Carnegie Mellon University

Intermediate Deep Learning for Engineers

Spring 2025, Deep Learning for Engineers
March 27, 2025, 6th Session

Amir Barati Farimani
Associate Professor of Mechanical Engineering and Bio-Engineering
Carnegie Mellon University

Sequence Modeling

A RNN Language Model

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

$\boldsymbol{h}^{(0)}$ is the initial hidden state

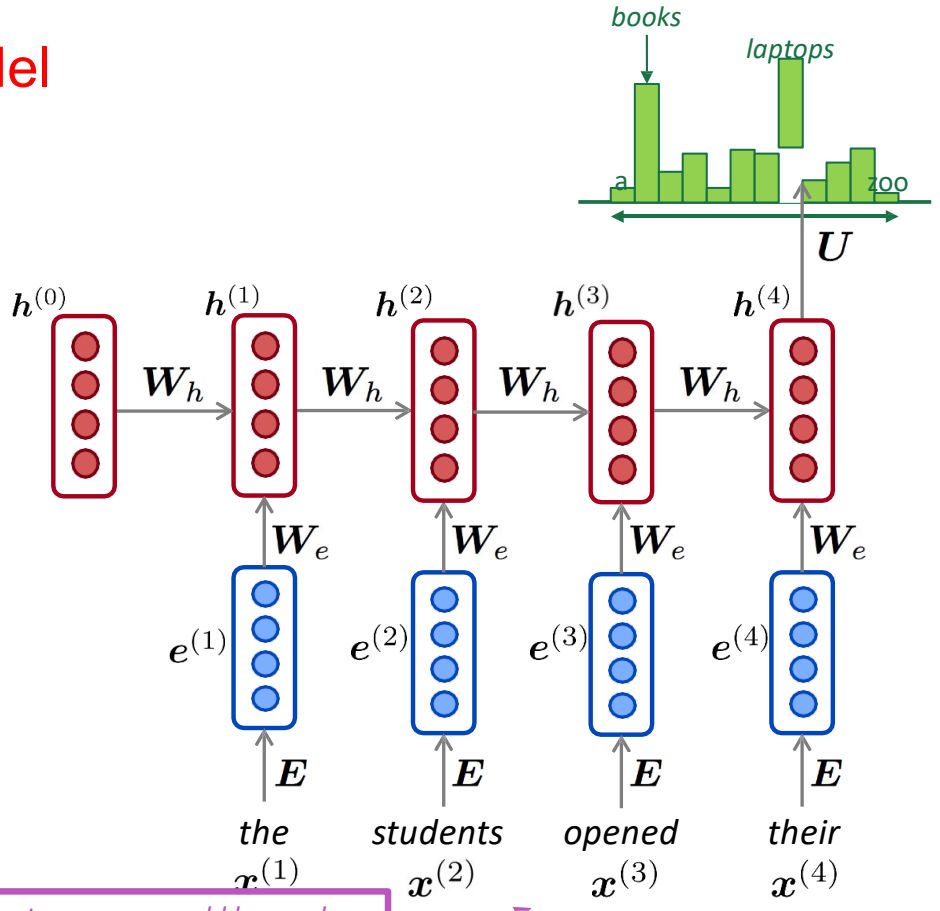
word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$

$\boldsymbol{y}^{(4)} = P(\boldsymbol{x}^{(5)}|\text{the students opened their})$



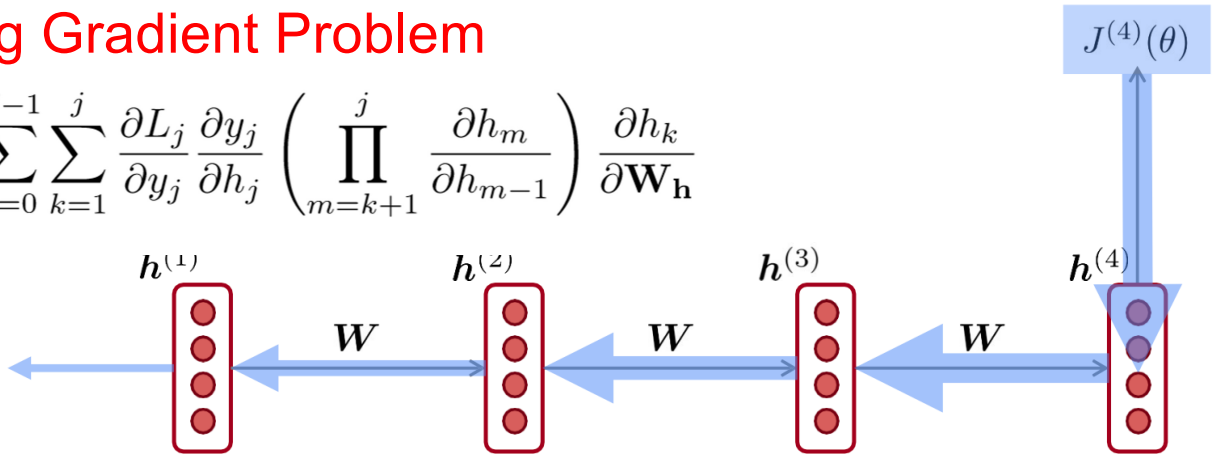
***Note**: this input sequence could be much longer, but this slide doesn't have space!*

Carnegie Mellon University

Sequence Modeling

Vanishing Gradient Problem

$$\frac{\partial L}{\partial \mathbf{W_h}} = \sum_{j=0}^{T-1} \sum_{k=1}^{j} \frac{\partial L_j}{\partial y_j} \frac{\partial y_j}{\partial h_j} \left(\prod_{m=k+1}^{j} \frac{\partial h_m}{\partial h_{m-1}} \right) \frac{\partial h_k}{\partial \mathbf{W_h}}$$



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}}{\partial h^{(4)}}$$

What happens if these are small?

Vanishing gradient problem:
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

Carnegie Mellon University

Sequence Modeling

Challenges with RNNs

- Long range dependencies
- Gradient vanishing and explosion
- Large # of training steps
- Recurrence prevents parallel computation
- Slow computation for long sequences
- Vanishing or exploding gradients
- Difficulty in accessing information from long time ago

Attention

Attention in Computer Vision

– 2014: Attention used to highlight important parts of an

image that contribute to a desired output

• Attention in NLP

– 2015: Aligned machine translation

– 2017: Language modeling with **Transformer networks**

Carnegie Mellon University

Why Attention

Attention is a recent and important component to the success of modern neural networks

We want neural nets that **automatically weigh** relevance of the input and **use these weights** to perform a task

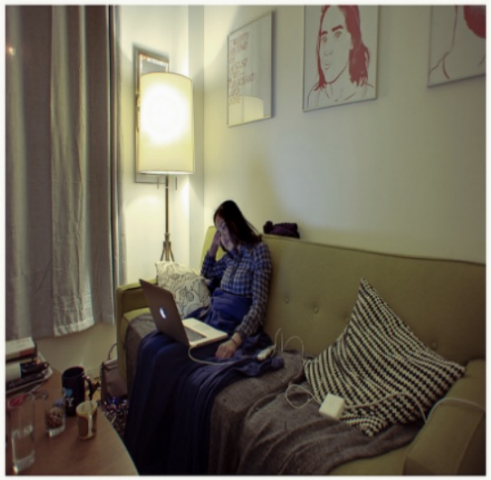
Main advantages:

- performance gain
- none or few parameters
- fast (easy to parallelize)
- drop-in implementation
- tool for "interpreting" predictions

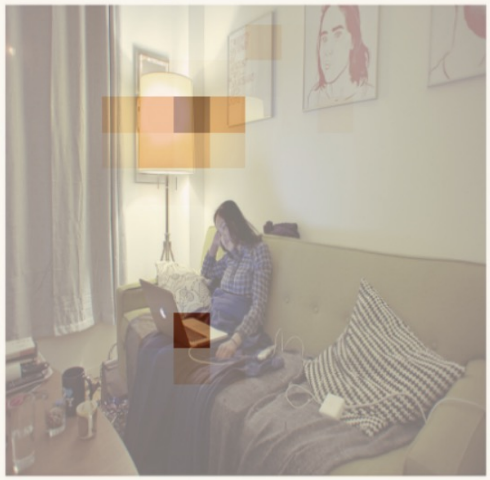
Carnegie Mellon University

Attention Example

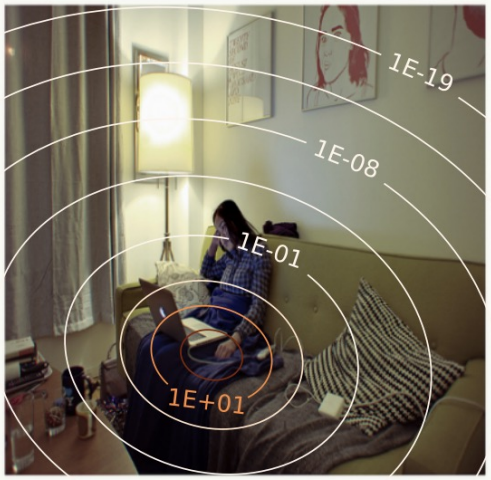
What is the woman looking at?



tv



computer

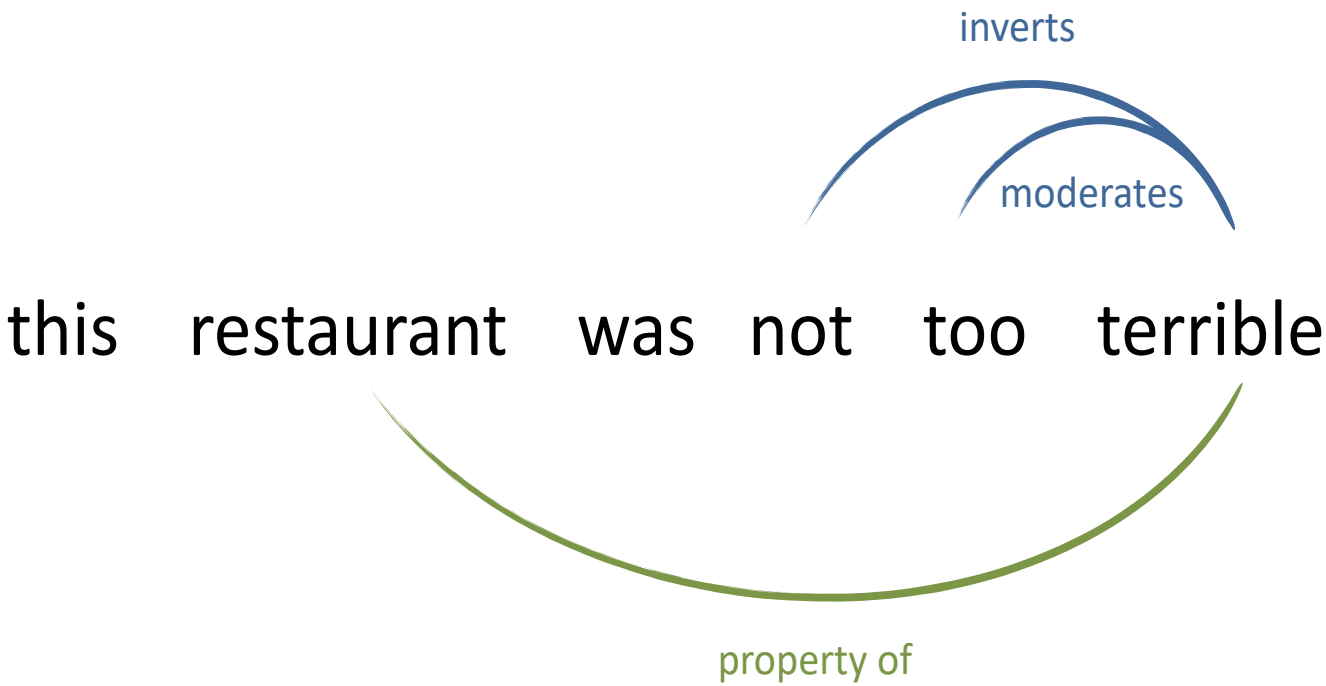


computer

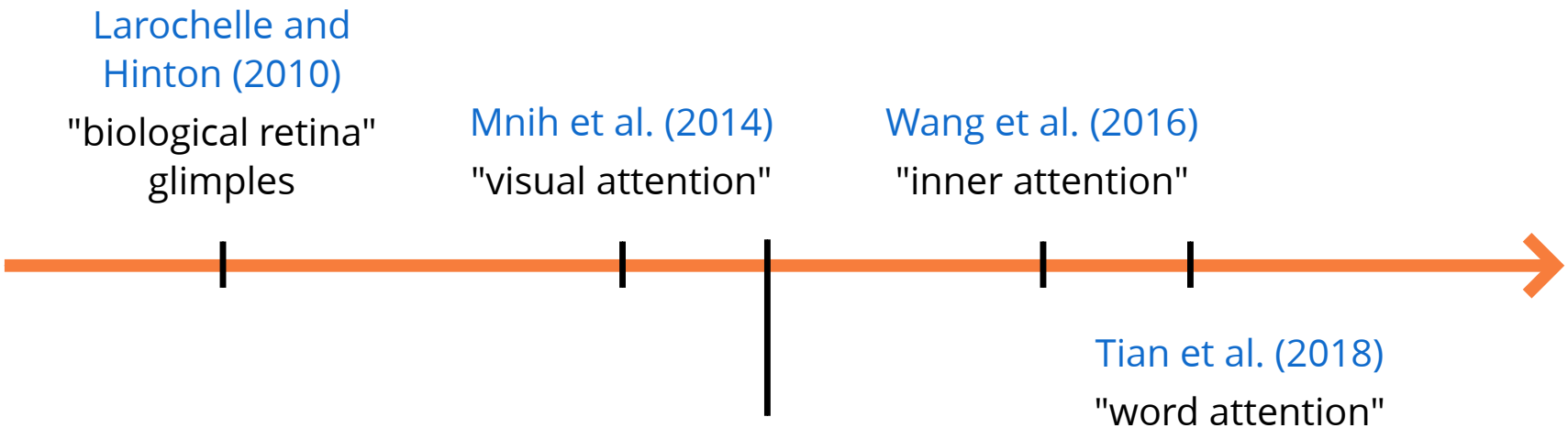
(Martins et al., 2020)

Carnegie Mellon University

Attention Example

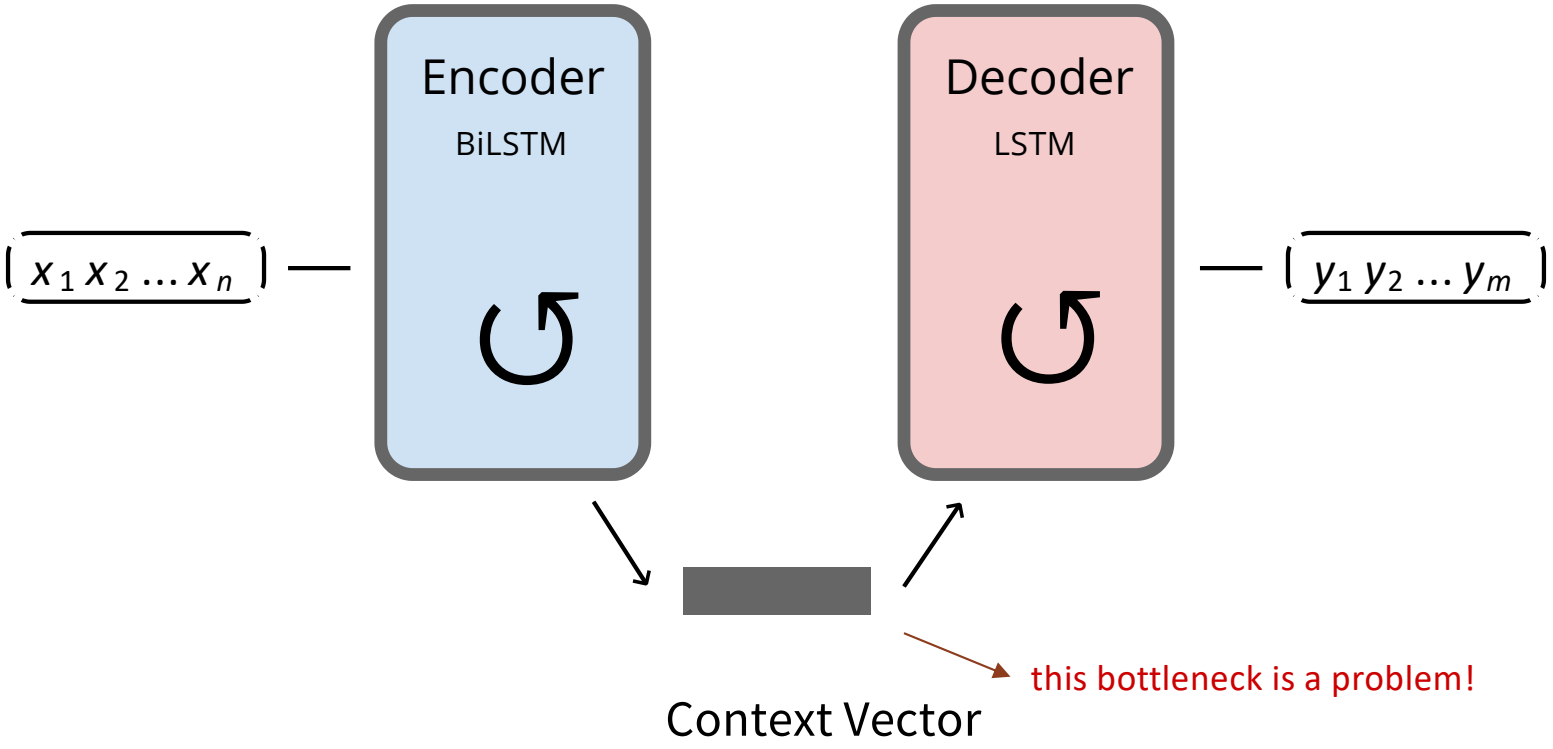


Attention History

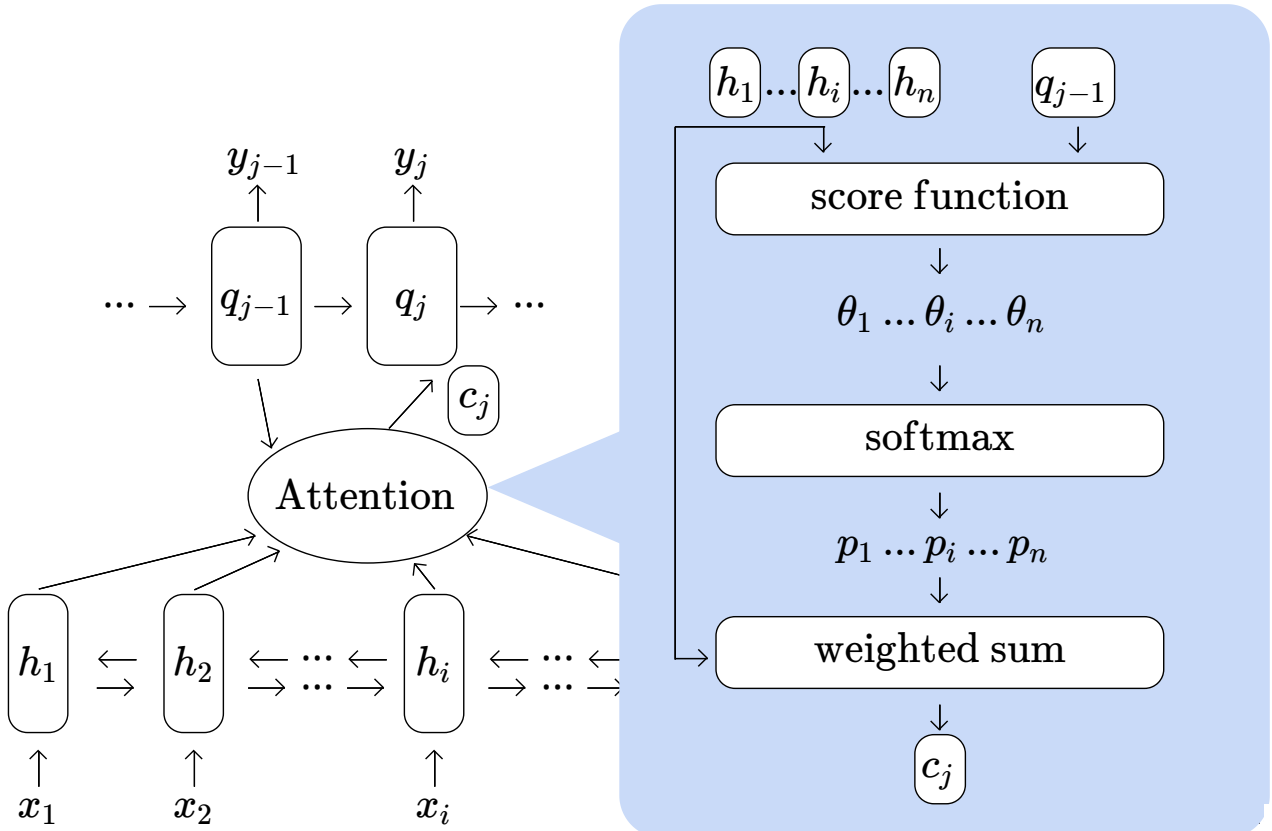


"first" introduced in NLP for Machine Translation by Bahdanau et al. (2015)

RNN-based seq2seq



Attention in Seq2Seq (2015)

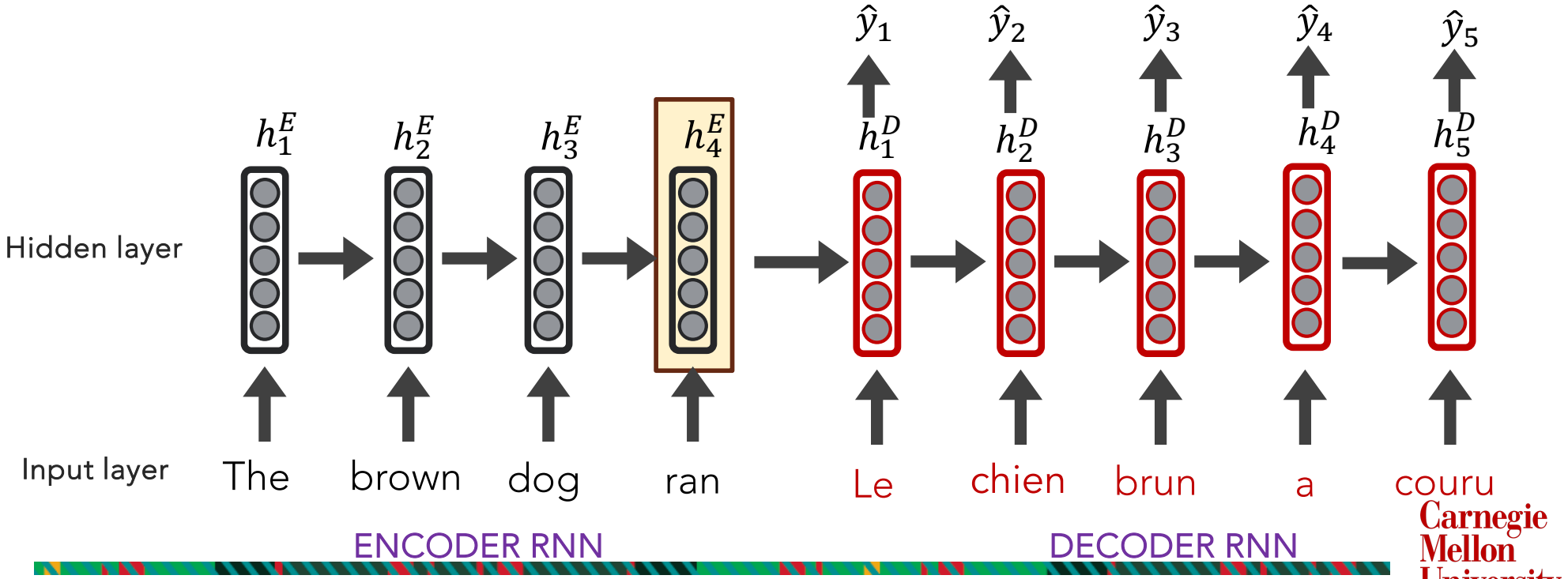


Bahdanau et al. (2015)

Carnegie Mellon University

Sequence-to-Sequence (seq2seq)

It's crazy that the entire "meaning" of the 1st sequence is expected to be packed into this one embedding, and that the encoder then never interacts w/ the decoder again. Hands free.



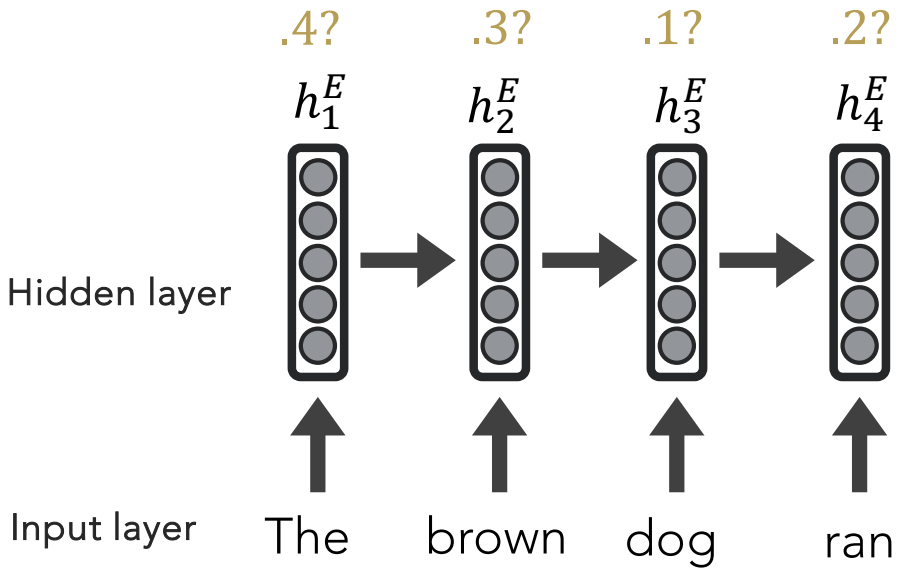
Carnegie Mellon University

Seq2Seq models

Instead, what if the decoder, at each step, pays attention to a *distribution* of all of the encoder's hidden states?

Seq2Seq models

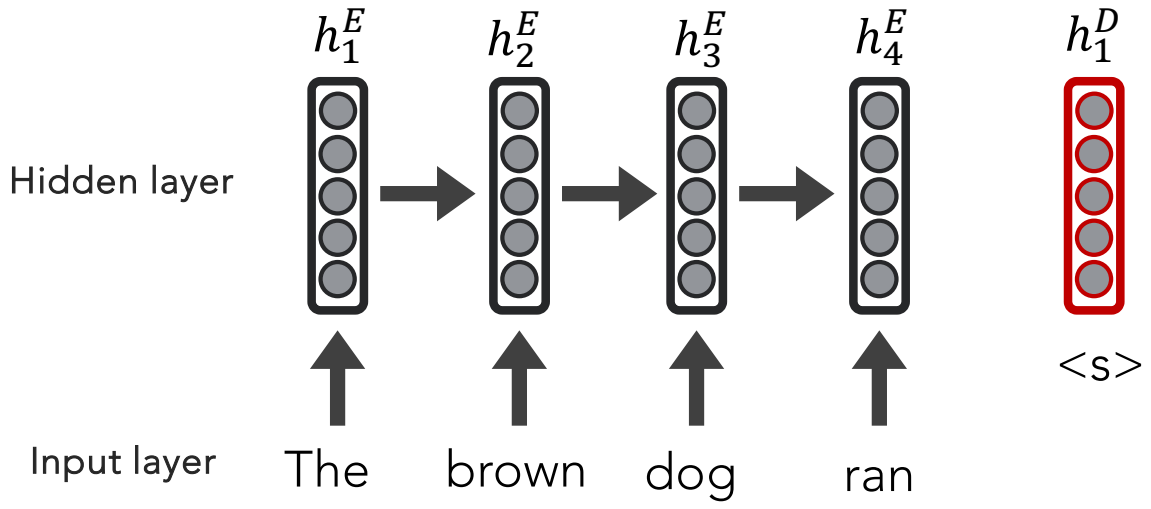
Q: How do we determine how much to pay attention to each of the encoder's hidden layers?



Seq2Seq + Attention models

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden layers!

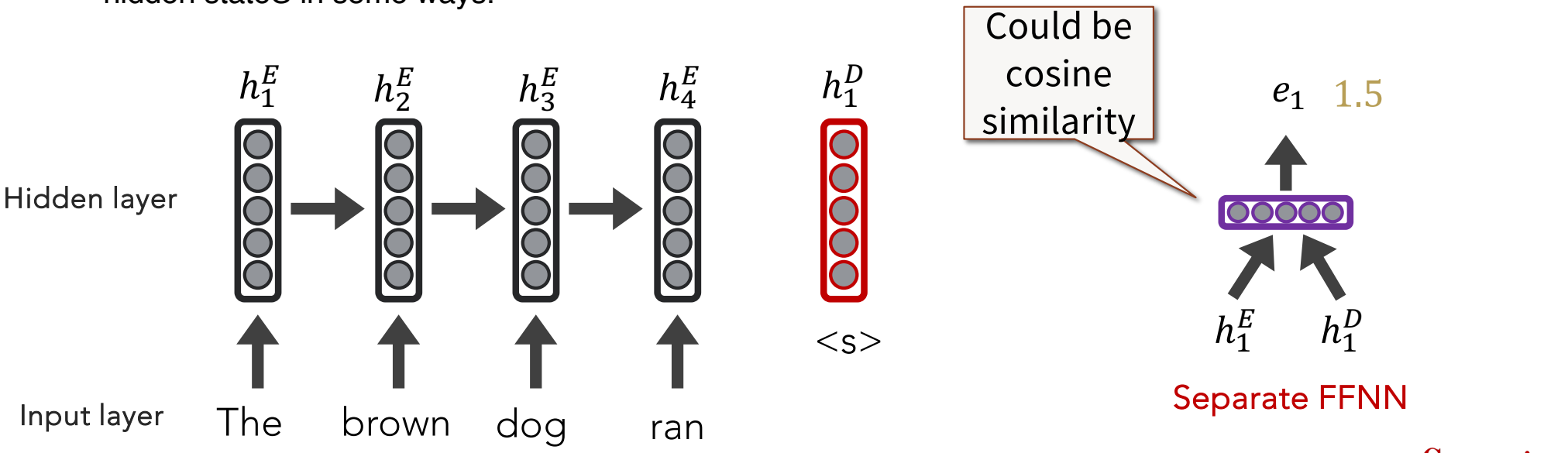


ENCODER RNN

Seq2Seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden layers! We want to measure similarity between decoder hidden state and encoder hidden stateS in some ways.

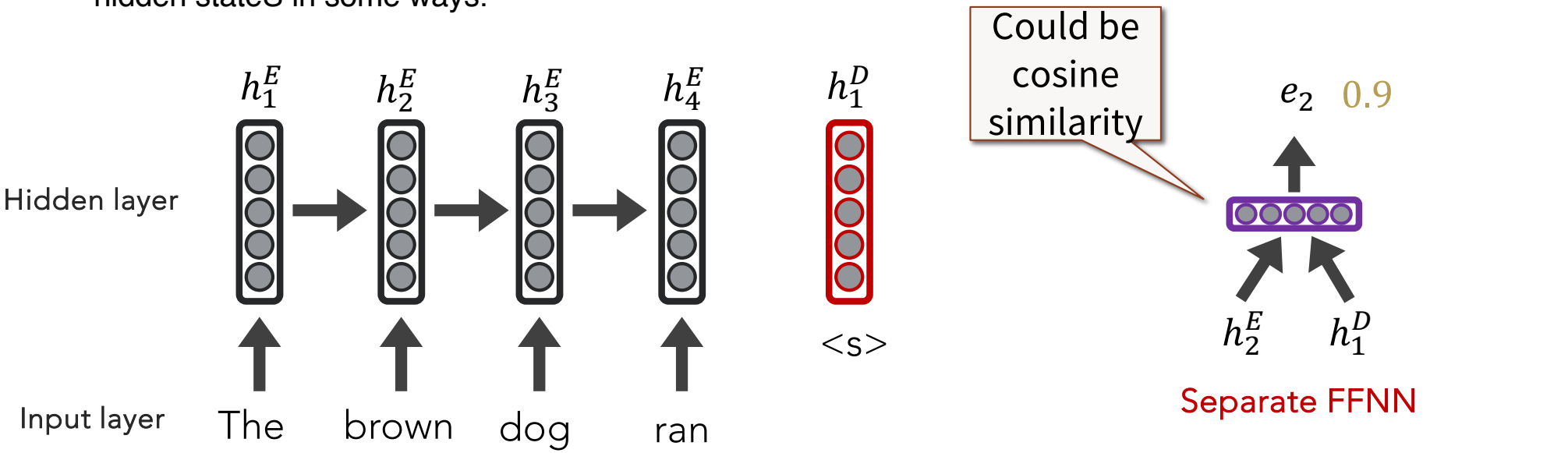


Carnegie Mellon University

Seq2Seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden layers! We want to measure similarity between decoder hidden state and encoder hidden stateS in some ways.

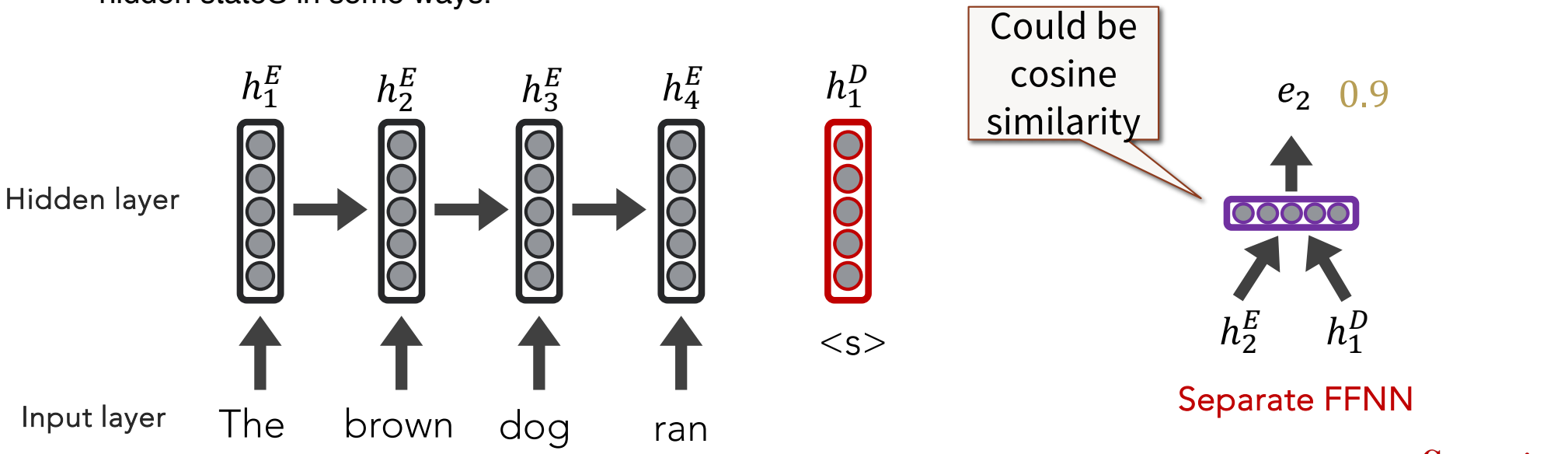


Carnegie Mellon University

Seq2Seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden layers! We want to measure similarity between decoder hidden state and encoder hidden stateS in some ways.

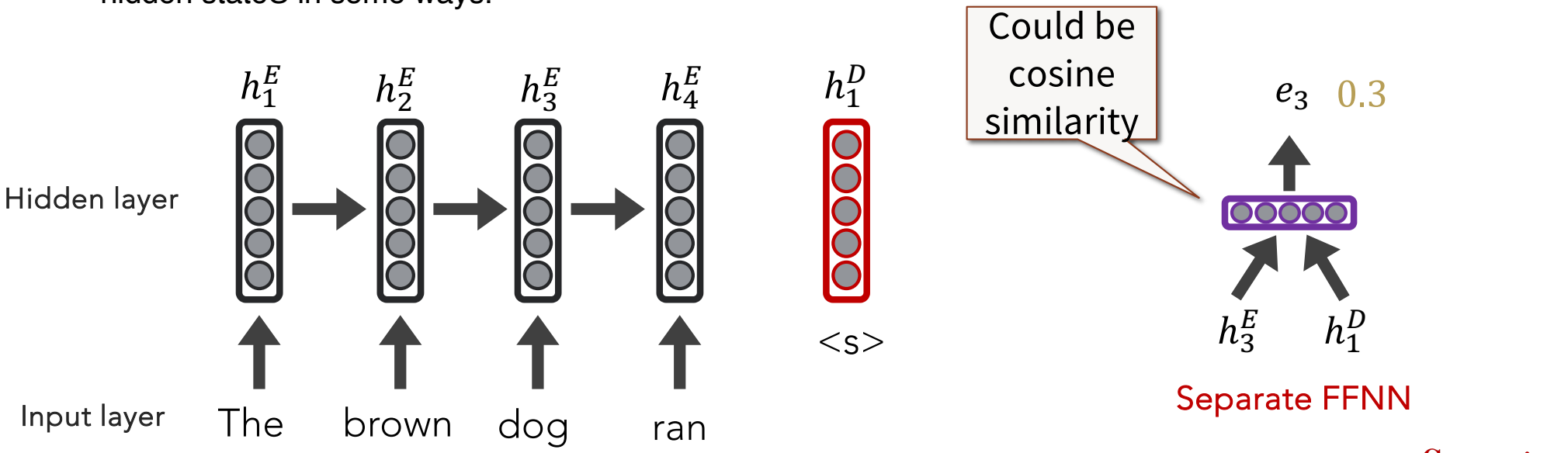


Carnegie Mellon University

Seq2Seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden layers! We want to measure similarity between decoder hidden state and encoder hidden stateS in some ways.

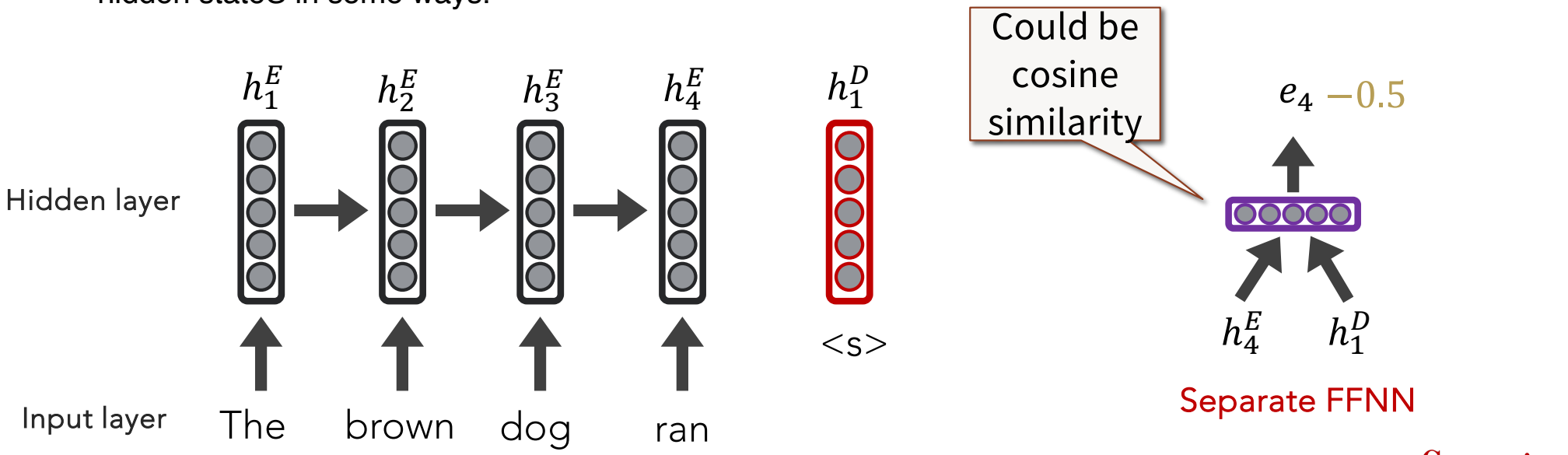


Carnegie Mellon University

Seq2Seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden layers! We want to measure similarity between decoder hidden state and encoder hidden stateS in some ways.

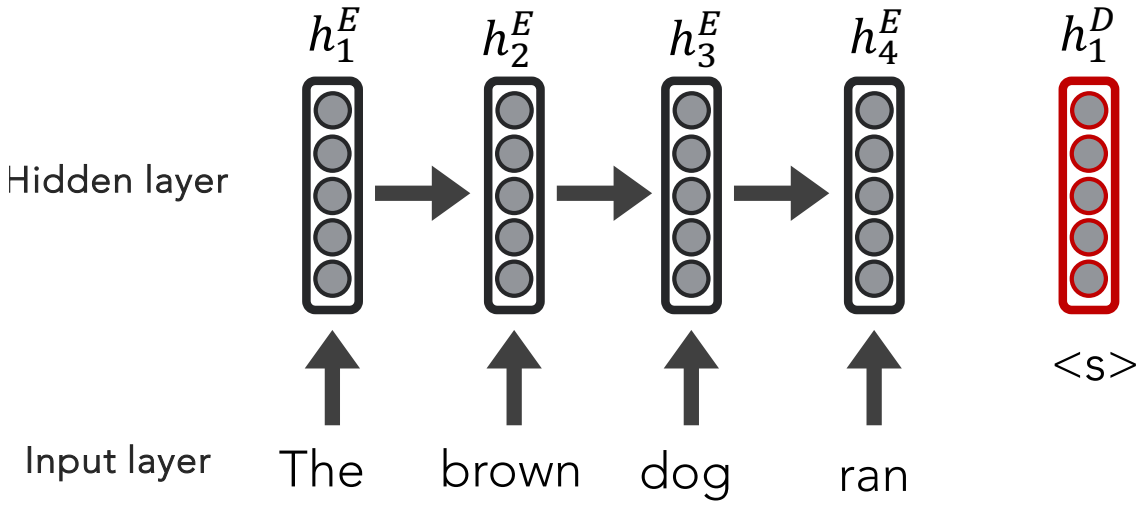


Carnegie Mellon University

Seq2Seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden layers!



Attention (raw scores)

e_1 1.5
e_2 0.9
e_3 0.2
e_4 −0.5

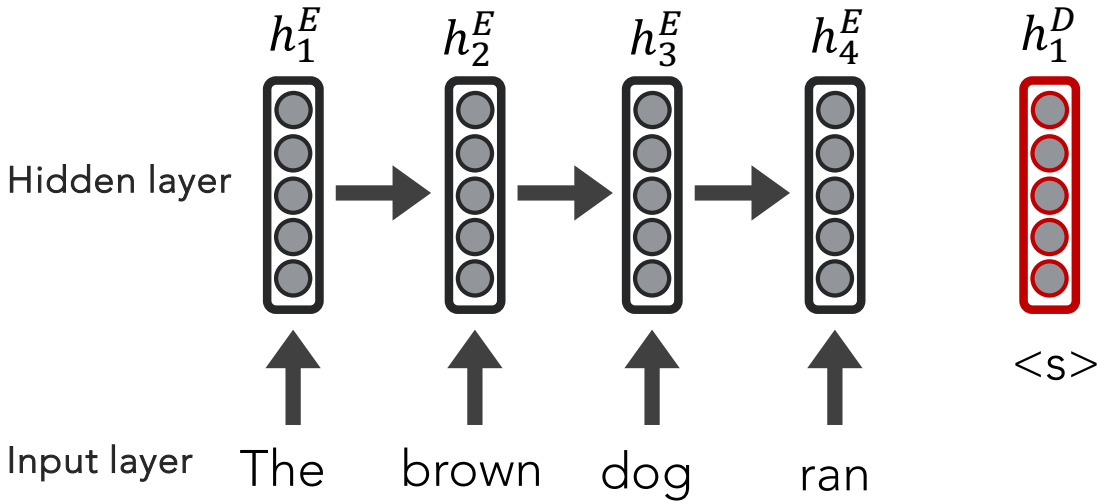
ENCODER RNN DECODER RNN ATTENTION LAYER

Carnegie Mellon University

Seq2Seq + Attention

Q: How do we determine how much to pay attention to each of the encoder's hidden layers?

A: Let's base it on our decoder's previous hidden state (our latest representation of meaning) and all of the encoder's hidden layers!



Attention (raw scores)

e_1 1.5
e_2 0.9
e_3 0.2
e_4 −0.5

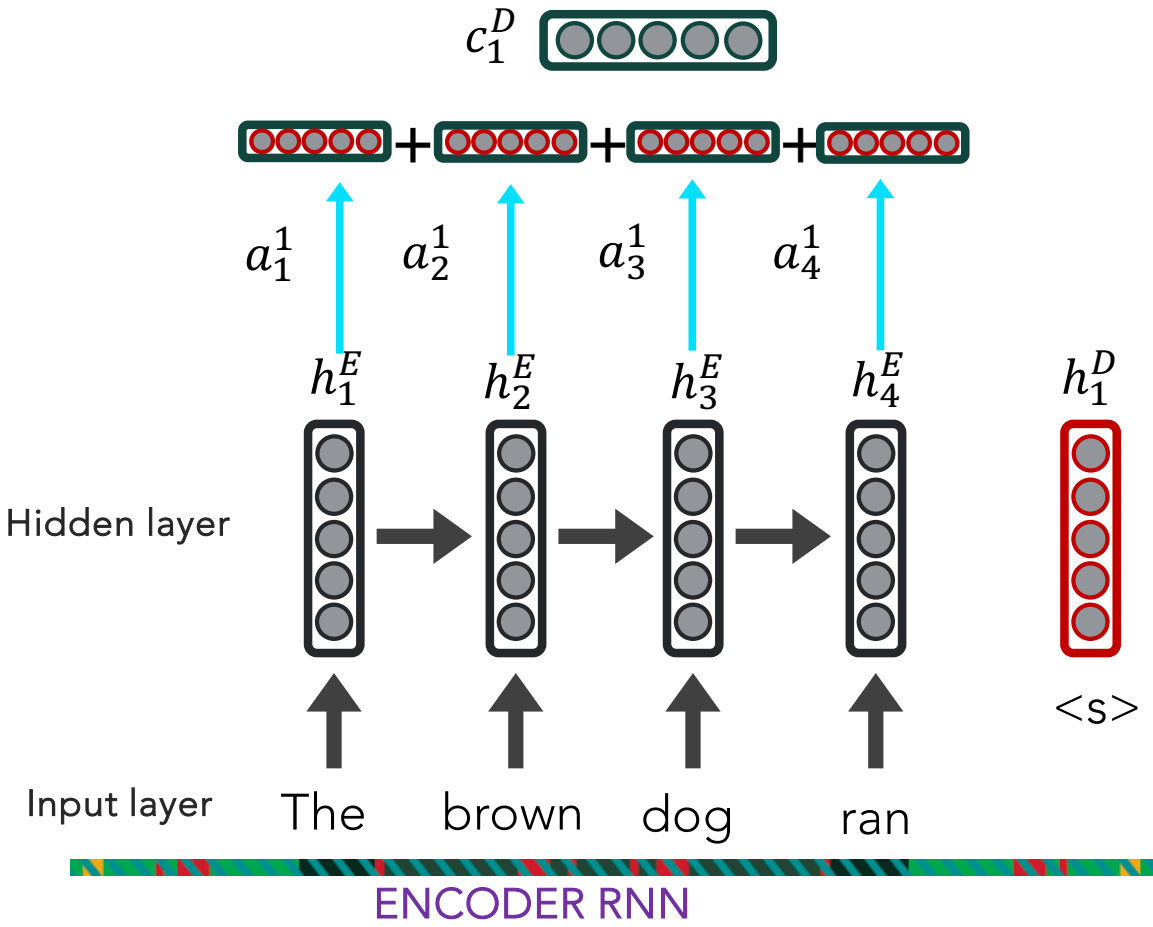
Attention (softmax'd)

$$a_i^1 = \frac{\exp(e_i)}{\sum_i^N \exp(e_i)}$$

ATTENTION LAYER

Carnegie Mellon University

Seq2Seq + Attention



We multiply each hidden layer by its a_i^1 attention weights and then add the resulting vectors to create a context vector c_1^D

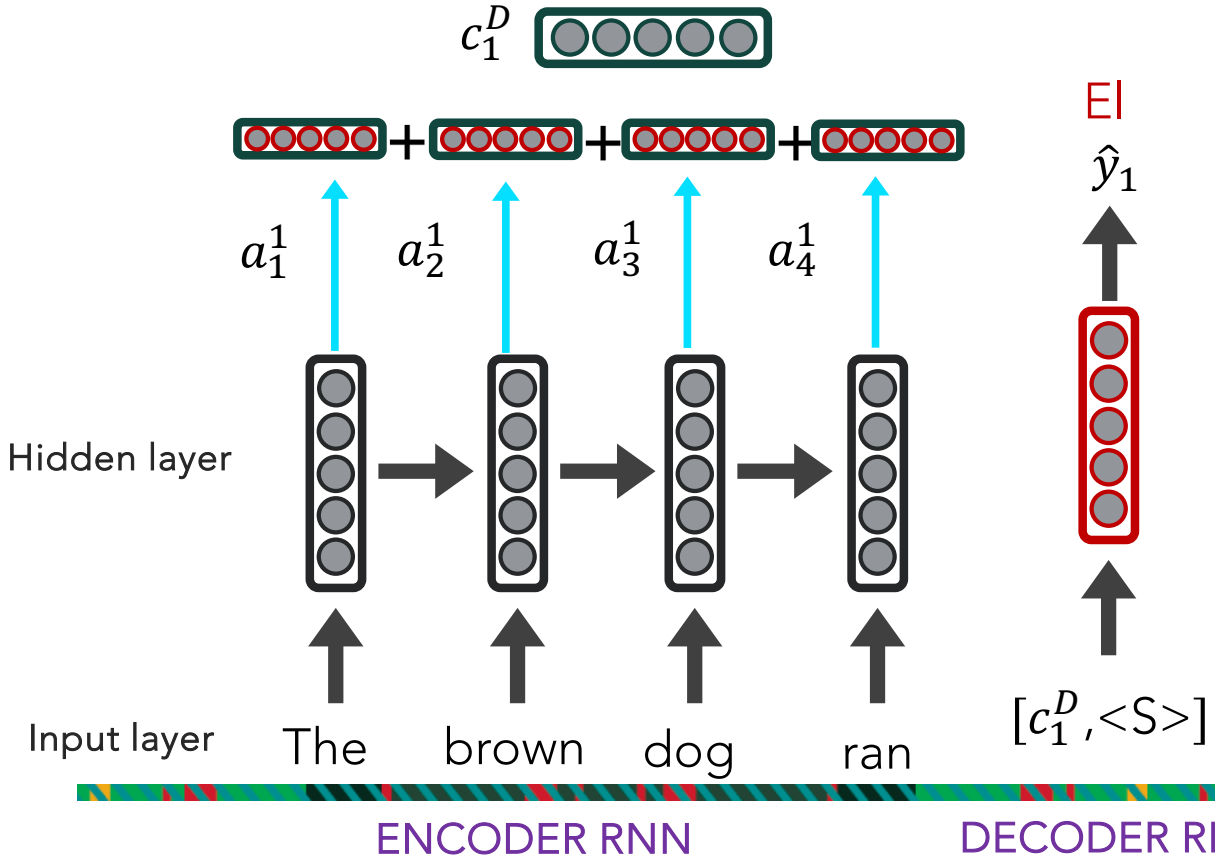
Attention (softmax'd)

$$a_1^1 = 0.51$$
$$a_2^1 = 0.28$$
$$a_3^1 = 0.14$$
$$a_4^1 = 0.07$$

Carnegie Mellon University

Seq2Seq + Attention

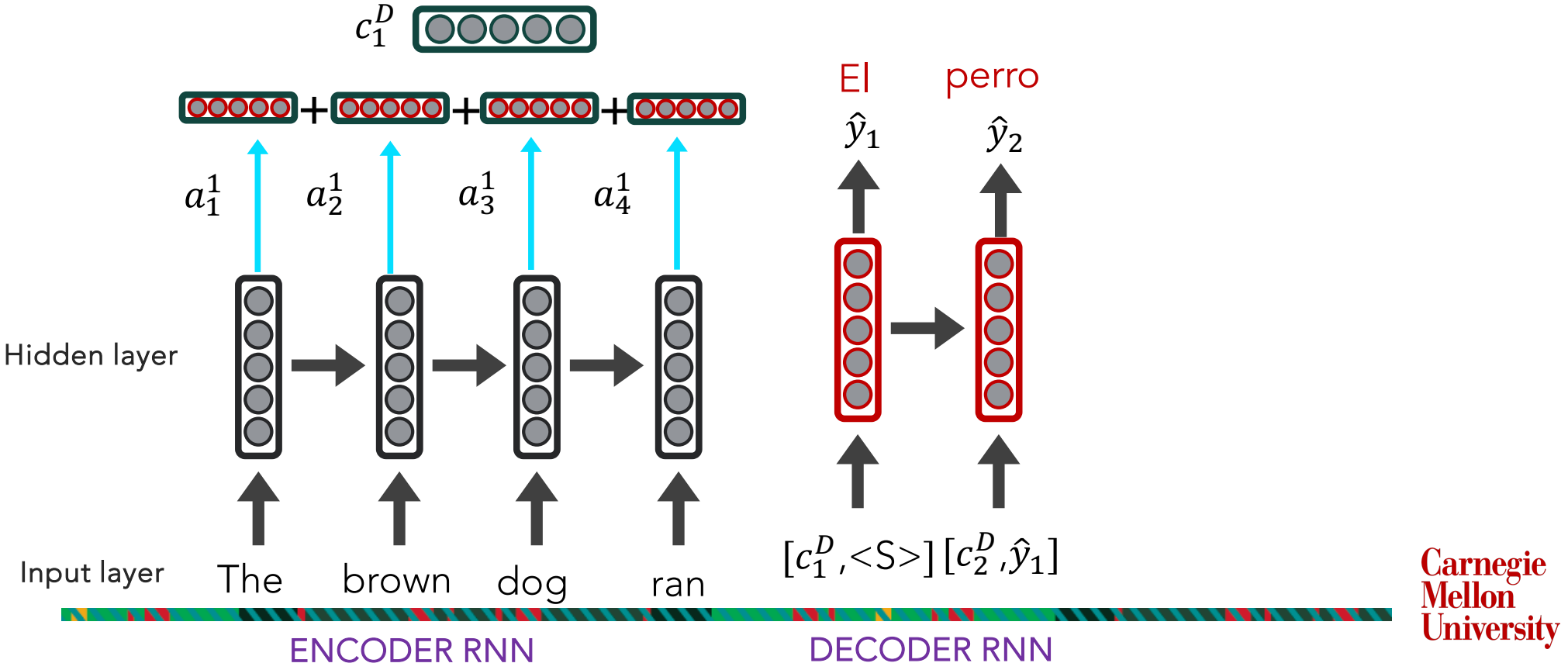
NOTE: each attention weight a_i^j is based on the decoder's current hidden state, too.



Carnegie Mellon University

Seq2Seq + Attention

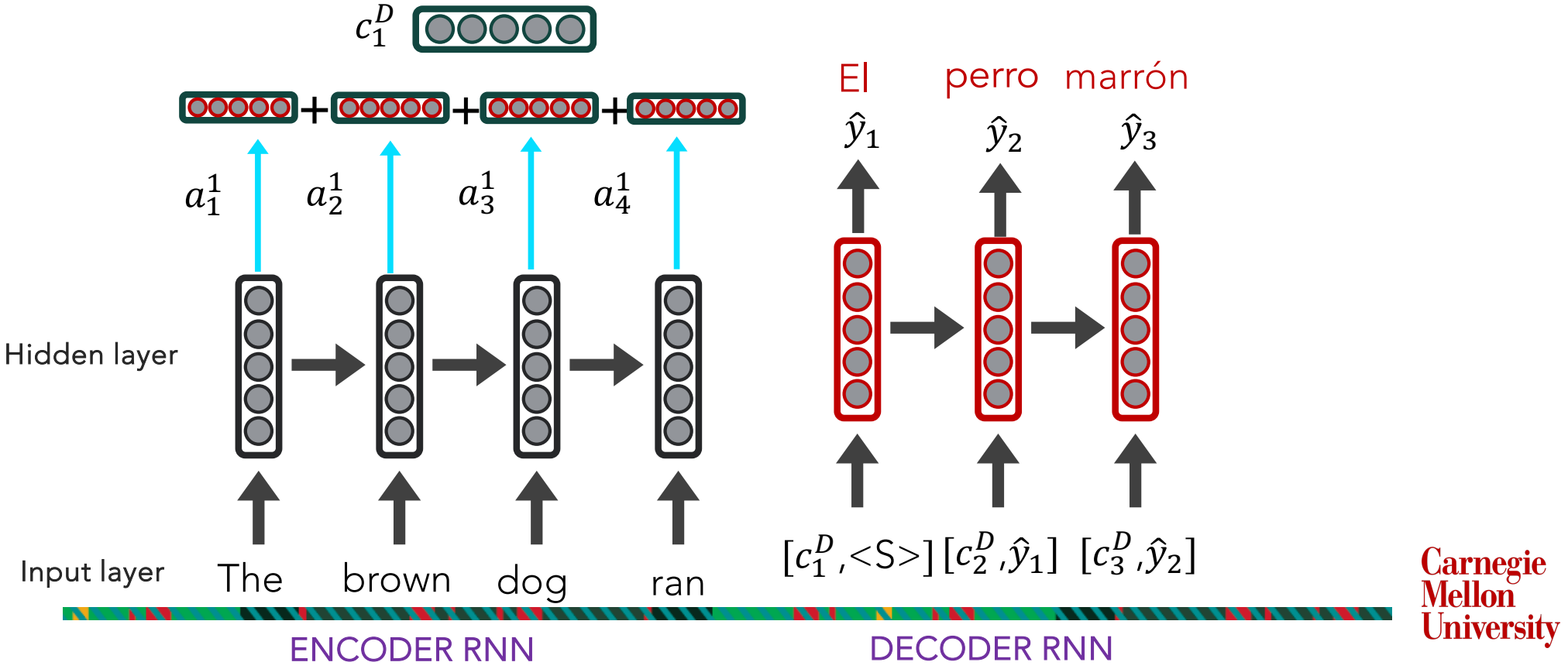
NOTE: each attention weight a_i^j is based on the decoder's current hidden state, too.



Carnegie Mellon University

Seq2Seq + Attention

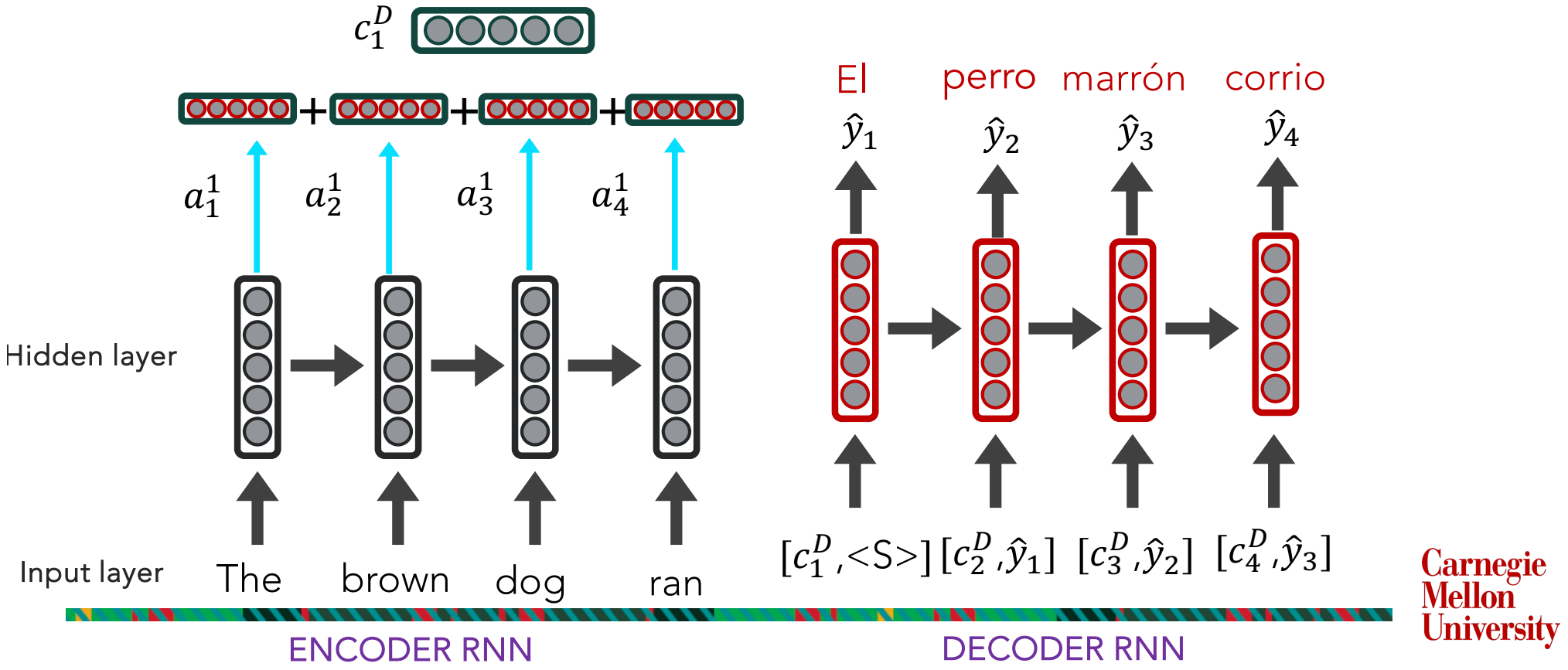
NOTE: each attention weight a_i^j is based on the decoder's current hidden state, too.



Carnegie Mellon University

Seq2Seq + Attention

NOTE: each attention weight a_i^j is based on the decoder's current hidden state, too.



Carnegie Mellon University

Seq2Seq + Attention

Attention:

- greatly improves seq2seq results
- allows us to visualize the contribution each word gave during each step of the decoder

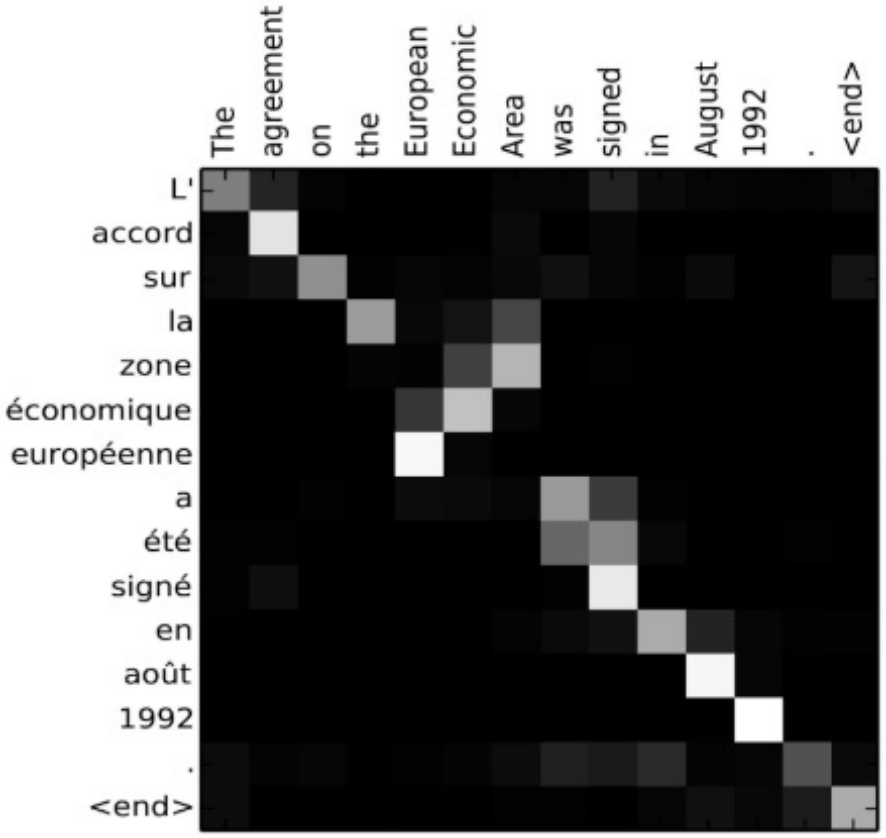


Image source: Fig 3 in Bahdanau et al., 2015

But still the problem of Sequential Computing Exist Because Backprop in time exists

Image source: Fig 3 in Bahdanau et al., 2015

Carnegie Mellon University

Transformers

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

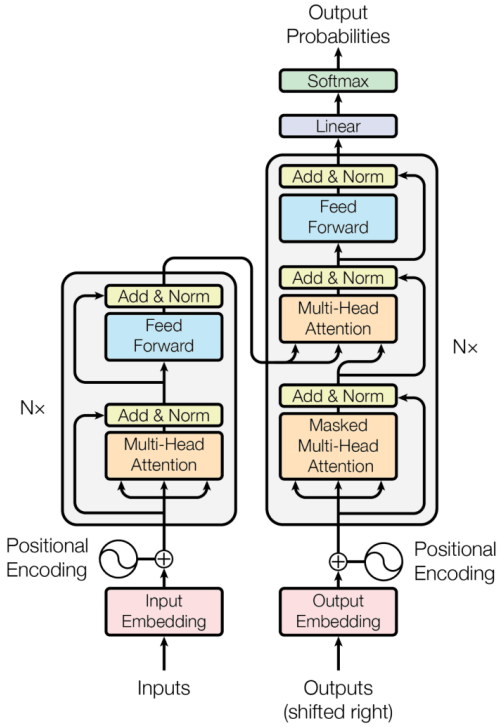
Jakob Uszkoreit* Google Research usz@google.com

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

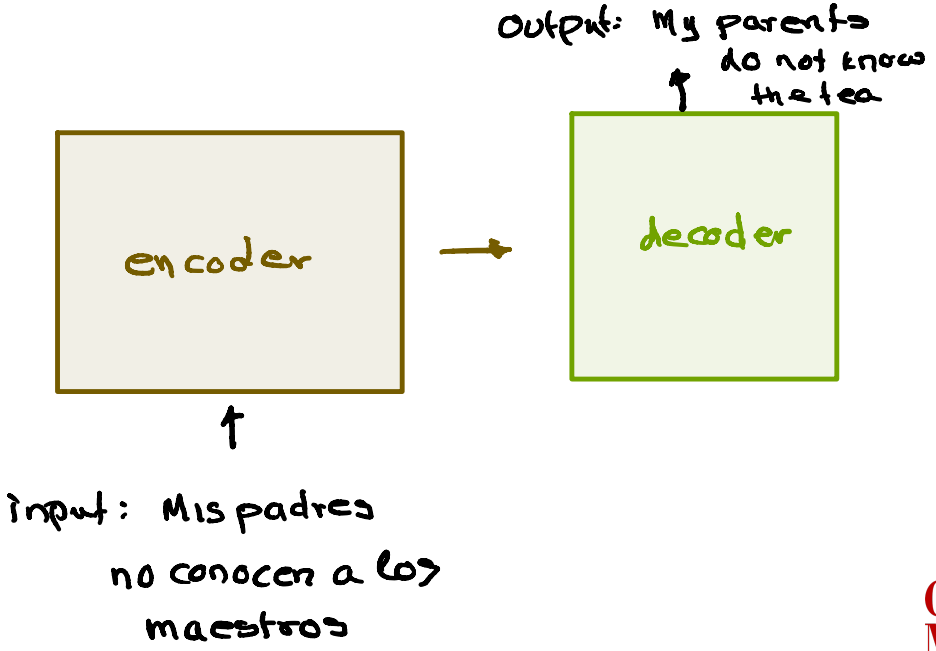


Carnegie Mellon University

Transformers

The Transformer is a model that uses attention to boost the speed with which seq2seq with attention models can be trained. The biggest benefit, however, comes from how The Transformer lends itself to **parallelization**. We will break it apart and look at how it functions.

In its heart it contains an encoding component, a decoding component, and connections between them.

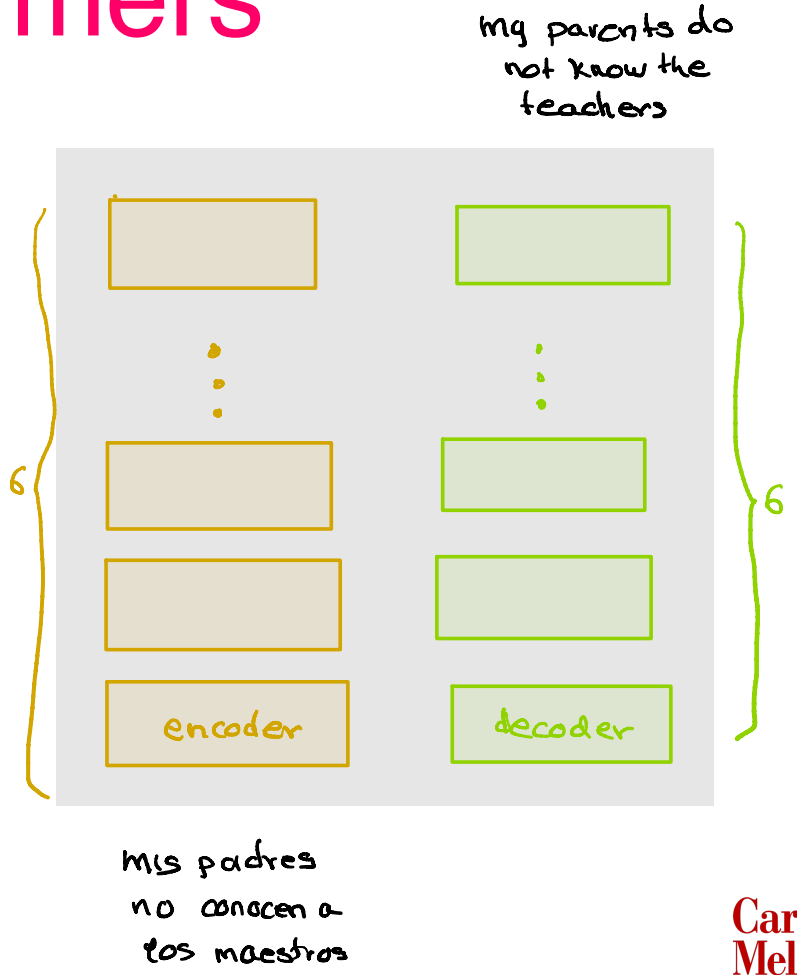


Transformers

The encoding is a stack of encoders.

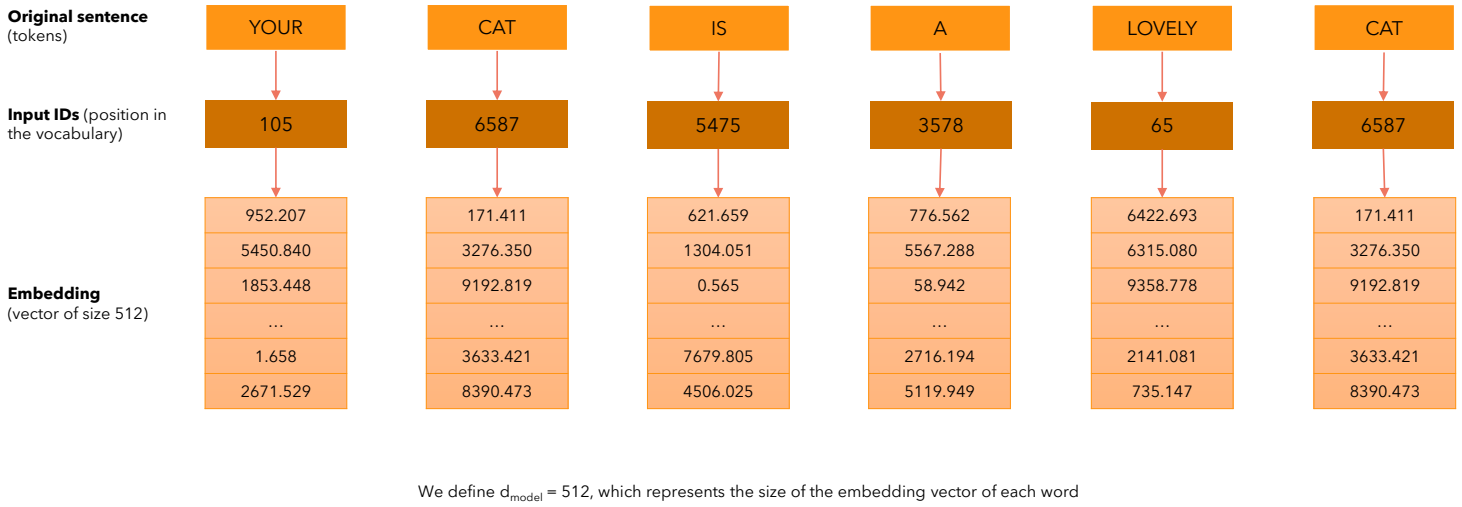
The original paper stacks six of them on top of each other – there's nothing magical about the number six, one can definitely experiment with other arrangements).

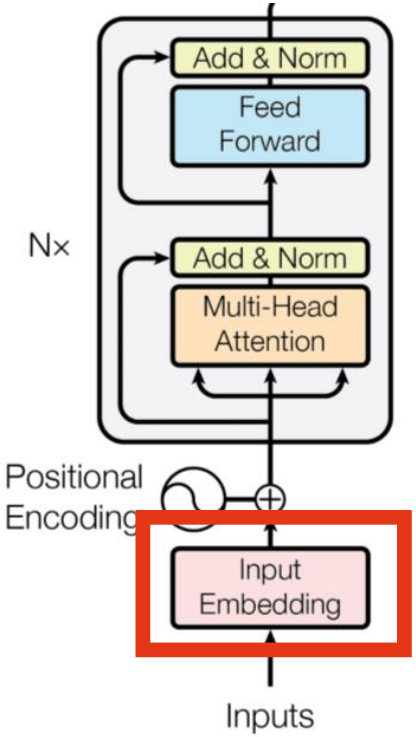
The decoding is a stack of decoders of the same number.



Transformers (Embedding)

Input embedding of words based on dictionary:





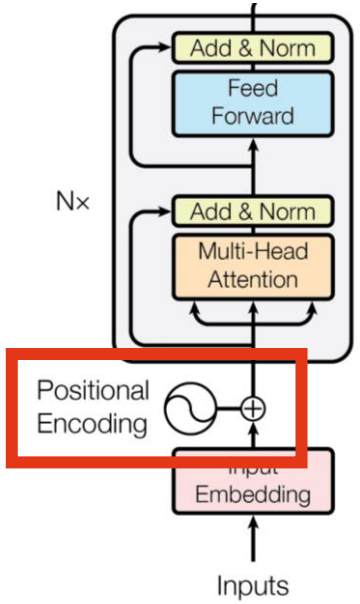
We define d_{model} = 512, which represents the size of the embedding vector of each word

Carnegie Mellon University

Transformers (Pos Encoding)

Positional Encoding

- We want each word to carry some information about its position in the sentence.
- We want the model to treat words that appear close to each other as “close” and words that are distant as “distant”.
- We want the positional encoding to represent a pattern that can be learned by the model.



Carnegie Mellon University

Transformers (Embed+Pos)

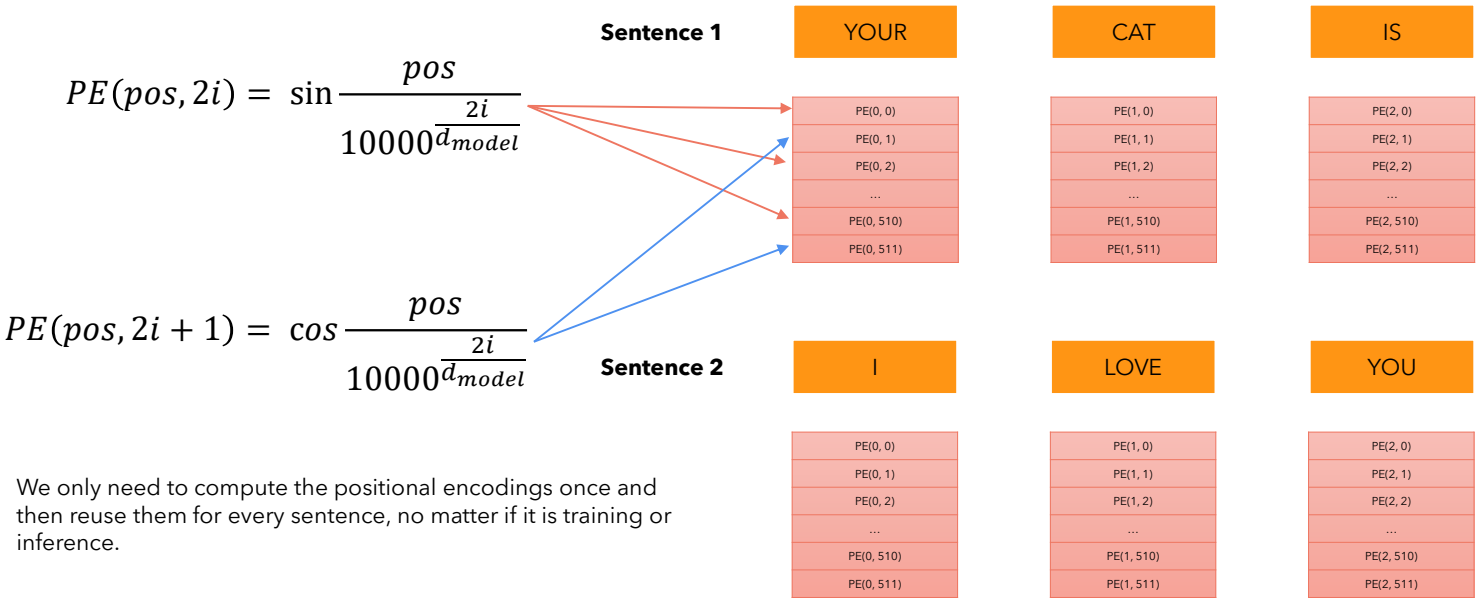
Adding the positional encoding to the embedding

Transformers (Positional Encoding)

What is positional encoding and how is it implemented

$$PE(pos, 2i) = \sin \frac{pos}{10000^{\frac{2i}{d_{model}}}}$$

$$PE(pos, 2i+1) = \cos \frac{pos}{10000^{\frac{2i}{d_{model}}}}$$



We only need to compute the positional encodings once and then reuse them for every sentence, no matter if it is training or inference.

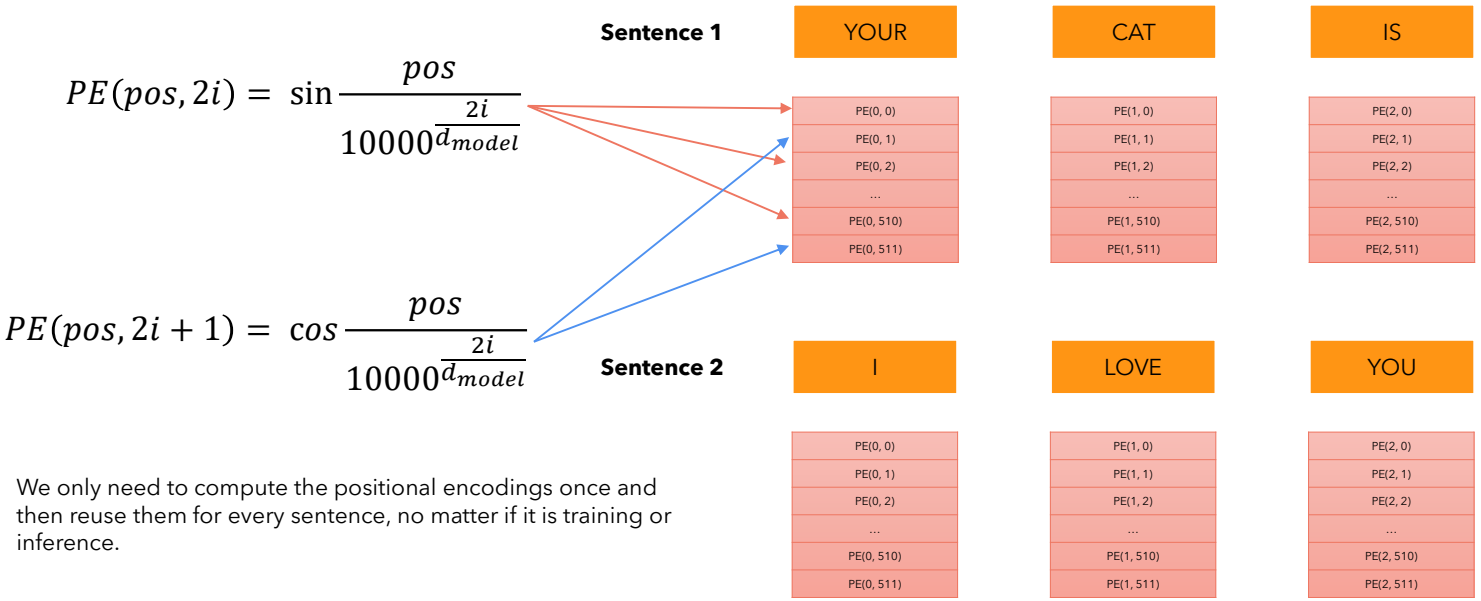
Carnegie Mellon University

Transformers (Positional Encoding)

What is positional encoding and how is it implemented

$$PE(pos, 2i) = \sin \frac{pos}{10000^{\frac{2i}{d_{model}}}}$$

$$PE(pos, 2i+1) = \cos \frac{pos}{10000^{\frac{2i}{d_{model}}}}$$



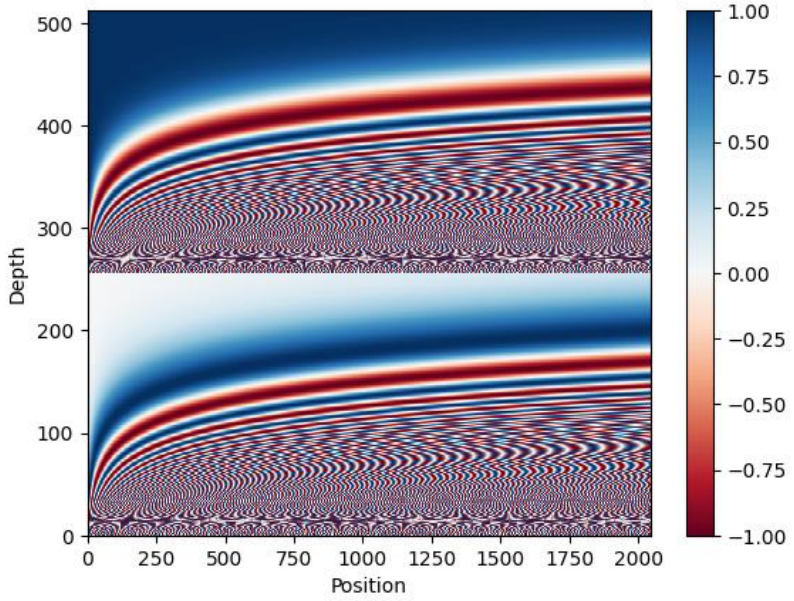
We only need to compute the positional encodings once and then reuse them for every sentence, no matter if it is training or inference.

Carnegie Mellon University

Transformers (Positional Encoding)

What is positional encoding and how is it implemented

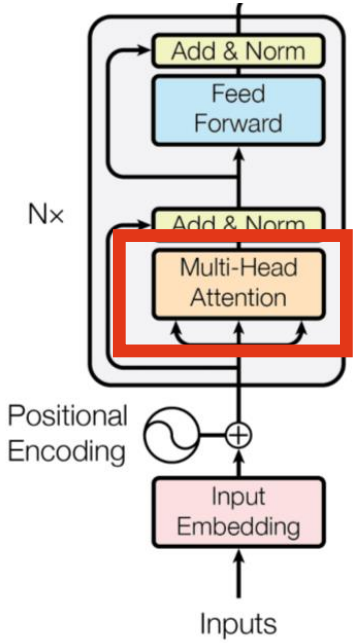
Trigonometric functions like **cos** and **sin** naturally represent a pattern that the model can recognize as continuous, so relative positions are easier to see for the model. By watching the plot of these functions, we can also see a regular pattern, so we can hypothesize that the model will see it too.



Carnegie Mellon University

Transformers (Multi-head Attention)

First we need to learn Self attention



Carnegie Mellon University

Self Attention

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Self-Attention allows the model to relate words to each other.
In this simple case we consider the sequence length **seq** = 6 and $\mathbf{d_{model}} = \mathbf{d_k}$= 512. The matrices **Q**, **K** and **V** are just the input sentence.

$$\text{softmax}\left(\frac{\mathbf{Q}}{\sqrt{512}}\right) \times \mathbf{K^T} =$$

Q: (6, 512)

K^T: (512, 6)

	YOUR	CAT	IS	A	LOVELY	CAT	Σ
YOUR	0.268	0.119	0.134	0.148	0.179	0.152	1
CAT	0.124	0.278	0.201	0.128	0.154	0.115	1
IS	0.147	0.132	0.262	0.097	0.218	0.145	1
A	0.210	0.128	0.206	0.212	0.119	0.125	1
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174	1
CAT	0.195	0.114	0.203	0.103	0.157	0.229	1

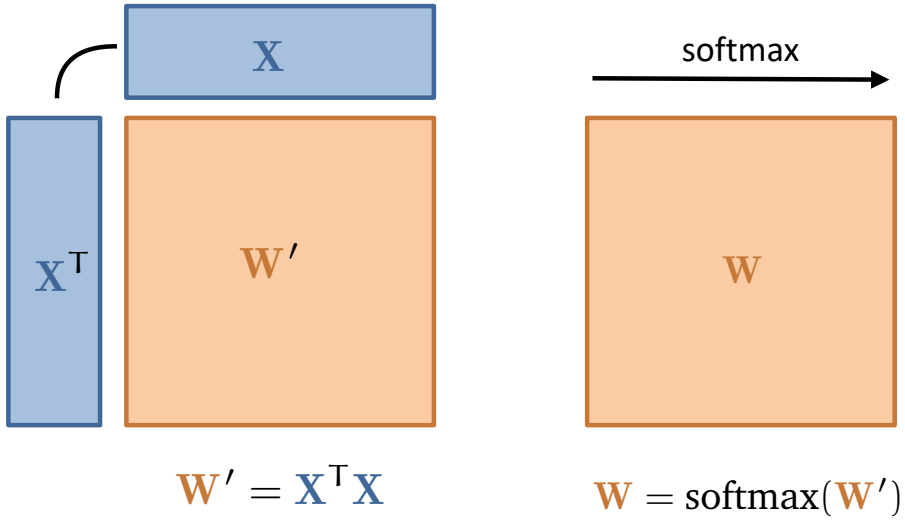
(6, 6)

* all values are random.

Carnegie Mellon University

Self Attention

VECTORIZED



$$w'_{ij} = x_i{}^T x_j$$

$$w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}}$$

Self Attention

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	0.119	0.134	0.148	0.179	0.152
CAT	0.124	0.278	0.201	0.128	0.154	0.115
IS	0.147	0.132	0.262	0.097	0.218	0.145
A	0.210	0.128	0.206	0.212	0.119	0.125
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174
CAT	0.195	0.114	0.203	0.103	0.157	0.229

(6, 6)

X

V

(6, 512)

=

Attention

(6, 512)

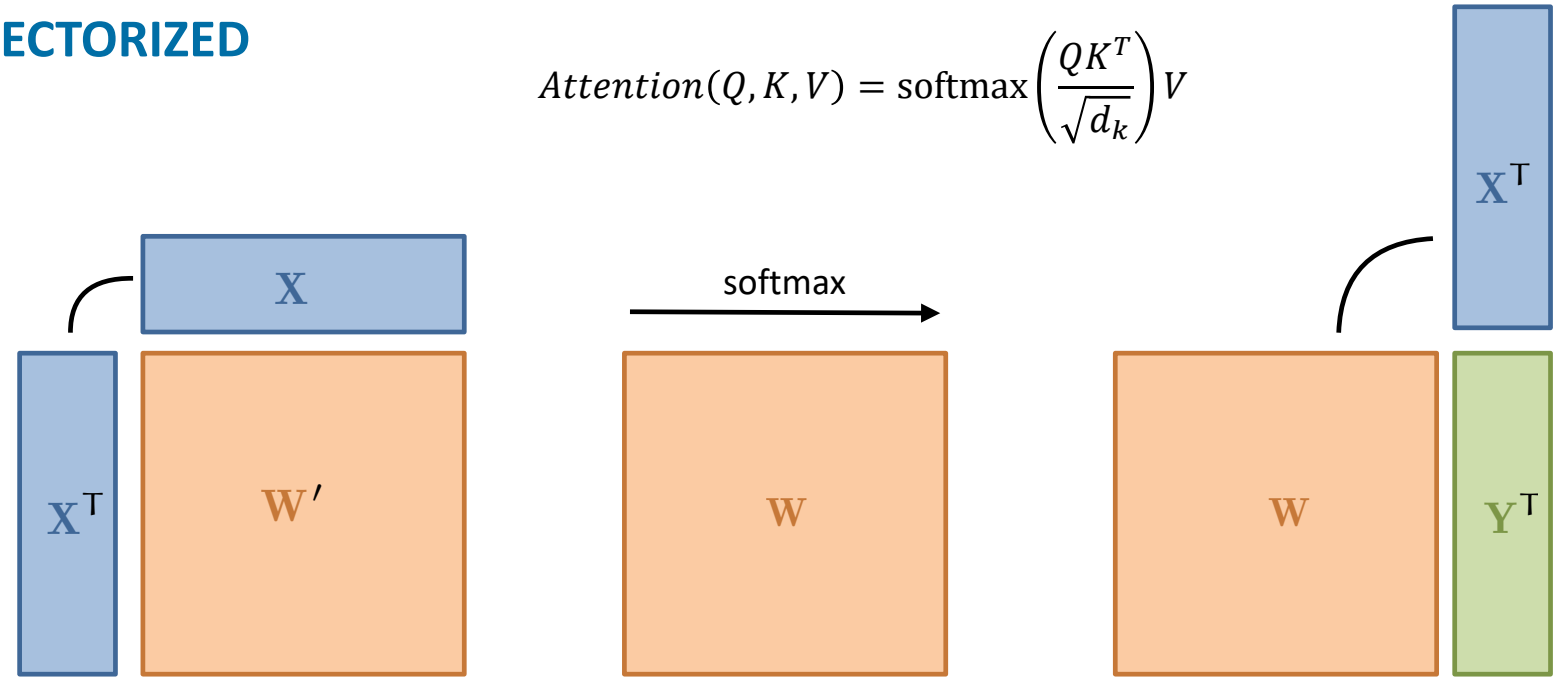
Each row in this matrix captures not only the meaning (given by the embedding) or the position in the sentence (represented by the positional encodings) but also each word's interaction with other words.

Carnegie Mellon University

Self Attention

VECTORIZED

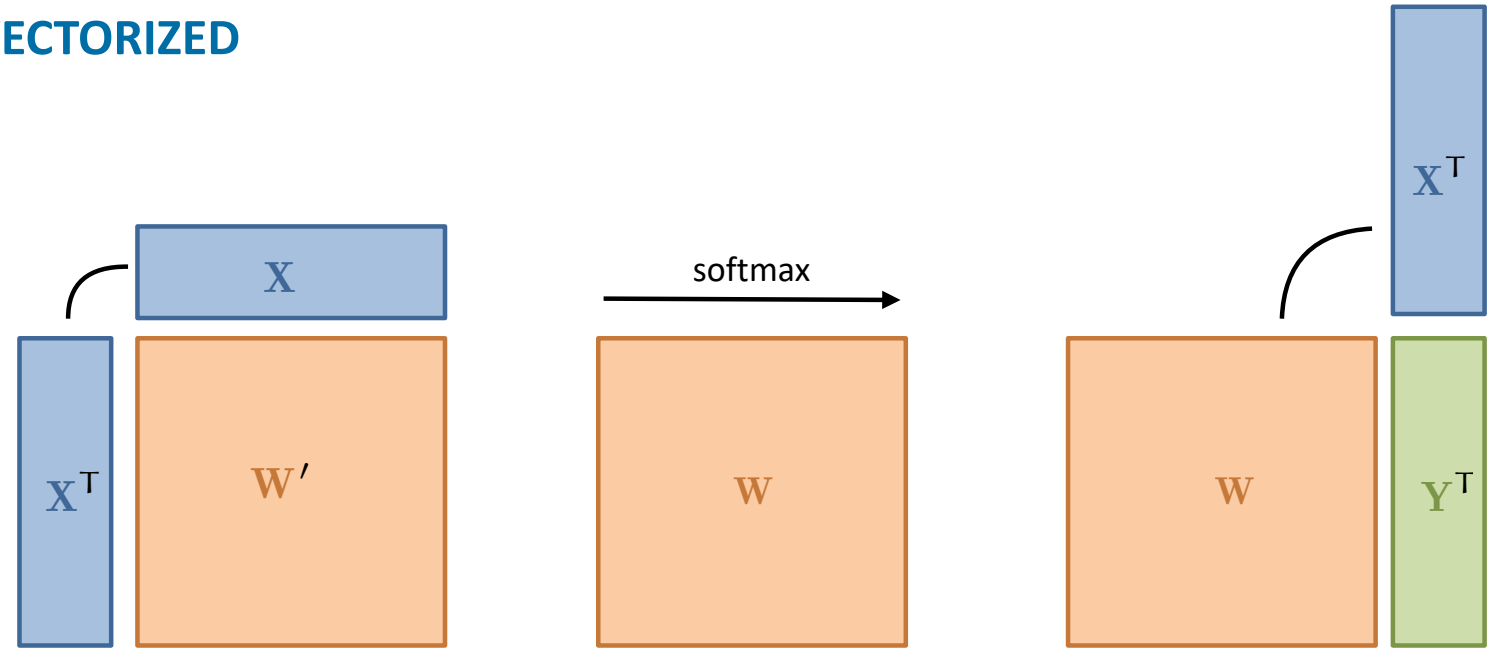
$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



$\mathbf{W}' = \mathbf{X}^T\mathbf{X}$ $\quad \mathbf{W} = \text{softmax}(\mathbf{W}')$ $\quad \mathbf{Y}^T = \mathbf{W}\mathbf{X}^T$

Self Attention

VECTORIZED



$$W' = X^T X$$

$$W = \text{softmax}(W')$$

$$Y^T = W X^T$$

Carnegie Mellon University

Self Attention

•Self-Attention is permutation invariant.

•Self-Attention requires no parameters. Up to now the interaction between words has been driven by their embedding and the positional encodings. This will change later.

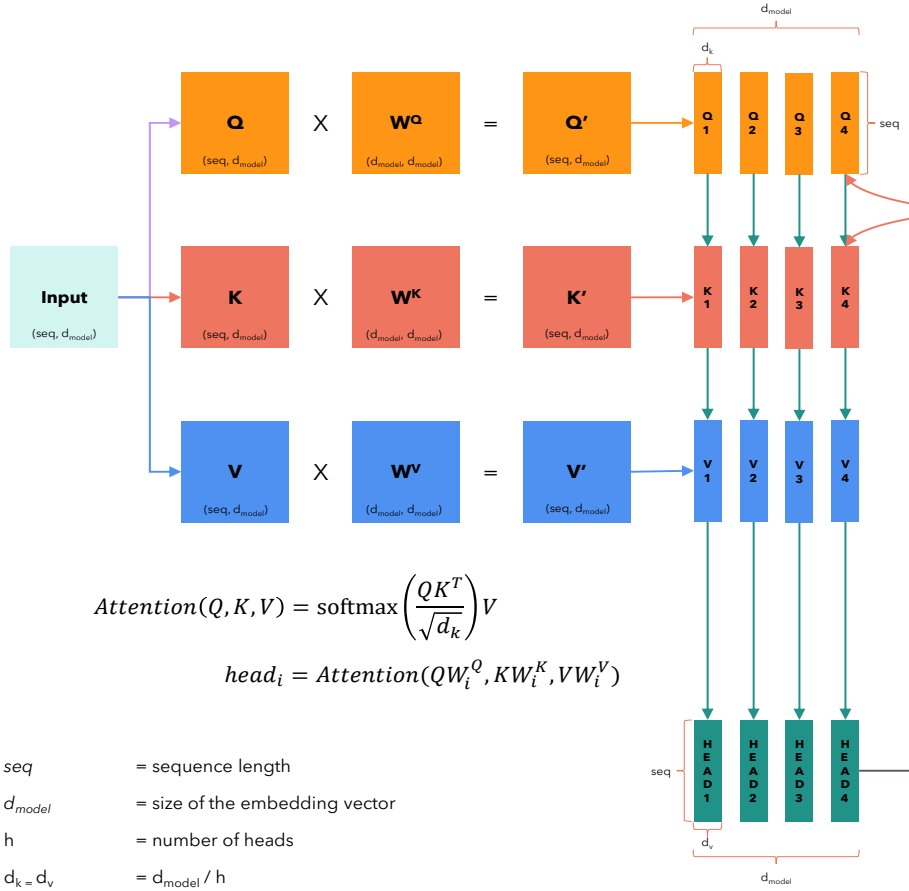
•We expect values along the diagonal to be the highest.

•If we don't want some positions to interact, we can always set their values to $-\infty$ before applying the *softmax* in this matrix and the model will not learn those interactions. We will use this in the decoder.

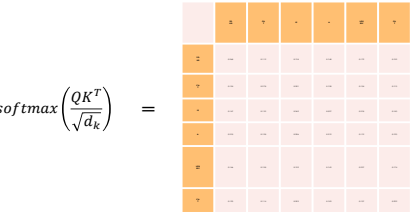
	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	0.119	0.134	0.148	0.179	0.152
CAT	0.124	0.278	0.201	0.128	0.154	0.115
IS	0.147	0.132	0.262	0.097	0.218	0.145
A	0.210	0.128	0.206	0.212	0.119	0.125
LOVELY	0.146	0.158	0.152	0.143	0.227	0.174
CAT	0.195	0.114	0.203	0.103	0.157	0.229

Carnegie Mellon University

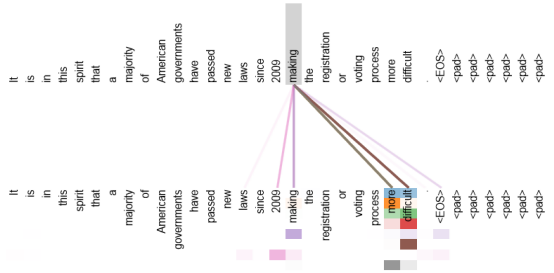
Multi-head Attention



$$softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) =$$

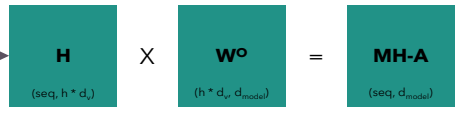


Attention Visualizations



$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$



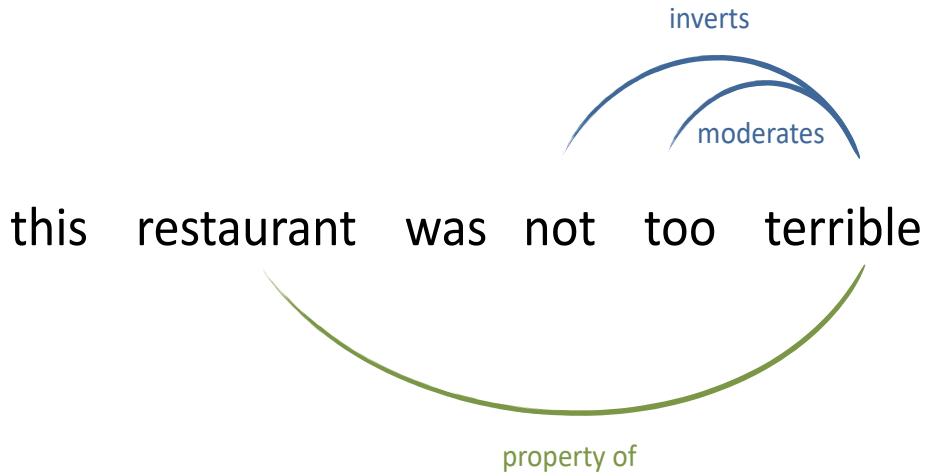
seq	= sequence length
d_{model}	= size of the embedding vector
h	= number of heads
$d_k = d_v$	= d_{model} / h

Carnegie Mellon University

Attention Learns functional relation via Multihead

In many sentences, there are different relations to model. Here, the word meaning of the word “terrible” is inverted by “not” and moderated by “too”. Its relation to the word restaurant is completely different: it describes a property of the restaurant.

The idea behind multi-head self-attention is that multiple relations are best captured by different self-attention operations.



Attention as a Soft Dictionary

- **Attention** is a *soft* dictionary
- key, query and value are vectors
- every key matches the query *to some extent* as determined by their dot-product
- a *mixture* of all values is returned with softmax-normalized dot products as mixture weights
- ***Self*-attention** Attention with keys, queries and values from the same set.

```
d = {'a' : 1, 'b' : 2, 'c' : 3}

d['b'] = 3
```

<- query

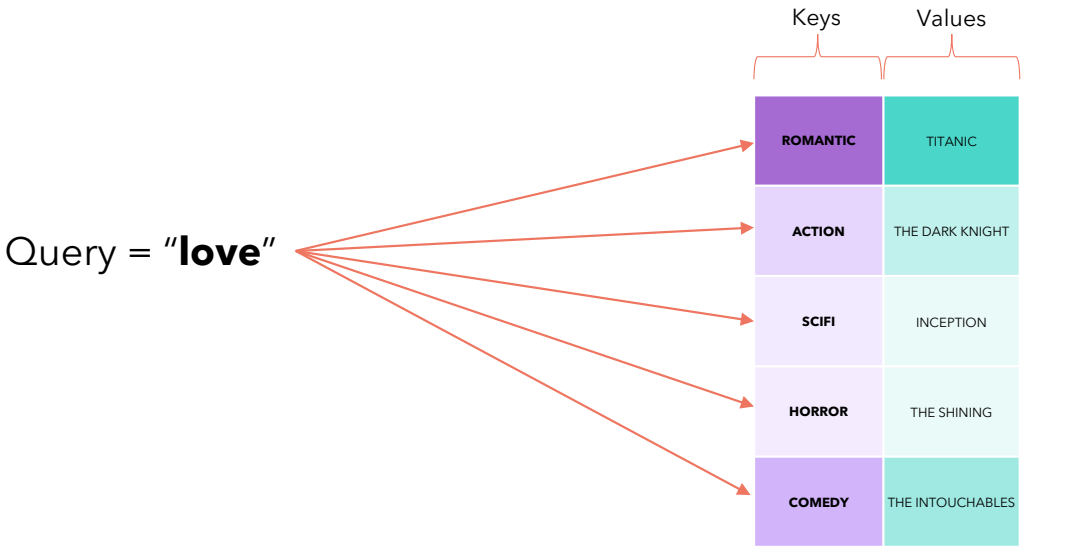
<- key

<- value

key	value
a	1
b	2
c	3

Carnegie Mellon University

K, Q, V interpretation



* this could be a Python dictionary or a database table.

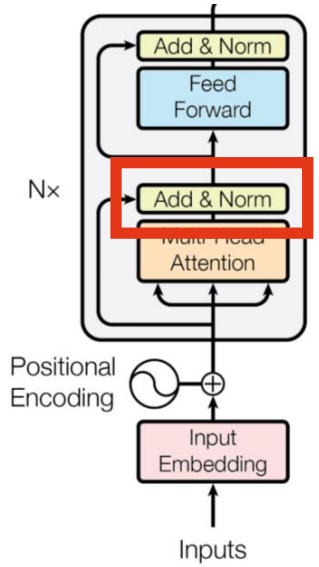
Carnegie Mellon University

Add&Norm

Batch of 3 items

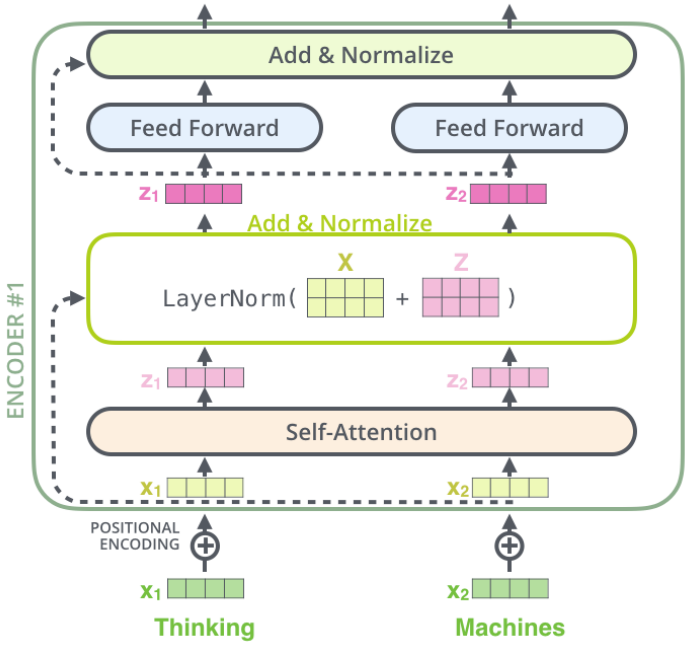
ITEM 1	ITEM 2	ITEM 3
50.147	1242.223	9.370
3314.825	688.123	4606.674
...	...	...
...	...	...
8463.361	434.944	944.705
8.021	149.442	21189.444
μ_1	μ_2	μ_3
σ_1^2	σ_2^2	σ_3^2

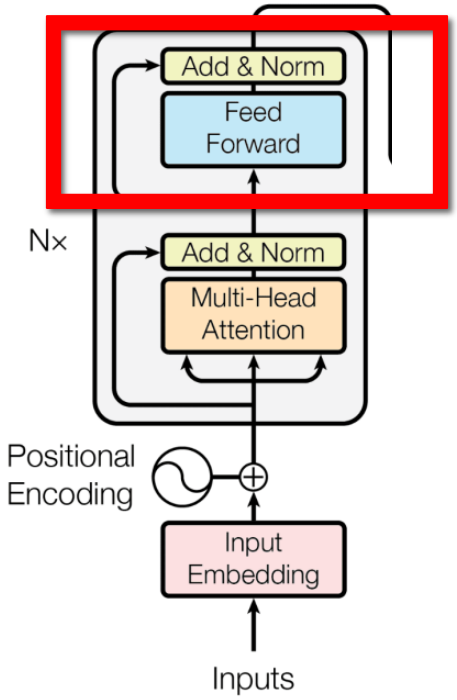
$$\hat{x}_j = \frac{x_j - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$



Carnegie Mellon University

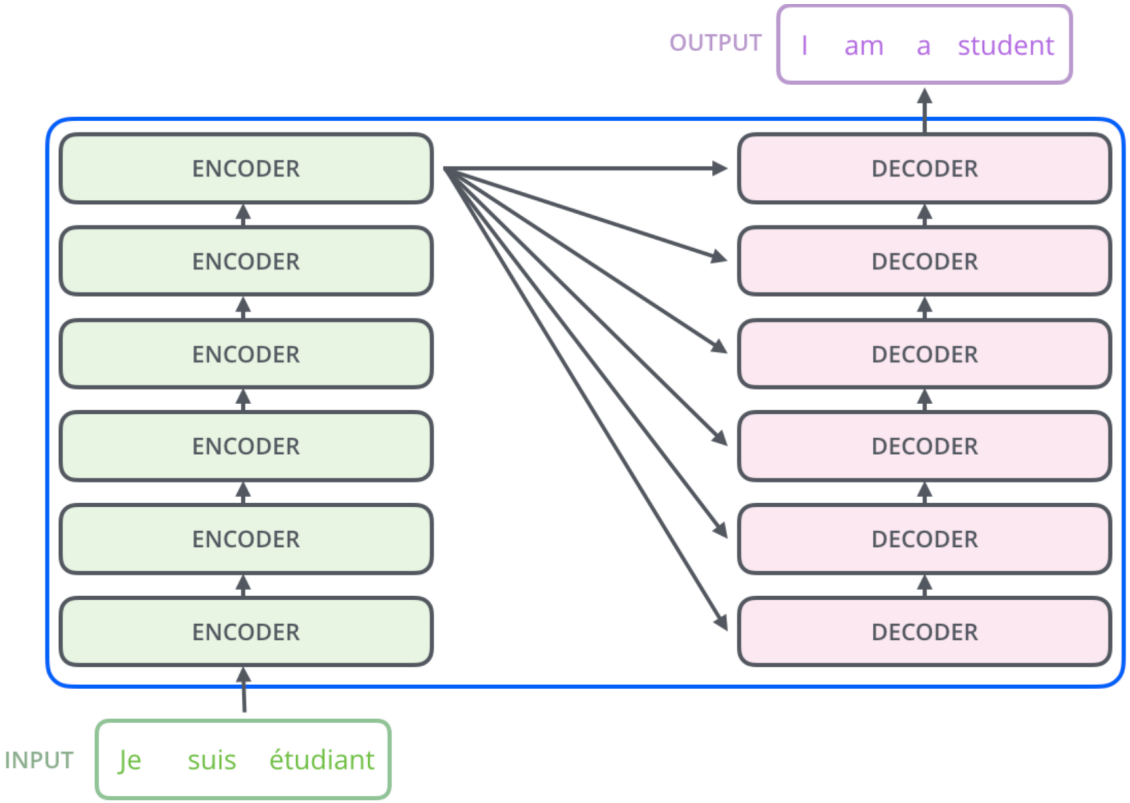
Feed Forward NN





Carnegie Mellon University

Encoders on top of each other



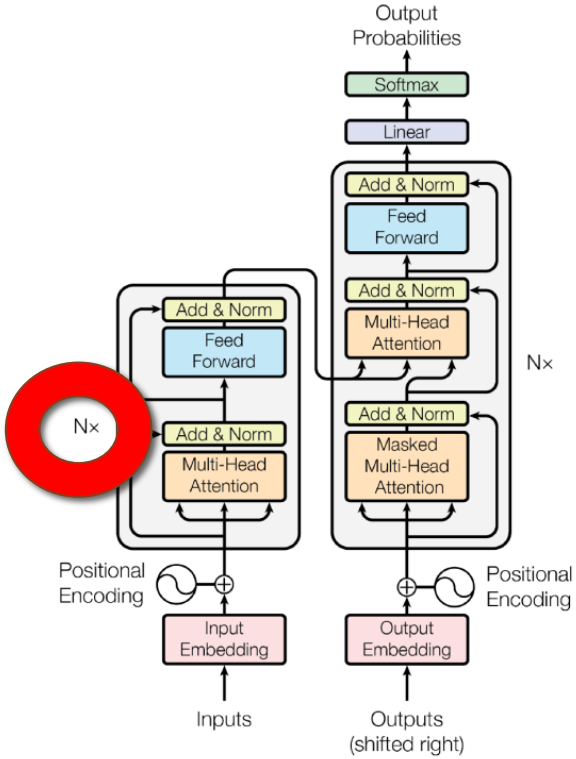


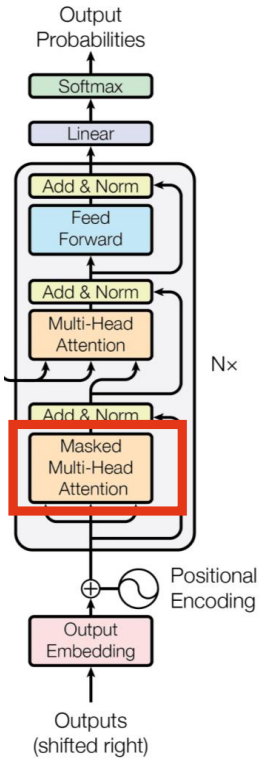
Figure 1: The Transformer - model architecture.

Carnegie Mellon University

Decoder

What is Masked Multi-Head Attention?

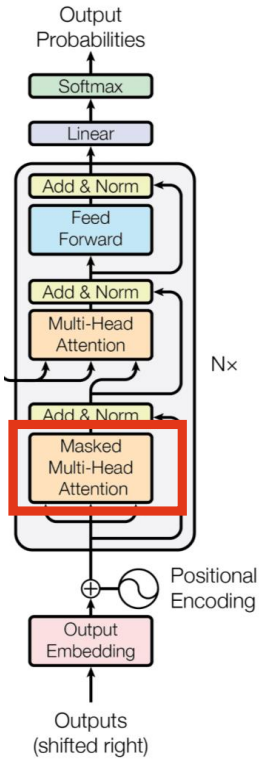
Our goal is to make the model causal: it means the output at a certain position can only depend on the words on the previous positions. The model **must not** be able to see future words.



Decoder

What is Masked Multi-Head Attention?

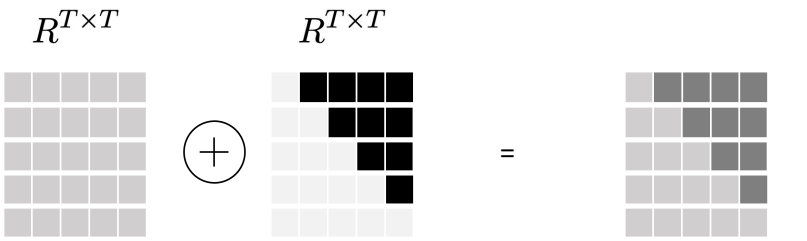
	YOUR	CAT	IS	A	LOVELY	CAT
YOUR	0.268	~~0.~~	~~0.134~~	~~0.148~~	~~0.179~~	~~0.152~~
CAT	0.124	0.278	~~0.~~	~~0.128~~	~~0.154~~	~~0.115~~
IS	0.147	0.132	0.262	~~0.097~~	~~0.218~~	~~0.145~~
A	0.210	0.128	0.206	0.212	~~0.119~~	~~0.125~~
LOVELY	0.146	0.158	0.152	0.143	0.227	~~0.174~~
CAT	0.195	0.114	0.203	0.103	0.157	0.229

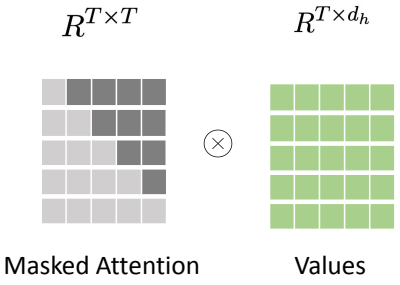


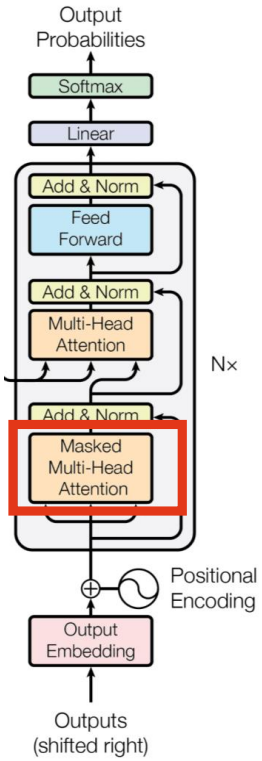
Carnegie Mellon University

Decoder

What is Masked Multi-Head Attention?

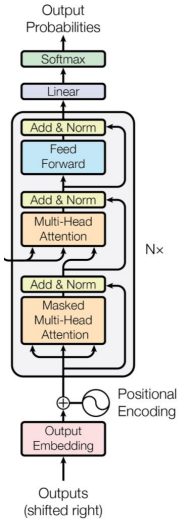


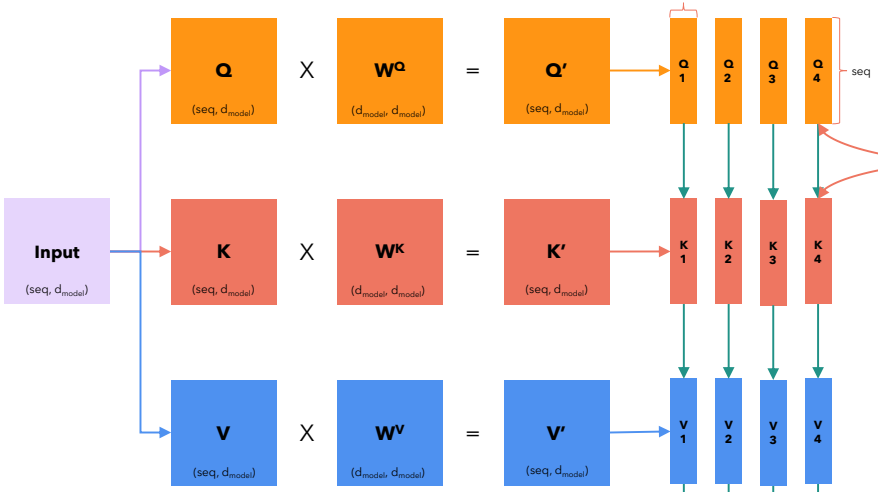


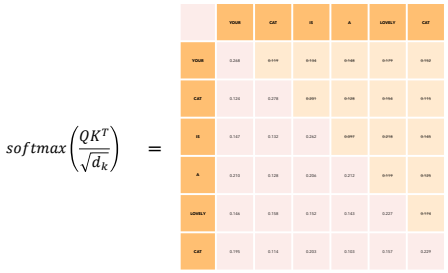


Carnegie Mellon University

Decoder



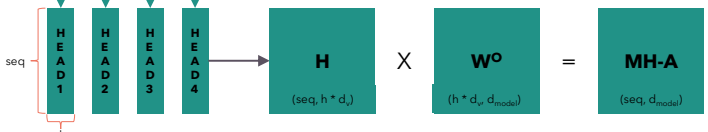




All the values above the diagonal are replace with **-∞** *before* applying the softmax, which will replace them with zero.

$$Attention(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$



$$MultiHead(Q,K,V) = Concat(head_1 \ldots head_h)W^O$$

seq = sequence length

d_{model} = size of the embedding vector

h = number of heads

Carnegie Mellon University

Training

I love you very much

Ti amo molto

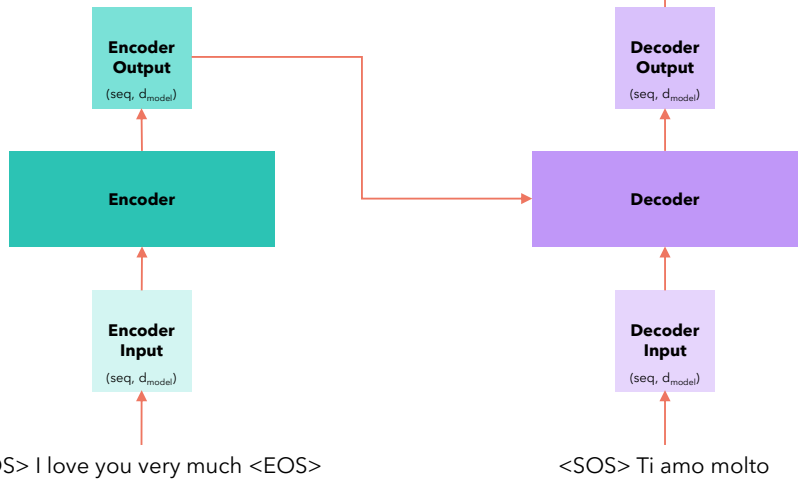
Training

Training

Time Step = 1

It all happens in one time step!

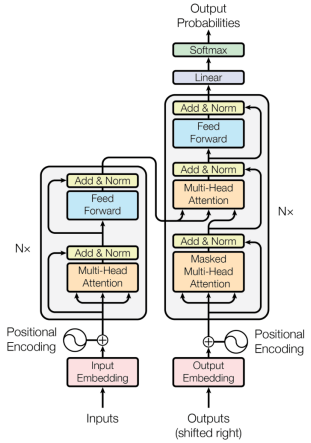
The encoder outputs, for each word a vector that not only captures its meaning (the embedding) or the position, but also its interaction with other words by means of the multi-head attention.



Ti amo molto <EOS>

* This is called the "label" or the "target"

Cross Entropy Loss



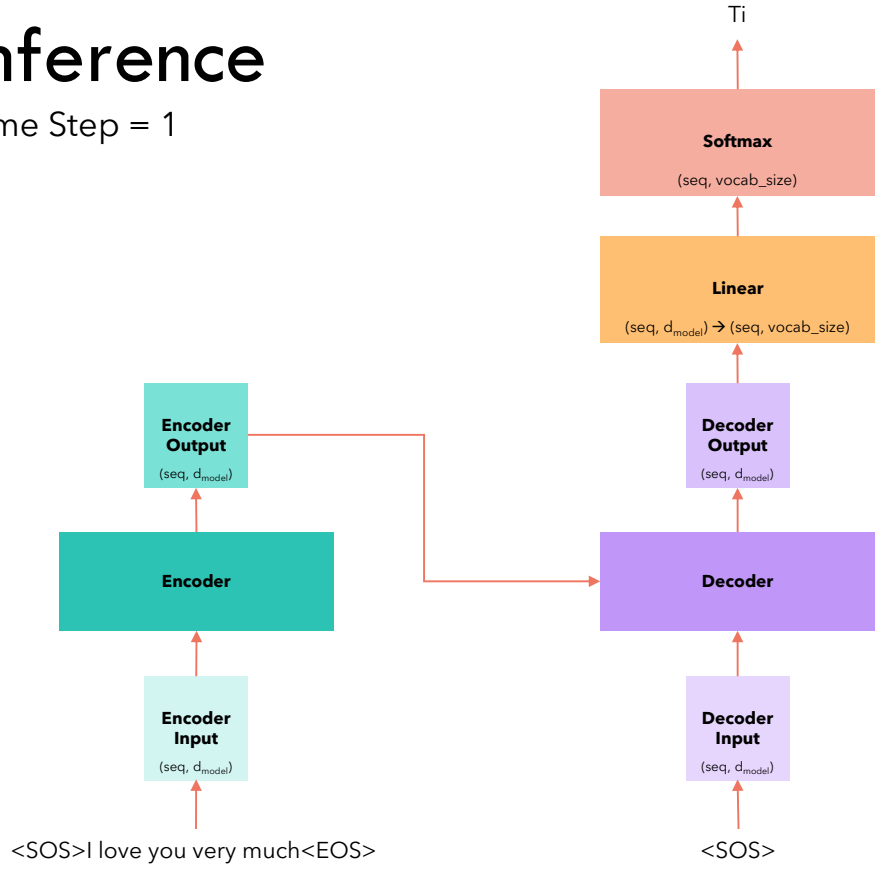
We prepend the <SOS> token at the beginning. That's why the paper says that the decoder input is shifted right.

Carnegie Mellon University

Inference

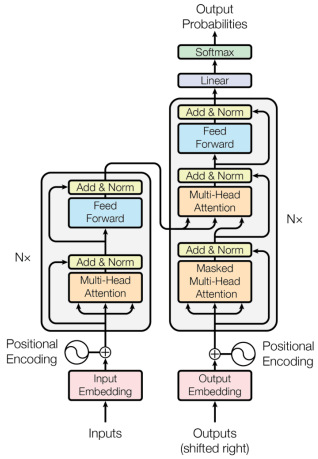
Inference

Time Step = 1



We select a token from the vocabulary corresponding to the position of the token with the maximum value.

The output of the last layer is commonly known as **logits**



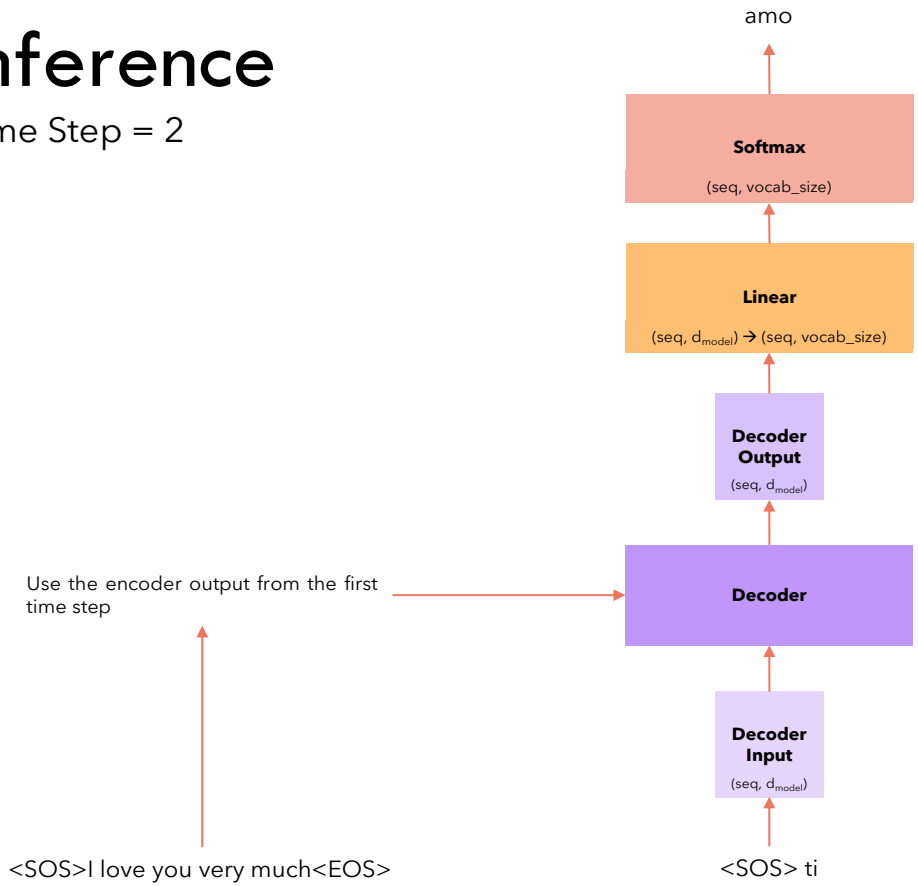
* Both sequences will have same length thanks to padding

Carnegie Mellon University

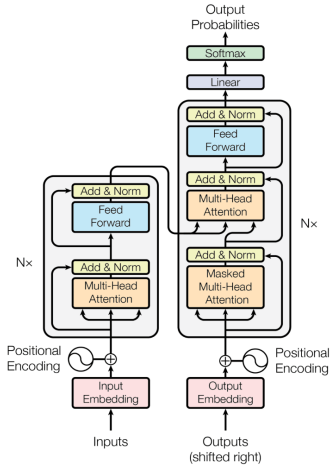
Inference

Inference

Time Step = 2



Since decoder input now contains **two** tokens, we select the softmax corresponding to the second token.



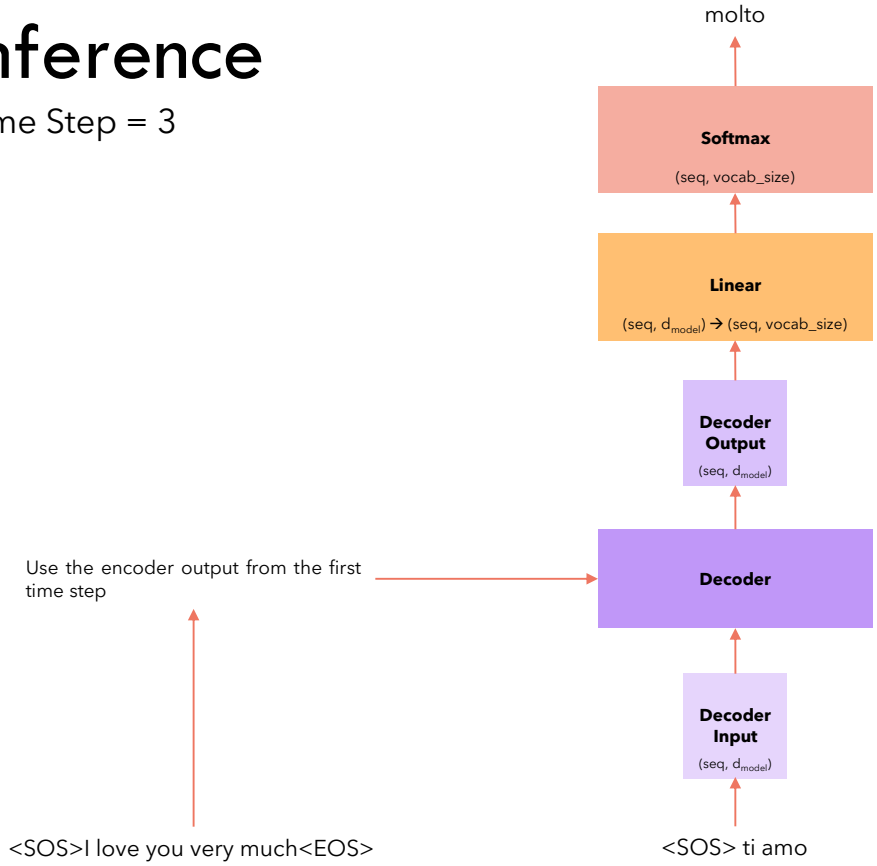
Append the previously output word to the decoder input

Carnegie Mellon University

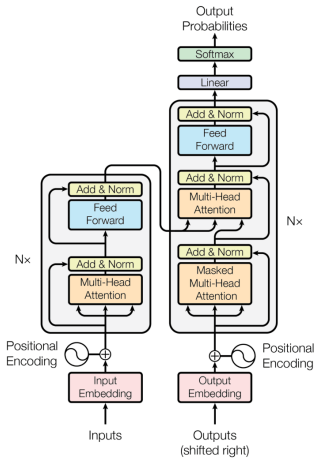
Inference

Inference

Time Step = 3



Since decoder input now contains **three** tokens, we select the softmax corresponding to the third token.



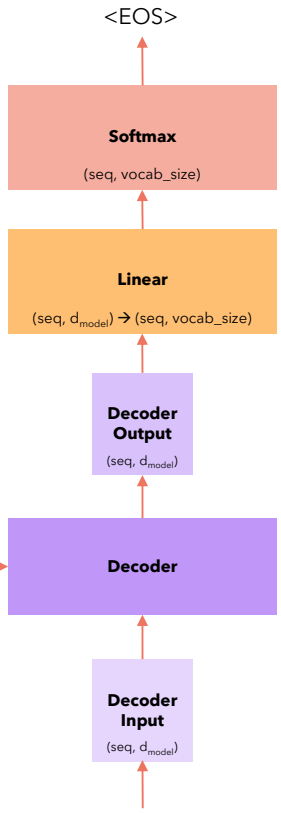
Append the previously output word to the decoder input

Carnegie Mellon University

Inference

Inference

Time Step = 4

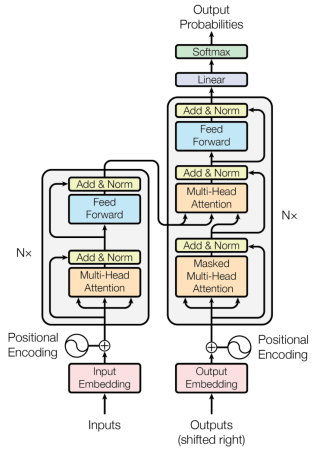


Use the encoder output from the first time step

<SOS>I love you very much<EOS>

<SOS> ti amo molto

Since decoder input now contains **four** tokens, we select the softmax corresponding to the fourth token.



Append the previously output word to the decoder input

Carnegie Mellon University