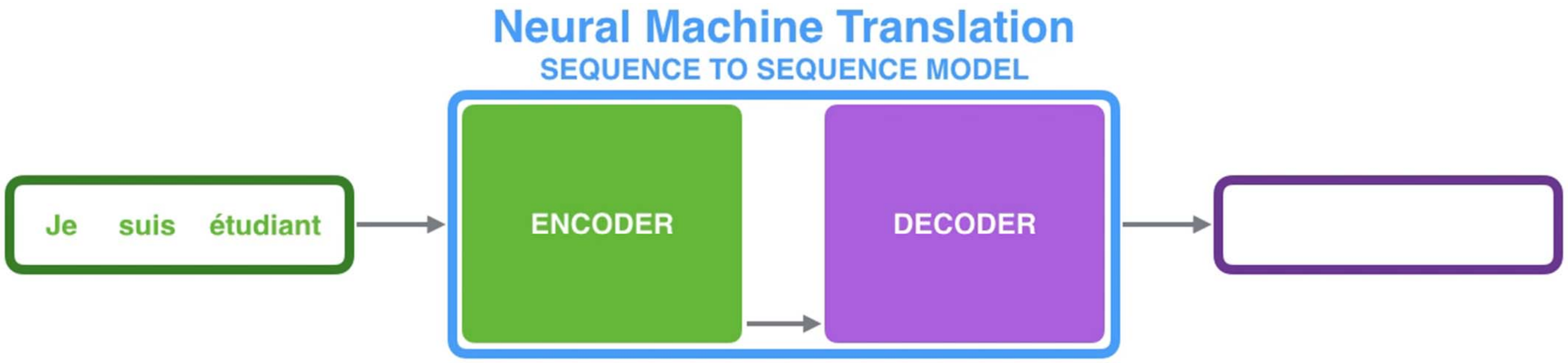
Under the hood, the model is composed of an encoder and a decoder.

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.



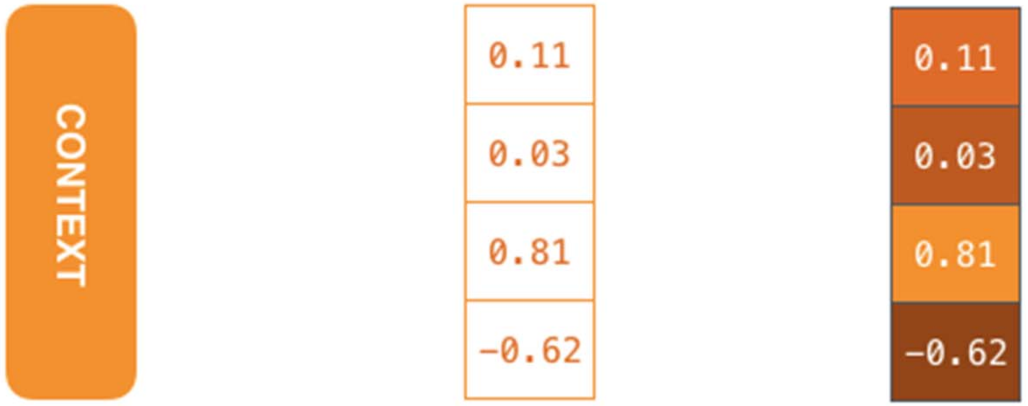
Looking under the hood

The same applies in the case of machine translation.



Looking under the hood

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 (Be sure to check out Luis Serrano's A friendly introduction to Recurrent Neural Networks for an intro to RNNs).

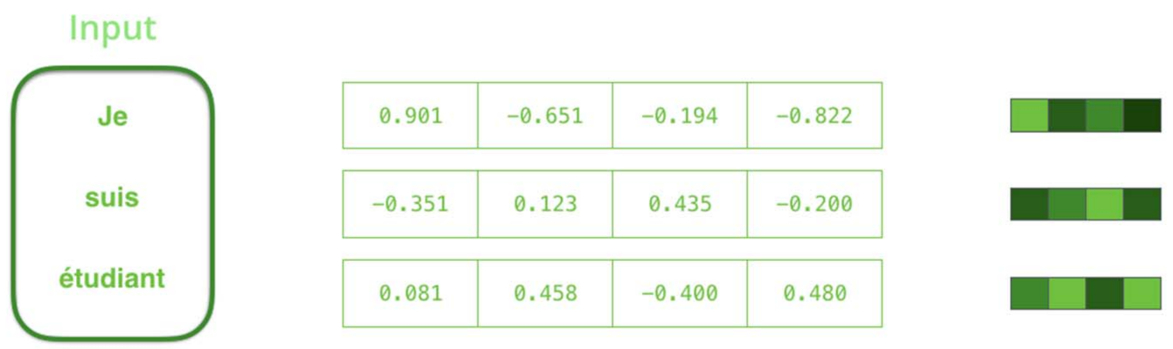


The context is a vector of floats. Later in this post we will visualize vectors in color by assigning brighter colors to the cells with higher values.

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. These visualizations show a vector of size 4, but in real world applications the context vector would be of a size like 256, 512, or 1024.

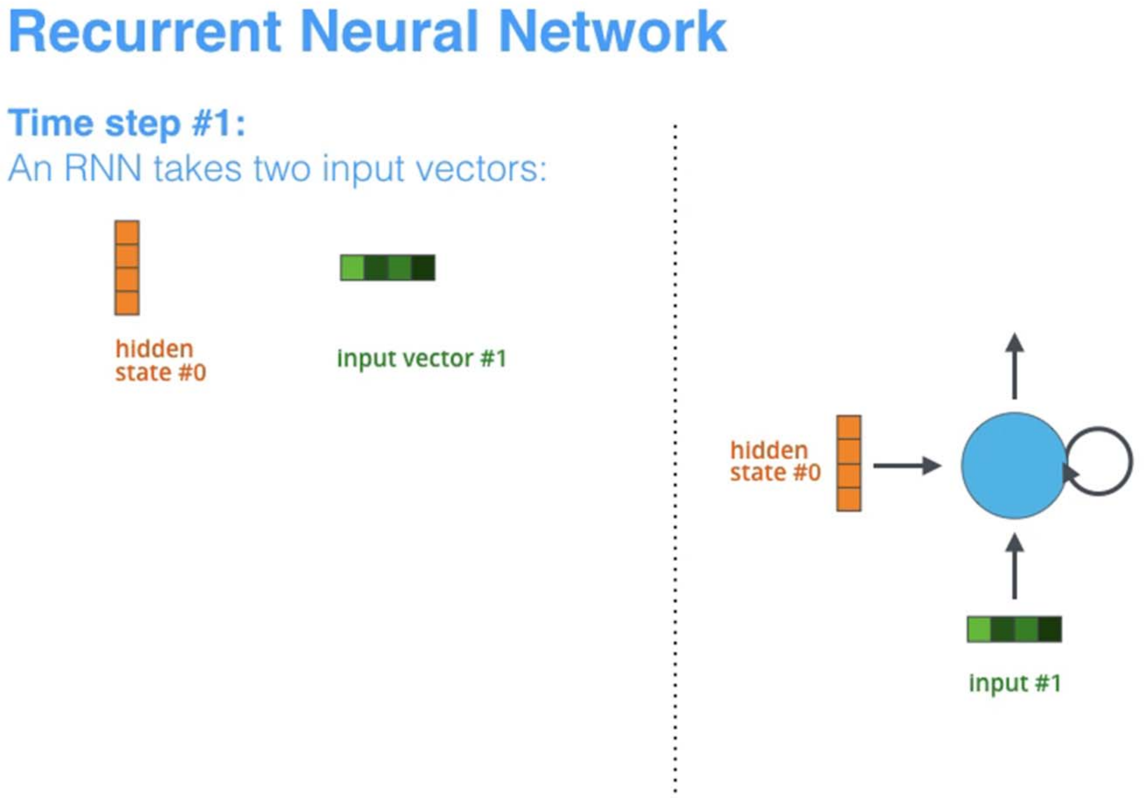
Recurrent Neural Network

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.

Recurrent Neural Network

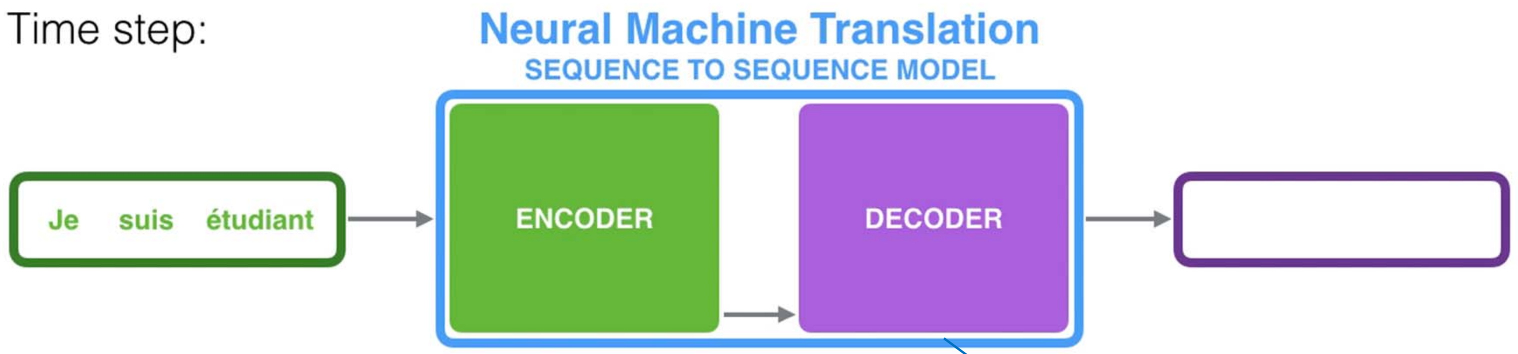


The next RNN step takes the second input vector and hidden state #1 to create the output of that time step.

Recurrent Neural Network

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.

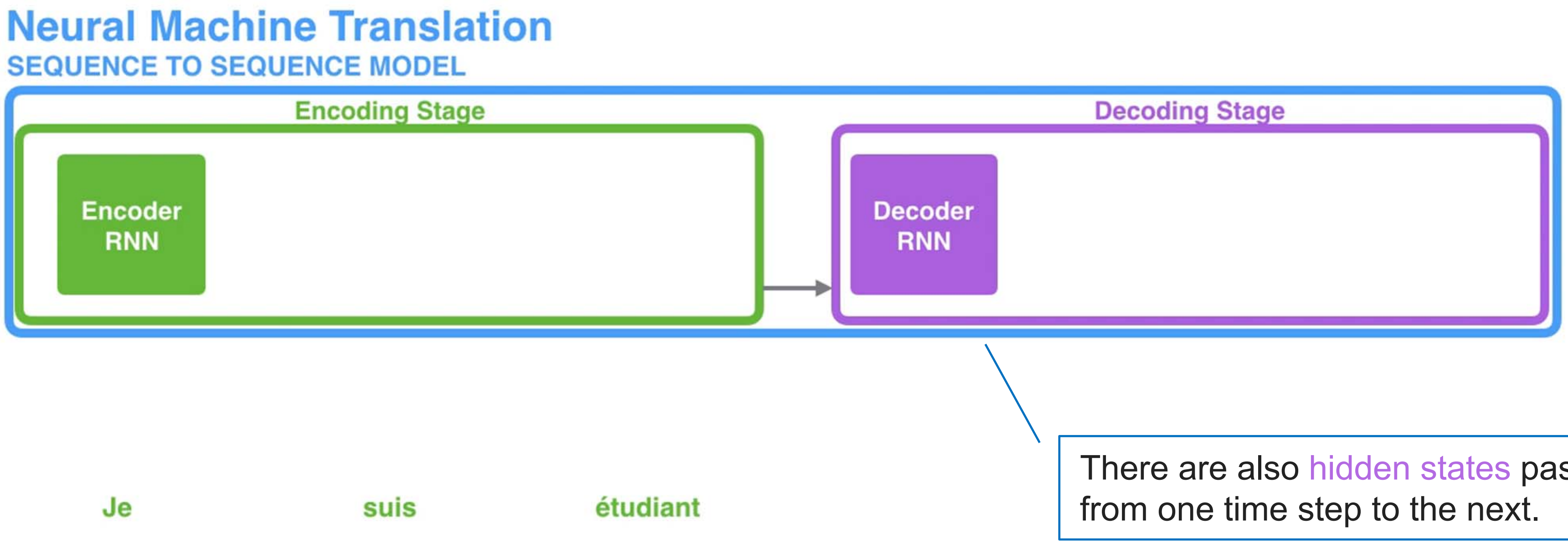
Let's look at the hidden states for the encoder. Notice how the last hidden state is actually the context we pass along to the decoder. The decoder also maintains a hidden states that it passes from one time step to the next.



There are also hidden states passing from one time step to the next.

Recurrent Neural Network

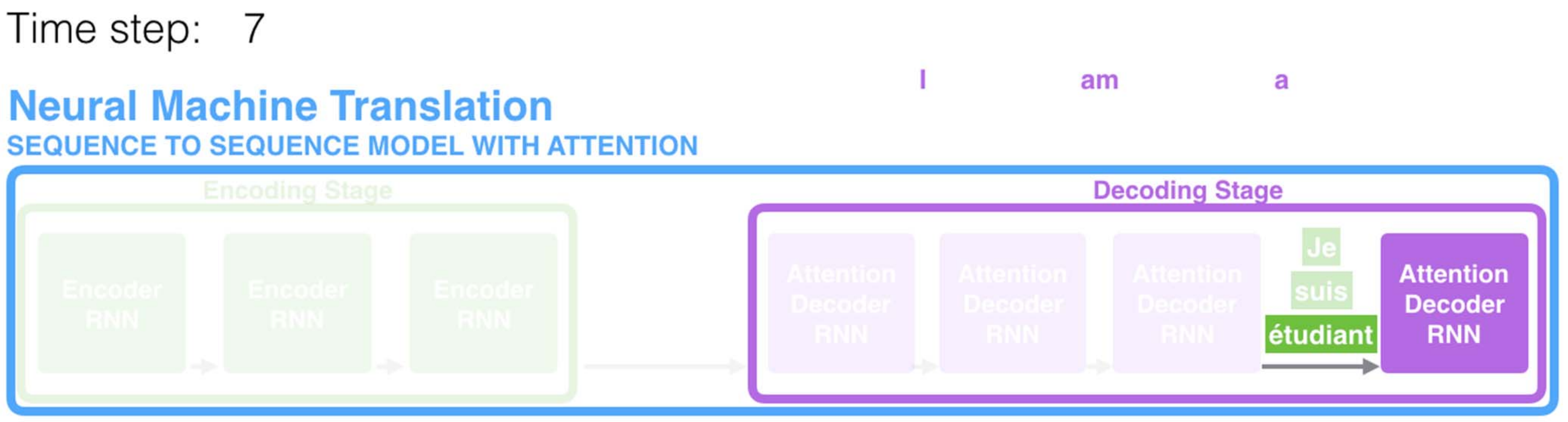
This animation will make it easier to understand the static graphics that describe these models. This is called an “unrolled” view where instead of showing the one decoder, we show a copy of it for each time step. This way we can look at the inputs and outputs of each time step.



There are also hidden states passing from one time step to the next.

The 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.

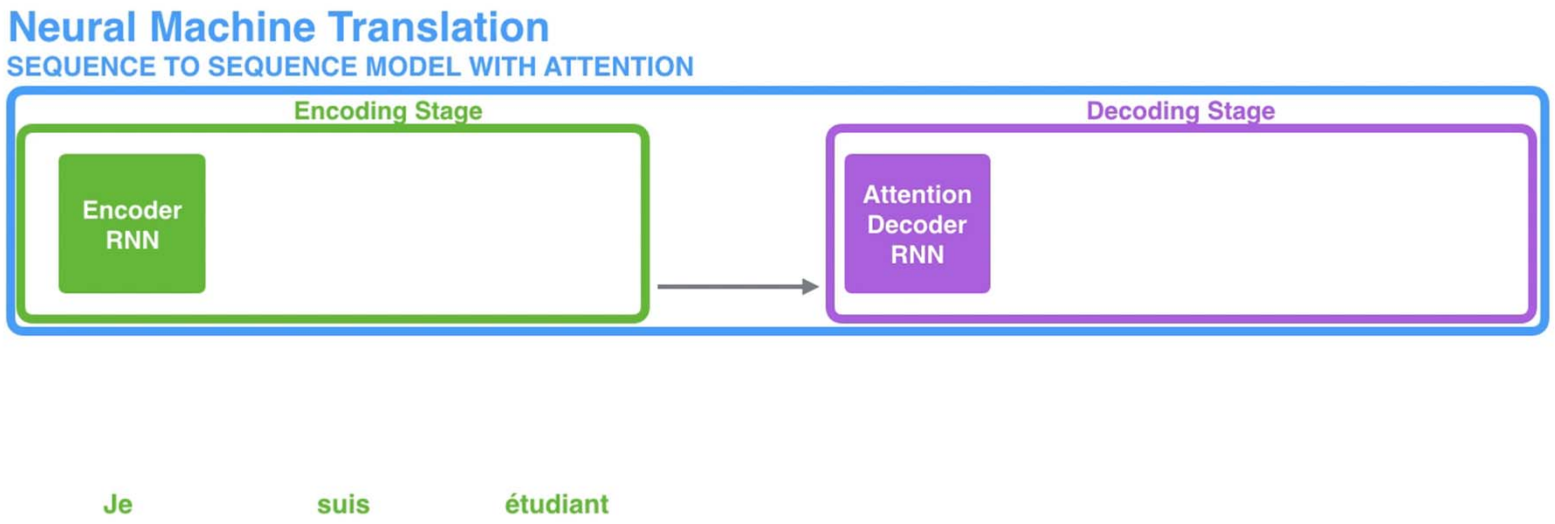


At time step 7, the attention mechanism enables the decoder to focus on the word "étudiant" ("student" in French) before it generates the English translation. This ability to amplify the signal from the relevant part of the input sequence makes attention models produce better results than models without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

The Attention

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:



The 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 states is most associated with a certain word in the input sentence
2.Give each hidden states a score
3.Multiply each hidden states by its softmaxed score, thus amplifying hidden states with high scores, and drowning out hidden states with low scores
This scoring exercise is done at each time step on the decoder side.

Attention at time step 4

The Attention

Let us now bring the whole thing together in the following visualization and look at how the attention process works:

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.

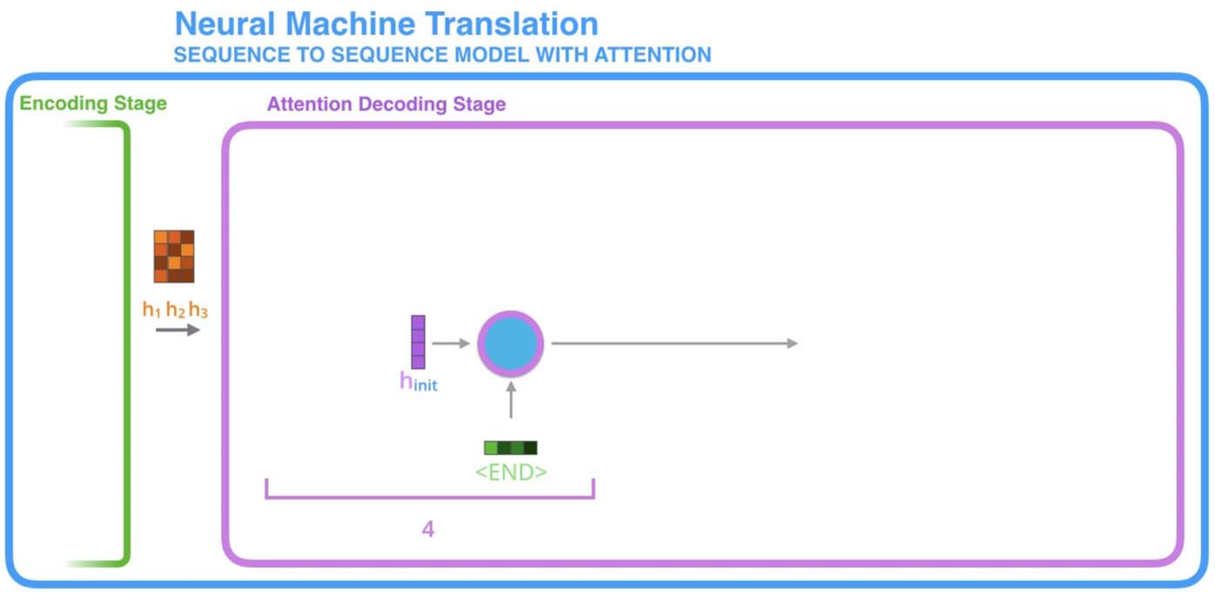
3.Attention Step: We use the encoder hidden states and the h4 vector to calculate a context vector (C4) for this time step.

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.

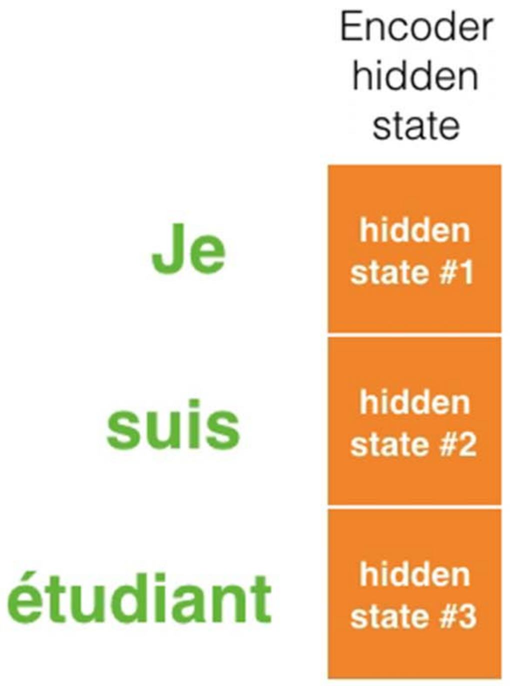
7.Repeat for the next time steps



Neural Machine Translation
SEQUENCE TO SEQUENCE MODEL WITH ATTENTION
Encoding Stage
Attention Decoding Stage
h1 h2 h3
hinit
<END>
4

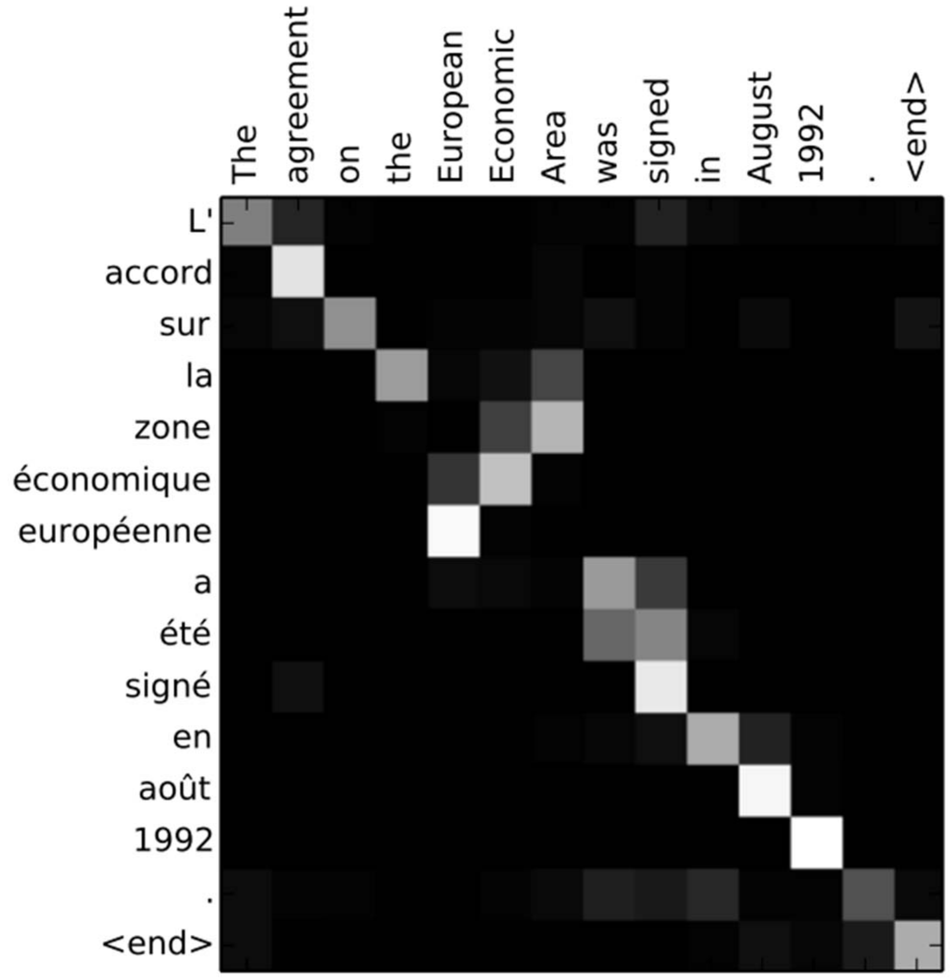
The Attention

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



The 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). An example for how precise this mechanism can be comes from the attention papers listed above:



You can see how the model paid attention correctly when outputing "European Economic Area". In French, the order of these words is reversed ("zone économique européenne") as compared to English. Every other word in the sentence is in similar order.

问题及讨论

Q&A