Attention Models in Deep Learning

By
Chandresh Kumar Maurya
CSE deptt, IIT Indore

https://chandu8542.github.io/

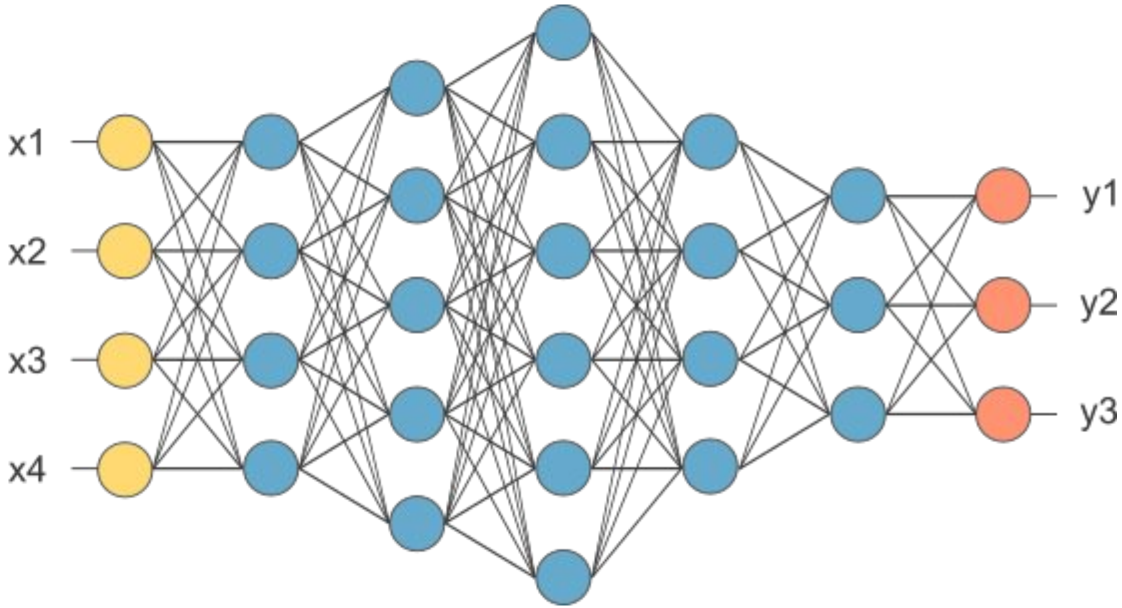
Slides credit

1. Prof. Ming Li, Deep Learning and NLP course, UoW
2. Amaia Salvador, DCU, Dublin
3. Prof. Chris Manning, Stanford NLP course CS 224n

CONTENT

1. Introduction and Motivation
2. Seq-2-seq
3. Attention
4. Transformers

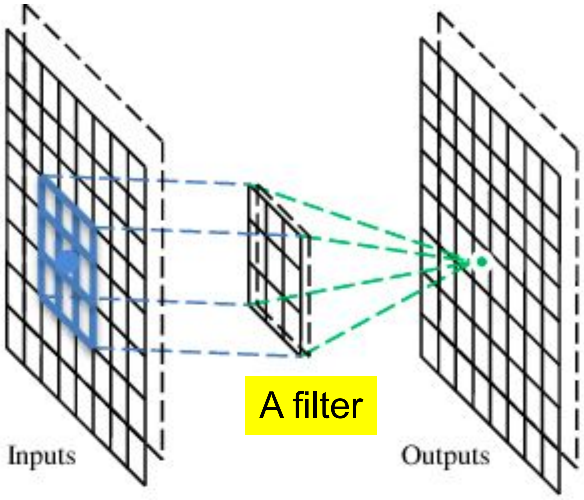
1. Fully connected network, feedforward network



To learn the weights on the edges

2. CNN

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that do convolutional operation.



Convolutional layer

These are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

Convolution Operation

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

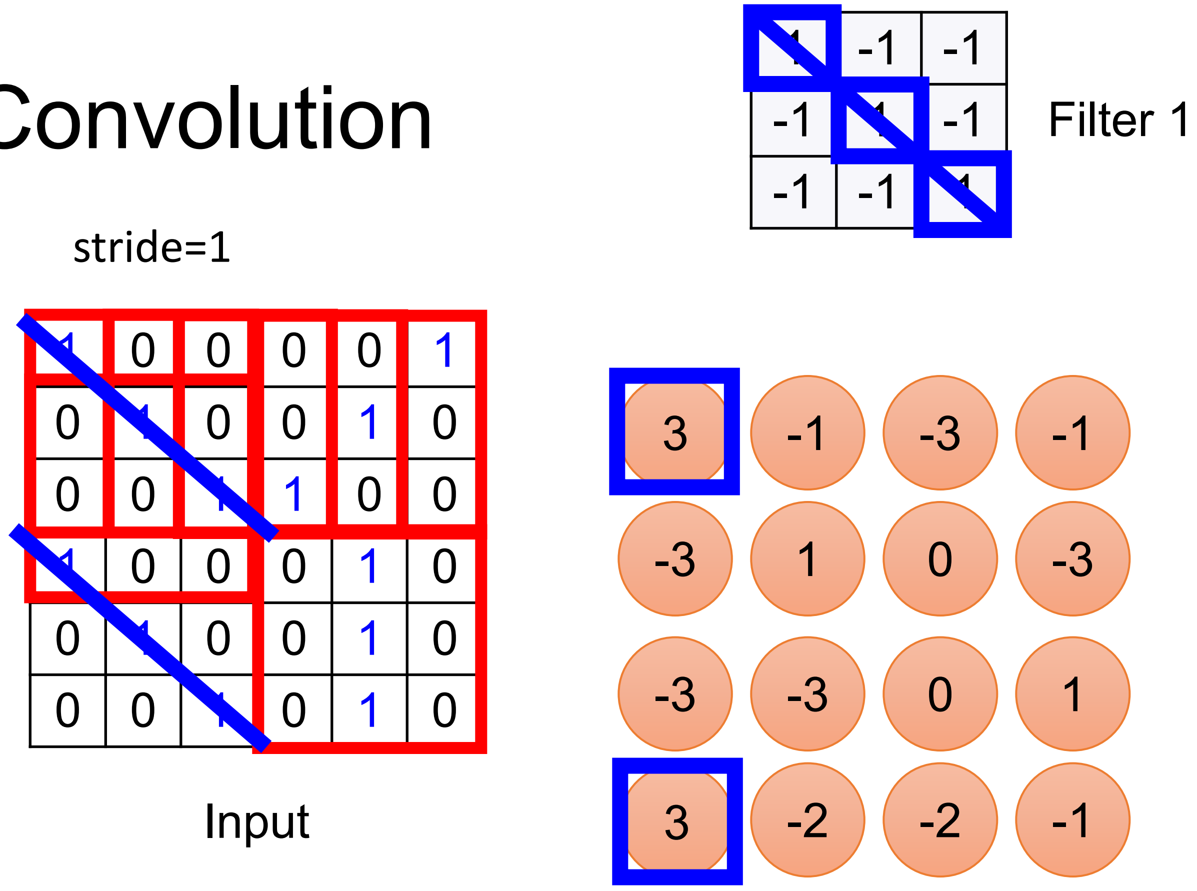
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input

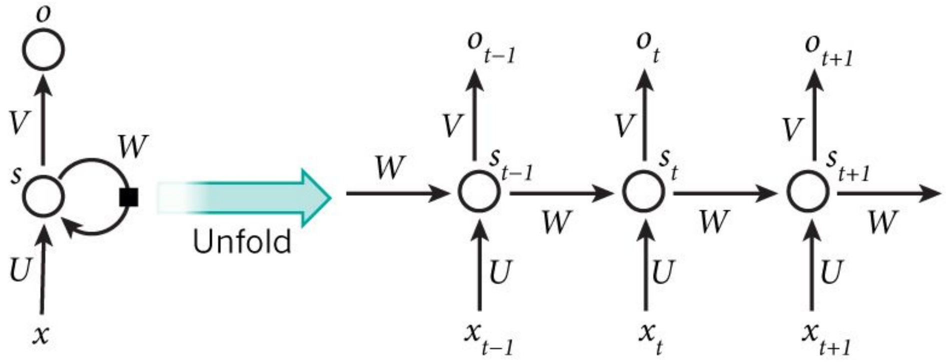
Dot product → 3 -1

Convolution

stride=1

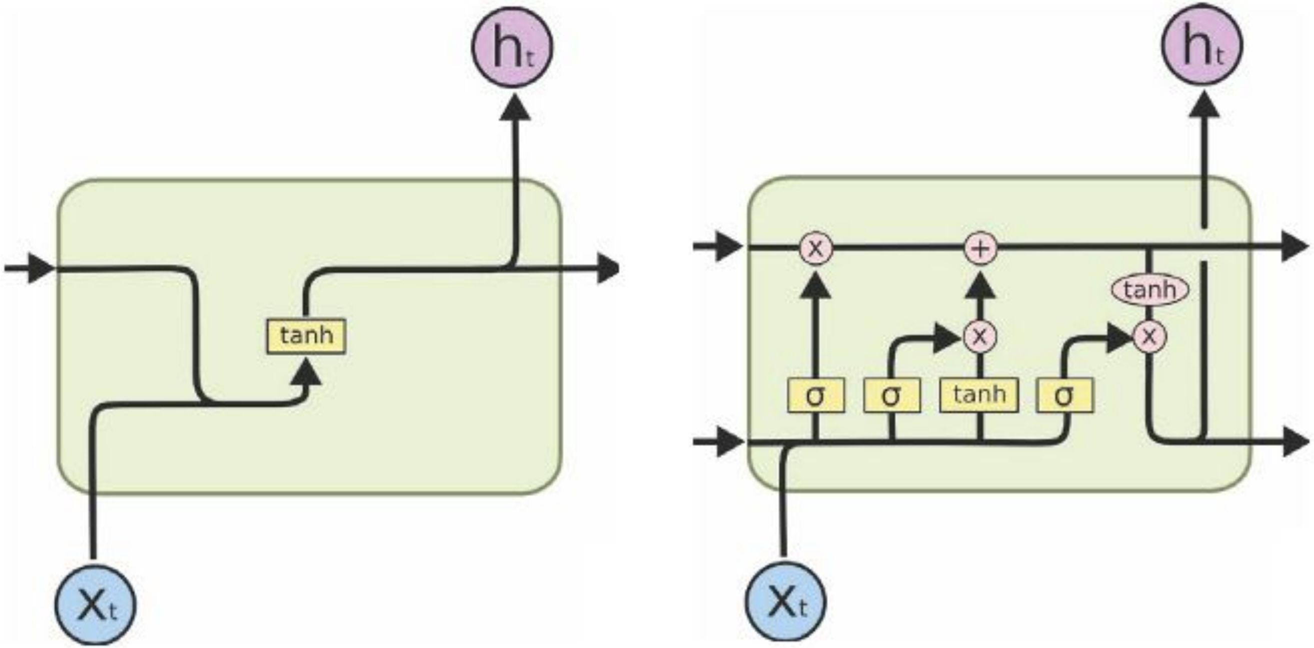


3. RNN



Parameters to be learned:
U, V, W

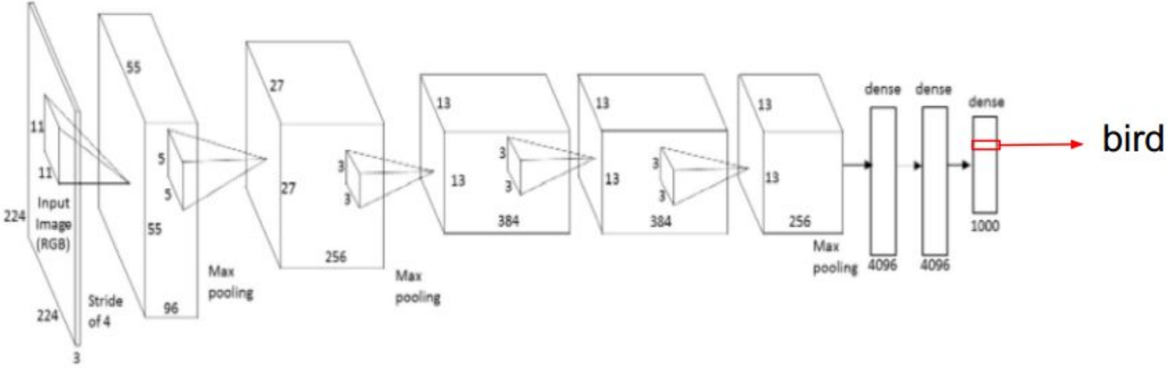
Simple RNN vs LSTM



(a) RNN (b) LSTM

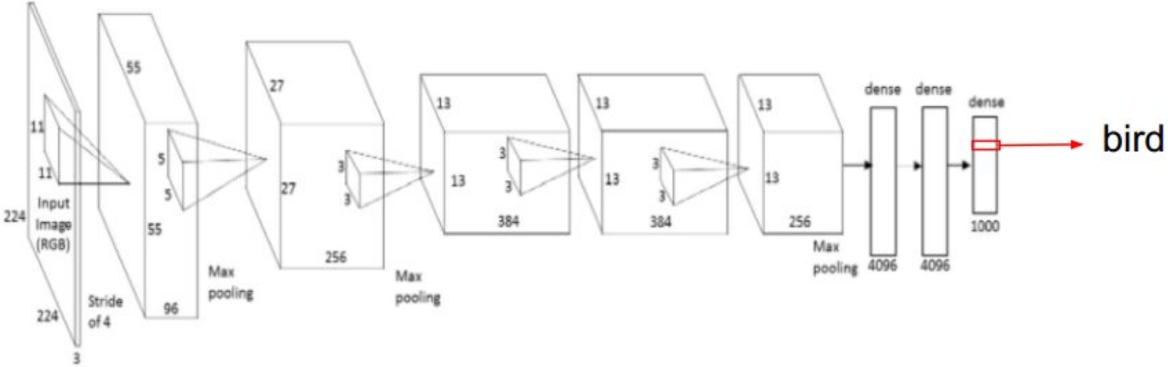
Attention Models: Motivation

Image:
H x W x 3



Attention Models: Motivation

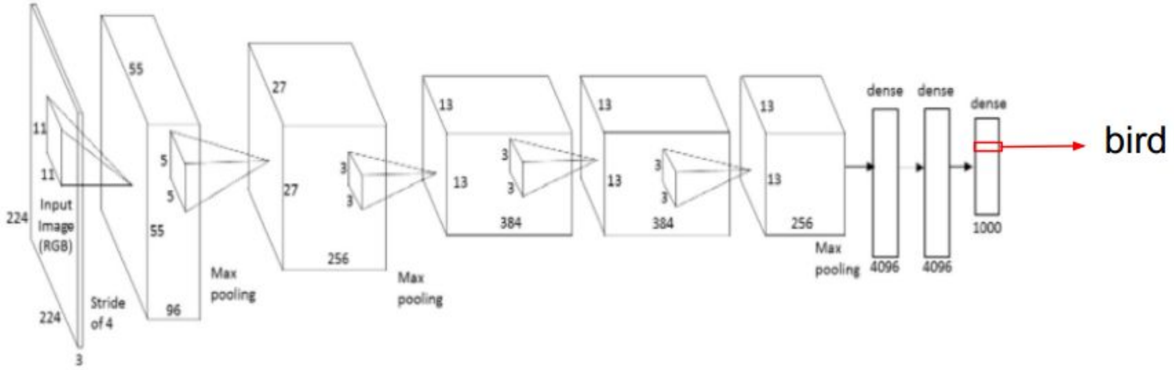
Image:
H x W x 3



What is wrong here?

Attention Models: Motivation

Image:
H x W x 3



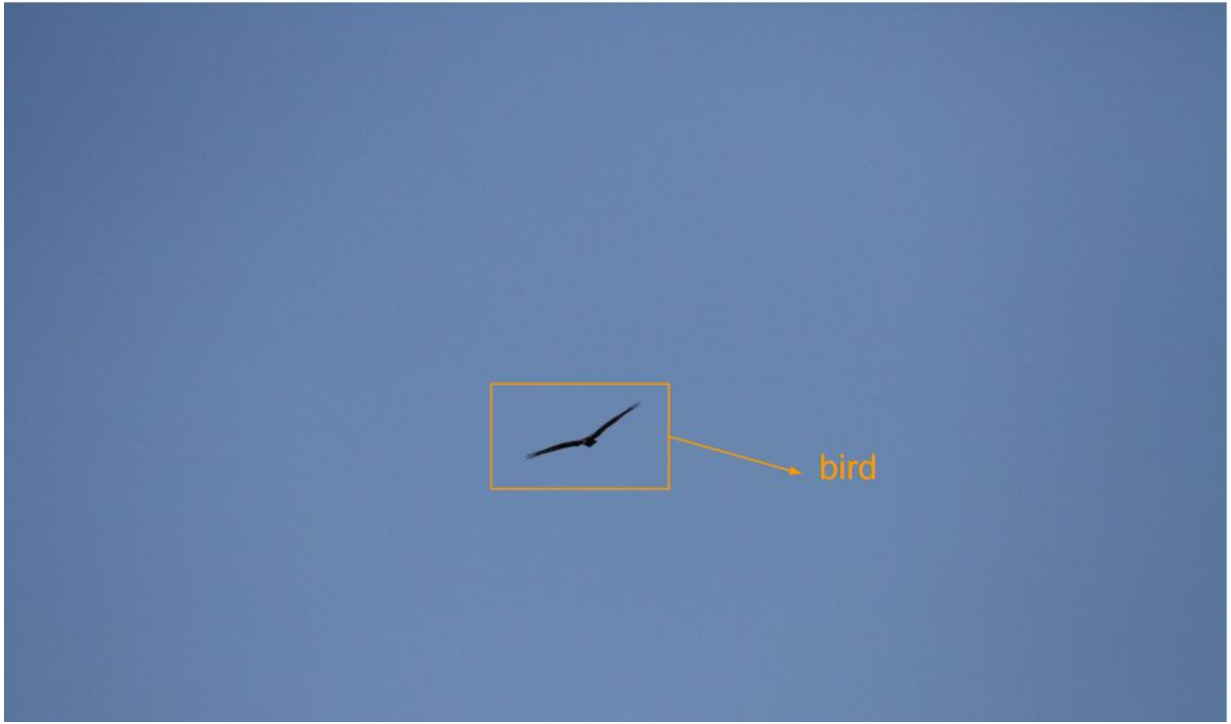
What is wrong here? The whole input volume is used to predict the output… despite the fact that not all pixels are equally important

Attention Models: Motivation

Attention models can
relieve computational burden

Helpful when processing big
images !

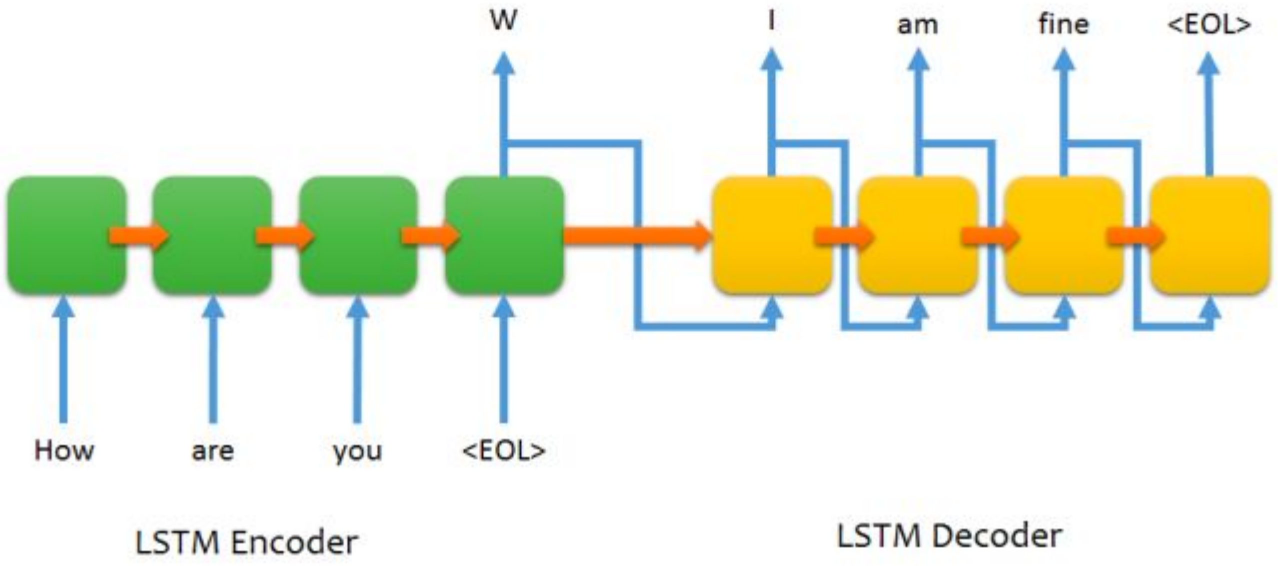
Attention Models: Motivation



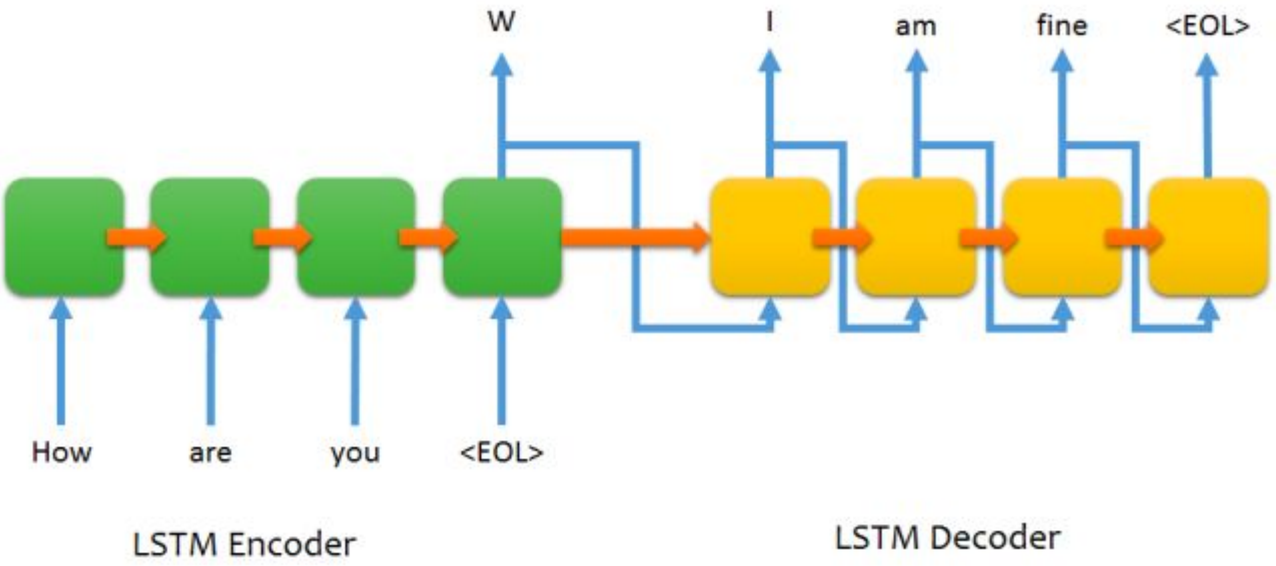
Attention models can relieve computational burden

Helpful when processing big images !

Encoder-Decoder LSTM structure for chatting (for non-intelligent beings)

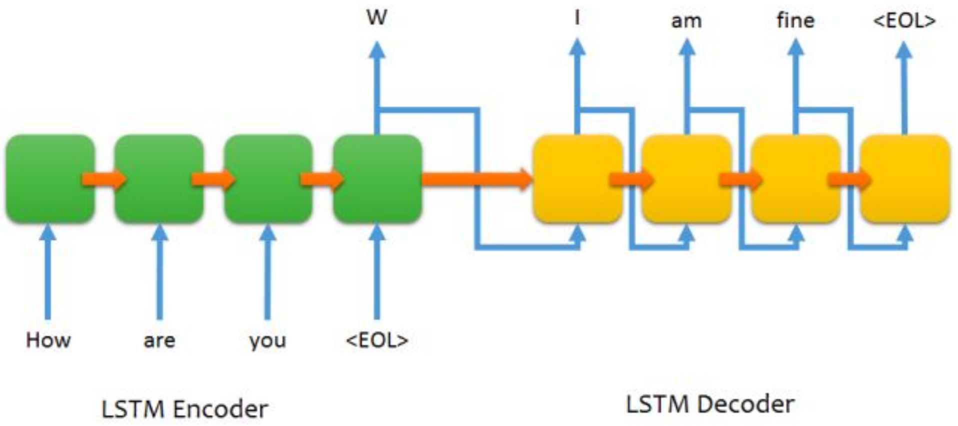


Encoder-Decoder LSTM structure for chatting (for non-intelligent beings)



What is wrong here?

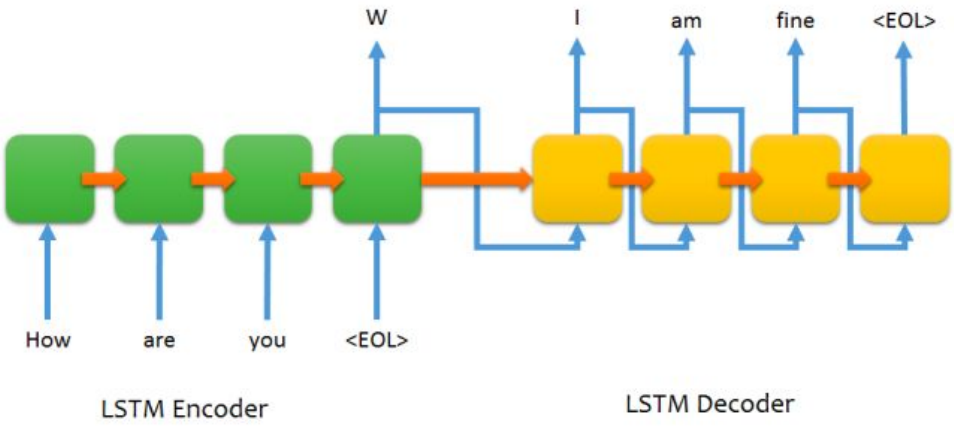
Encoder-Decoder LSTM structure for chatting (for non-intelligent beings)



What is wrong here?

Limtation 1: The whole information is encoded in a fixed-size vector, no matter the length of the input sentence.

Encoder-Decoder LSTM structure for chatting (for non-intelligent beings)



What is wrong here?

Limitation 2: All output predictions are based on the final and static recurrent state of the encoder (hT). No matter the output word being predicted at each time step, all input words are considered in an equal way.

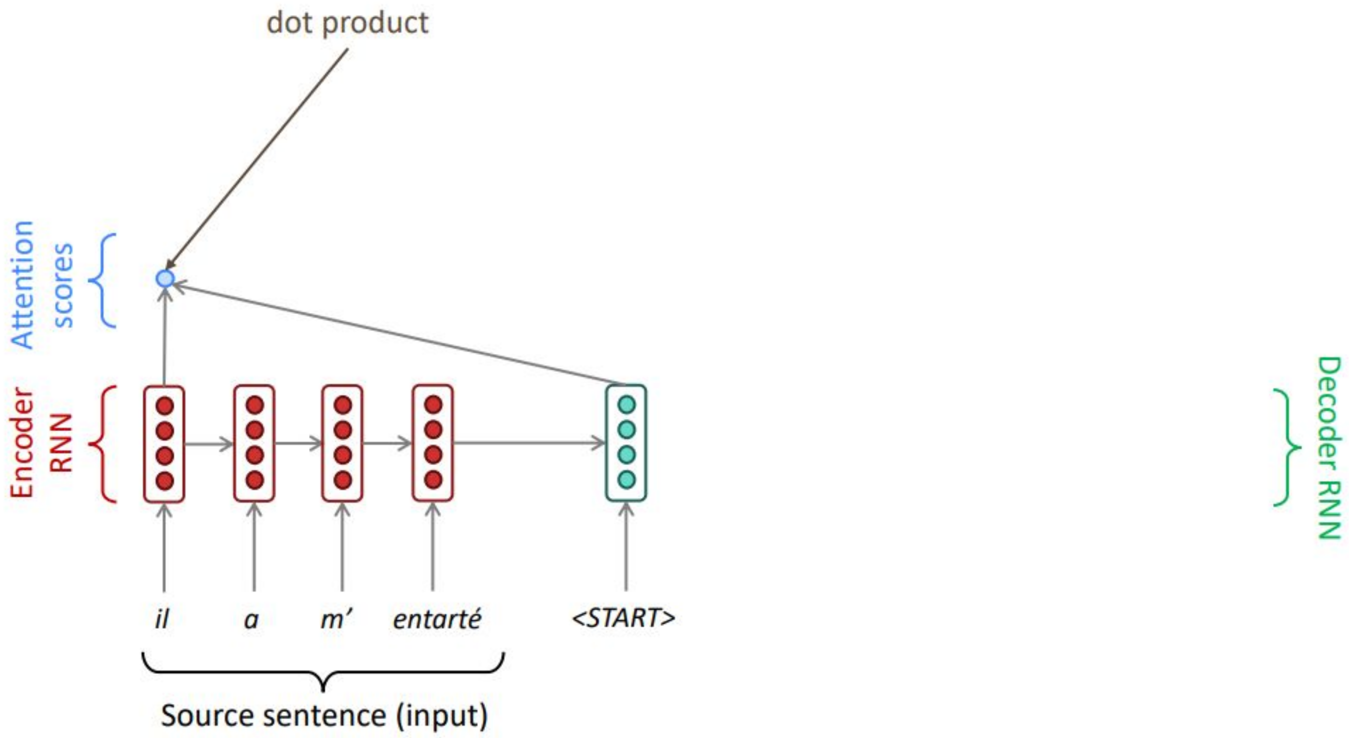
Attention!

- **Attention** provides a solution to the bottleneck problem.

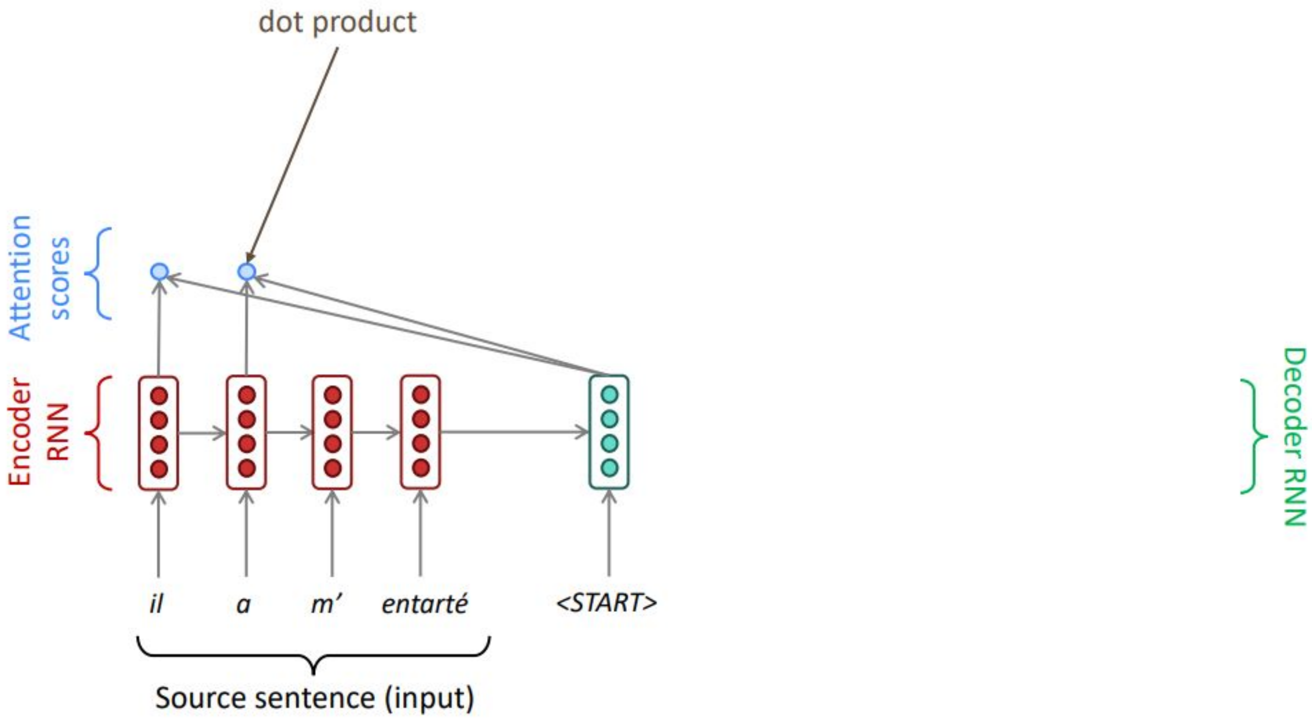
- Core idea: on each step of the decoder, use *direct connection to the encoder* to *focus on a particular part* of the source sequence

- First, we will show via diagram (no equations), then we will show with equations

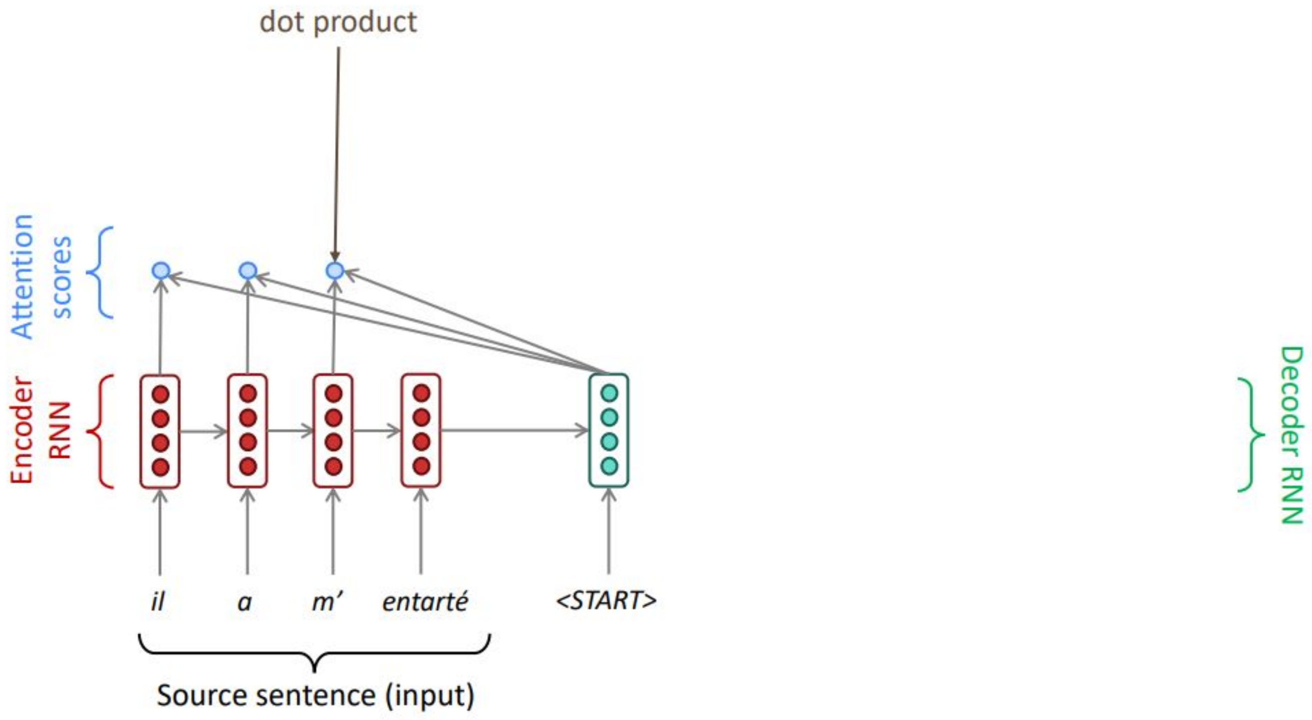
Seq-2-seq with Attention



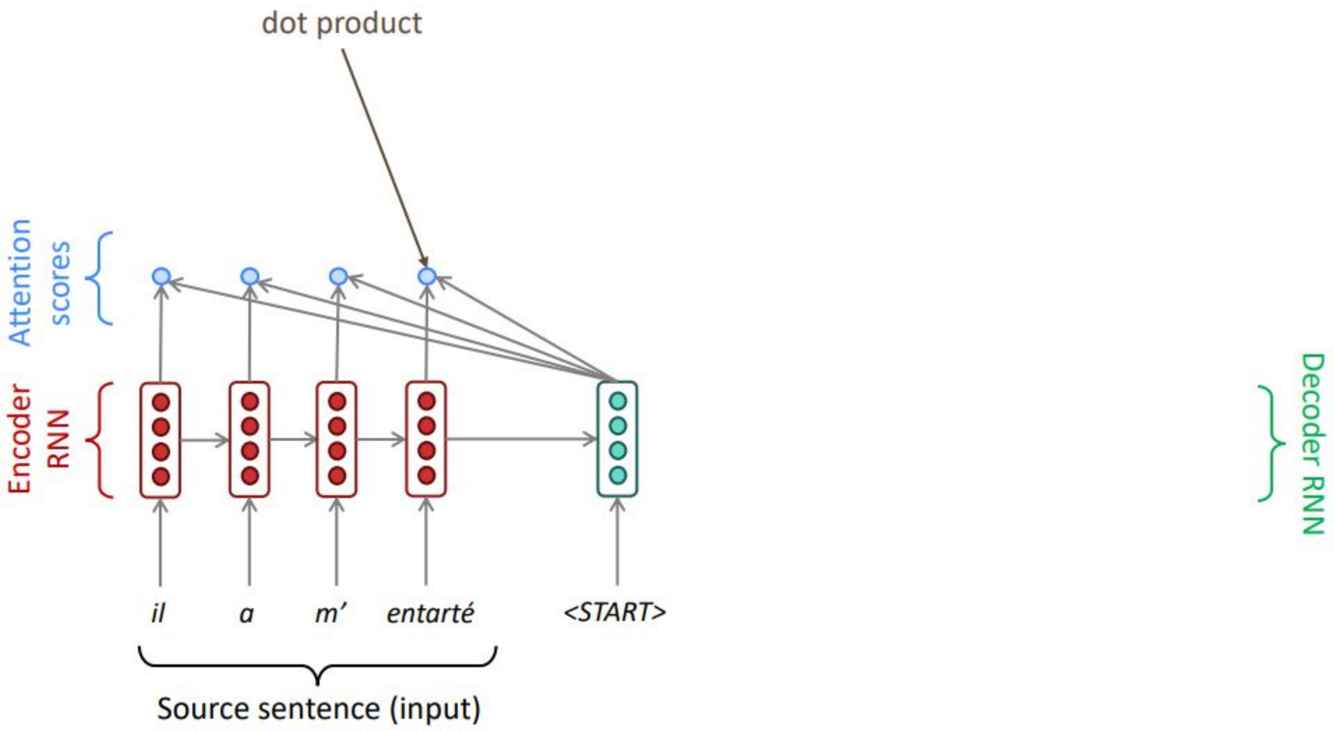
Seq-2-seq with Attention



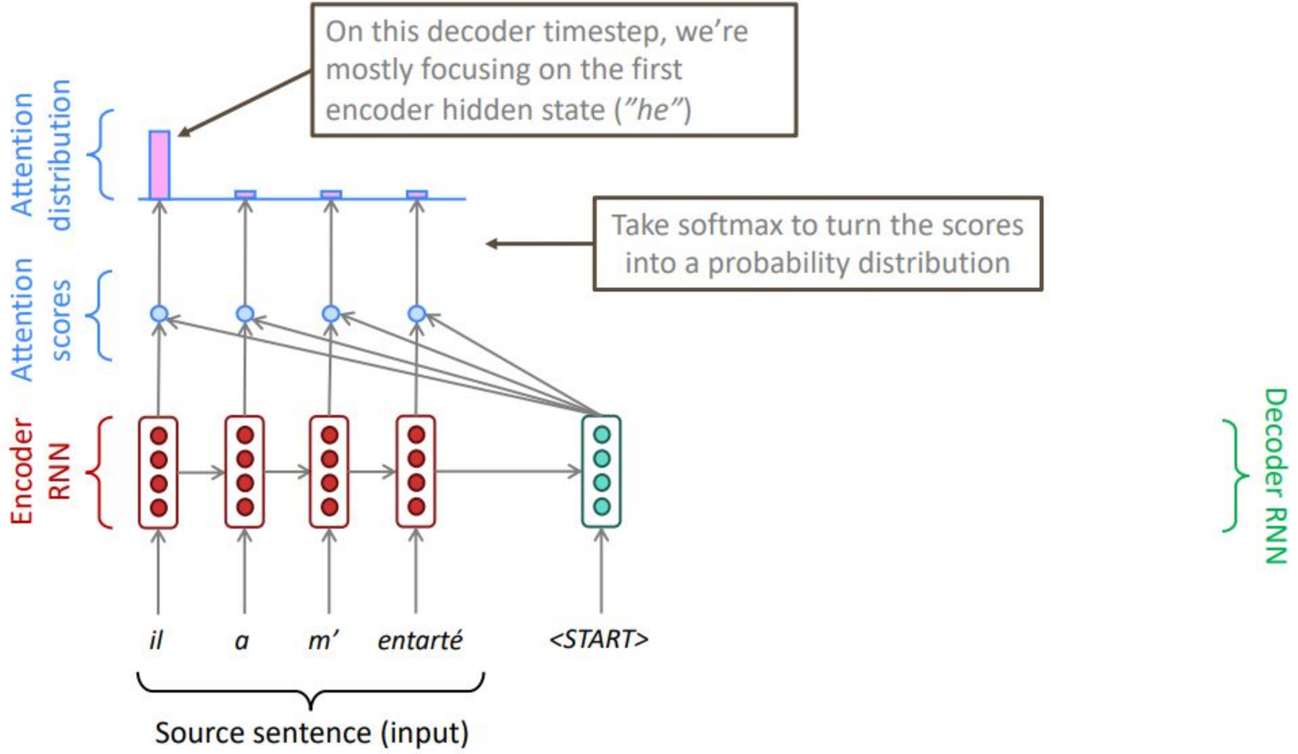
Seq-2-seq with Attention



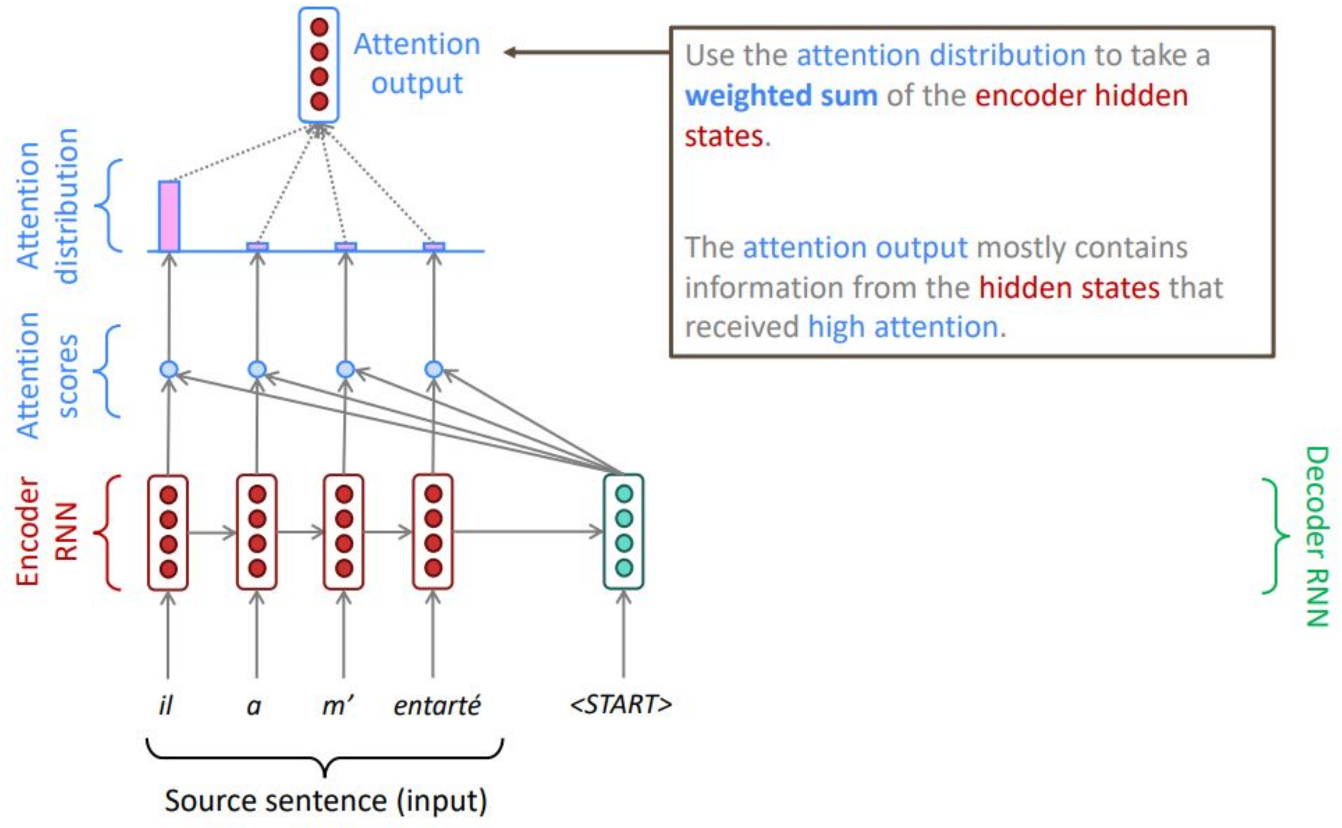
Seq-2-seq with Attention



Seq-2-seq with Attention



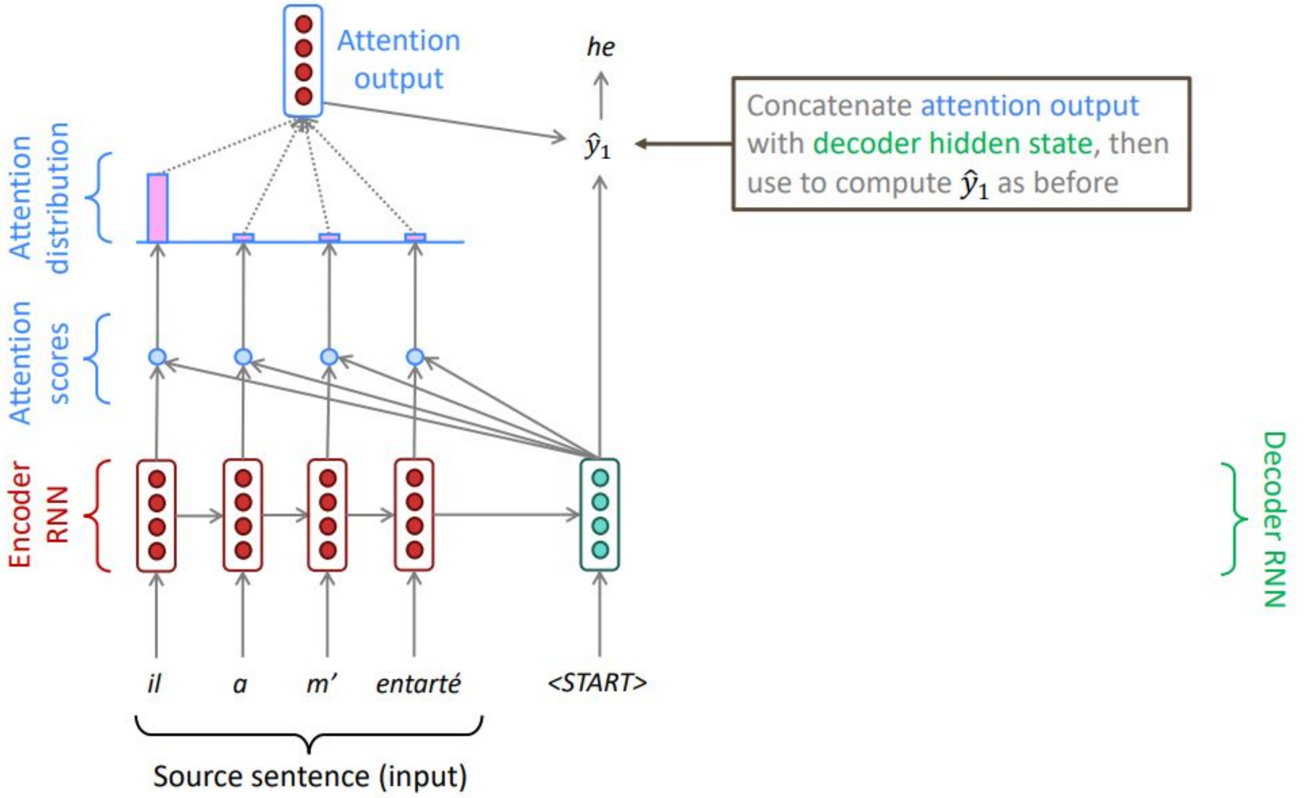
Seq-2-seq with Attention



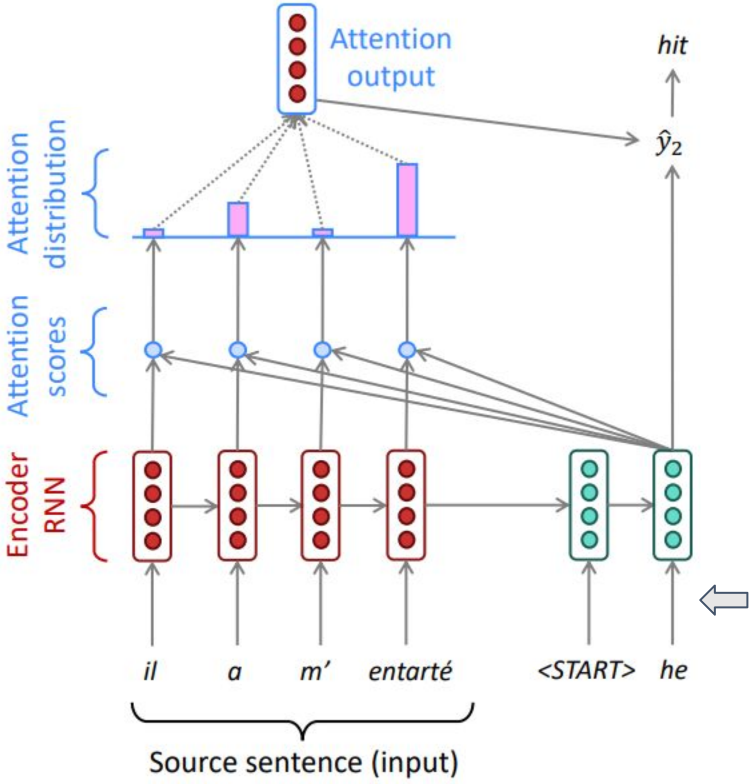
Use the attention distribution to take a **weighted sum** of the encoder hidden states.

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

Seq-2-seq with Attention

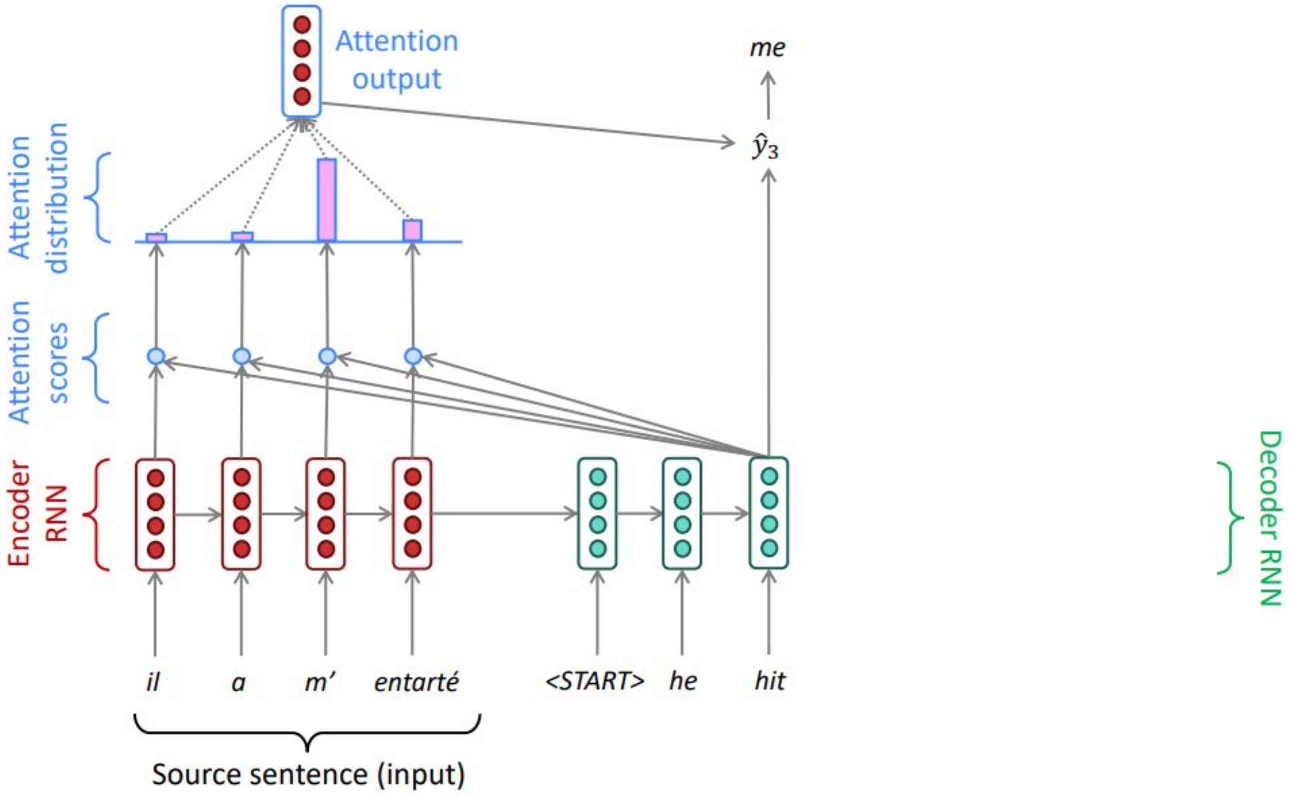


Seq-2-seq with Attention

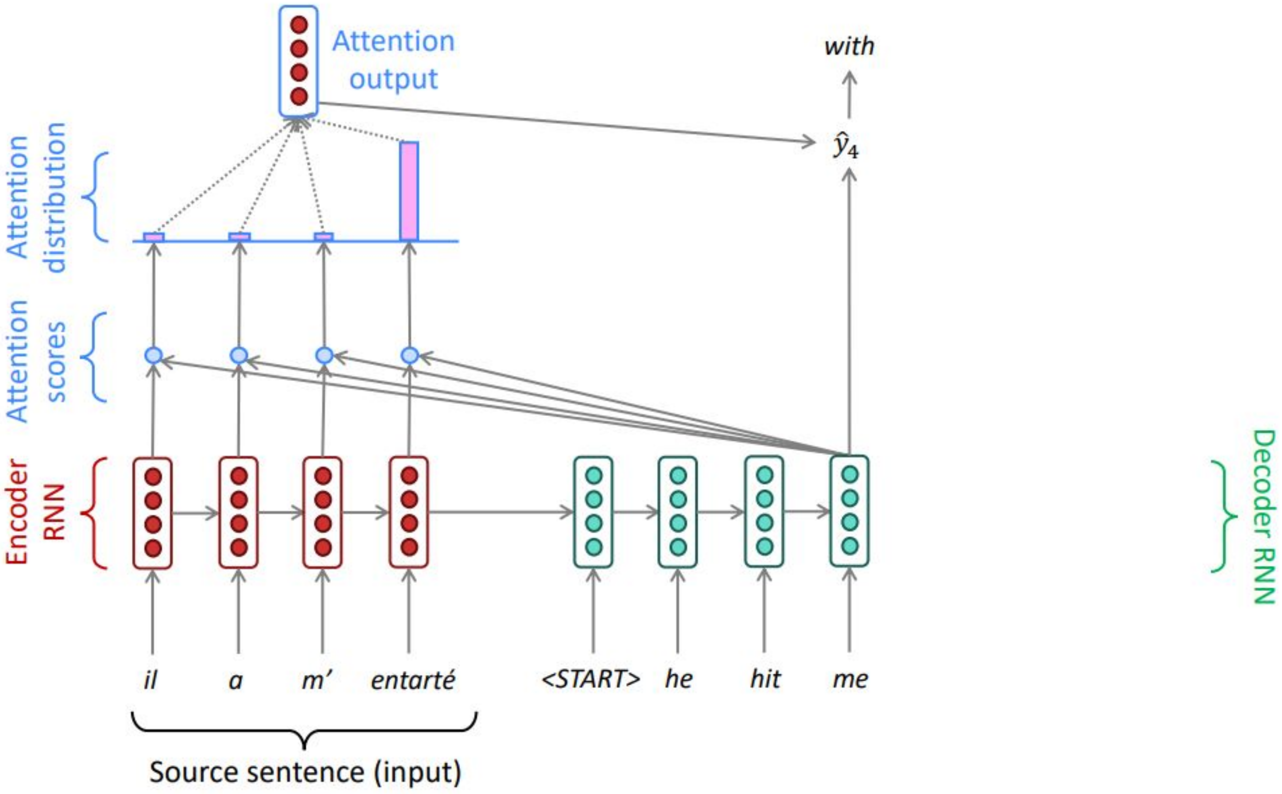


Sometimes we take the attention output from the previous step, and also feed it into the decoder (along with the usual decoder input).

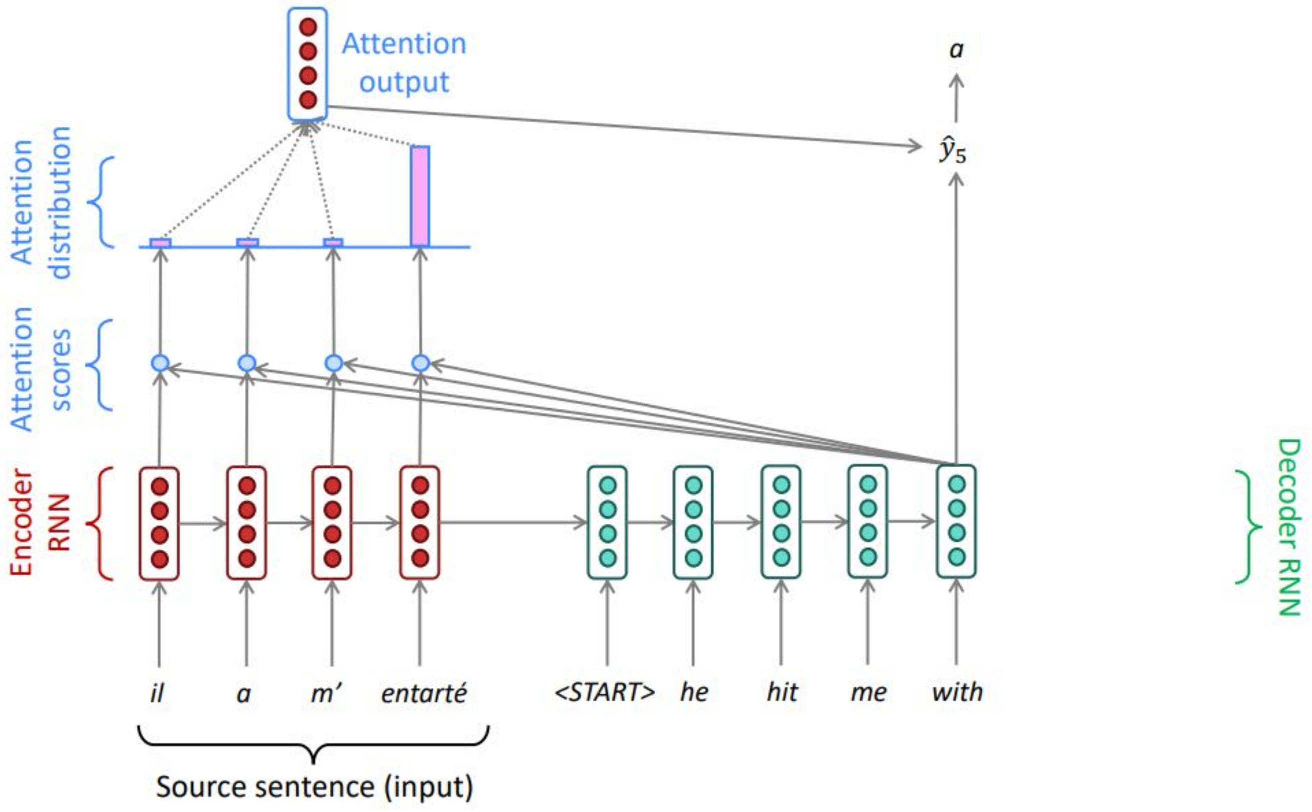
Seq-2-seq with Attention



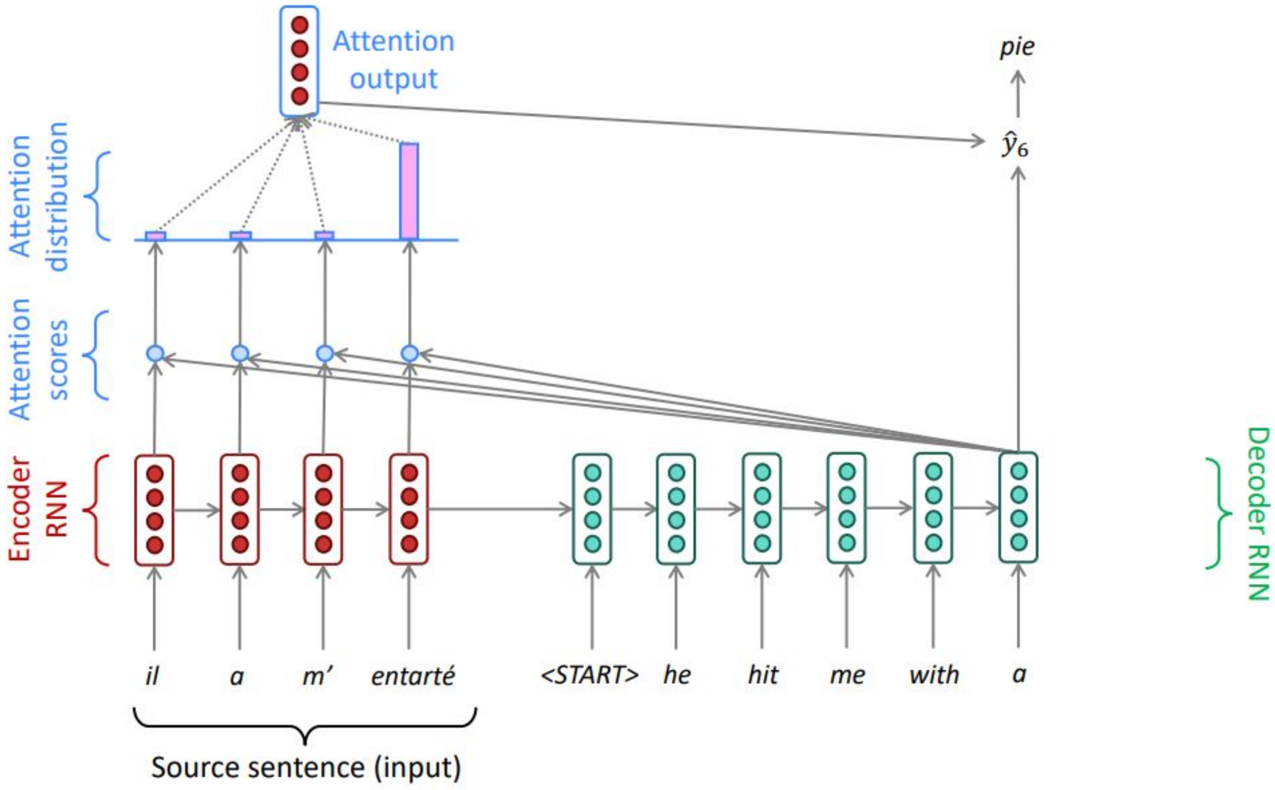
Seq-2-seq with Attention



Seq-2-seq with Attention



Seq-2-seq with Attention



Attention: in equations

- 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 $\boldsymbol{e}^t$ 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)

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

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

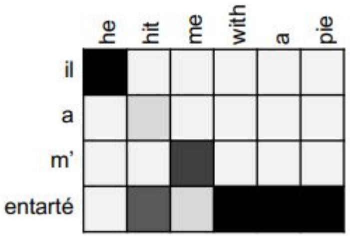
$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

- Finally we concatenate the attention output $\boldsymbol{a}_t$ with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[\boldsymbol{a}_t; \boldsymbol{s}_t] \in \mathbb{R}^{2h}$$

Attention is Great

- Attention significantly improves NMT performance
  - It's very useful to allow decoder to focus on certain parts of the source
- Attention solves the bottleneck problem
  - Attention allows decoder to look directly at source; bypass bottleneck
- Attention helps with vanishing gradient problem
  - Provides shortcut to faraway states
- Attention provides some interpretability
  - By inspecting attention distribution, we can see what the decoder was focusing on
  - We get (soft) alignment for free!
  - This is cool because we never explicitly trained an alignment system
  - The network just learned alignment by itself



Attention is a general Deep Learning technique

- We've seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- <u>However</u>: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)

- **<u>More general definition of attention</u>**:
  - Given a set of vector *values*, and a vector *query*, **<u>attention</u>** is a technique to compute a weighted sum of the values, dependent on the query.

- We sometimes say that the query *attends to* the values.
- For example, in the seq2seq + attention model, each decoder hidden state (query) *attends to* all the encoder hidden states (values).

Attention is a general Deep Learning technique

More general definition of attention:

Given a set of vector *values*, and a vector *query*, **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).

There are several attention variants

- We have some *values* $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and a *query* $\boldsymbol{s} \in \mathbb{R}^{d_2}$

- Attention always involves:
  1. Computing the *attention scores* $\boldsymbol{e} \in \mathbb{R}^N$ ← There are multiple ways to do this
  2. Taking softmax to get *attention distribution* α:

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

  3. Using attention distribution to take weighted sum of values:

$$\boldsymbol{a} = \sum_{i=1}^{N} \alpha_i \boldsymbol{h}_i \in \mathbb{R}^{d_1}$$

  thus obtaining the *attention output* ***a*** (sometimes called the *context vector*)

There are several attention variants

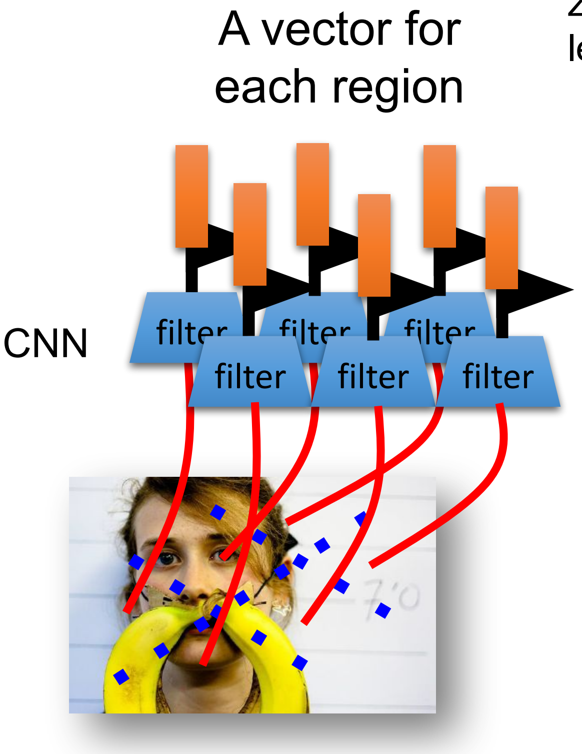
There are several ways you can compute $\boldsymbol{e} \in \mathbb{R}^N$ from $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N \in \mathbb{R}^{d_1}$ and $\boldsymbol{s} \in \mathbb{R}^{d_2}$:

- 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

- 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

Image caption generation using attention



z^0 is initial parameter, it is also learned

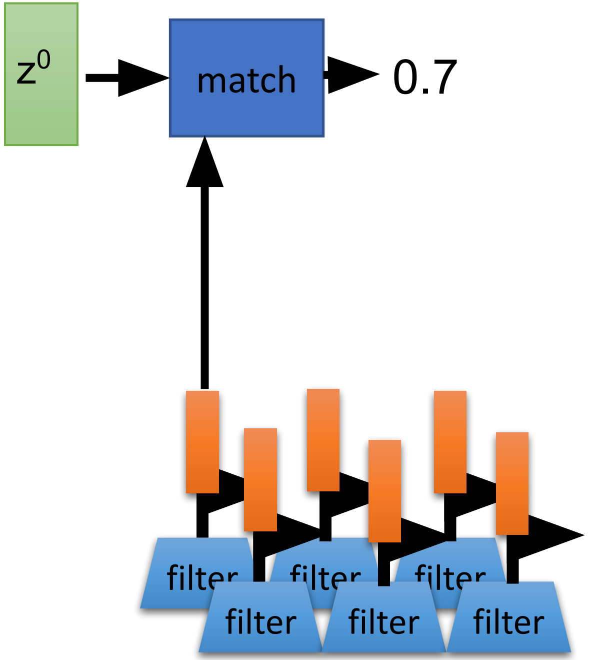


Image caption generation using attention

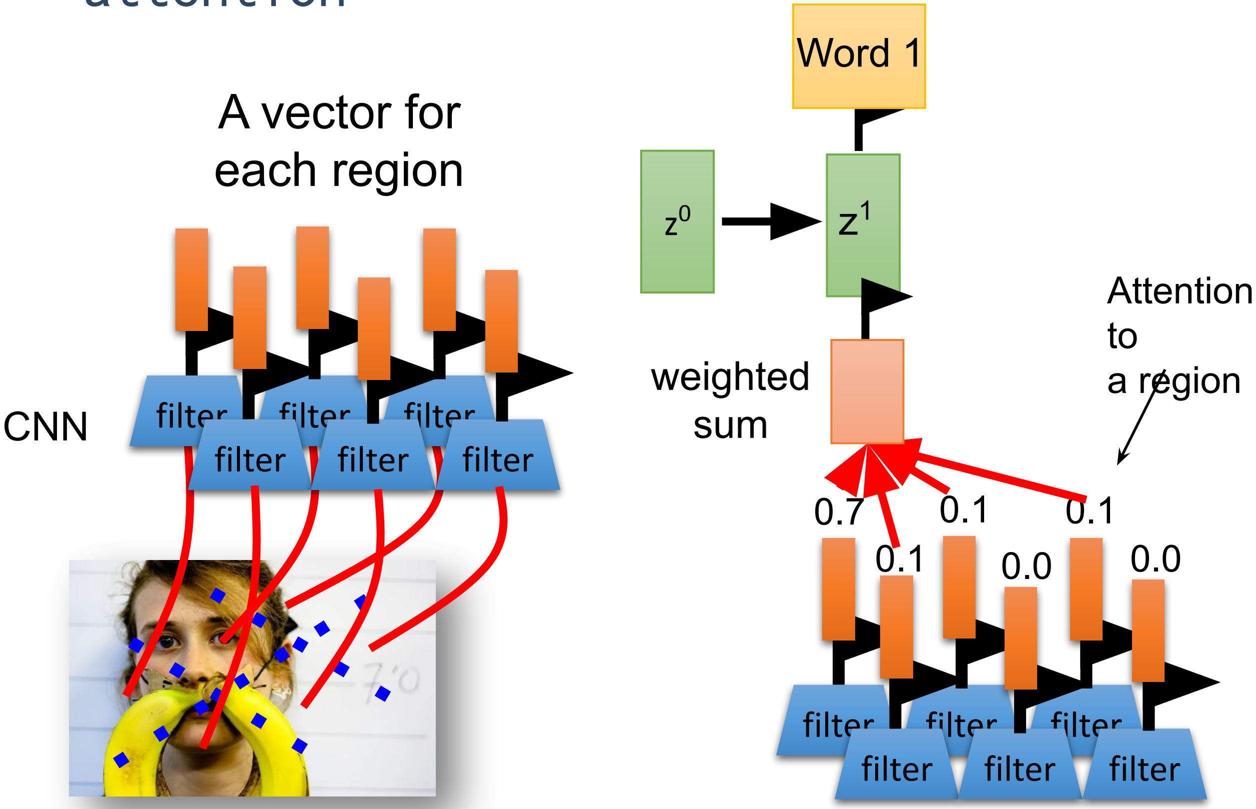
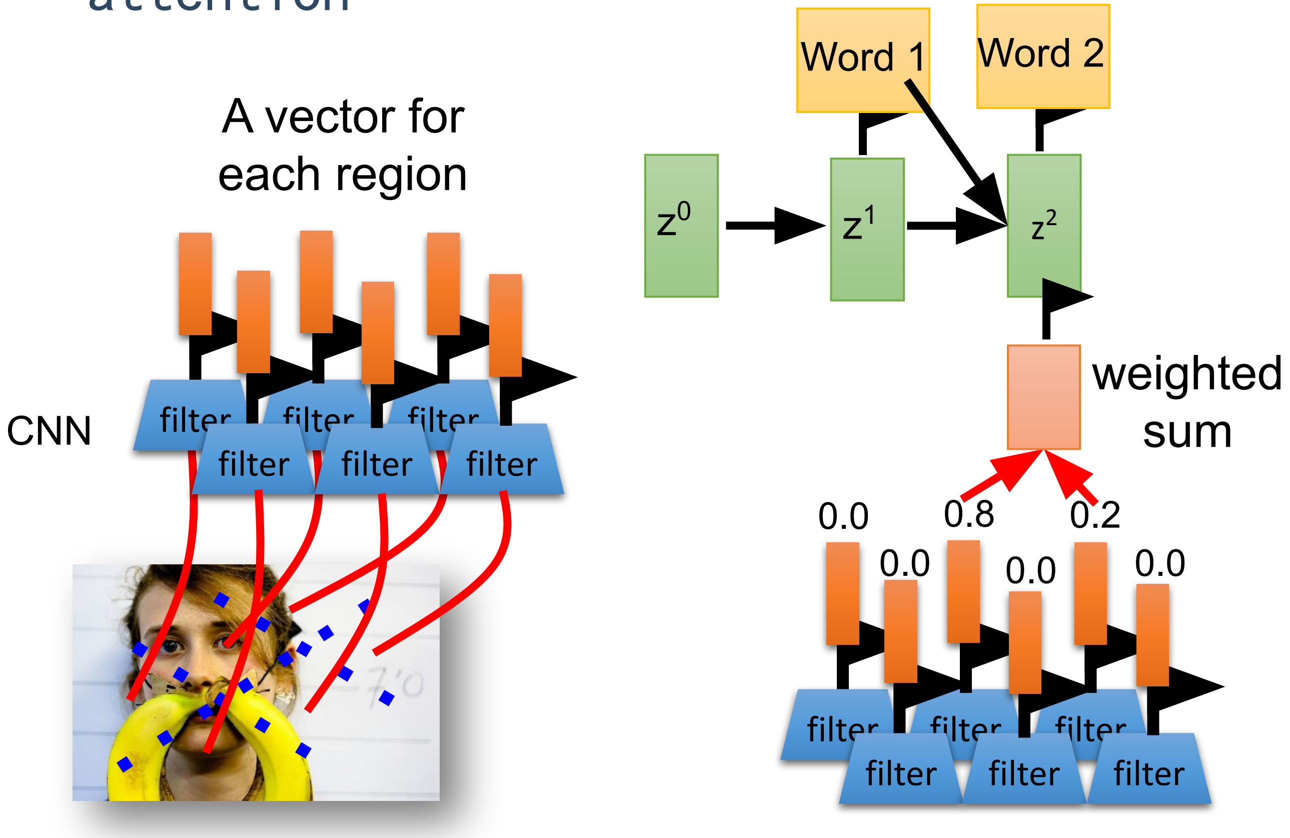


Image caption generation using attention



Compute alignments scores (scalars):

$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$

$f_{att}(.)$ is an MLP

Normalize to get attention weights:

$$\boxed{a_{t,:,:} = softmax(e_{t,:,:})}$$

$0 < a_{t,i,j} < 1$,
attention values sum to 1

Image caption generation using attention

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, Yoshua Bengio,
"Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML, 2015

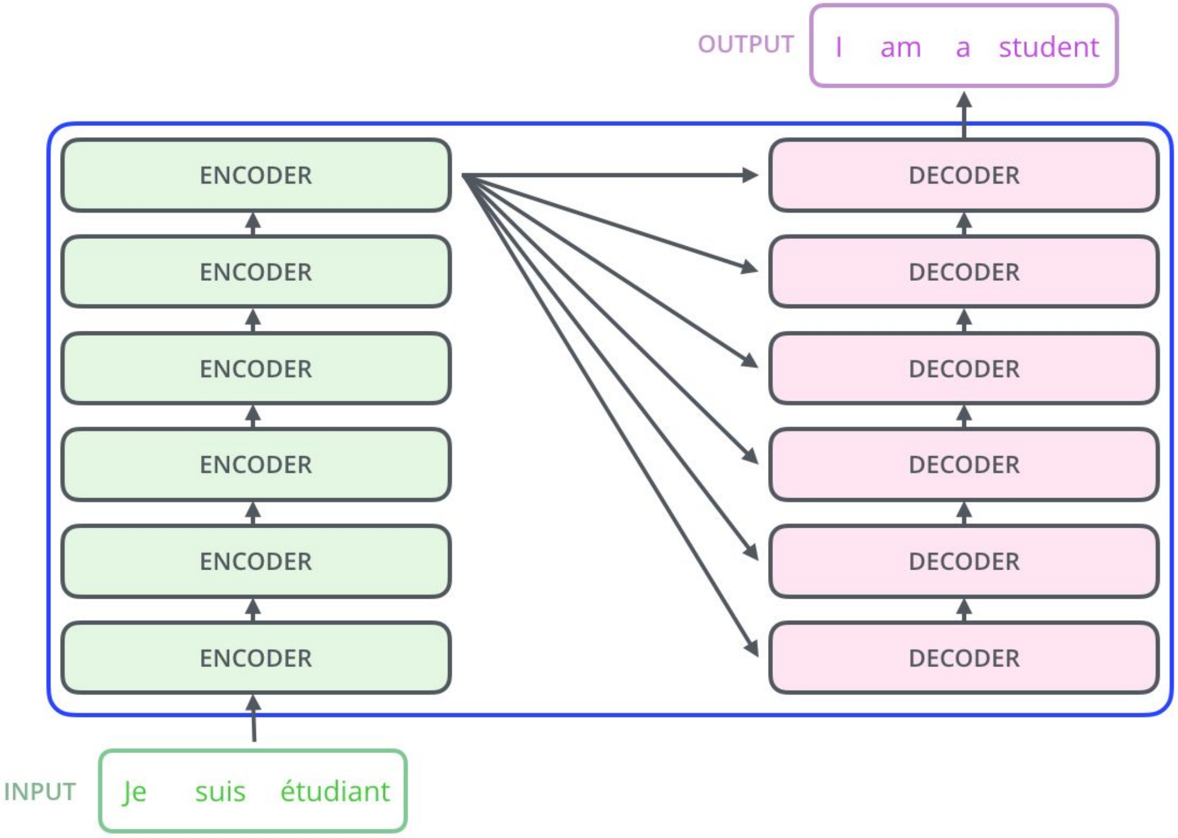
Attention and Transformers

Many new ideas

1. ULM-FiT, pre-training, transfer learning in NLP
2. Recurrent models require linear sequential computation, hard to parallelize. ELMo, bidirectional LSTM.
3. In order to reduce such sequential computation, several models based on CNN are introduced, such as ConvS2S and ByteNet. Dependency for ConvS2S needs linear depth, and ByteNet logarithmic.
4. The transformer is the first transduction model relying entirely on self-attention to compute the representations of its input and output without using RNN or CNN.

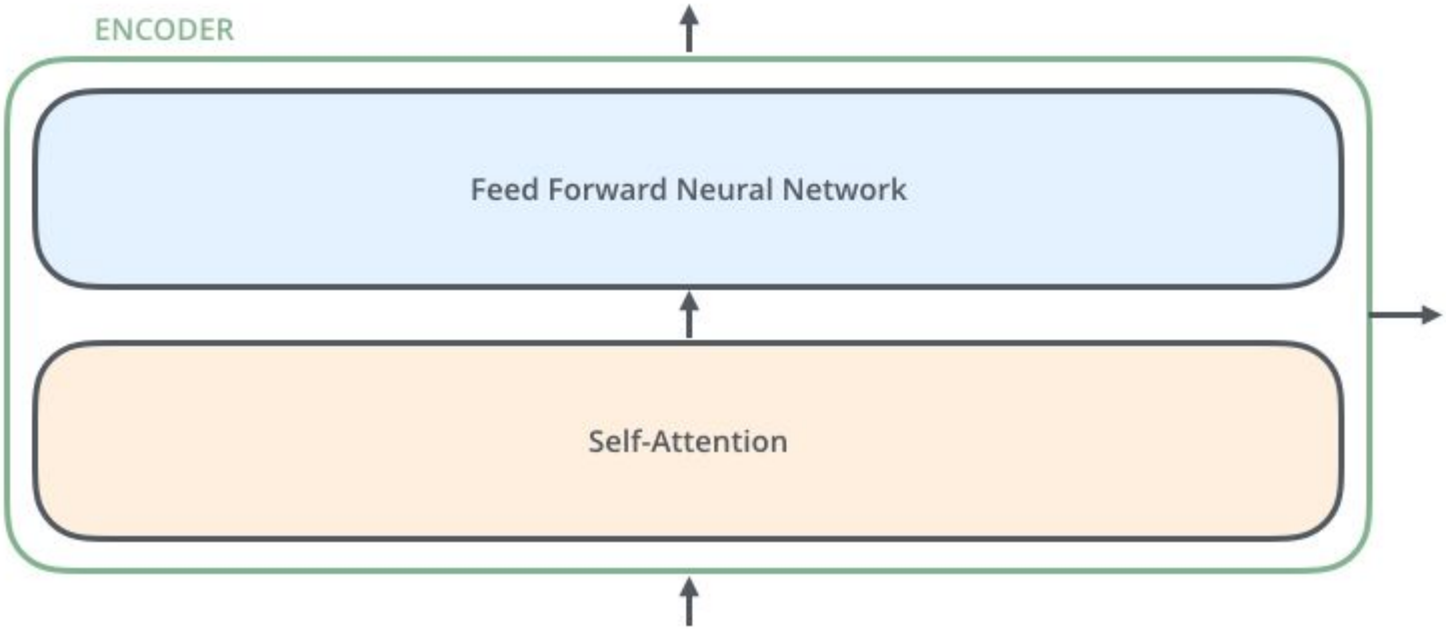
Attention and Transformers

Transformer



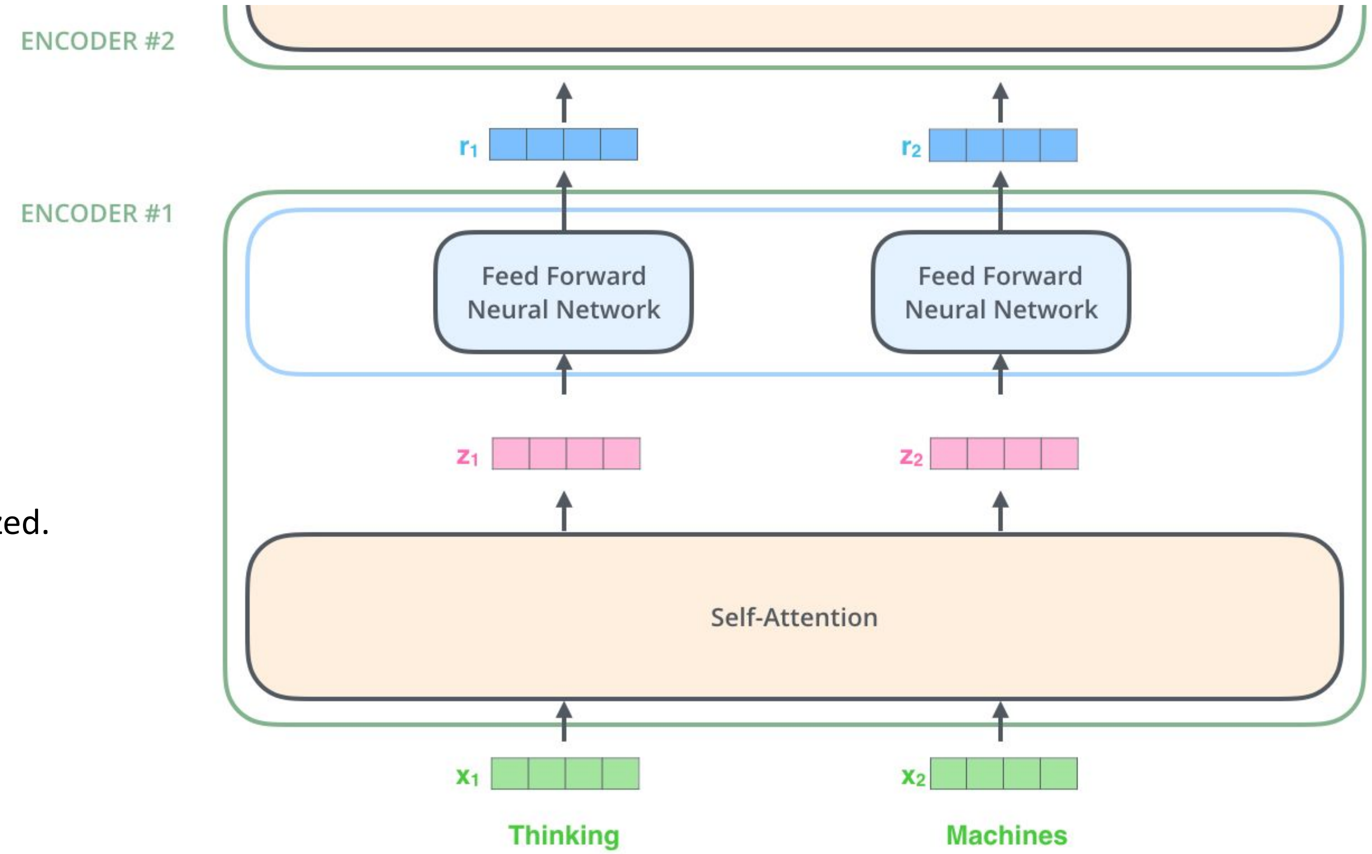
Attention and Transformers

An Encoder Block: same structure, different parameters



Attention and Transformers

Encoder

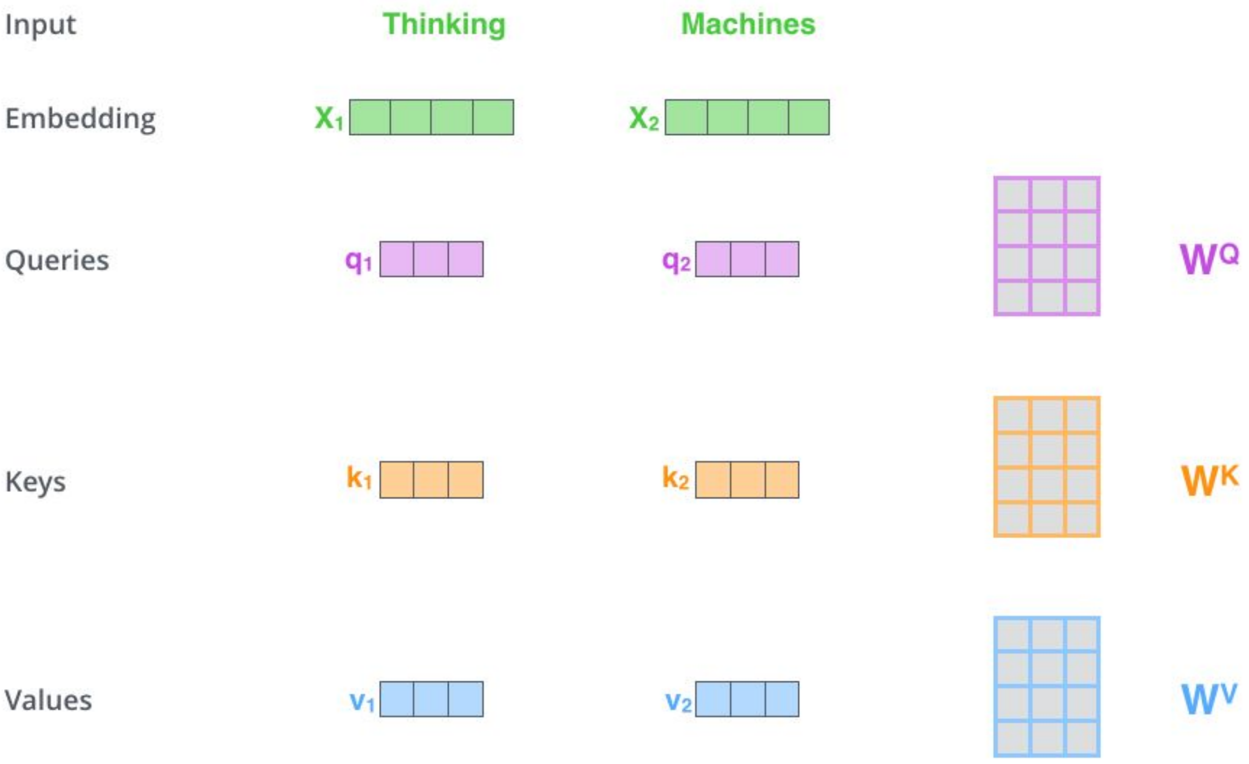


Note: The ffnn is independent for each word. Hence can be parallelized.

Attention and Transformers

Self

Attention



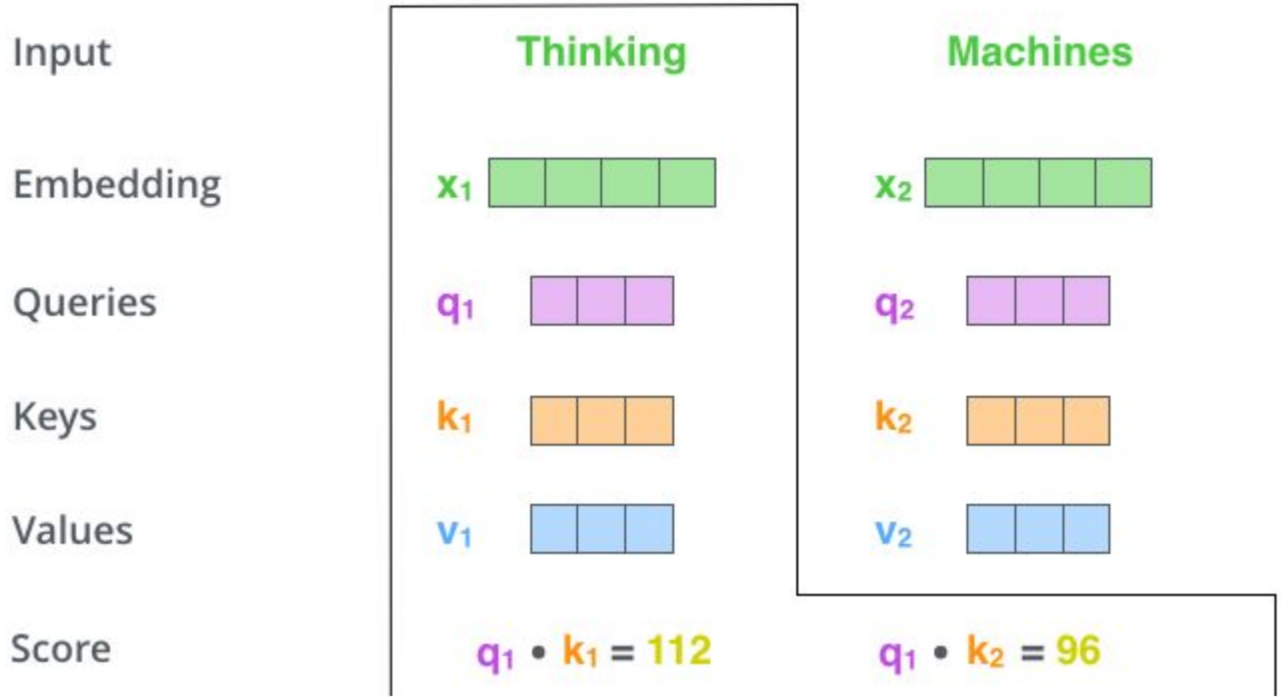
First we create three vectors by multiplying input embedding (1x512) x_i with three matrices (64x512):

$q_i = x_i W^Q$
$K_i = x_i W^K$
$V_i = x_i W^V$

Attention and Transformers

Self

Attention



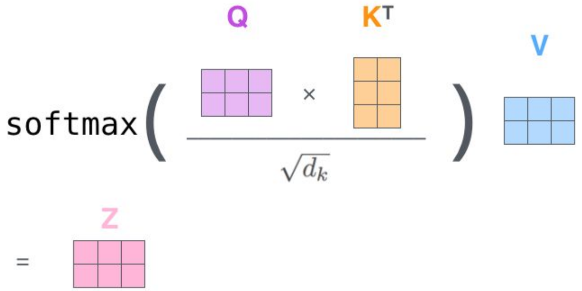
Now we need to calculate a score to determine how much focus to place on other
Parts of the input.

Attention and Transformers

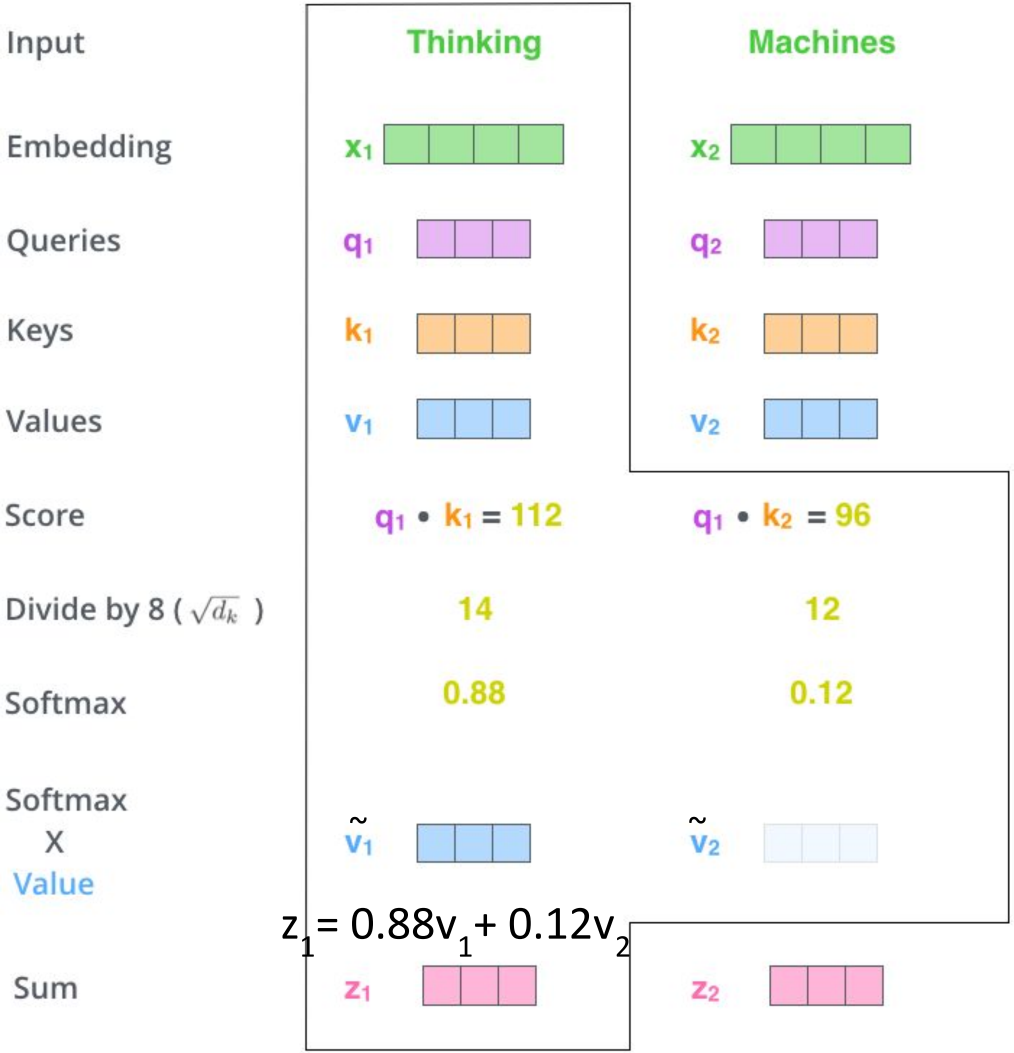
Self

Attention

Formula



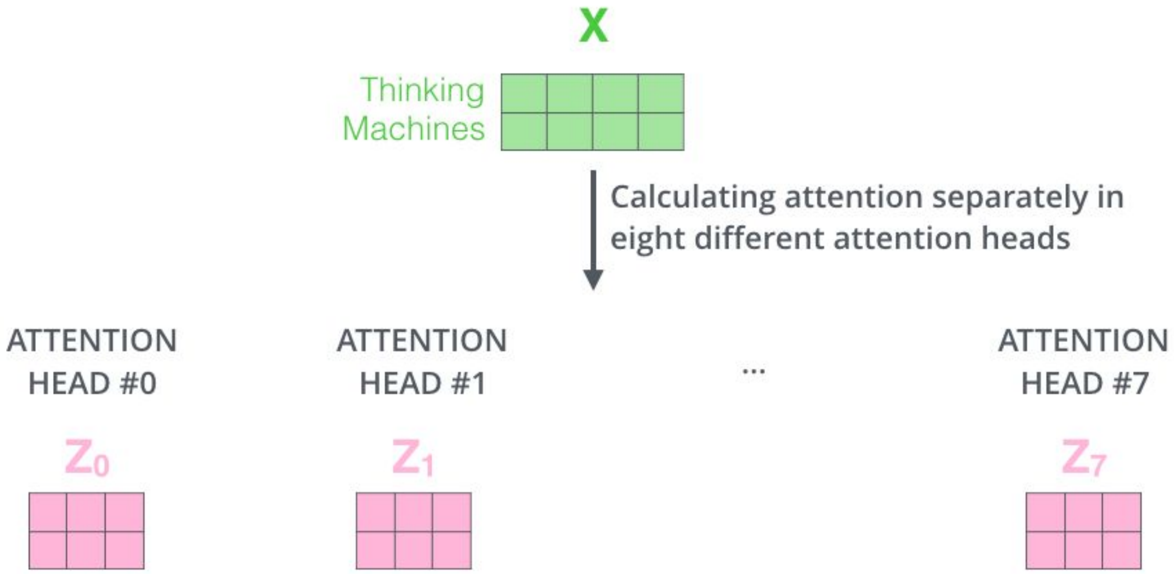
d_k=64 is dimension of key vector



Attention and Transformers

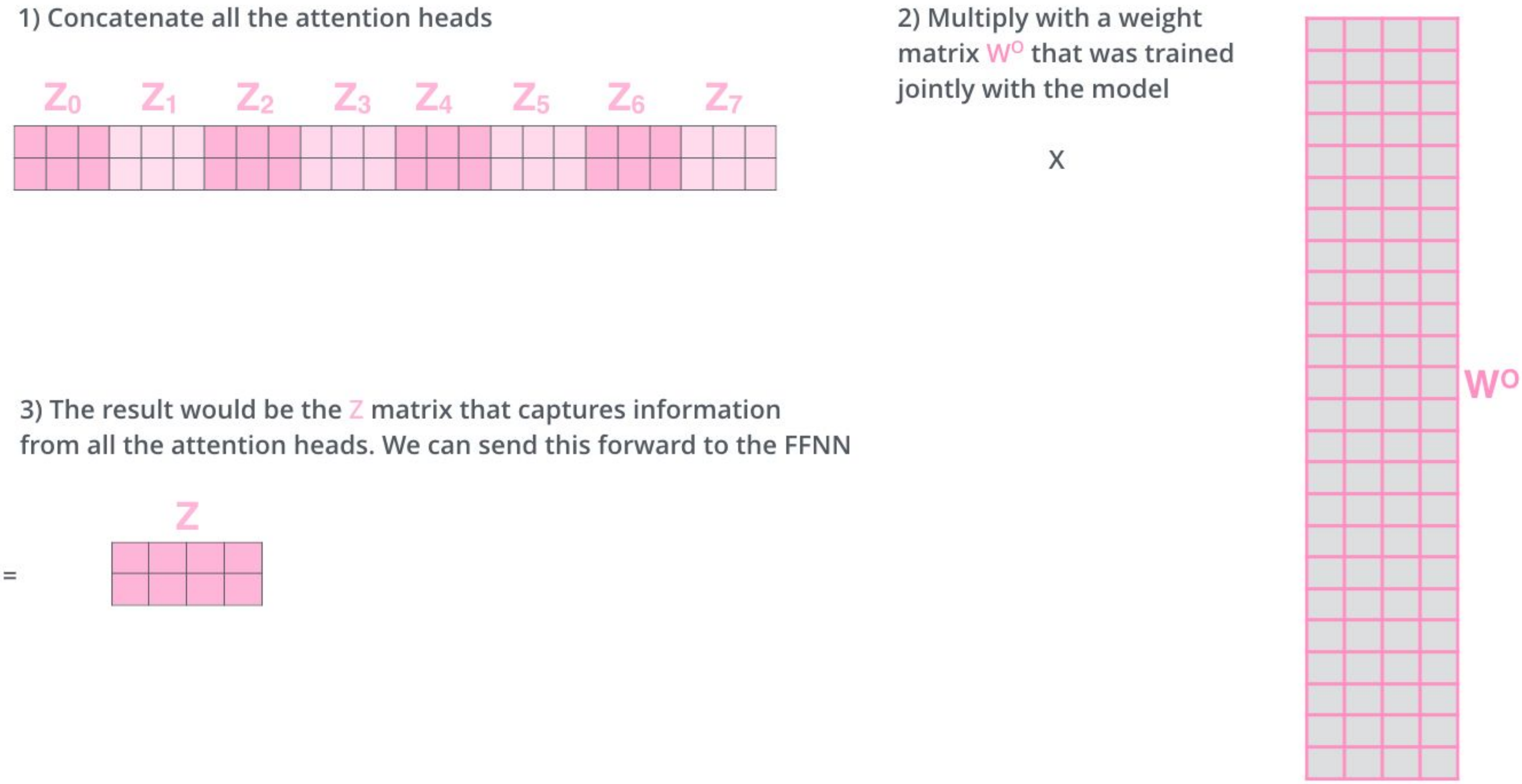
Multiple heads

1. It expands the model's ability to focus on different positions.
2. It gives the attention layer multiple "representation subspaces"



Attention and Transformers

The output is expecting only a 2x4 (|input|x64) matrix, hence,

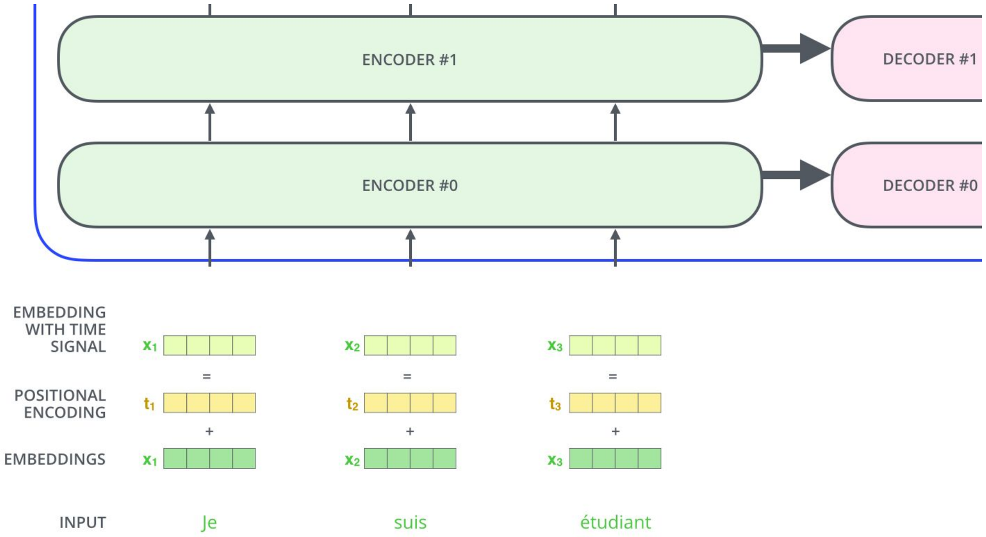


Attention and Transformers

Representing the input order (positional encoding)

The transformer adds a vector to each input embedding. These vectors follow a specific pattern that the model learns, which helps it determine the position of each word, or the distance between different words in the sequence. The intuition here is that adding these values to the embeddings provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention.

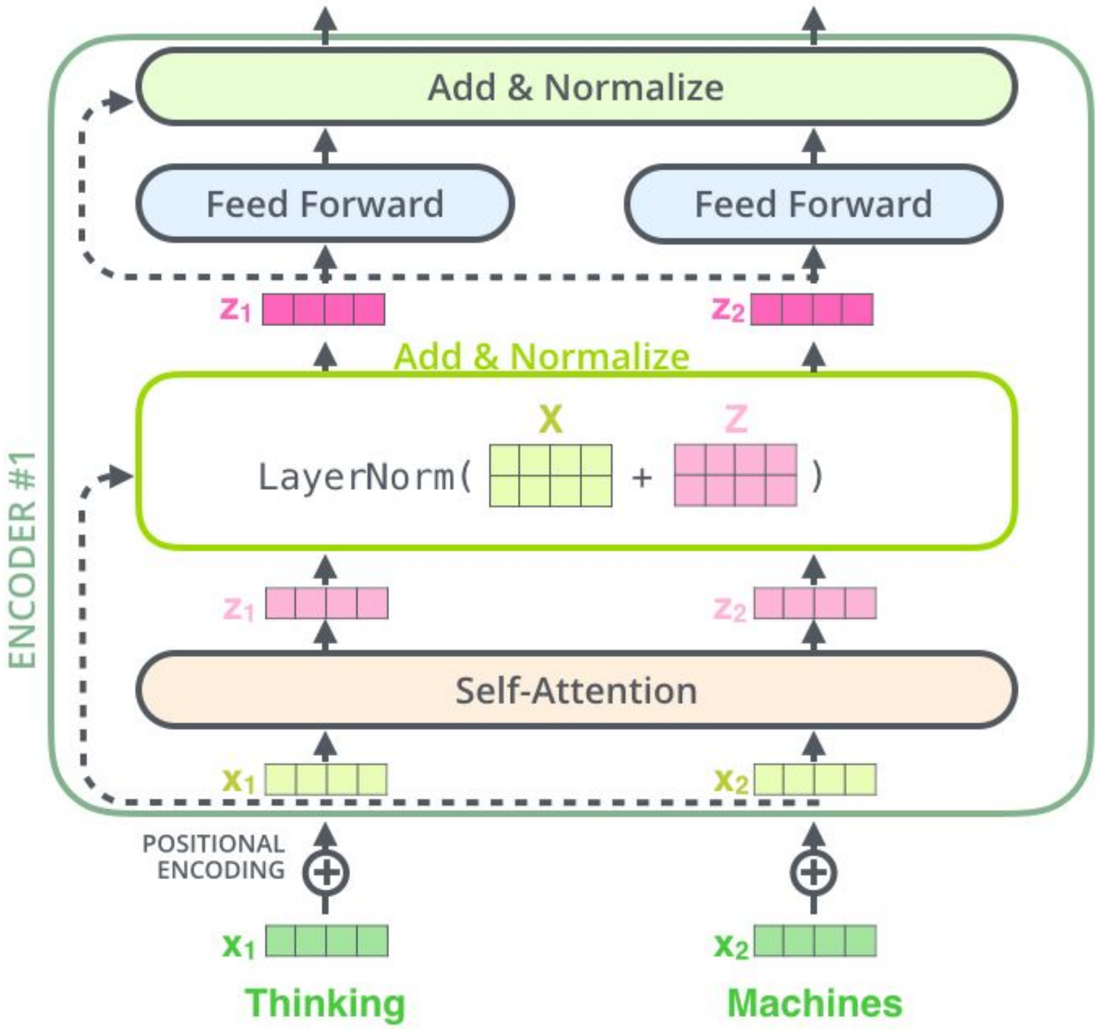
Can somebody present positional encoding following https://kazemnejad.com/blog/transformer_architecture_positional_encoding/



Attention and Transformers

Add and Normalize

In order to regulate the computation, this is a normalization layer so that each feature (column) have the same average and deviation.

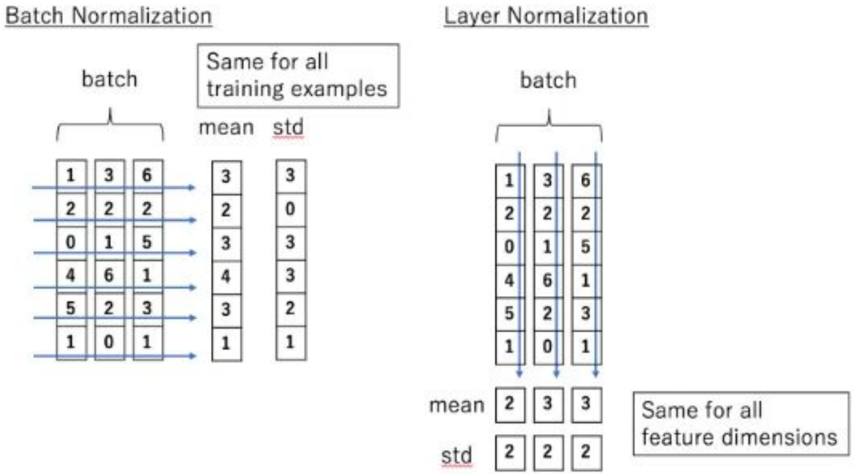


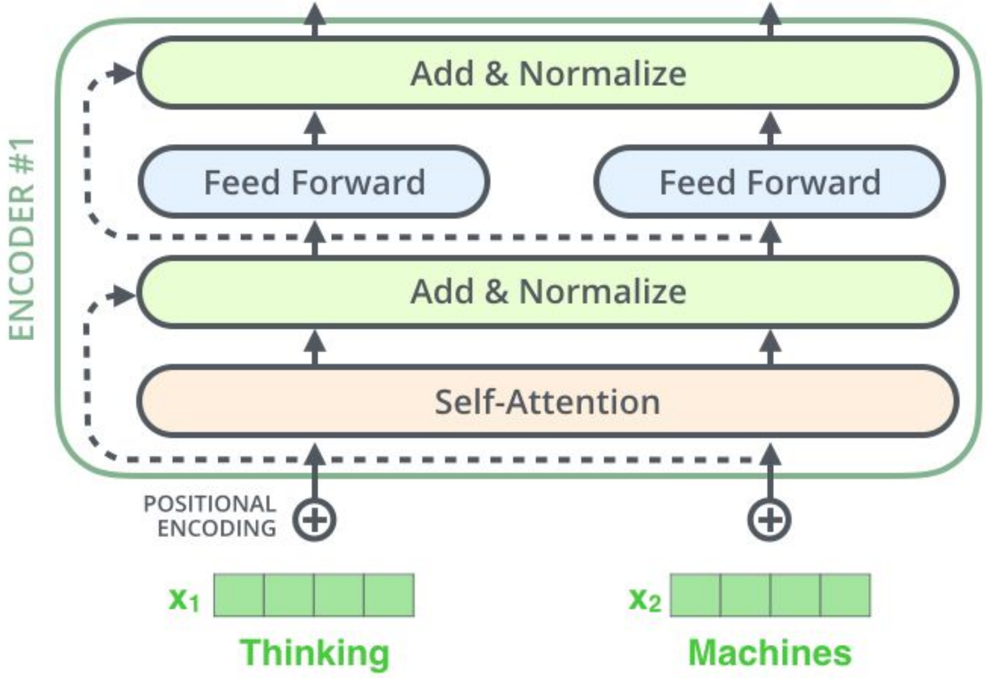
Attention and Transformers

Layer Normalization

(Hinton)

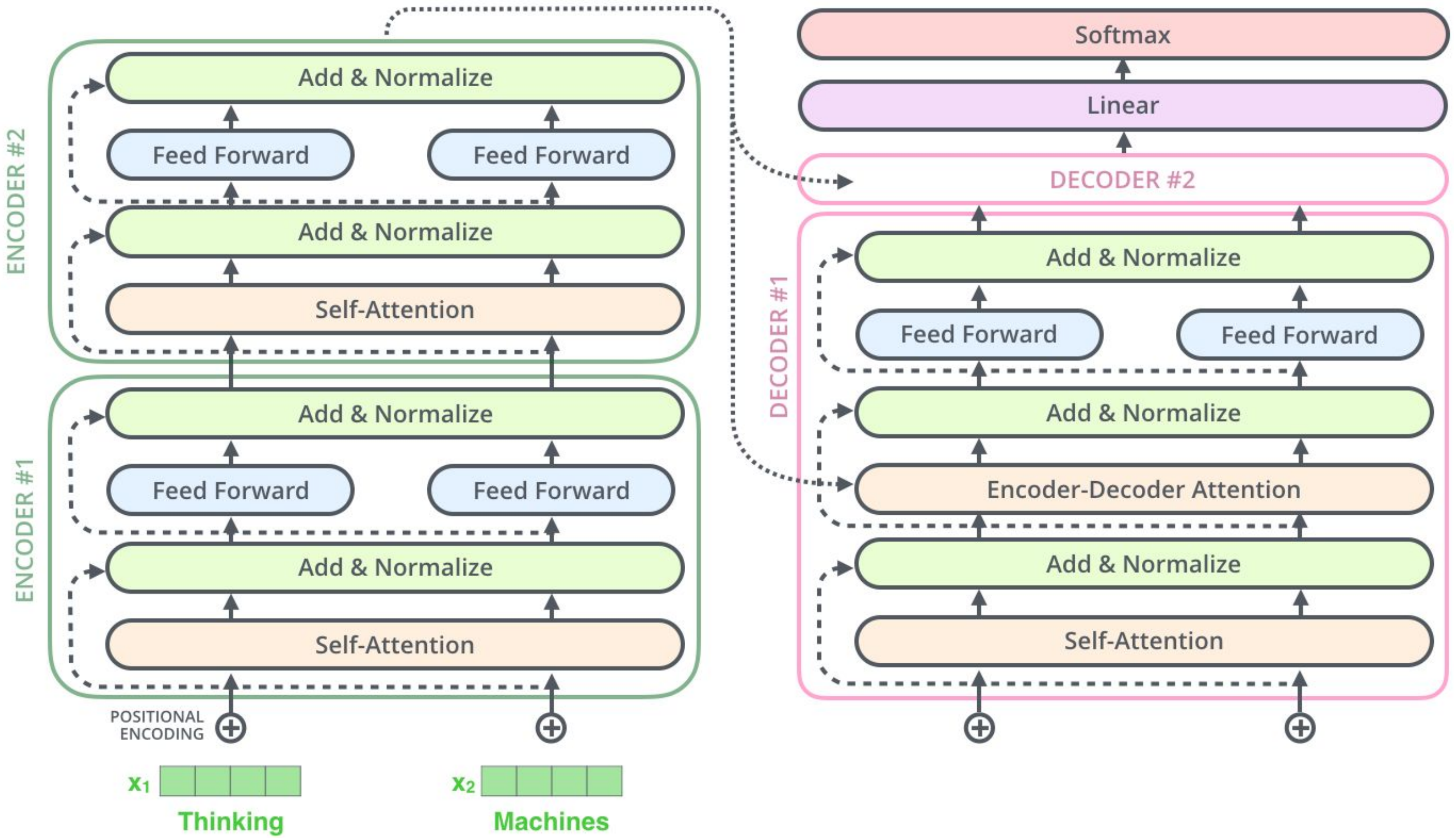
Layer normalization normalizes the inputs across the features.





The complete transformer

The encoder-decoder attention is just like self attention, except it uses K, V from the top of encoder output, and its own Q



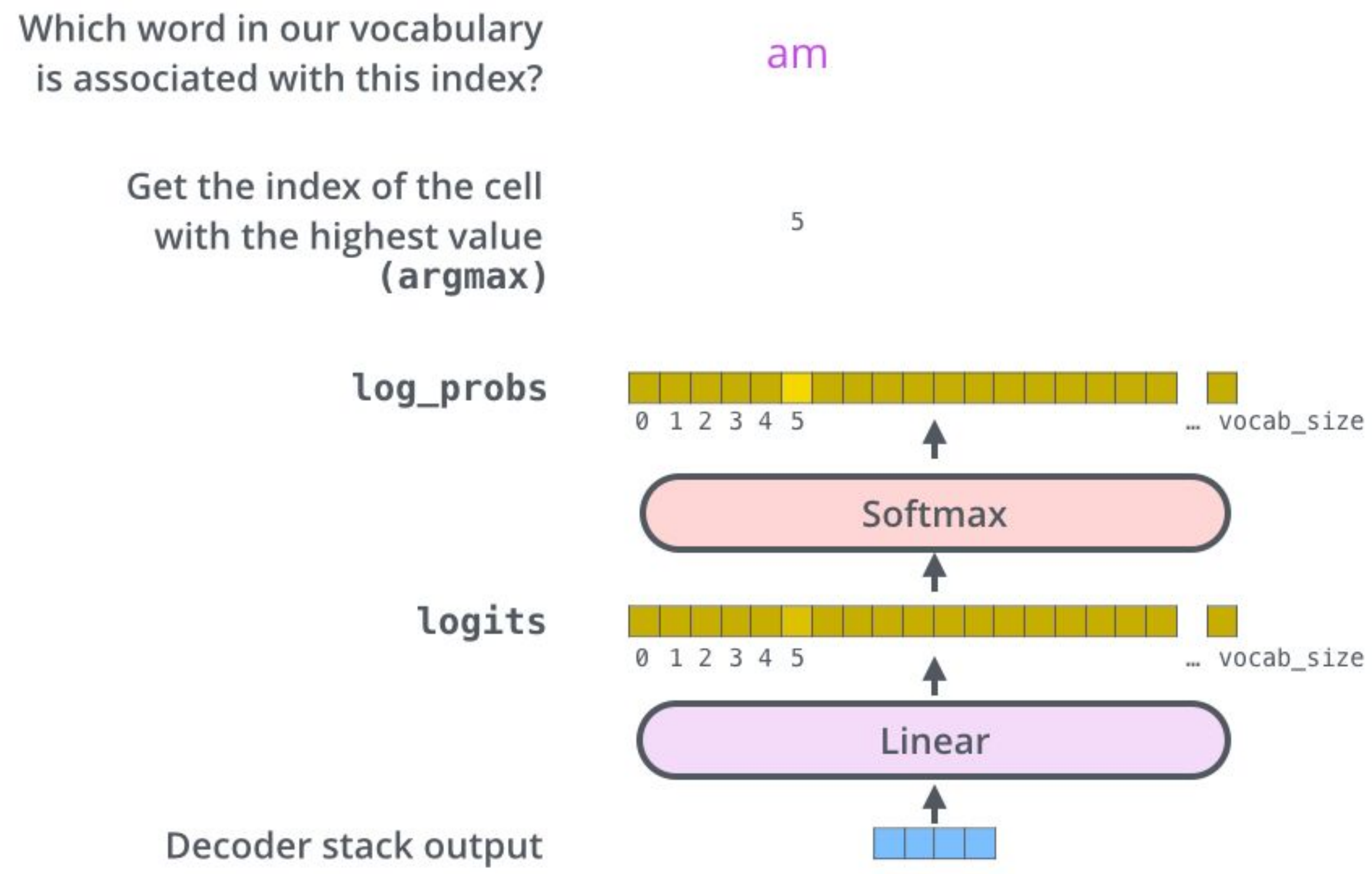
Attention and Transformers

Decoder’ s

Output

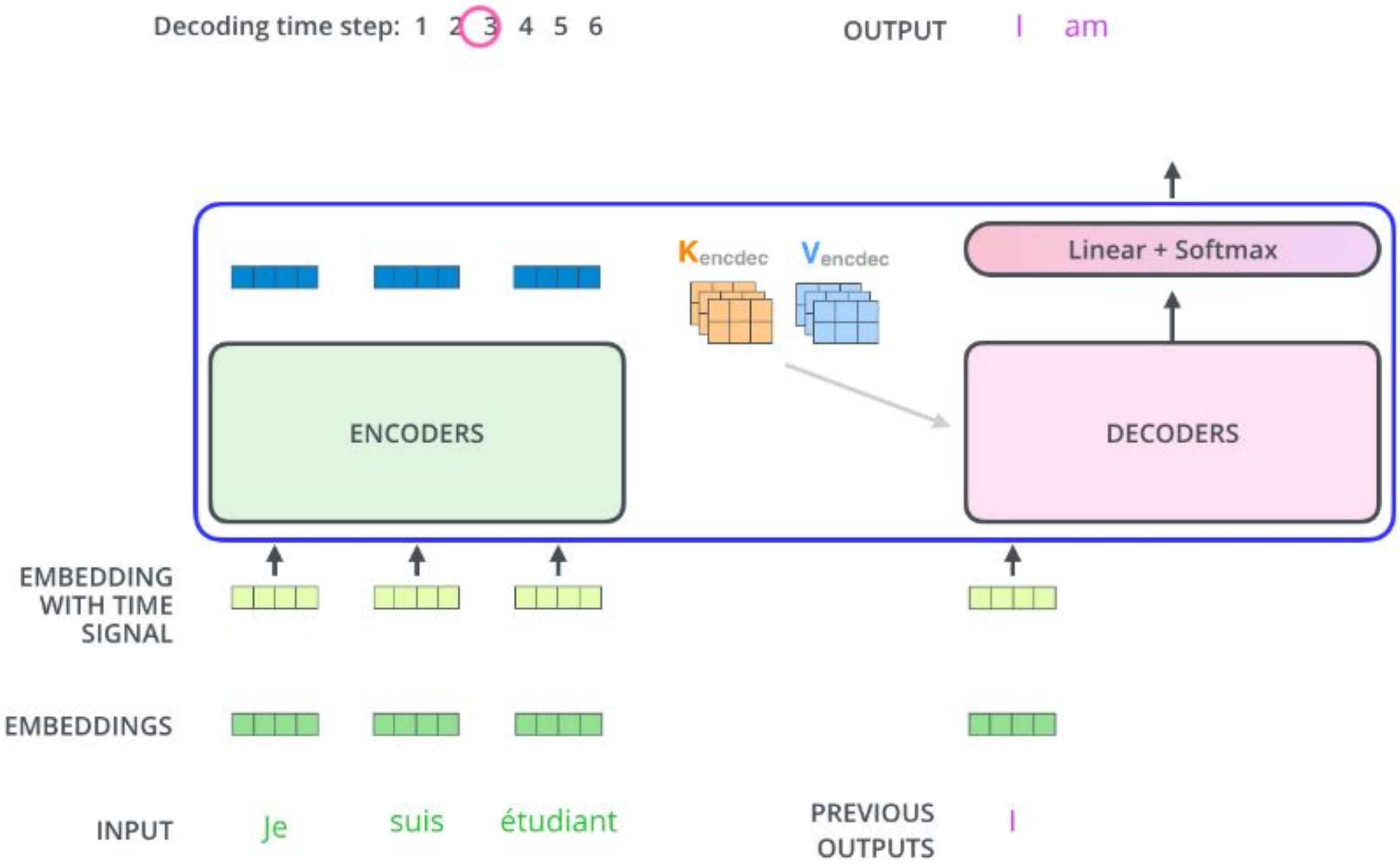
Linear

Layer



Attention and Transformers

How it works



But what about Self-attention?

Attention and Transformers

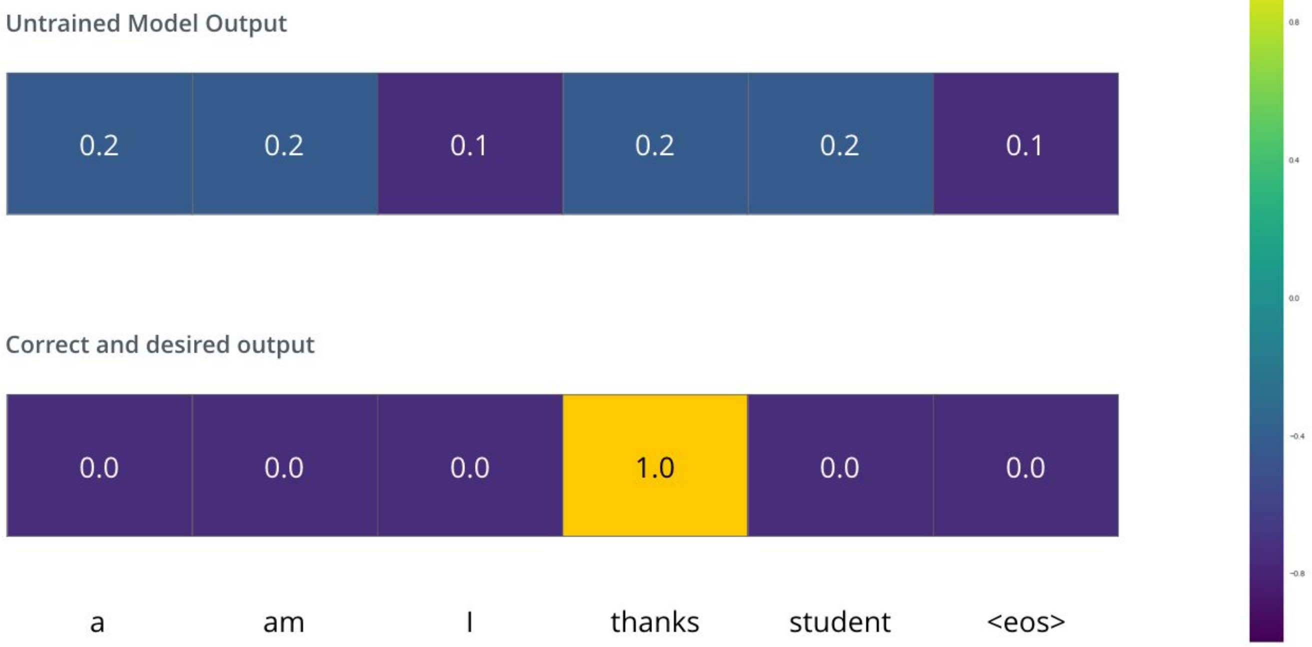
But what about self-attention when the input is "incomplete"?

The solution is to set future unknown values with "-inf".

The same for Encoder-Decoder Attention.

Attention and Transformers

Training and the Loss Function



We can use cross Entropy.

We can also optimize two words at a time: using BEAM search: keep a few alternatives for the first word.

Cross Entropy and KL (Kullback-Leibler) divergence

- Entropy: $E(P) = - \Sigma_i P(i) \log P(i)$ - expected code length (also optimal)
- Cross Entropy: $C(P) = - \Sigma_i P(i) \log Q(i)$ – expected coding length using optimal code for Q
- KL divergence:

  $D_{KL}(P \,||\, Q) = \Sigma_i P(i) \log[P(i)/Q(i)] = \Sigma_i P(i)[\log P(i) - \log Q(i)]$, extra bits

- JSD$(P||Q) = ½\, D_{KL}(P||M) + ½\, D_{KL}(Q||M)$, $M = ½\,(P+Q)$, symmetric KL

* JSD = Jensen-Shannon Divergency

Attention and Transformers

Transformer Results

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

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**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Thank you!

Q?