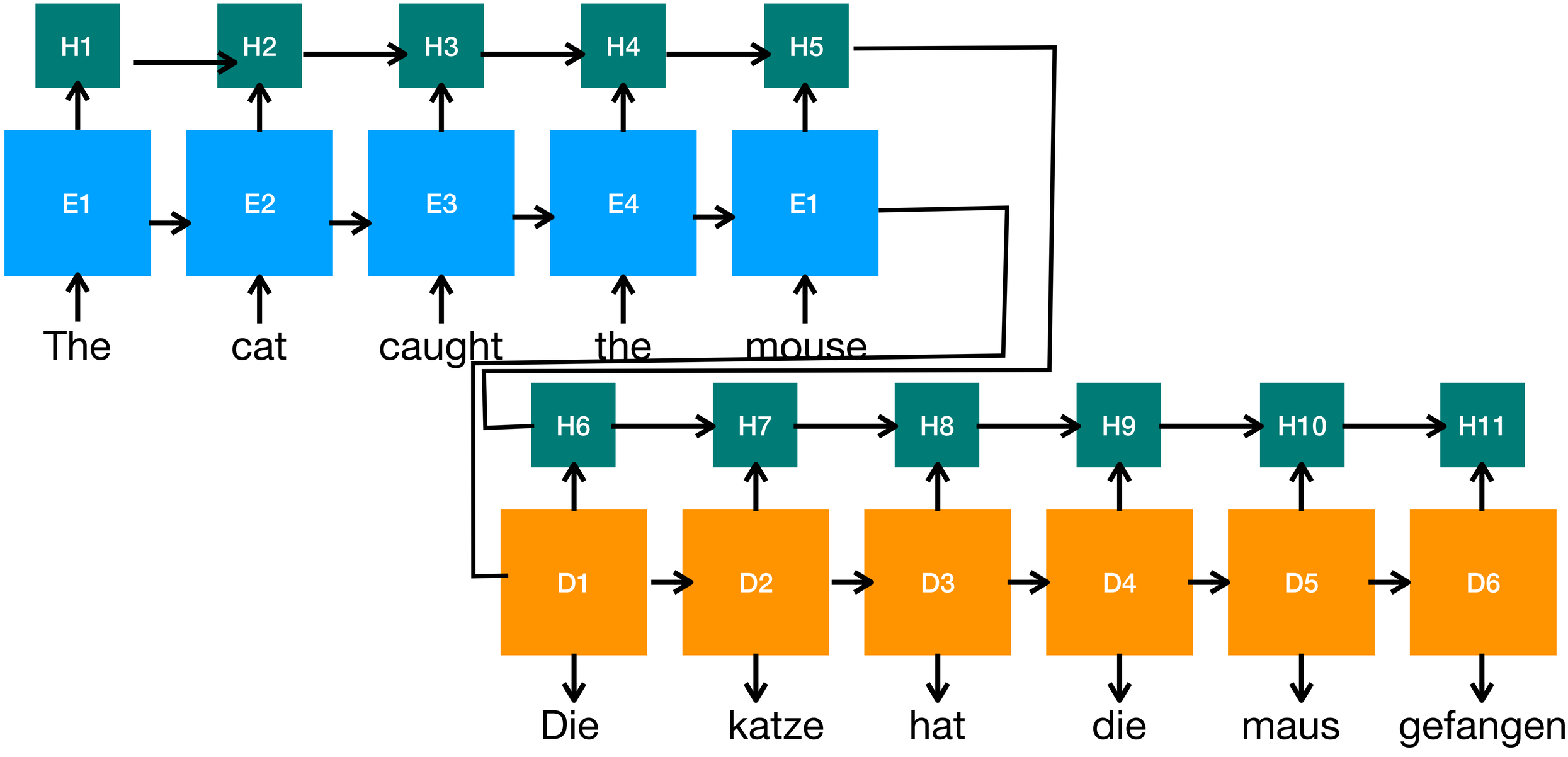
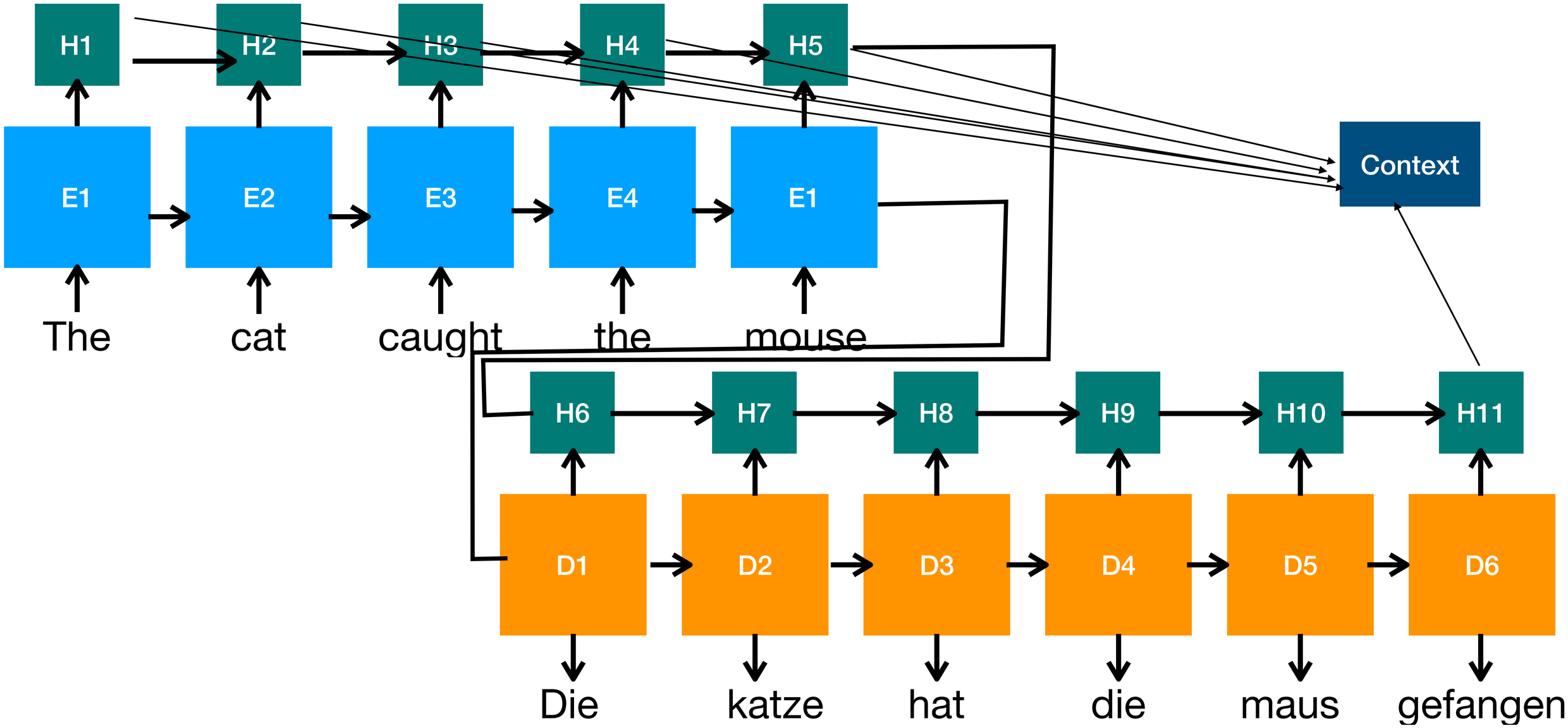
Understanding Transformers

Vinay P. Namboodiri

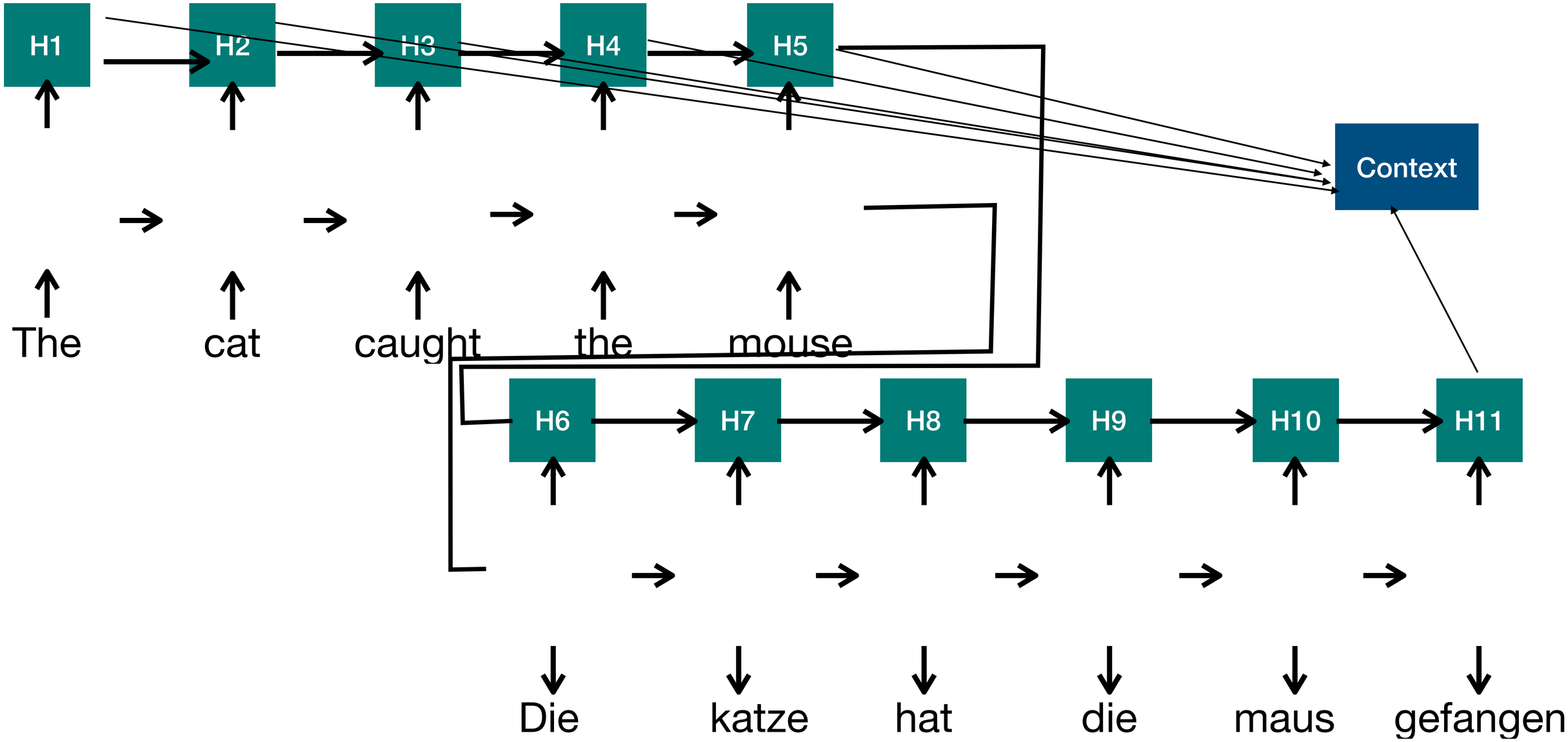
Machine Translation using LSTM



LSTM with attention



Main idea of transformers - Attention is all you need



Why 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

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-

Attention is all you need

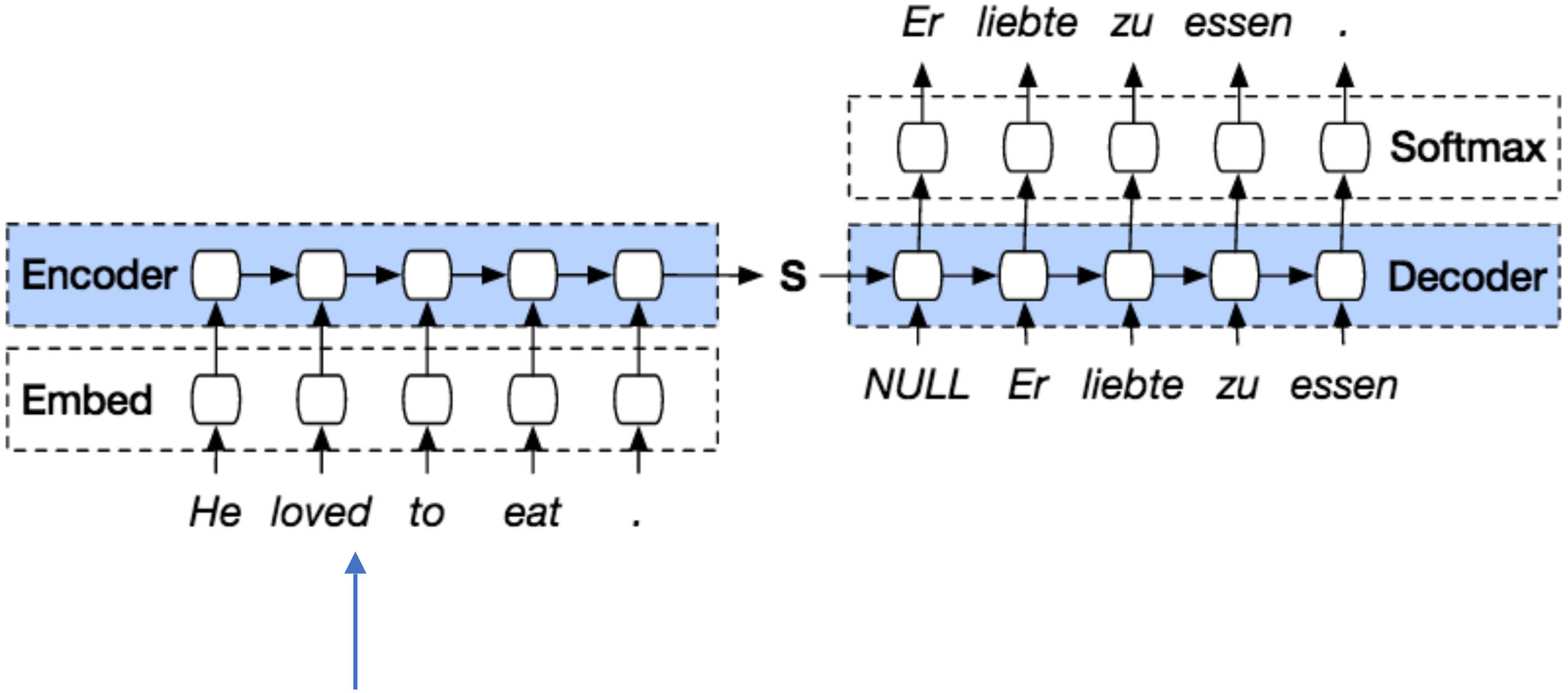
A Vaswani, N Shazeer, N Parmar… - Advances in neural …, 2017 - proceedings.neurips.cc

The dominant sequence transduction models are based on complex recurrent orconvolutional neural networks in an encoder and decoder configuration. The best …

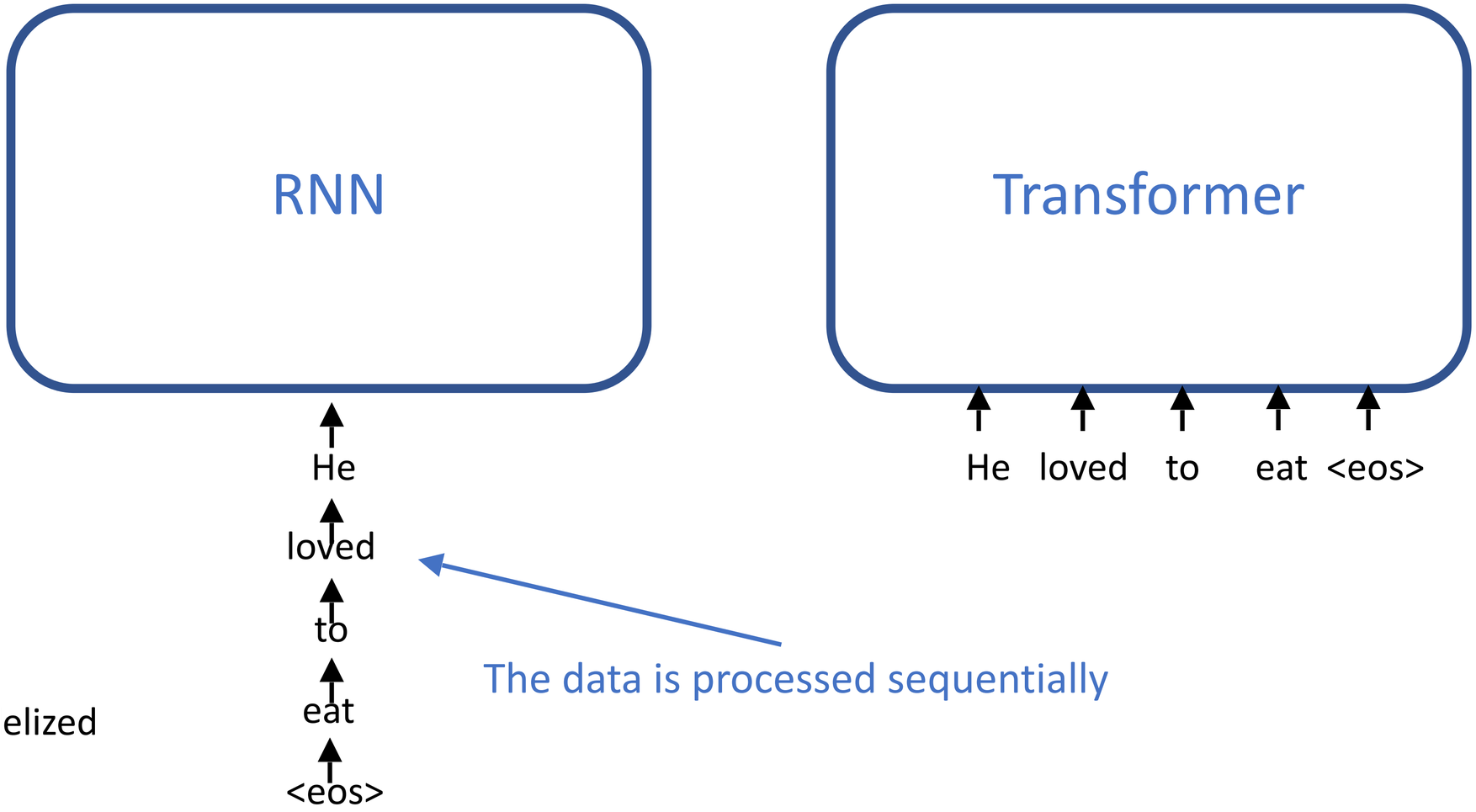
☆ Save 99 Cite Cited by 116698 Related articles »

Vaswani et al. “Attention is all you need”. NIPS 2017.

Neural Machine Translation with RNNs (LSTM, GRU)

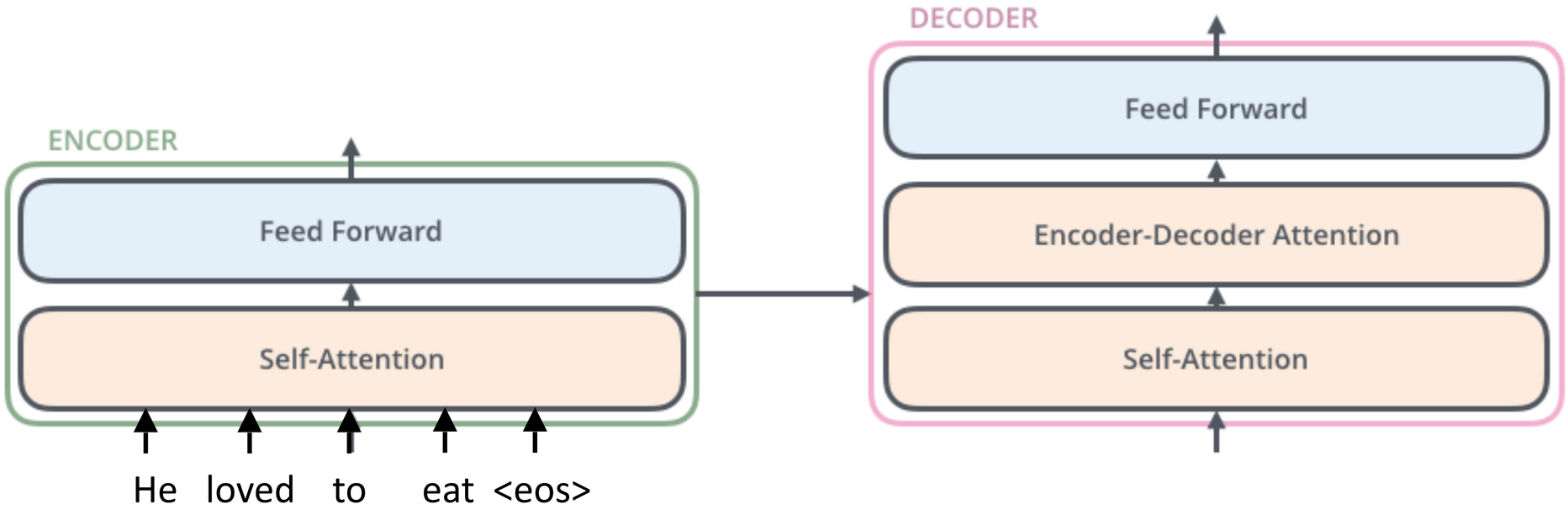


Limitation of RNNs (LSTM, GRU)



- Can’t be parallelized
- Slow to train
- Transfer learning can hardly to be used, i.e., need specific labelled dataset to your task

Neural Machine Translation with Transformer

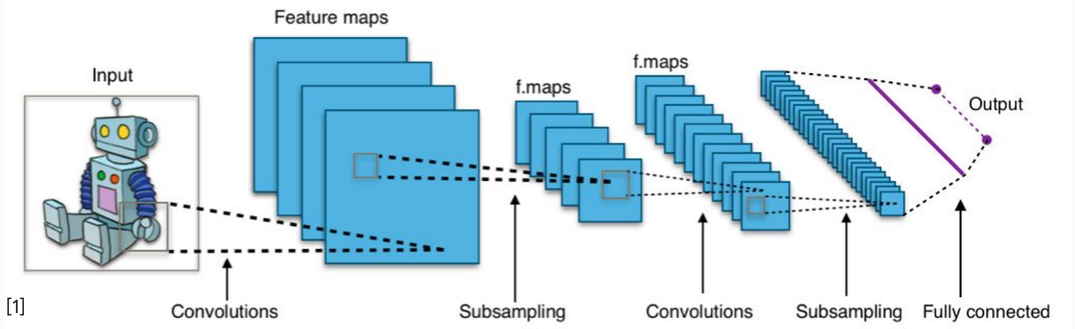


- Discard the recurrent structure
- Solely based on attention mechanism
- Parallelizable, train faster
- Can be pre-trained

Image from: http://jalammar.github.io/illustrated-transformer/

Computer Vision

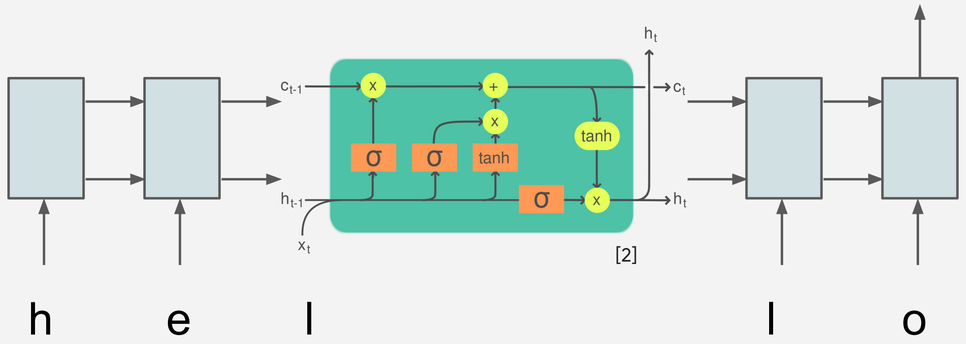
Convolutional NNs (+ResNets)



[1]

Natural Lang. Proc.

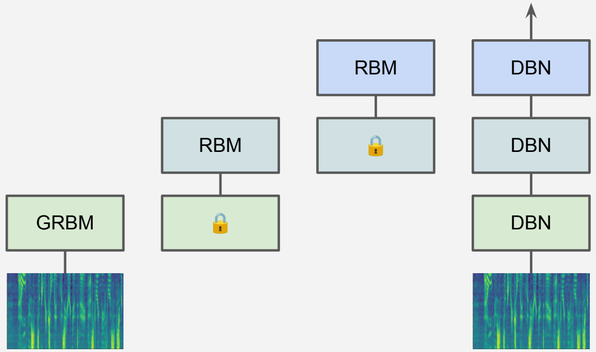
Recurrent NNs (+LSTMs)



[2]

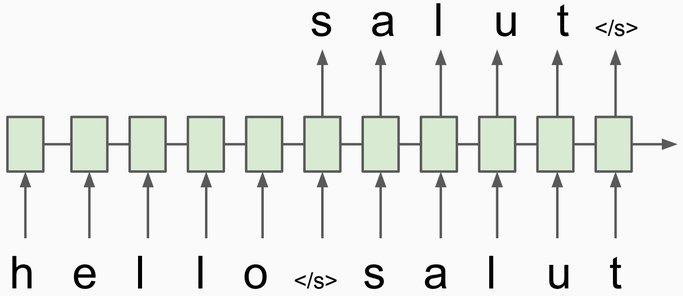
Speech

Deep Belief Nets (+non-DL)



Translation

Seq2Seq



RL

BC/GAIL

Algorithm 1 Generative adversarial imitation learning

1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
2: **for** $i = 0, 1, 2, \ldots$ **do**
3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))] \tag{17}$$

5: Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

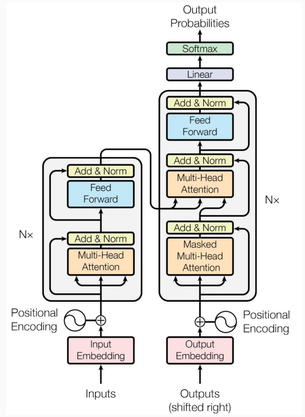
$$\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta),$$
$$\text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}] \tag{18}$$

6: **end for**

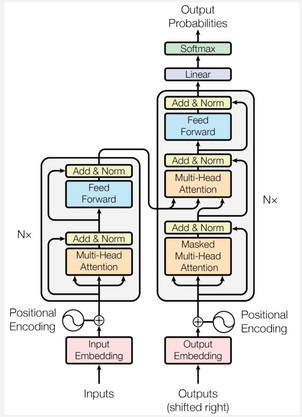
[1] CNN image CC-BY-SA by Aphex34 for Wikipedia https://commons.wikimedia.org/wiki/File:Typical_cnn.png
[2] RNN image CC-BY-SA by GChe for Wikipedia https://commons.wikimedia.org/wiki/File:The_LSTM_Cell.svg

Slide credit Lucas Beyer

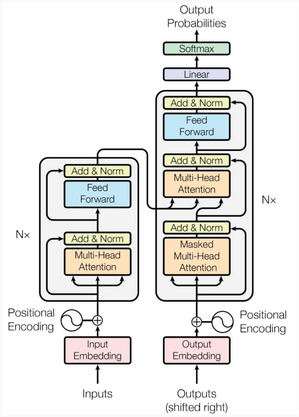
Computer Vision



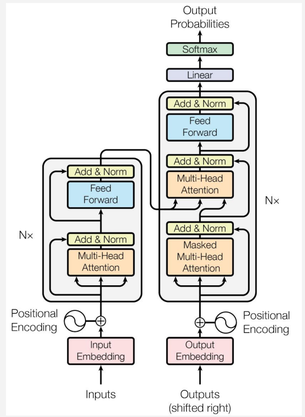
Natural Lang. Proc.



Reinf. Learning

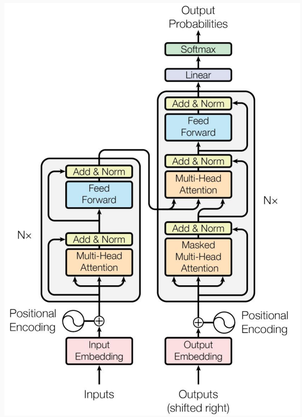


Speech

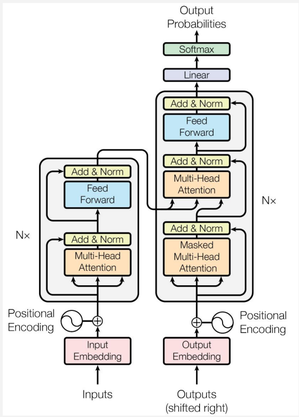


sformer image source: "Attention Is All You Need" paper

Translation



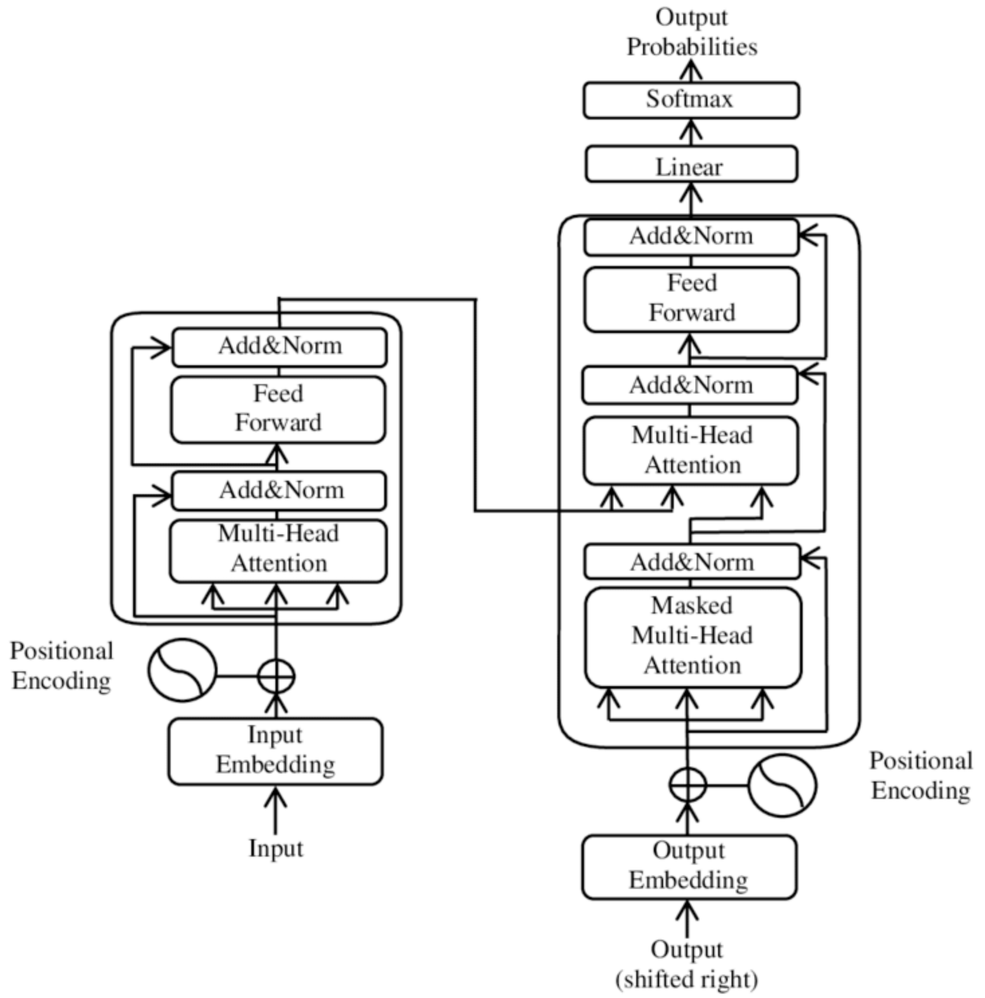
Graphs/Science



Slide credit Lucas Beyer

What is the transformer?

- A ‘simple’ architecture with few operation
- Embedding
- Positional Encoding
- Layer-normalization
- Multi-headed attention
- MLP
- Residual Layer



The Attention Layer

Self-Attention

- **Intuition**: to bake the “understanding” of other relevant words into the one we are currently processing.

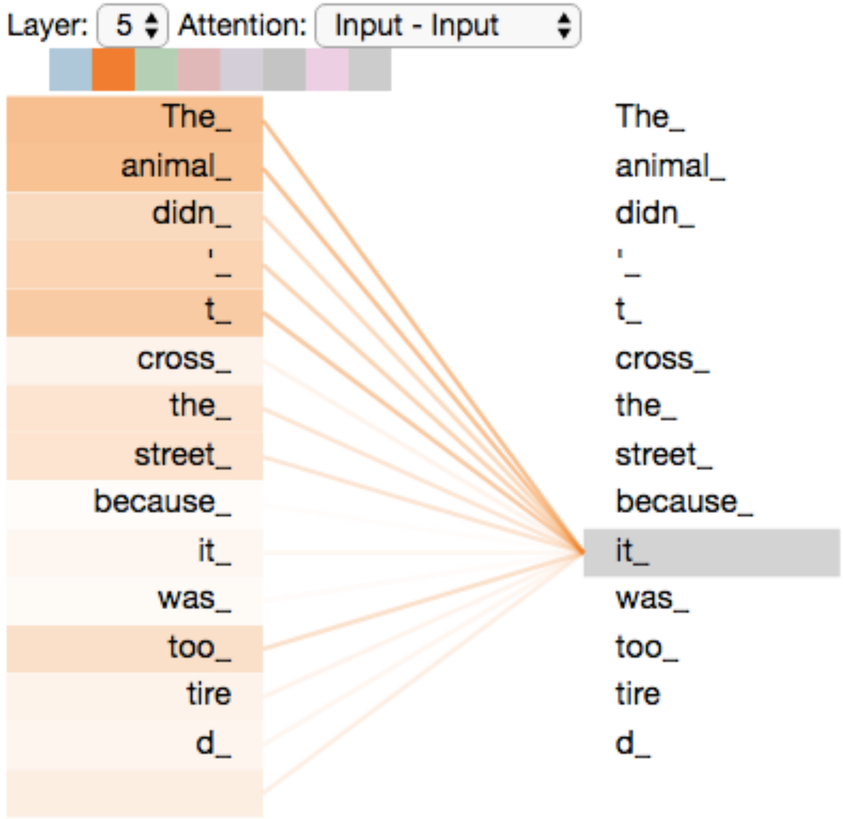
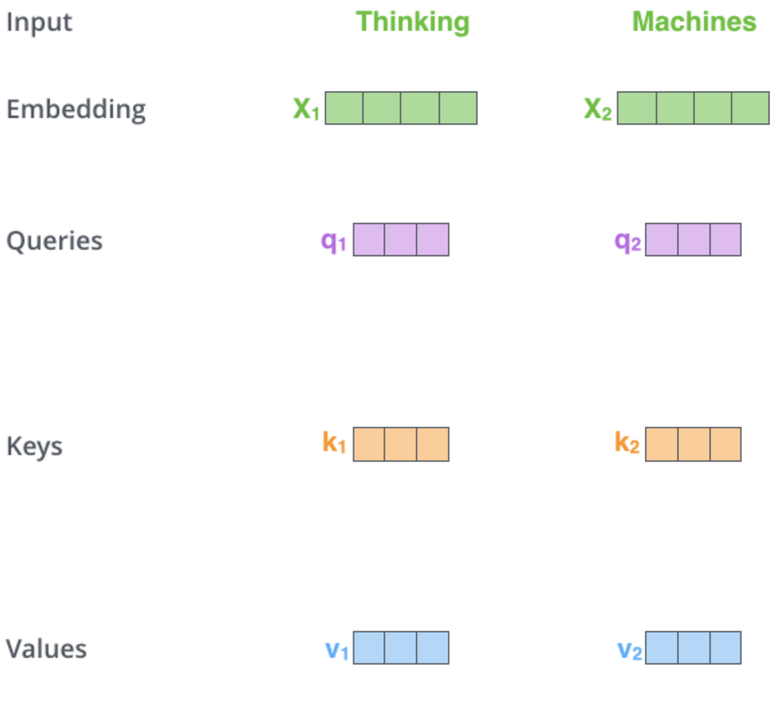
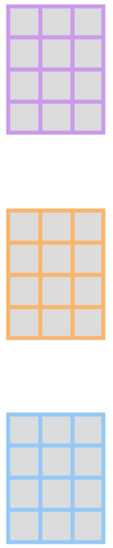


Image from: http://jalammar.github.io/illustrated-transformer/

STEP 1: create three vectors by multiplying the embedding with three weight matrices (will be learnt during training)

Self-Attention





W^Q

$$q_i = W^Q x_i$$

W^K

$$k_i = W^K x_i$$

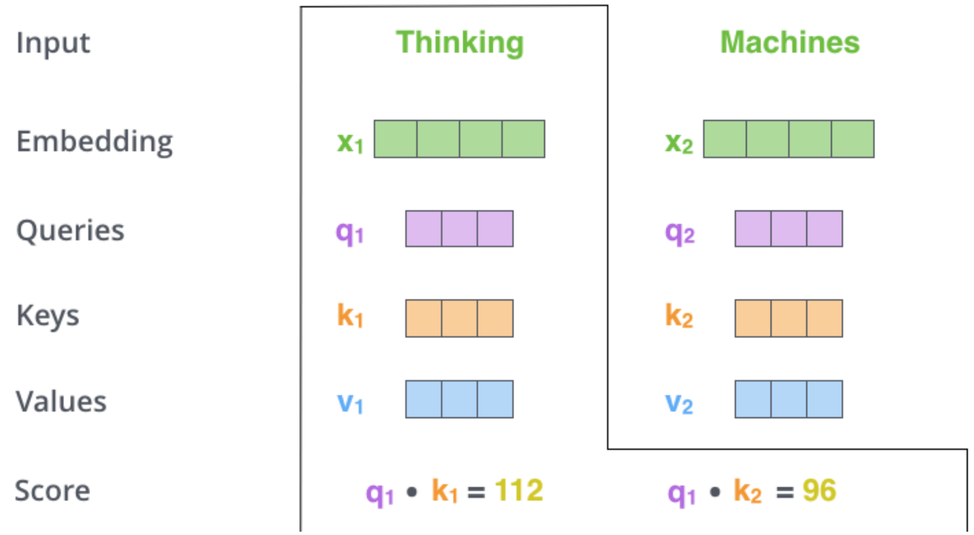
W^V

$$v_i = W^V x_i$$

Image from: http://jalammar.github.io/illustrated-transformer/

STEP 2: Calculate the **attention score** which determines how much focus to place on other words as we encode one word.

Self-Attention

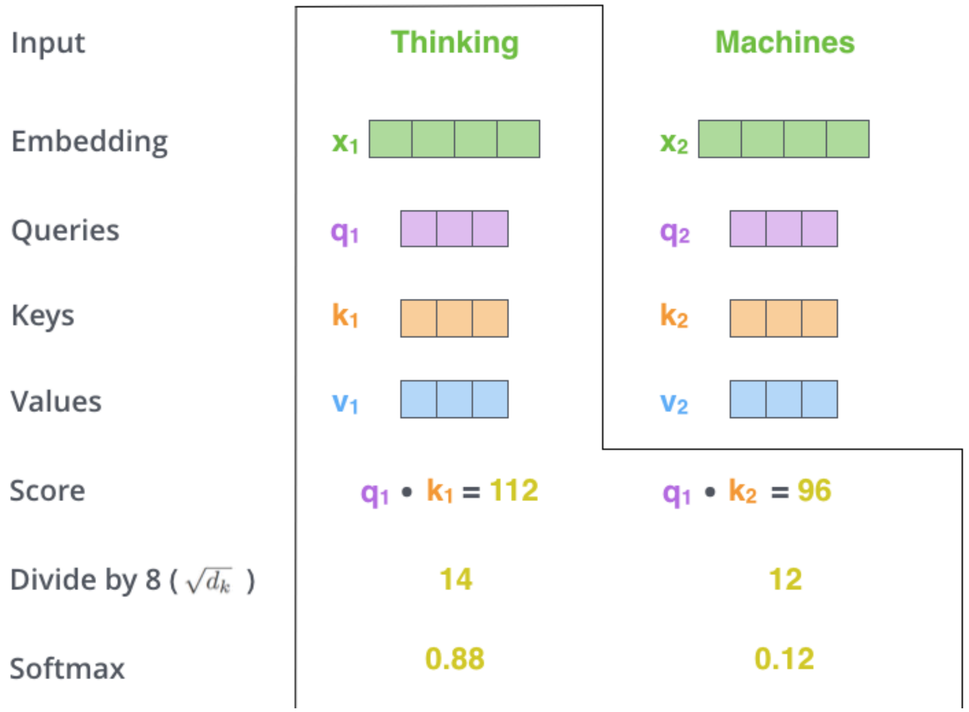


$$\text{score}\left(x_i,\ x_j\right) = q_i^T \bullet k_j$$

Image from: http://jalammar.github.io/illustrated-transformer/

STEP 3: Convert the score into a number in [0, 1] using softmax function.

Self-Attention



$$\alpha_{ij} = \frac{\exp(\text{score}(x_i,\ x_j)/\sqrt{d_k})}{\sum_{j'} \exp(\text{score}(x_i,\ x_{j'})/\sqrt{d_k})}$$

d_k : the dimension of the three vectors (q_i, k_i, v_i), is 64 in the paper.

Image from: http://jalammar.github.io/illustrated-transformer/

STEP 4: sum up the weighted value vectors.

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

Self-Attention

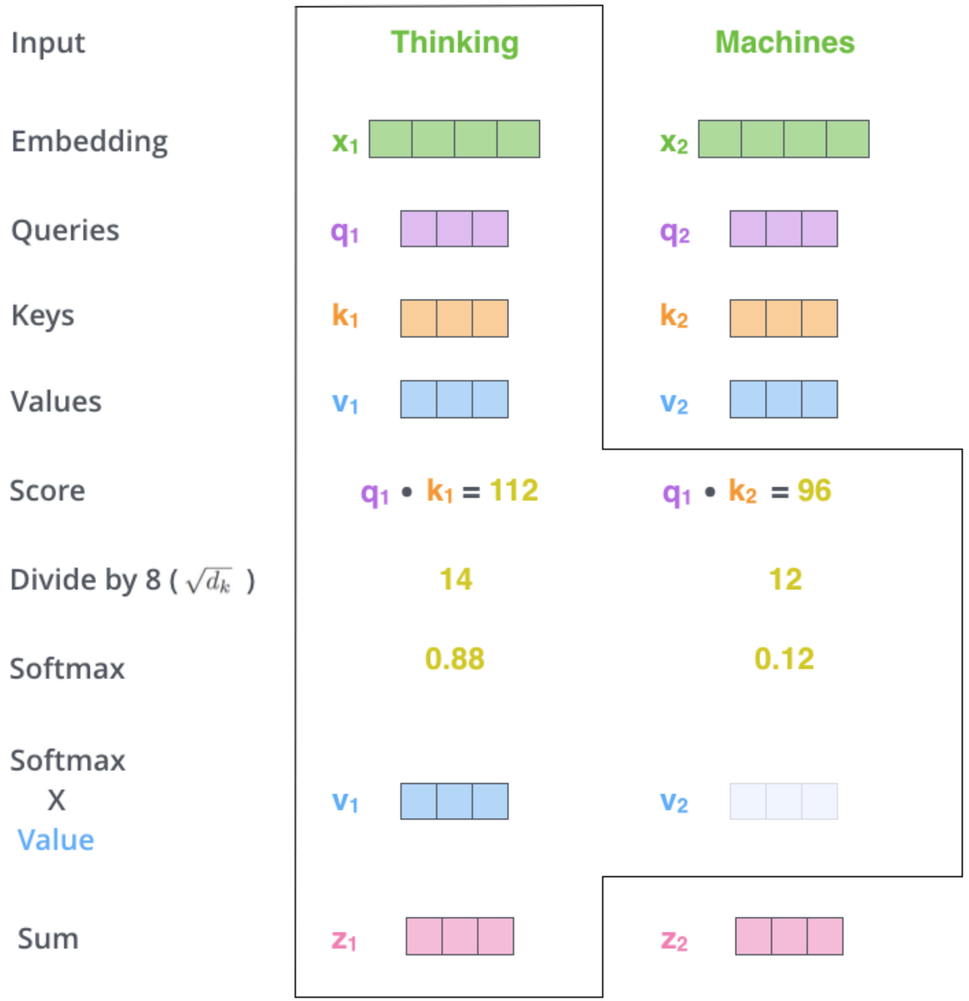
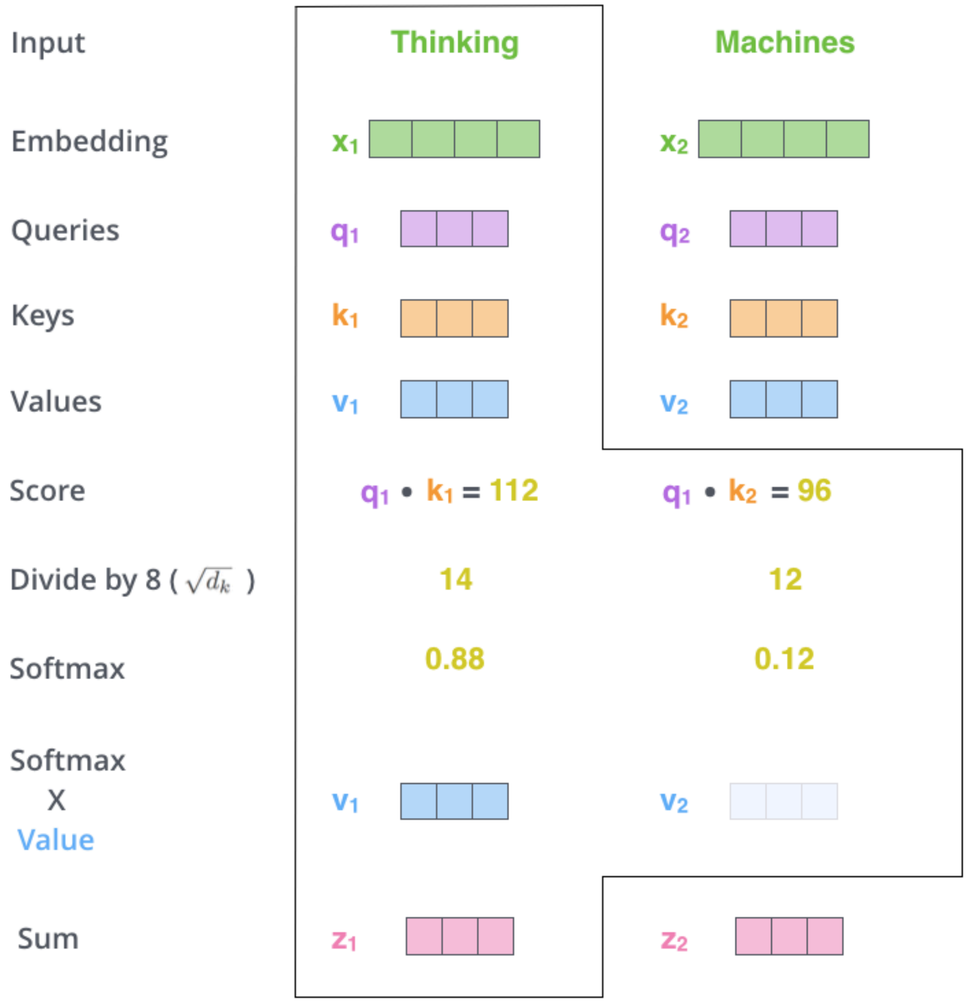


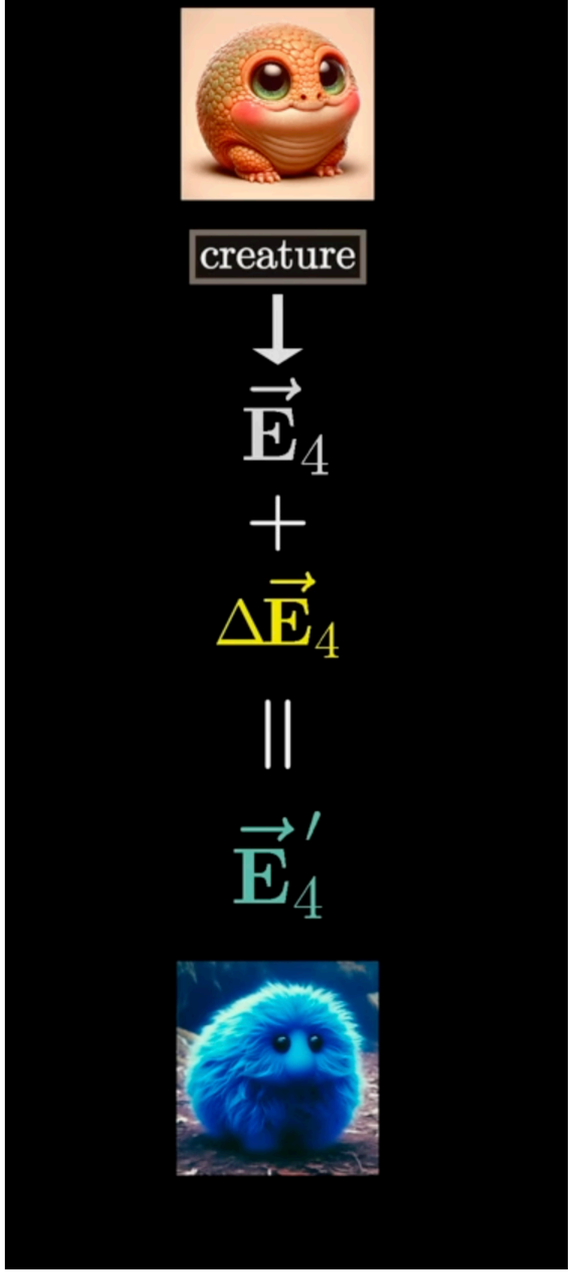
Image from: http://jalammar.github.io/illustrated-transformer/

Self-Attention

STEP 4: sum up the weighted value vectors.

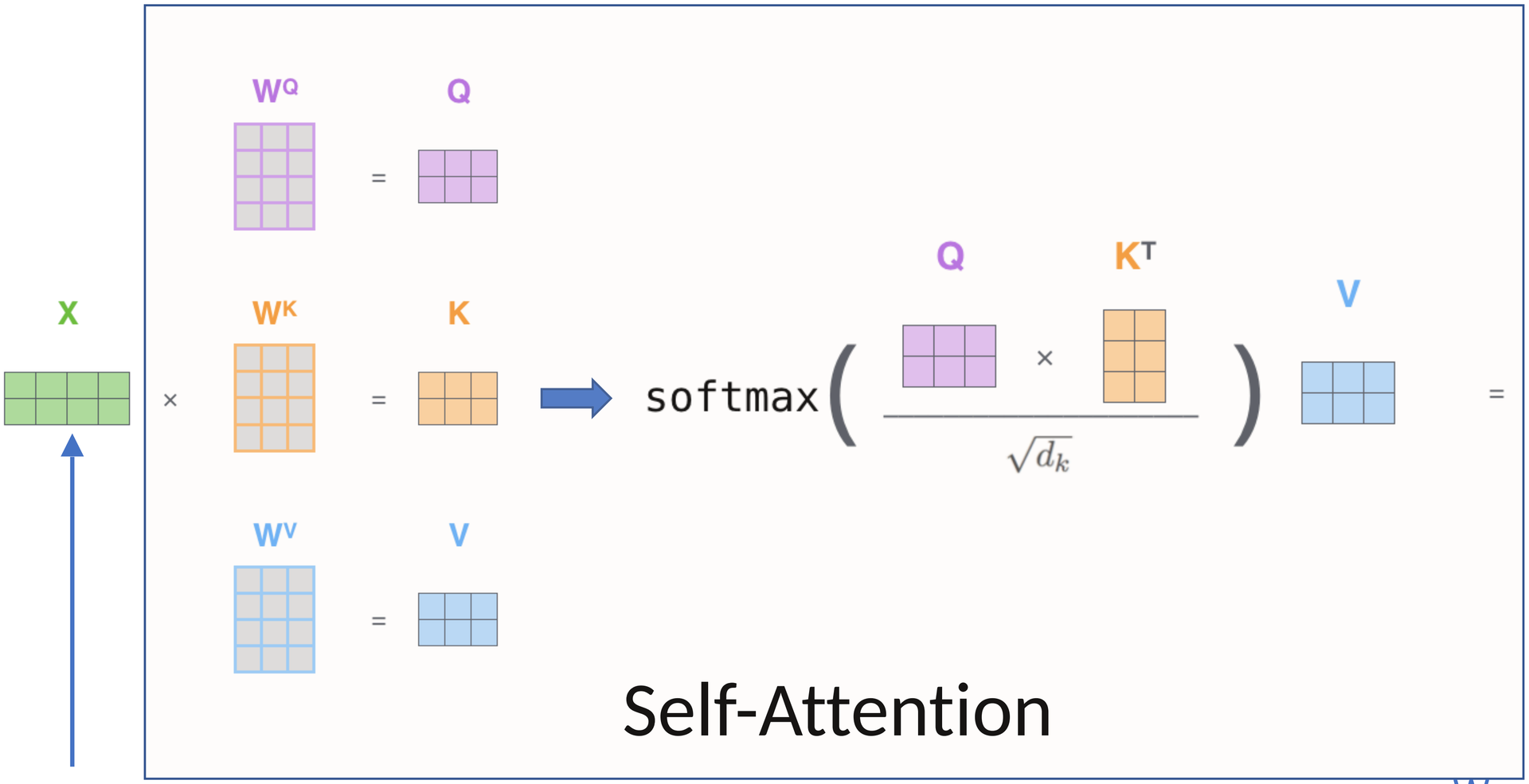
$$z_i = \sum_j \alpha_{ij} v_j$$





Attending to “fluffy blue”

Image by 3Blue1Brown



Self-Attention

Word embedding

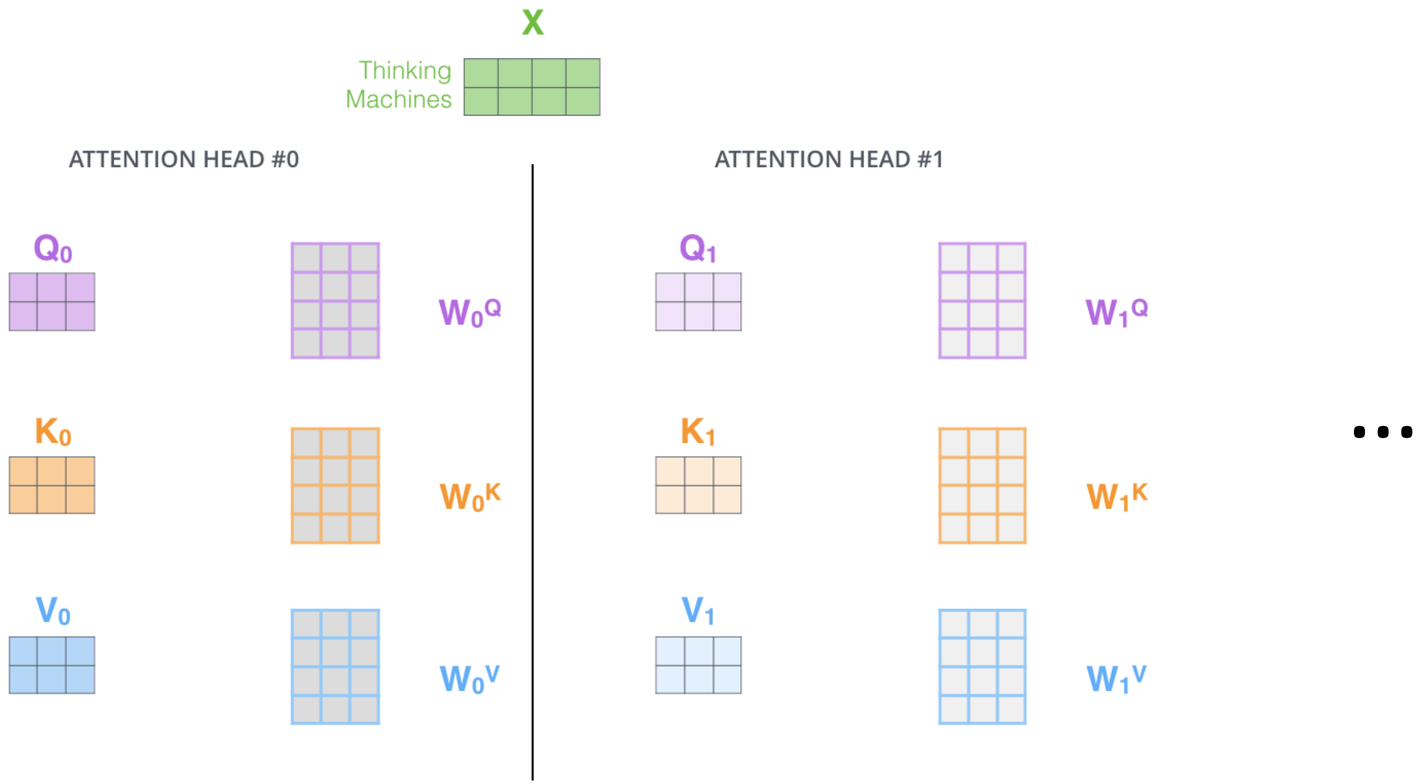
Word embedding with attention

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

Image from: http://jalammar.github.io/illustrated-transformer/

Multi-head Attention

- **Multi-head Attention**: multiple self-attentions, running in parallel

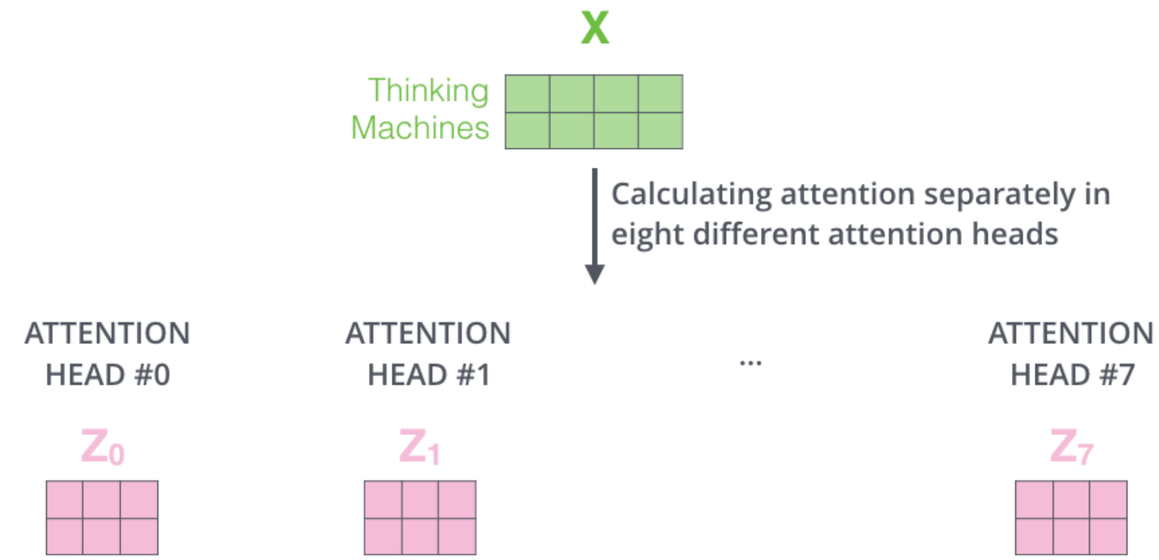


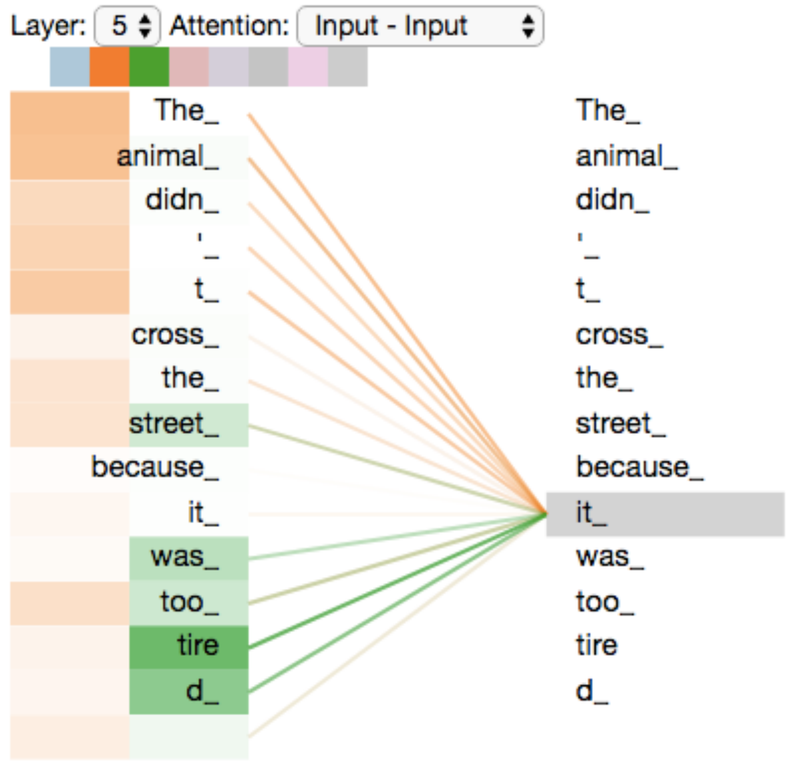
Multiple sets of Query/Key/Value weight matrices are randomly initialized and then trained.

Image from: http://jalammar.github.io/illustrated-transformer/

Multi-head Attention

- **Multi-head Attention**: multiple self-attentions, running in parallel





It enables the network to contain various kind of “attention”, e.g., one for grammar, one for vocabulary, one for conjugation, etc.

Image from: http://jalammar.github.io/illustrated-transformer/

Put all together

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

Thinking
Machines

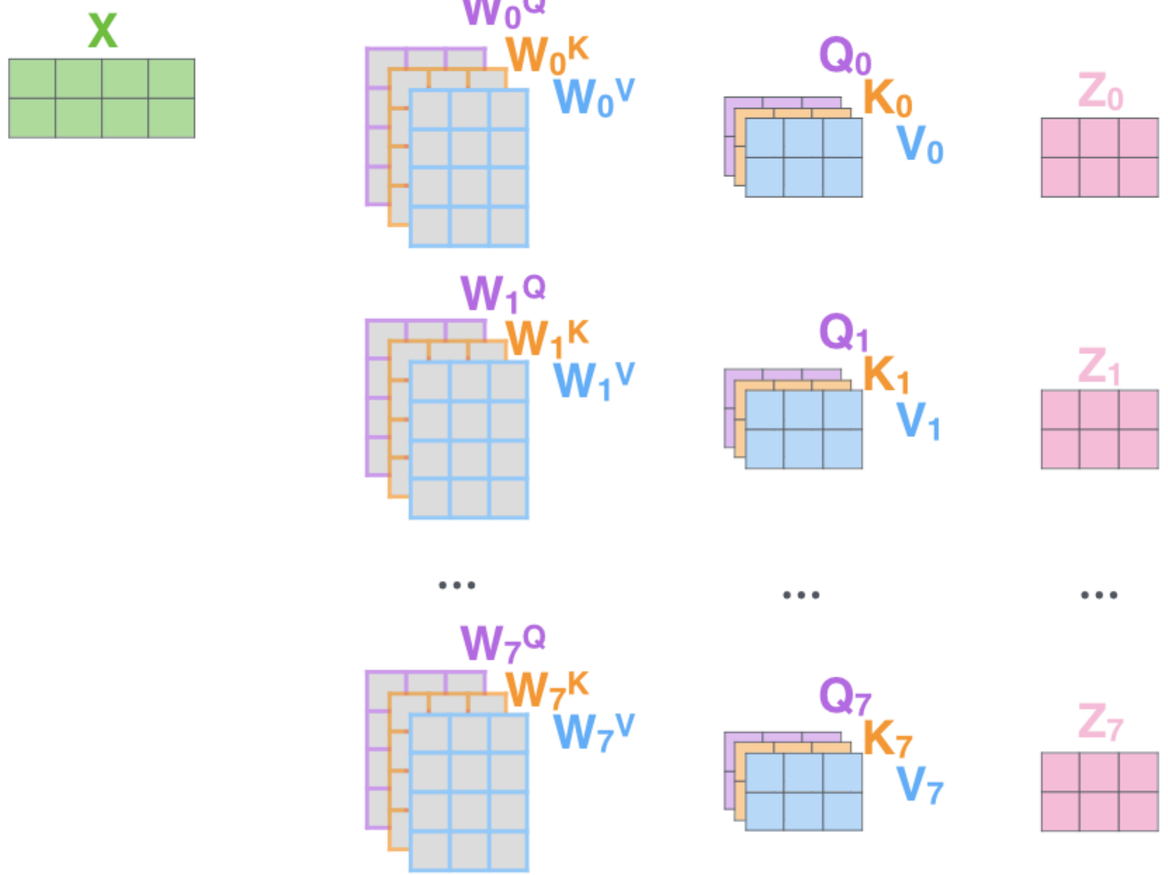


Image from: http://jalammar.github.io/illustrated-transformer/

Put all together

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

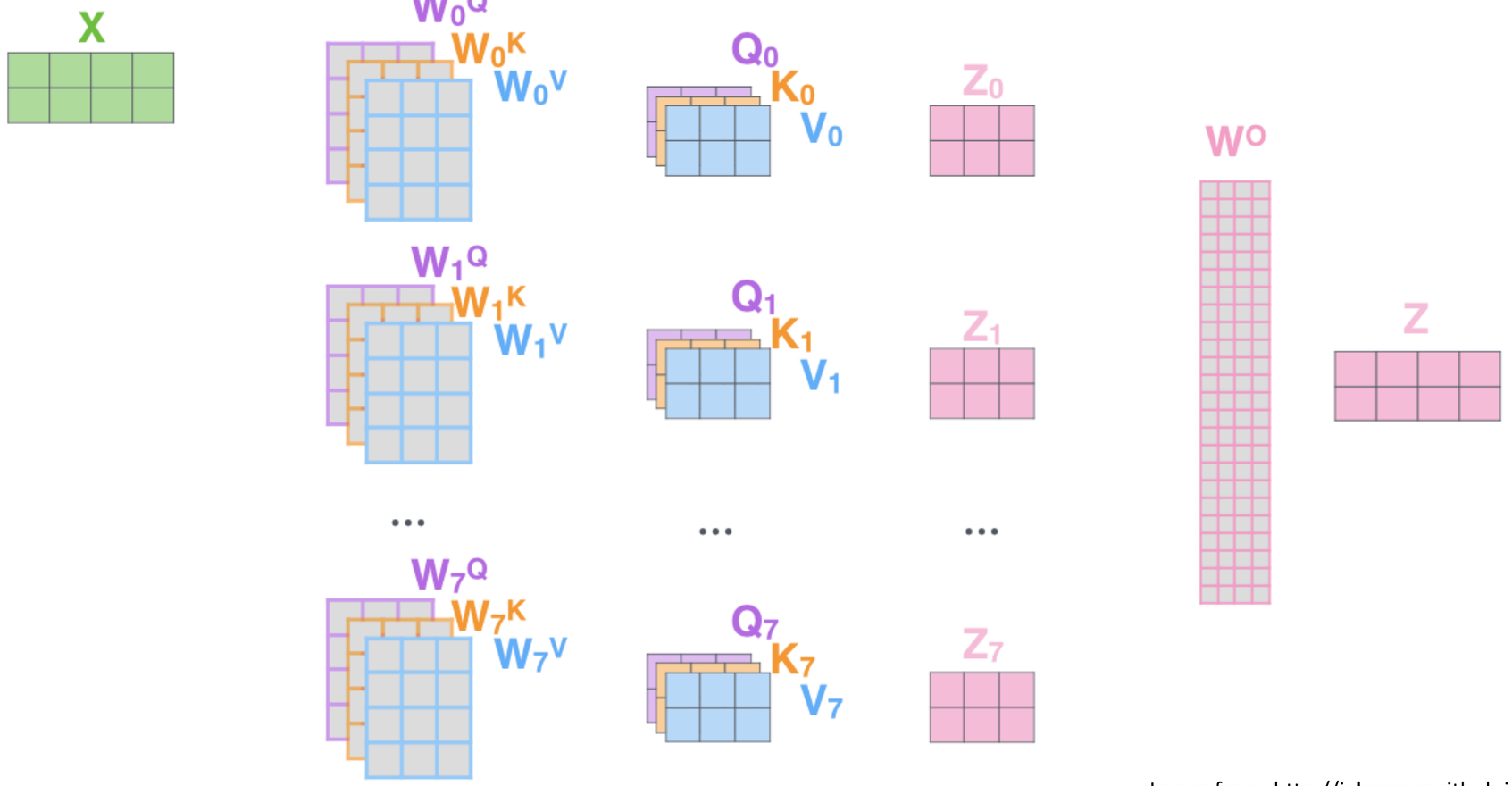
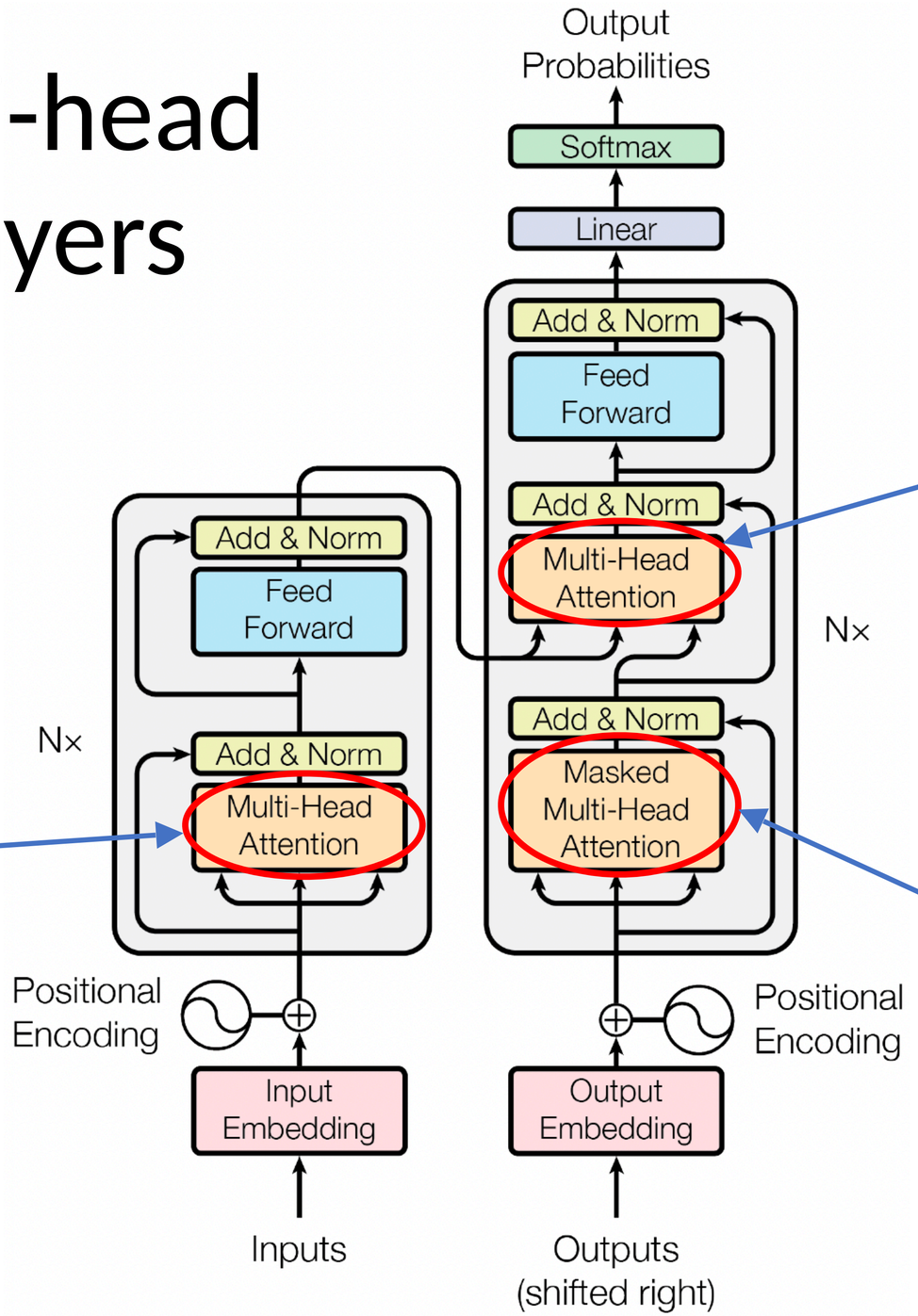


Image from: http://jalammar.github.io/illustrated-transformer/

Three multi-head attention layers

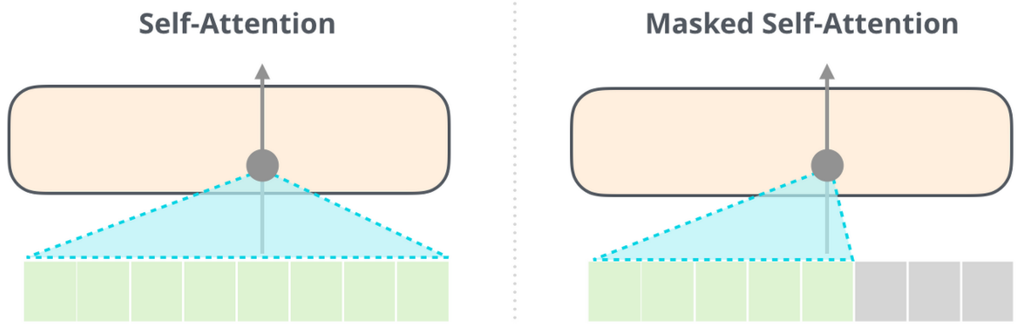


"**Encoder-decoder attention**": Keys, values from the encoder; Queries from the previous decoder layer.

"**Encoder self-attention**": Keys, values, queries all from input sequence.

"**Decoder self-attention**": Keys, values, queries all from the decoder; but only calculate the attention till the current position.

Masked Multi-Head Attention



<start> I am no man <end>

	<start>	I	am	no	man	<end>
<start>	33.6	7.6	15.5	3.8	20.8	22.3
I	7.6	34.0	30.6	8.3	26.5	27.2
am	15.5	30.6	35.9	3.8	34.0	34.8
no	3.8	8.3	3.8	34.8	33.3	33.6
man	20.8	26.5	34.0	33.3	37.0	35.9
<end>	3.8	5.7	11.3	15.1	16.6	37.4

Attention scores

+

	<start>	I	am	no	man	<end>
<start>	0	-inf	-inf	-inf	-inf	-inf
I	0	0	-inf	-inf	-inf	-inf
am	0	0	0	-inf	-inf	-inf
no	0	0	0	0	-inf	-inf
man	0	0	0	0	0	-inf
<end>	0	0	0	0	0	0

Mask filter

=

	<start>	I	am	no	man	<end>
<start>	33.6	-inf	-inf	-inf	-inf	-inf
I	7.6	34.0	-inf	-inf	-inf	-inf
am	15.5	30.6	35.9	-inf	-inf	-inf
no	3.8	8.3	3.8	34.8	-inf	-inf
man	20.8	26.5	34.0	33.3	37.0	-inf
<end>	3.8	5.7	11.3	15.1	16.6	37.4

Masked
Attention scores

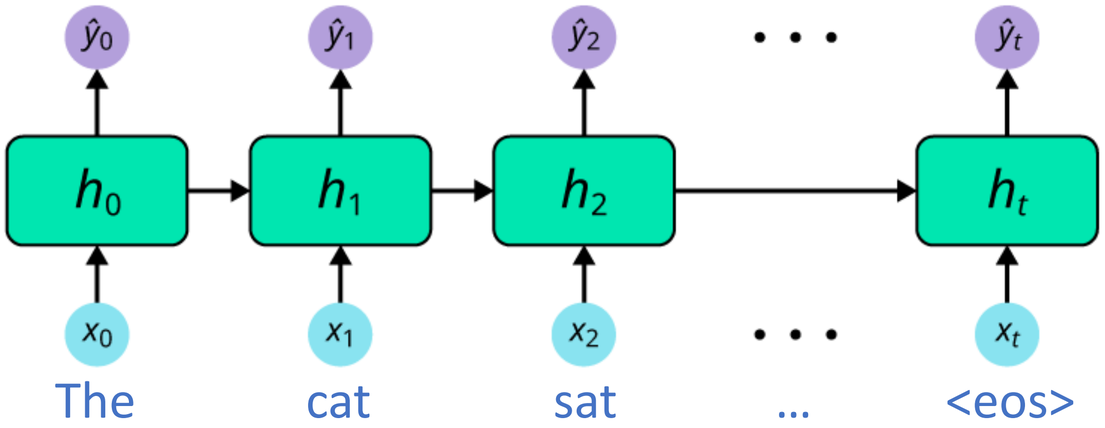
Image from: https://www.youtube.com/watch?v=gJ9kaJsE78k

Positional Encoding

The order matters

"The cat sat on the mat."

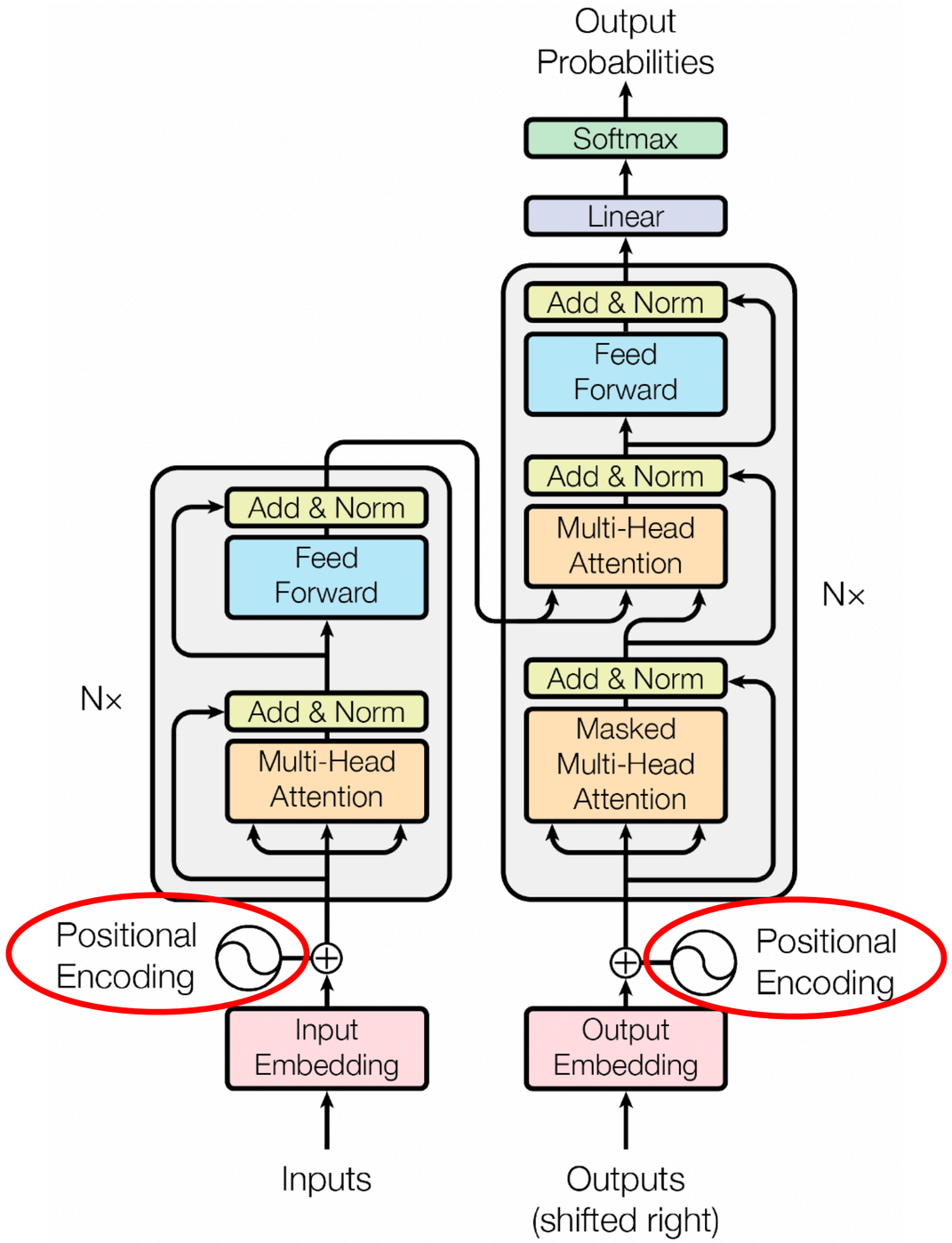
RNN



Transformer?

Q: Since transformer abandons the recurrent structure, how could it embed the positional information?
A: **Positional Encoding**.

Image from: http://jalammar.github.io/illustrated-transformer/



Positional Encoding

Key idea: add a new vector (same length) containing positional information to the word embedding.

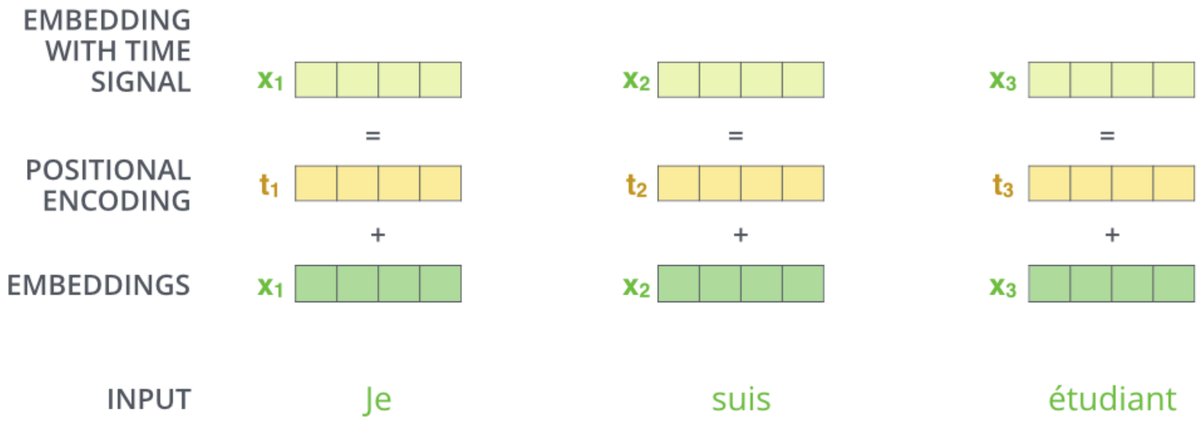


Image from: http://jalammar.github.io/illustrated-transformer/

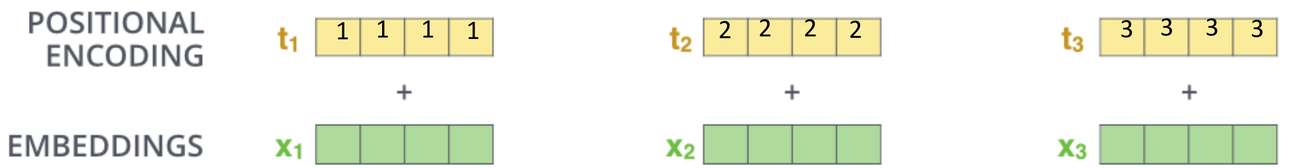
Criteria of the Positional Encoding Vector

- Unique encoding for each time-step
- Consistent distance between any two time-steps
- Should be bounded, generalize to longer sentences
- Deterministic

Criteria of the Positional Encoding Vector

- Unique encoding for each time-step ✓
- Consistent distance between any two time-steps ✓
- Should be bounded, generalize to longer sentences ✗
- Deterministic ✓

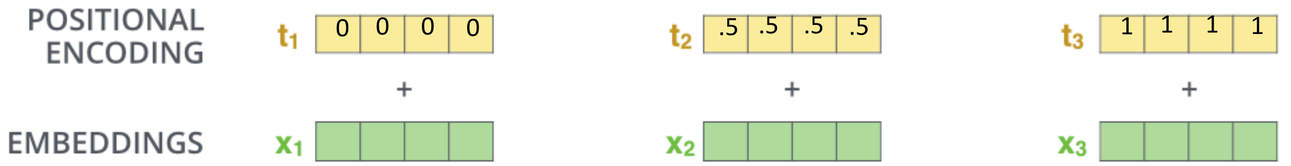
Option 1: How about using the order number?



Criteria of the Positional Encoding Vector

- Unique encoding for each time-step ✓
- Consistent distance between any two time-steps ✗
- Should be bounded, generalize to longer sentences ✓
- Deterministic ✗

Option 2: How about using fractions (1/(n-1))?



Positional Embedding in Transformer

i: i-th dimension of the vector

Key idea: using frequencies.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

pos: the position of the word in the sentence.

d_{model}: the dimension of the word embedding (=512 in the paper)

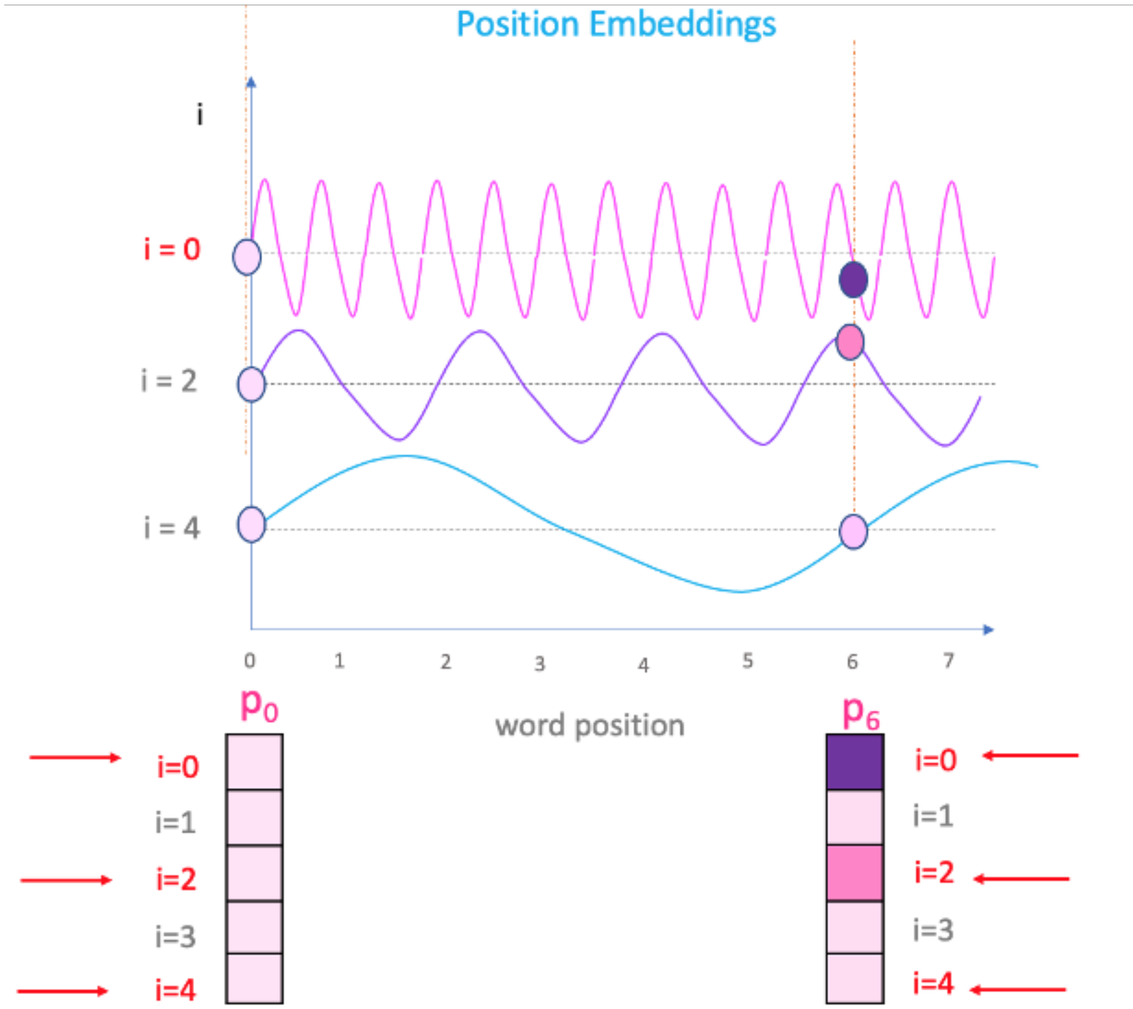
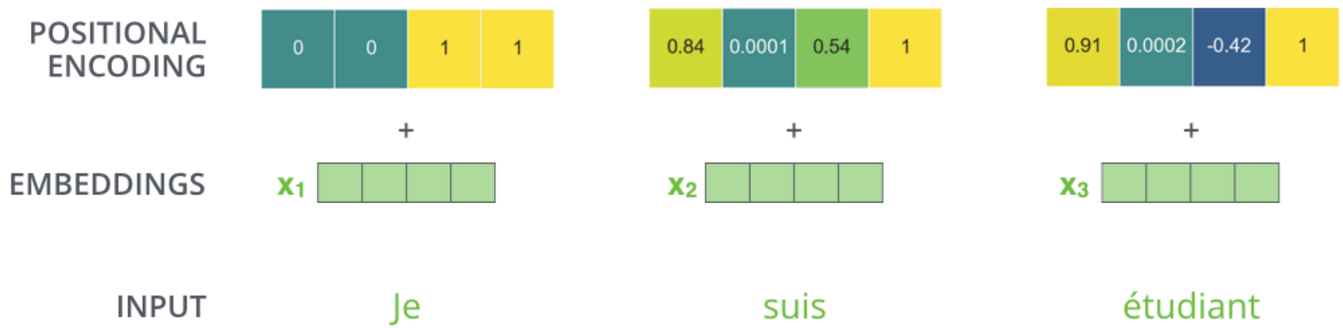


Image from: https://www.youtube.com/watch?v=dichIcUZfOw

Positional Embedding in Transformer

- Unique encoding for each time-step ✓
- Consistent distance between any two time-steps ✓
- Should be bounded, generalize to longer sentences ✓
- Deterministic ✓

When $d_{\text{model}} = 4$:



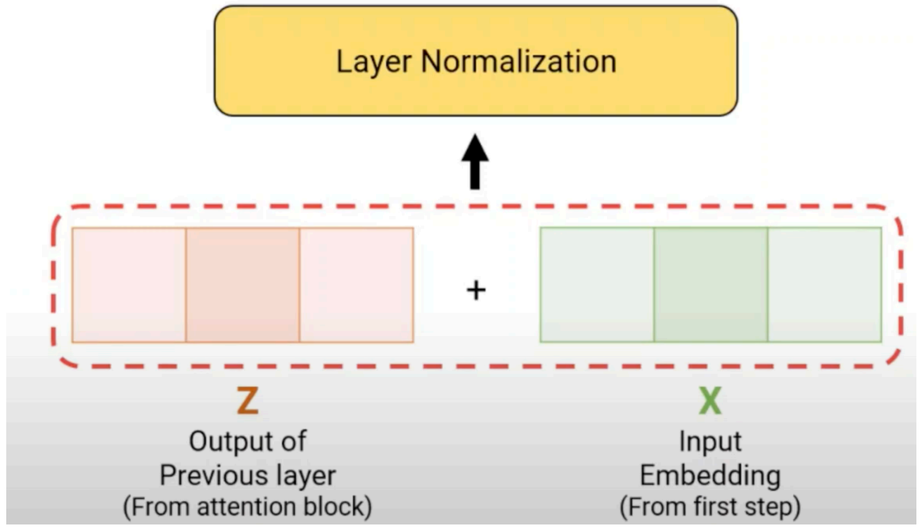
References

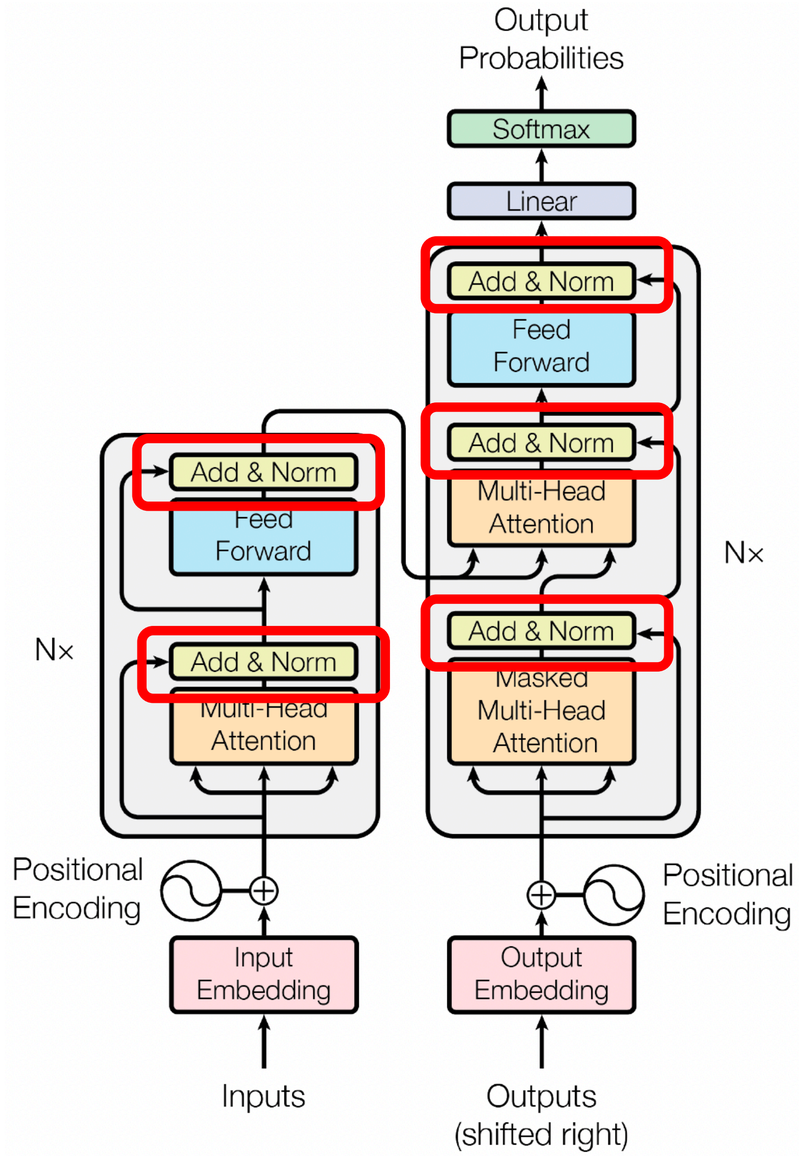
- Video by Hedu. “Visual Guide to Transformer Neural Networks – (Episode 1) Positional Embeddings”, 2021. https://www.youtube.com/watch?v=dichIcUZfOw
- Video by Halfling Wizard. “Attention is All You Need – Paper Explained”, 2021. https://www.youtube.com/watch?v=XowwKOAWYoQ

Other Layers

Add & Normalization

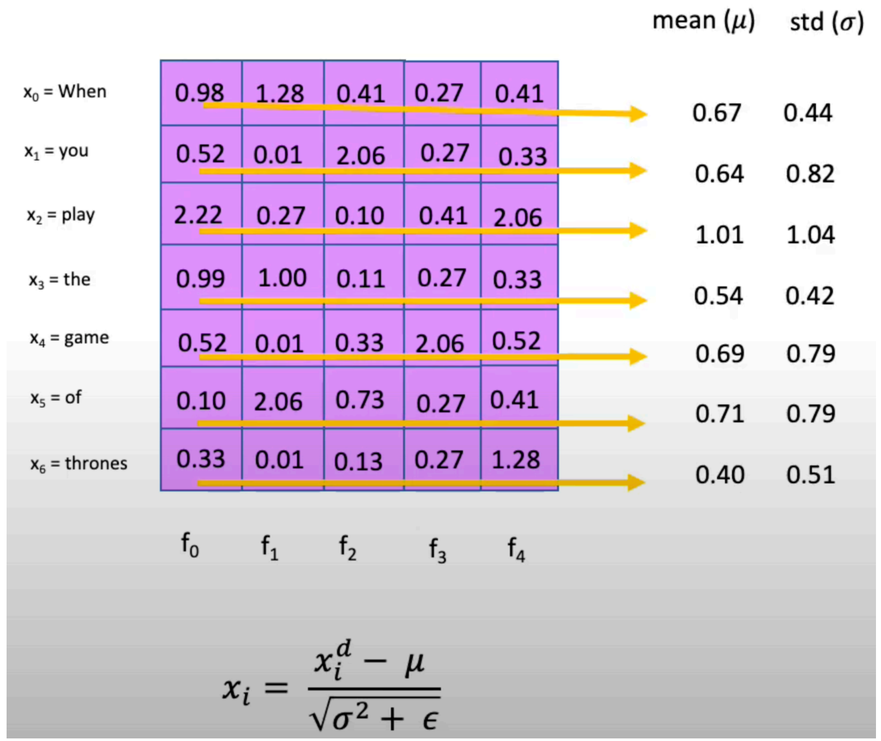
- Residual connection
  - Preserve (reuse) the earlier information
  - Prevent vanishing gradient problem





Add & Normalization

- Layer normalization
  - Stabilize training
  - Faster training
  - Prevent weight explosion



Layer normalization

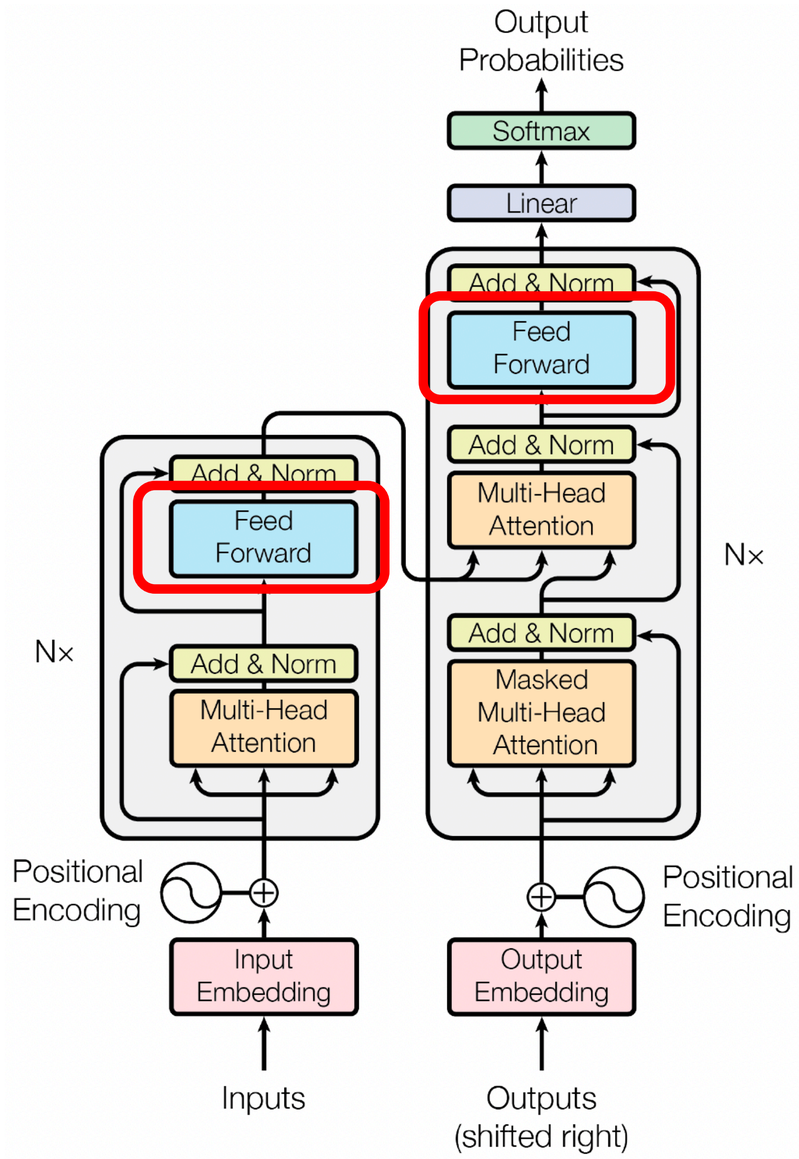
0.71	1.40	-0.59	-0.92	-0.59
-0.14	-0.77	1.74	-0.45	-0.38
1.16	-0.72	-0.88	-0.58	1.01
1.06	1.09	-1.02	-0.64	-0.50
-0.21	-0.85	-0.45	1.73	-0.21
-0.78	1.71	0.02	-0.56	-0.39
-0.15	-0.78	-0.54	-0.27	1.73

Feed Forward

- Two fully-connected layers

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

- The exact same feed-forward network (with different parameters) is independently applied to the encoder and decoder.



Final linear and Softmax Layer

Which word in our vocabulary is associated with this index?

am

Get the index of the cell with the highest value (**argmax**)

5

log_probs

0 1 2 3 4 5 … vocab_size

Softmax

logits

0 1 2 3 4 5 … vocab_size

Linear

Decoder stack output

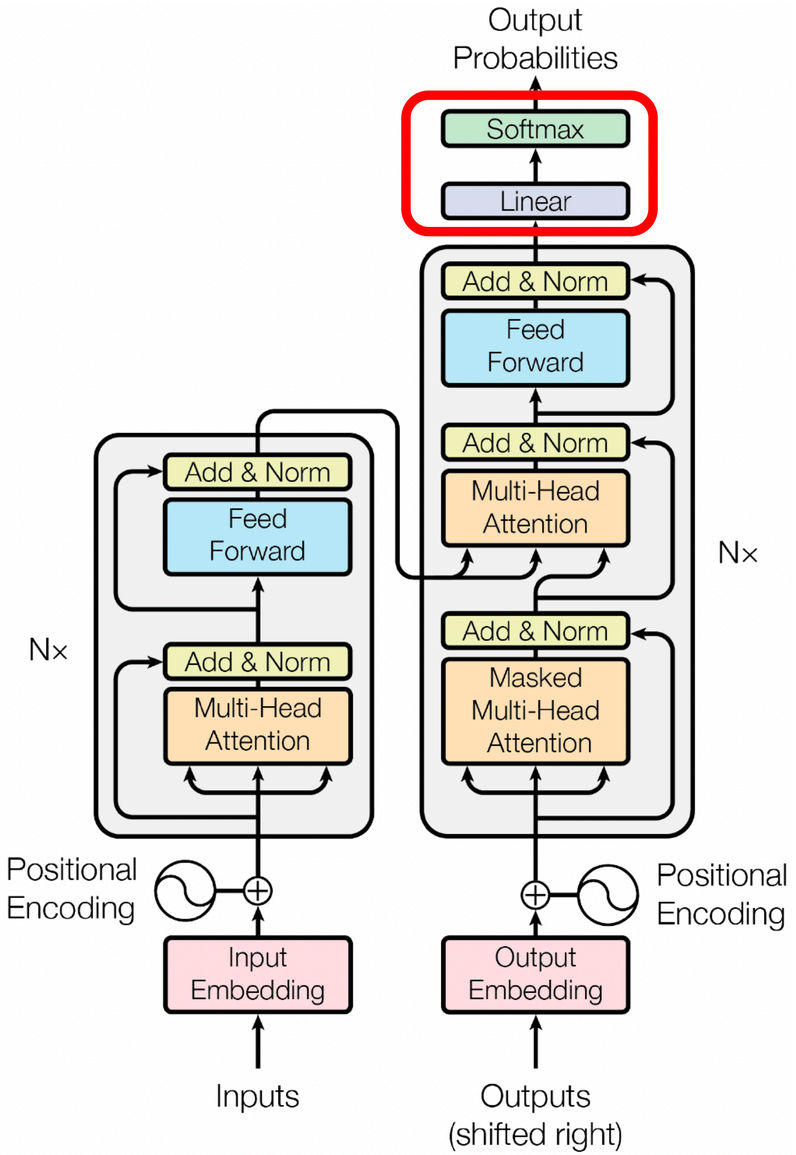
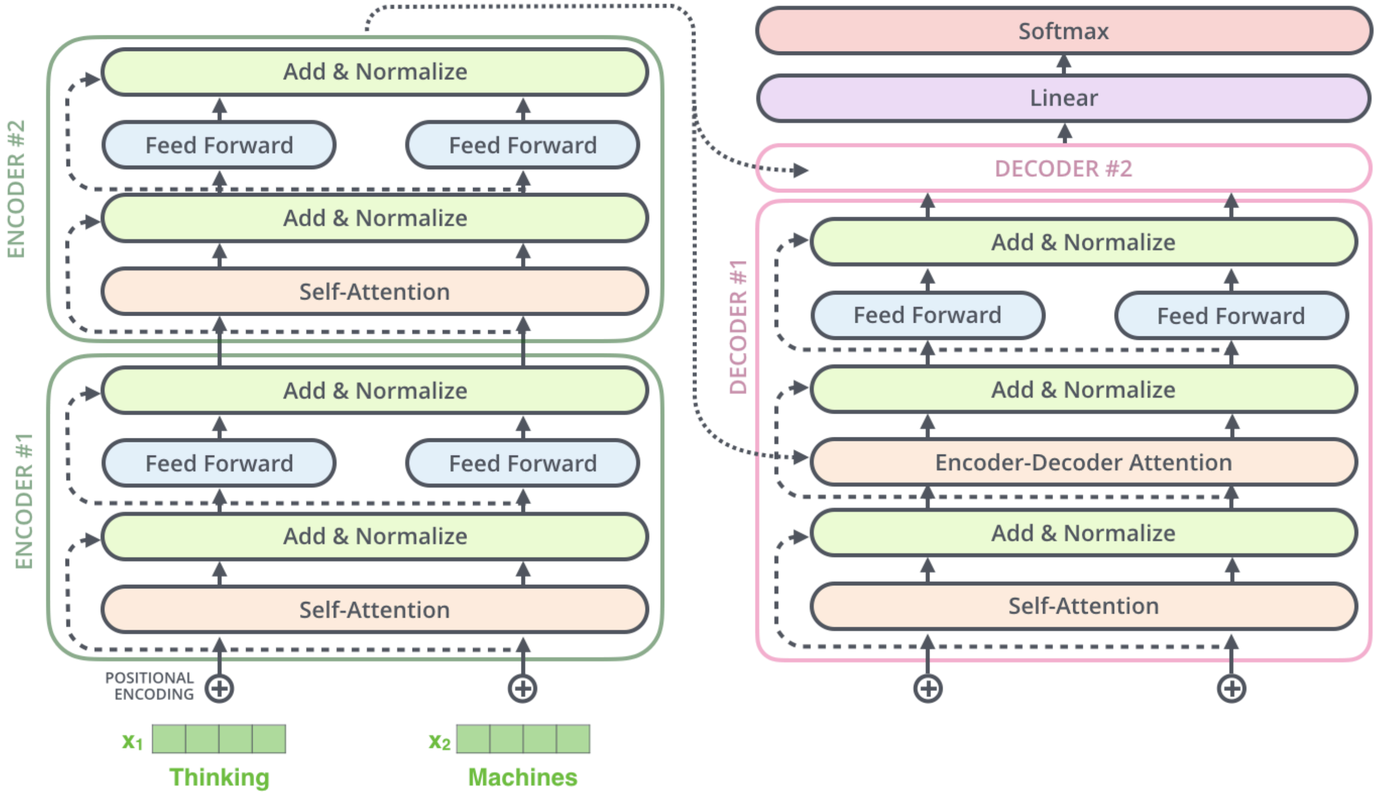


Image from: http://jalammar.github.io/illustrated-transformer/

Stack of Encoders and Decoders



N=6 in the paper

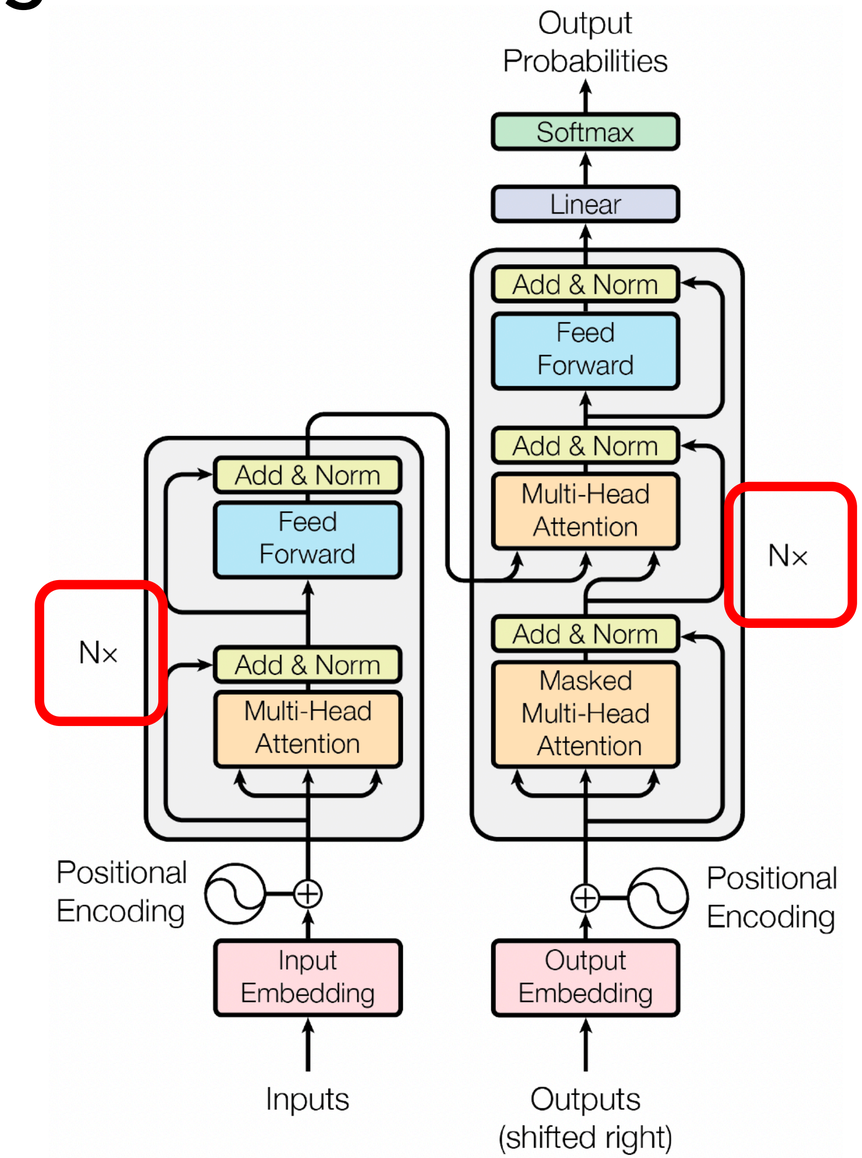


Image from: http://jalammar.github.io/illustrated-transformer/

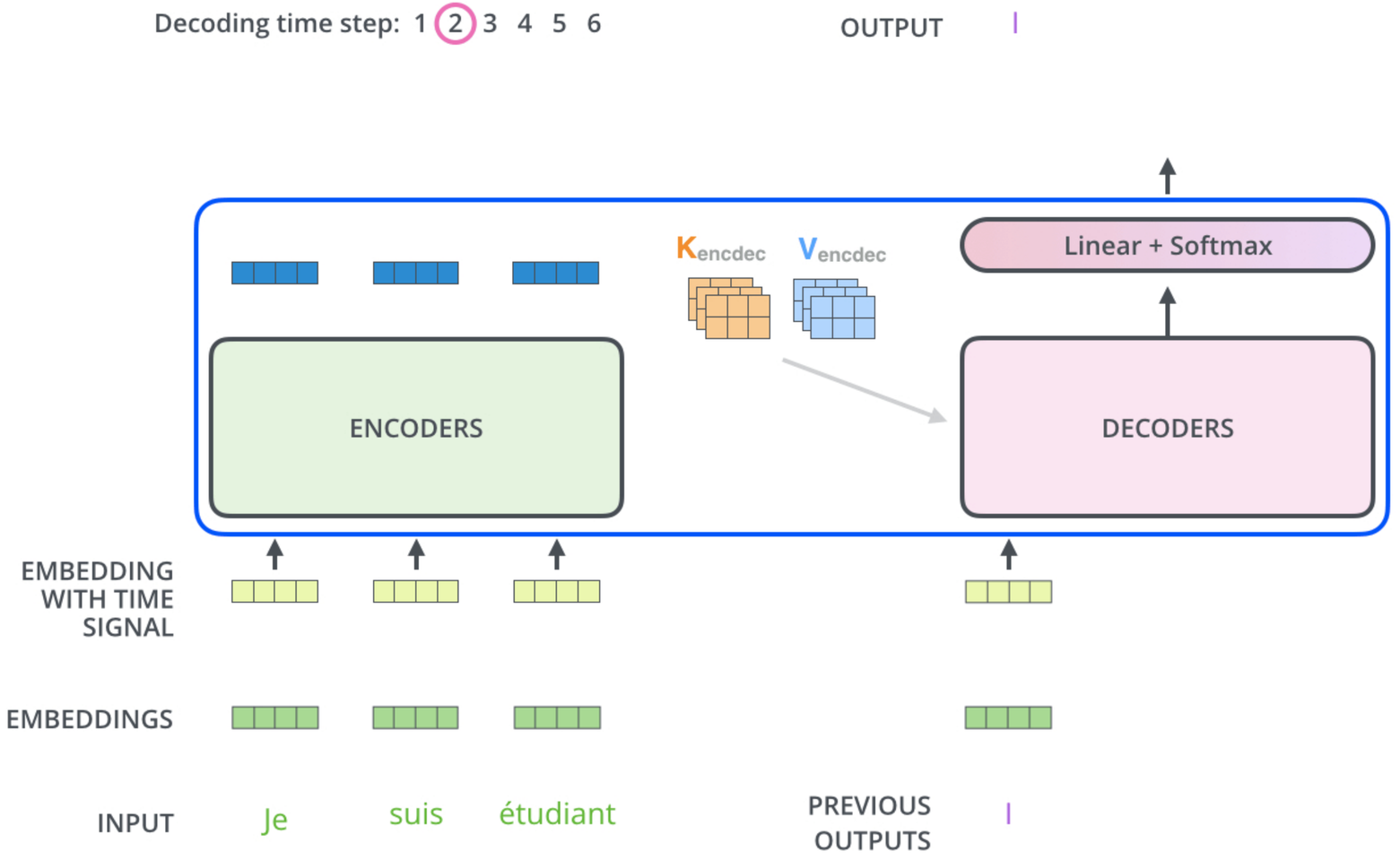
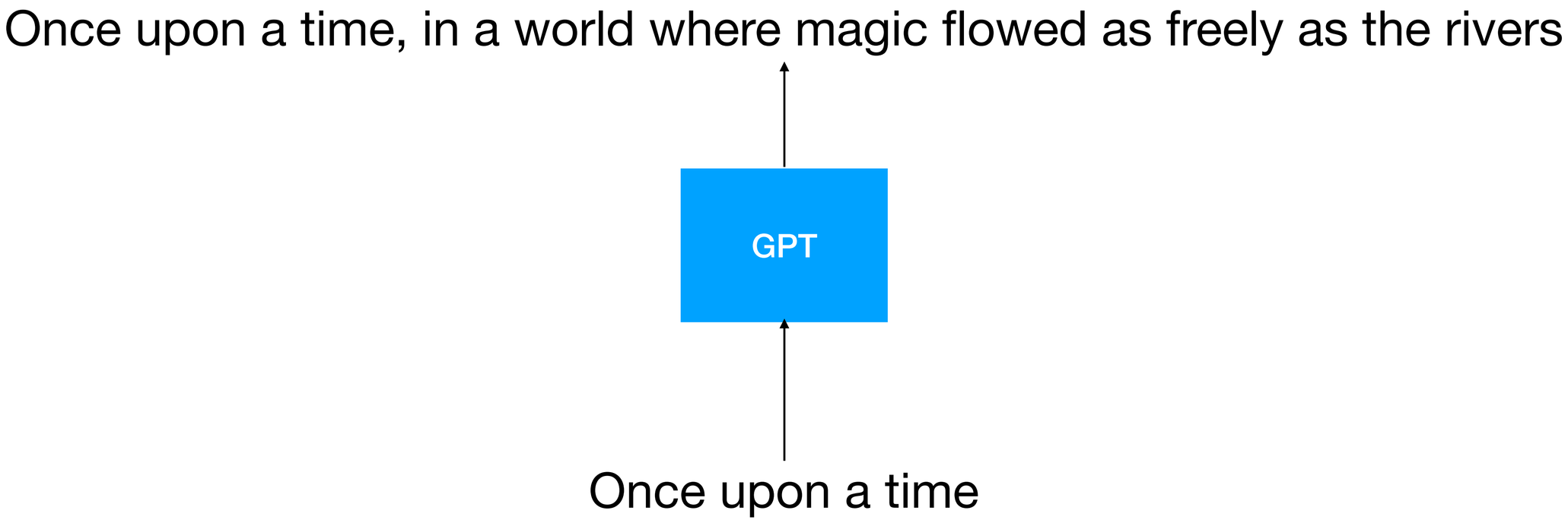


Image from: http://jalammar.github.io/illustrated-transformer/

Putting it together

First we will try to understand GPT

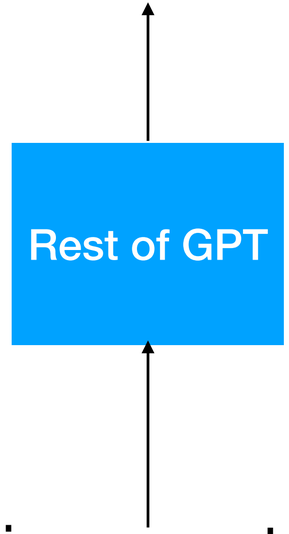
We want to understand what happens when we provide a prompt to GPT



First we will try to understand GPT

We want to understand what happens when we provide a prompt to GPT

Once upon a time, in a world where magic flowed as freely as the rivers



The text is converted into token vectors -

Once upon a time [12805, 5304, 264, 892]

Once upon a time

First we will try to understand GPT

We want to understand what happens when we provide a prompt to GPT

Once upon a time, in a world where magic flowed as freely as the rivers

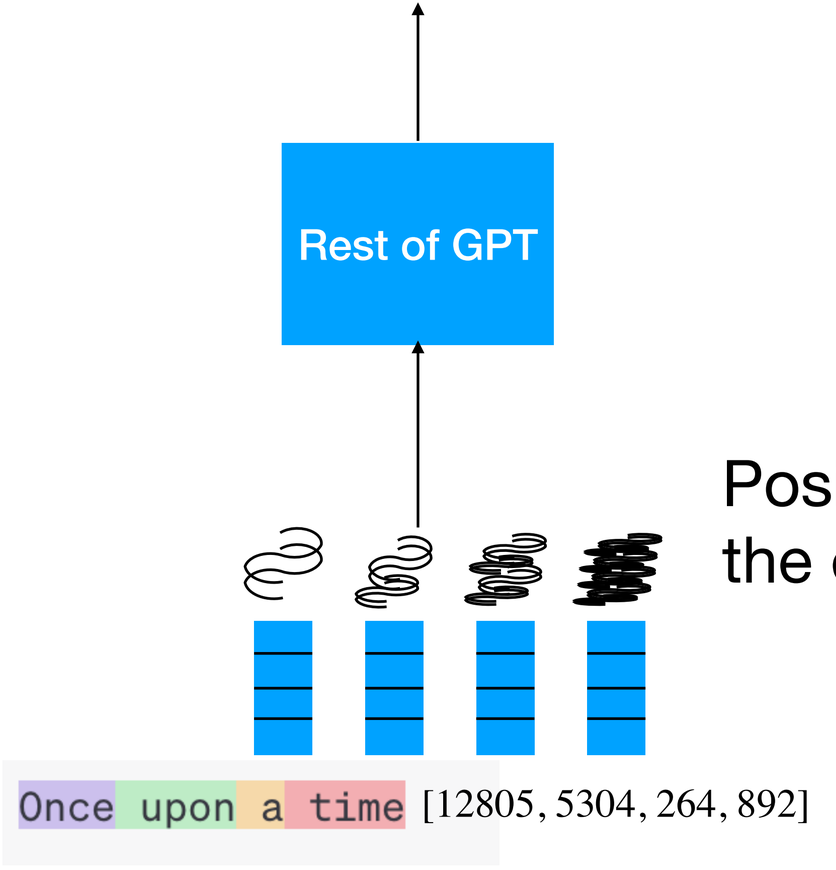
Sequence of integers is mapped to a set of embedding vectors by an embedding matrix W_E

Once upon a time [12805, 5304, 264, 892]

First we will try to understand GPT

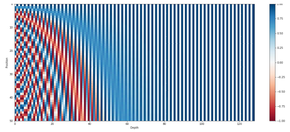
We want to understand what happens when we provide a prompt to GPT

Once upon a time, in a world where magic flowed as freely as the rivers



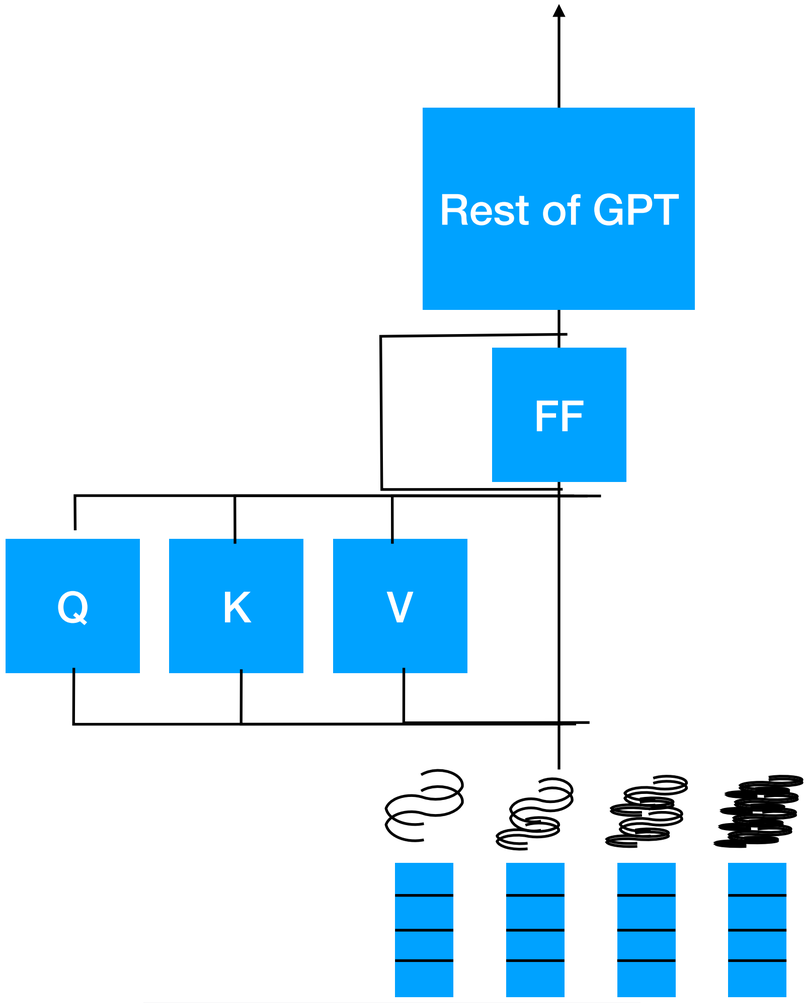
Positional encoding is added to the embedding -

$$\psi'(w_t) = \psi(w_t) + \overrightarrow{p_t}$$



First we will try to understand GPT

We want to understand what happens when we provide a prompt to GPT

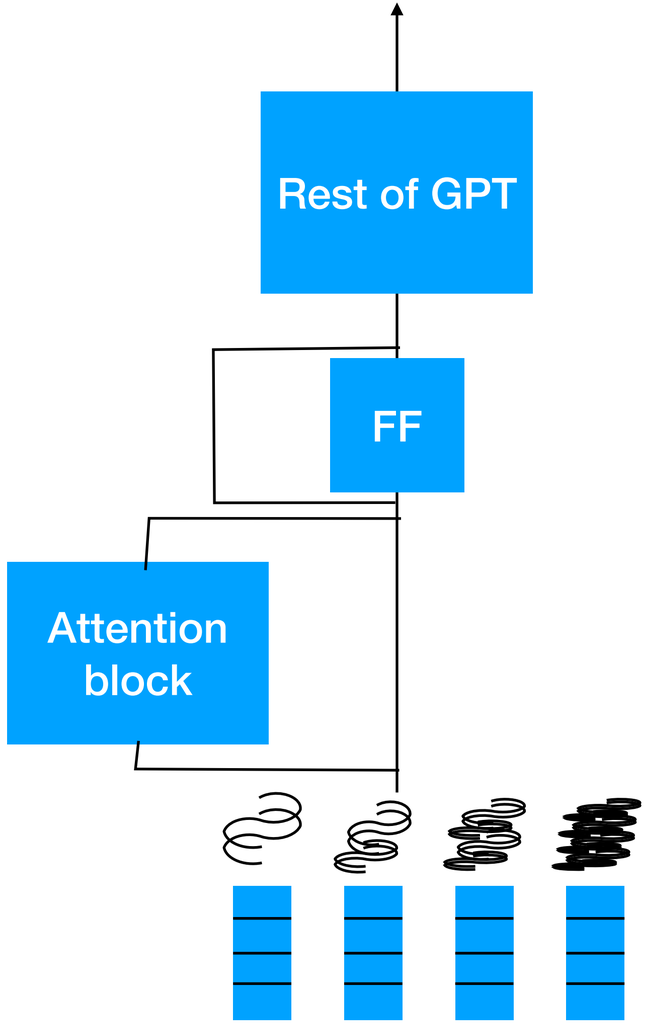


This residual stream is then passed through a ‘self attention + FF layer block

Once upon a time [12805, 5304, 264, 892]

First we will try to understand GPT

We want to understand what happens when we provide a prompt to GPT

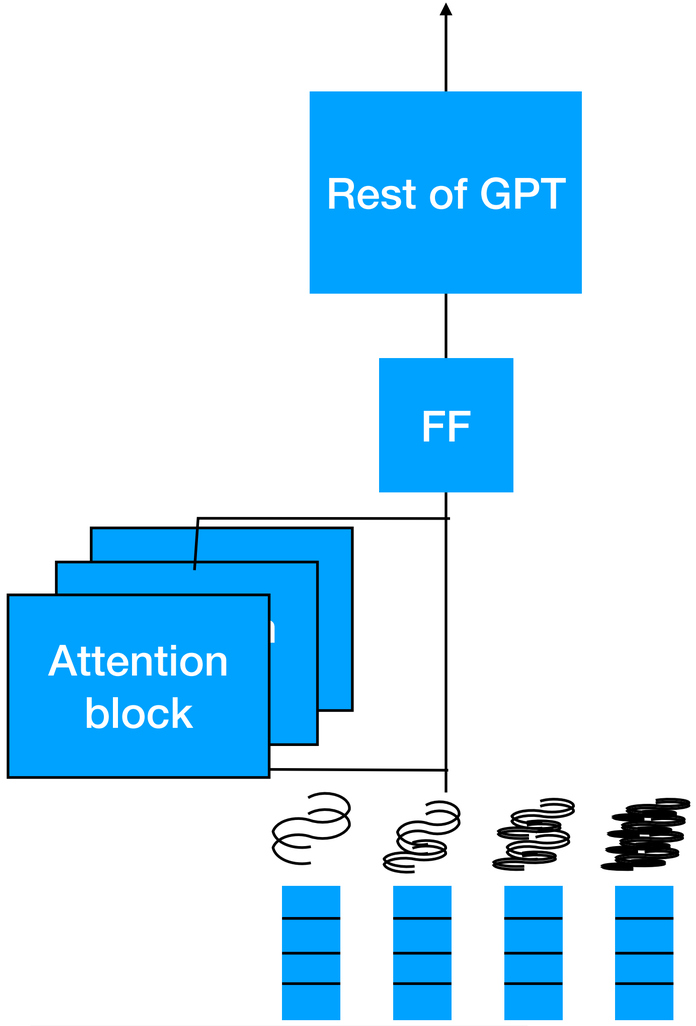


This residual stream is then passed through a ‘self attention + FF layer block

Once upon a time [12805, 5304, 264, 892]

First we will try to understand GPT

We want to understand what happens when we provide a prompt to GPT

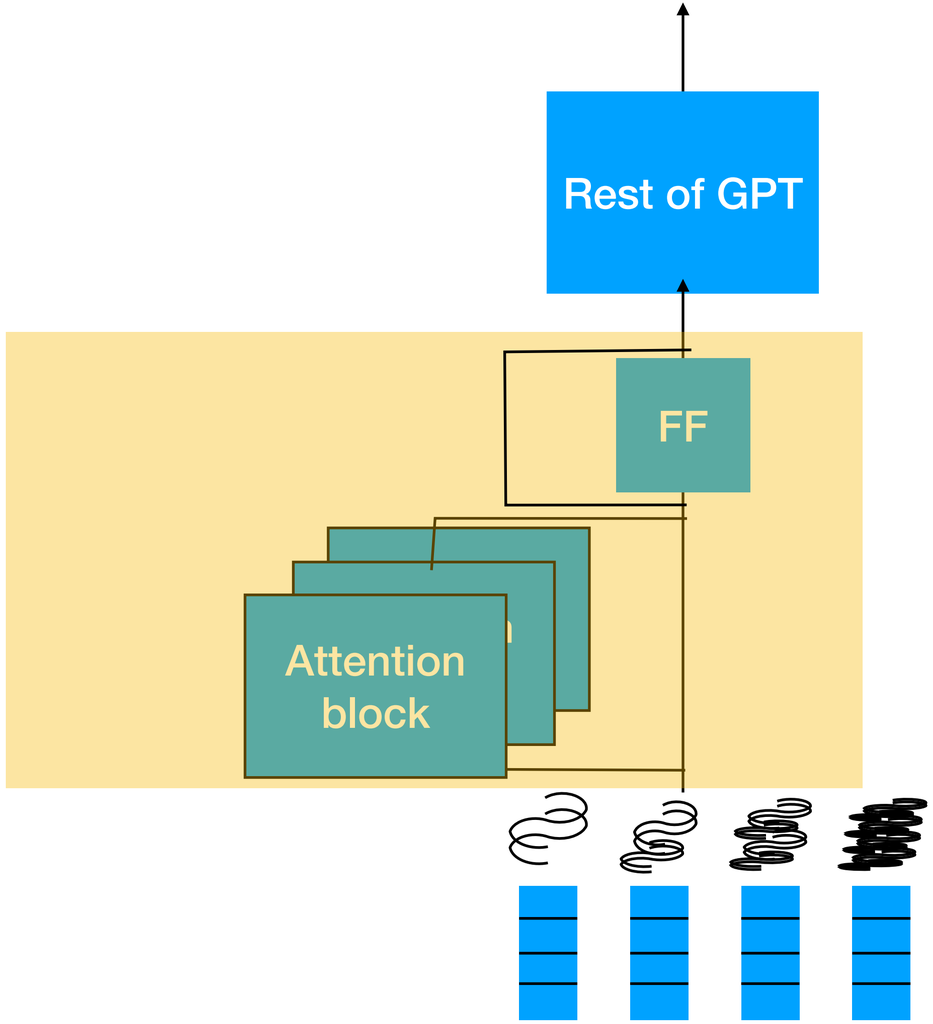


In practice, there are multiple heads running in parallel

Once upon a time [12805, 5304, 264, 892]

First we will try to understand GPT

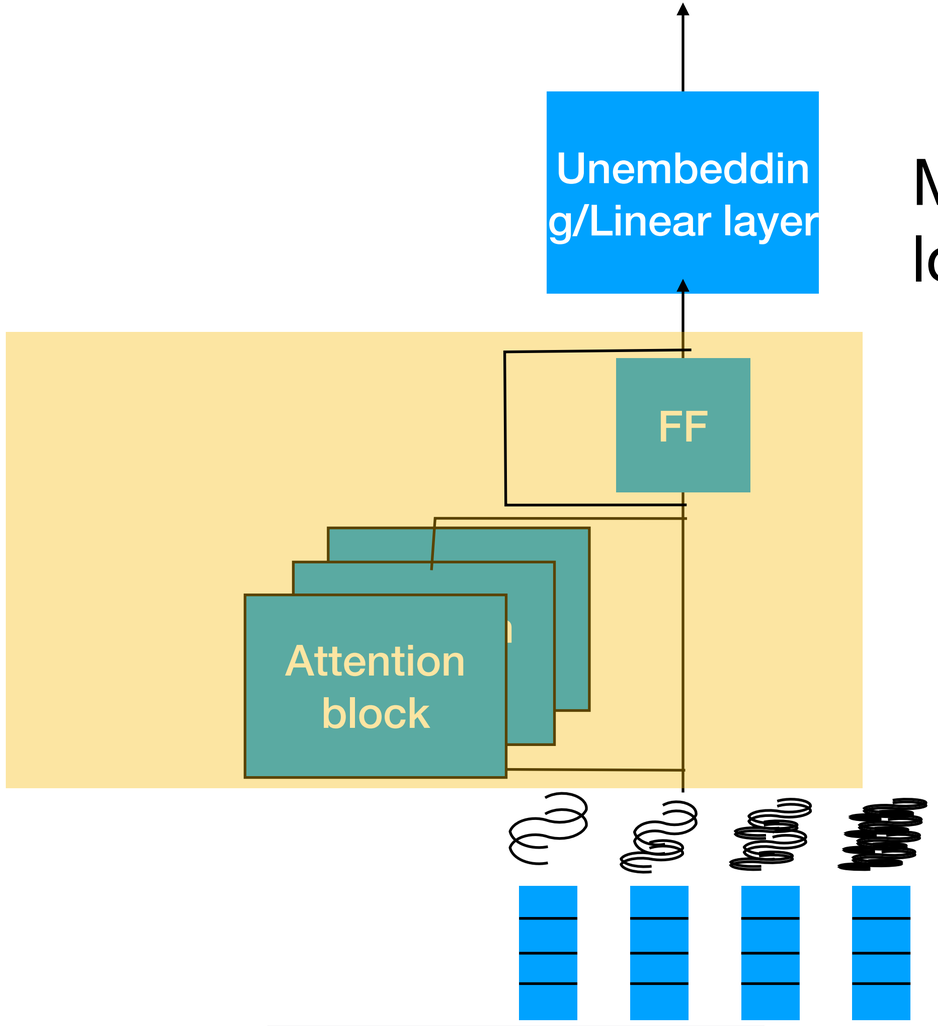
We want to understand what happens when we provide a prompt to GPT



This is one layer of the transformer - this structure is repeated many times

First we will try to understand GPT

We want to understand what happens when we provide a prompt to GPT



Maps from embedding space to logits for the token space

Many layers of this structure

References

- 3Blue1Brown - Video 1, Video2
- Andrej Karpathy - NanoGPT implementation, Video Tutorial


?
That’s all folks
QUESTIONS?