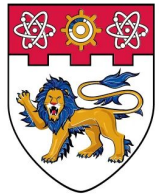


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Natural Language Processing

SC4002 / CE4045 / CZ4045

by Wang Wenya

Email: wangwy@ntu.edu.sg

Contents adapted from Dr. Joty Shafiq's notes

Click here for Lecture 4

Modules we will cover

ML & DL

Introduction to machine/deep learning

Word

Word vectors, language modeling

Transformer

Attention mechanism, encoder/decoder

Sequence

Sequence modeling, seq2seq learning

Pretraining

Masking, natural language generation

Prompting

Prompts, in-context learning

Outline for today

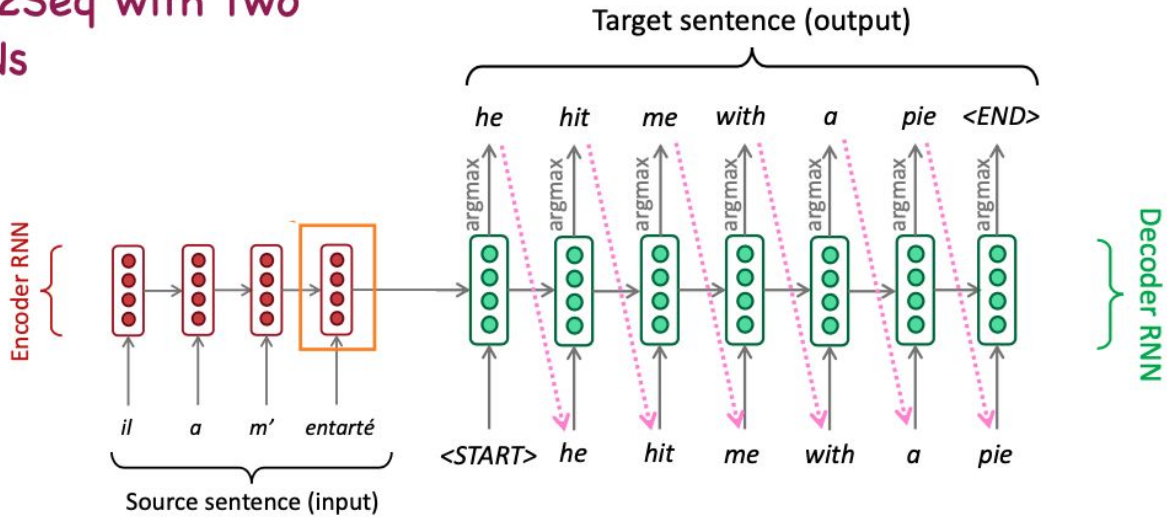
01 From RNNs to Attentions

02 Transformer Model

03 Applications of Transformers

Seq2Seq Modeling with RNNs (Revisit)

- Seq2Seq with Two RNNs



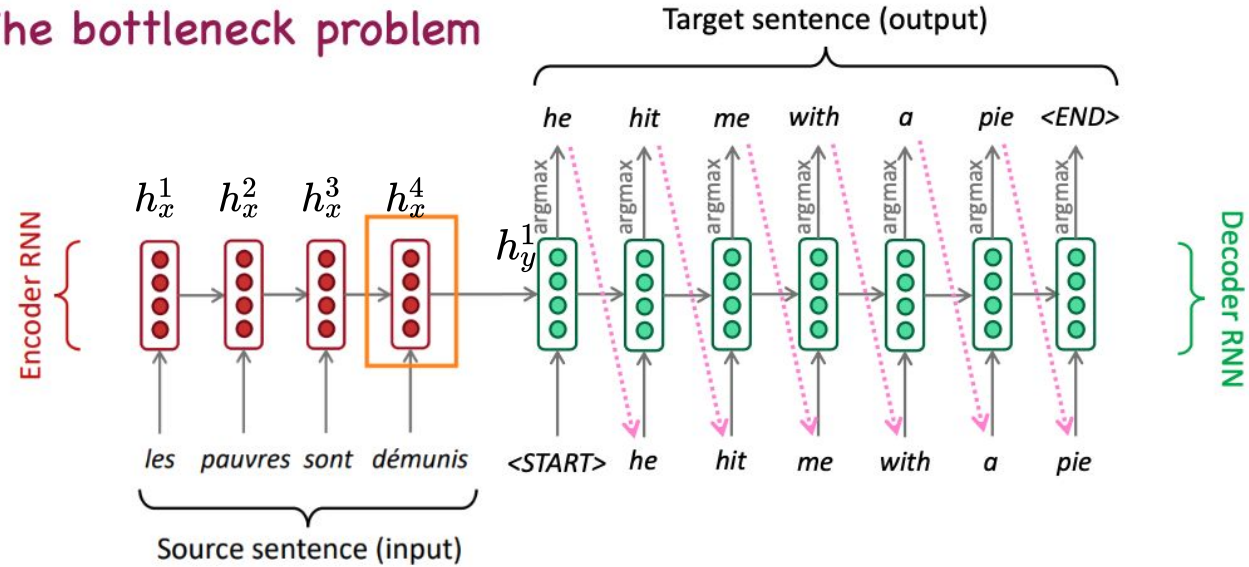
Encoder produces an encoding of the source sentence.

Provides Initial hidden state for Decoder

Decoder RNN is a **Conditional Language Model** that generates target sentence, conditioned on encoding.

Seq2Seq Modeling with RNNs (Revisit)

- The bottleneck problem



Encoding of the source sentence. This needs to capture all information about the source sentence. Information bottleneck!

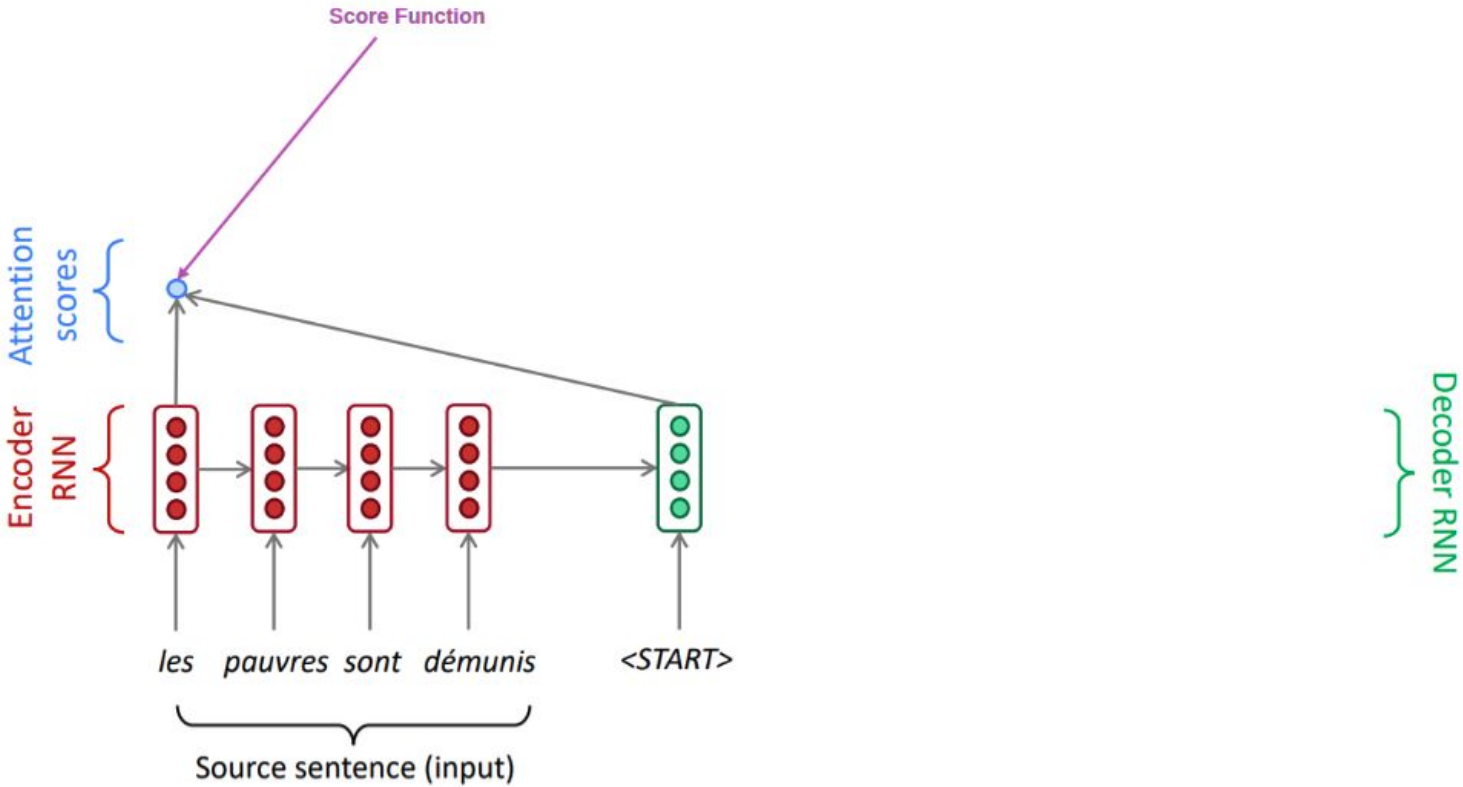
$$h_y^1 = f(W_{ye} \cdot e_{<START>} + W_{yh} \cdot h_x^4 + b_y)$$

$$h_x^1 = f(W_{xe} \cdot e_{les} + W_{xh} \cdot h_x^0 + b_x)$$

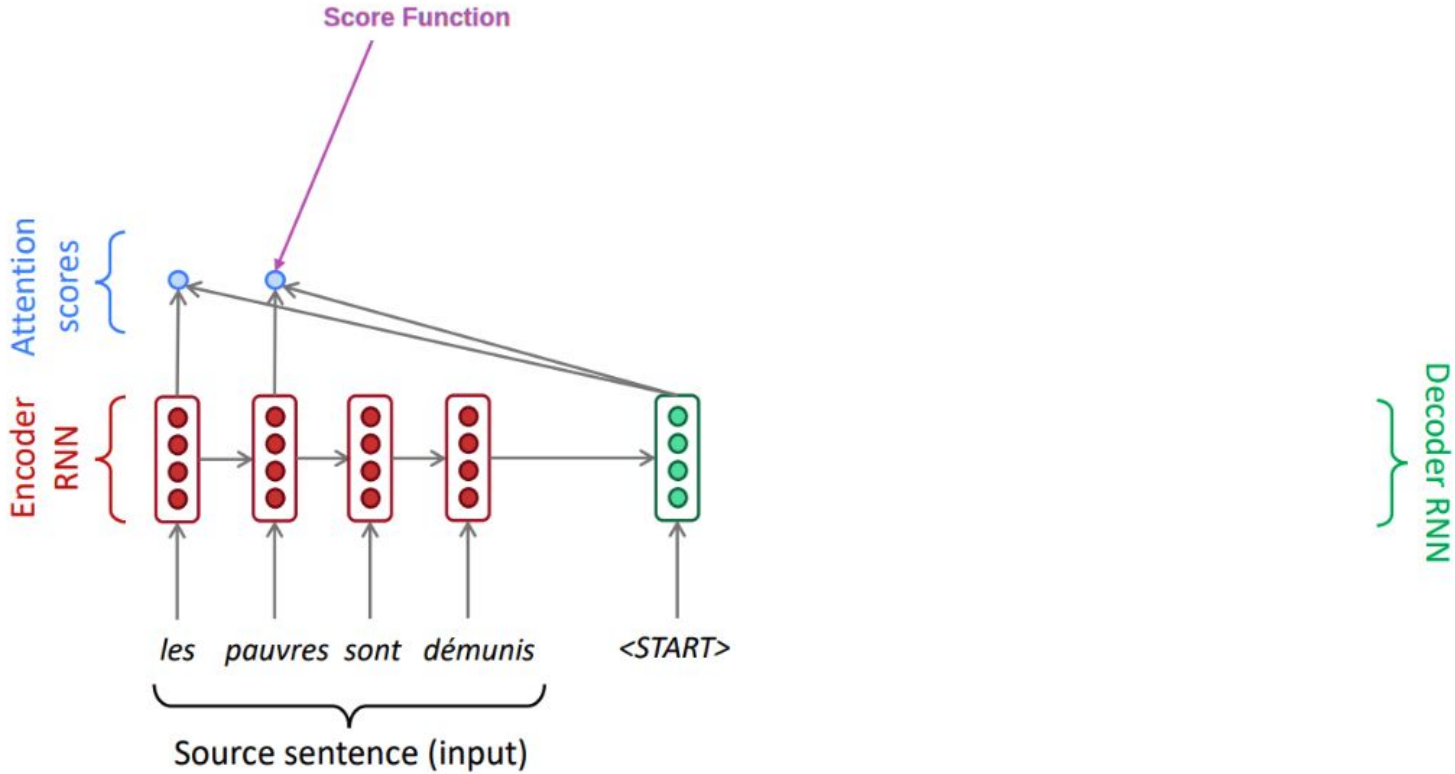
Solution: Attentions

- **Attention** provides a solution to the bottleneck problem.
- **Core idea:** on each step of the decoder, use direct connection to the encoder to focus on the relevant part of the source sequence

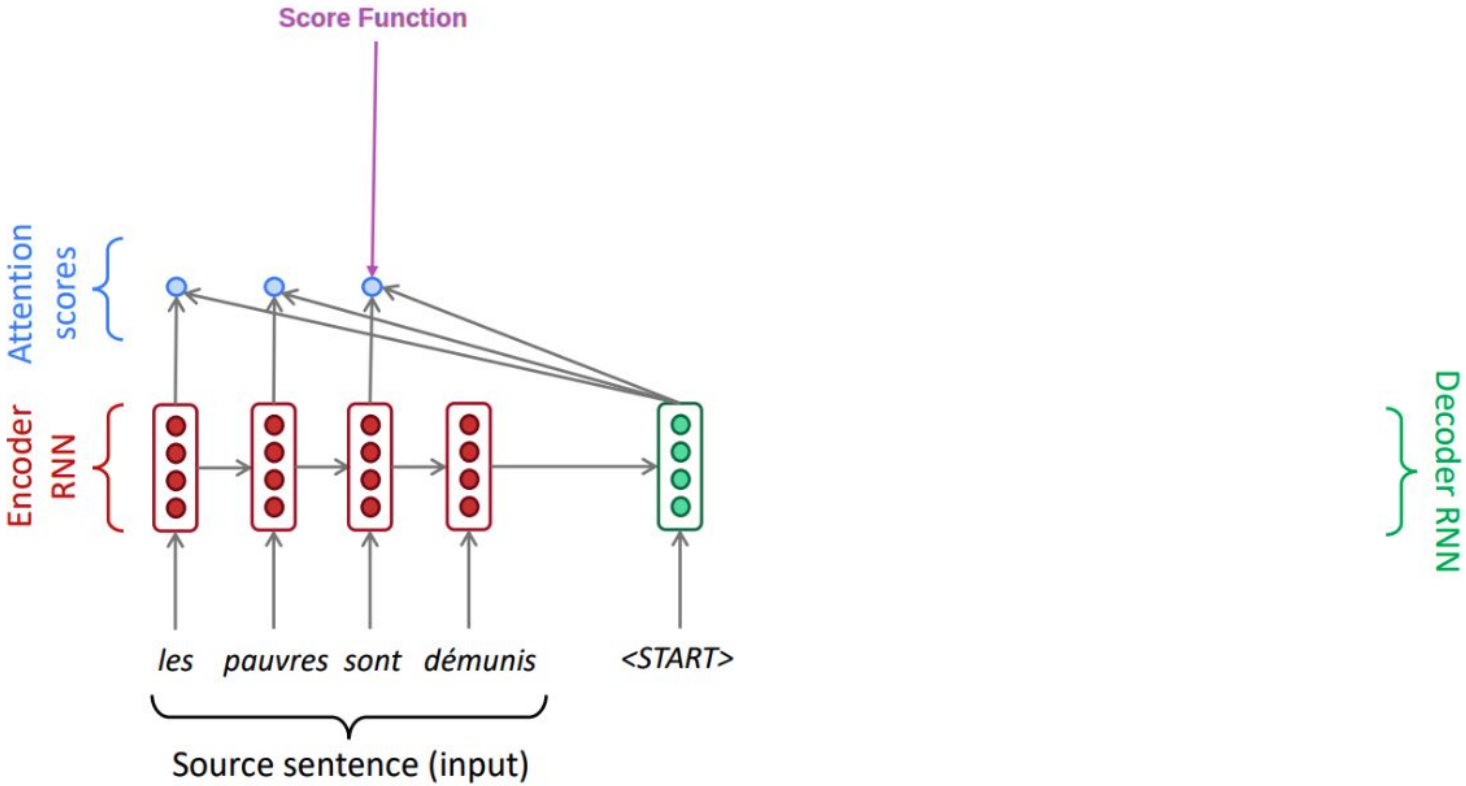
Solution: Attentions



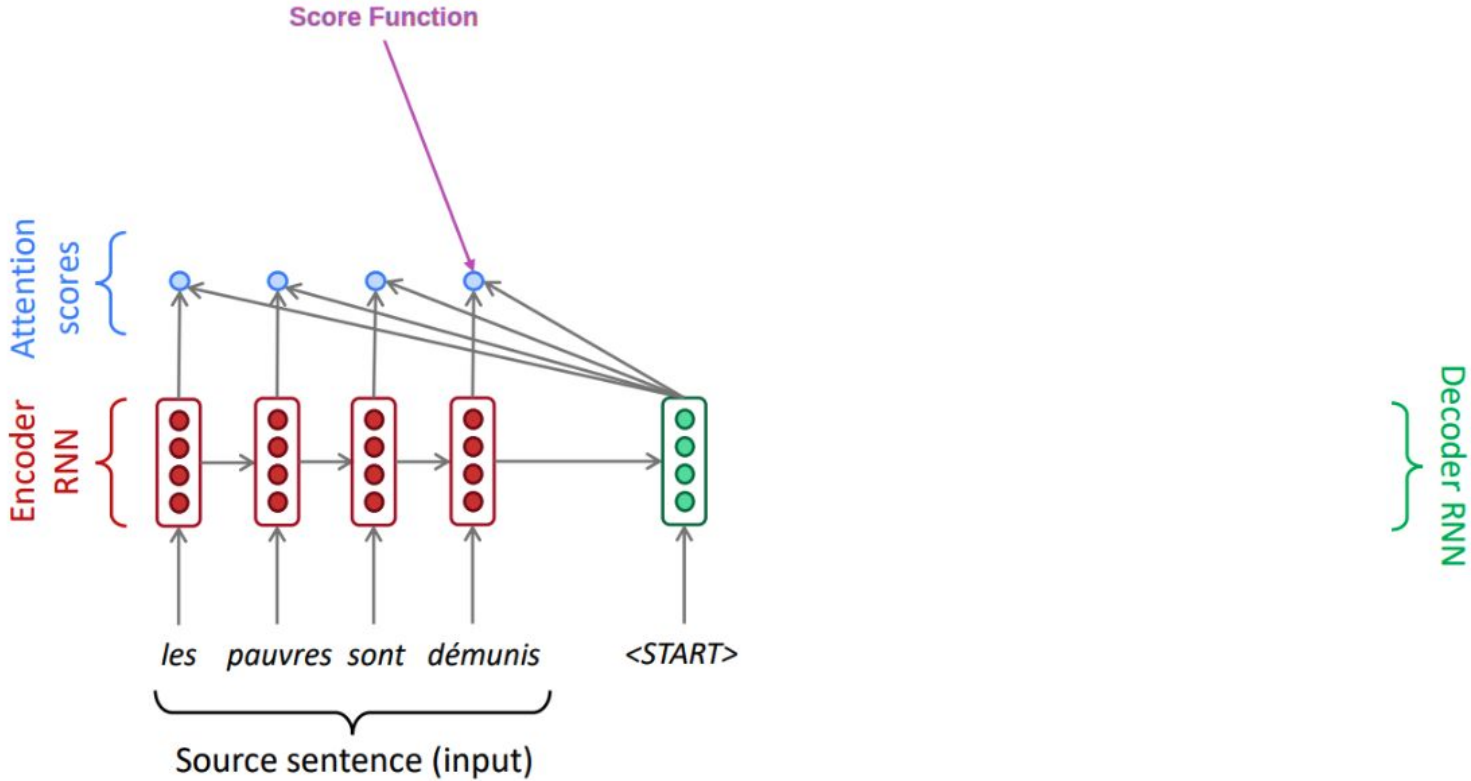
Solution: Attentions



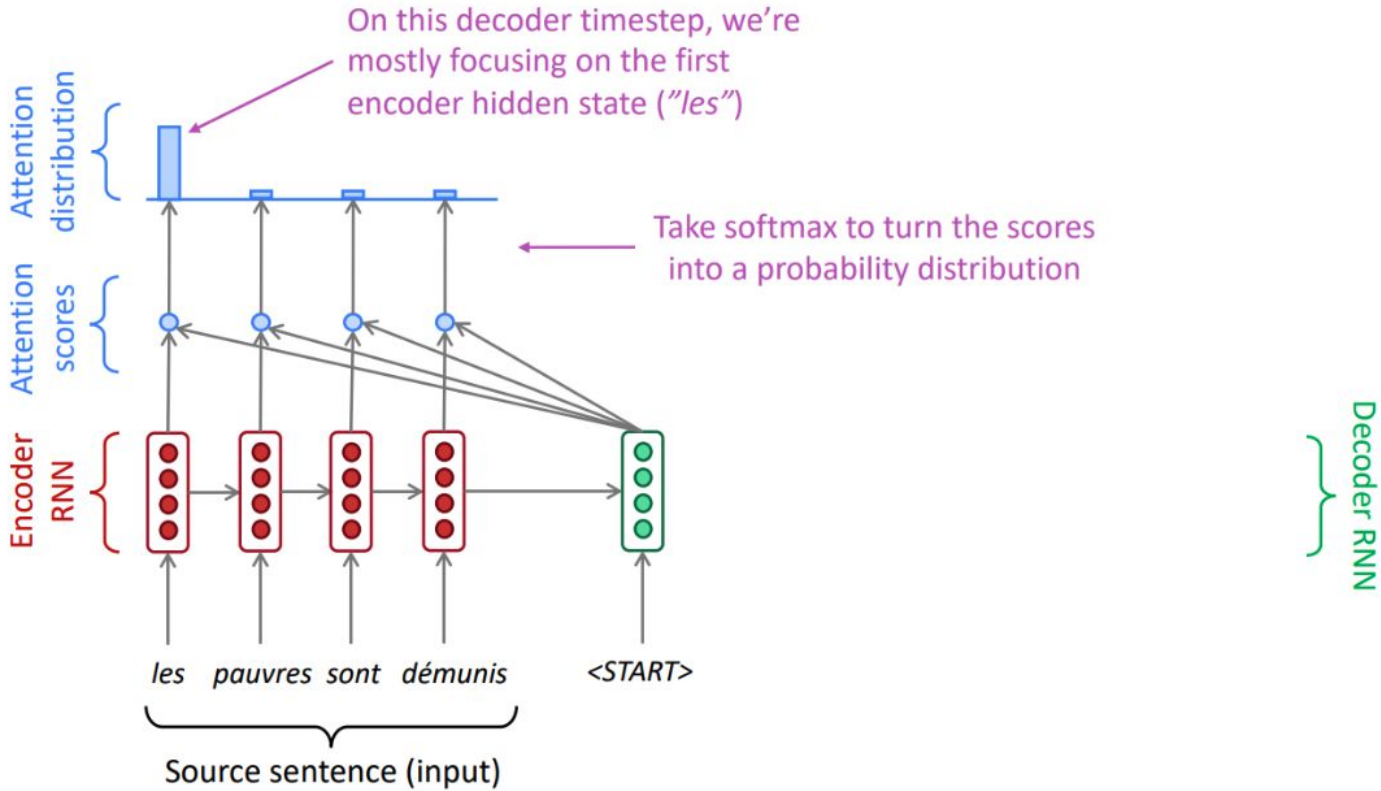
Solution: Attentions



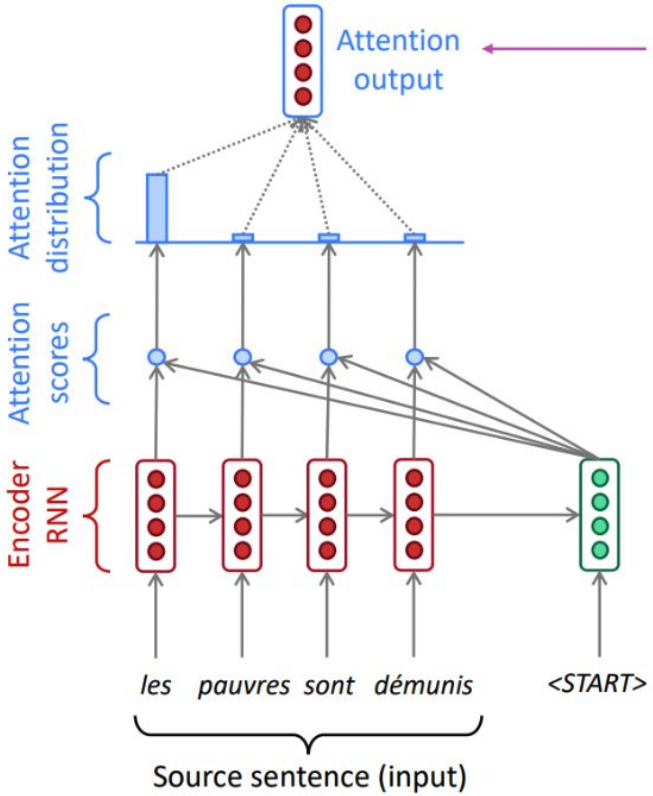
Solution: Attentions



Solution: Attentions



Solution: Attentions

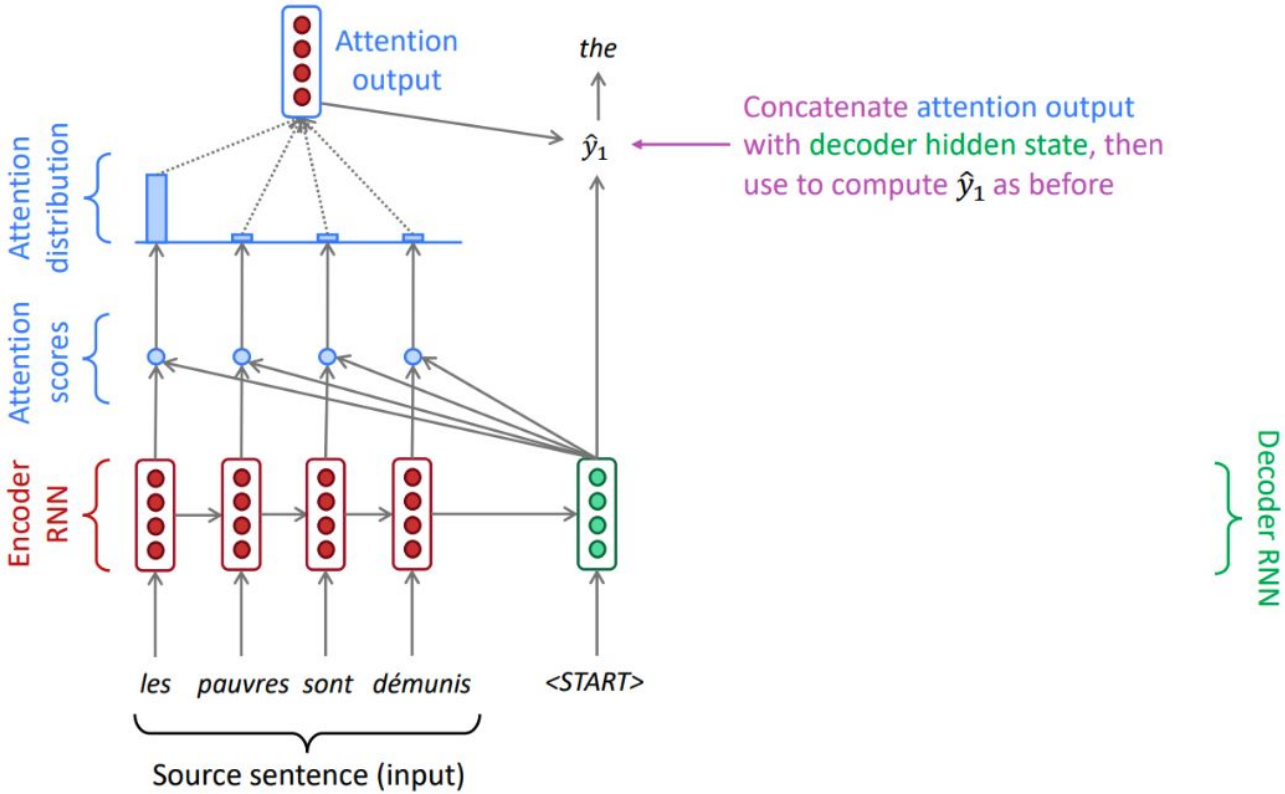


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

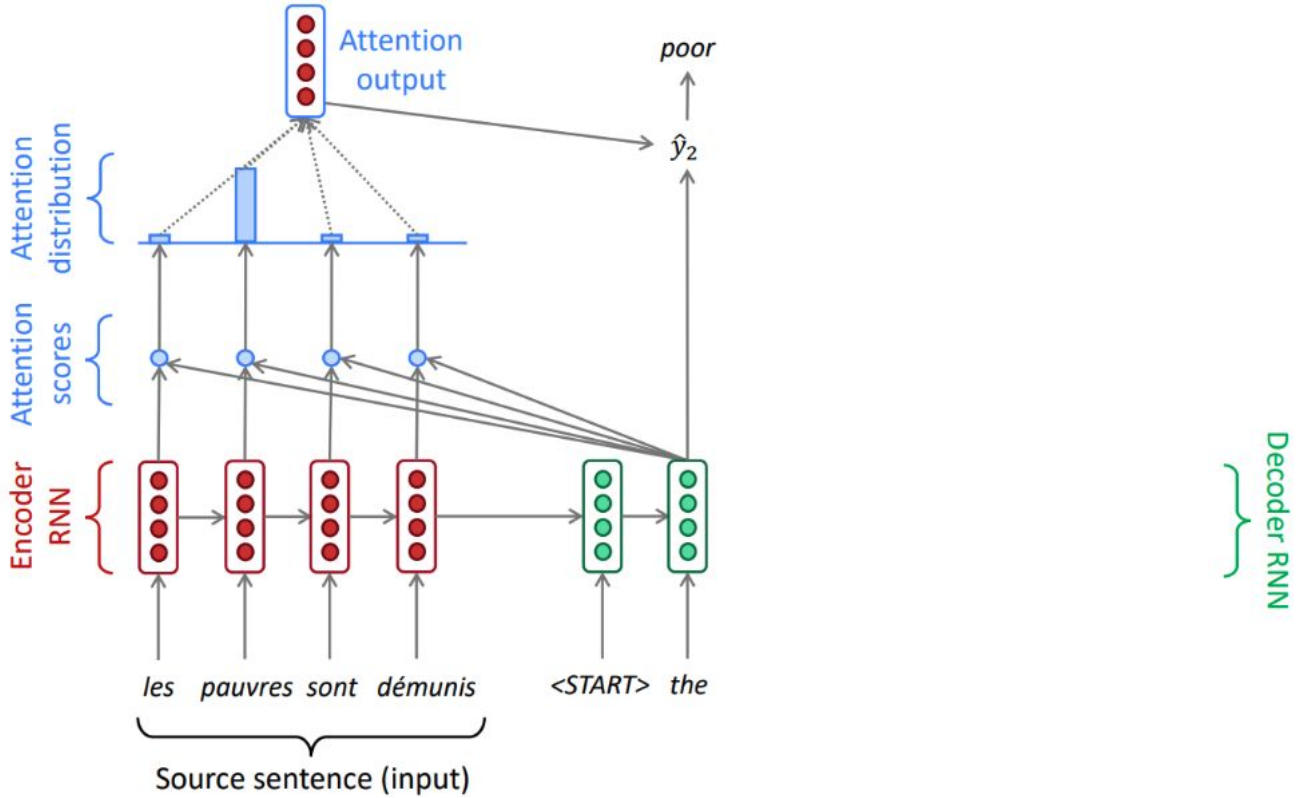
The attention output mostly contains information the hidden states that received high attention.

Decoder RNN

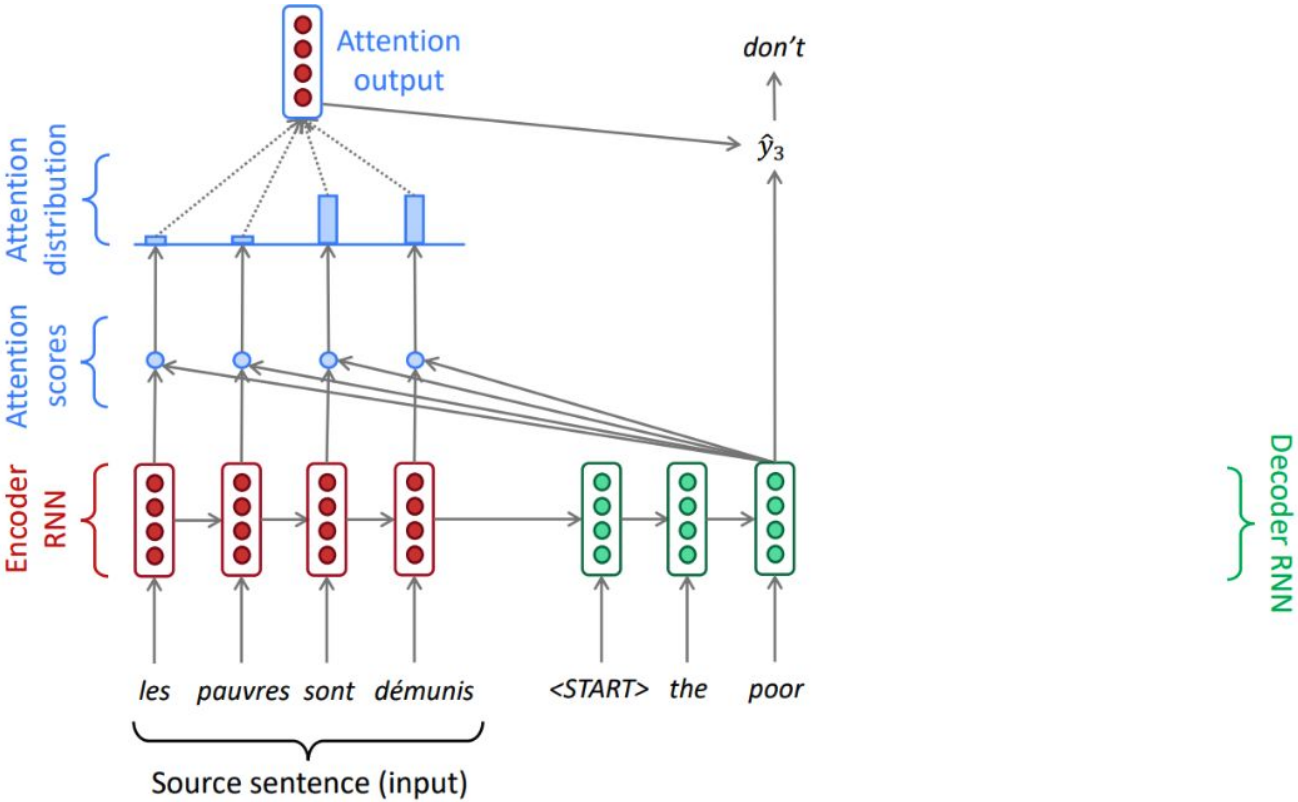
Solution: Attentions



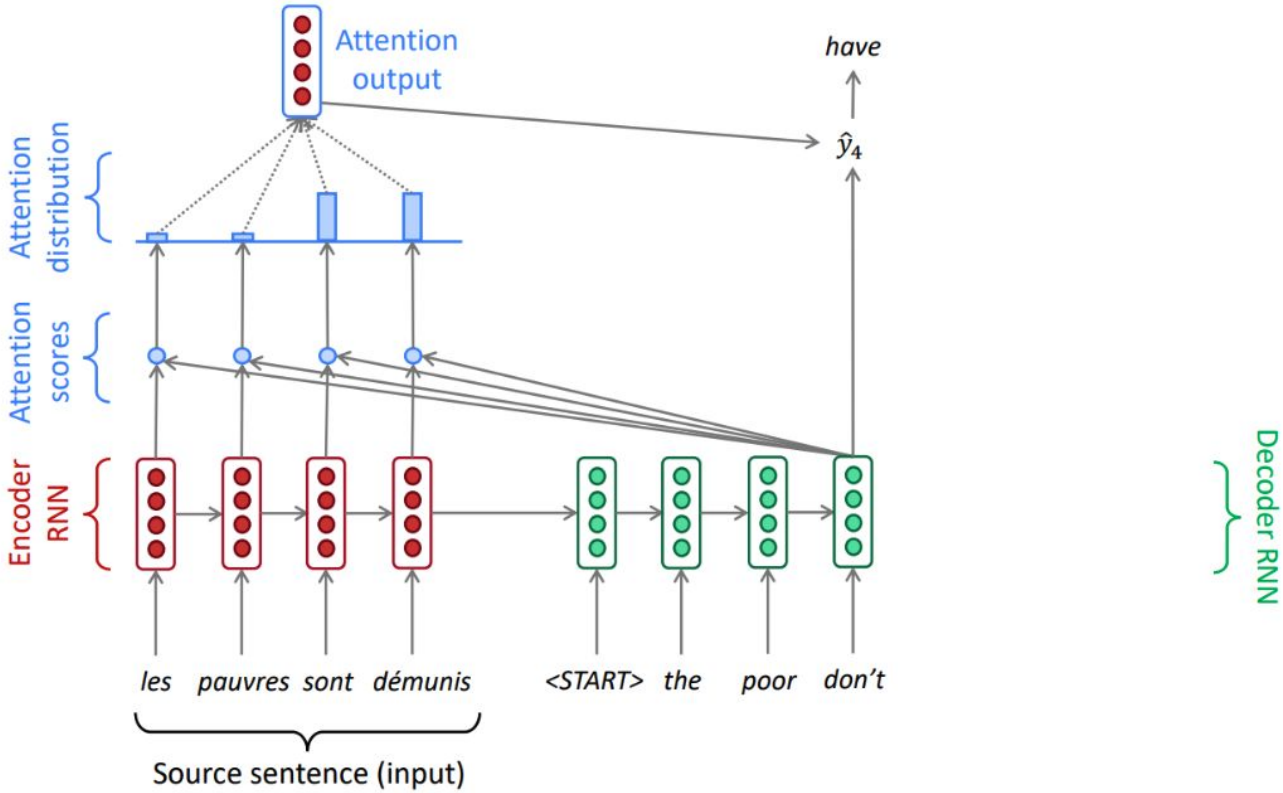
Solution: Attentions



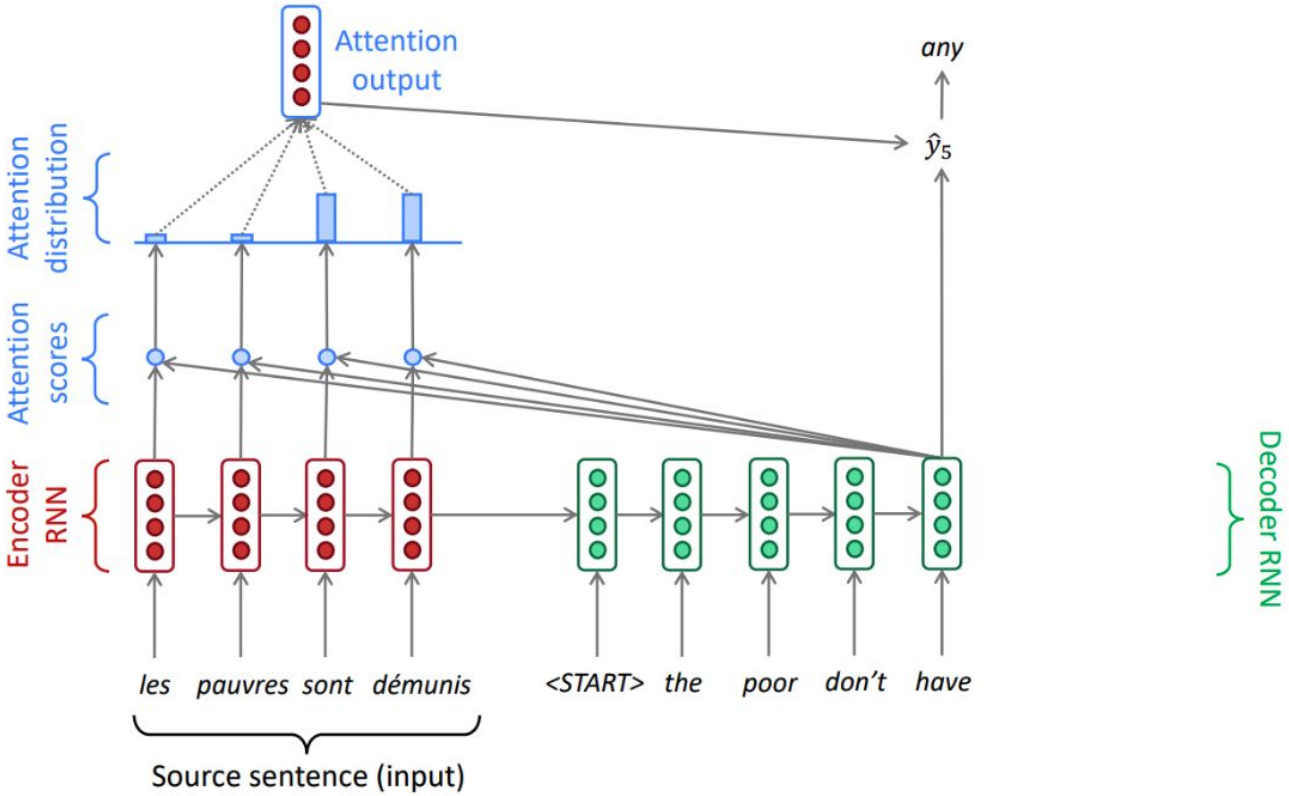
Solution: Attentions



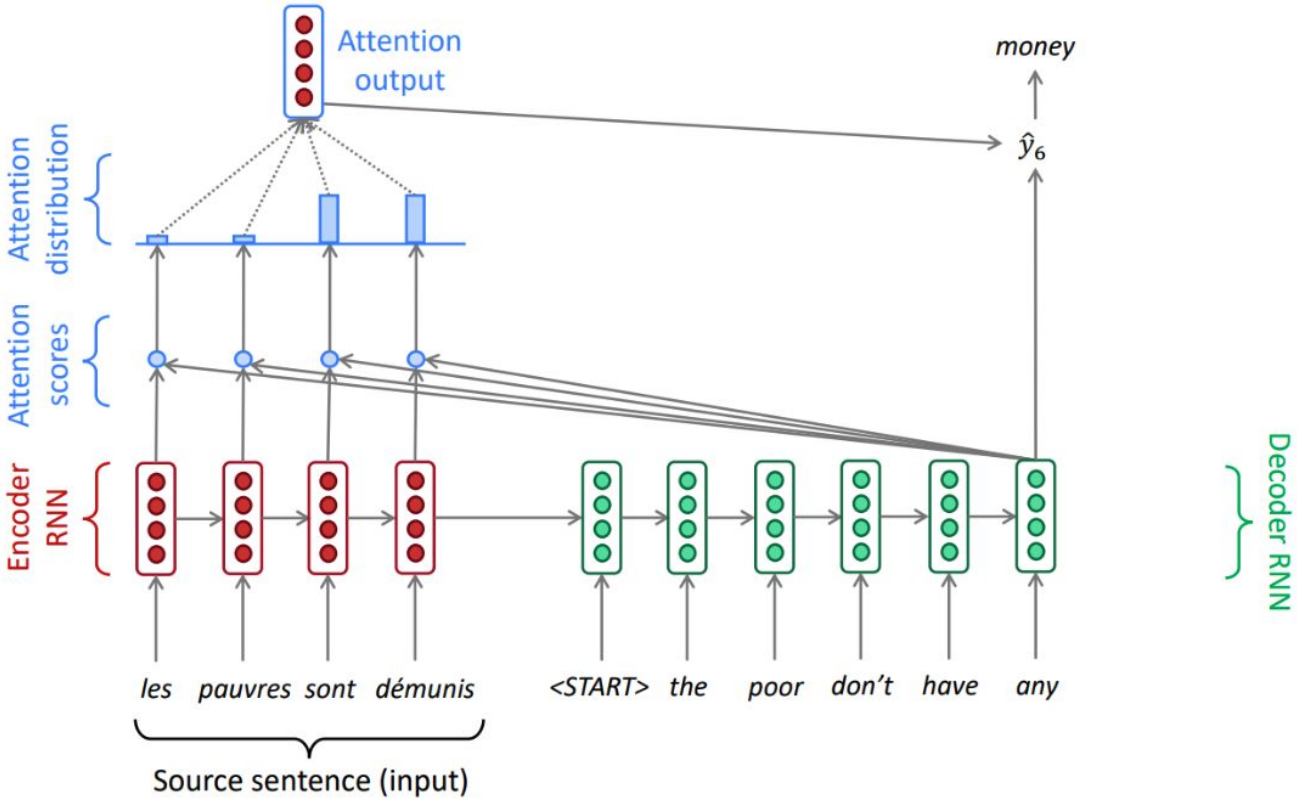
Solution: Attentions



Solution: Attentions



Solution: Attentions



Attentions (Formally)

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

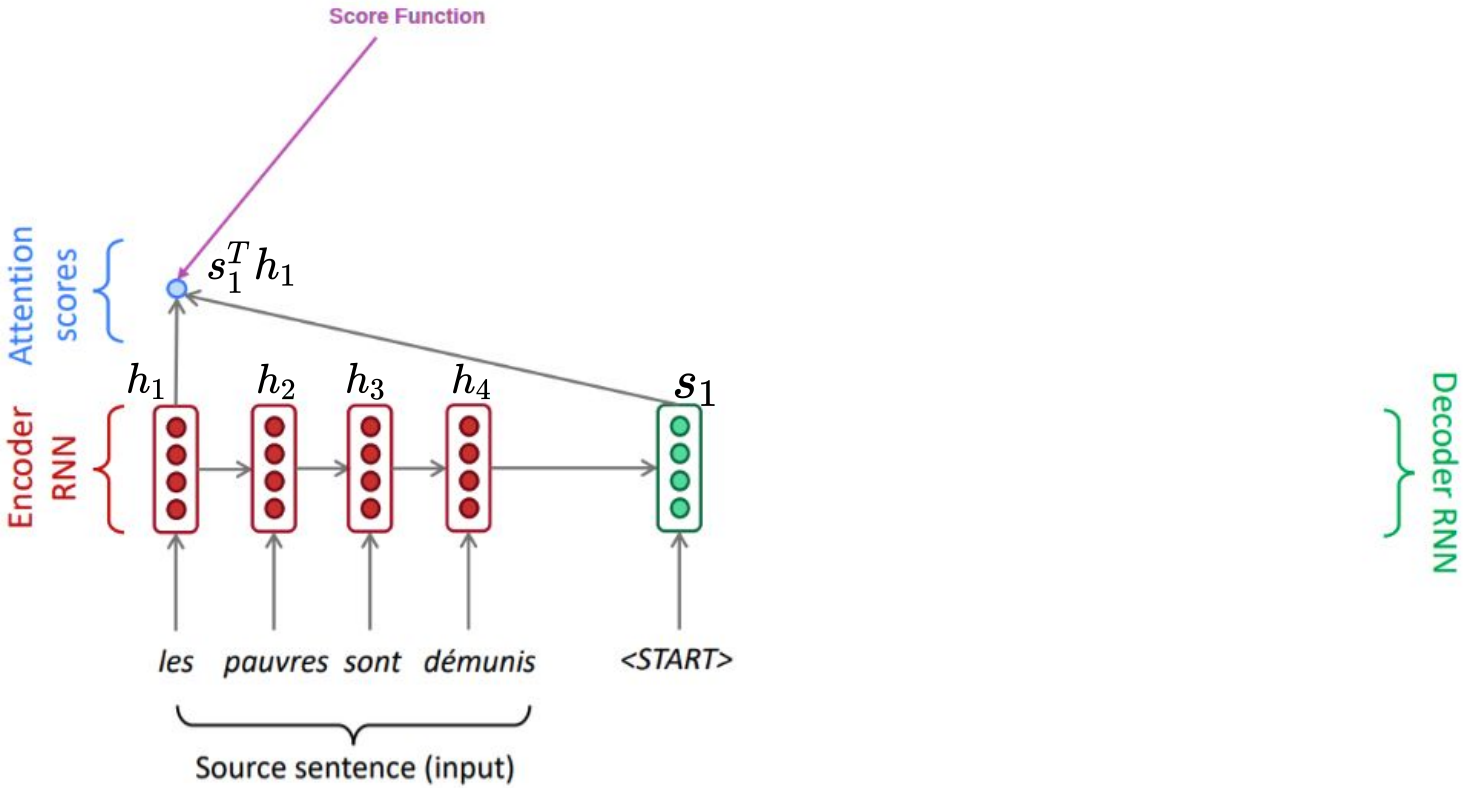
- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

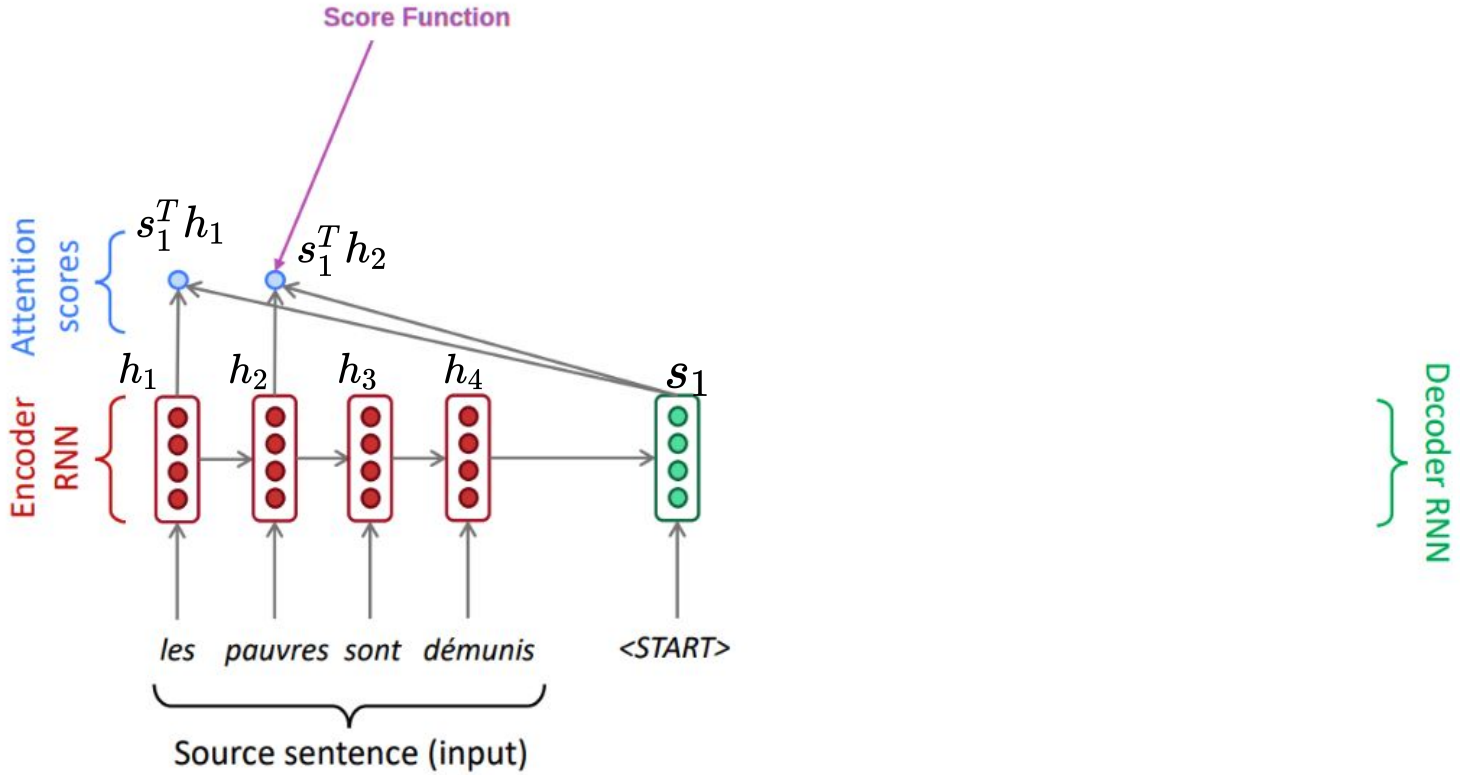
- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

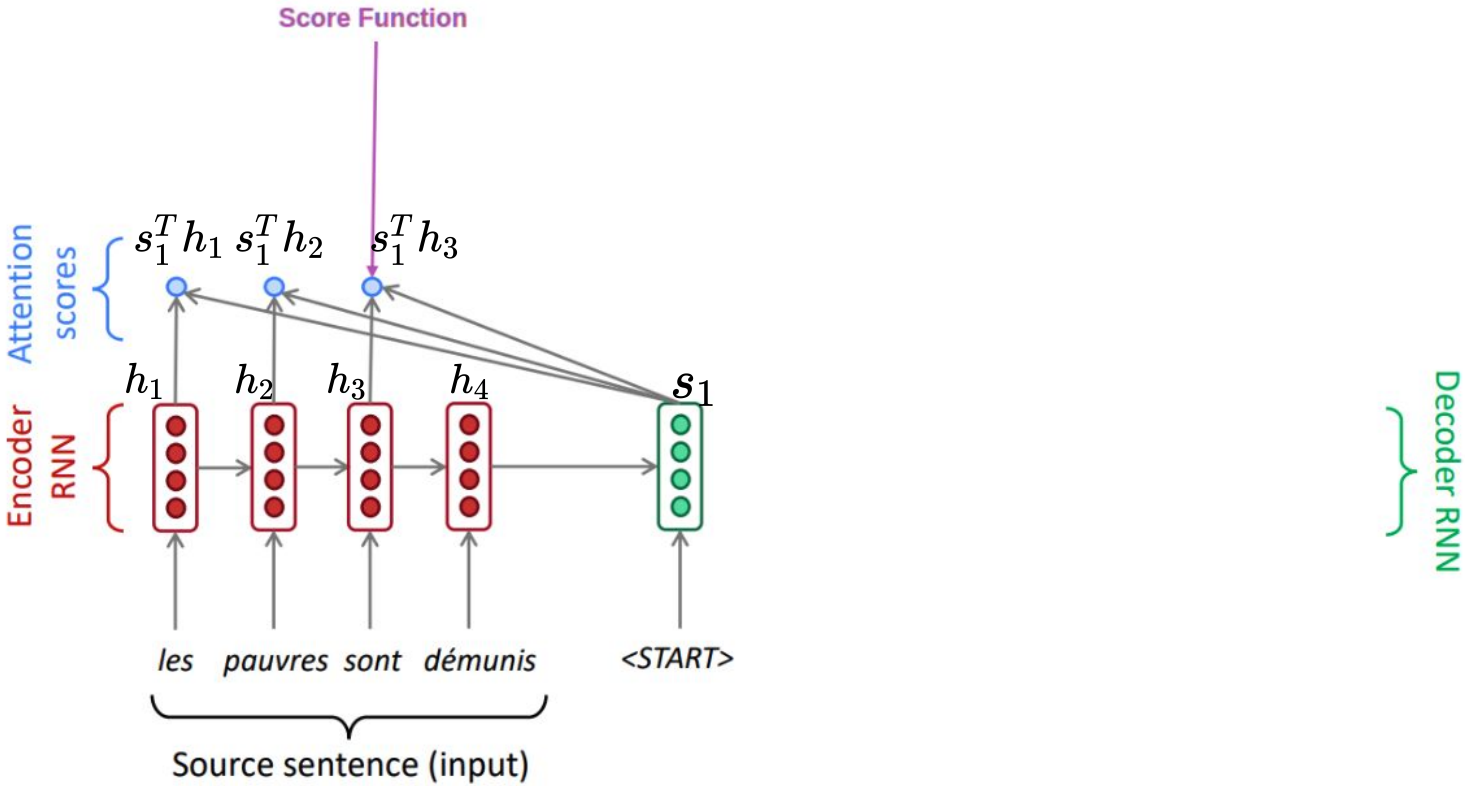
Solution: Attentions



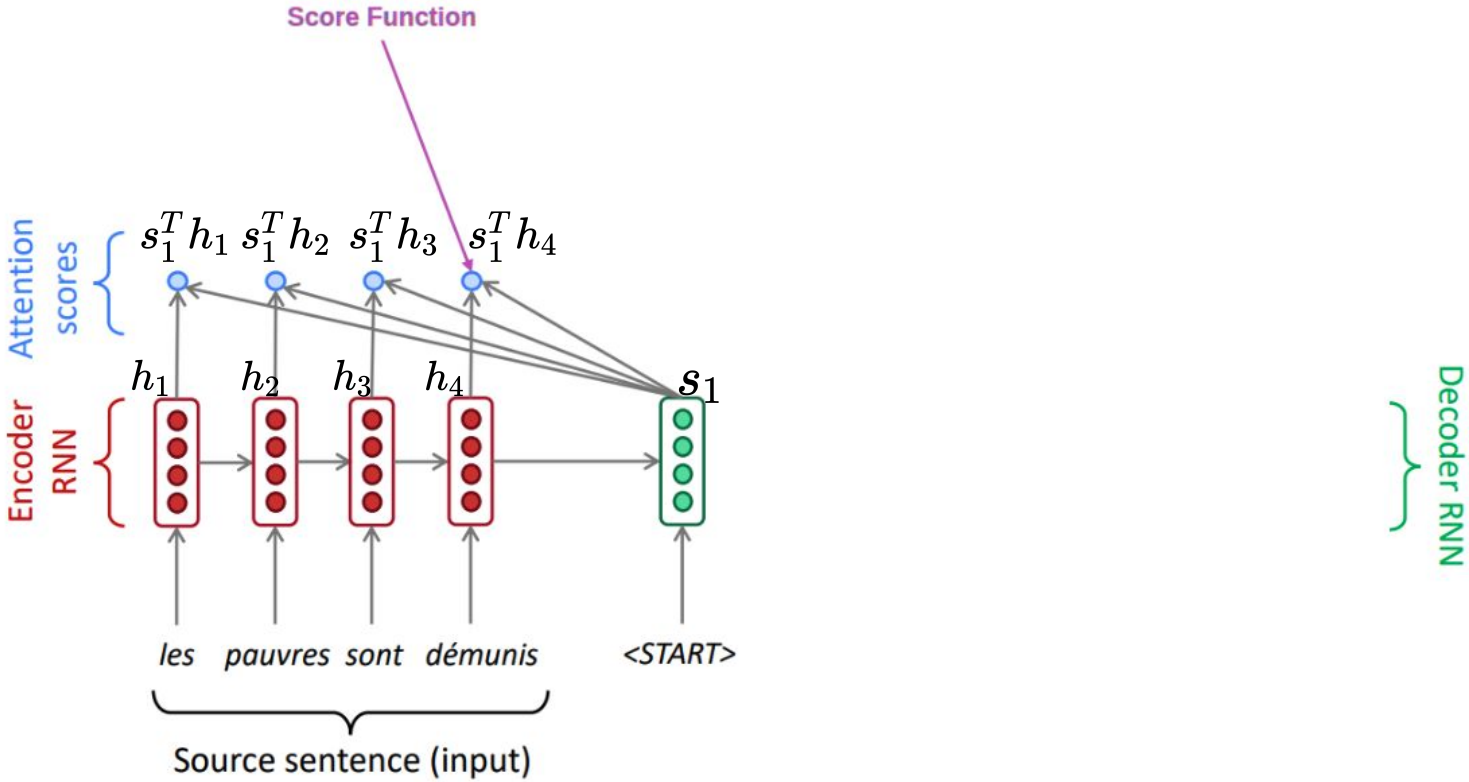
Solution: Attentions



Solution: Attentions

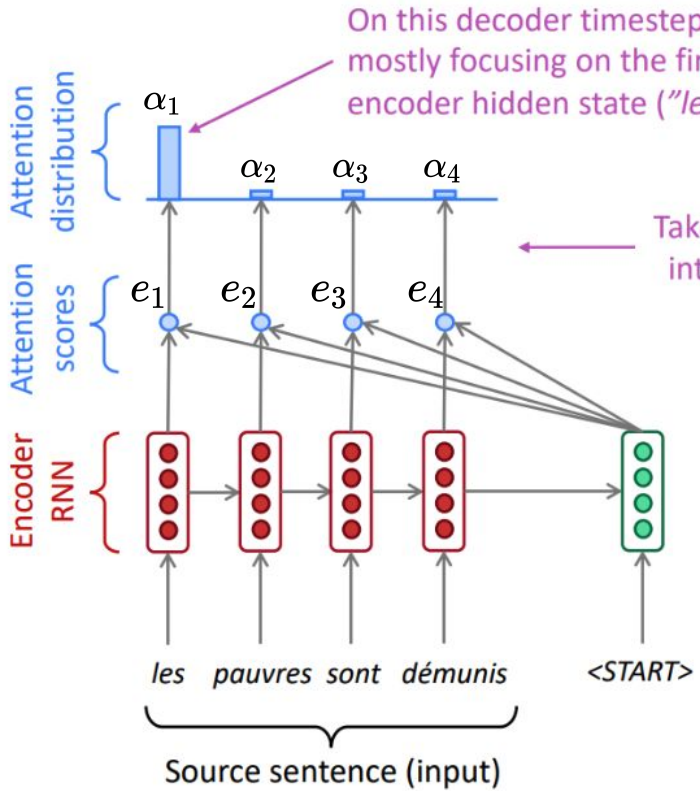


Solution: Attentions



Solution: Attentions

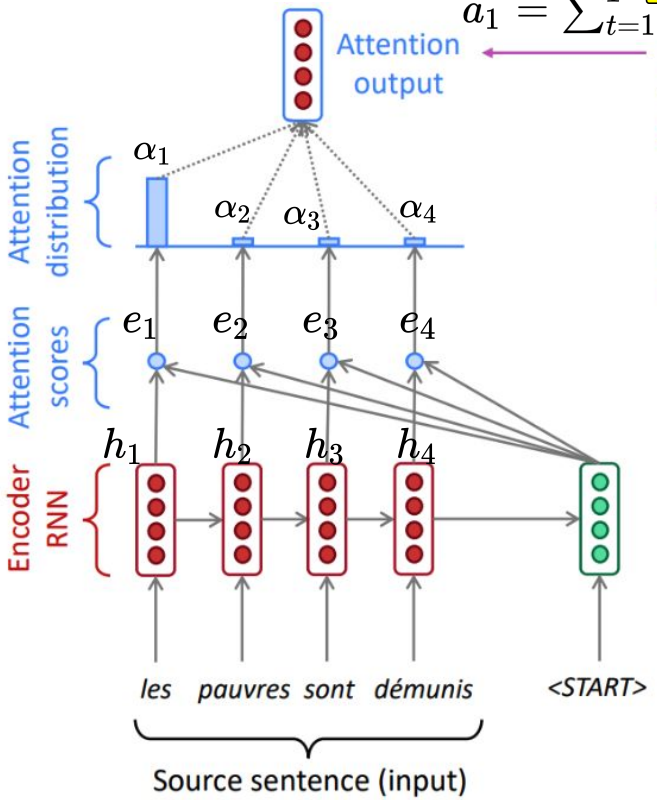
$$e_t = s_1^T h_t$$



$$\alpha_t = softmax(e_t) = \frac{\exp(e_t)}{\sum_{i=1}^{T} \exp(e_i)}$$

Solution: Attentions

$$a_1 = \sum_{t=1}^{T} \alpha_t \cdot h_t$$

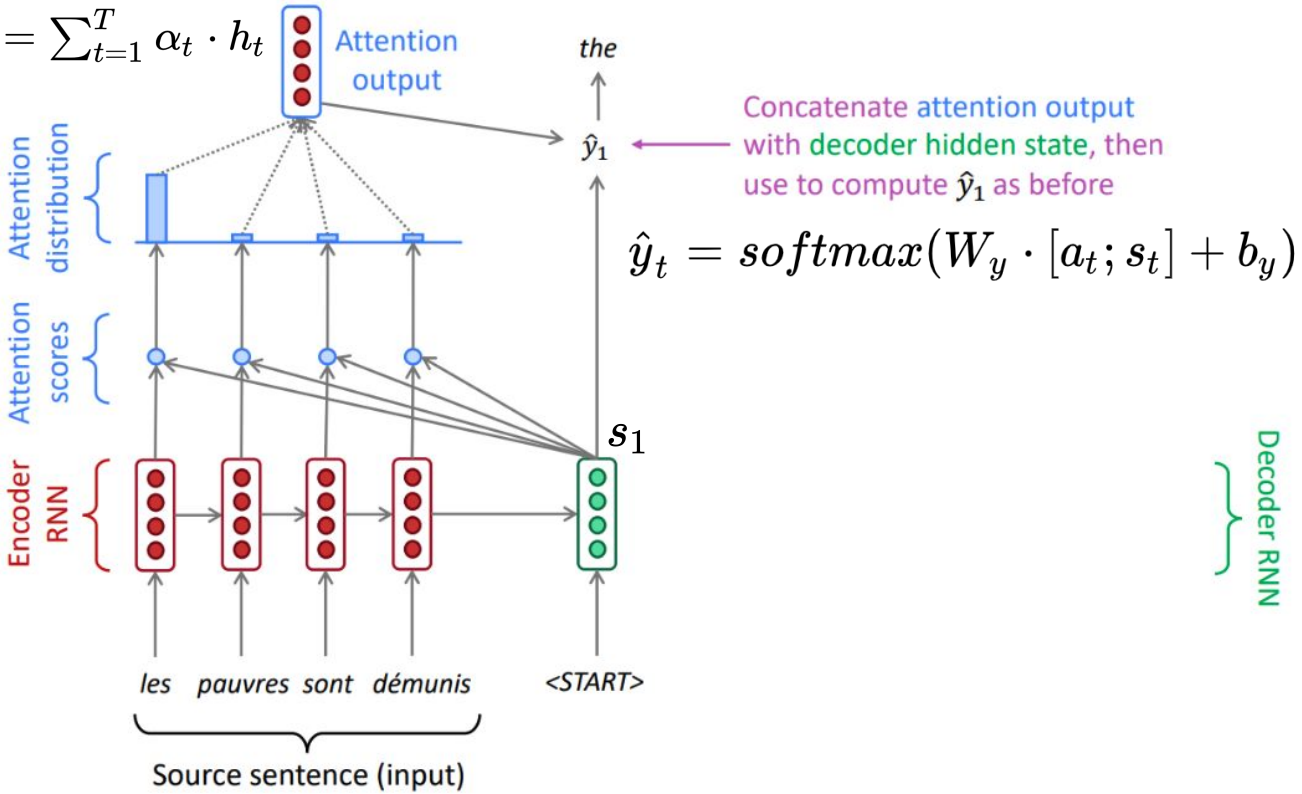


Use the attention distribution to take a **weighted sum** of the encoder hidden states.

The attention output mostly contains information the hidden states that received high attention.

Solution: Attentions

$$a_1 = \sum_{t=1}^{T} \alpha_t \cdot h_t$$



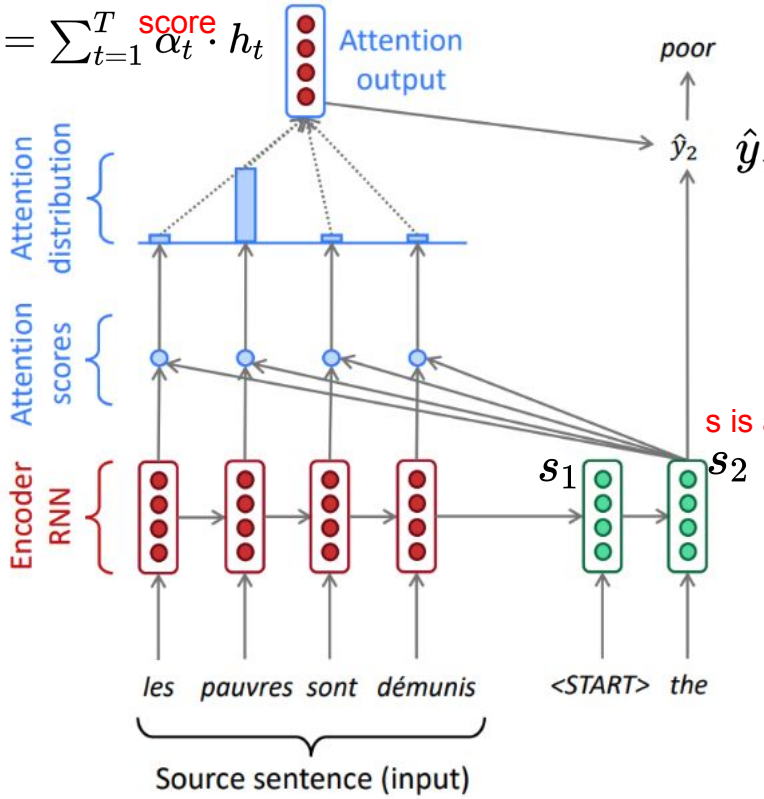
$$\hat{y}_t = softmax(W_y \cdot [a_t; s_t] + b_y)$$

Solution: Attentions

attention score

$$a_1 = \sum_{t=1}^{T} \alpha_t \cdot h_t$$

attention output



$$\hat{y}_t = softmax(W_y \cdot [a_t; s_t] + b_y)$$

these encoder are the keys

s is a query

$$s_1 = f(W_s \cdot h_4 + b_s)$$

$$s_{t+1} = RNN(s_t, y_t)$$

Decoder RNN

How to obtain s is not our focus.

Attentions (Formally)

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores e^t for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

There are several ways to do this

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

Attentions (Formally)

- Basic dot-product attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$
  - Note: this assumes $d_1 = d_2$
  - This is the version we saw earlier

$\boldsymbol{s} \in \mathbb{R}^{d_2}$

$\boldsymbol{h}_i \in \mathbb{R}^{d_1}$

- Multiplicative attention: $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{W} \boldsymbol{h}_i \in \mathbb{R}$
  - Where $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ is a weight matrix

- Additive attention: $\boldsymbol{e}_i = \boldsymbol{v}^T \tanh(\boldsymbol{W}_1 \boldsymbol{h}_i + \boldsymbol{W}_2 \boldsymbol{s}) \in \mathbb{R}$
  - Where $\boldsymbol{W}_1 \in \mathbb{R}^{d_3 \times d_1}$, $\boldsymbol{W}_2 \in \mathbb{R}^{d_3 \times d_2}$ are weight matrices and $\boldsymbol{v} \in \mathbb{R}^{d_3}$ is a weight vector.
  - d_3 (the attention dimensionality) is a hyperparameter

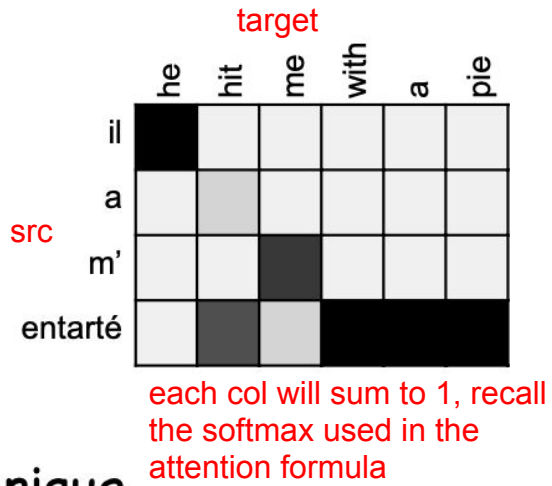
Advantages of Attentions

- Attention improves NMT performance significantly
  - It's necessary to allow decoder to focus on the relevant parts of the source
- Attention solves the bottleneck problem
  - allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states

Refer to https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3 for more illustrations

Advantages of Attentions

- Attention provides interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - (soft) alignment for free



each col will sum to 1, recall the softmax used in the attention formula

- Attention is a general Deep Learning technique
  - Not just seq2seq model
  - Not just NMT

Advantages of Attentions

- Attention is a general Deep Learning technique
  - We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$ **Keys/Values**
  - On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$ **Query**
- Given a set of value vectors, and a query vector, **attention** is a technique to compute a weighted sum of the values, dependent on the query.

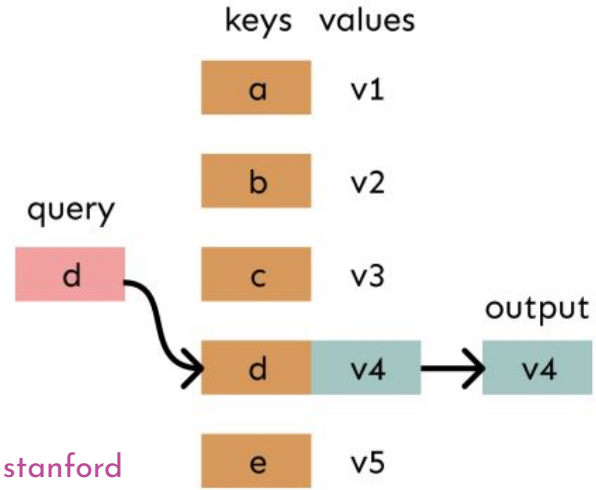
Intuition:

- The weighted sum is a selective summary of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a fixed-size representation of an arbitrary set of representations (the values), dependent on some other representation (the query).

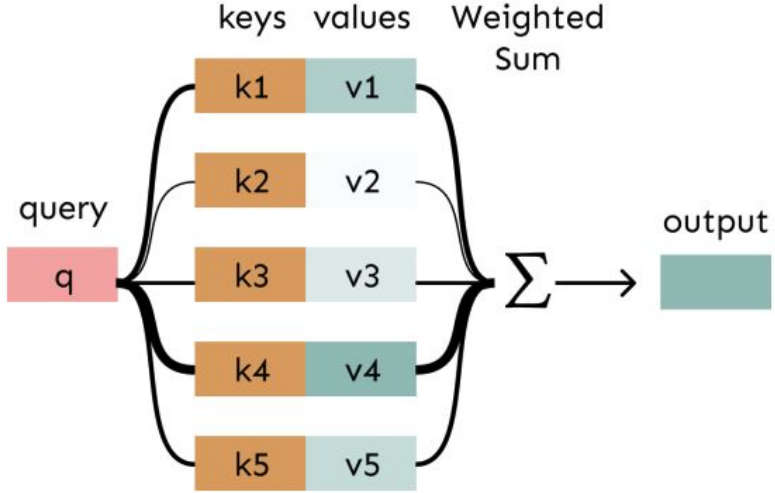
Attentions As Query-Key-Value Computation

We can think of **attention** as performing fuzzy lookup in a key-value store.

In a **lookup table**, we have a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.

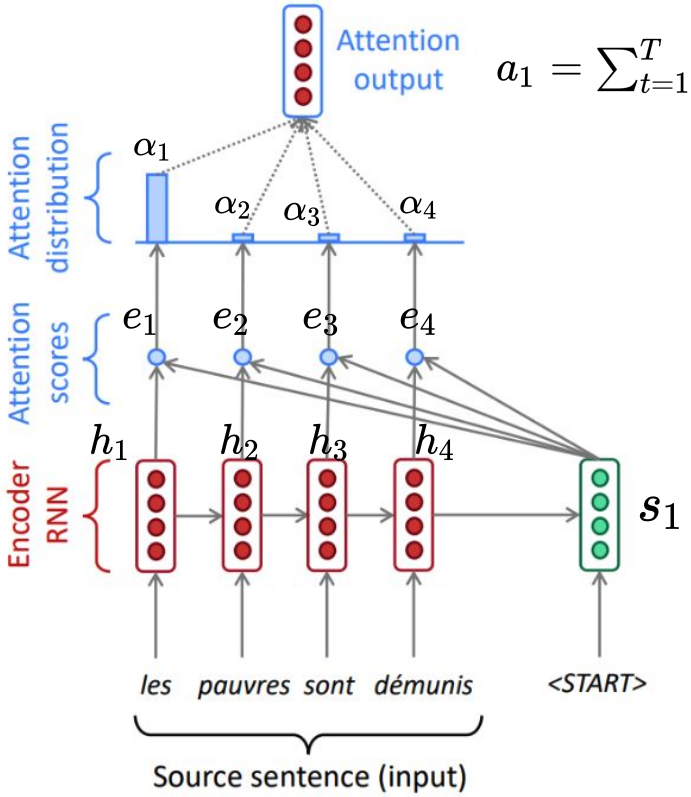


In **attention**, the **query** matches all **keys** *softly*, to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.



Source: stanford 224n

Attentions As Query-Key-Value Computation



$$a_1 = \sum_{t=1}^{T} \alpha_t \cdot h_t$$

$$\alpha_t = softmax(e_t) = \frac{\exp(e_1)}{\sum_{i=1}^{T} \exp(e_i)}$$

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

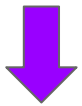
h is the **key** vector and **value** vector

s is the **query** vector

Self Attentions

- Attentions can be generally applied between encoder and decoder for sequence-to-sequence learning.
- How about attentions within a single sequence?

Why not use attention for representations?

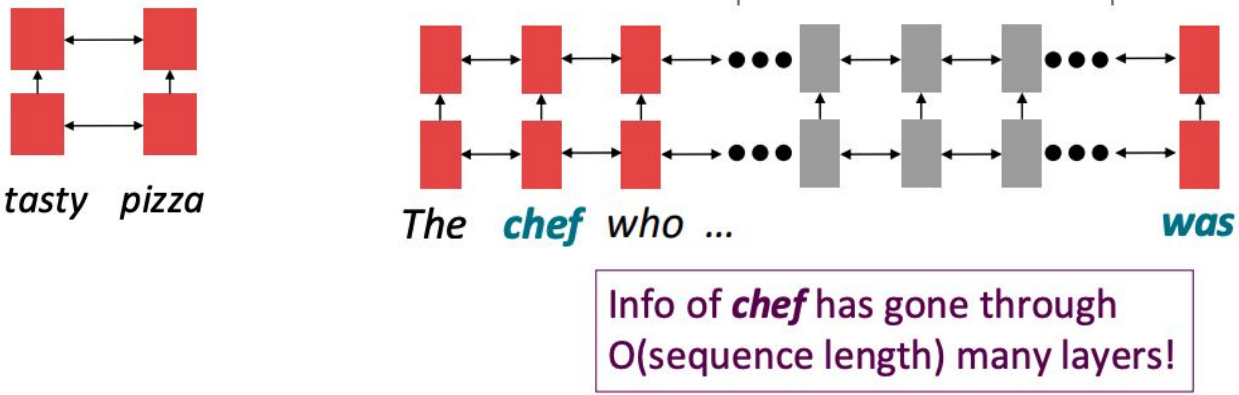


Self Attentions !!!

Why Self Attentions?

Source: stanford 224n

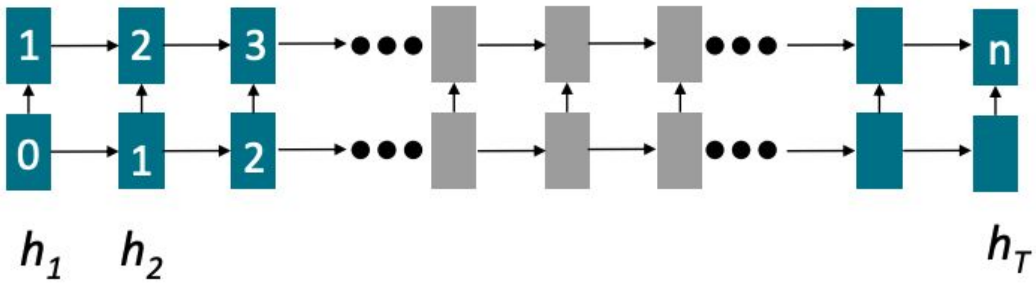
- RNNs are unrolled "left-to-right" or "right-to-left"
- Only encodes linear locality: nearby words affect more
- Takes several steps for distant word pairs to interact



Why Self Attentions?

Source: stanford 224n

- Forward and backward passes have O(sequence length) unparallelizable operations
- Future RNN hidden states can't be computed in full before past RNN hidden states have been computed
- Inhibits training on very large datasets



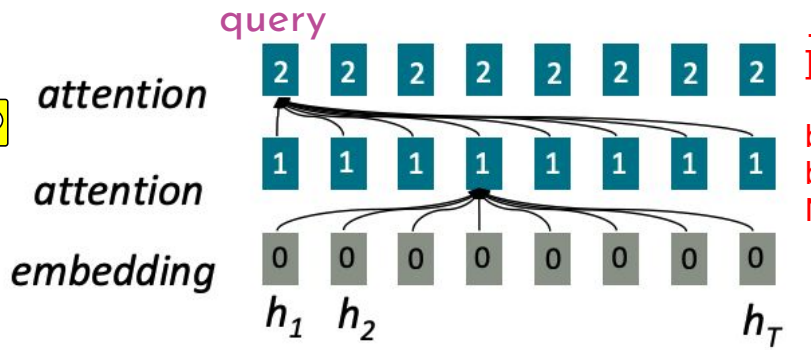
Numbers indicate min # of steps before a state can be computed

Why Self Attentions?

Source: stanford 224n

- Treats each word's representation as a query to access and incorporate information from a set of values.
- Easy to parallelize (per layer).
- Maximum interaction distance: O(1), since all words interact at every layer!

Each word can be query, key, value



attention score is
[e_4^t e_1
e_4^t e_2
…
]

but can we do better? Next slide

All words attend to all words in previous layer; most arrows here are omitted

Self Attention Mechanism

Source: stanford 224n

Let $\boldsymbol{w}_{1:n}$ be a sequence of words in vocabulary V, like *Zuko made his uncle tea*.

For each $\boldsymbol{w}_i$, let $\boldsymbol{x}_i = E\boldsymbol{w}_i$, where $E \in \mathbb{R}^{d \times |V|}$ is an embedding matrix.

1. Transform each word embedding with weight matrices $\mathrm{Q}, \mathrm{K}, \mathrm{V}$, each in $\mathbb{R}^{d \times d}$

$\boldsymbol{q}_i = Q\boldsymbol{x}_i$ (queries) $\boldsymbol{k}_i = K\boldsymbol{x}_i$ (keys) $\boldsymbol{v}_i = V\boldsymbol{x}_i$ (values)

there is a problem, in the attention calculation, dot product of itself result in highest score, we need to handle that

2. Compute pairwise similarities between keys and queries; normalize with softmax

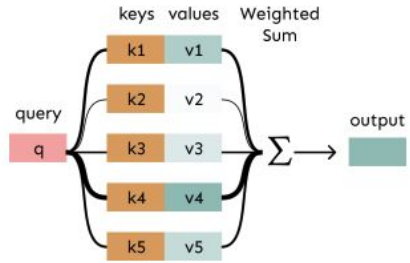
q is a single vector

$$\boldsymbol{e}_{ij} = \boldsymbol{q}_i^\top \boldsymbol{k}_j \qquad \boldsymbol{\alpha}_{ij} = \frac{\exp(\boldsymbol{e}_{ij})}{\sum_{j'} \exp(\boldsymbol{e}_{ij'})}$$

note that k_j is a entire seq
look at img on right

3. Compute output for each word as weighted sum of values

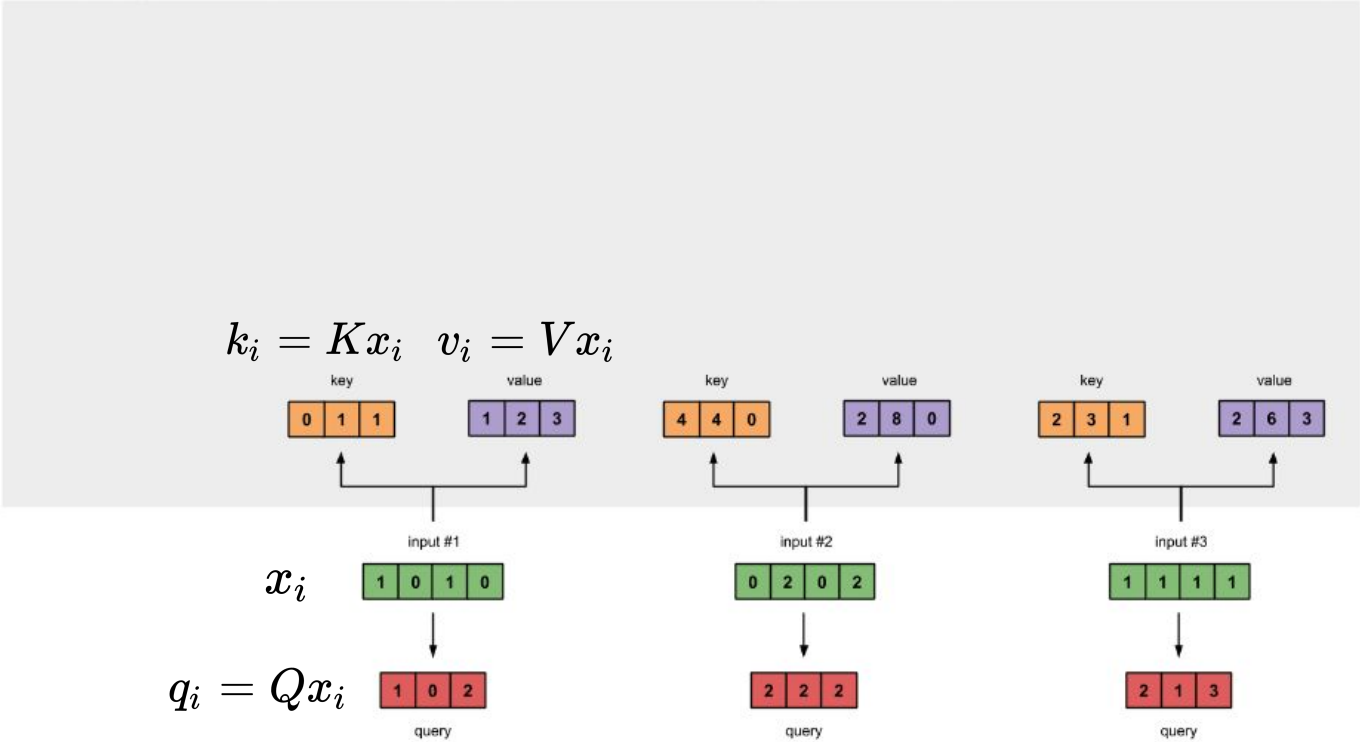
$$o_i = \sum_j \alpha_{ij} v_j$$



Self Attention Mechanism

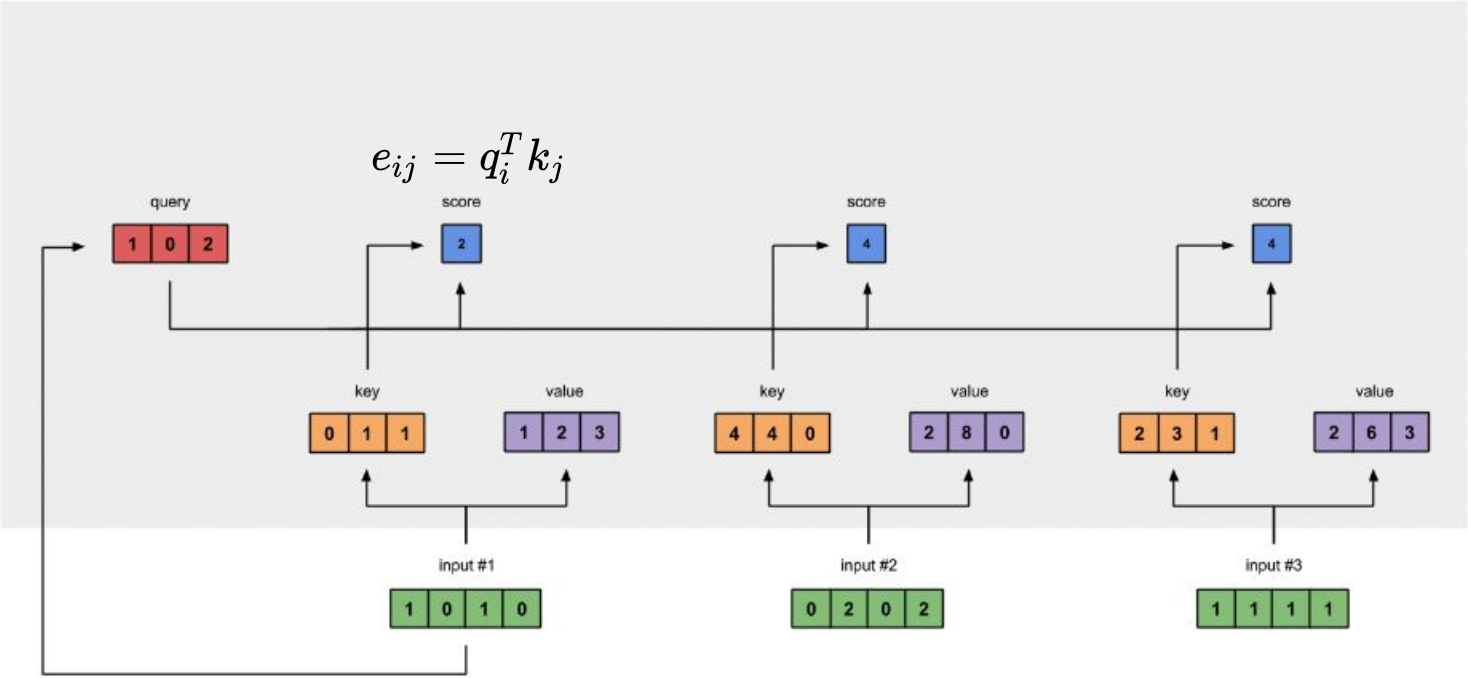
Source: https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self Attention Mechanism



Source: https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

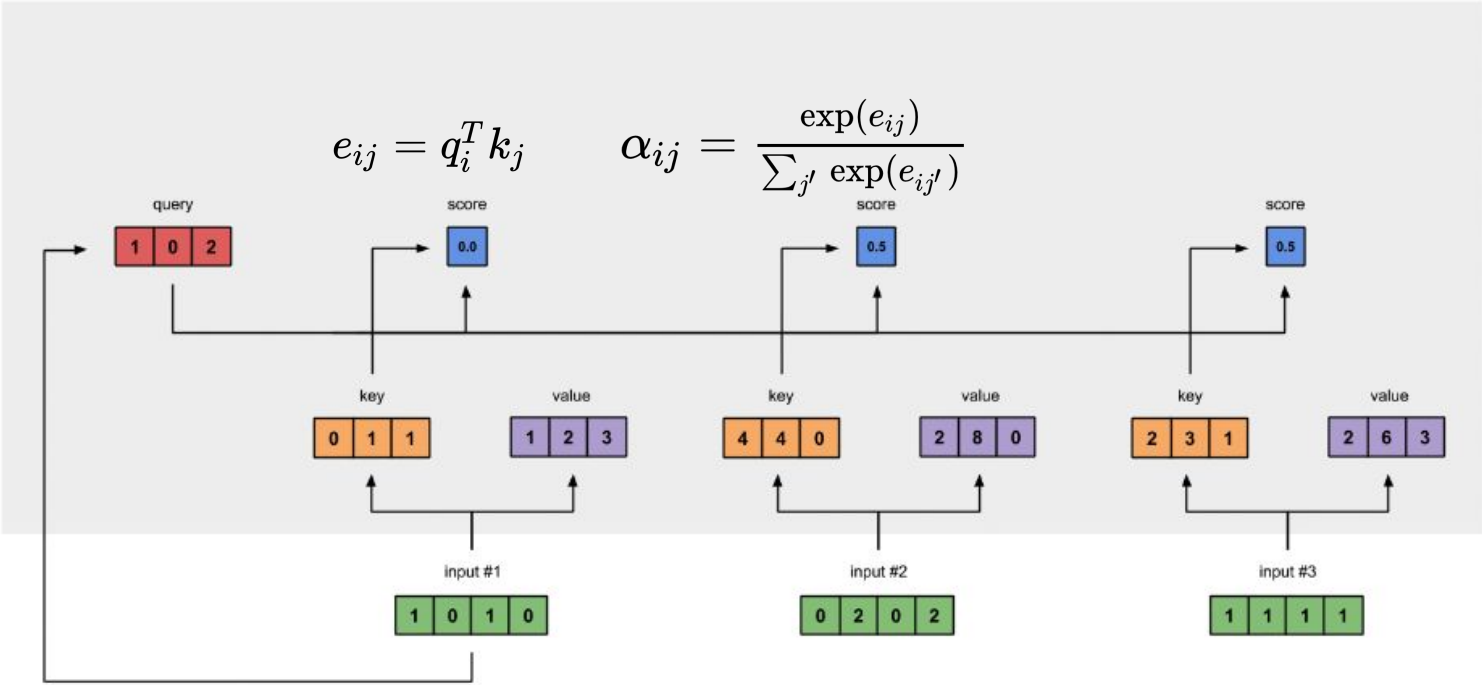
Self Attention Mechanism



Source: https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

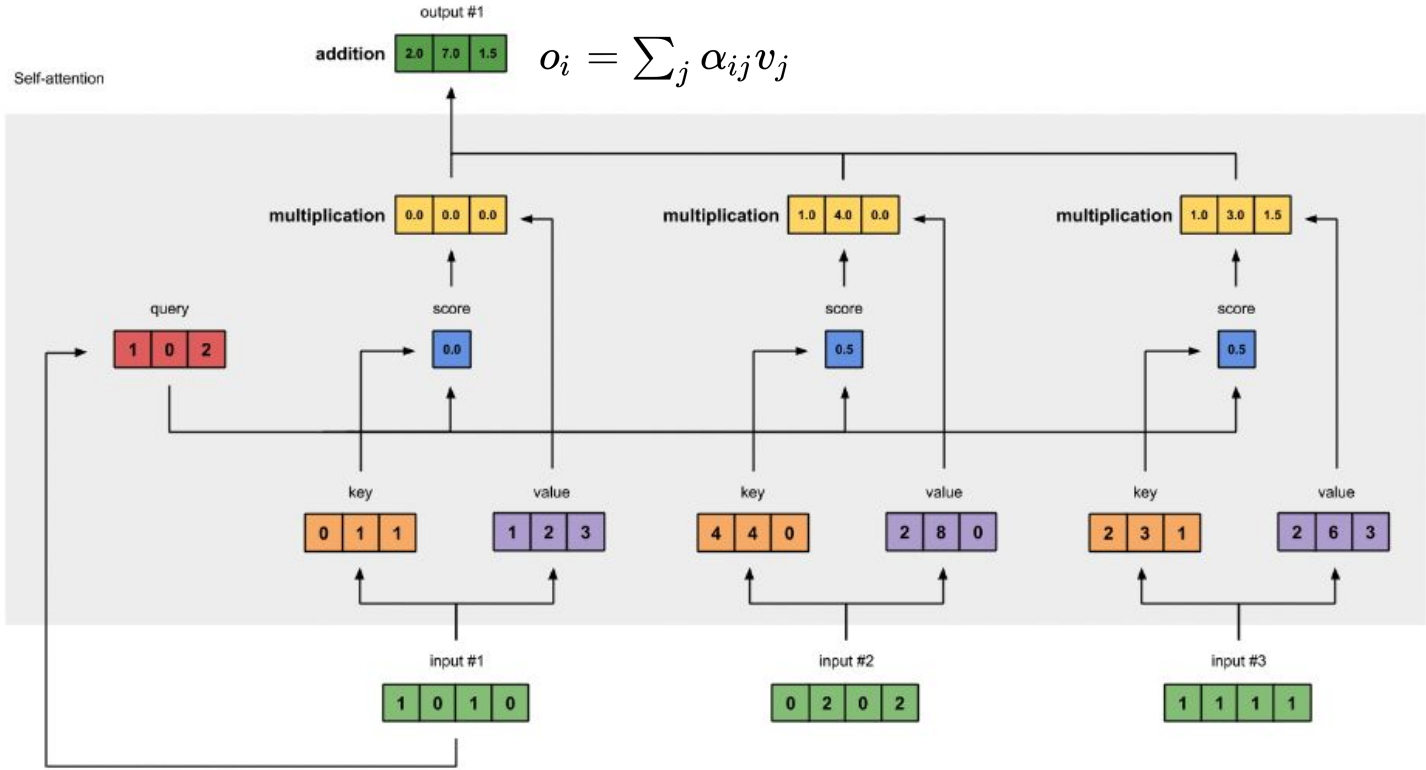
Self Attention Mechanism

$$e_{ij} = q_i^T k_j \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$



Source: https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self Attention Mechanism



Source: https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Outline for today

01 From RNNs to Attentions

02 Transformer Model

03 Applications of Transformers

From Self Attention to Transformer

- Transformers are basic modules commonly adopted in various language models.
- It's a lot like our minimal self attention architecture, but with a few more components.
  - Adding position embeddings
  - Adding non-linearities
  - Single to multi-head self attention
  - Multiple layers

Position Embedding

Barriers

- Doesn't have an inherent notion of order!

Solutions

Source: stanford 224n

Position Embedding

- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each **sequence index** as a **vector**

$$\boldsymbol{p}_i \in \mathbb{R}^d, \text{ for } i \in \{1,2,\dots,n\} \text{ are position vectors}$$

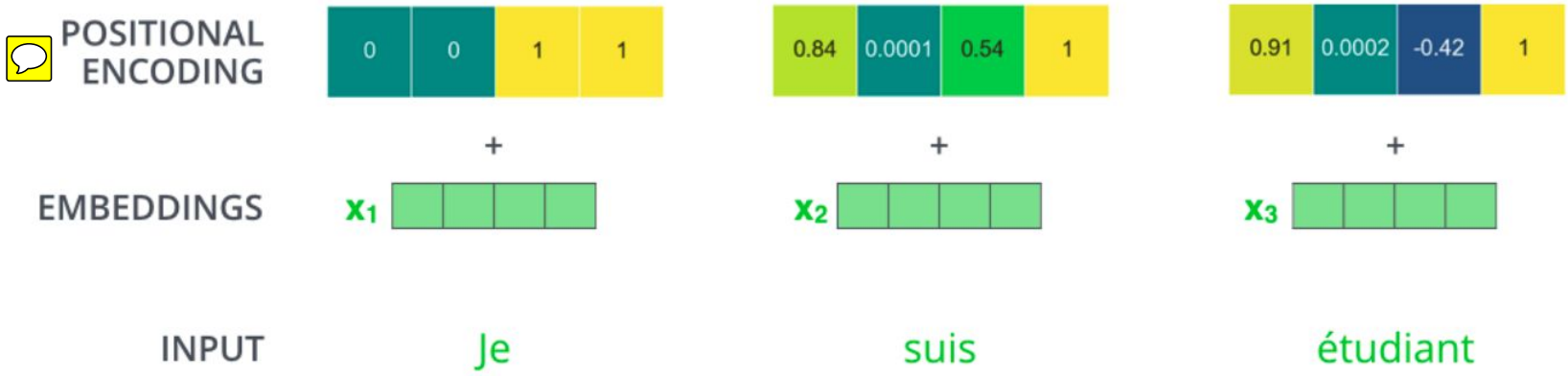
- Don't worry about what the p_i are made of yet!
- Easy to incorporate this info into our self-attention block: just add the $\boldsymbol{p}_i$ to our inputs!
- Recall that $\boldsymbol{x}_i$ is the embedding of the word at index i. The positioned embedding is:

$$\boldsymbol{x}_i = \boldsymbol{x}_i + \boldsymbol{p}_i$$

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add...

Source: stanford 224n

Position Embedding



$$x_i = x_i + p_i$$

In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add...

Position Embedding

- **Learned absolute position representations:** Let all p_i be learnable parameters!
  Learn a matrix $\boldsymbol{p} \in \mathbb{R}^{d \times n}$, *and let each* $\boldsymbol{p}_i$ *be a column of that matrix!*

- Pros:
  - Flexibility: each position gets to be learned to fit the data
- Cons:
  - Definitely can't extrapolate to indices outside $1, \ldots, n$.
- Most systems use this!

Source: stanford 224n

Non-linearities

Barriers		Solutions
• Doesn't have an inherent notion of order!	→	• Add position representations to the inputs
• No nonlinearities for deep learning! It's all just weighted averages	→	

Source: stanford 224n

Nonlinearities

Source: stanford 224n

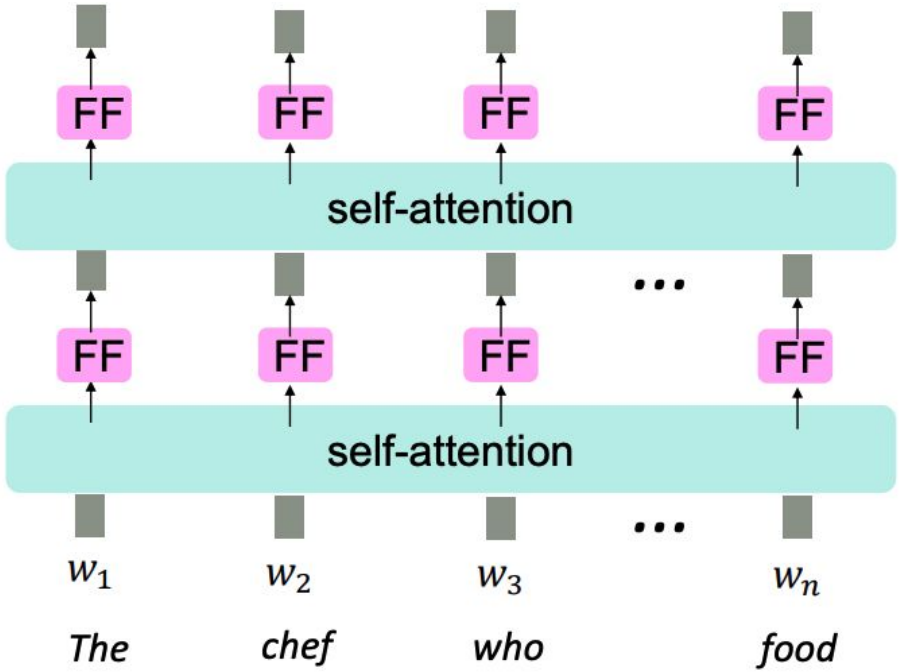
- Note that there are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages **value** vectors (Why? Look at the notes!)

- Easy fix: add a **feed-forward network** to post-process each output vector.

multi layer perceptron

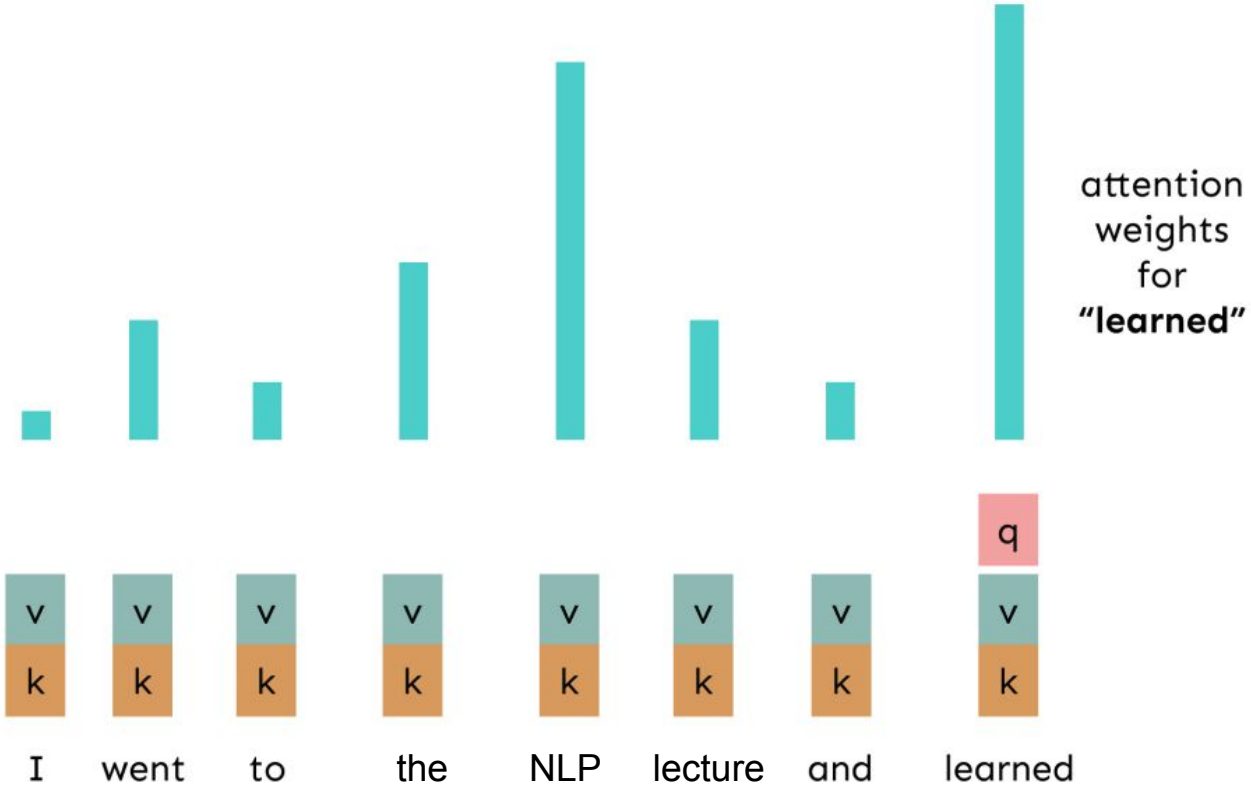
$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1\ \text{output}_i + b_1) + b_2$$

like the relu here



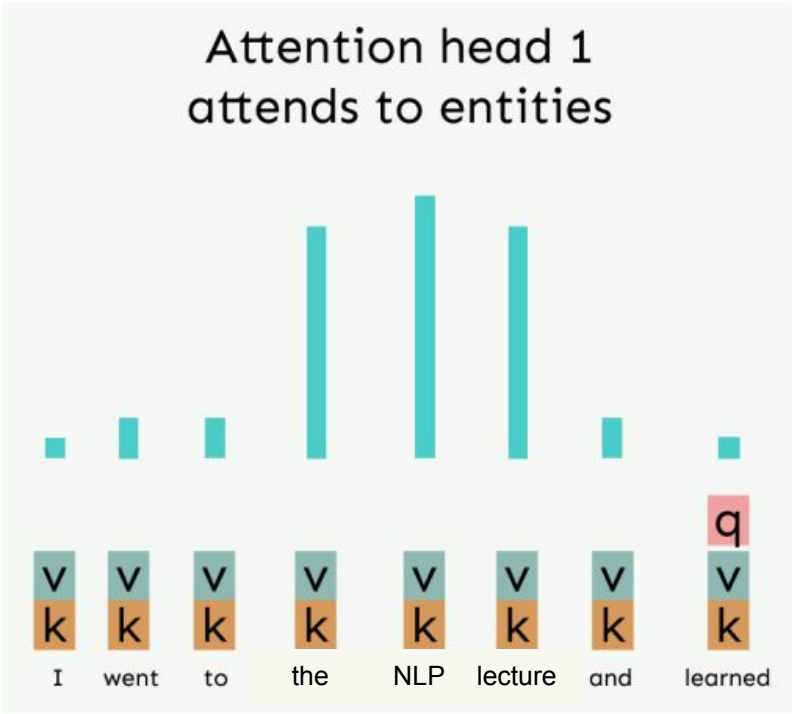
Recall a Single Self Attention

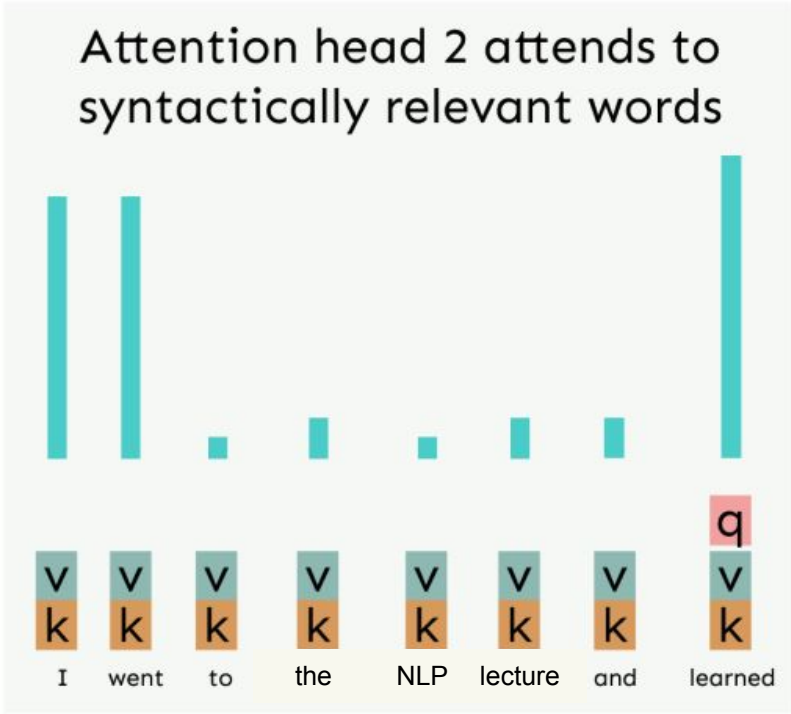
Source: stanford 224n



Multi-head Self Attention

Source: stanford 224n





Matrix-Form Computation

- Let's look at how query-key-value attention is computed in matrix form
  - Let $X = [x_1; \ldots; x_n] \in \mathbb{R}^{n \times d}$ be the concatenation of input vectors
  - Compute query, key, value matrices: $XQ, XK, XV \in \mathbb{R}^{n \times d}$

- Compute the output using self attentions
  - Generate attention score matrix

$$A = softmax(XQ \cdot (XK)^T) \in \mathbb{R}^{n \times n}$$

$A =$

	Hello	I	love	you
Hello	0.8	0.1	0.05	0.05
I	0.1	0.6	0.2	0.1
love	0.05	0.2	0.65	0.1
you	0.2	0.1	0.1	0.6

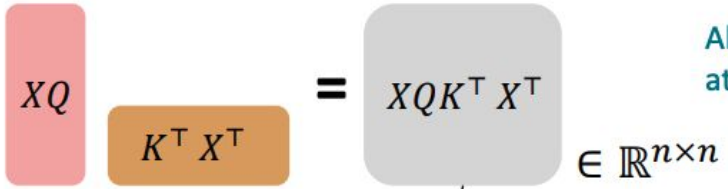
  - Aggregate with value matrix

$$output = A \cdot (XV) \in \mathbb{R}^{n \times d}$$

Matrix-Form Computation

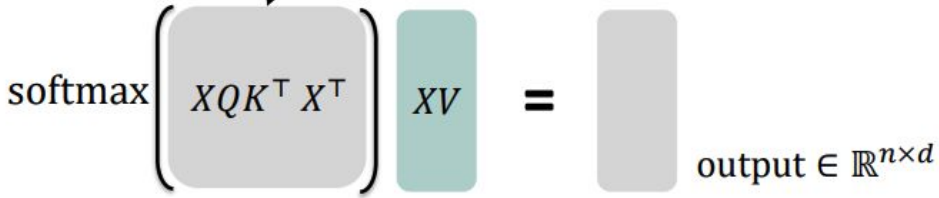
Source: stanford 224n

First, take the query-key dot products in one matrix multiplication: $XQ(XK)^\top$



All pairs of attention scores!

Next, softmax, and compute the weighted average with another matrix multiplication.



Matrix-Form Computation

Source: stanford 224n

- What if we want to look in multiple places in the sentence at once?
  - For word i, self-attention "looks" where $x_i^\top Q^\top K x_j$ is high, but maybe we want to focus on different j for different reasons?
- We'll define **multiple attention "heads"** through multiple Q,K,V matrices
- Let, $Q_\ell, K_\ell, V_\ell \in \mathbb{R}^{d \times \frac{d}{h}}$, where h is the number of attention heads, and ℓ ranges from 1 to h.

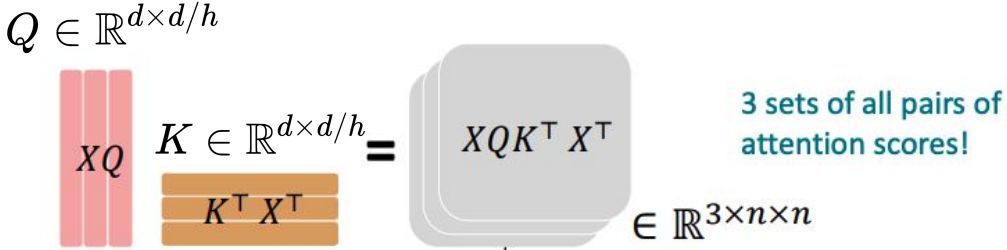
we shrink the 2nd dimension so that we can concat the output

- Each attention head performs attention independently:
- $\text{output}_\ell = \text{softmax}\left(XQ_\ell K_\ell^\top X^\top\right) * XV_\ell$, where $\text{output}_\ell \in \mathbb{R}^{d/h}$
- Then the outputs of all the heads are combined!
  - $\text{output} = [\text{output}_1; \ldots; \text{output}_h]Y$, where $Y \in \mathbb{R}^{d \times d}$
- Each head gets to "look" at different things, and construct value vectors differently.

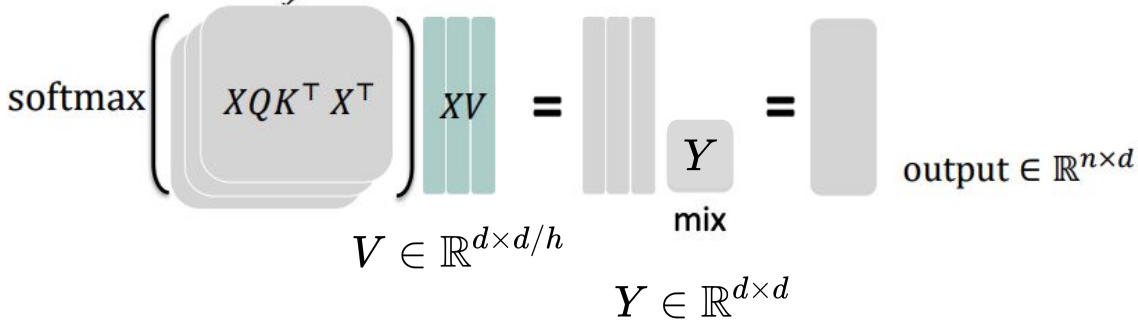
Matrix-Form Computation

Source: stanford 224n

First, take the query-key dot products in one matrix multiplication: $XQ(XK)^\top$



Next, softmax, and compute the weighted average with another matrix multiplication.



Scaled Dot Product

- **"Scaled Dot Product"** attention aids in training.
- When dimensionality d becomes large, dot products between vectors tend to become large.
  - Because of this, inputs to the softmax function can be large, making the gradients small.
- Instead of the self-attention function we've seen:

$$\text{output}_\ell = \text{softmax}\left(XQ_\ell K_\ell^\top X^\top\right) * XV_\ell$$

- We divide the attention scores by $\sqrt{d/h}$, to stop the scores from becoming large just as a function of d/h (The dimensionality divided by the number of heads.)

$$\text{output}_\ell = \text{softmax}\left(\frac{XQ_\ell K_\ell^\top X^\top}{\sqrt{d/h}}\right) * XV_\ell$$

Process

1) This is our input sentence*

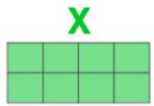
2) We embed each word*

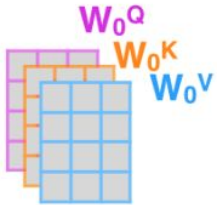
3) Split into 8 heads. We multiply X or R with weight matrices

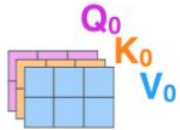
4) Calculate attention using the resulting Q/K/V matrices

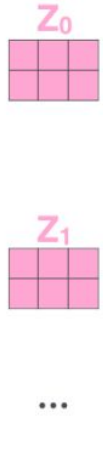
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

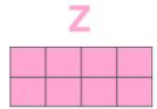
Thinking Machines





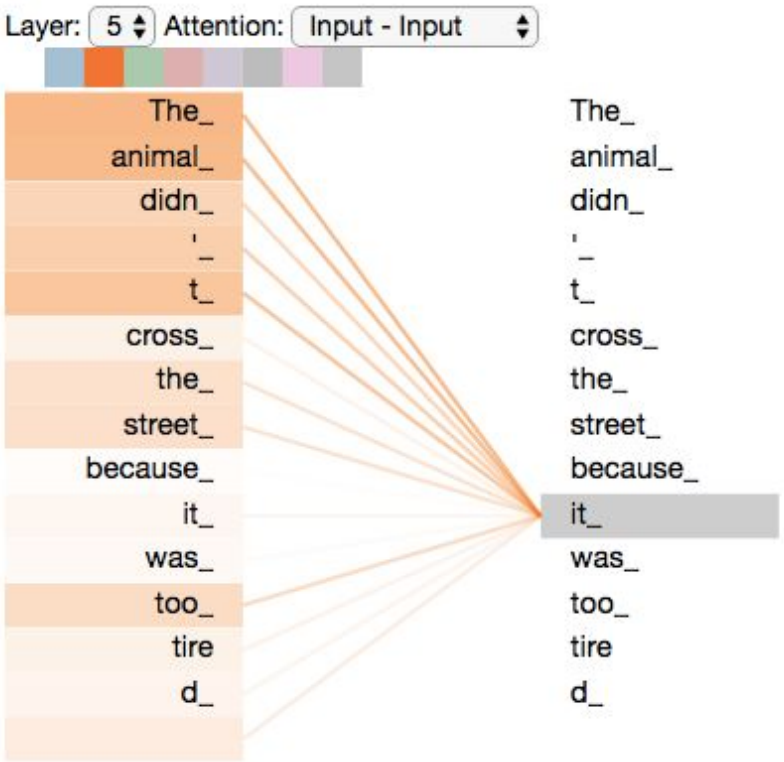


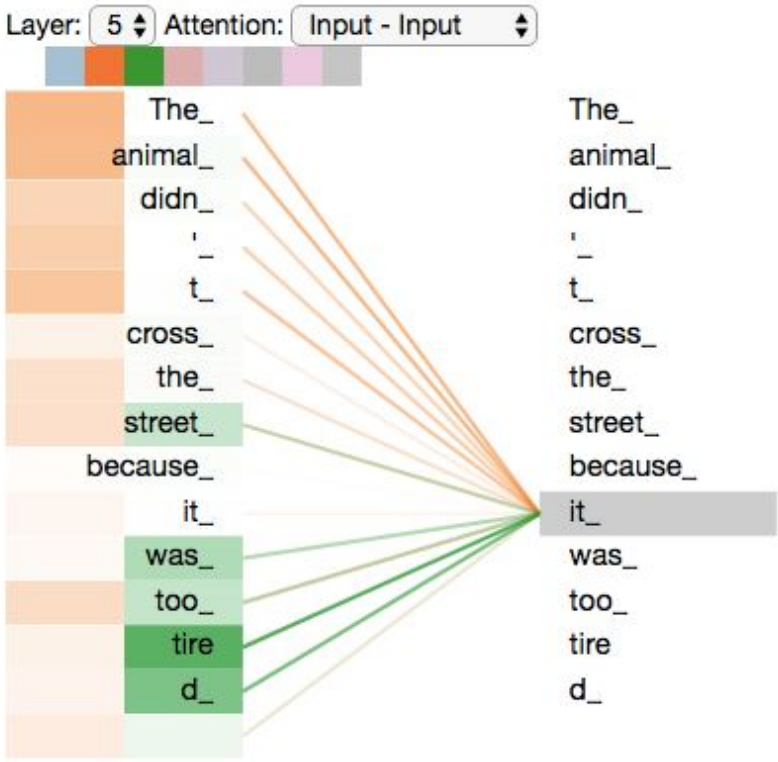




* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

An Example

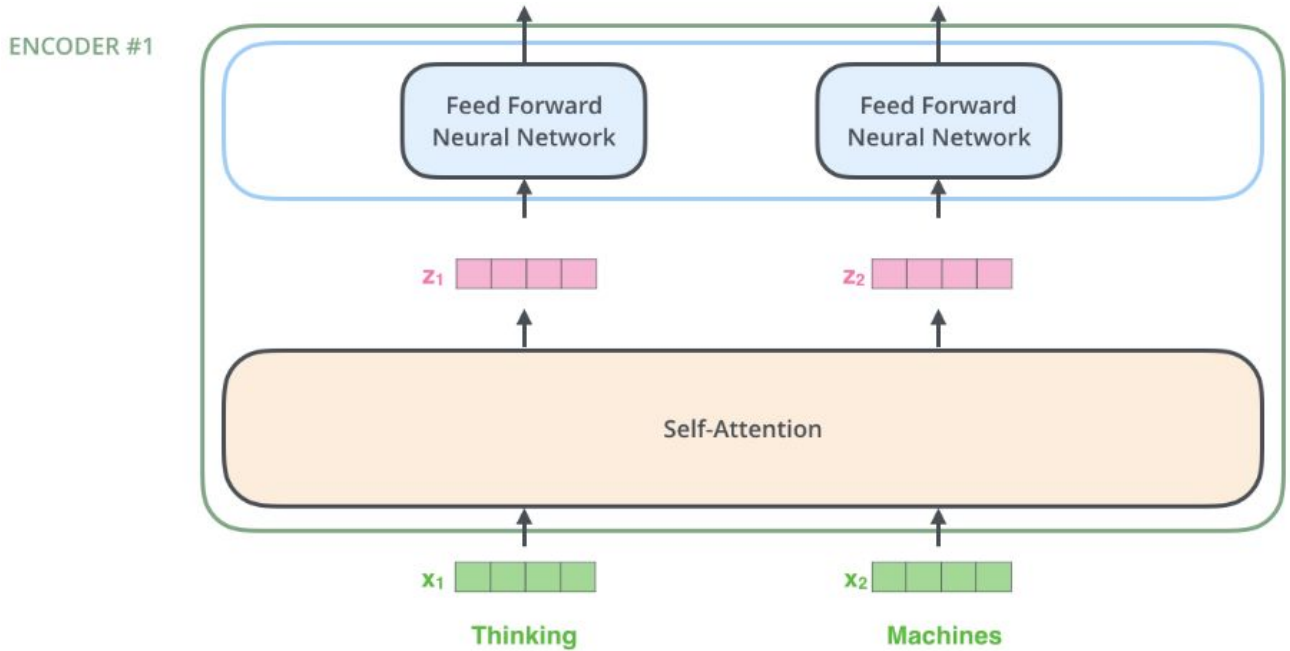

Layer: 5 Attention: Input - Input
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_


Layer: 5 Attention: Input - Input
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_

From Self Attention to Transformer

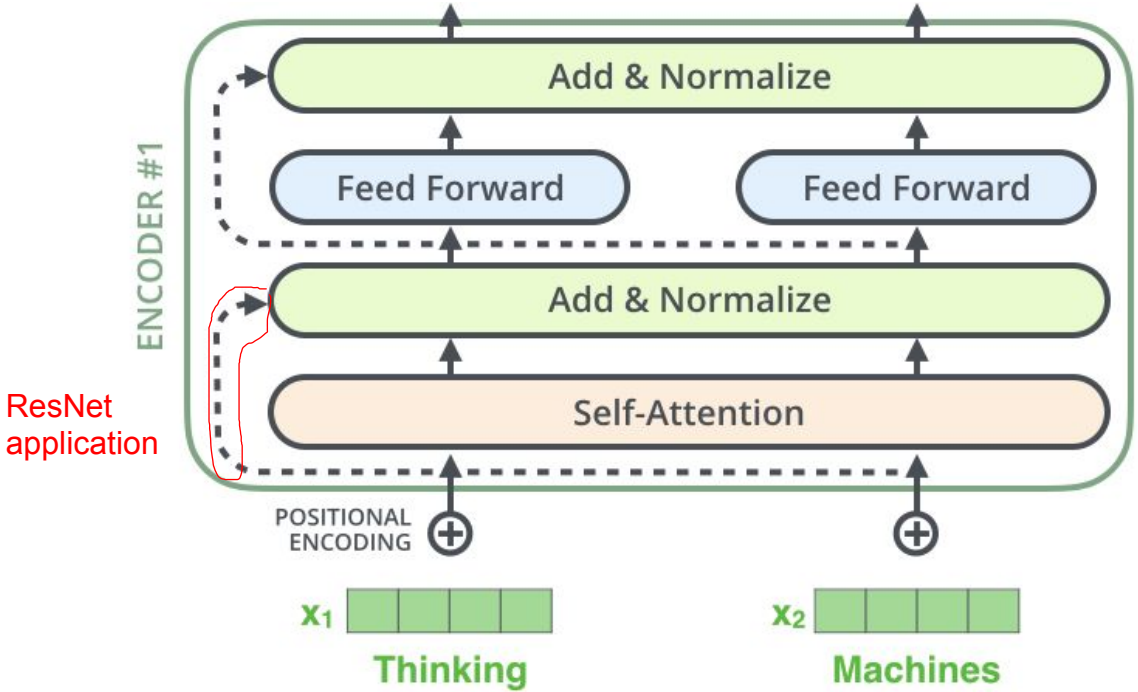
- Now that we've replaced self attention with multi-head self attention, we'll go through two optimization tricks
  - Residual connections
  - Layer normalizations
- In most Transformer diagrams, these are often written together as "Add & Norm"

Self Attention



Source: http://jalammar.github.io/illustrated-transformer/

Self Attention with Add & Norm



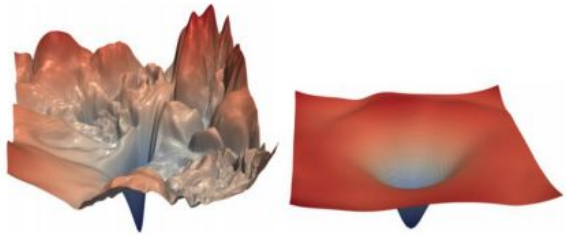
Source: http://jalammar.github.io/illustrated-transformer/

Residual Connections (Recap)

- **Residual connections** are a trick to help models train better.
  - Instead of $X^{(i)} = \text{Layer}(X^{(i-1)})$ (where i represents the layer)

  - We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn "the residual" from the previous layer)

- Gradient is **great** through the residual connection; it's 1!
- Bias towards the identity function!



[no residuals] [residuals]

[Loss landscape visualization, Li et al., 2018, on a ResNet]

He et al. Deep Residual Learning for Image Recognition

Layer Normalization [Ba et al. 2016]

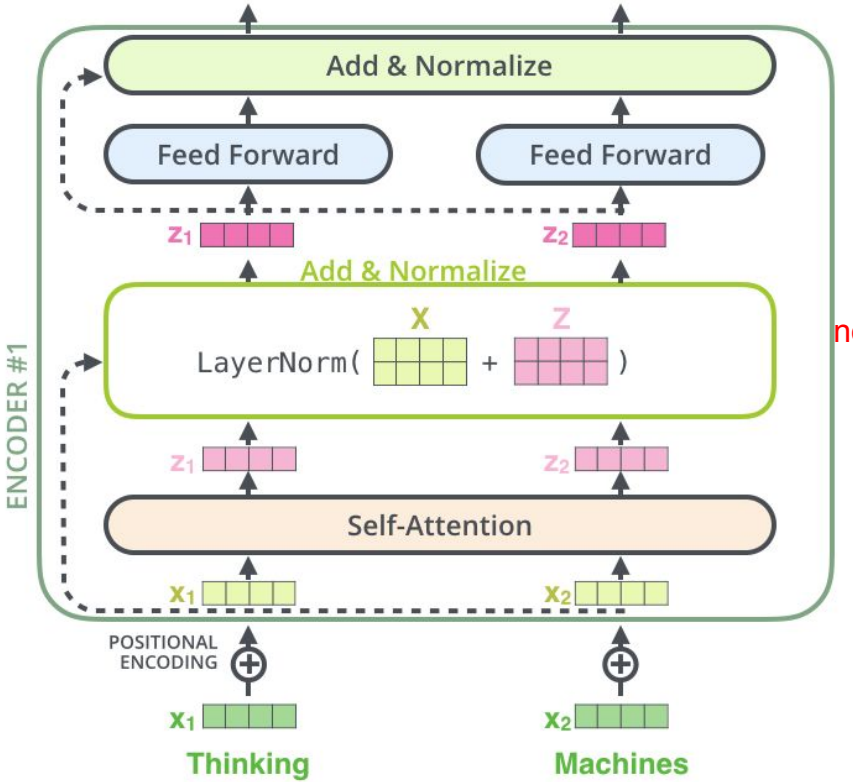
- **Layer normalization** is a trick to help models train faster.
- Idea: cut down on uninformative variation in hidden vector values by normalizing to unit mean and standard deviation **within each layer**.
  - LayerNorm's success may be due to its normalizing gradients [Xu et al., 2019]
- Let $x \in \mathbb{R}^d$ be an individual (word) vector in the model.
- Let $\mu = \sum_{j=1}^{d} x_j$; this is the mean; $\mu \in \mathbb{R}$.
- Let $\sigma = \sqrt{\frac{1}{d}\sum_{j=1}^{d}(x_j - \mu)^2}$; this is the standard deviation; $\sigma \in \mathbb{R}$.
- Let $\gamma \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^d$ be learned "gain" and "bias" parameters. (Can omit!)
- Then layer normalization computes:

$$\text{output} = \frac{x - \mu}{\sqrt{\sigma + \epsilon}} * \gamma + \beta$$

Normalize by scalar mean and variance

Modulate by learned elementwise gain and bias

Self Attention with Add & Norm



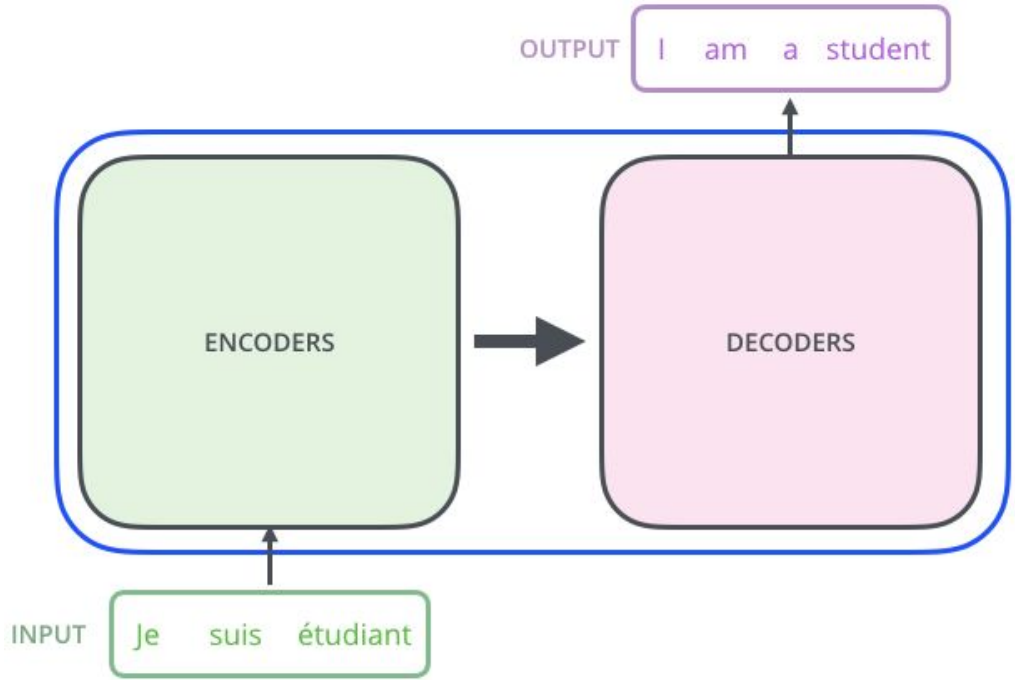
normalization for controlled dist

Source: http://jalammar.github.io/illustrated-transformer/

Transformer Encoder-Decoder

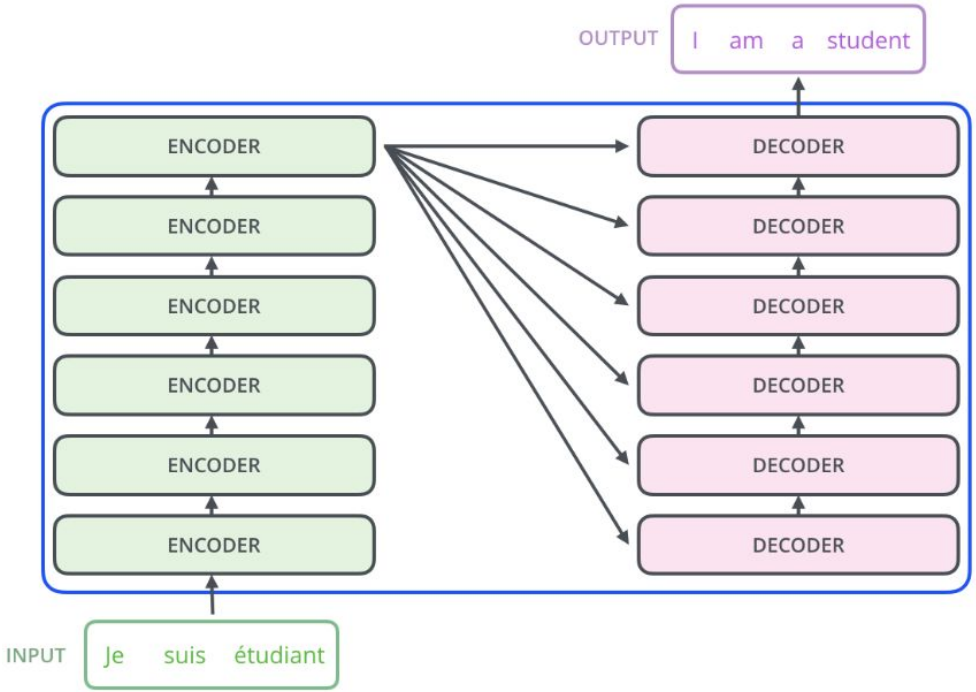
Source: http://jalammar.github.io/illustrated-transformer/

Transformer Encoder-Decoder



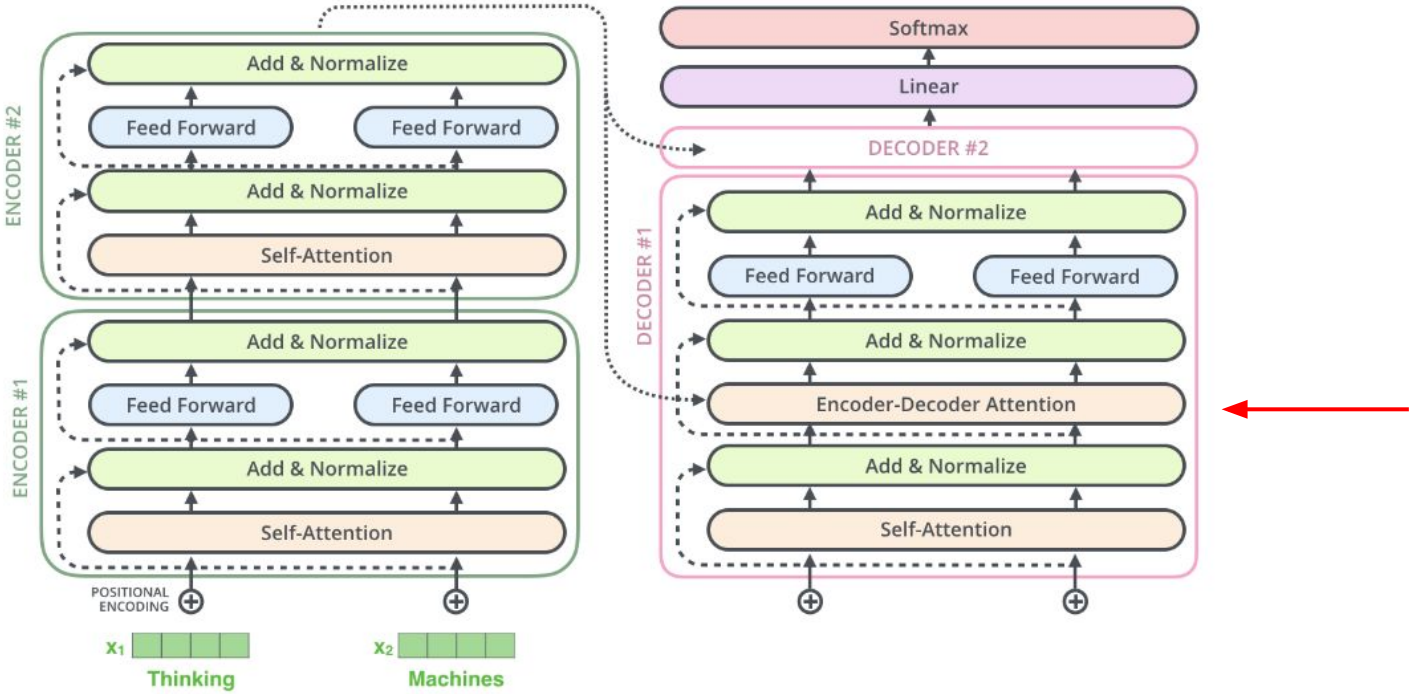
Source: http://jalammar.github.io/illustrated-transformer/

Transformer Encoder-Decoder



Source: http://jalammar.github.io/illustrated-transformer/

Transformer Encoder-Decoder (Optional)

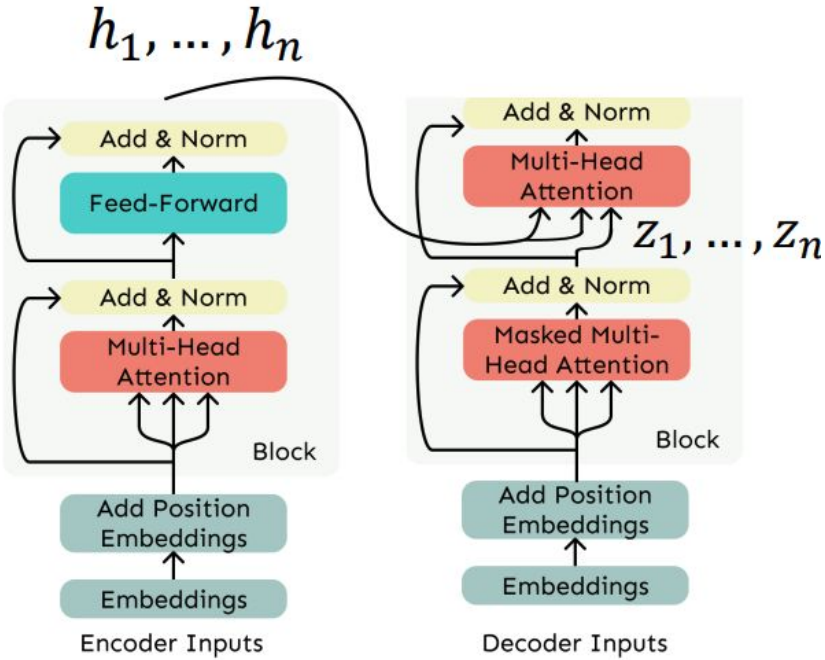


Source: http://jalammar.github.io/illustrated-transformer/

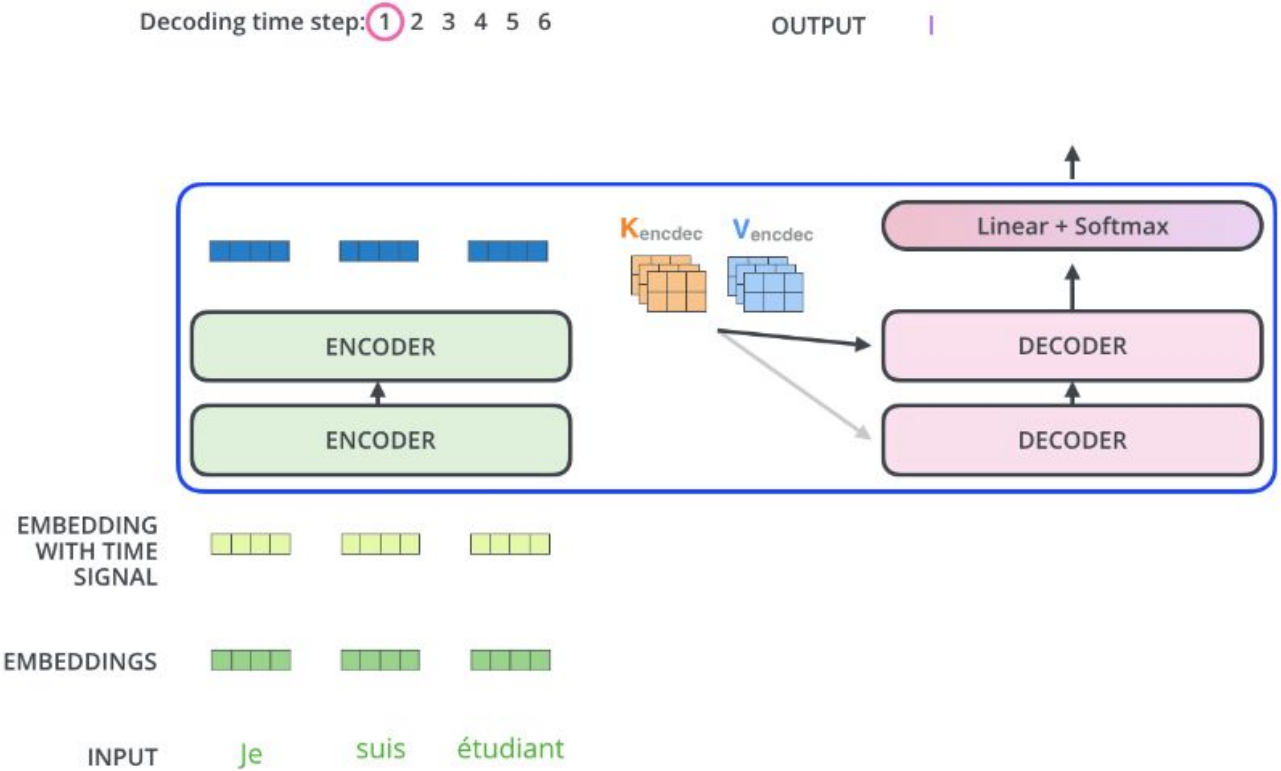
Encoder-Decoder Attention (Optional)

Source: stanford 224n

- We saw that self-attention is when keys, queries, and values come from the same source.
- In the decoder, we have attention that looks more like what we saw last week.
- Let $h_1, \dots, h_n$ be **output** vectors **from** the Transformer **encoder**; $x_i \in \mathbb{R}^d$
- Let $z_1, \dots, z_n$ be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$
- Then keys and values are drawn from the **encoder** (like a memory):
  - $k_i = Kh_i, v_i = Vh_i.$
- And the queries are drawn from the **decoder**, $q_i = Qz_i$.

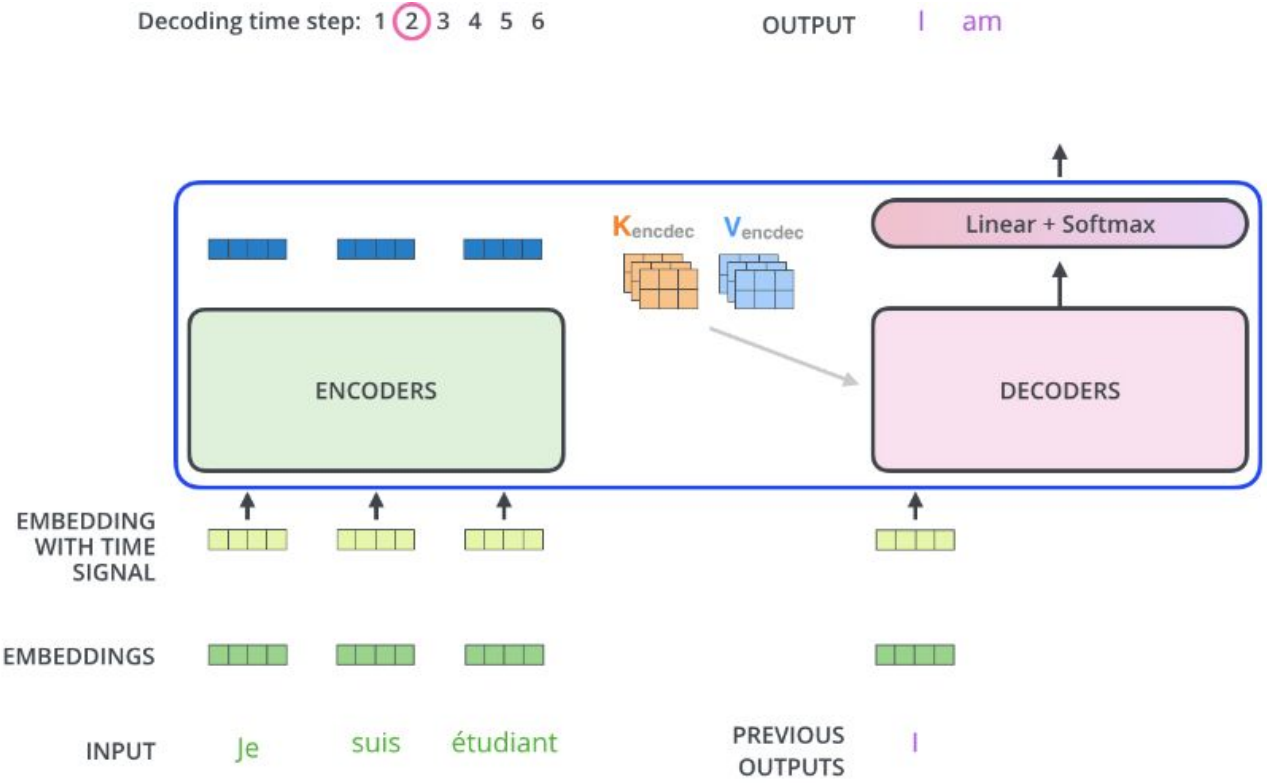


Transformer Encoder-Decoder



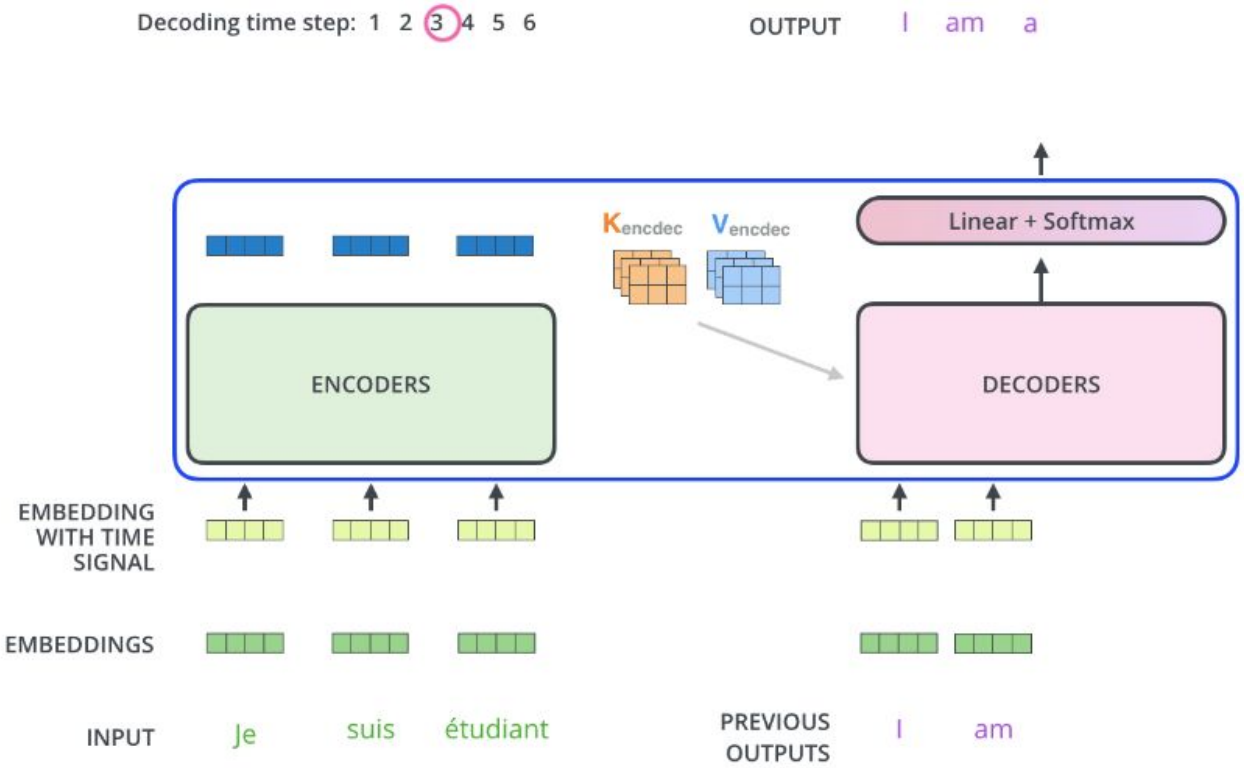
Source: http://jalammar.github.io/illustrated-transformer/

Transformer Encoder-Decoder



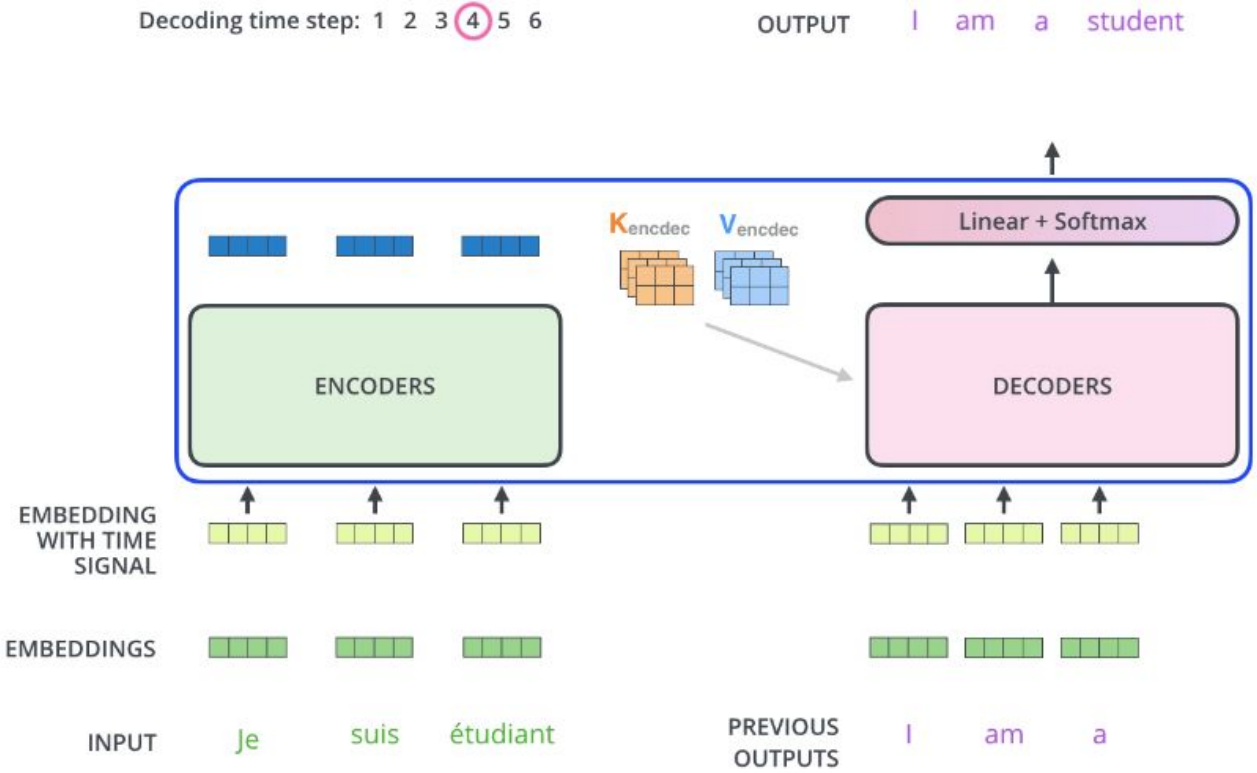
Source: http://jalammar.github.io/illustrated-transformer/

Transformer Encoder-Decoder



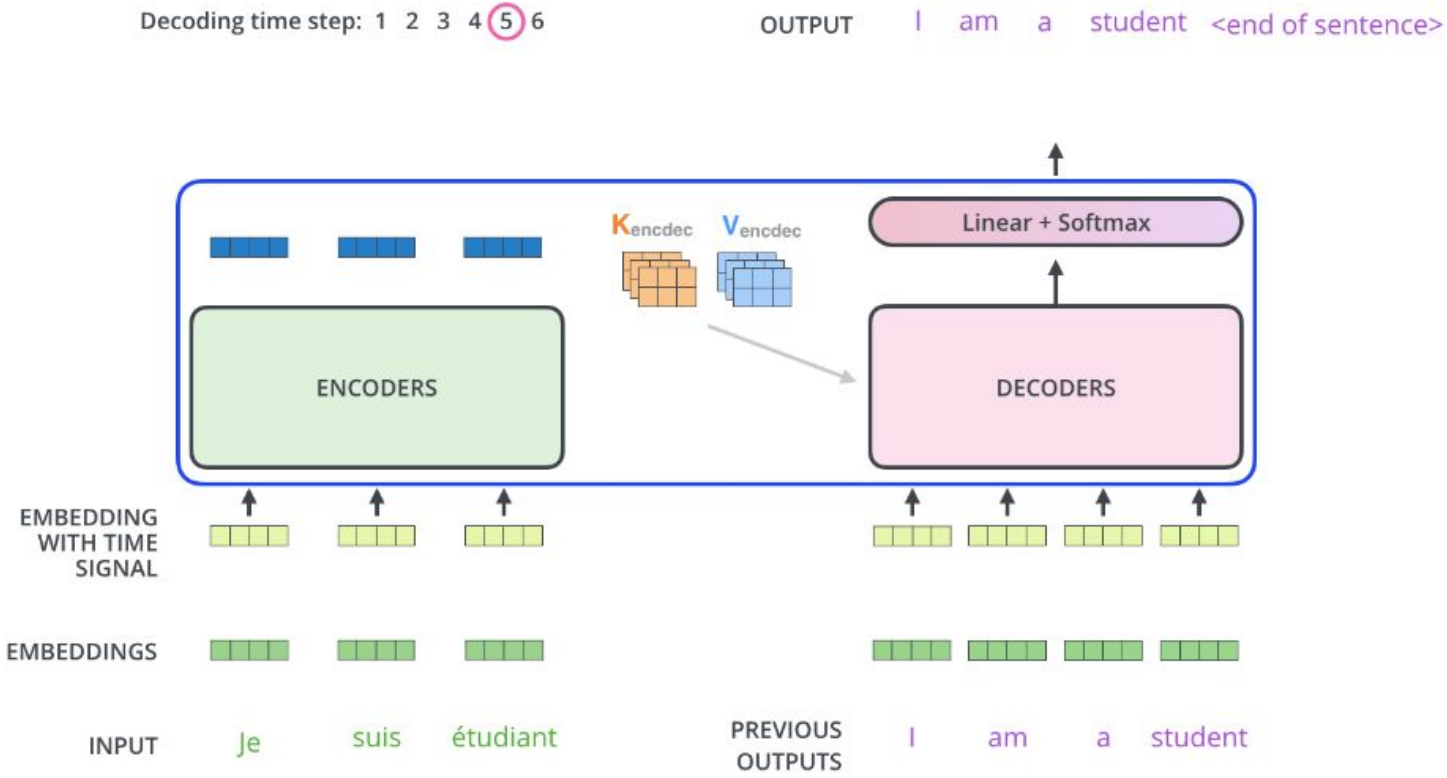
Source: http://jalammar.github.io/illustrated-transformer/

Transformer Encoder-Decoder



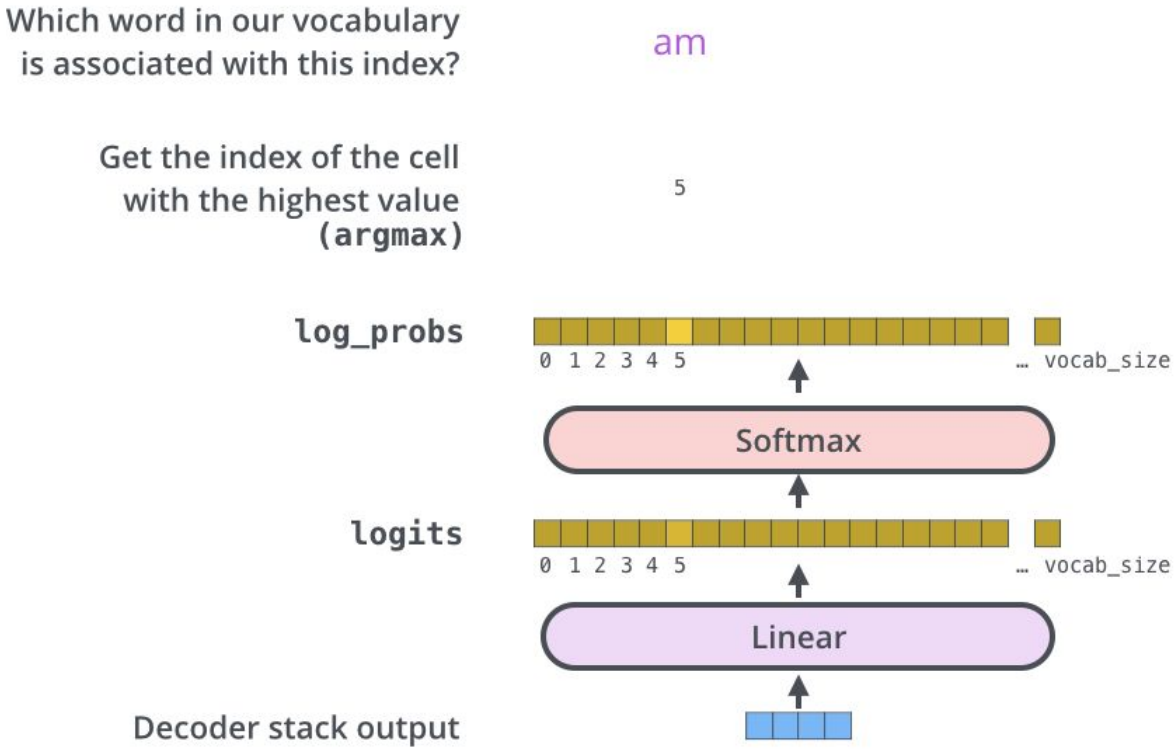
Source: http://jalammar.github.io/illustrated-transformer/

Transformer Encoder-Decoder



Source: http://jalammar.github.io/illustrated-transformer/

Transformer Encoder-Decoder



Source: http://jalammar.github.io/illustrated-transformer/

Outline for today

01 From RNNs to Attentions

02 Transformer Model

03 Applications of Transformers

Great Results with Transformers

Source: stanford 224n

First, Machine Translation from the original Transformers paper!

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$

WMT 2014 English-German and English-French, Vaswani et al., 2017

Great Results with Transformers

Source: stanford 224n

Next, document generation!

Model	Test perplexity	ROUGE-L
seq2seq-attention, $L = 500$	5.04952	12.7
Transformer-ED, $L = 500$	2.46645	34.2
Transformer-D, $L = 4000$	2.22216	33.6
Transformer-DMCA, no MoE-layer, $L = 11000$	2.05159	36.2
Transformer-DMCA, MoE-128, $L = 11000$	1.92871	37.9
Transformer-DMCA, MoE-256, $L = 7500$	1.90325	38.8

The old standard

Transformers all the way down.

WikiSum dataset, Liu et al., 2018

Great Results with Transformers

Source: stanford 224n

Transformers' parallelizability allows for efficient pretraining, and have made them the de-facto standard.

On this popular aggregate benchmark, for example:

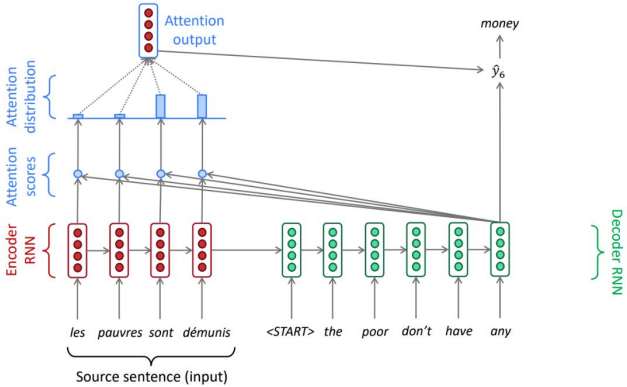
All top models are Transformer (and pretraining)-based.

	Rank	Name	Model	URL	Score
	1	DeBERTa Team - Microsoft	DeBERTa / TuringNLRv4		90.8
	2	HFL iFLYTEK	MacALBERT + DKM		90.7
+	3	Alibaba DAMO NLP	StructBERT + TAPT		90.6
+	4	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6
	5	ERNIE Team - Baidu	ERNIE		90.4
	6	T5 Team - Google	T5		90.3

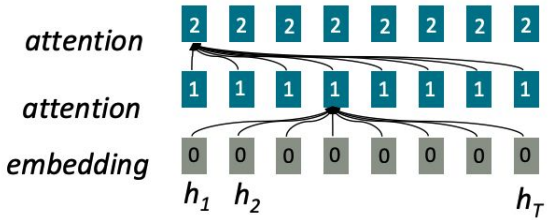
Learn Transformers (for your interest)

- Key recommended resource:
  - The annotated transformer
    http://nlp.seas.harvard.edu/2018/04/03/attention.html
  - The illustrated transformer
    https://jalammar.github.io/illustrated-transformer/

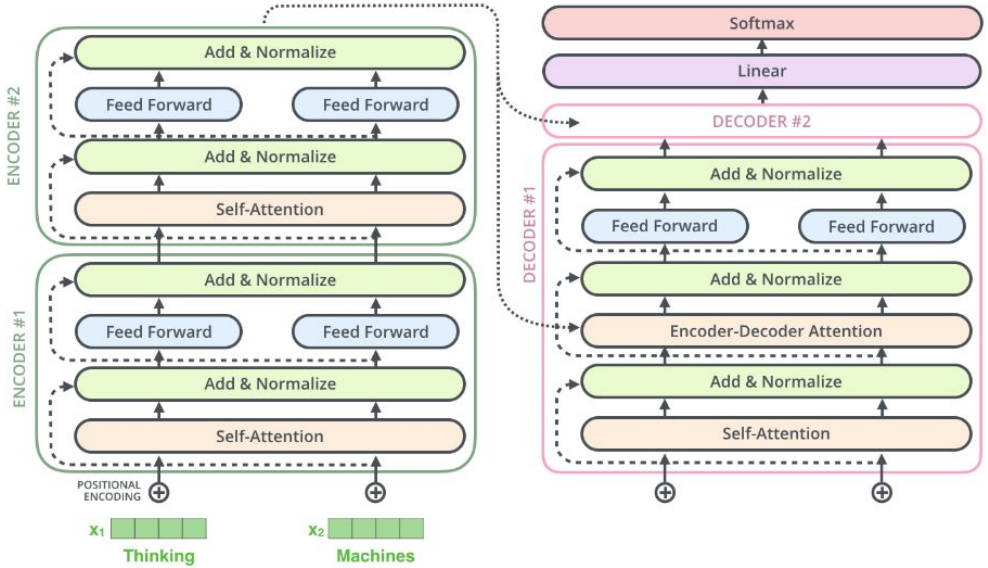
Summary



Attention-based encoder-decoder



Self attentions



Transformers