CIS 6930 Topics in Computing for Data Science
Week 8: Transformers (1)

10/19/2021

Yoshihiko (Yoshi) Suhara

2pm-3:20pm

Week 8: Transformers

- **Week 8: Transformers**
- **Week 9: Pre-trained Language Models**
- Week 10: More Machine Learning Techniques
- Week 11: More Deep Learning Techniques for NLP (Text generation, Text summarization, Information Extraction etc.)
- Week 12: Advanced Techniques and Challenges
- Week 13: Final project presentations

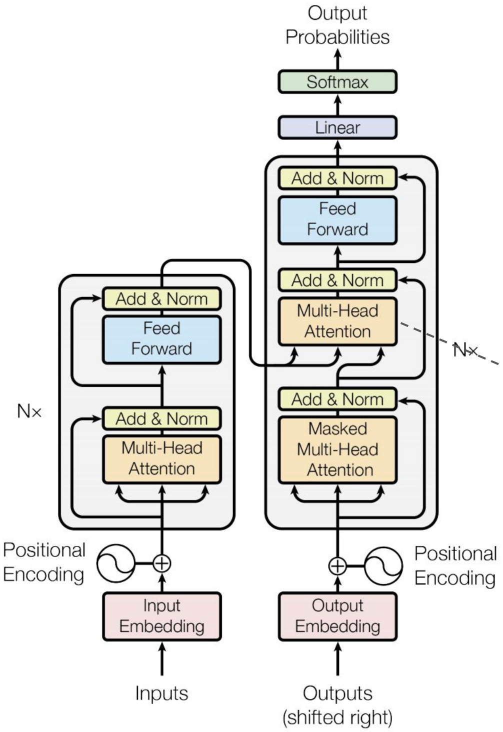
Attention!

- Today's content is extremely important for understanding modern Deep Learning techniques

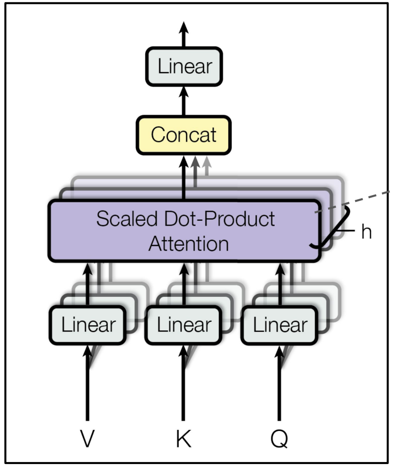
- The content might be technically difficult and complicated 
  - Don't worry. The concept is relatively simple

- Please make sure that you follow the explanation of each step
  - Don't hesitate to interrupt and ask questions if you don't

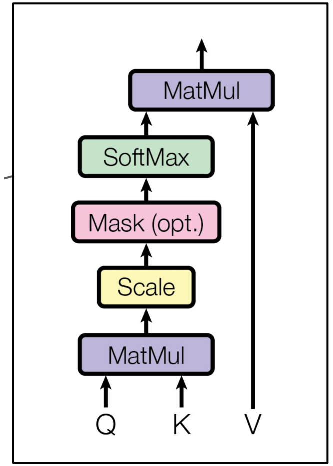
This Week's Goal:
Understanding the Transformer Architecture



Model Architecture



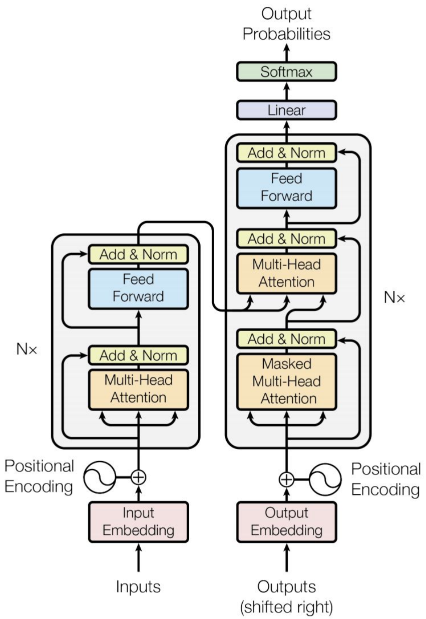
Multi-Head Attention



Scaled Dot-Product Attention

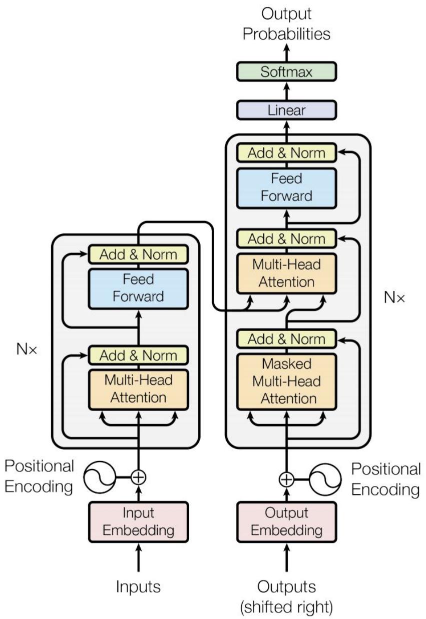
What is Transformer and Why So Important?

- Yet another **Neural Network architecture** that replaces CNN or RNN

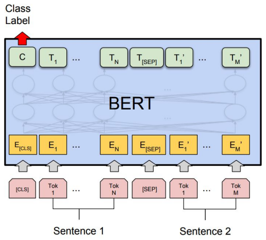


What is Transformer and Why So Important?

- Yet another **Neural Network architecture** that replaces CNN or RNN



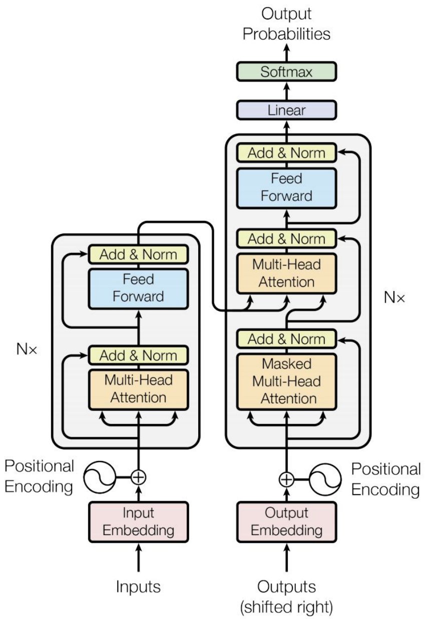
Pre-trained Language Models



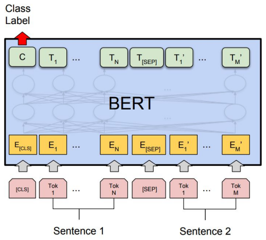
OpenAI
GPT-3

What is Transformer and Why So Important?

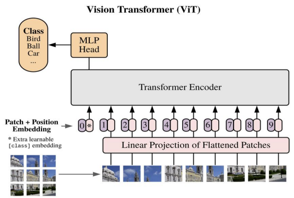
- Yet another **Neural Network architecture** that replaces CNN or RNN

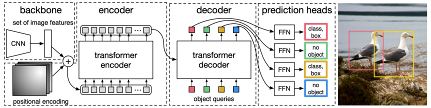


Pre-trained Language Models



Transformers for Computer Vision

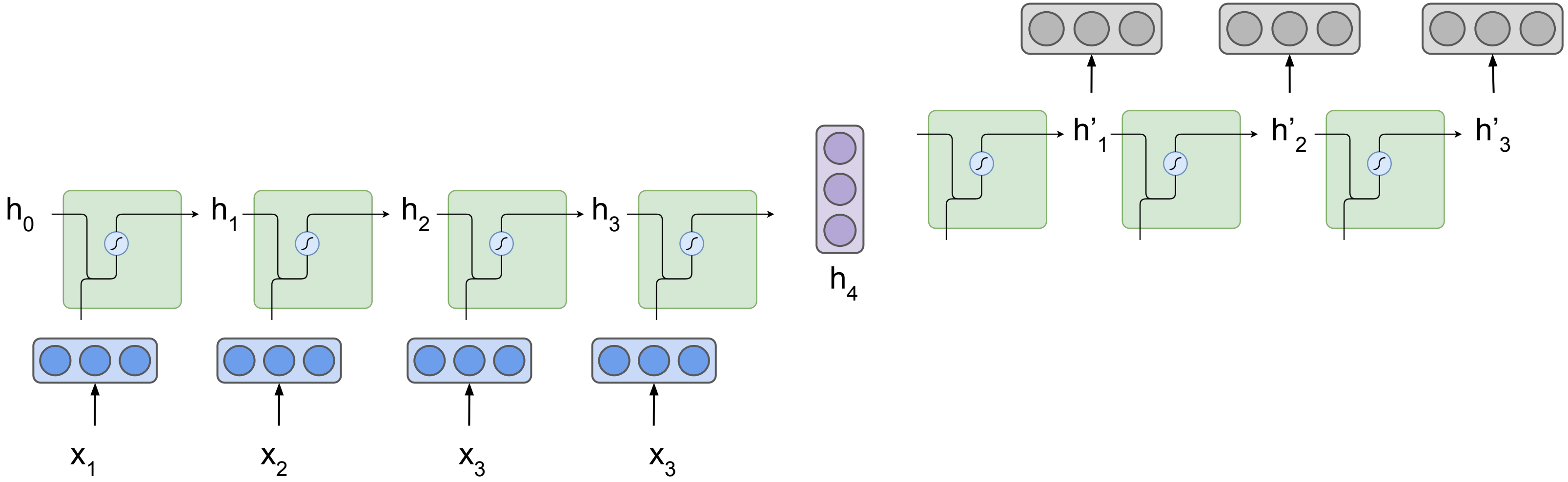




Quick Recap + Attention Mechanism (Important Background)

Encoder-Decoder Model
aka Sequence-to-sequence (seq2seq) Model

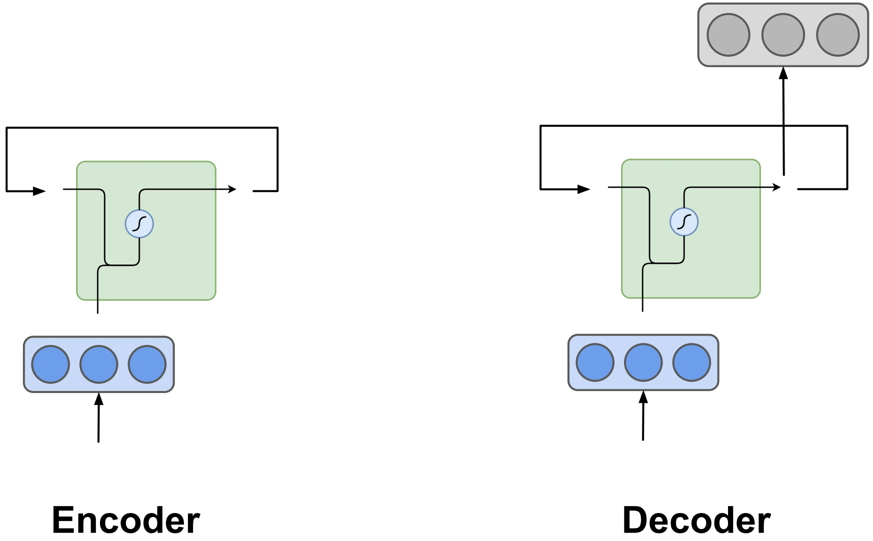
- Input: Variable length sequence
- Output: Variable length sequence



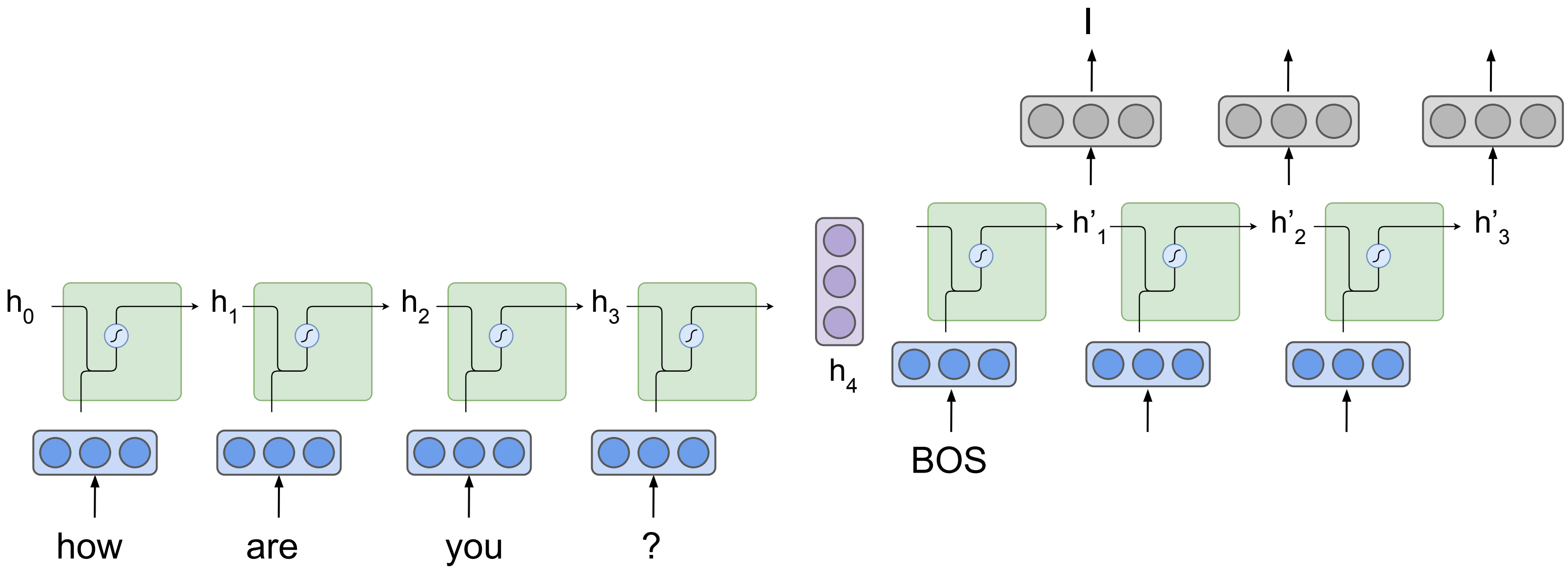
(* RNN Cell can be LSTM/GRU Cell)

Encoder Model + Decoder Model

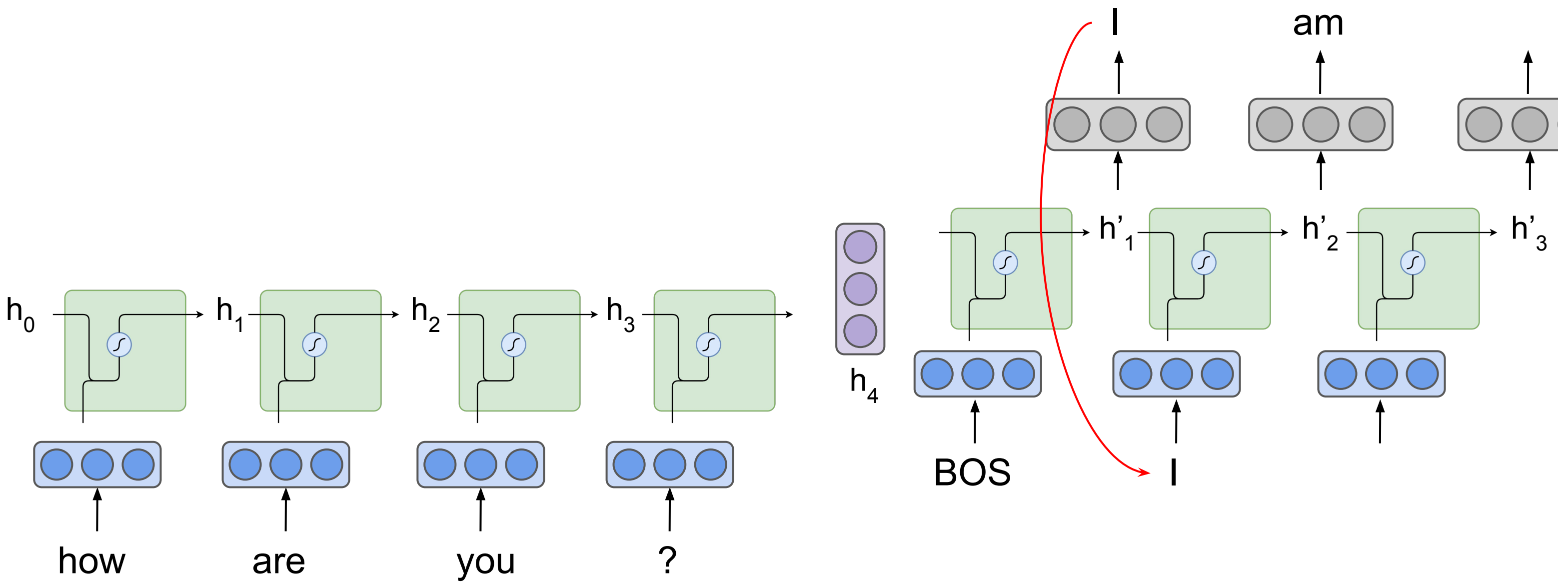
- Two different RNN models



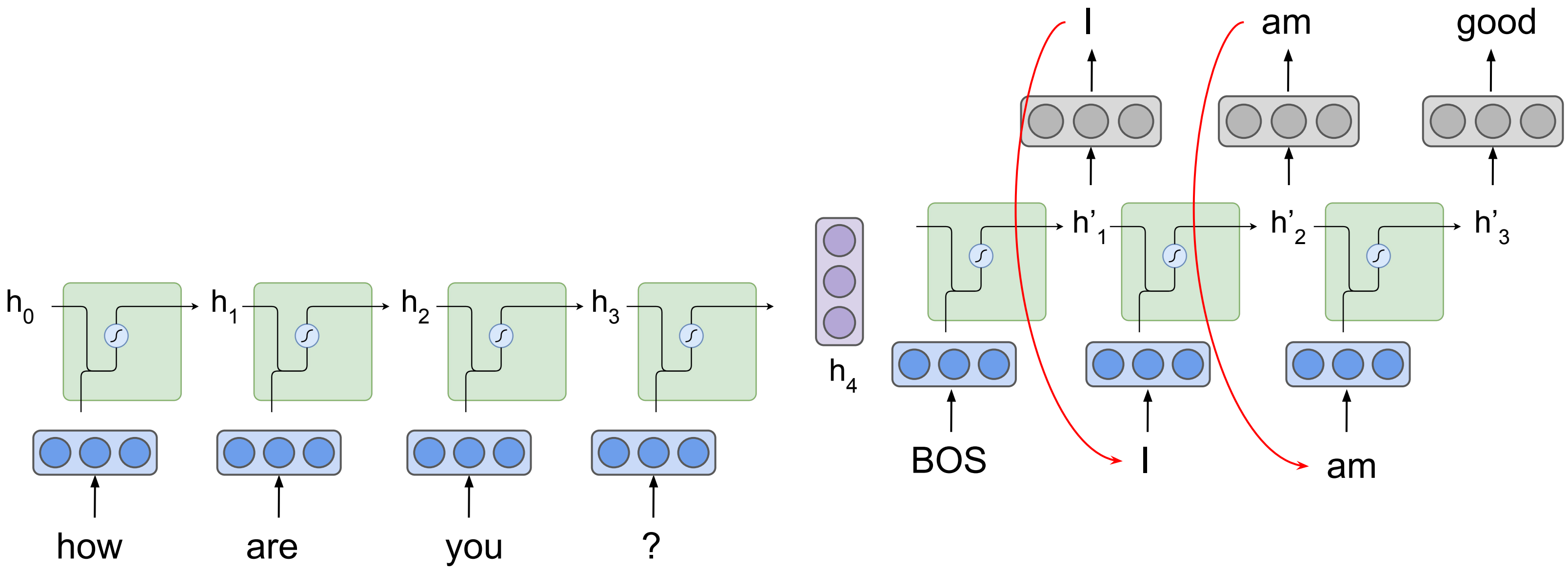
Example 1: Response Generation



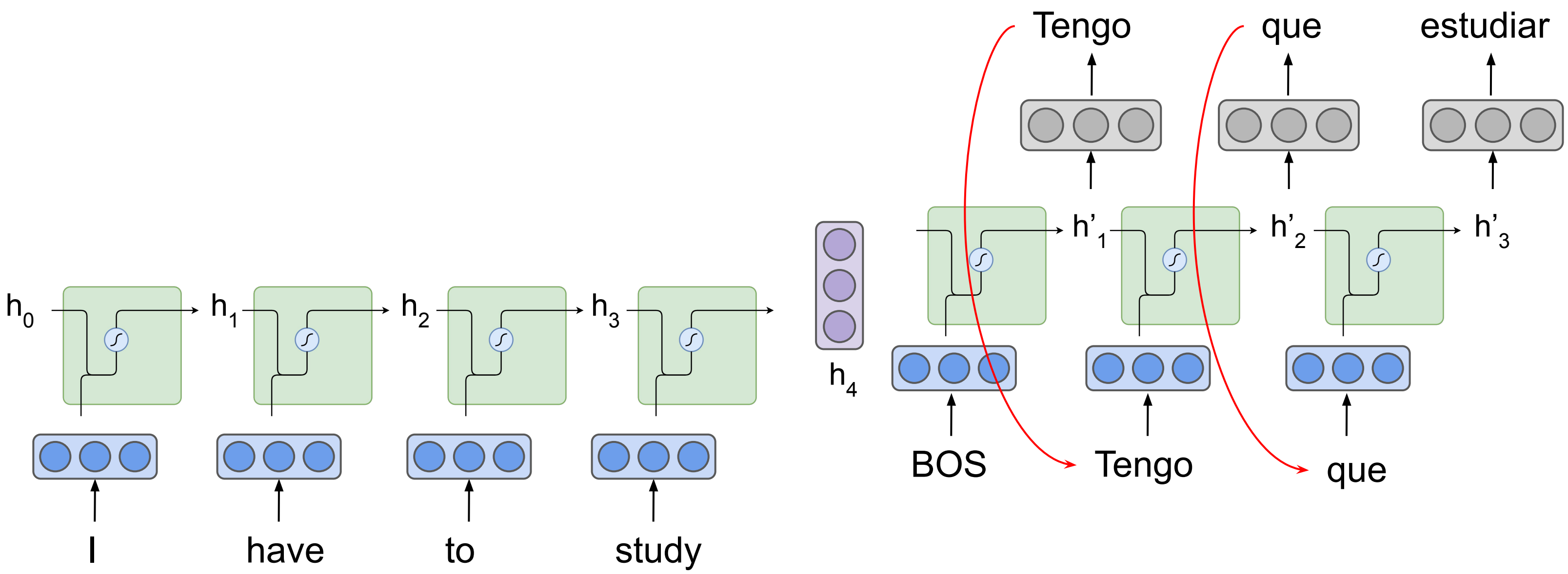
Example 1: Response Generation



Example 1: Response Generation

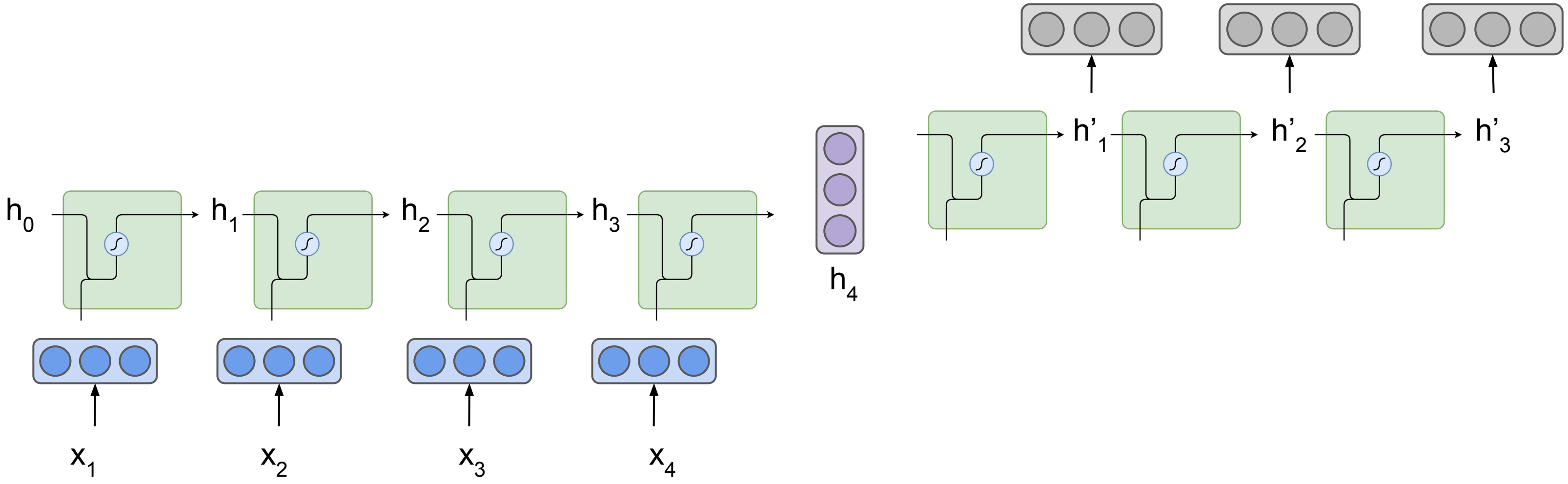


Example 2: Machine Translation



Check: Input & Output

- **Token ID** → One-hot vector → Dense vector → **RNN**
- **RNN** → Output layer + softmax → **Token ID**

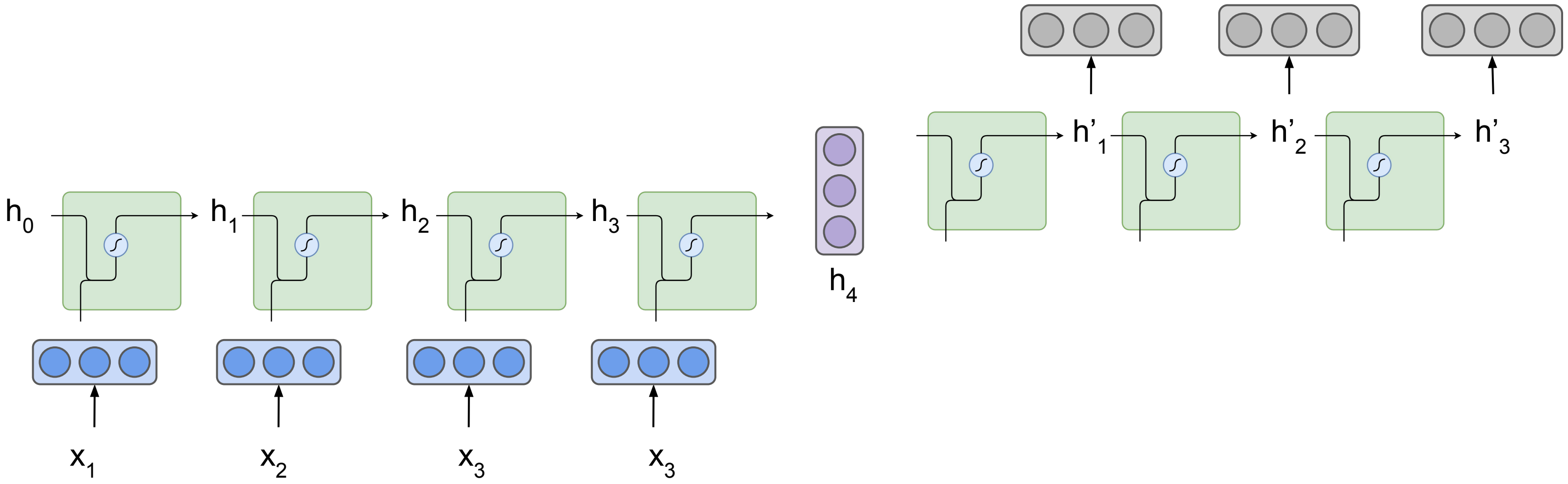


Key Concept

- Encoder-Decoder Model

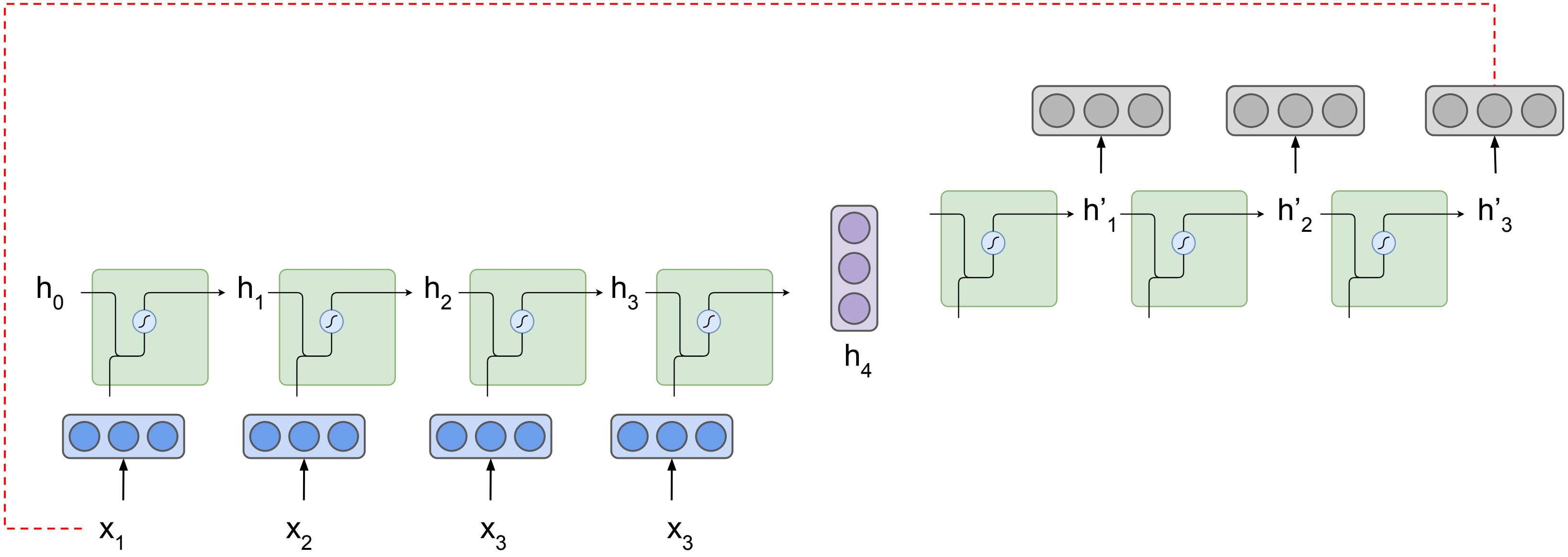
  = Encoder + Decoder + Decoding algorithm

  Model **Not model**



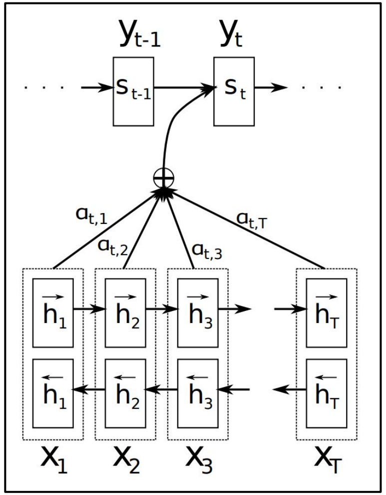
What's the issue with Encoder-Decoder Models?

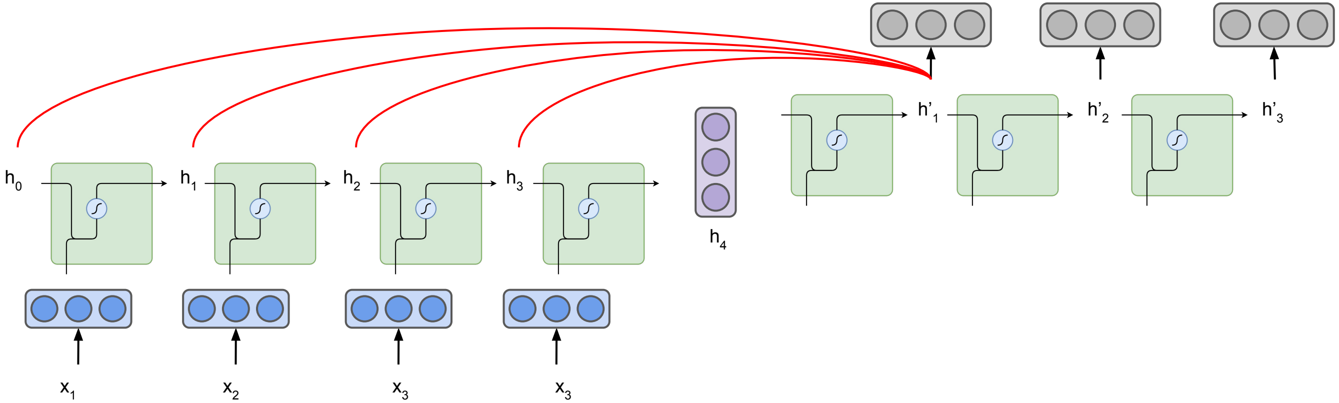
- Not a very good design for **long input/output sequence** even with LSTM/GRU
  - e.g., dependency b/w the beginning of input & the end of output



Attention Mechanism [Bahdanau, Cho, and Bengio ICLR 2016]

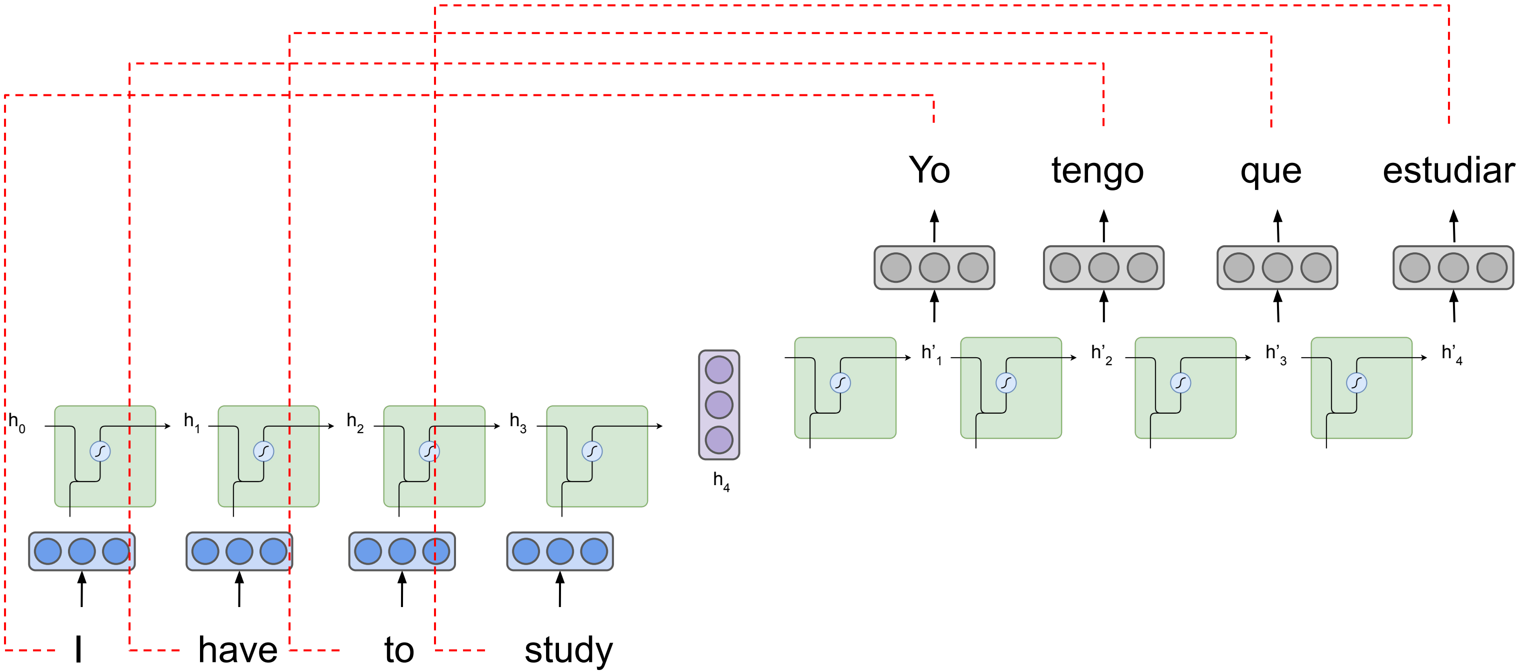
- Additional input to the decoder based on **the alignments b/w encoder and decoder steps**



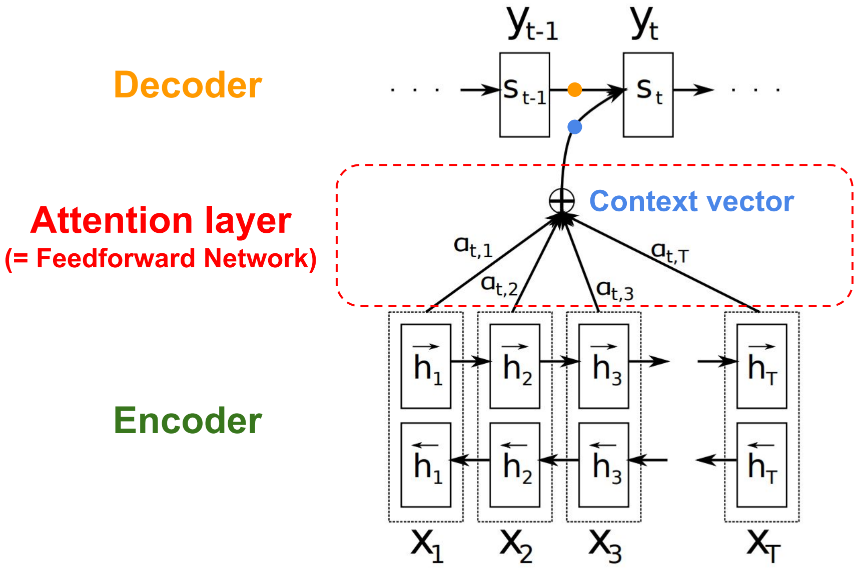


Attention Mechanism: Intuition

- “Direct” connections between any steps in the encoder/decoder model



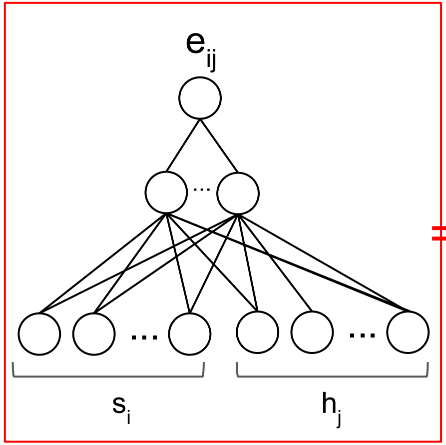
Attention Mechanism: Attention Layer

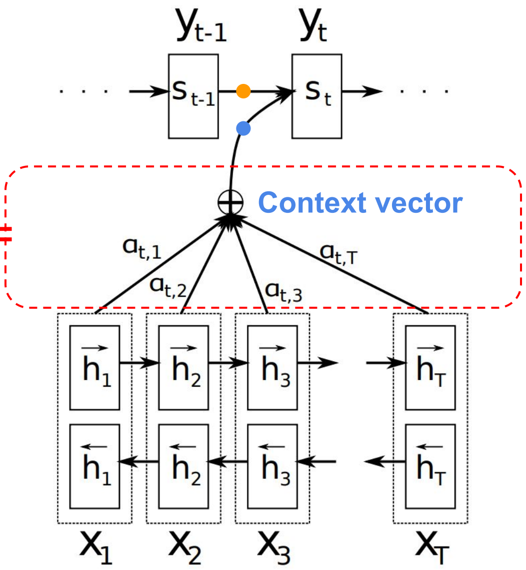


Previous hidden state
+ context vector

$$c_i = \sum_{j=1}^{T_x} \underbrace{\alpha_{ij}}_{\text{alignment score}} h_j$$

Attention Layer in Depth





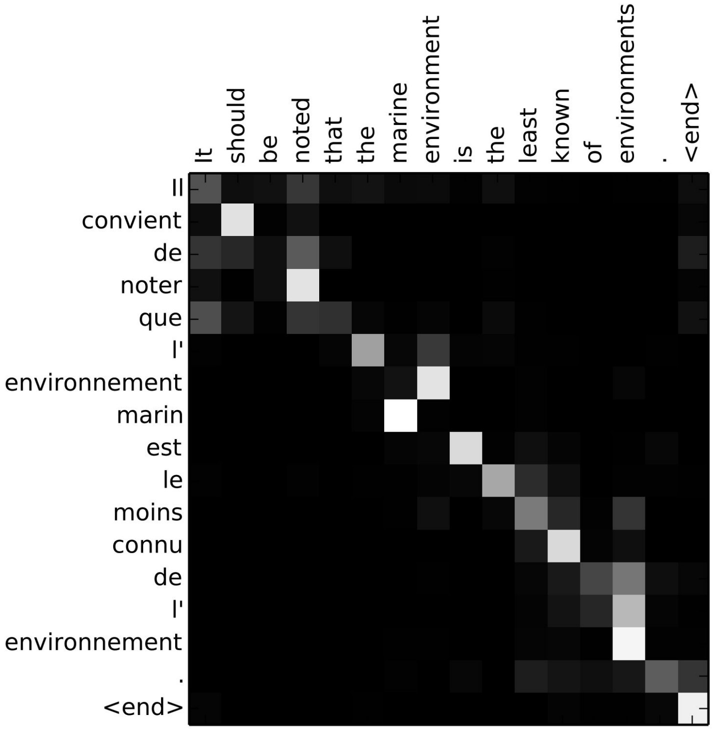
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

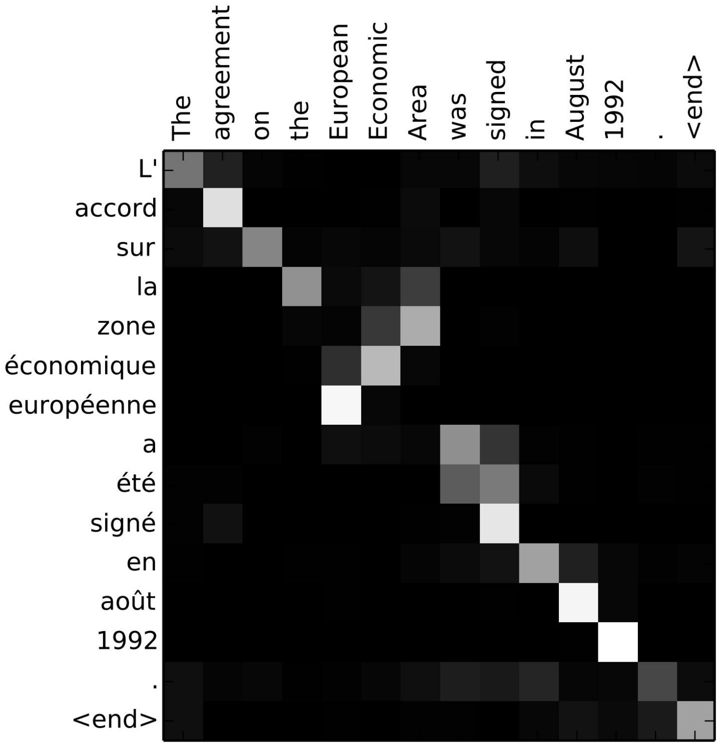
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

$$e_{ij} = \text{score}(s_i, h_j)$$

$$\text{score}(s_i, h_j) = \mathbf{v}_\mathbf{a}^\mathbf{T} \tanh(\mathbf{W}_a[s_i; h_j])$$

Attention Weight Analysis

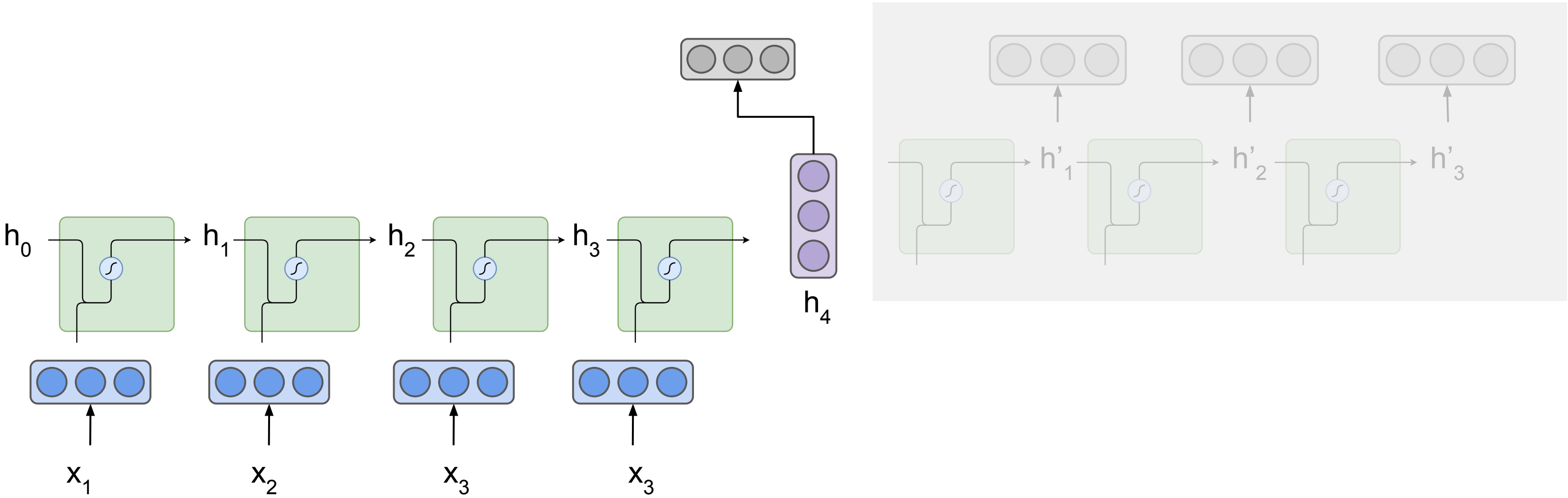




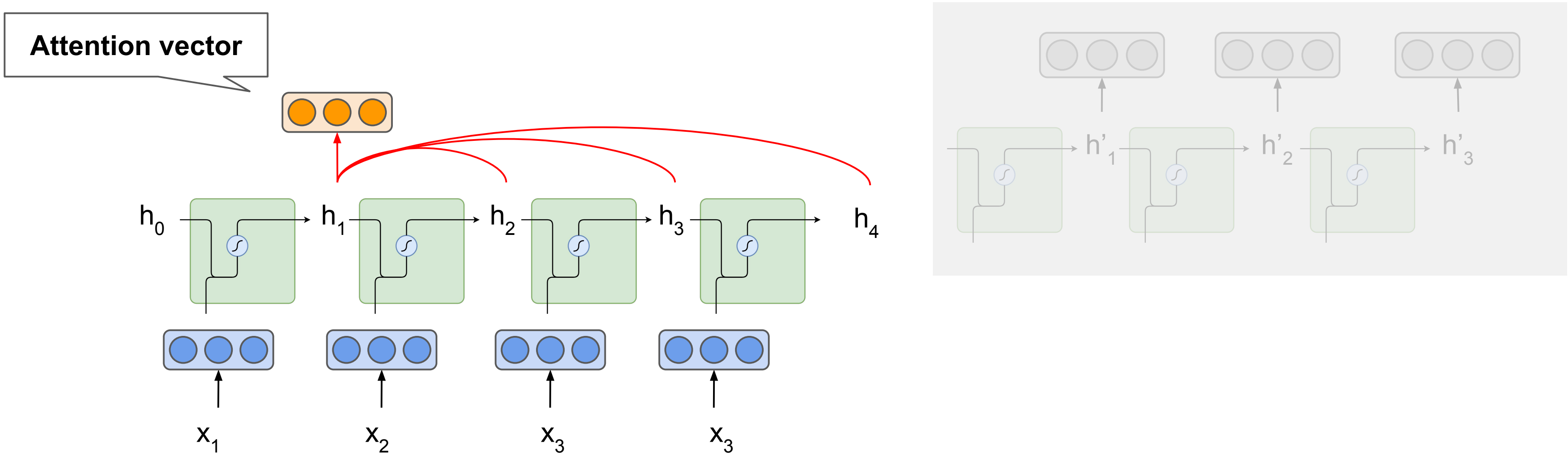
Alignment Score Options

	Name	Alignment score function	Citation
	Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Original →	Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
	Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
	General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
	Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Transformer →	Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

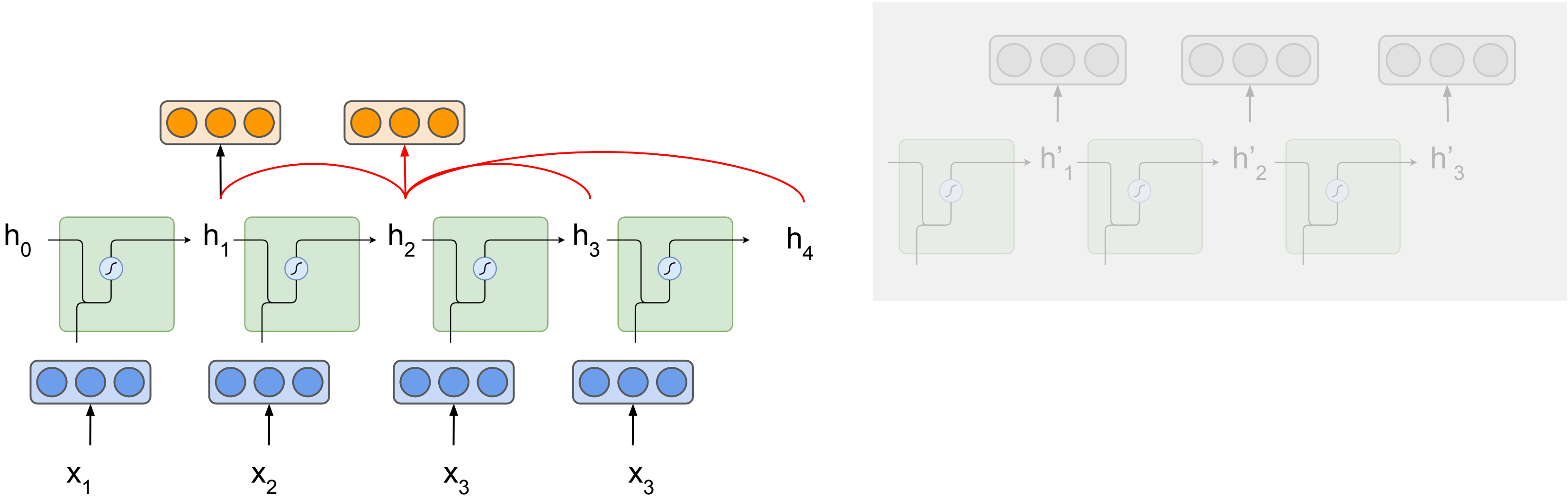
Self-Attention: Attention for Encoder-only Models



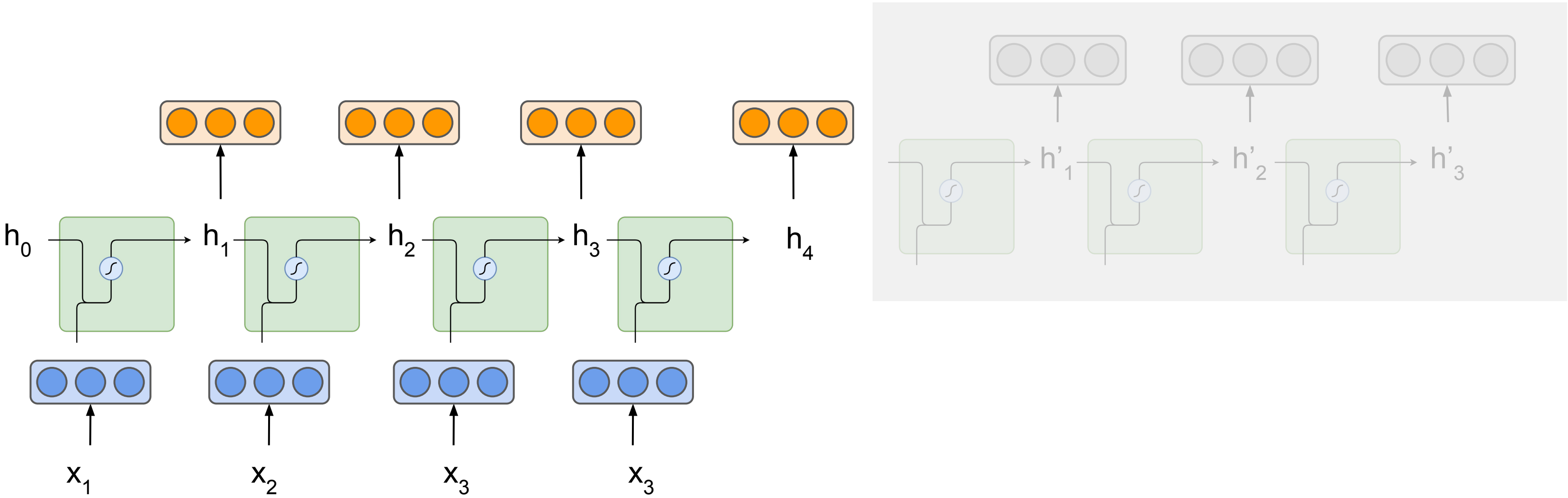
Self-Attention: Attention for Encoder-only Models



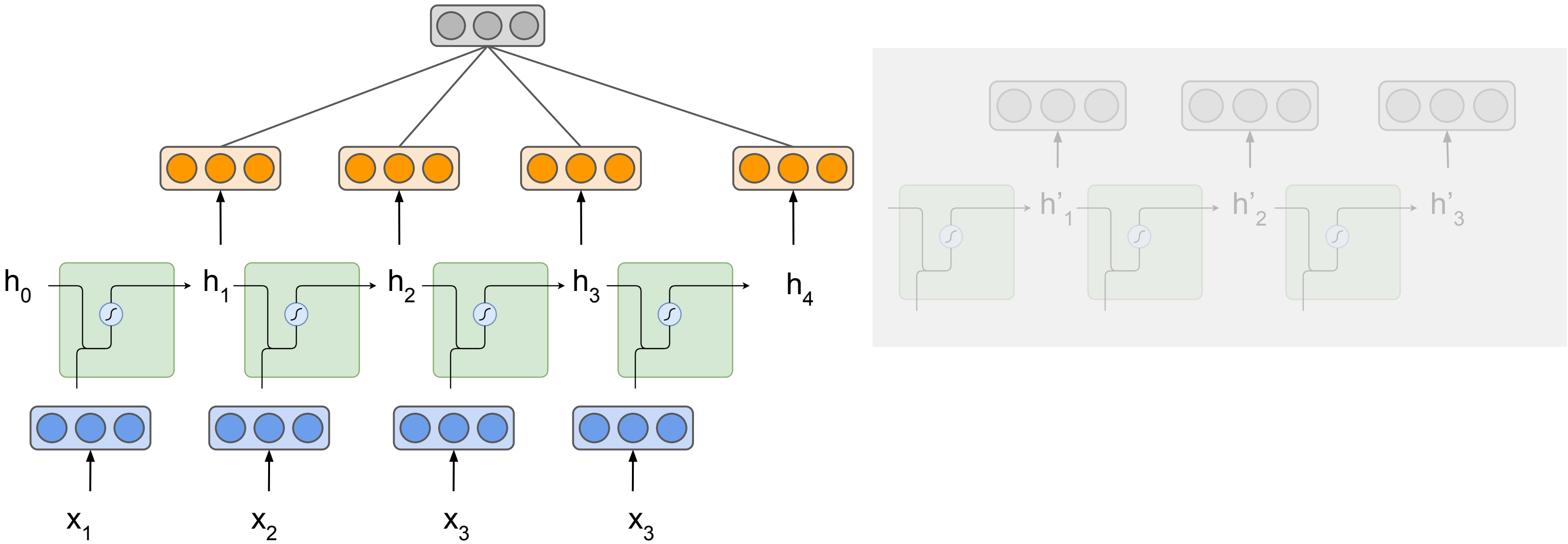
Self-Attention: Attention for Encoder-only Models



Self-Attention: Attention for Encoder-only Models

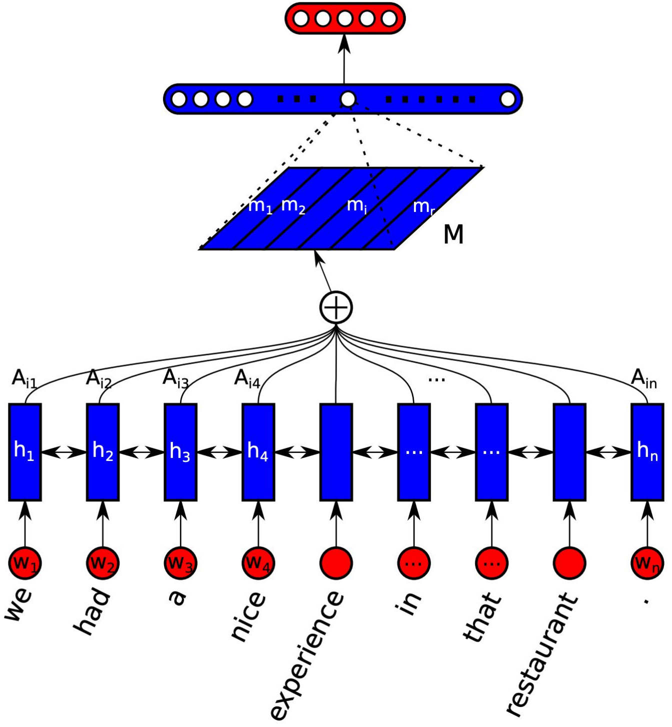


Self-Attention: Attention for Encoder-only Models



Self-Attention Mechanism for Text Classification

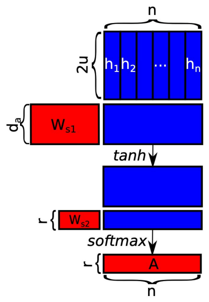
BiLSTM RNN



$$M = AH$$

$$A = softmax\left(W_{s2} tanh\left(W_{s1} H^T\right)\right)$$

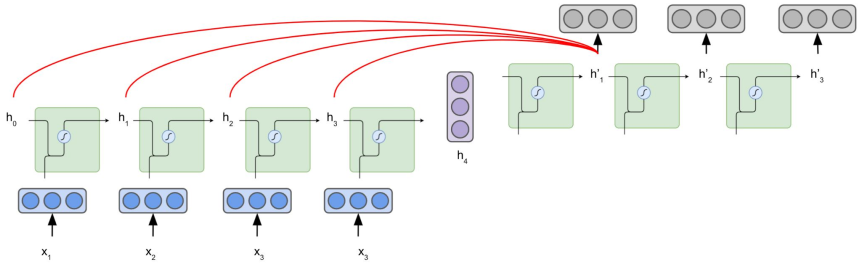
$$H = (\mathbf{h_1}, \mathbf{h_2}, \cdots \mathbf{h_n})$$



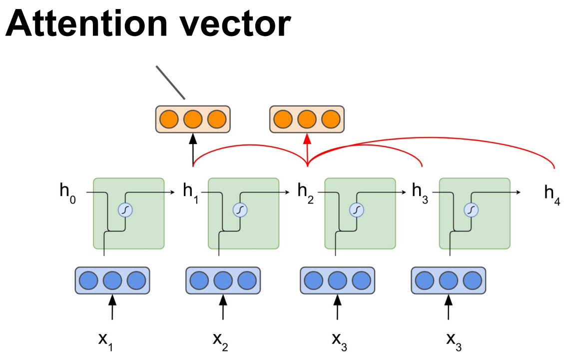
Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, Yoshua Bengio, A Structured Self-attentive Sentence Embedding, ICLR 2017.

Terminology: Attention Mechanism

- Encoder-decoder attention (aka cross-attention)
- Self-attention
- Attention vector



Encoder-decoder attention



Self-attention

Questions?

The Rise of the Transformer

Hey, the attention mechanism is so strong! RNN models with attentions establish new state-of-the-art performance on many many tasks!

Hey, the attention mechanism is so strong! RNN models with attentions establish new state-of-the-art performance on many many tasks!

Cool! What makes it so strong?

Hey, the attention mechanism is so strong! RNN models with attentions establish new state-of-the-art performance on many many tasks!

Cool! What makes it so strong?

The attention helps the RNN model incorporate **direct connections between any steps**

Hey, the attention mechanism is so strong! RNN models with attentions establish new state-of-the-art performance on many many tasks!

Cool! What makes it so strong?

The attention helps the RNN model incorporate **direct connections between any steps**

That makes sense! Then, is the **RNN cell still necessary?**

🐶 Hey, the attention mechanism is so strong! RNN models with attentions establish new state-of-the-art performance on many many tasks!

Cool! What makes it so strong? 🐱

🐶 The attention helps the RNN model incorporate **direct connections between any steps**

That makes sense! Then, is the **RNN cell still necessary?** 🐱

🐶 🤔

Attention Is All You Need [Vaswani et al. 2017]

Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[*, †]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*, ‡]
illia.polosukhin@gmail.com

Transformer established new state-of-the-art performance on Machine Translation without RNN

Why is This Title?

- 2014: RNN [Sutskever+ NIPS '14] # the original seq2seq paper
- 2015: RNN + Attention [Bahdanau ICLR '15]
- ...
- 2017: Attention ← The Transformer!

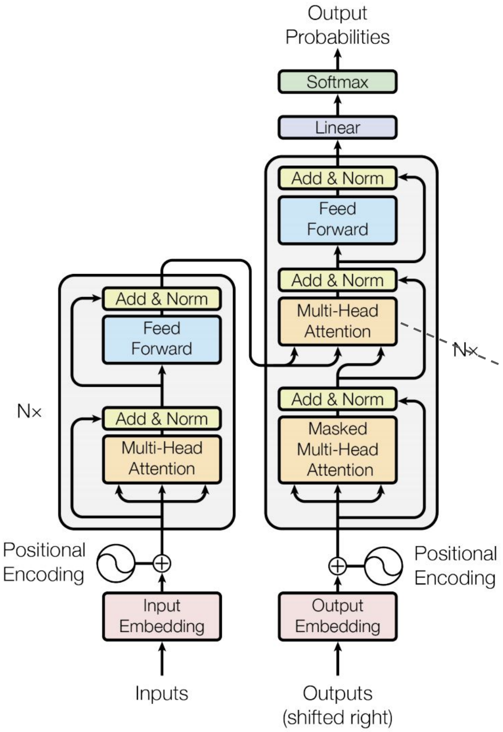
The claim is "recurrent architecture (RNN)" is **Unnecessary**

[Sutskever+ NIPS '14] Sequence to Sequence Learning with Neural Networks, NIPS '14
[Bahdanau ICLR '15] Neural Machine Translation by Jointly Learning to Align and Translate, ICLR '15

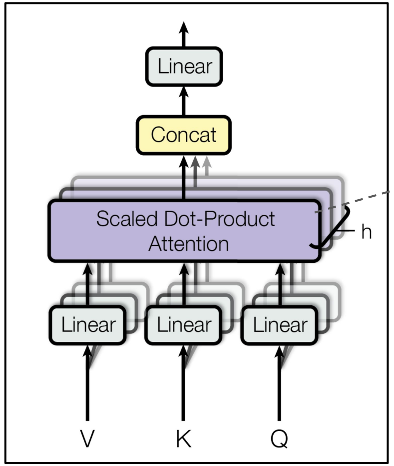
The Transformer: Basic Concepts

The figures and explanations are from the following excellent blog article (highly recommended)
[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

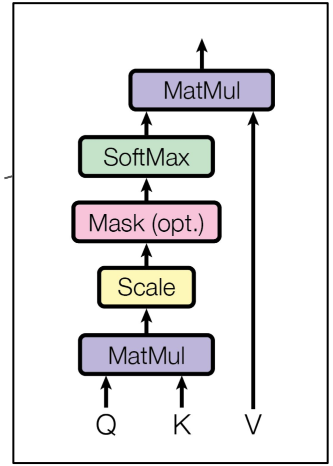
The Transformer



Model Architecture



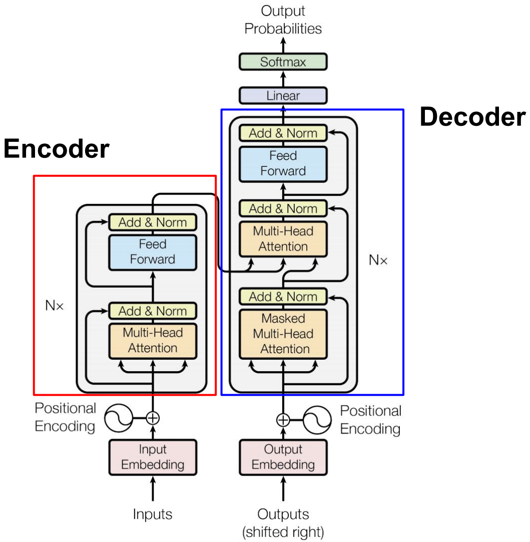
Multi-Head Attention



Scaled Dot-Product Attention

What is the Transformer?

- Transformer is an **Encoder-Decoder model!**
    - Can be used for any sequence-to-sequence generation tasks
- Transformer = **Transformer Encoder + Transformer Decoder**



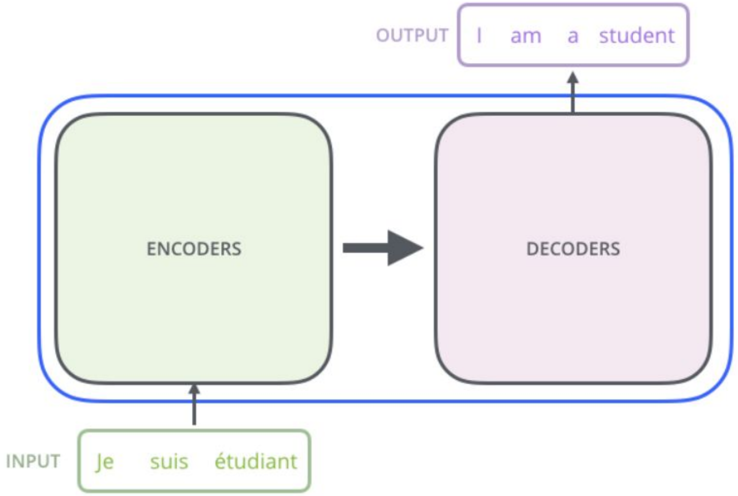
Transformer

100%

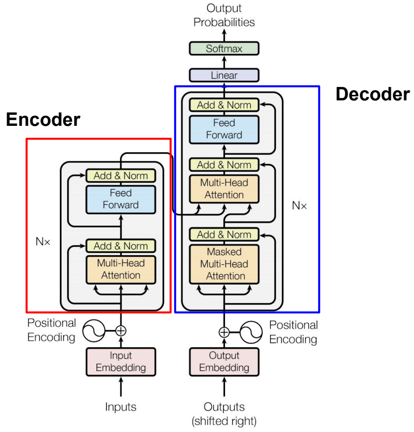
Transformer

100%

150%

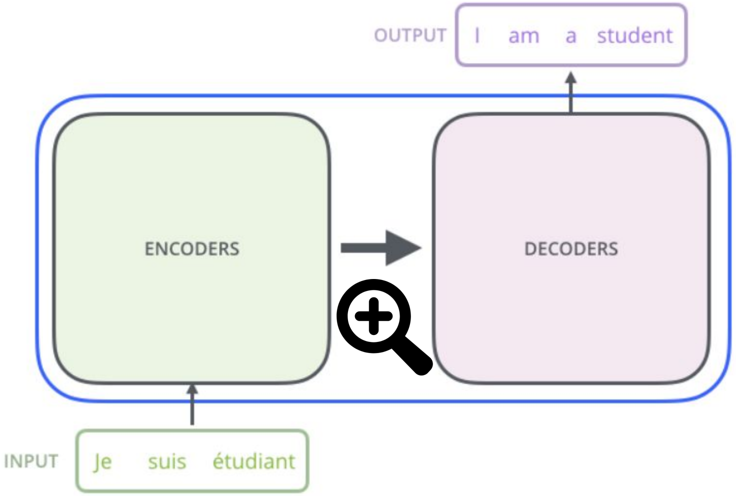


OUTPUT
I am a student
ENCODERS
DECODERS
INPUT
Je suis étudiant



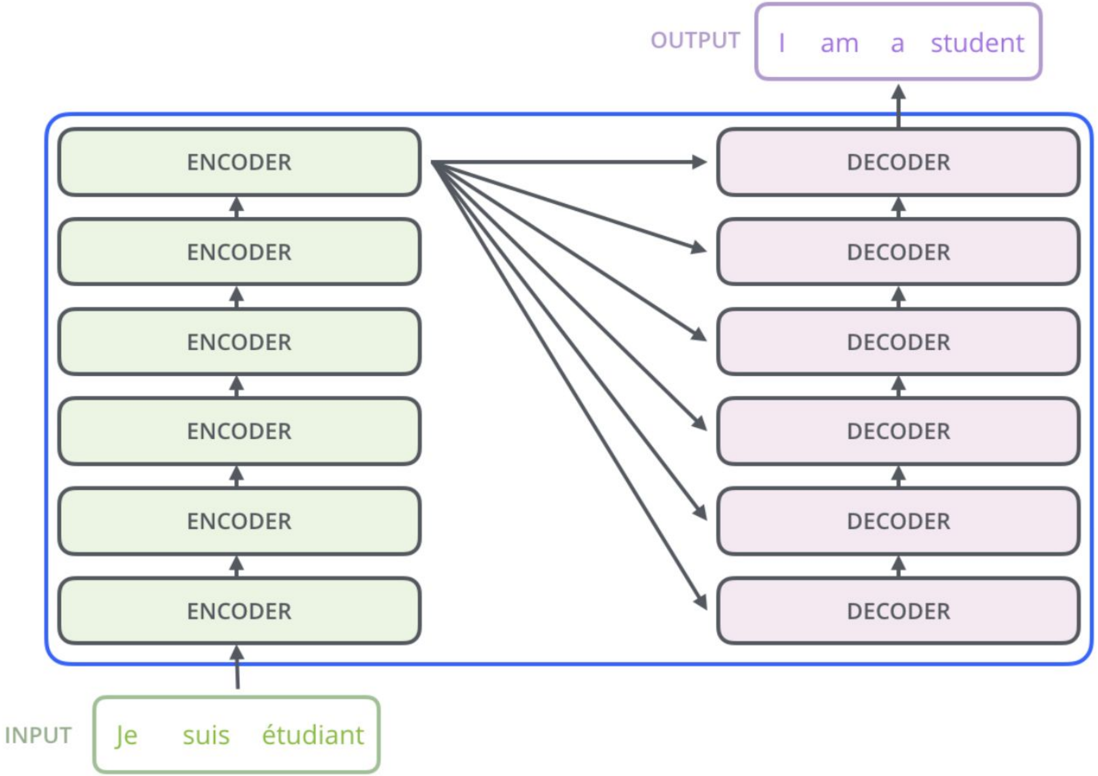
Output Probabilities
Softmax
Linear
Decoder
Encoder
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

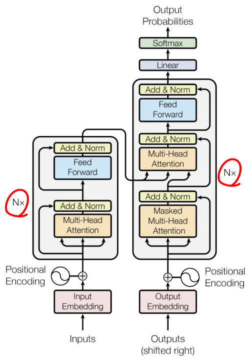
150%


OUTPUT
I am a student
ENCODERS
DECODERS
INPUT
Je suis étudiant

Encoder/Decoder
= Stack of Encoder/Decoder Blocks

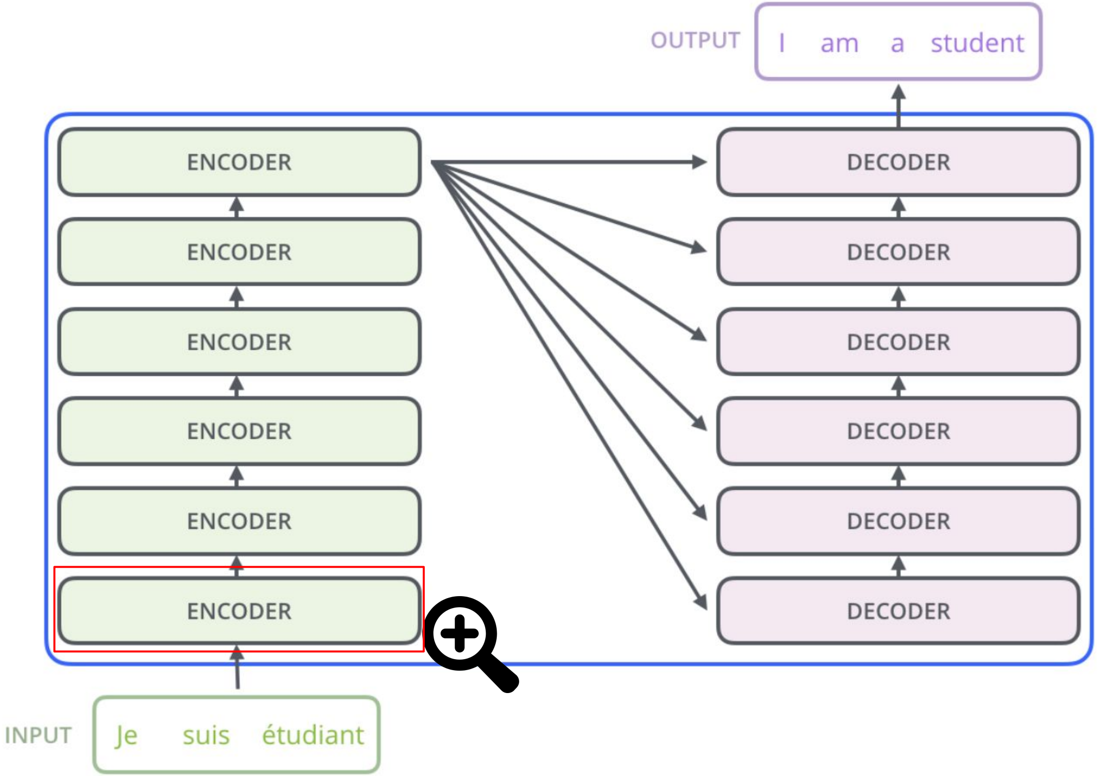
300%





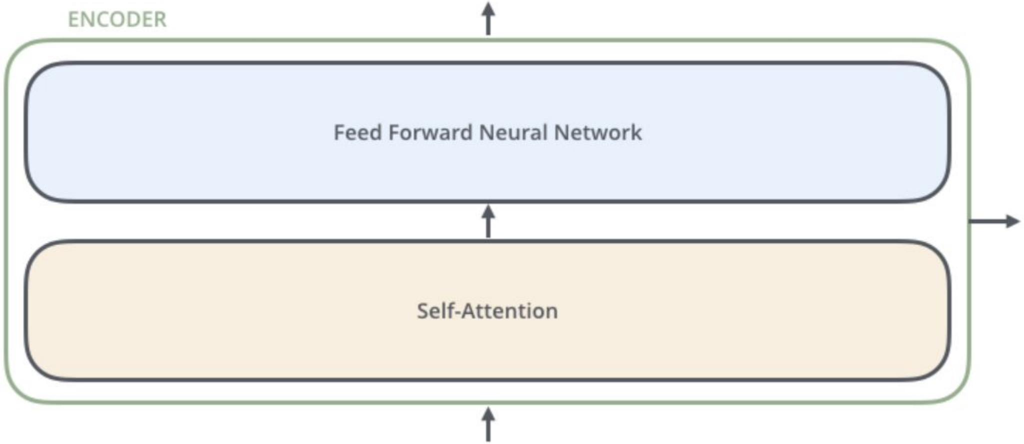
Encoder/Decoder
= Stack of Encoder/Decoder Blocks

300%

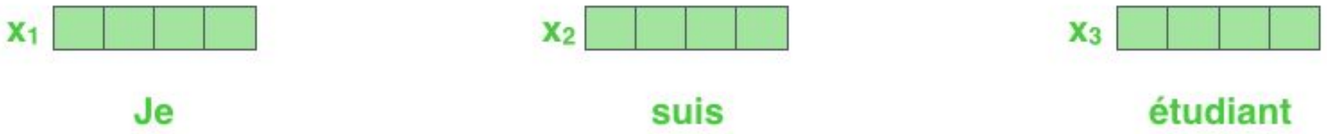


Encoder Block
= Self-Attention + Feed Forward NN

500%

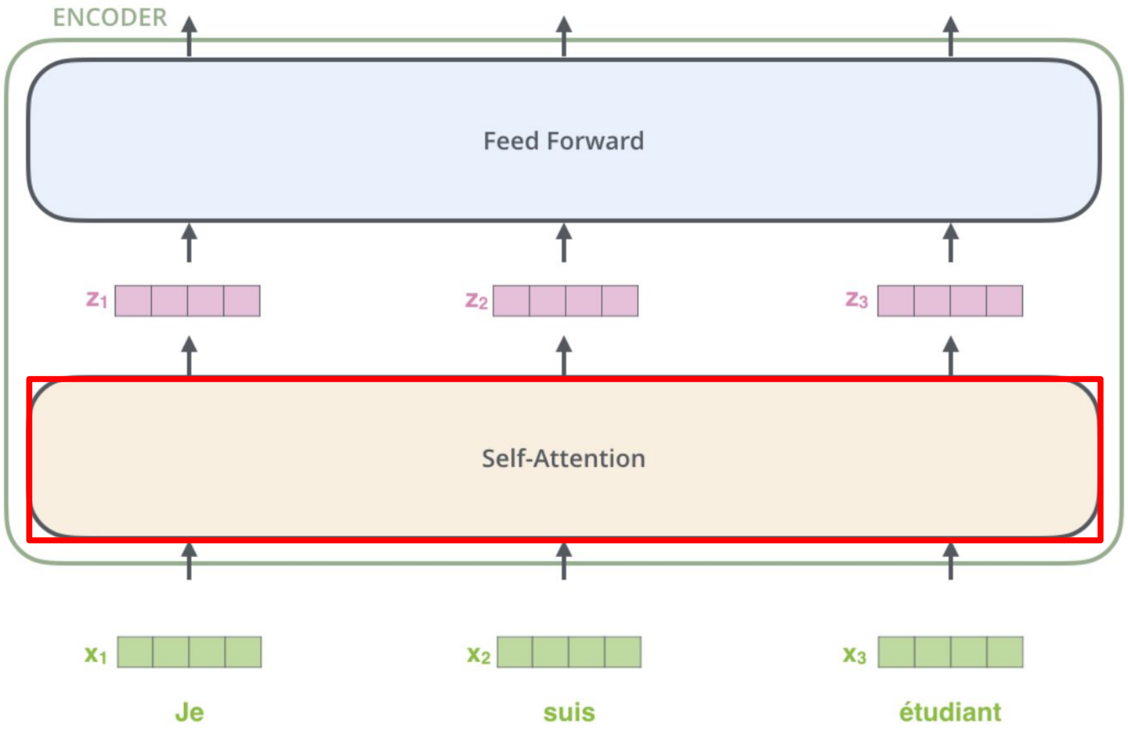


Step 1: Input Tokens → Token Embeddings



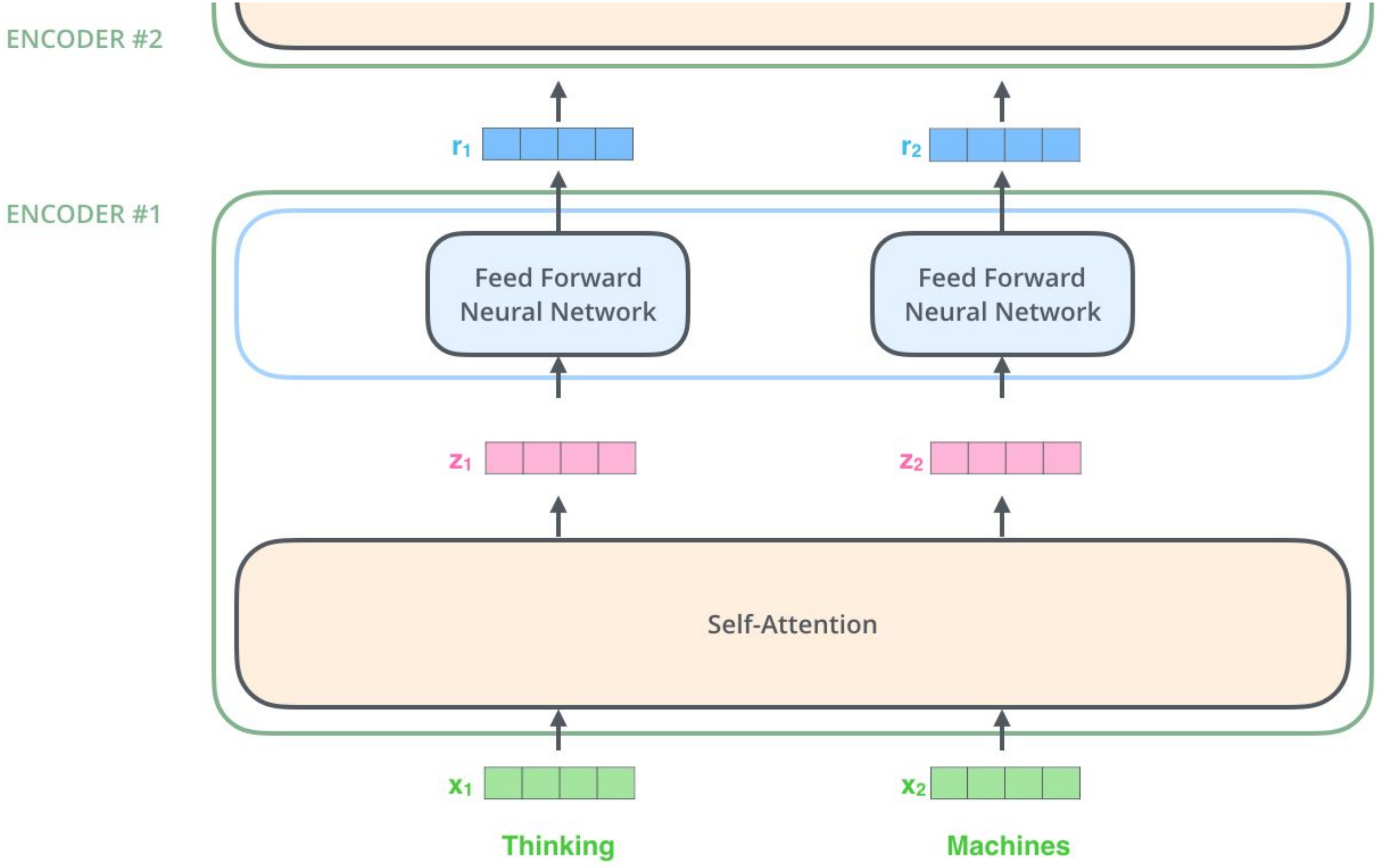
Will come back to the positional encoding later

Step 2: Token Embeddings → Attention Vectors



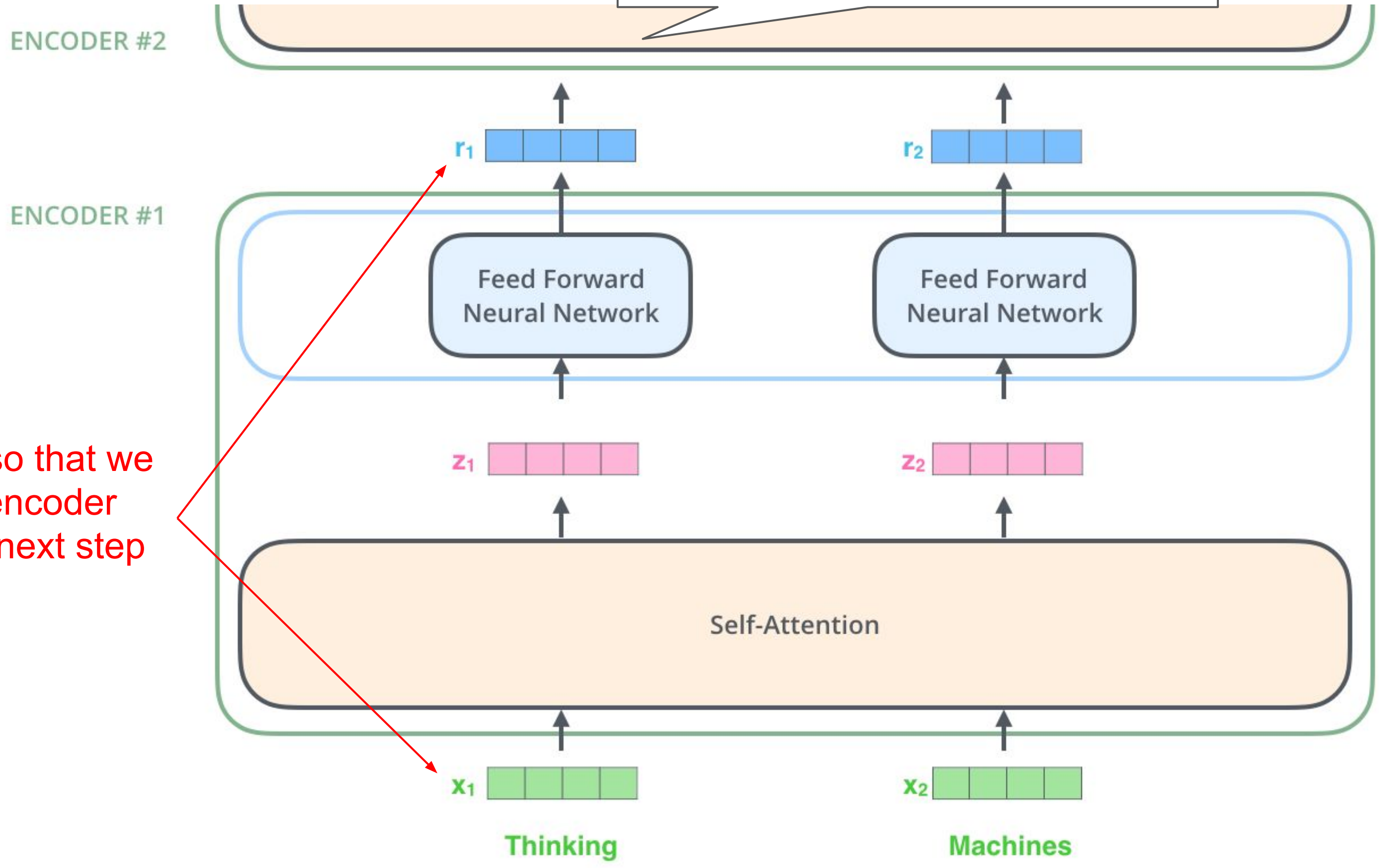
← Attention vectors

Step 3: Attention Vectors → Output Vectors



Repeat Steps 1-3 N Times

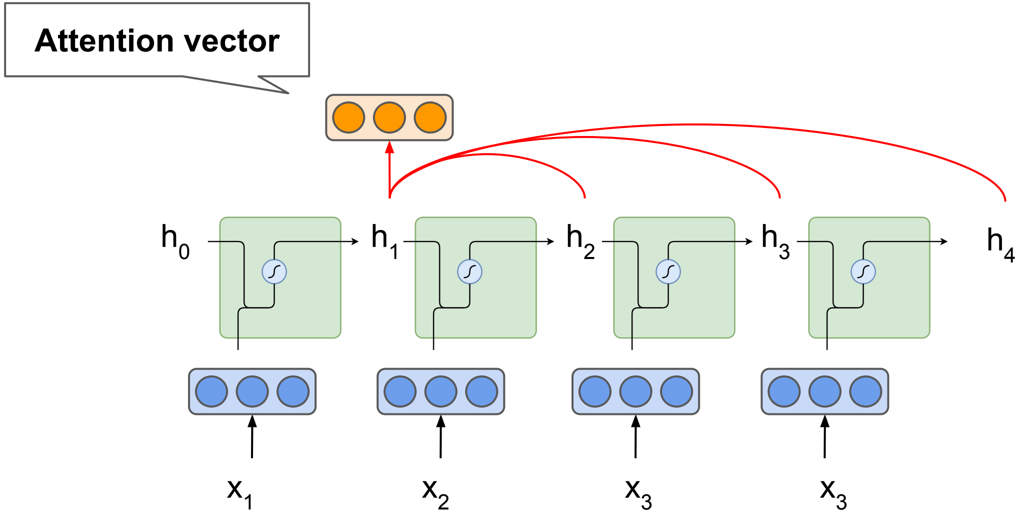
Encoder #2 takes r_* as if token embeddings (e.g., x_*)



Same dimensions so that we can use the same encoder architecture as the next step

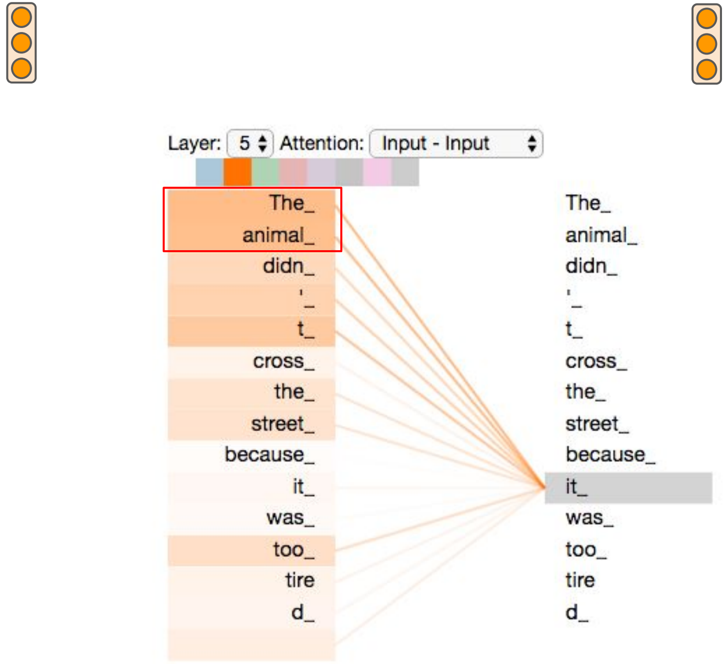
Self-Attention in Detail

Recap: Self-Attention for RNNs



Self-Attention: Key Idea

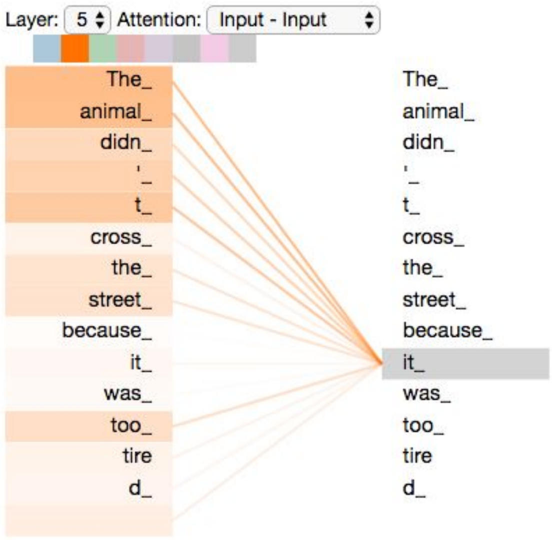
- Self-Attention looks at **all tokens** in the input sequence to calculate **the target token embedding** based on the context
- Example: “The animal didn’t cross the street because **it** was too tired.”



Self-Attention: Key, Query, Value Vectors

- The Self-Attention Layer encodes token embeddings by averaging **Value vectors** of input tokens <u>based on the similarities</u> between **Key vectors** of input tokens and **Query vector** of the target token

Key vectors

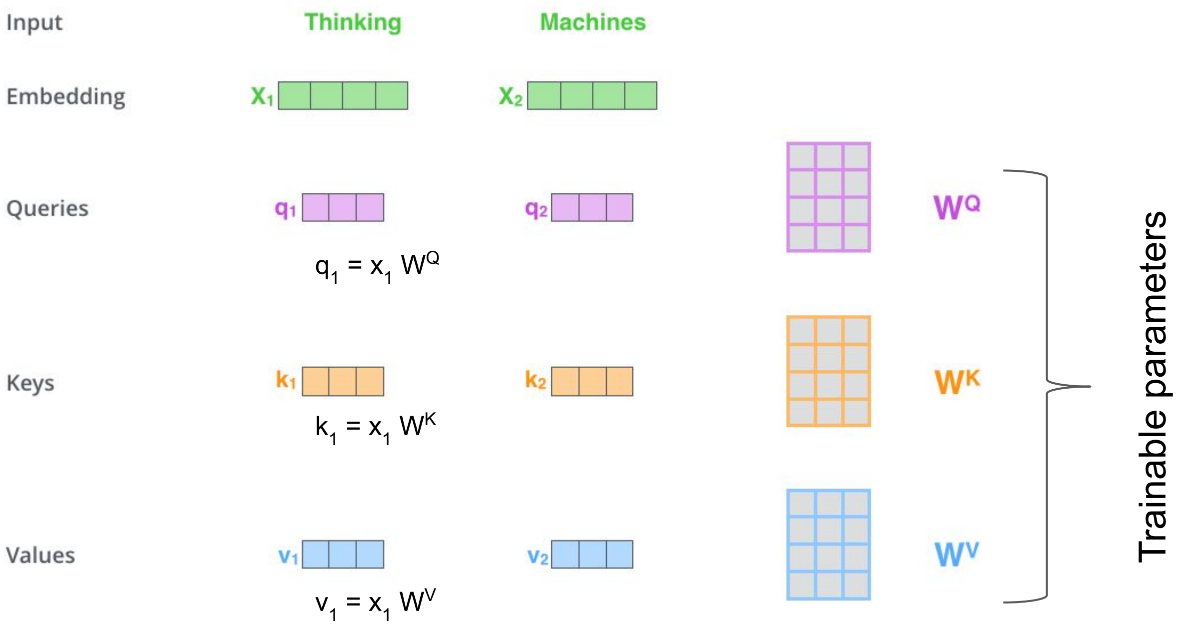


Query vector

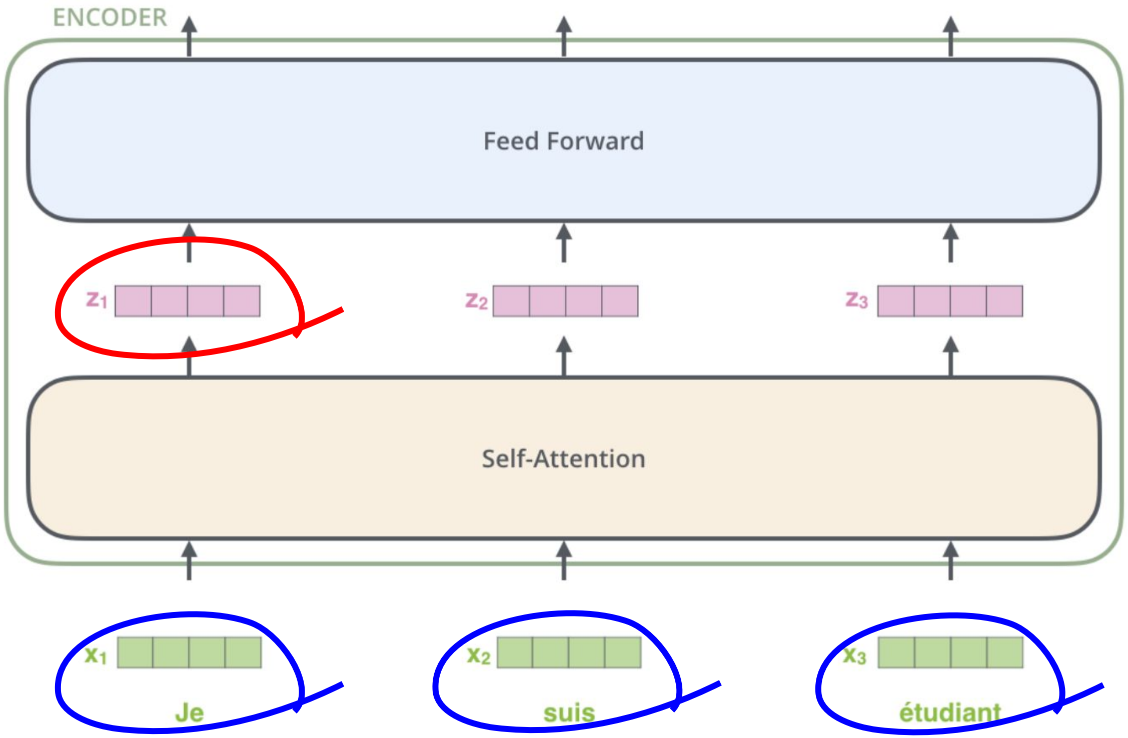
x **Value vectors** **= Attention vector**

Token Embeddings → Query, Key, Value Vectors

- Self-attention layer has **transformation matrices** for Query, Key, and Value vectors

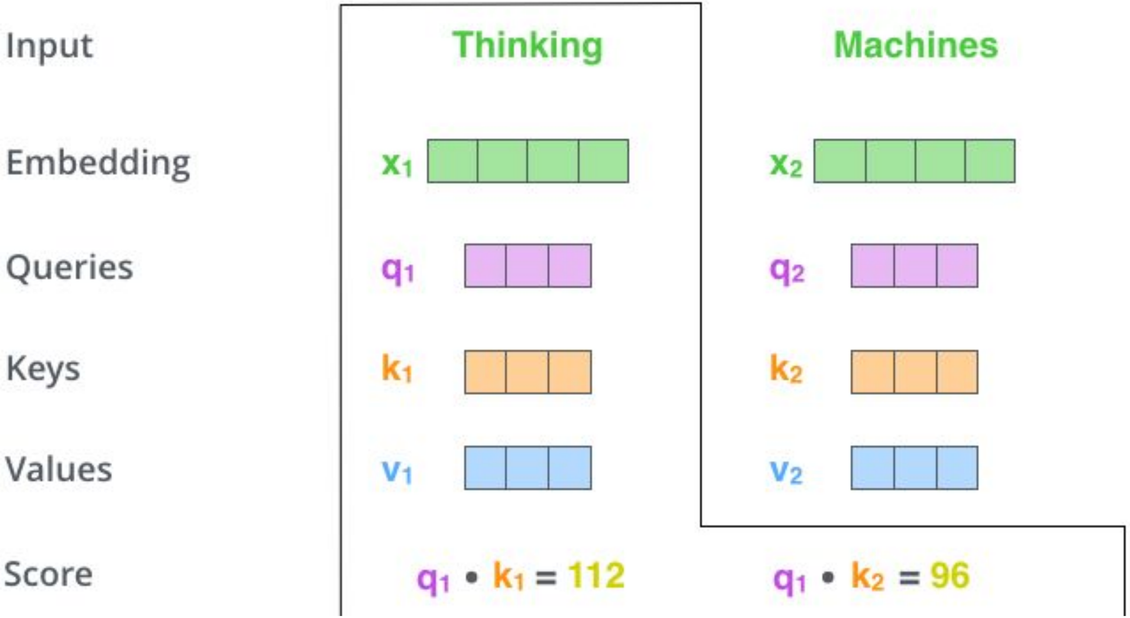


Calculating z_1 from Input Token Embeddings $x_{1\text{-}3}$



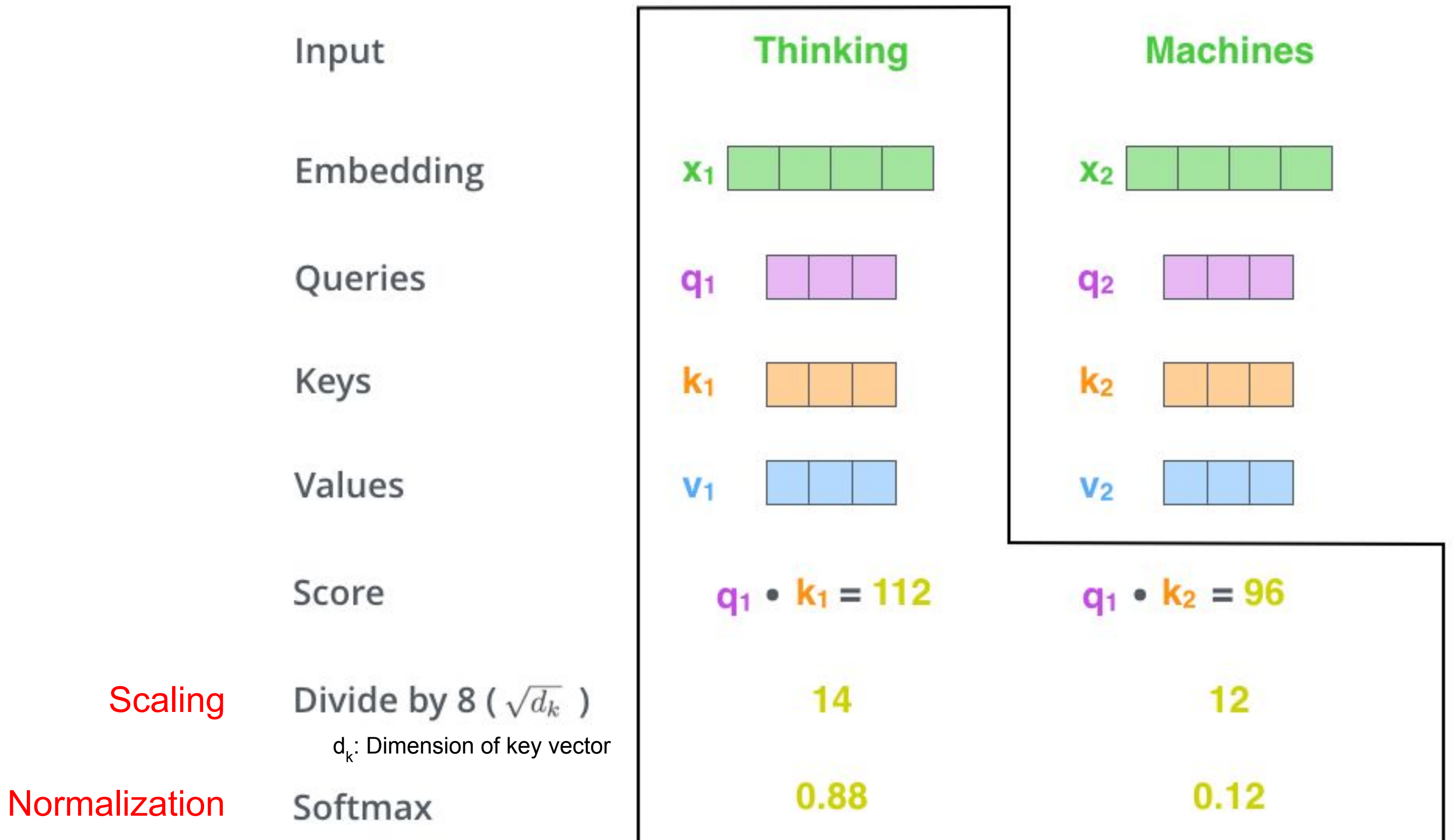
Example: Calculating Attention Vector for "Thinking"
Step 1: Query-Key Similarity Calculation

- The inner product of q_1 and k_*



Example: Calculating Attention Vector for “Thinking”

Step 2: Scaling and normalization



Side Topic: Why scaling?

- To make the softmax results reasonable
  - Why scaling? Basically, inner-product values are proportional to the dimension size
  - Why sqrt(d_k)? Scaling by d_K is too much

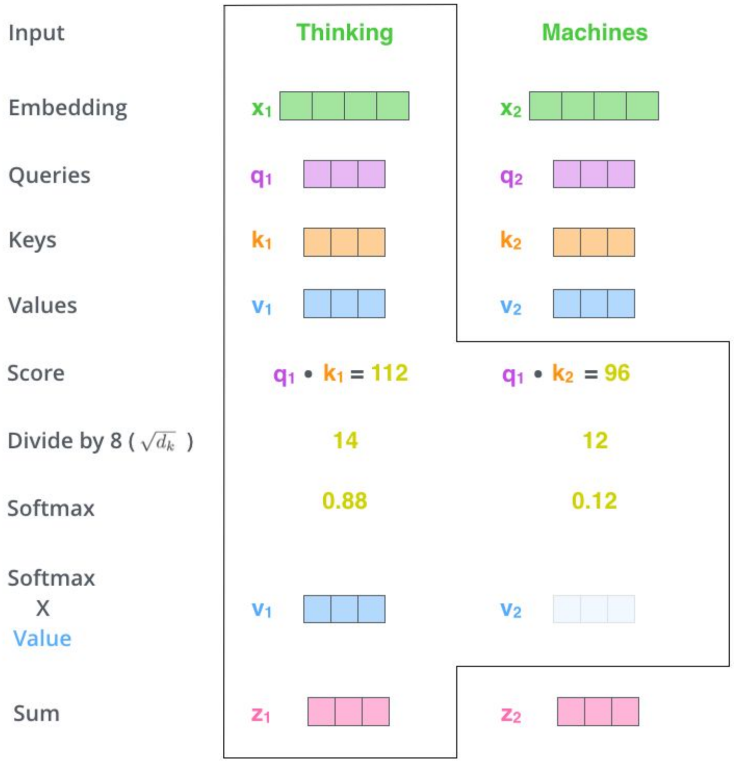
```
# Original
>>> torch.softmax(torch.Tensor([112, 96]), dim=0)
tensor([1.0000e+00, 1.1254e-07])  # Too skewed!

# Scaling by sqrt(dk)
>>> torch.softmax(torch.Tensor([14, 12]), dim=0)
tensor([0.8808, 0.1192])  # Looks good!

# Scaling by dk
>>> torch.softmax(torch.Tensor([112/64, 96/64]), dim=0)
tensor([0.5622, 0.4378])  # Too flat!
```

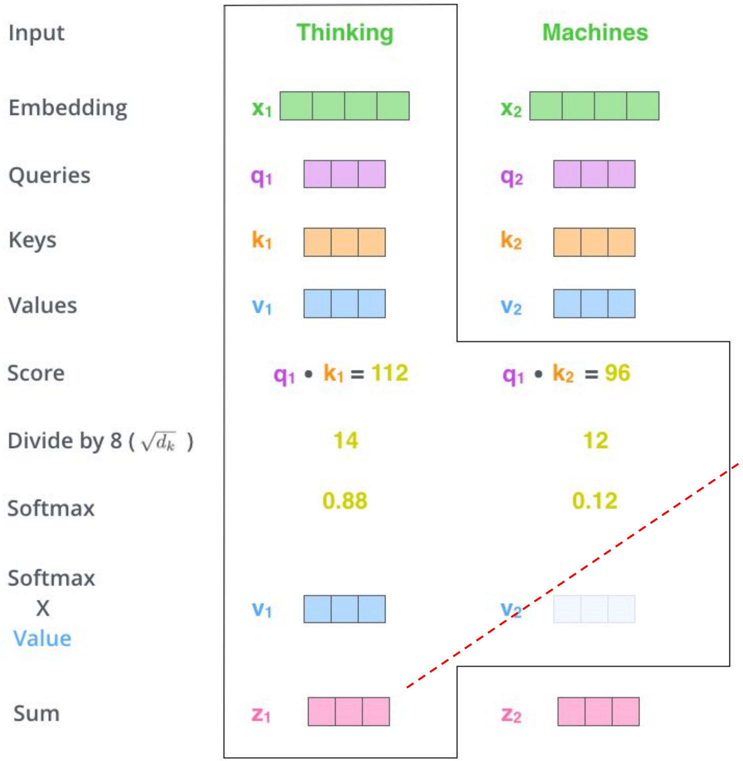
Example: Calculating Attention Vector for “Thinking”

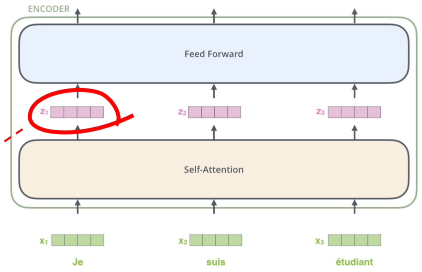
Step 3: Weighted Average of Value Vectors



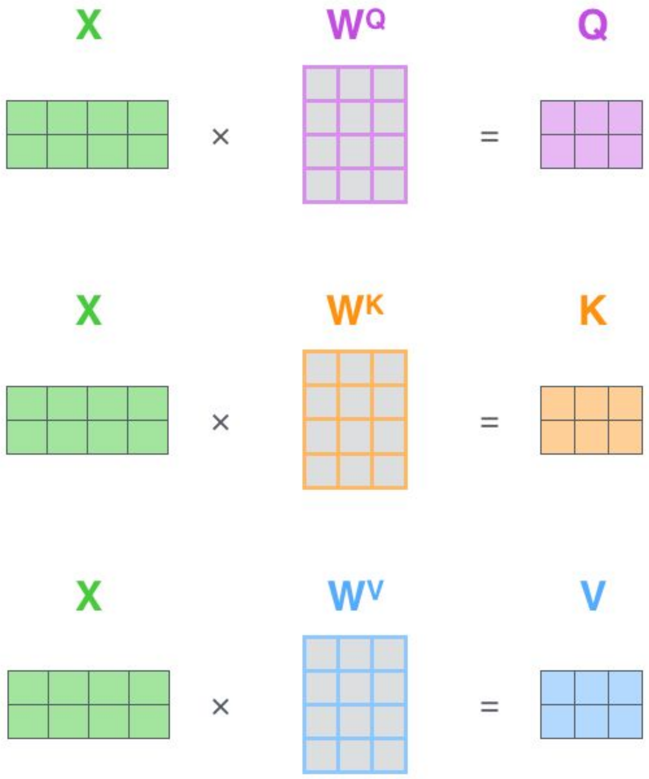
Example: Calculating Attention Vector for “Thinking”

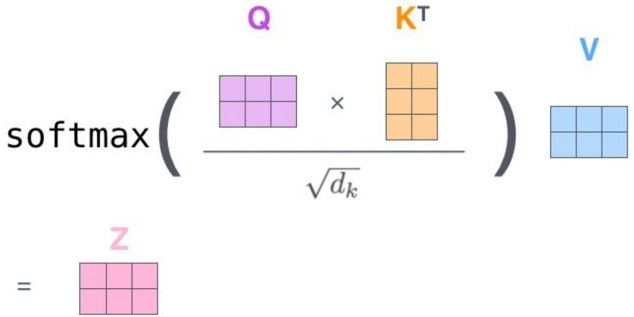
Step 3: Weighted Average of Value Vectors





Understanding Self-Attention in Matrix Calculation





Key Takeaway

- Model Architecture
    - The Transformer is an **Encoder-Decoder model**
        - Transformer Encoder + Transformer Decoder (not explained yet)
    - Transformer Encoder = Stack of **Transformer Encoder Blocks**
    - Transformer Encoder Block = **Self-Attention Layer** + Feed Forward NN
- **Self-Attention Layer**
    - Token Embeddings → Query, Key, Value Vectors → Attention Vectors
    - Multi-head Attention
- Additional techniques
    - Positional Encoding
    - Residual Connection + Layer Normalization

Questions?