

Transformers

PREPARED BY: AHMAD ALAA ALDINE

PRESENTED BY: AHMAD ALAA ALDINE

Outline

- Recurrent Neural Network (RNN)
- Encoder Decoder
- Attention
- Transformers
- BERT

RNN

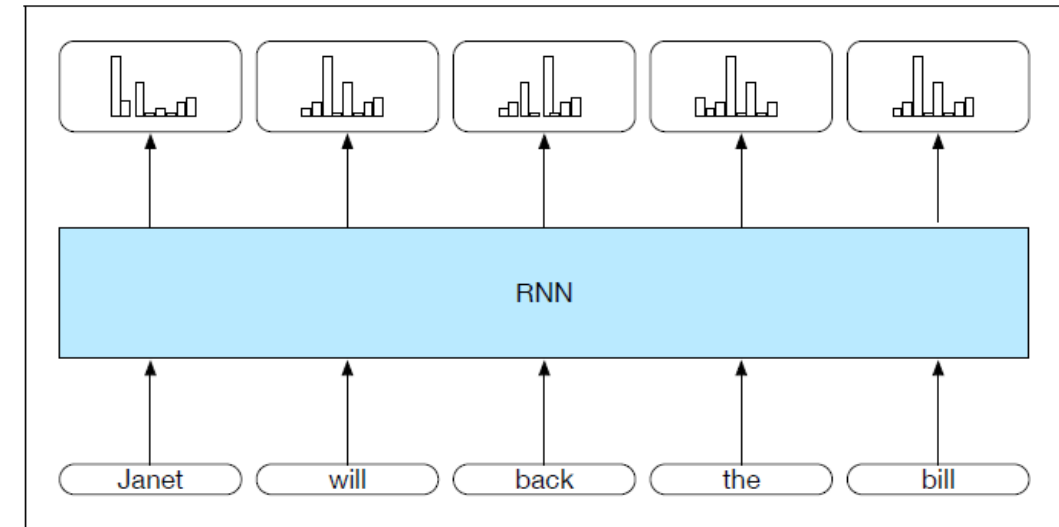
RNN (1/3)

RNN: input sequence is transformed into output sequence in a one-to-one fashion.

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

Applications:

- Machine Translation
- Summarization
- Question Answering



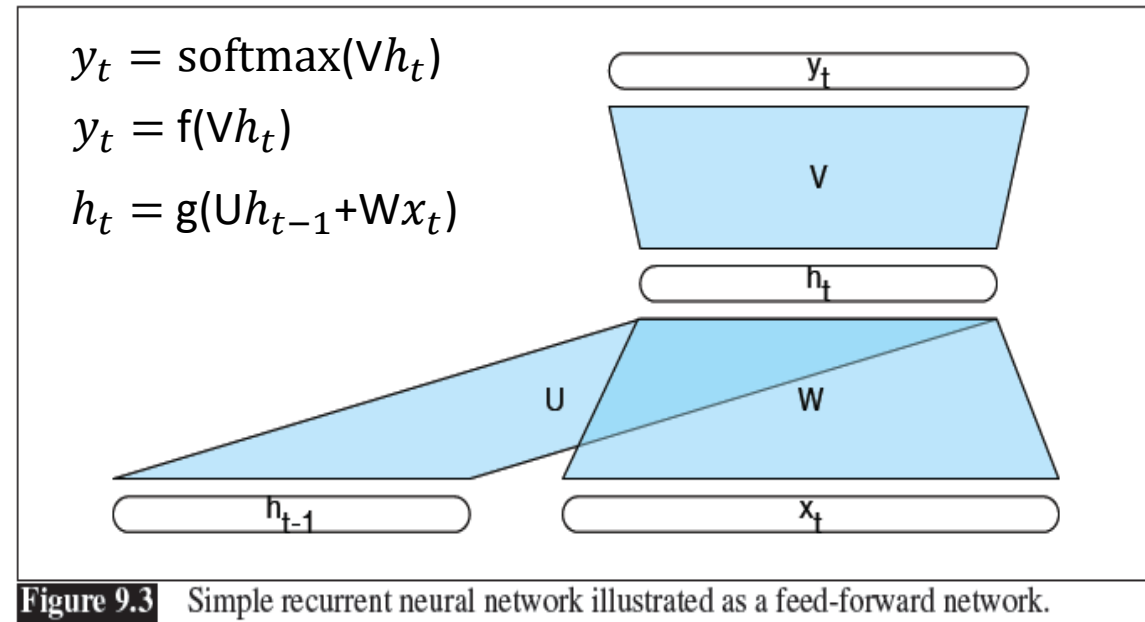
RNN (2/3)

Most significant change

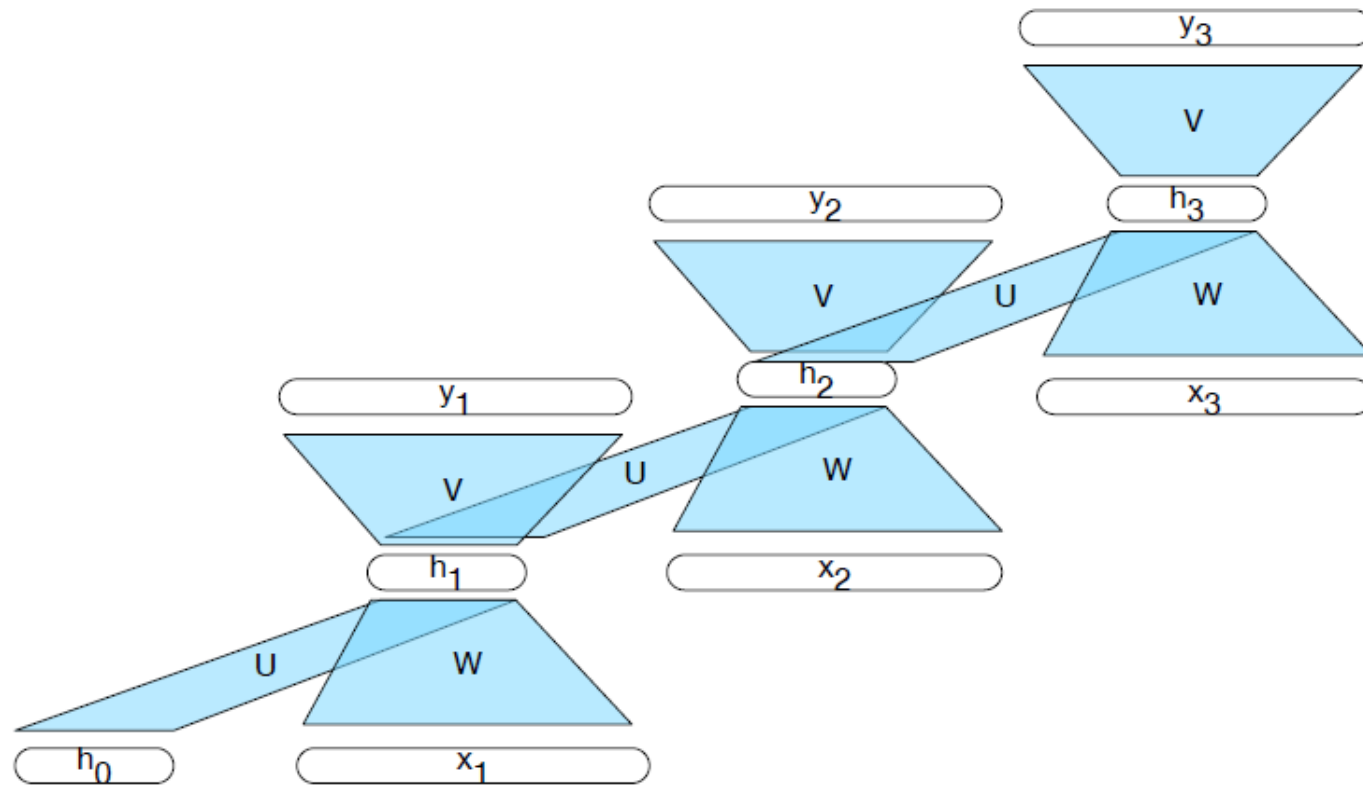
- new set of weights $\rightarrow U$

Connect the hidden layer from the previous time step to the current hidden layer.

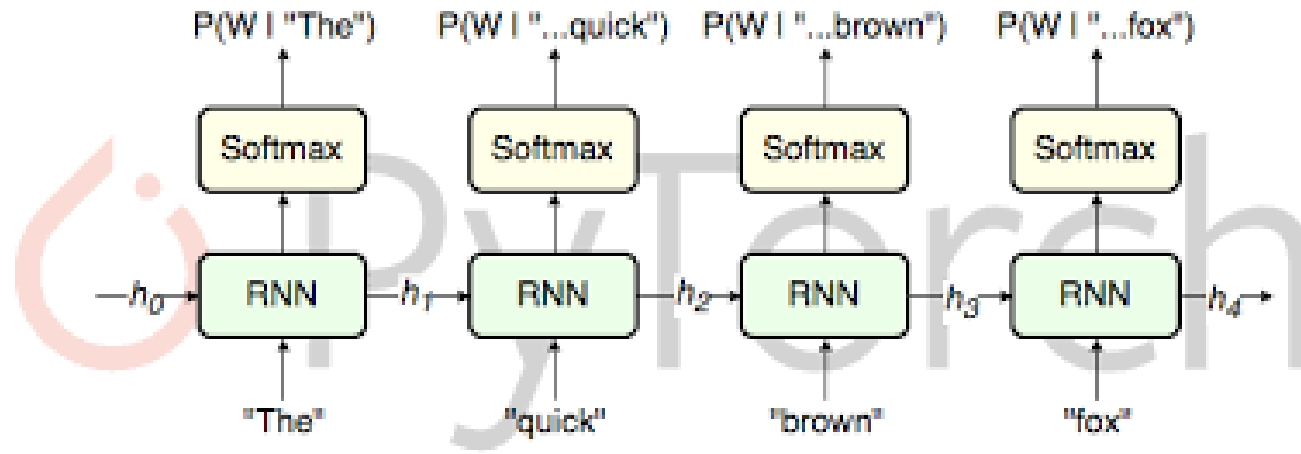
Determine how the network should use past context to calculate the output for the current input.



RNN (3/3)



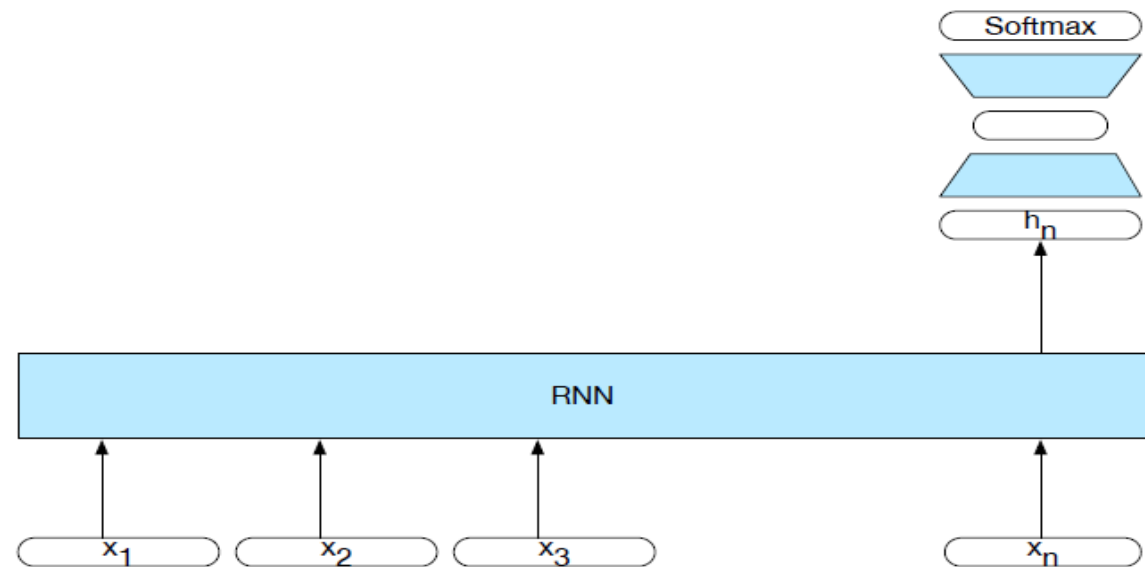
RNN – Language Modeling



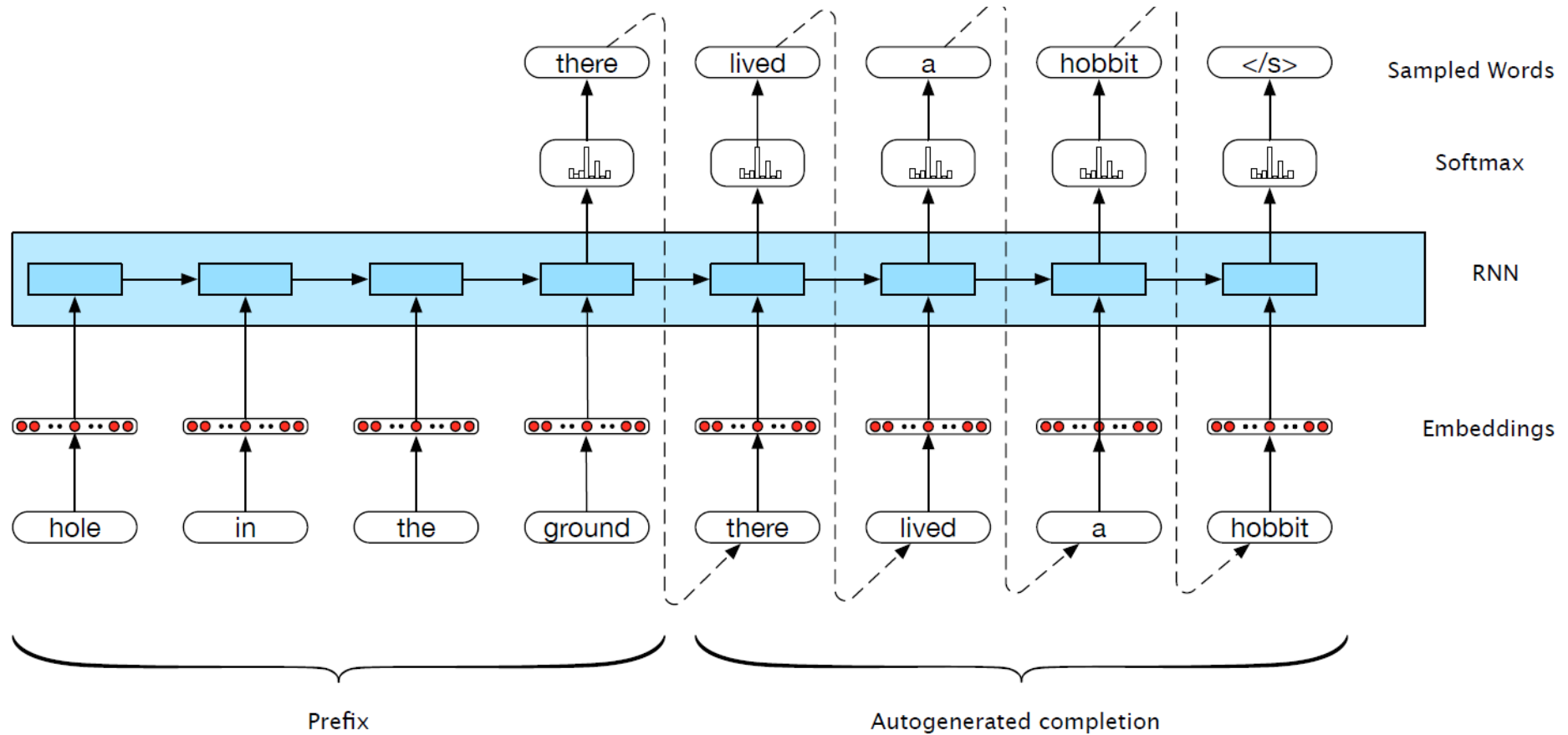
RNN – Sequence Classification

Sequence Classification Using RNN

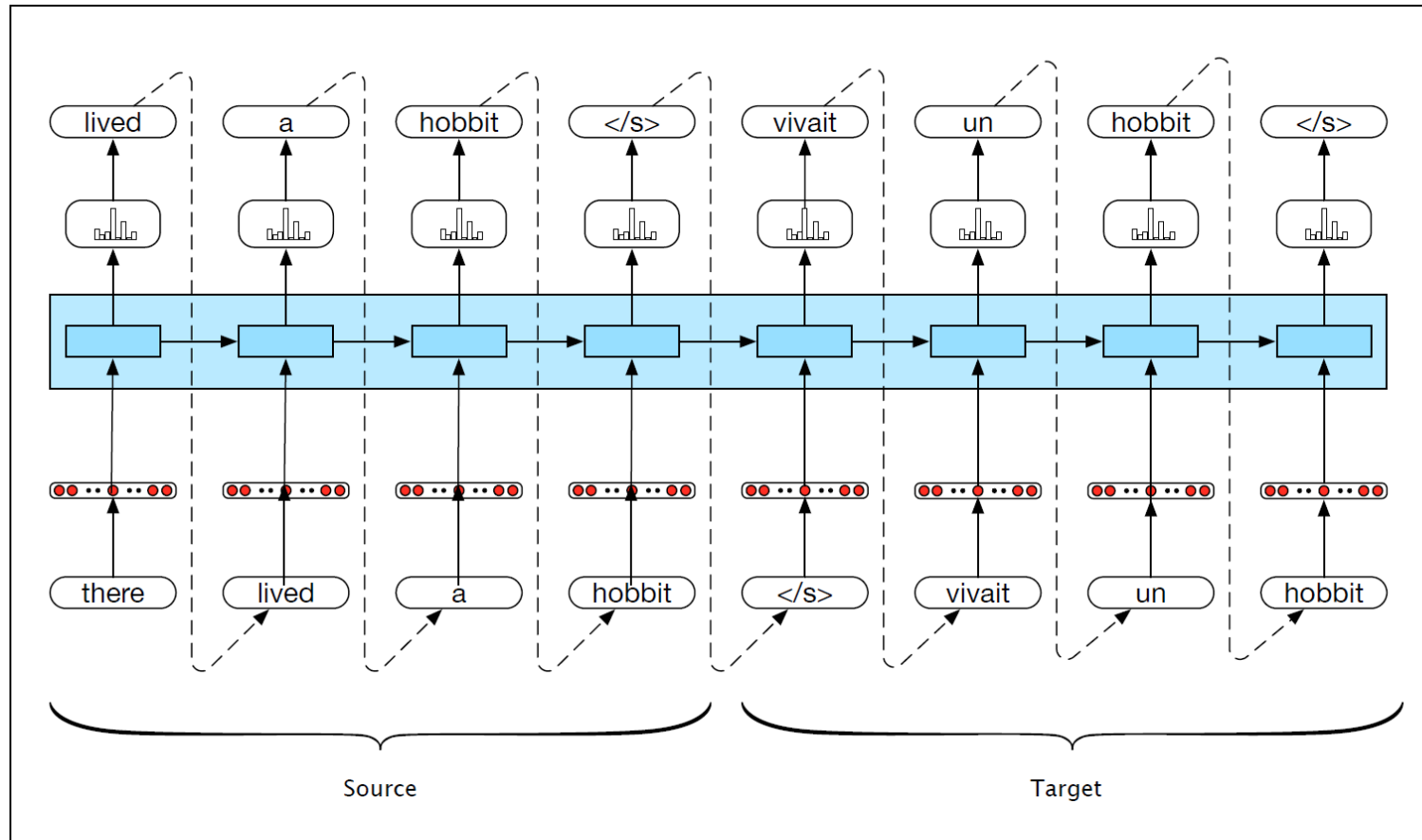
- Sentiment Analysis
- Topic Classification
- Spam Detection



RNN – Sentence Completion



RNN – Translation



Encoder Decoder

Encoder Decoder

1. Encoder

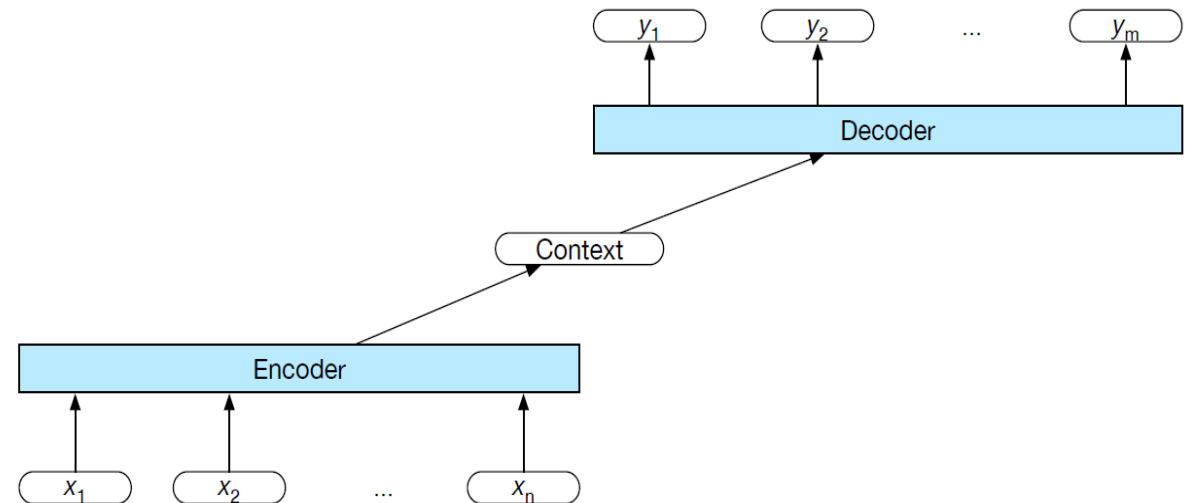
- Accepts an input sequence, $x_{1:n}$ and generates a corresponding sequence of contextualized representations, $h_{1:n}$

2. Context vector c

- Function of $h_{1:n}$ and conveys the essence of the input to the decoder.

3. Decoder

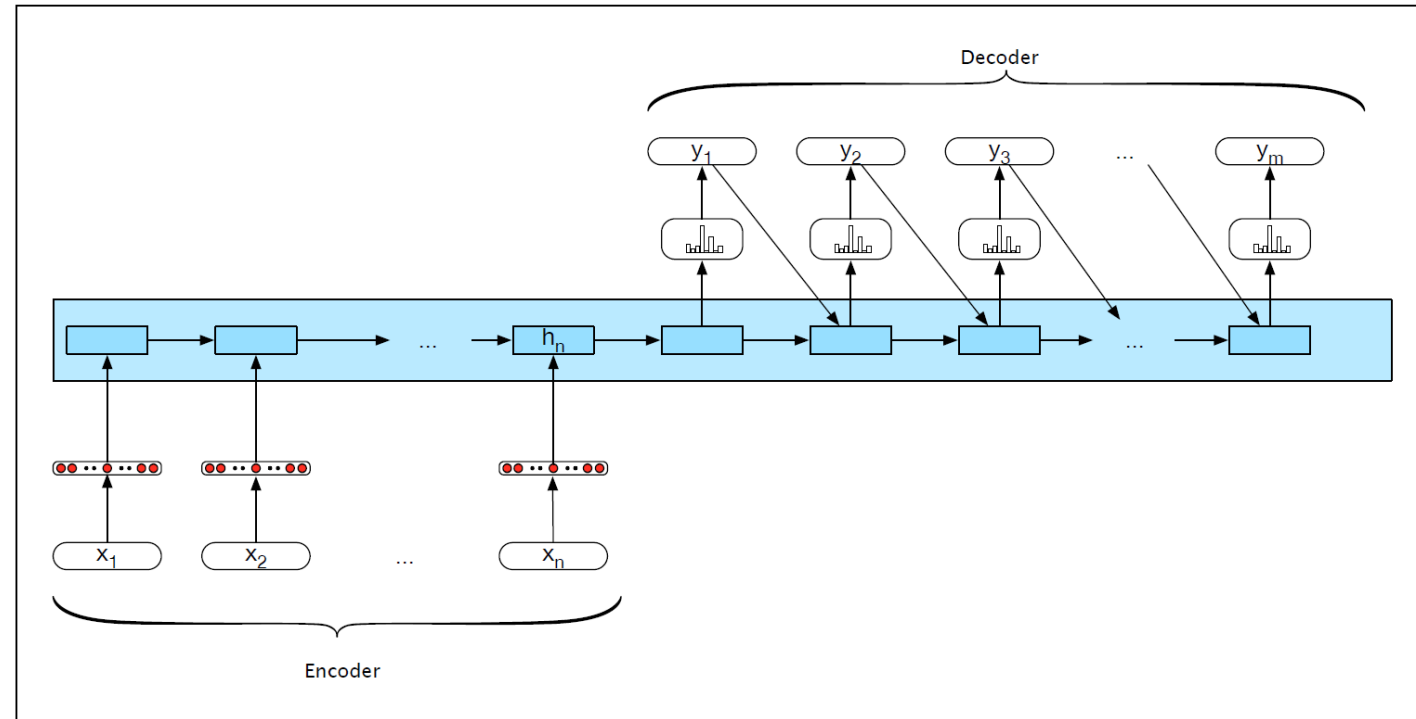
- Accepts c as input and generates an arbitrary length sequence of hidden states $h_{1:m}$ from which a corresponding sequence of output states $y_{1:m}$ can be obtained.



Basic RNN-based encoder-decoder architecture

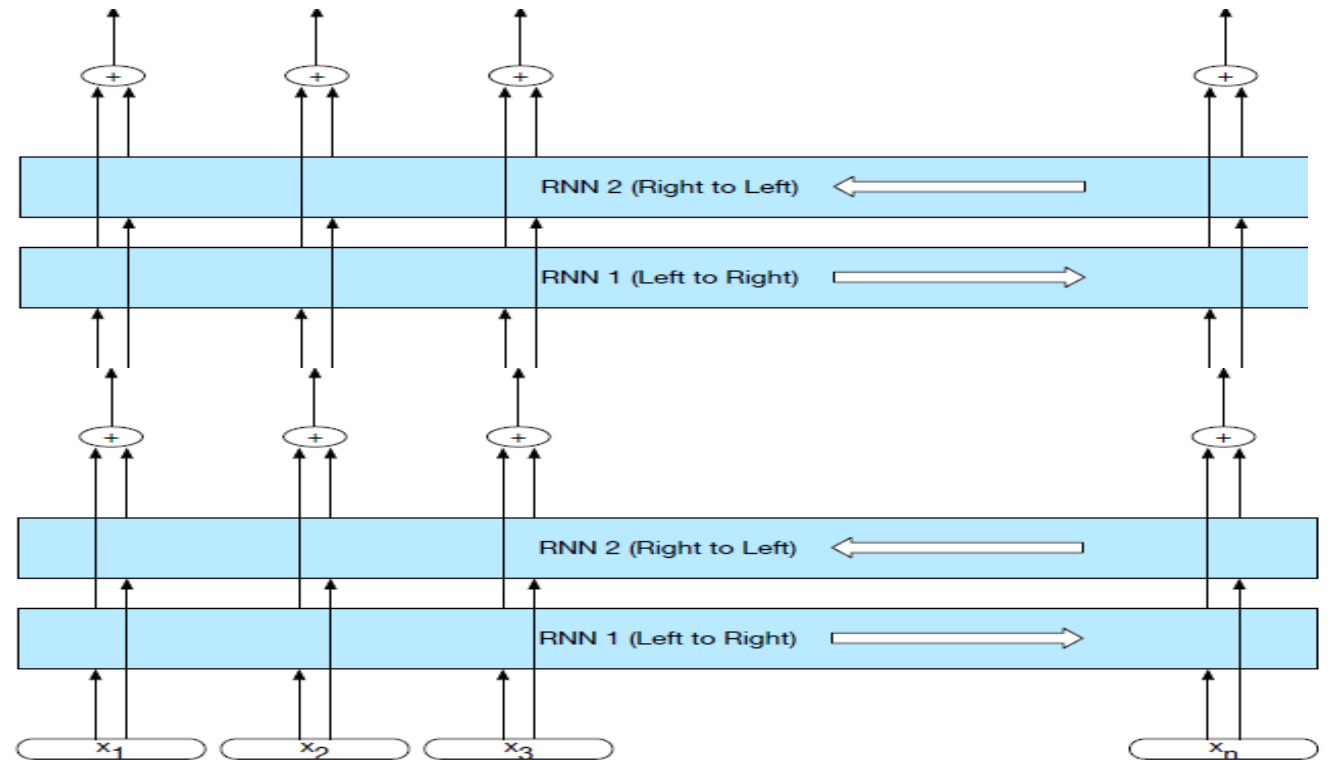
Encoder and Decoder assumed to have the same internal structure

Final state of the Encoder is the only context available to D as its initial hidden state



Encoder Popular Design

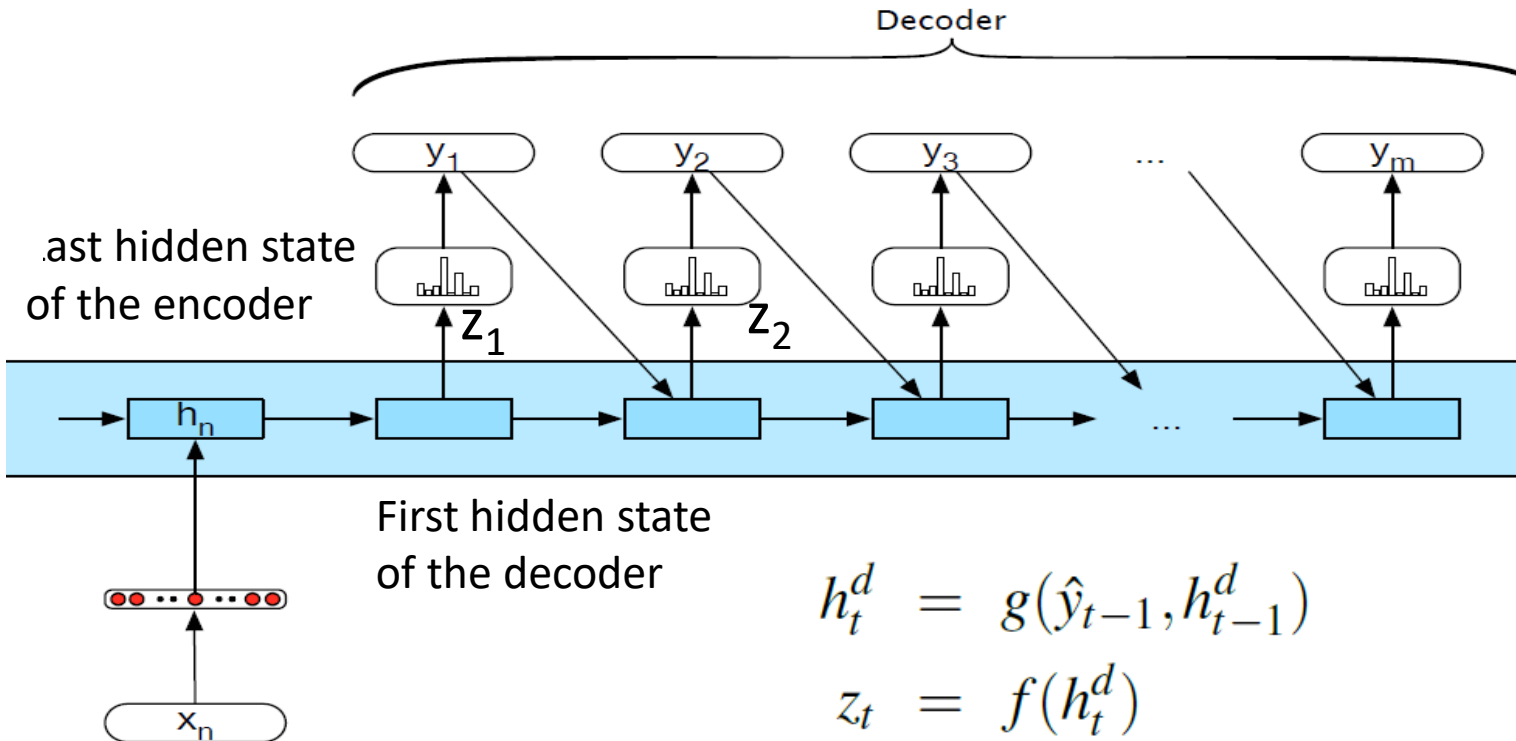
A widely used encoder design makes use of stacked Bi-LSTMs where the hidden states from the forward and backward passes are concatenated



Decoder Basic Design

$$c = h_n^e$$

$$h_0^d = c$$



$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d)$$

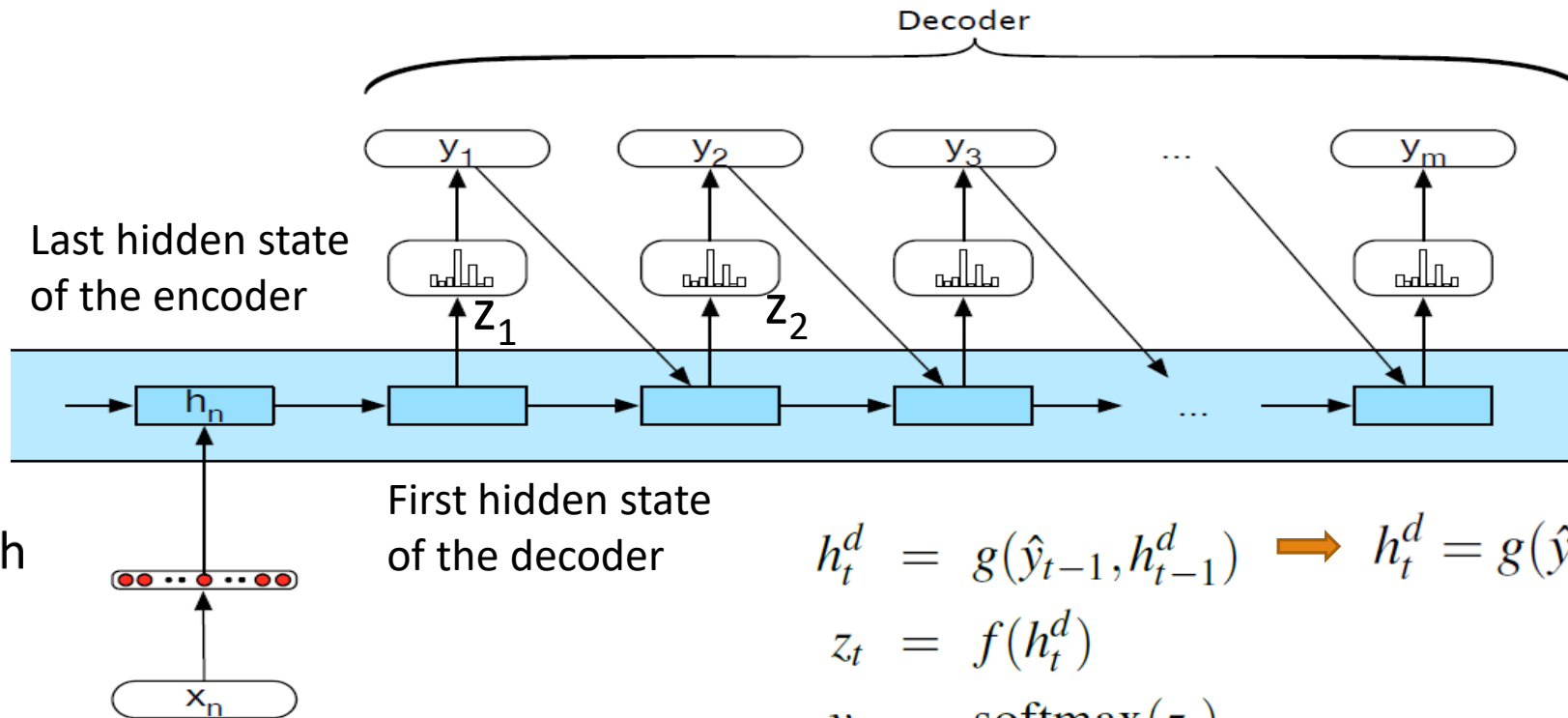
$$z_t = f(h_t^d)$$

$$y_t = \text{softmax}(z_t)$$

Decoder Enhanced Design

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

Context available at each step of decoding



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

$$z_t = f(h_t^d)$$

$$y_t = \text{softmax}(z_t)$$

Attention

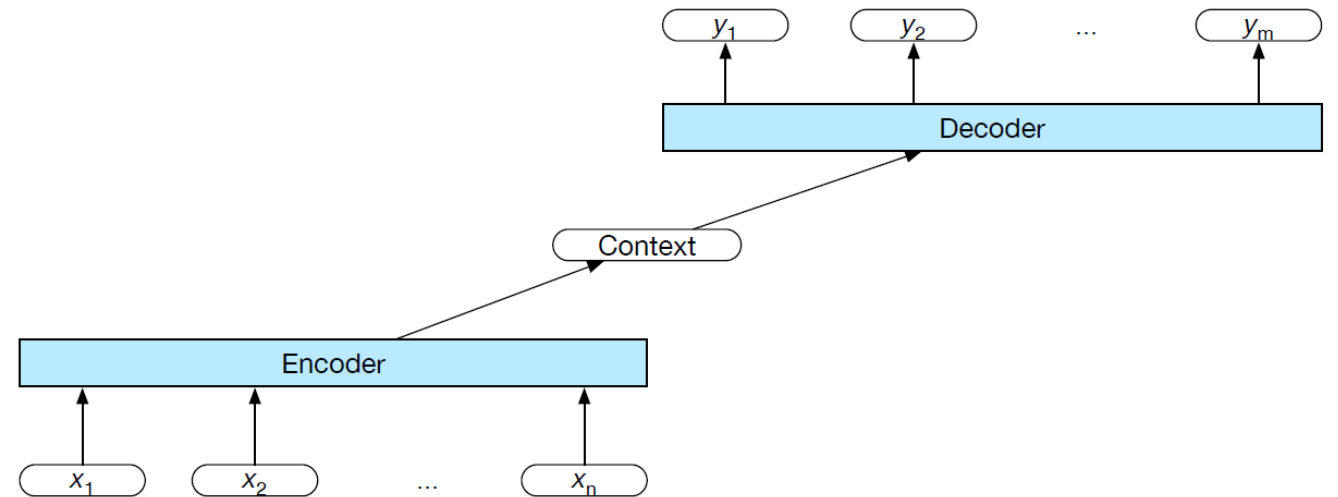
Attention (1/2)

Context vector c

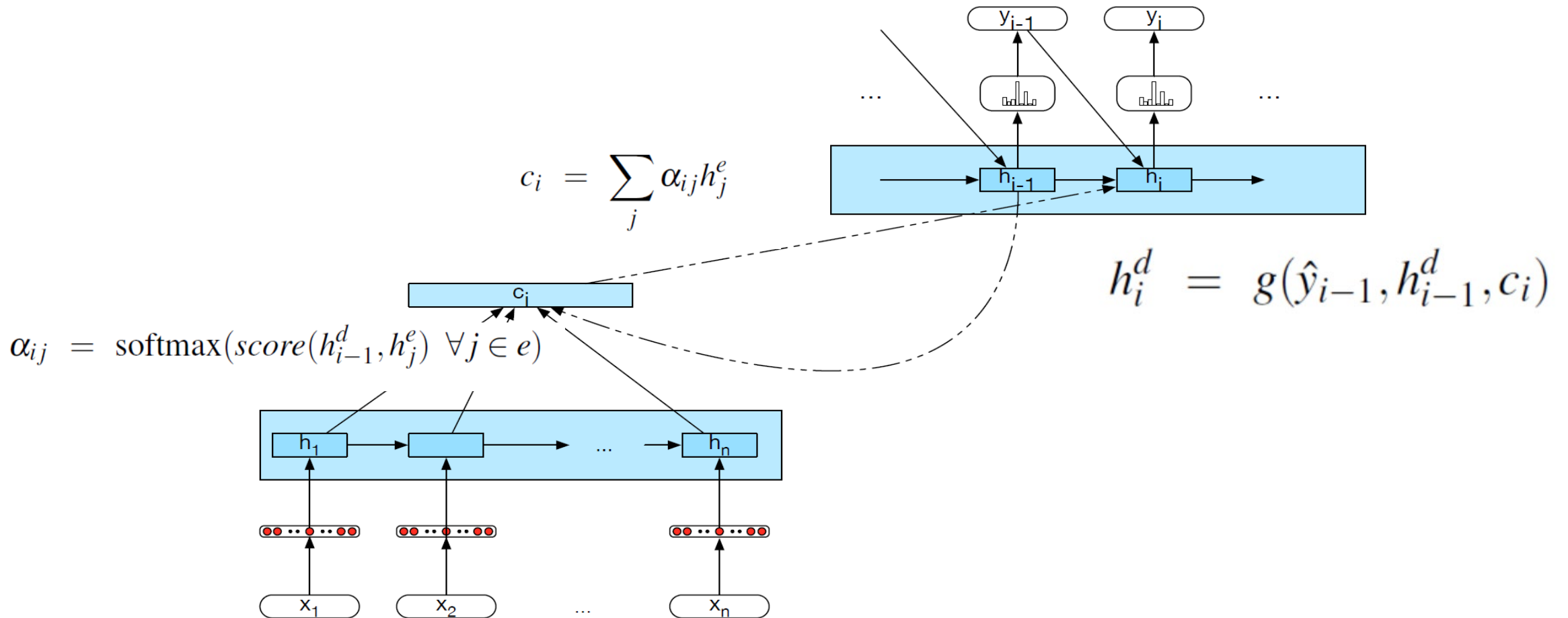
- function of $h_{1:n}$ and conveys the essence of the input to the decoder

Flexible?

- Different for each h_i of the decoder
- Flexibly combining each h_i of the encoder



Attention (2/2)

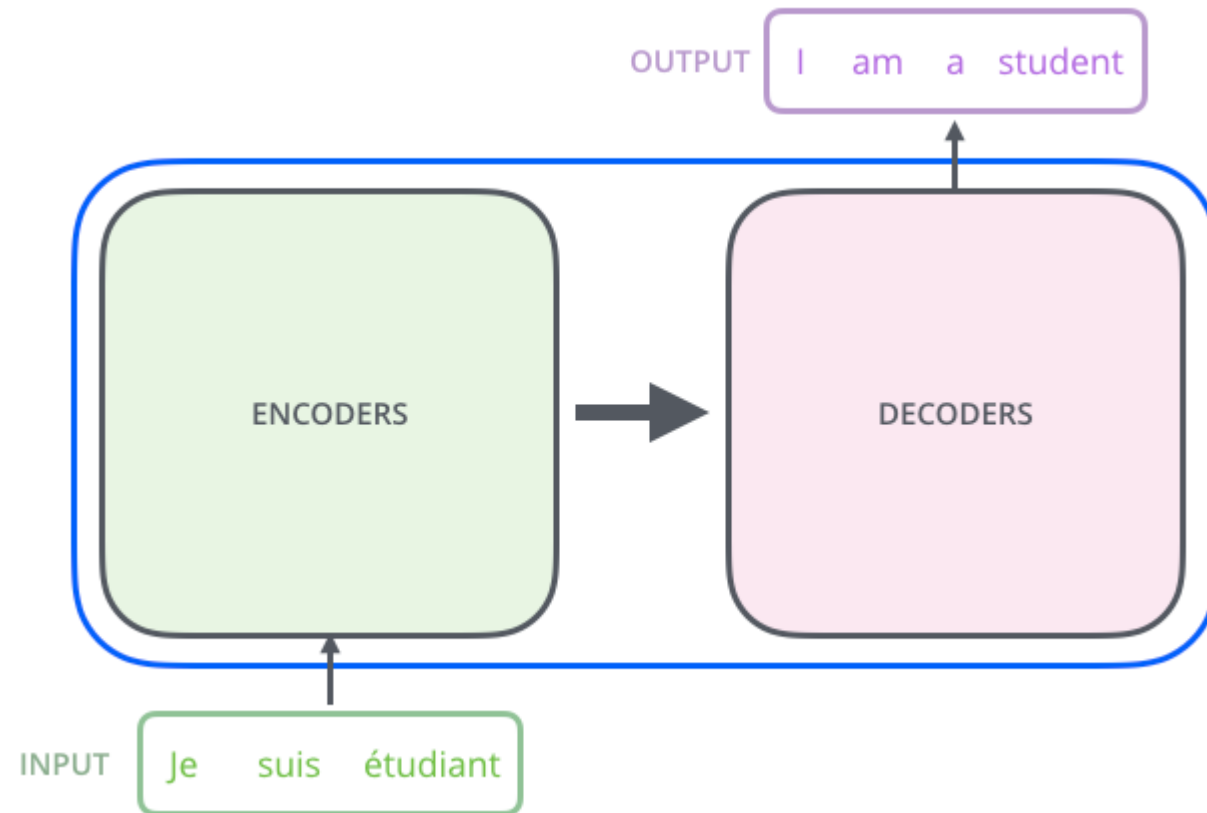


Transformers

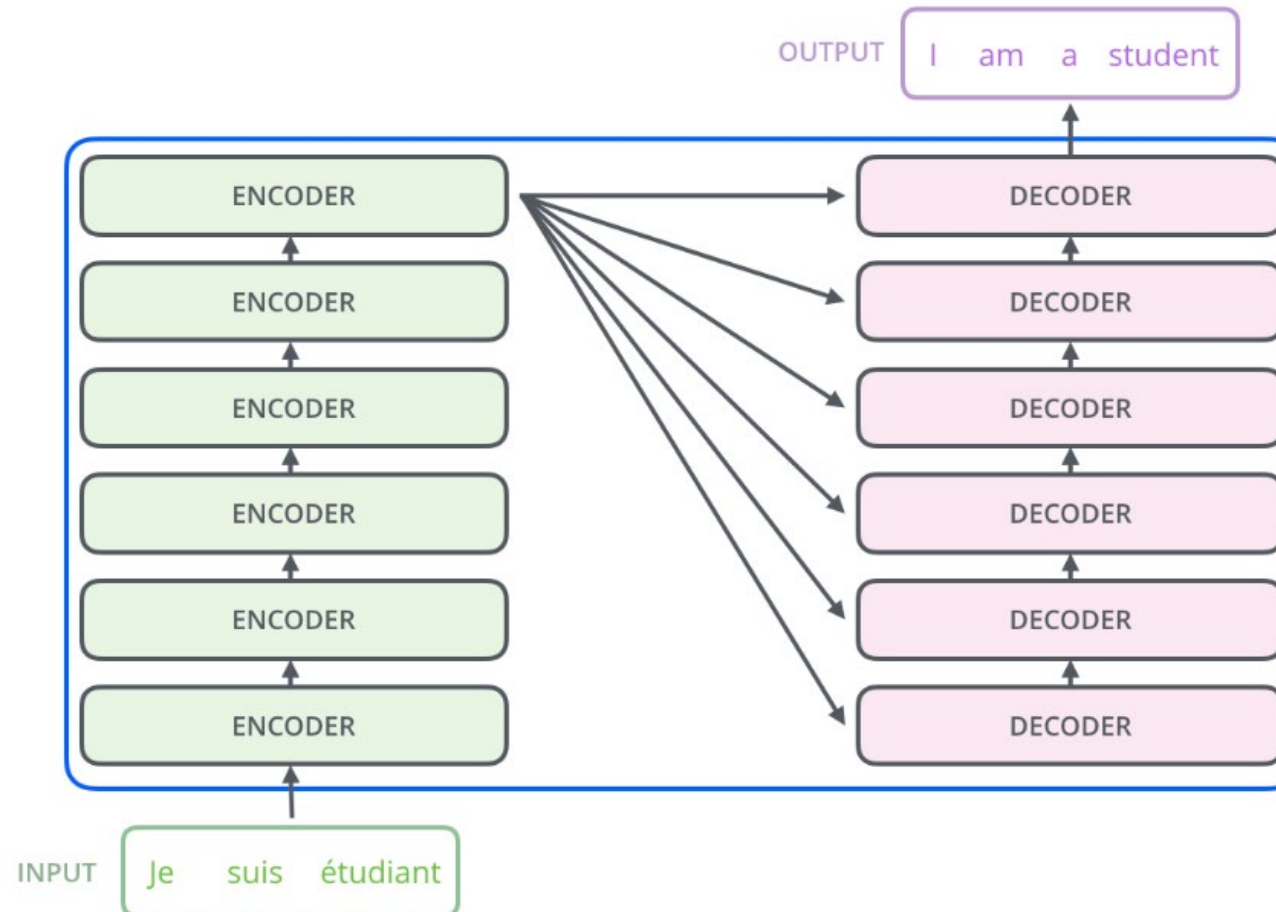
Transformers (1/5)



Transformers (2/5)



Transformers (3/5)

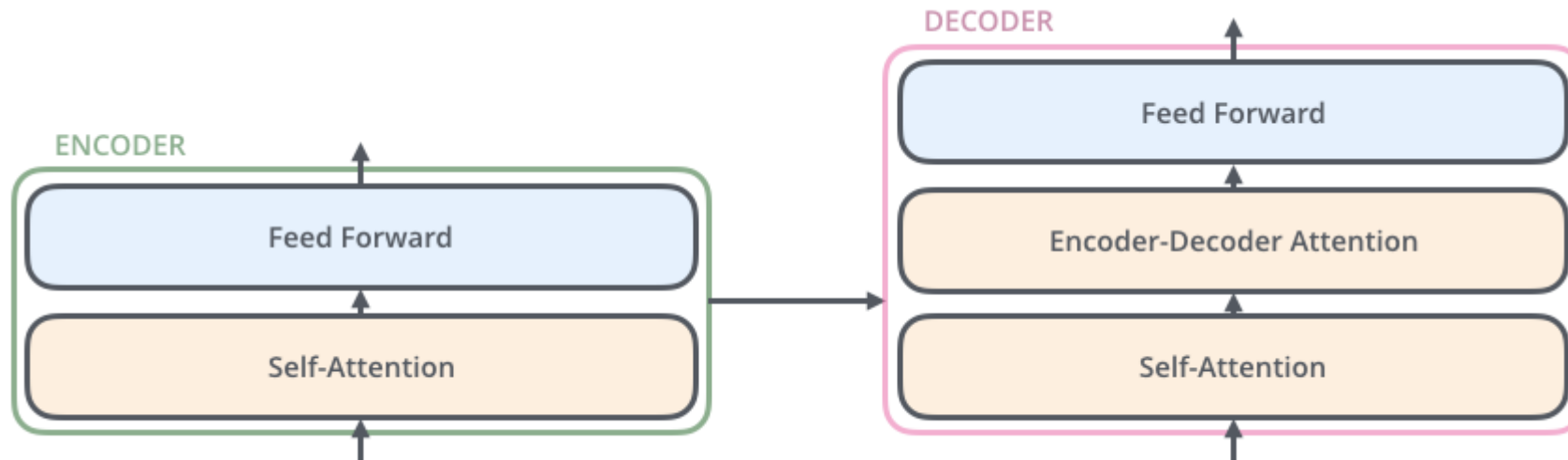


Transformers (4/5)

The encoders are all identical in structure (yet they do not share weights).

Each one is broken down into two sub-layers.

The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence.



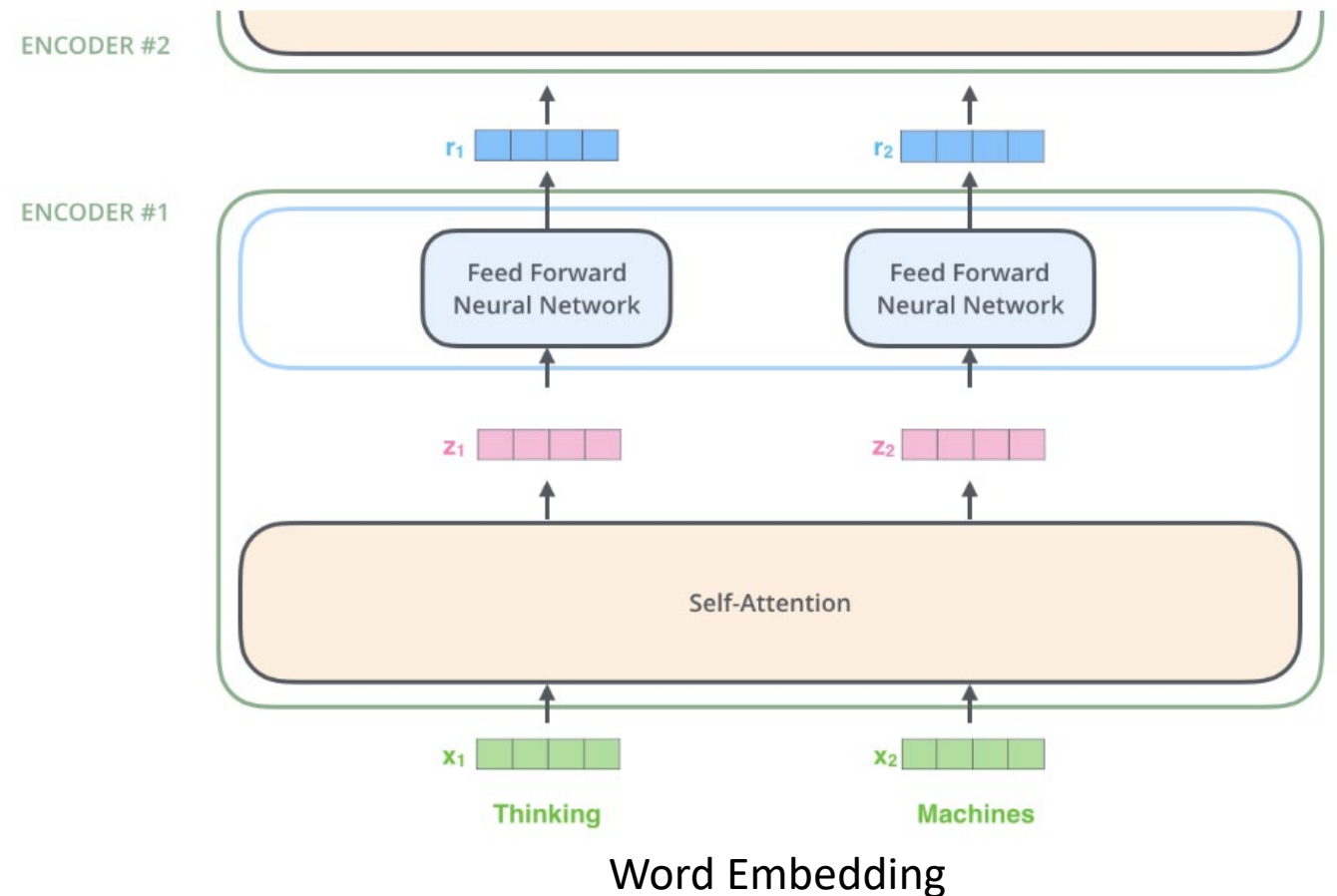
Transformers (5/5)

Word in each position flows through its path in the encoder.

There are dependencies between these paths in the self-attention layer.

The feed-forward layer does not have those dependencies.

Thus the various paths can be executed in parallel while flowing through the feed-forward layer.



Self Attention Layer (1/3)

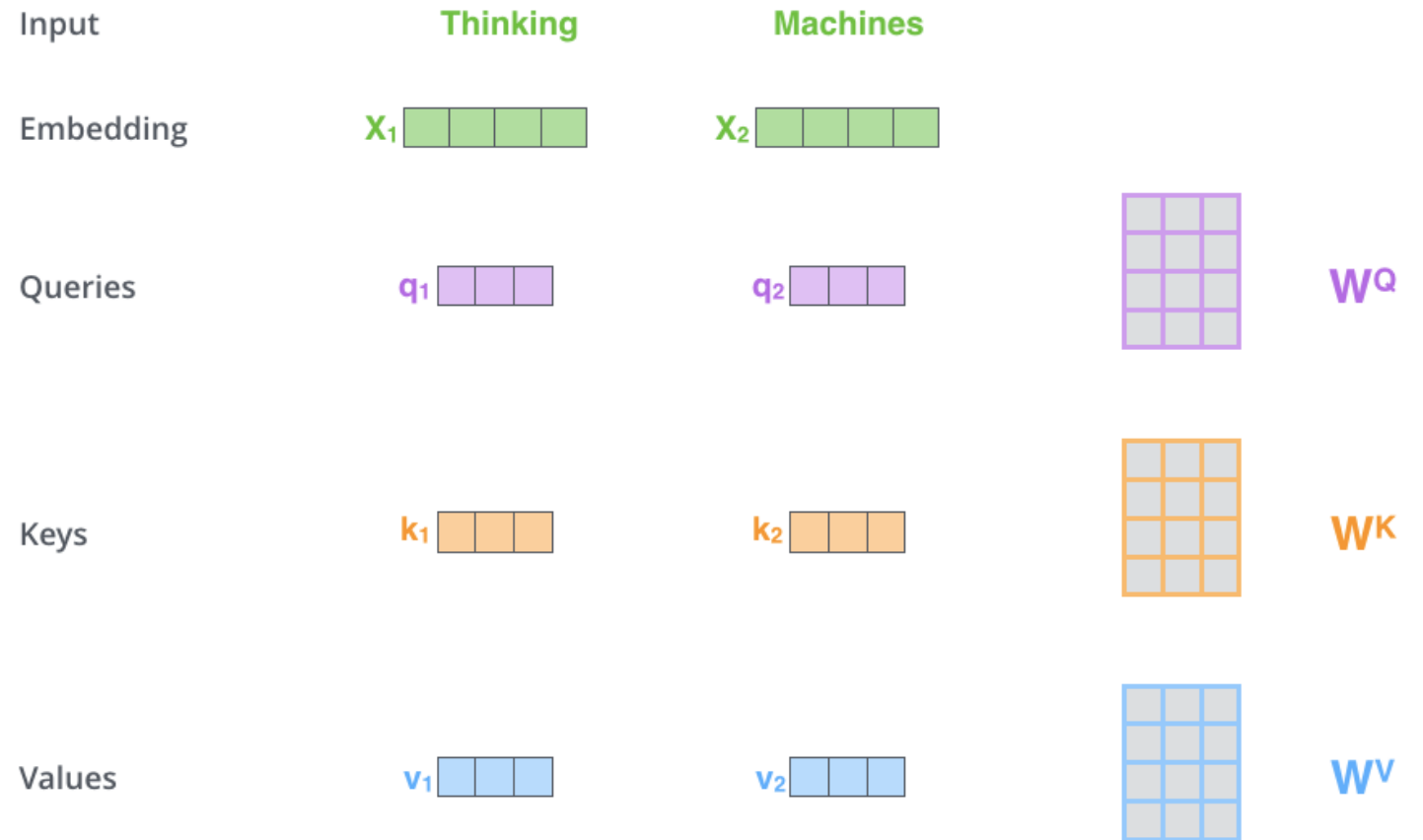
Step 1:

- Create three vectors from each of the encoder's input vectors

1. Query
2. Key
3. Value

- These vectors are created by multiplying the embedding by three matrices that we trained during the training process

- W^Q
- W^K
- W^V



Self Attention Layer (2/3)

Step 2:

- Calculate the score vector for each word
- Dot product of the **query vector** with the **key vector** of the respective word we're scoring

Input

Embedding

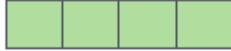
Queries

Keys

Values

Score

Thinking

x_1 

q_1 

k_1 

v_1 

$$q_1 \cdot k_1 = 112$$

Machines

x_2 

q_2 

k_2 

v_2 

$$q_1 \cdot k_2 = 96$$

Self Attention Layer (3/3)

Step 3:

- Divide by 8

Step 4:

- Softmax

Step 5:

- Multiply each value vector by the softmax score

Step 6:

- Sum up the weighted value vectors

Input

Embedding

Queries

Keys

Values

Score

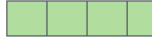
Divide by 8 ($\sqrt{d_k}$)

Softmax

Softmax
X
Value

Sum

Thinking

x_1 

q_1 

k_1 

v_1 

$q_1 \cdot k_1 = 112$

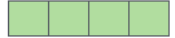
14

0.88

v_1 

z_1 

Machines

x_2 

q_2 

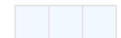
k_2 

v_2 

$q_1 \cdot k_2 = 96$

12

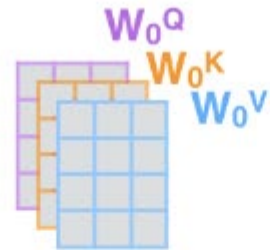
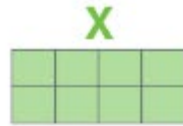
0.12

v_2 

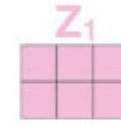
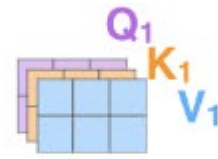
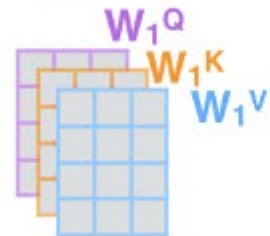
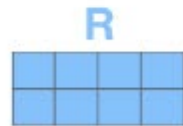
z_2 

Self Attention Layer – Multi Headed

Thinking
Machines



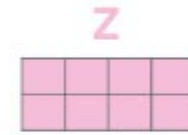
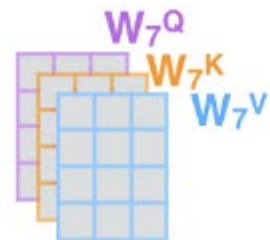
* In all encoders other than #0,
we don't need embedding.
We start directly with the output
of the encoder right below this one



...

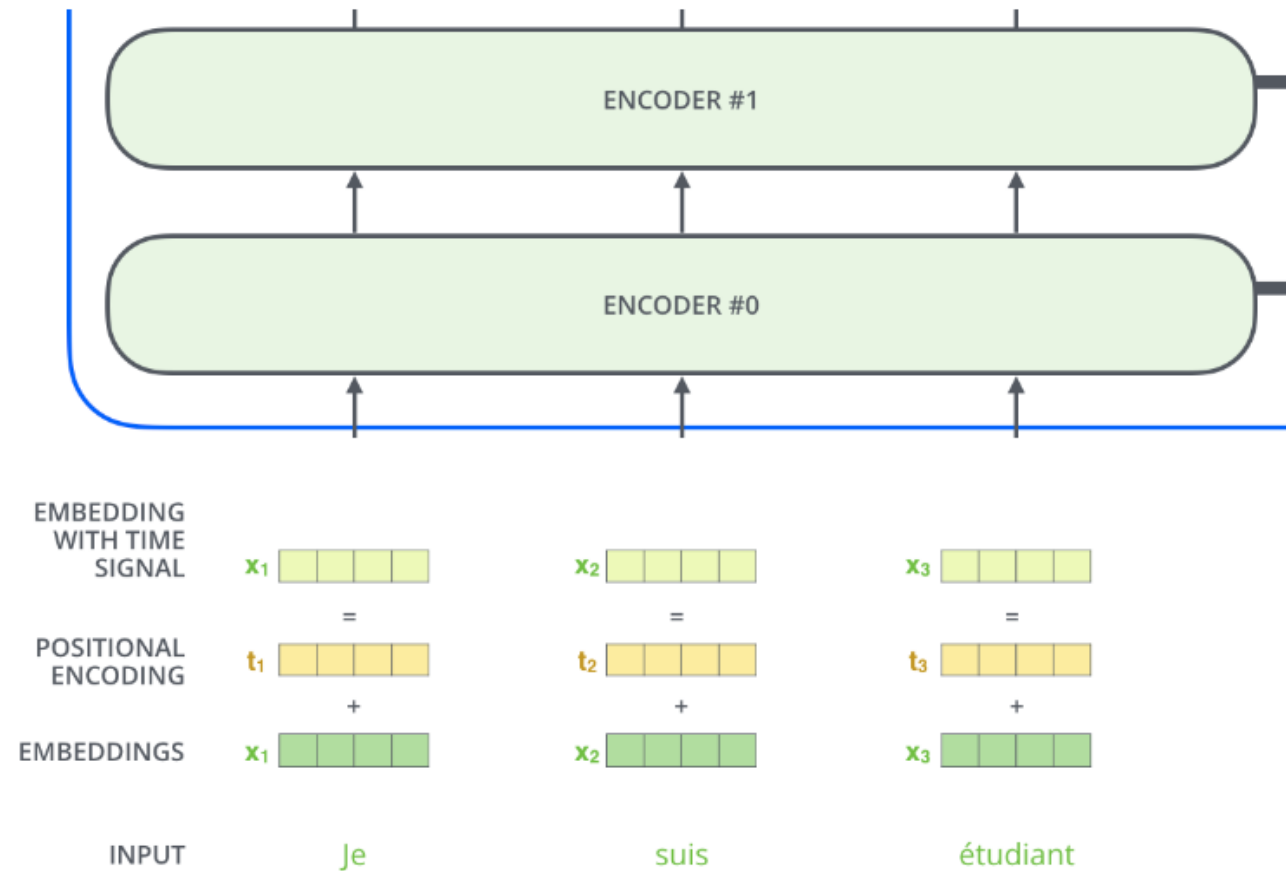
...

...

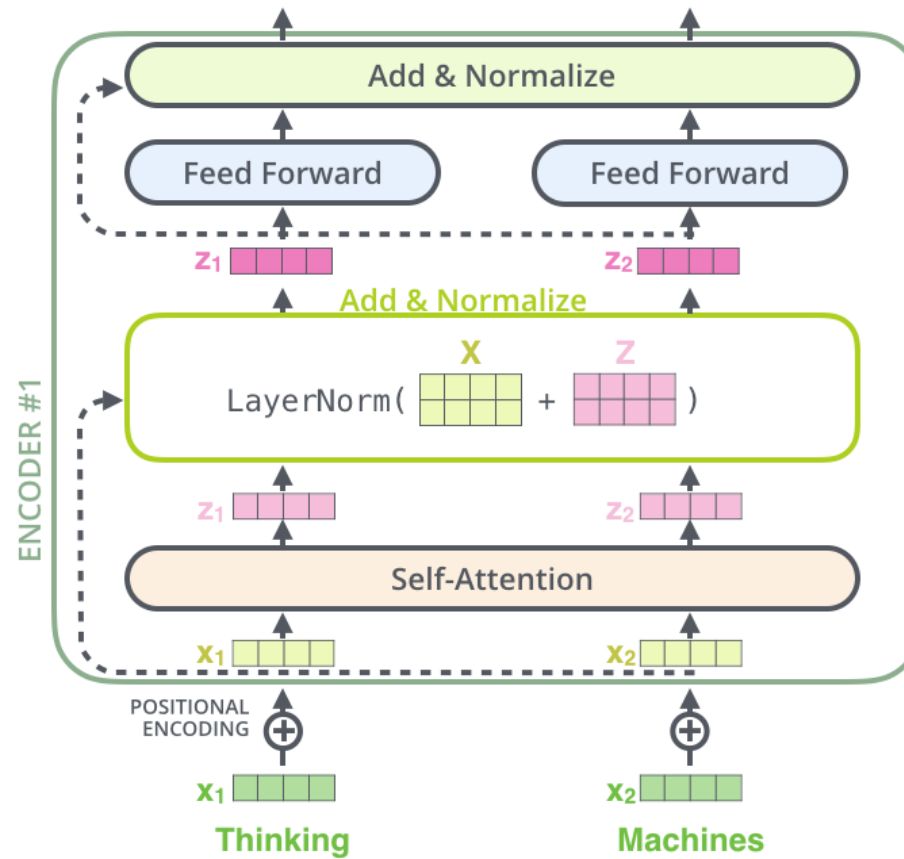


Positional Encoding

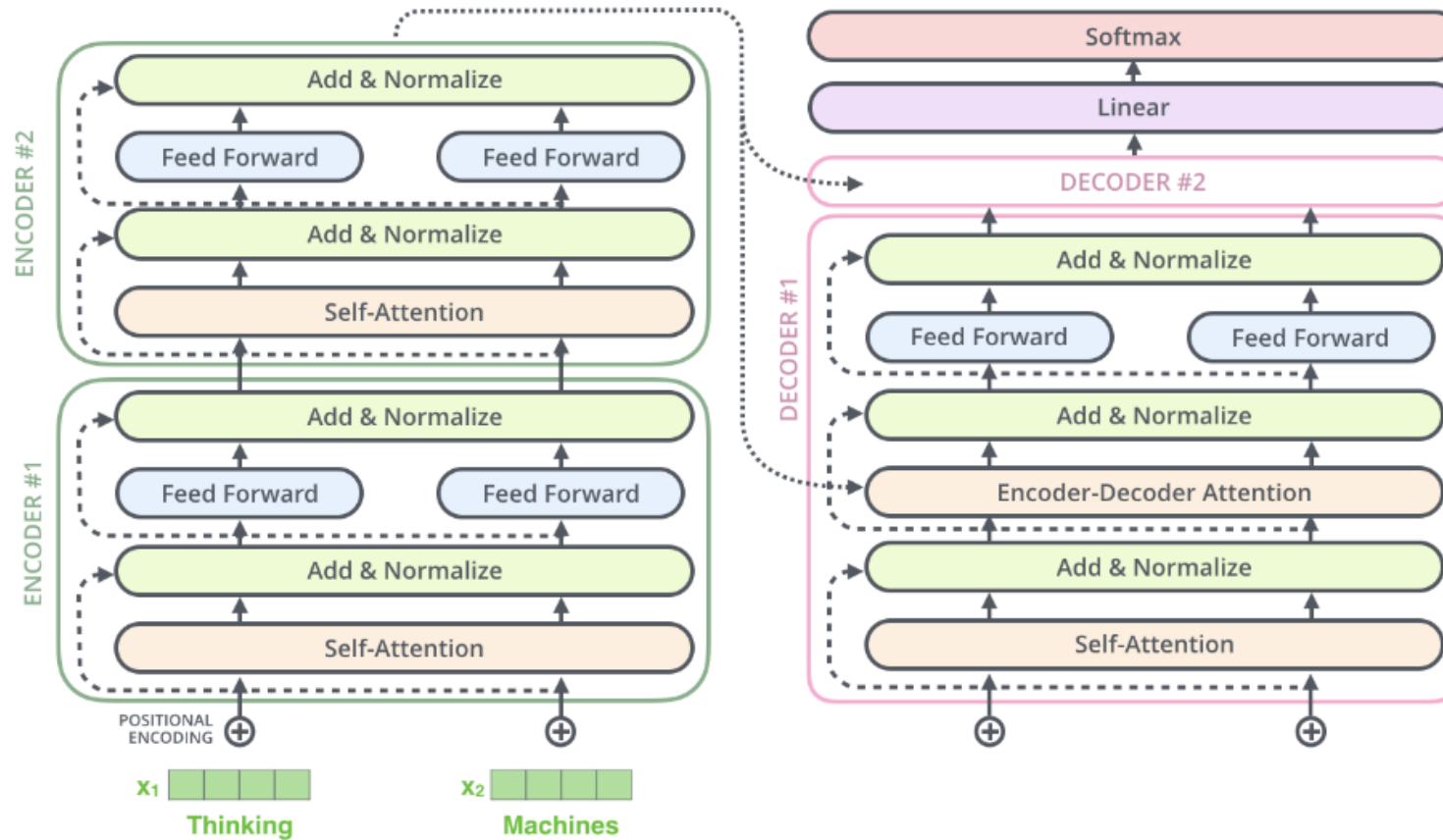
Positional Encoding is a way to account the order of the words in the input sequence.



More Details



More Details



The Decoder

The encoder starts by processing the input sequence.

The output of the top encoder is then transformed into a set of attention vectors K and V .

These are to be used by each decoder in its “encoder-decoder attention” layer which helps the decoder focus on appropriate places in the input sequence.

<https://jalammar.github.io/illustrated-transformer/>

BERT

BERT

Main issue with word2vec:

- Fixed Embeddings

He didn't receive fair treatment

Fun fair in New York City this summer

$$\text{fair} = \begin{bmatrix} 1 \\ 0.3 \\ 0.5 \\ 0.1 \\ 0.7 \end{bmatrix}$$

We need a model that can generate a contextualized meaning of a word.

BERT

BERT stands for Bidirectional Encoder Representations from **Transformers**.

BERT is basically a trained Transformer Encoder stack.

The BERT was pre-trained using text from Wikipedia and can be fine-tuned.

It is pre-trained on two different NLP tasks:

- Masked Language Modeling
- Next Sentence Prediction

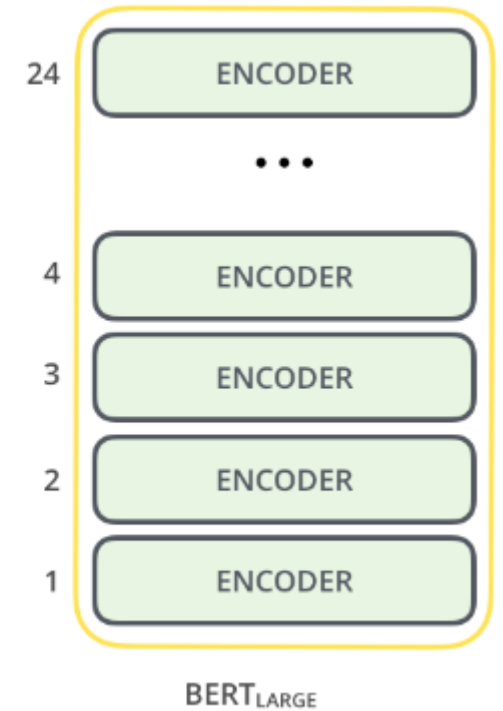
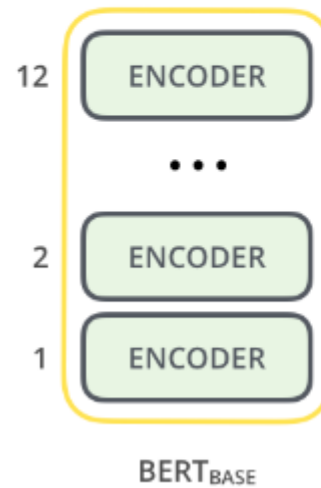
Bert can generate the contextualized meaning of a word.

It can also generate an embedding for sentences.

BERT

The paper presents two model sizes for BERT:

- BERT BASE
- BERT LARGE



BERT – Sentence Classification

