The Encoder-Decoder architecture and its comparison with the attention mechanism.

<u>Neural Machine Translation</u>:

In neural machine translation, the input is a series of words, processes one after the another.

The output is also a series of words

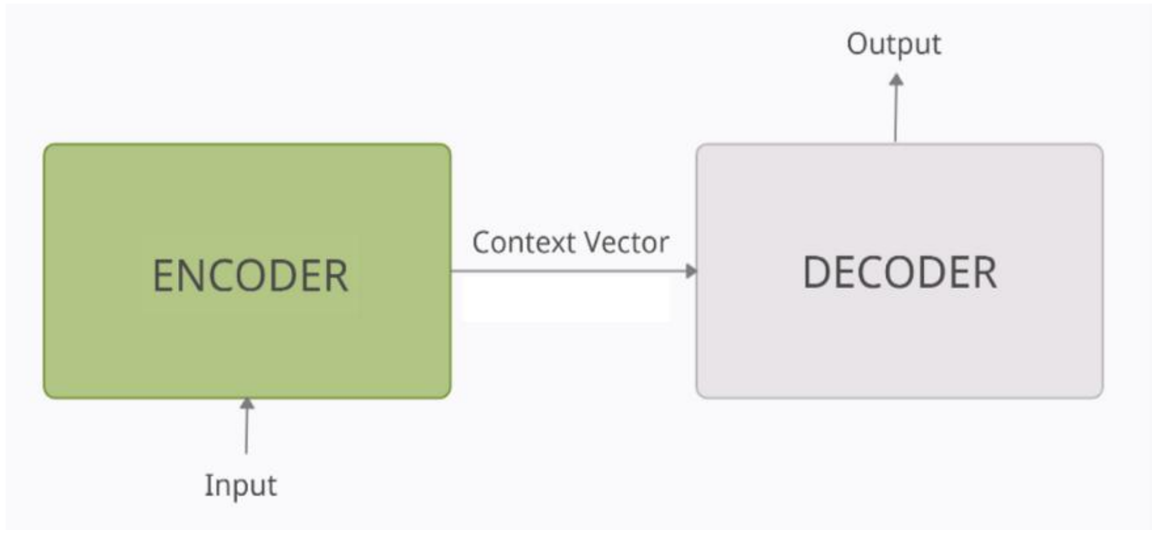
Task: Predict the Spanish translation for every English sentence used as an input

Eg: Input: English Sentence: "**nice to meet you**"
output: Spanish sentence: "**ravi de vous rencontrer**"

For the input-sequence "**nice to meet you**", we want our model to predict the target sequence "**ravi de vous rencontrer**"

Encoder – Decoder Architecture

- An encoder – decoder model can be thought of as two blocks
- The encoder and the decoder connected by a vector knows as ‘context vector’.



<u>Encoder:</u>

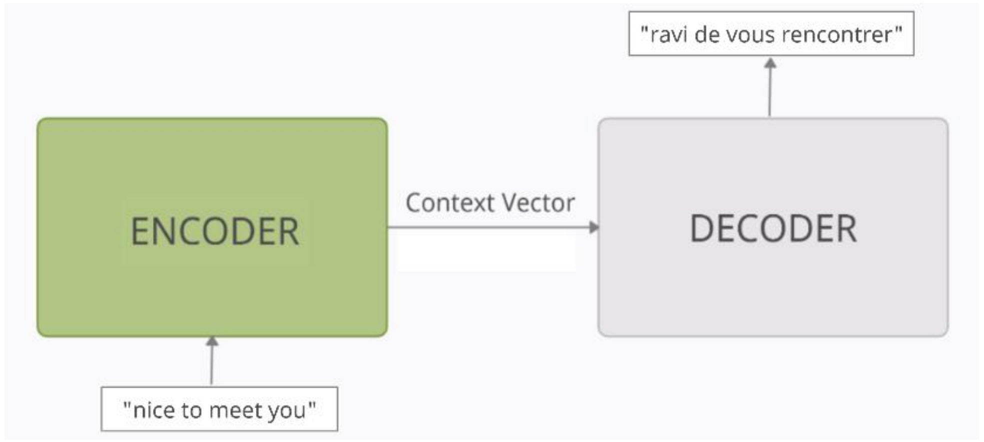
- Processes each token in the input-sequence.
- It tries to overload all the information about the input-sequence into a vector of fixed length i.e. the ‘context vector’.
- After going through all the tokens, the encoder passes this vector onto the decoder.

<u>Context Vector:</u>

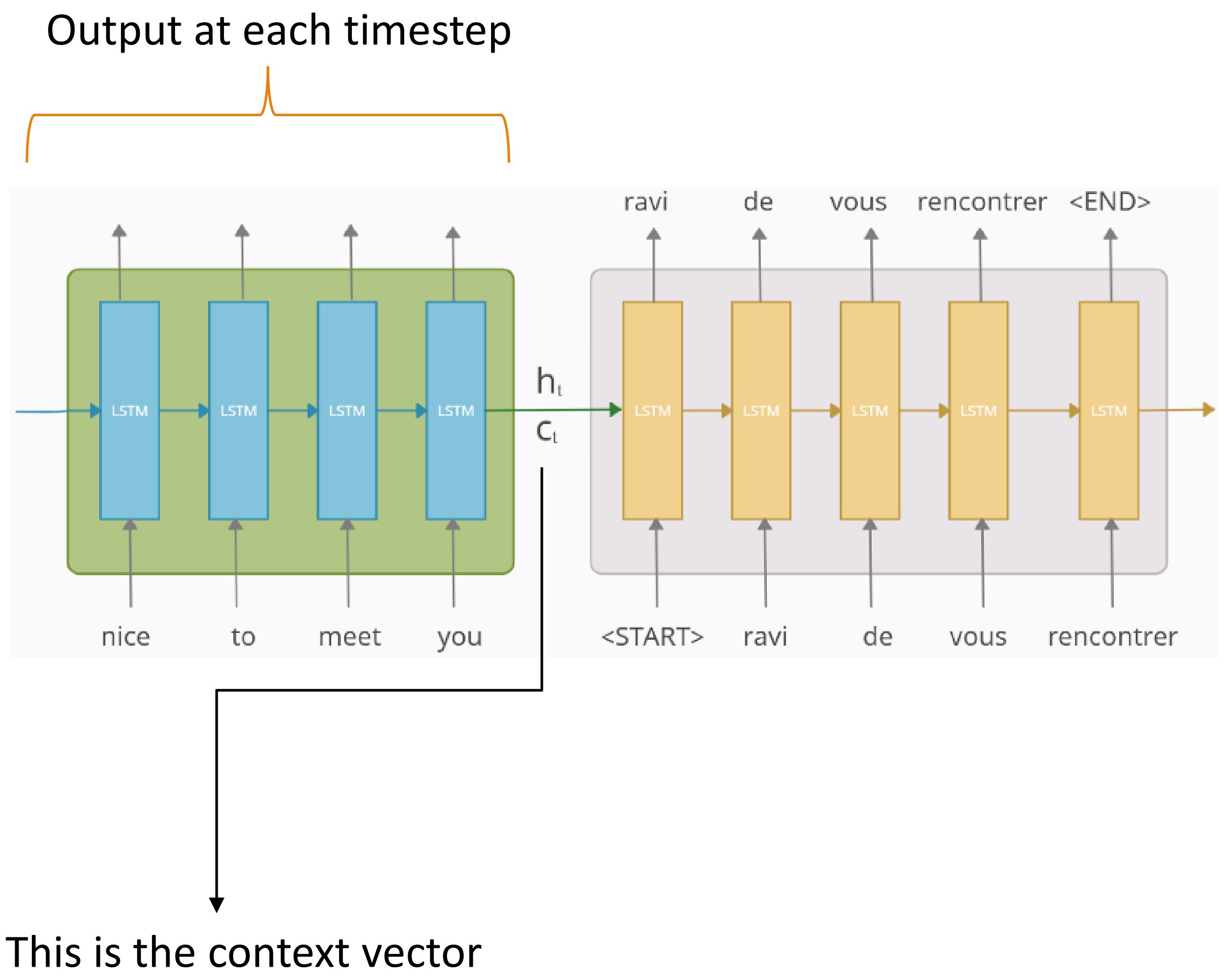
- The vector is built in such a way that it's expected to encapsulate the whole meaning of the input-sequence
- Help the decoder make accurate predictions.

<u>Decoder:</u>

- The decoder reads the context vector and tries to predict the target-sequence token by token.

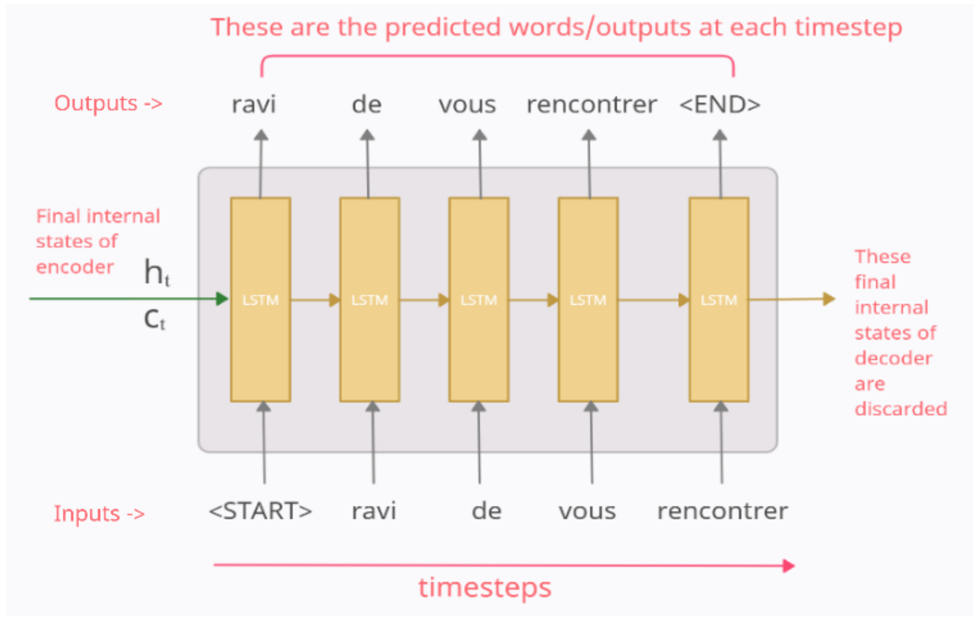


- It is fed in the input-sequence over time and it tries to encapsulate all its information
- Store it in its final internal states $\mathbf{h_t}$ (hidden state) and $\mathbf{c_t}$ (cell state).
- The internal states are then passed onto the decoder part.
- The outputs at each time-step of the encoder part are all discarded



Decoder:

- After reading the whole input-sequence, the encoder passes the internal states to the decoder.
- The initial states **(h_0, c_0)** of the decoder are set to the final states **(h_t, c_t)** of the encoder.
- The way decoder works, is, that its output at any time-step **t** is supposed to be the **t^{th}** word in the target-sequence


These are the predicted words/outputs at each timestep
Outputs ->
ravi
de
vous
rencontrer
<END>
Final internal states of encoder
h_t
C_t
LSTM
LSTM
LSTM
LSTM
LSTM
These final internal states of decoder are discarded
Inputs ->
<START>
ravi
de
vous
rencontrer
timesteps

<u>At time-step 1</u>

- The input fed to the decoder at the first time-step is a special symbol **“<START>”**.
- decoder uses this input and the internal states **(h_t, c_t)** to produce the output in the 1st time-step.

<u>At time-step 2</u>

- The output from the 1st time-step **“ravi”** is fed as input to the 2nd time-step.
- The output in the 2nd time-step is supposed to be the 2nd word in the target-sequence i.e. **‘de’**

Similarly….

- The output at each time-step is fed as input to the next time-step.
- This continues till we get the “<END>” symbol which is again a special symbol used to mark the end of the output-sequence.

The working of the decoder is different during the training

- We feed the **true** output (and **not the predicted** output) from the previous time-step as input to the current time-step.

Attention Mechanism:

Generally, A source sentence in English to an encoder gives complete information of the source sequence into a single real valued vector (context vector)

This context vector is passed on the decoder to produce an output sequence in a target language.

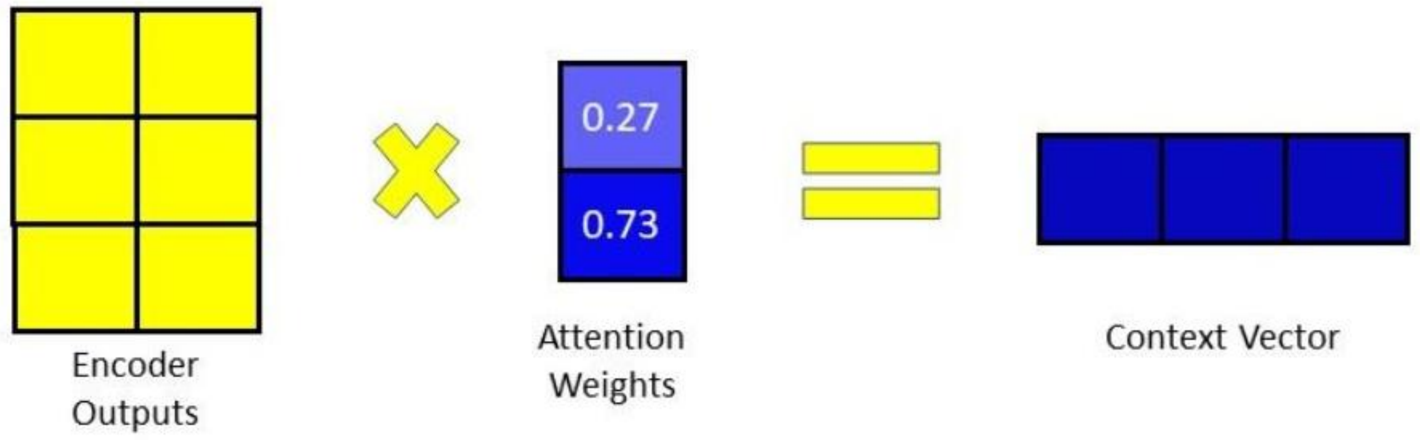
What if the input sequence is long?

Can single vector can hold all the relevant information to provide to the decoder?

The basic idea of Attention mechanism is to avoid attempting to learn a single vector representation for each sentence, instead, it pays attention to specific input vectors of the input sequence based on the attention weights.

Alignment Score:

- The alignment score is the essence of the Attention mechanism, as it quantifies the amount of "Attention" the decoder will place on each of the encoder outputs when producing the next output.
- After generating the alignment scores vector, we can then apply a SoftMax on this vector to obtain the attention weights.



Context Vector is derived from the weights and encoder outputs

Three Attention Mechanisms:

Bahdanau's attention
(concat product)

$$a_{it} = V_{att}^{T} \tanh(W_1 s_{t-1} + W_2 h_i)$$

Luong's attention
(general)

$$a_{it} = s_{t-1}^{T} W h_i$$

Dot product

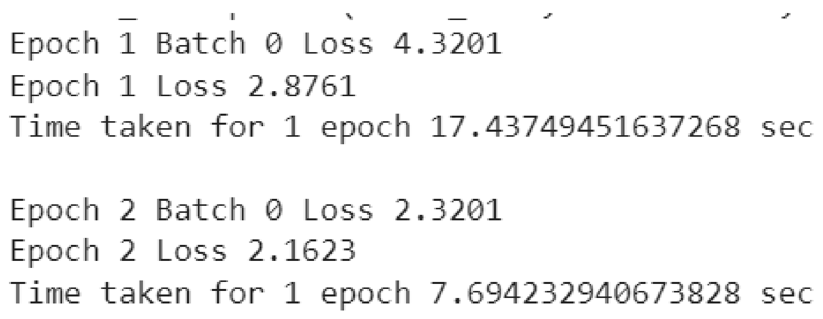
$$a_{it} = s_{t-1}^{T} h_i$$

- $\boldsymbol{a_{it}}$ is the probability
- $\boldsymbol{S_{t-1}}$ is the decoder state information at time (t-1)
- $\boldsymbol{h_i}$ is the encoder state information at time i

Comparing the effect of three attention mechanisms:

- When we implement three attention mechanisms the following were the result

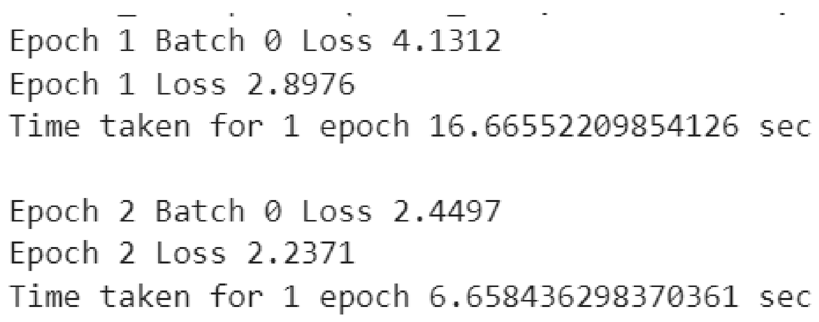
Bahdanau's attention



```
Epoch 1 Batch 0 Loss 4.3201
Epoch 1 Loss 2.8761
Time taken for 1 epoch 17.43749451637268 sec

Epoch 2 Batch 0 Loss 2.3201
Epoch 2 Loss 2.1623
Time taken for 1 epoch 7.694232940673828 sec
```

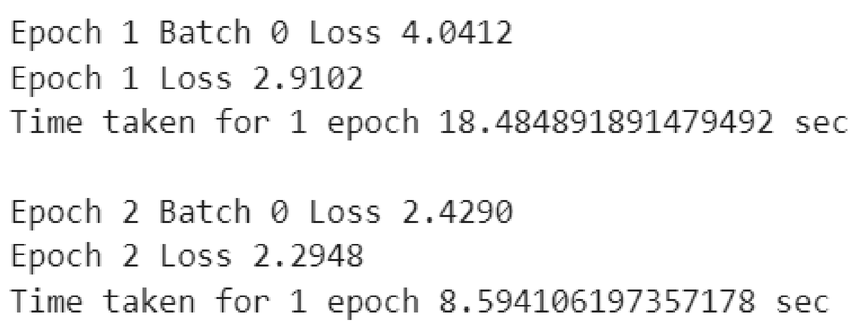
Dot attention



```
Epoch 1 Batch 0 Loss 4.1312
Epoch 1 Loss 2.8976
Time taken for 1 epoch 16.66552209854126 sec

Epoch 2 Batch 0 Loss 2.4497
Epoch 2 Loss 2.2371
Time taken for 1 epoch 6.658436298370361 sec
```

Luong's attention



```
Epoch 1 Batch 0 Loss 4.0412
Epoch 1 Loss 2.9102
Time taken for 1 epoch 18.484891891479492 sec

Epoch 2 Batch 0 Loss 2.4290
Epoch 2 Loss 2.2948
Time taken for 1 epoch 8.594106197357178 sec
```

Comparing the above three based on loss, Bahdanau's attention mechanism is performing best compare with the other two followed by Dot attention and then finally Luong's attention

THANK YOU