Transformers

Outline

- The challenge of modeling sequences
- The transformer architecture
  - The big picture
- Details and fine print
- The impact of transformers

Outline

- The challenge of modeling sequences
- The transformer architecture
  - The big picture
- Details and fine print
- The impact of transformers

Sequences abound in NLP

S a l t   L a k e   C i t y

John lives in Salt Lake City

John lives in Salt Lake City. He enjoys hiking with his dog. His cat hates hiking.

Noun Verb Preposition Noun Noun Noun

B-PER O O B-LOC I-LOC I-LOC

And we can get very creative how we encode complex objects

Example: We can encode parse trees as a sequence of decisions needed to construct the tree

Sequences abound in NLP

S a l t   L a k e   C i t y

John lives in Salt Lake City

Natural question: How do we model sequential inputs and outputs?

More concretely, we need a mechanism that allows us to

1. Capture sequential dependencies between inputs
2. Model uncertainty over sequential outputs

And we can get very creative how we encode complex objects

Example: We can encode parse trees as a sequence of decisions needed to construct the tree

The challenge of modeling sequences

A sequence may be arbitrarily long. We only have a finite number of parameters.

The challenge of modeling sequences

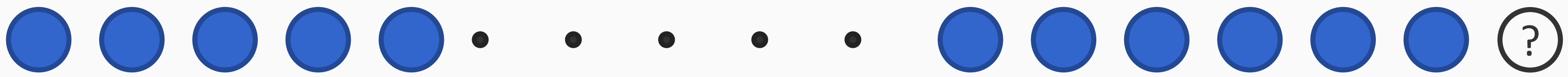
A sequence may be arbitrarily long. We only have a finite number of parameters.

1. How do we model what comes next using only a finite number of parameters?

2. How do we build “expressive enough” models without cutting off the history at an arbitrary point (i.e. a Markov assumption)?

3. Can we build models that can encode the meaning of items in a sequence in parallel?

The challenge of modeling sequences



A sequence may be arbitrarily long. We only have a finite number of parameters.

1. How do we model what comes next using only a finite number of parameters?

   ✓ Markov models

2. How do we build "expressive enough" models without cutting off the history at an arbitrary point (i.e. a Markov assumption)?

   X Markov models

3. Can we build models that can encode the meaning of items in a sequence in parallel?

   X Markov models

The challenge of modeling sequences

A sequence may be arbitrarily long. We only have a finite number of parameters.

1. How do we model what comes next using only a finite number of parameters?

   ✓ Markov models ✓ Recurrent Neural Networks

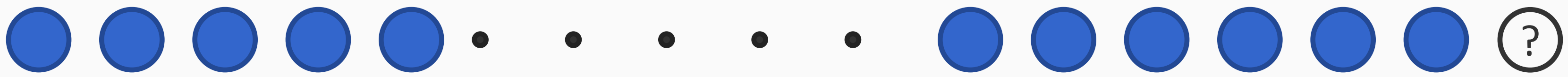
2. How do we build “expressive enough” models without cutting off the history at an arbitrary point (i.e. a Markov assumption)?

   X Markov models ✓ Recurrent Neural Networks

3. Can we build models that can encode the meaning of items in a sequence in parallel?

   X Markov models X Recurrent Neural Networks

The challenge of modeling sequences



A sequence may be arbitrarily long. We only have a finite number of parameters.

1. How do we model what comes next using only a finite number of parameters?

   ✓ Markov models ✓ Recurrent Neural Networks ✓ Transformers

2. How do we build "expressive enough" models without cutting off the history at an arbitrary point (i.e. a Markov assumption)?

   X Markov models ✓ Recurrent Neural Networks ✓ Transformers

3. Can we build models that can encode the meaning of items in a sequence in parallel?

   X Markov models X Recurrent Neural Networks ✓ Transformers

Outline

- The challenge of modeling sequences
- The transformer architecture
  - The big picture
- Details and fine print
- The impact of transformers

Recurrent neural networks revisited

RNNs consume one input at each time step

To process (i.e., apply forward pass or backward pass) a sequence with n elements, this will take $O(n)$ steps

- Doesn't seem like a problem. But recall this is for each example
  - And we will be training on millions-to-billions of sequences
- This affects the speed of training because we cannot parallelize over the sequence elements

Can we have a sequence model that operates on all n inputs in parallel?

- That is each forward pass is $O(1)$ time?

“The Transformer paper”: NeurIPS 2017

Focus: machine translation

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

“We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely.”

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-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task

Transformer architecture: Vaswani et al 2017

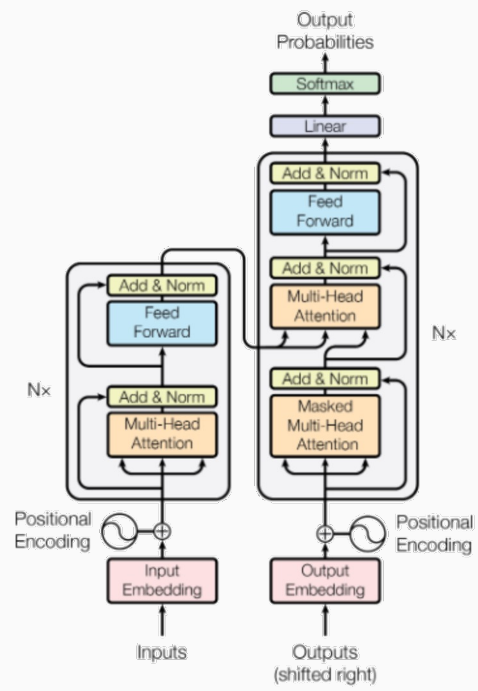
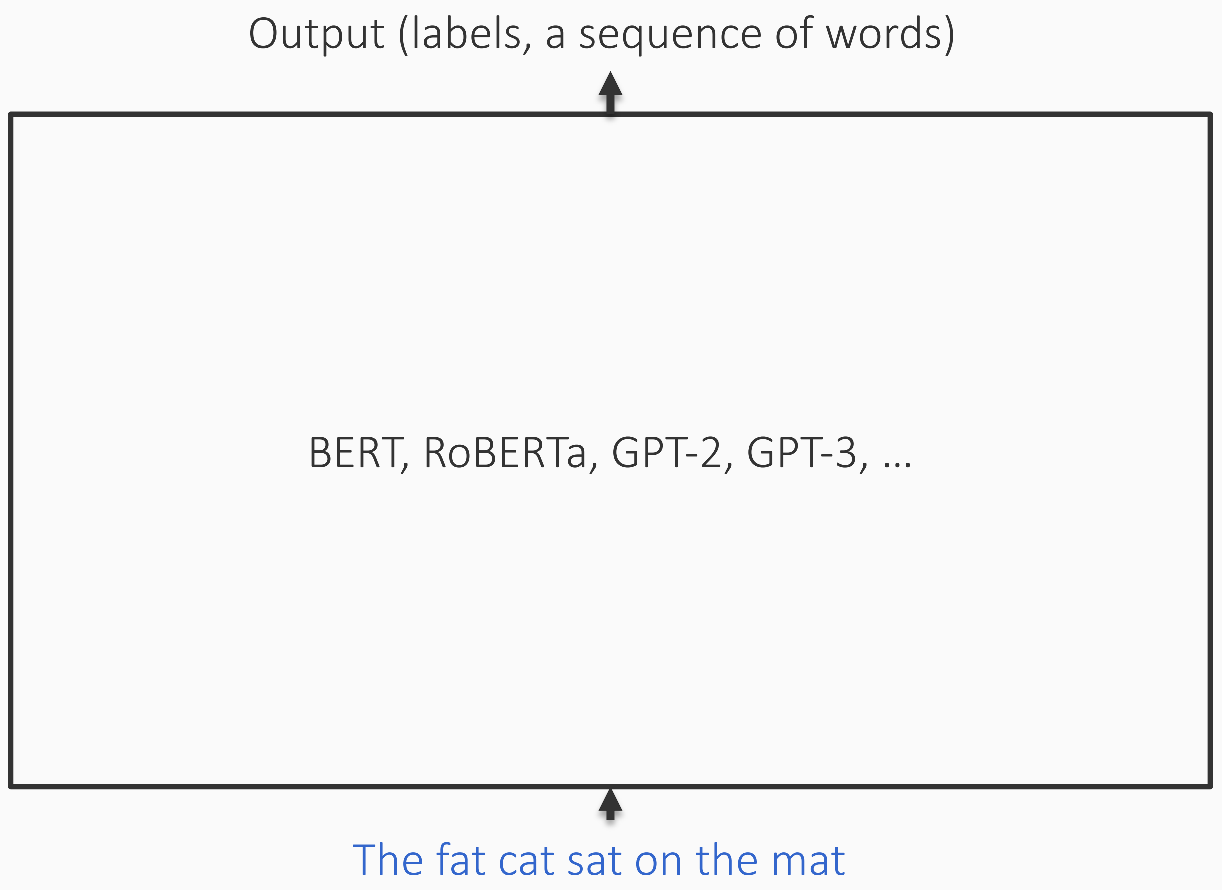


Figure 1: The Transformer - model architecture.

Let us unpack this


Output (labels, a sequence of words)
BERT, RoBERTa, GPT-2, GPT-3, …
The fat cat sat on the mat

Output (labels, a sequence of words)

BERT, RoBERTa, GPT-2, GPT-3, …

The fat cat sat on the mat

Output (labels, a sequence of words)

BERT, RoBERTa, GPT-2, GPT-3, ...

The, fat, cat, sat, on, the, mat

Tokenizer

The fat cat sat on the mat

Output (labels, a sequence of words)

BERT, RoBERTa, GPT-2, GPT-3, …

The, fat, cat, sat, on, the, mat

T = 7 tokens in this sequence.

Tokenizer

The fat cat sat on the mat

Output (labels, a sequence of words)

BERT, RoBERTa, GPT-2, GPT-3, ...

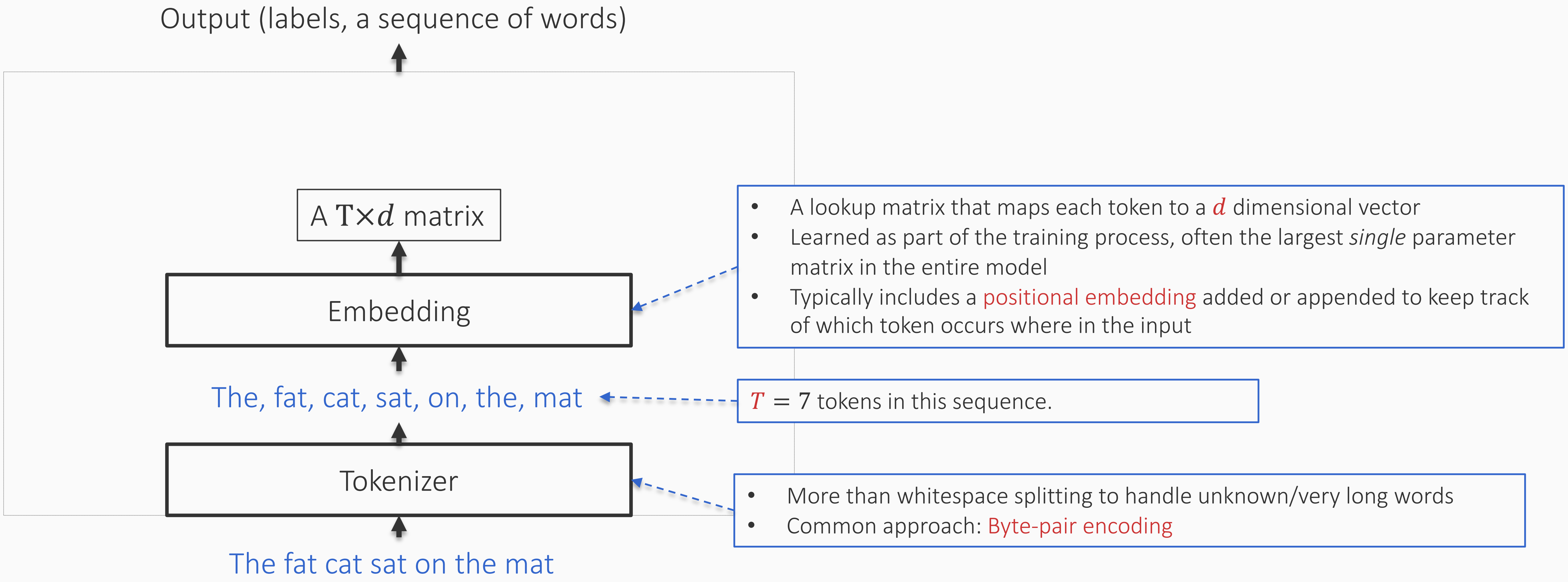
The, fat, cat, sat, on, the, mat

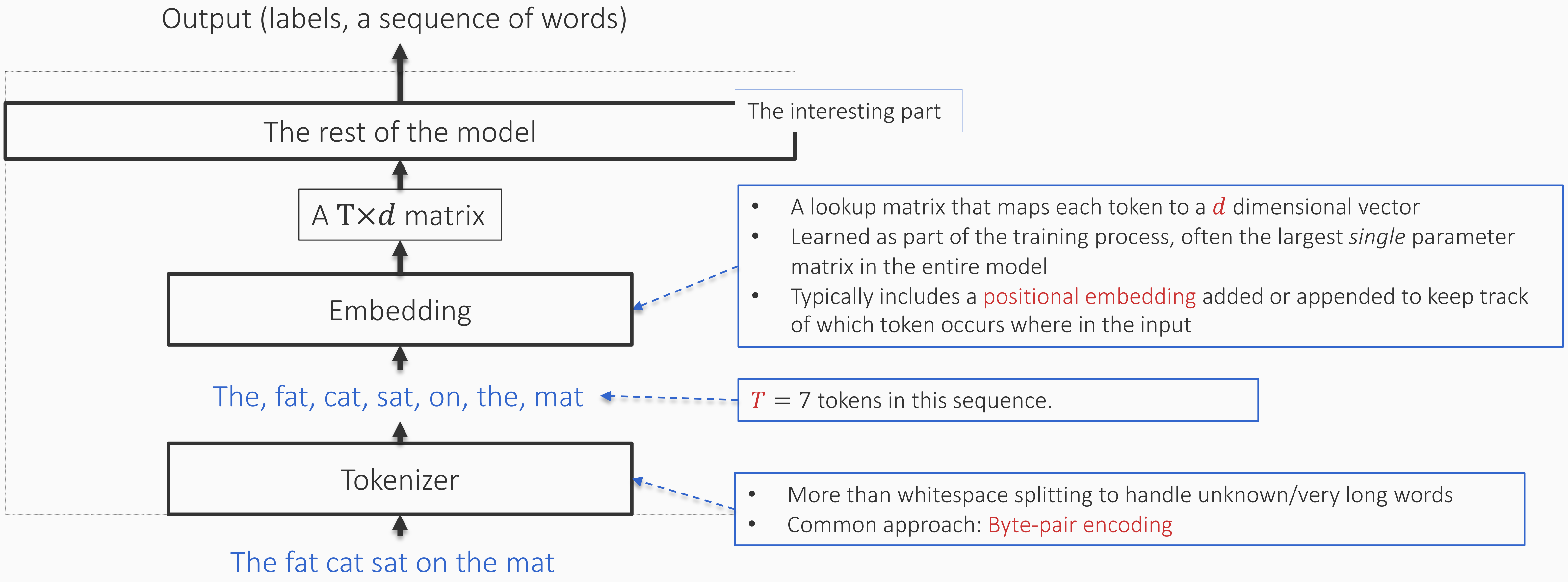
T = 7 tokens in this sequence.

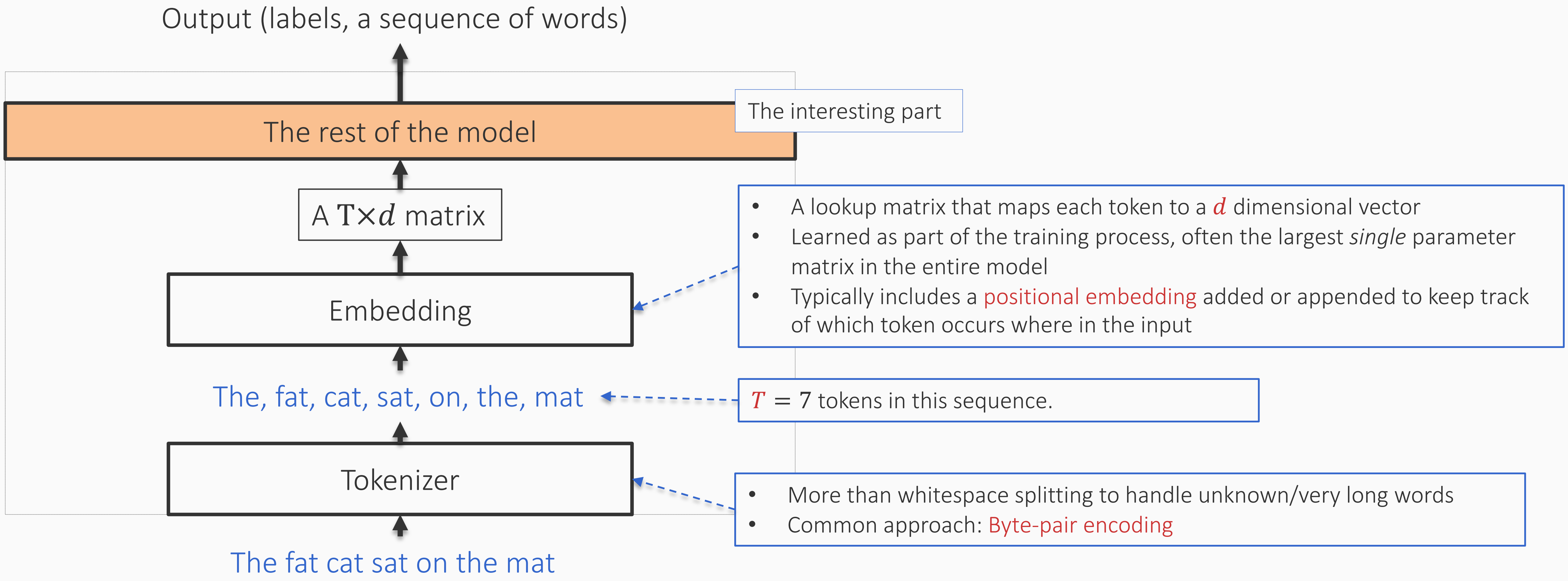
Tokenizer

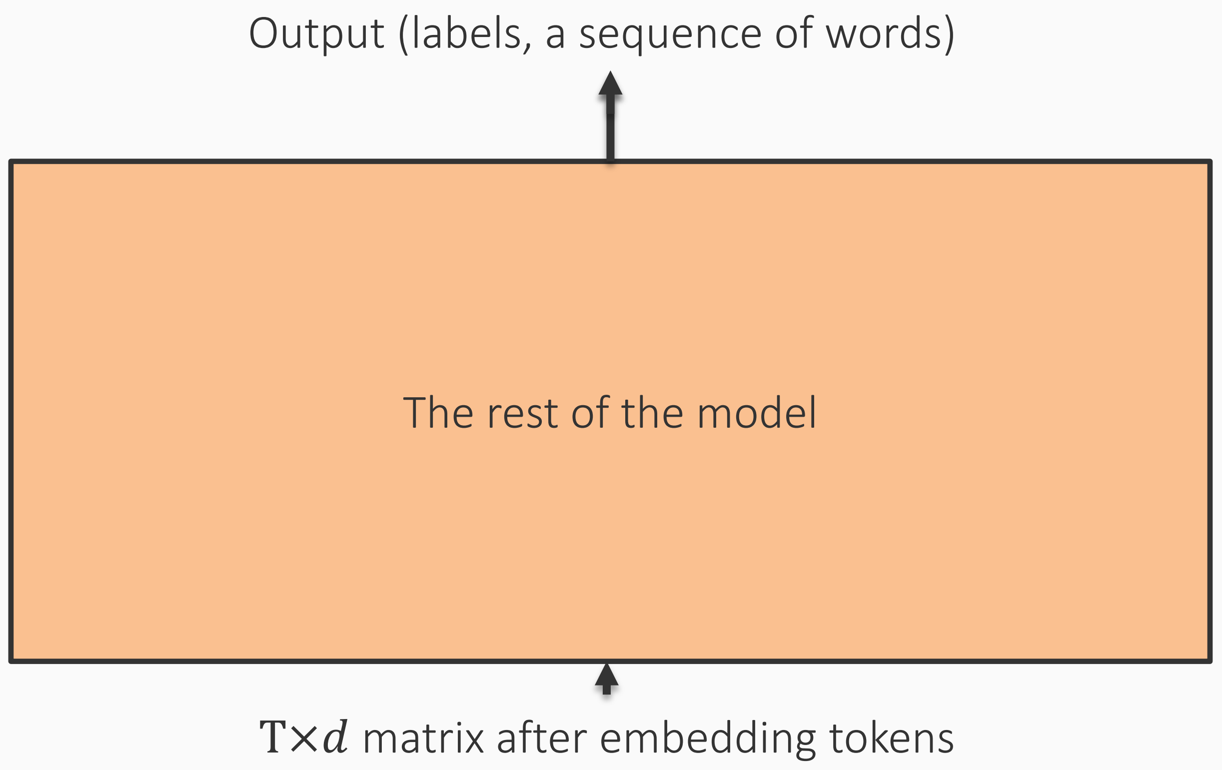
- More than whitespace splitting to handle unknown/very long words
- Common approach: Byte-pair encoding

The fat cat sat on the mat


Output (labels, a sequence of words)
A T×d matrix
Embedding
The, fat, cat, sat, on, the, mat
Tokenizer
The fat cat sat on the mat
• A lookup matrix that maps each token to a d dimensional vector
• Learned as part of the training process, often the largest *single* parameter matrix in the entire model
• Typically includes a positional embedding added or appended to keep track of which token occurs where in the input
T = 7 tokens in this sequence.
• More than whitespace splitting to handle unknown/very long words
• Common approach: Byte-pair encoding


Output (labels, a sequence of words)
The rest of the model
The interesting part
A T×d matrix
Embedding
• A lookup matrix that maps each token to a d dimensional vector
• Learned as part of the training process, often the largest *single* parameter matrix in the entire model
• Typically includes a positional embedding added or appended to keep track of which token occurs where in the input
The, fat, cat, sat, on, the, mat
$T = 7$ tokens in this sequence.
Tokenizer
• More than whitespace splitting to handle unknown/very long words
• Common approach: Byte-pair encoding
The fat cat sat on the mat


Output (labels, a sequence of words)
The rest of the model
The interesting part
A T×d matrix
Embedding
A lookup matrix that maps each token to a d dimensional vector
Learned as part of the training process, often the largest *single* parameter matrix in the entire model
Typically includes a positional embedding added or appended to keep track of which token occurs where in the input
The, fat, cat, sat, on, the, mat
T = 7 tokens in this sequence.
Tokenizer
More than whitespace splitting to handle unknown/very long words
Common approach: Byte-pair encoding
The fat cat sat on the mat

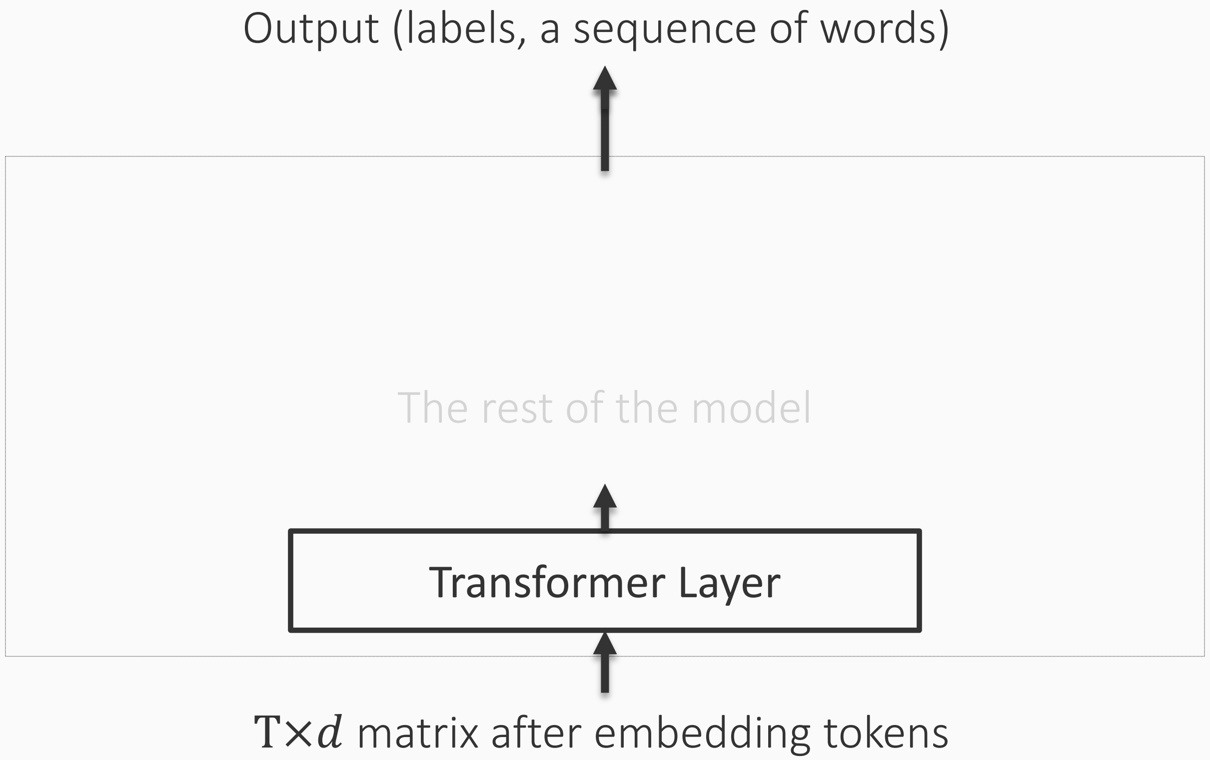

Output (labels, a sequence of words)
The rest of the model
T×d matrix after embedding tokens

Output (labels, a sequence of words)

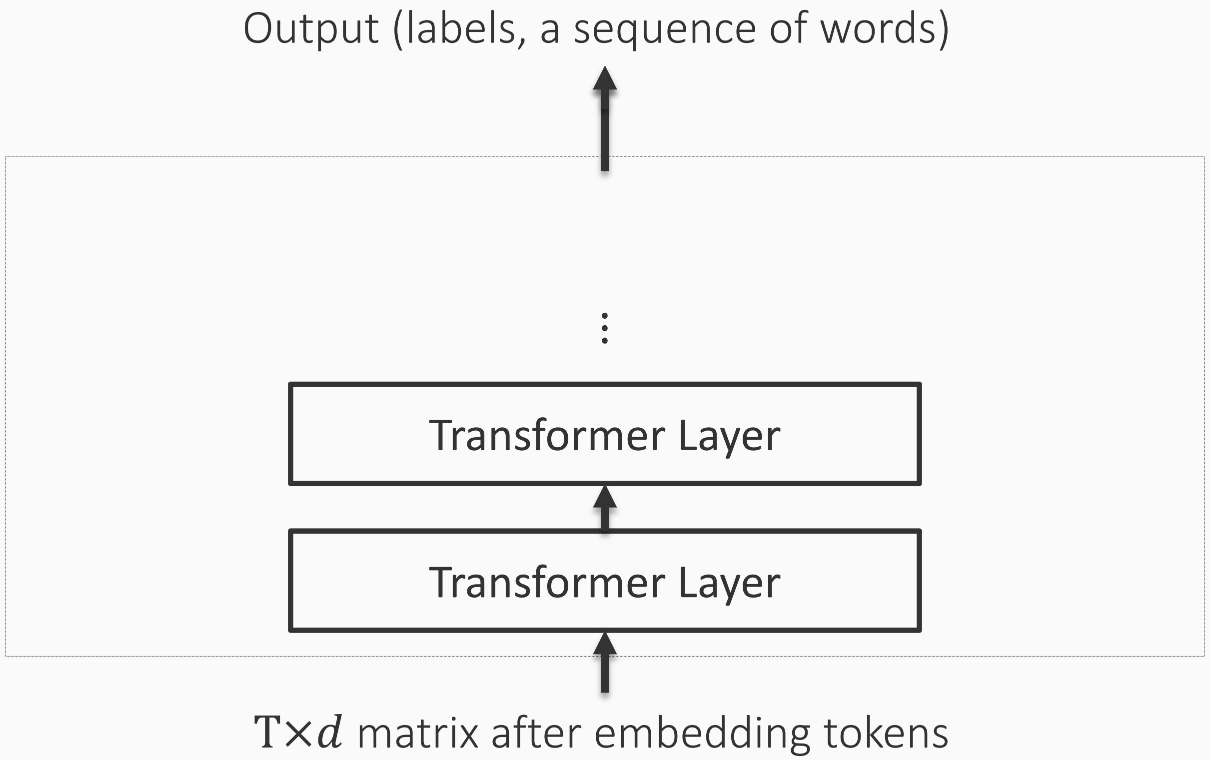
The rest of the model

$T \times d$ matrix after embedding tokens

T: Sequence length
d: Embedding size

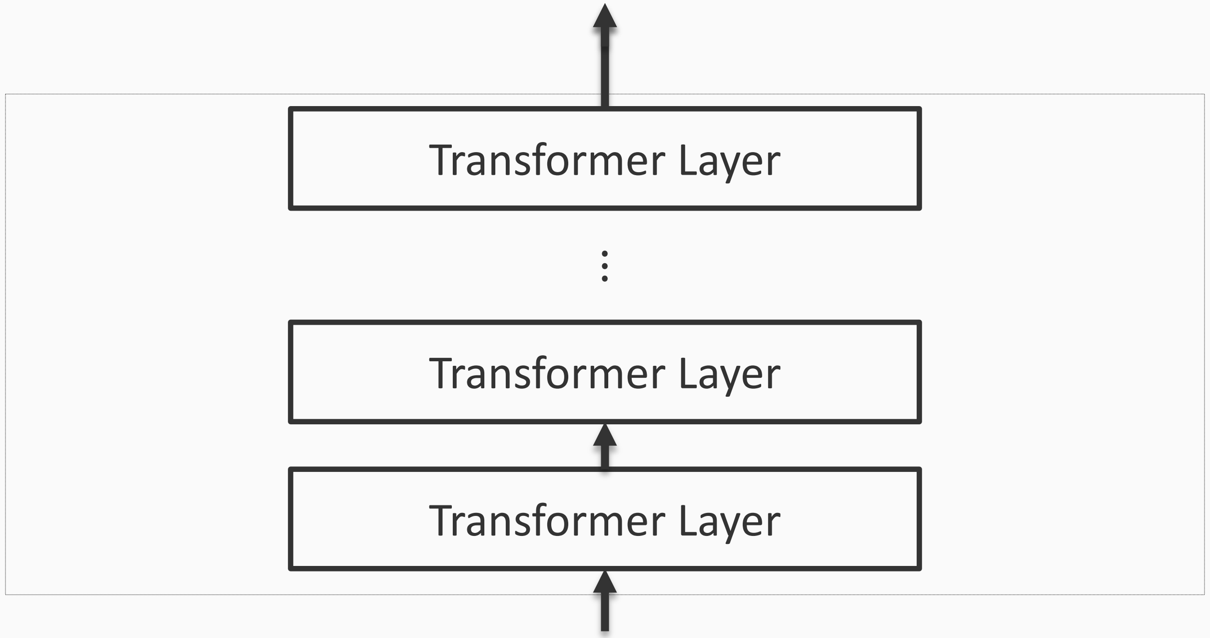


T: Sequence length
d: Embedding size



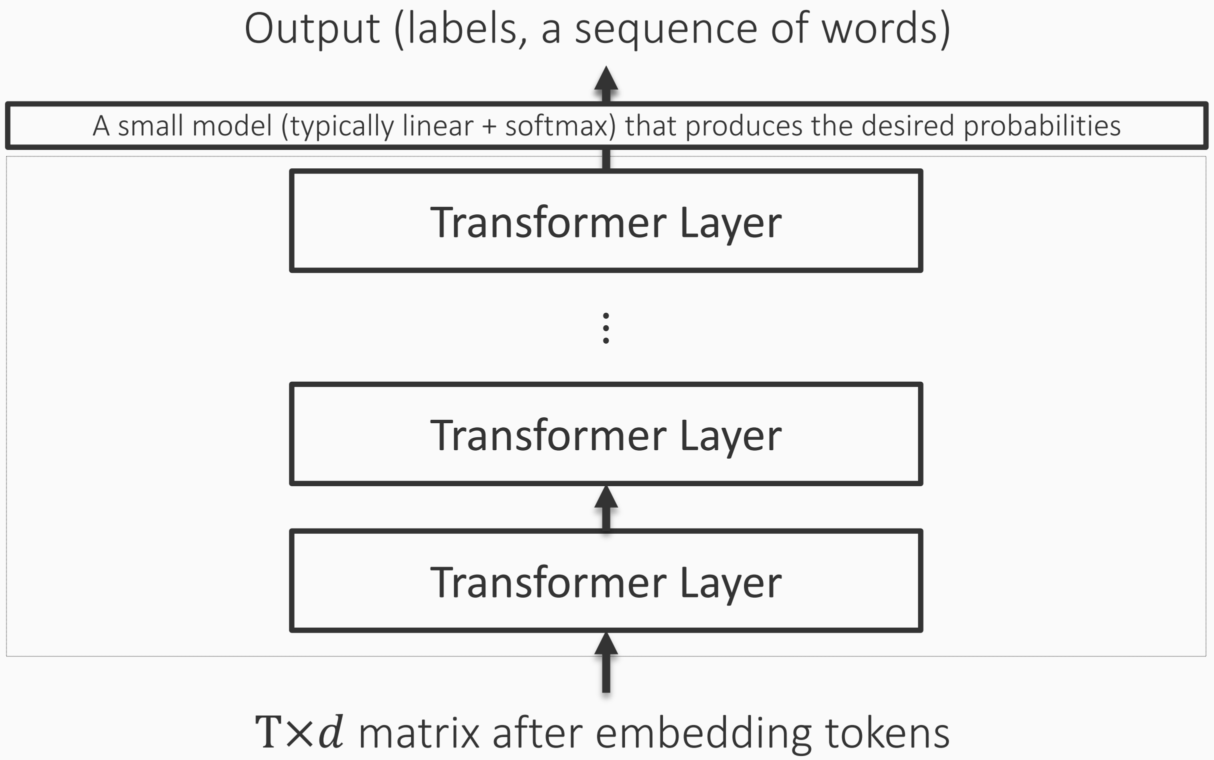
T: Sequence length
d: Embedding size

Output (labels, a sequence of words)

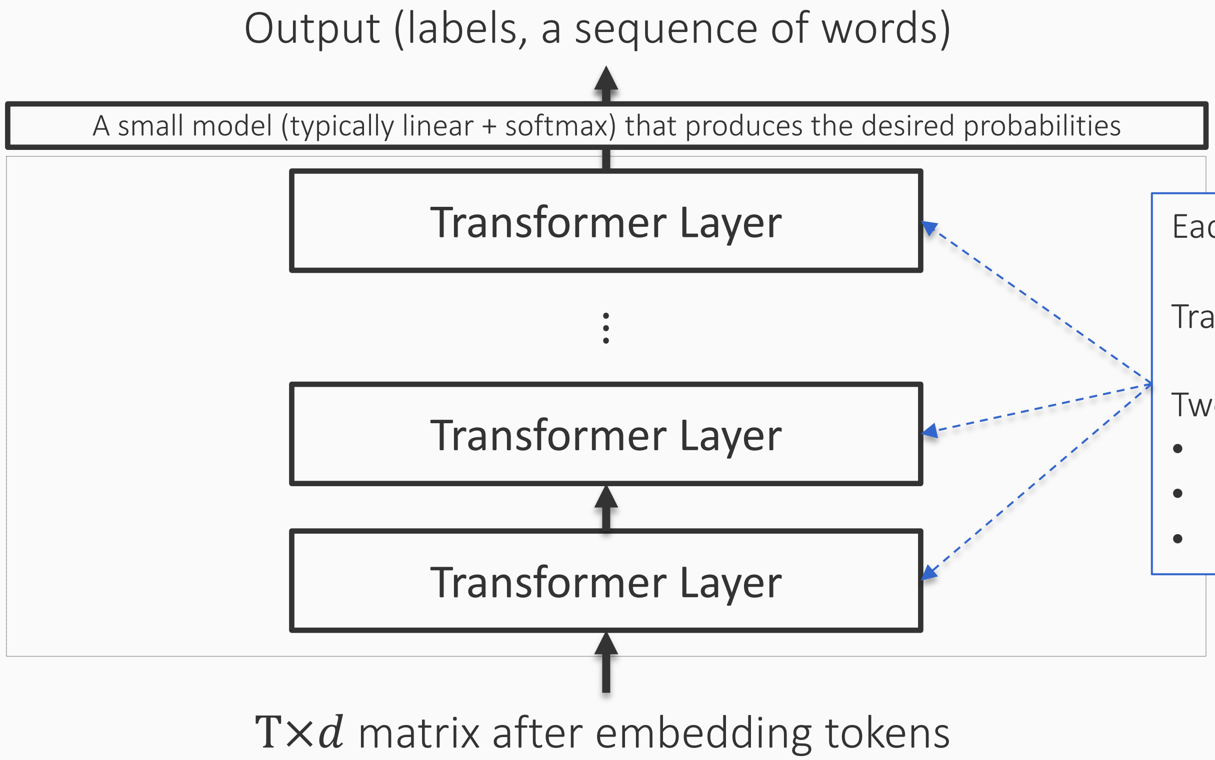


$\mathrm{T}\times d$ matrix after embedding tokens

T: Sequence length
d: Embedding size



T: Sequence length
d: Embedding size



Each transformer layer converts a $\mathrm{T}\times d$ matrix into a ”transformed” $\mathrm{T}\times d$ matrix

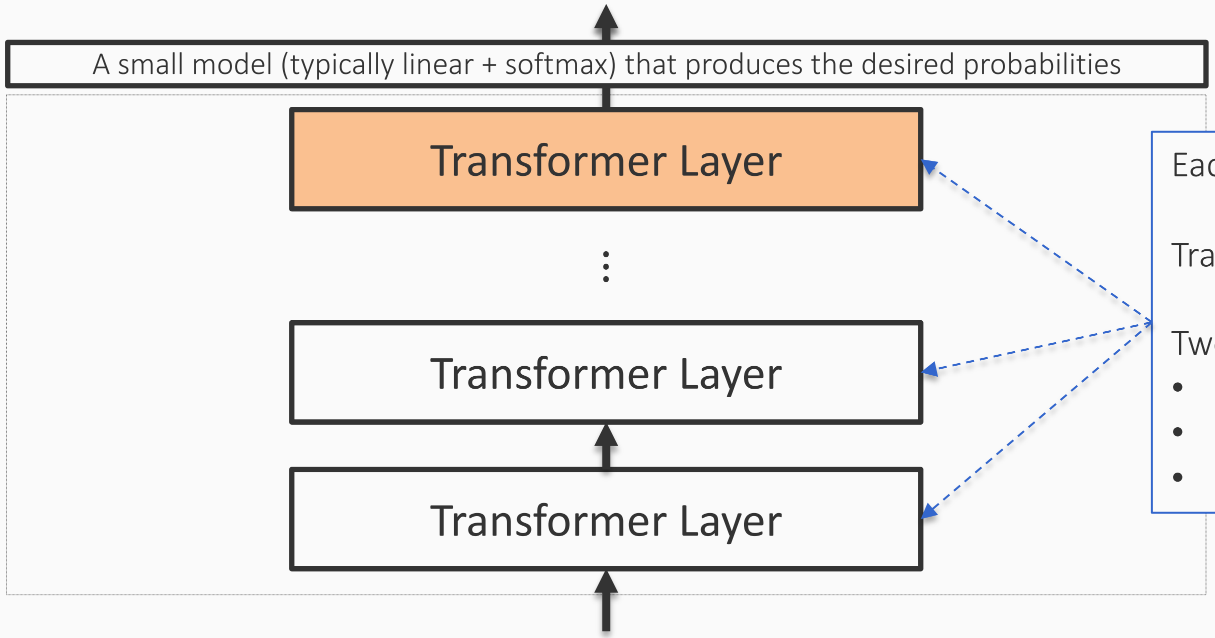
Transformer layers are structurally identical, but have their own parameters

Two different types of transformers in the original paper:

- Encoder: for BERT, etc whose goal is to embed text
- Decoder: for GPT etc whose goal is to generate text
- Minor differences between them

T: Sequence length
d: Embedding size

Output (labels, a sequence of words)



$\mathrm{T}\times d$ matrix after embedding tokens

Each transformer layer converts a $\mathrm{T}\times d$ matrix into a "transformed" $\mathrm{T}\times d$ matrix

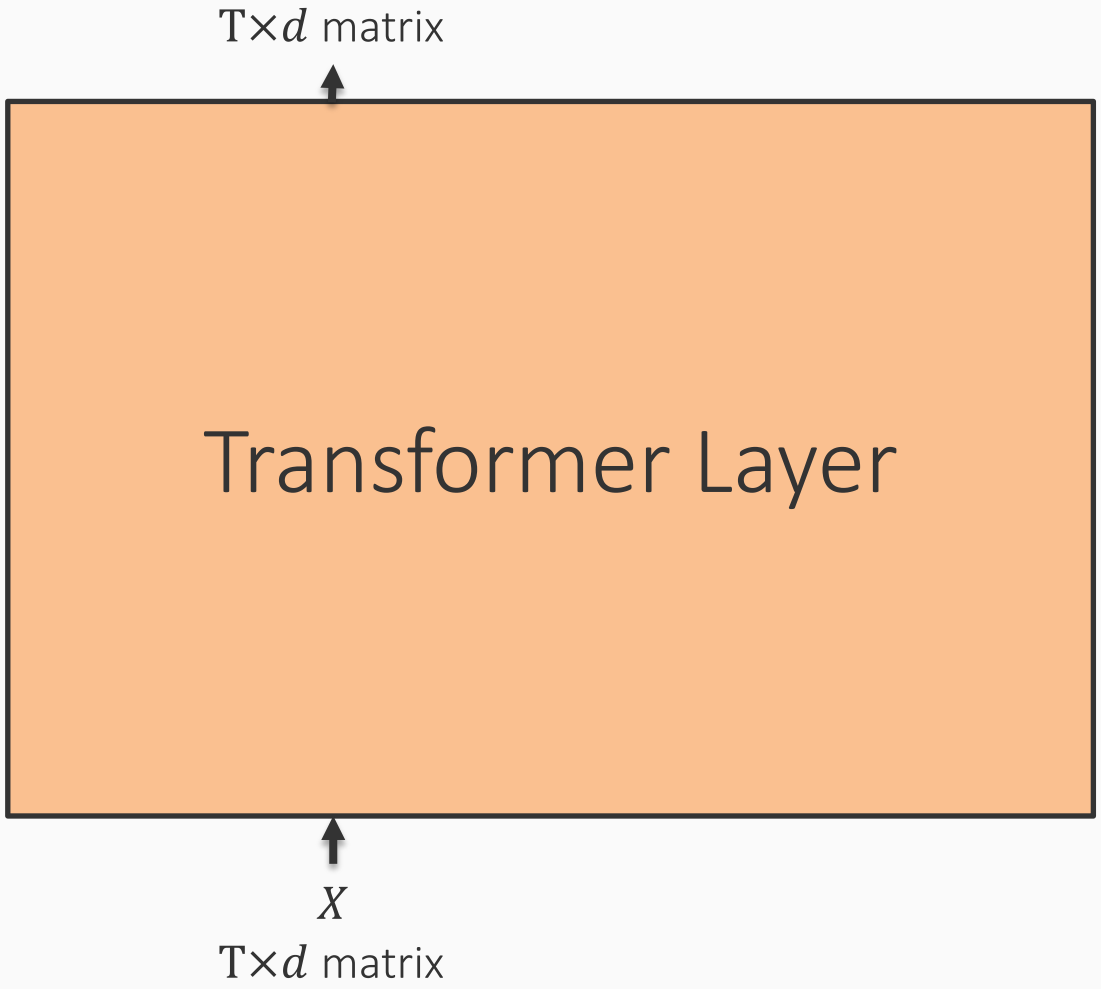
Transformer layers are structurally identical, but have their own parameters

Two different types of transformers in the original paper:

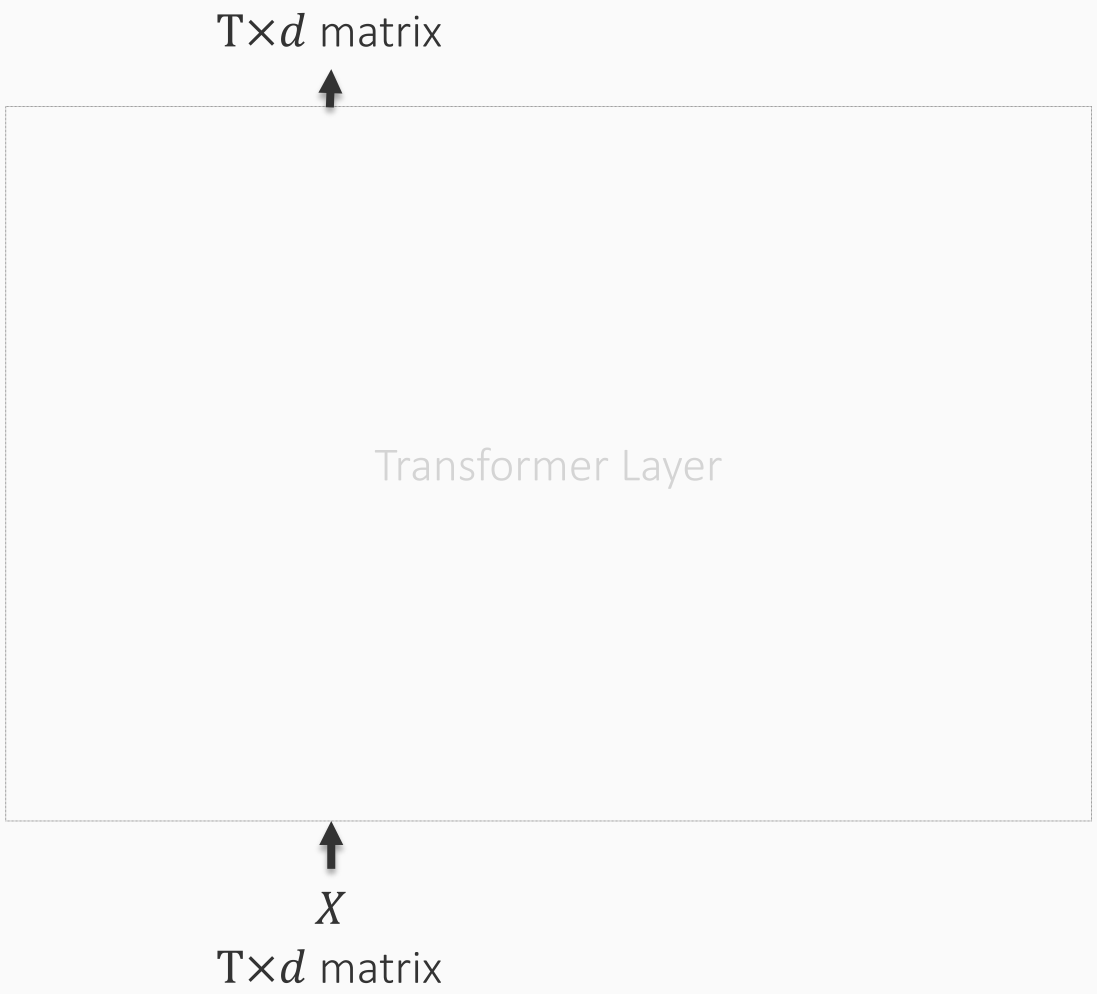
- Encoder: for BERT, etc whose goal is to embed text
- Decoder: for GPT etc whose goal is to generate text
- Minor differences between them

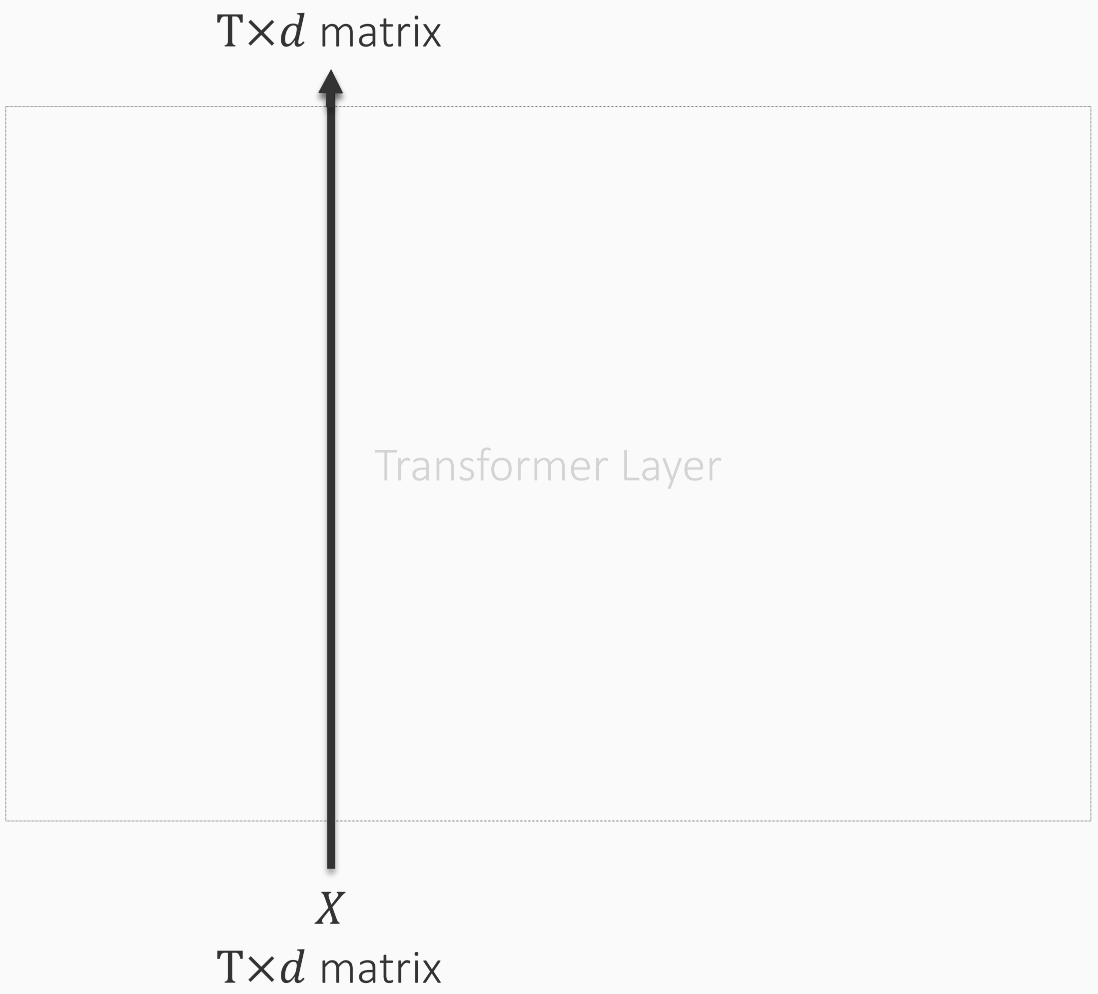
T: Sequence length

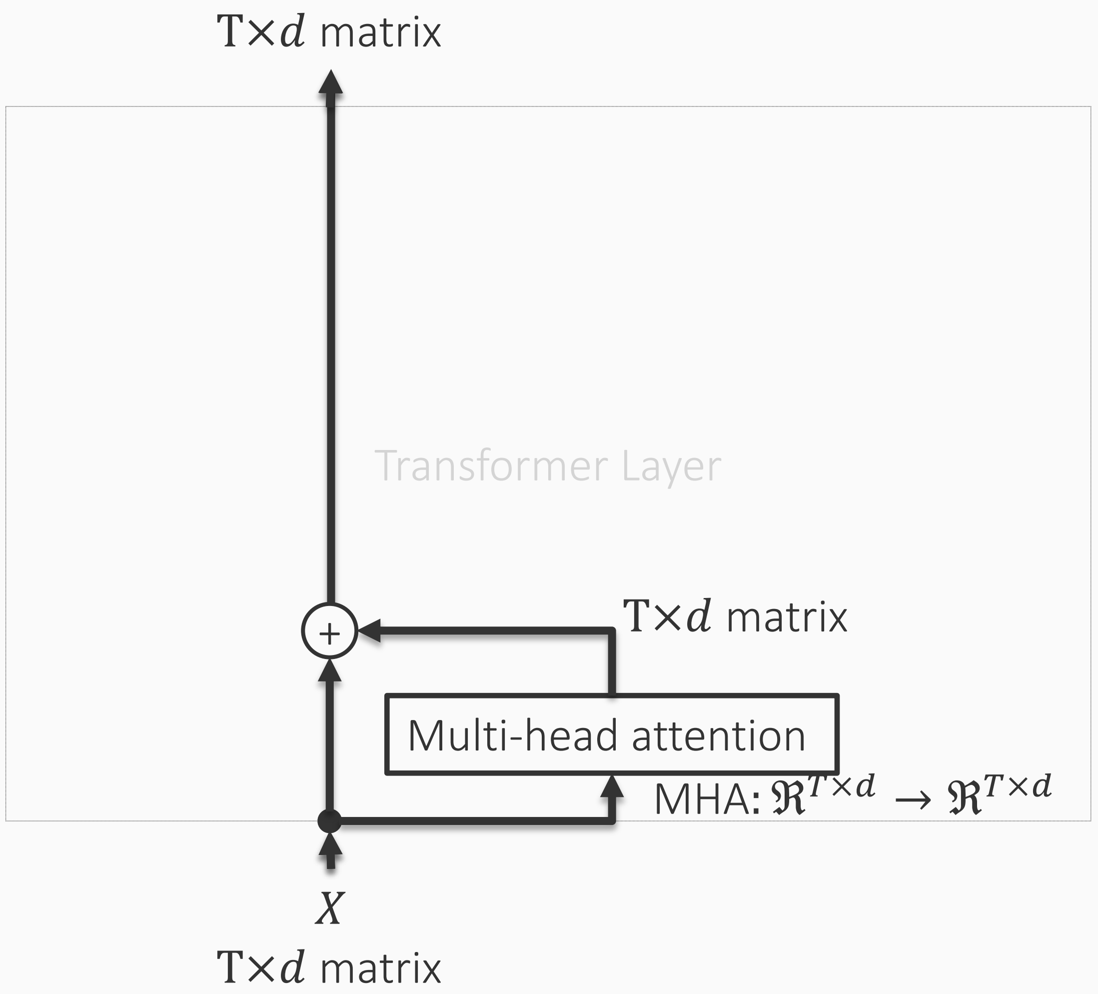
d: Embedding size



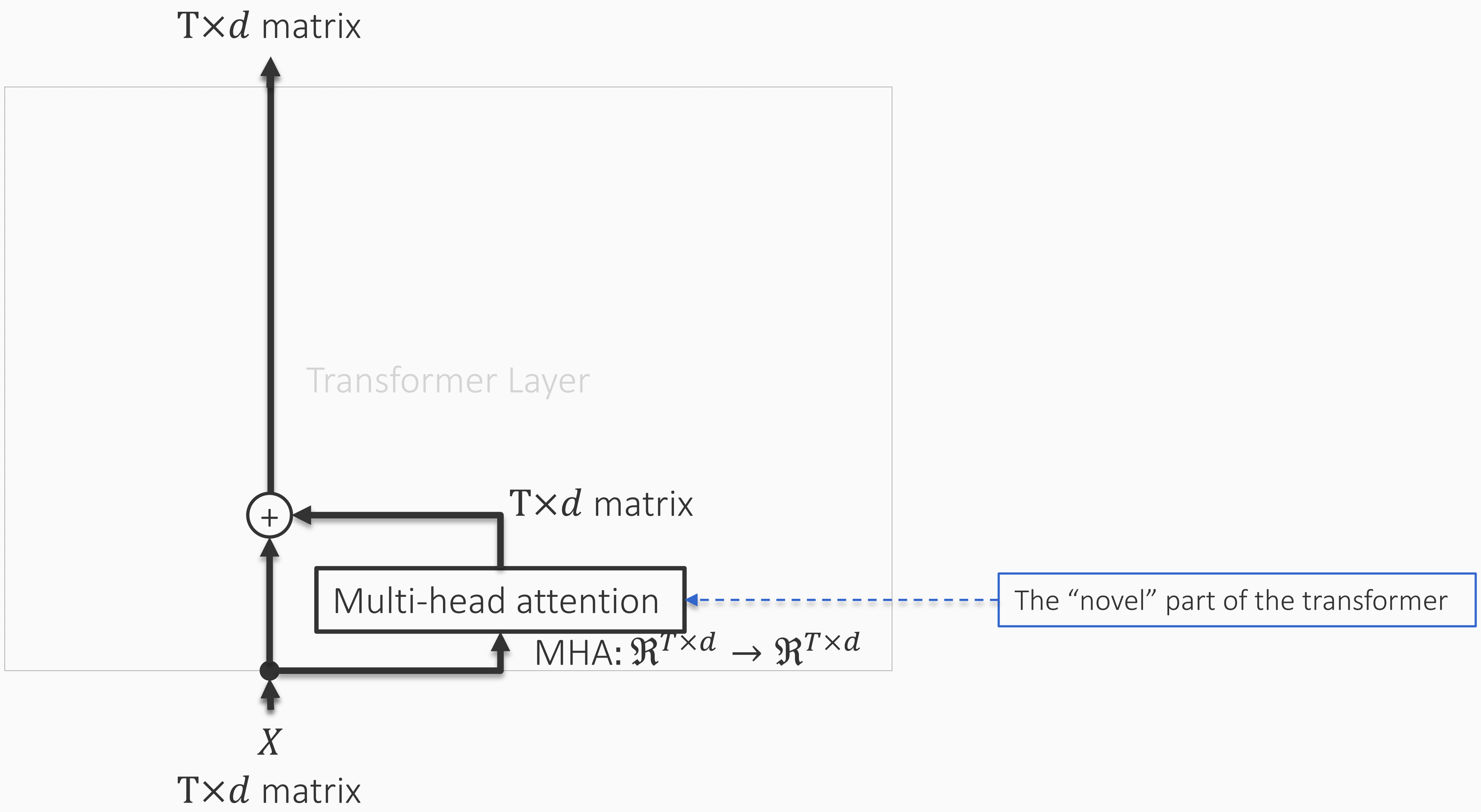
T×d matrix
Transformer Layer
X
T×d matrix

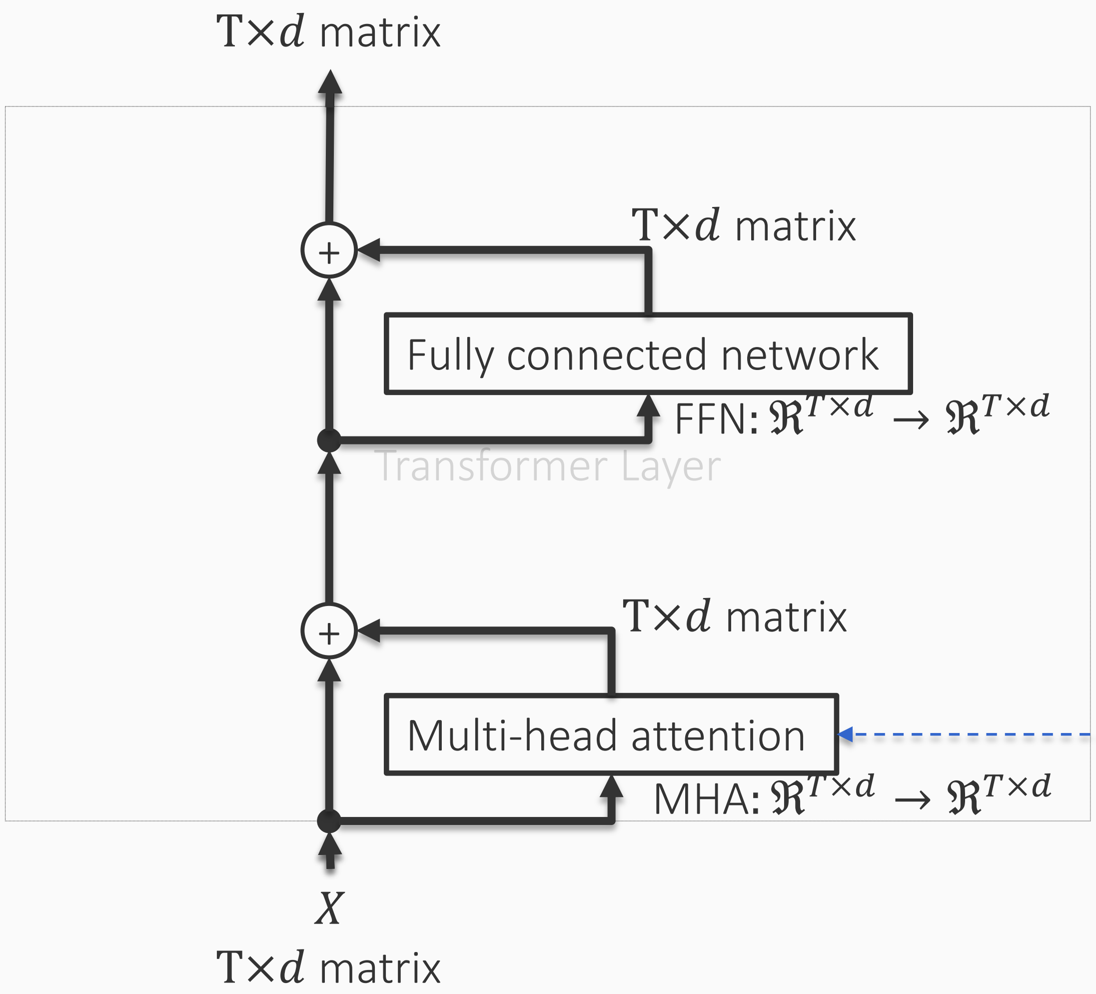

$T \times d$ matrix
Transformer Layer
X
$T \times d$ matrix


T×d matrix
Transformer Layer
X
T×d matrix



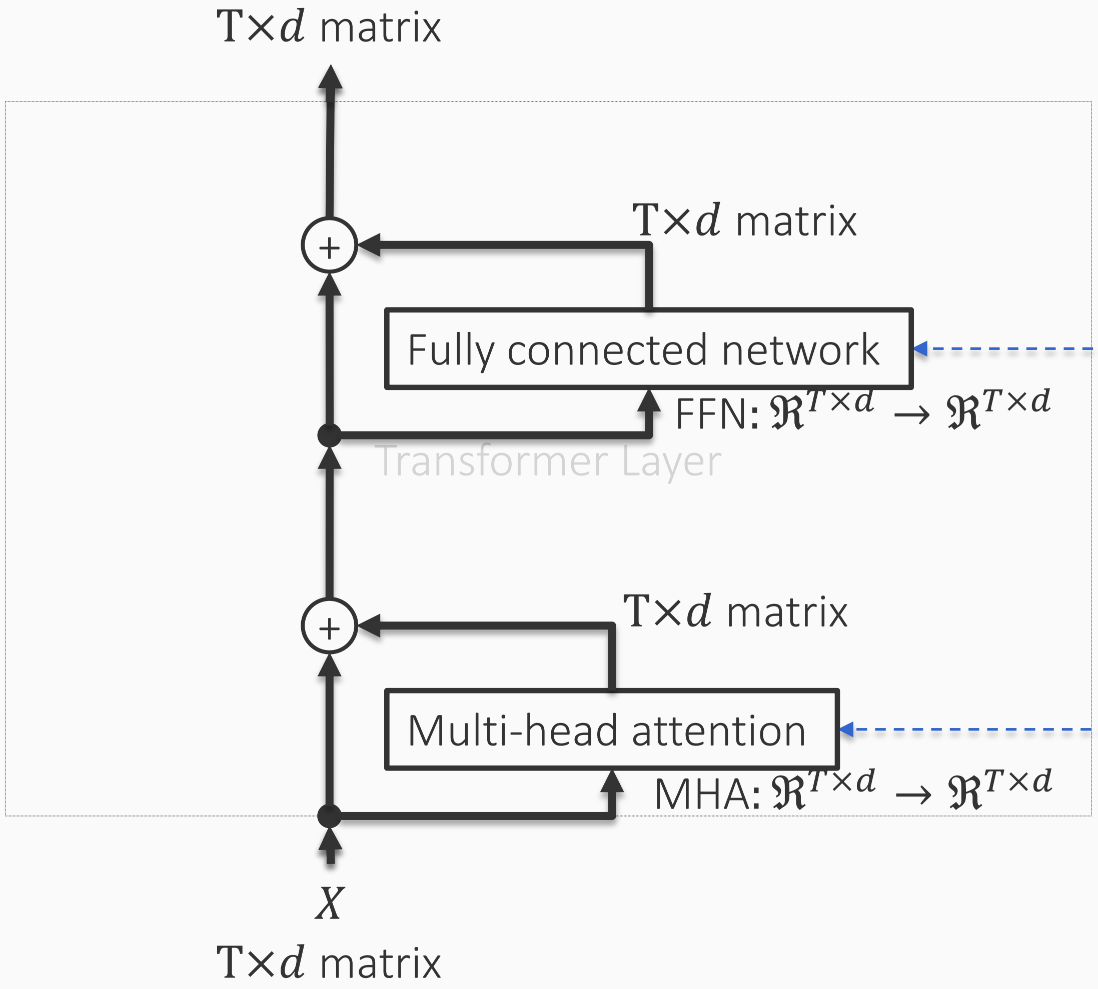
A residual connection


T×d matrix
Transformer Layer
T×d matrix
+
Multi-head attention
MHA: $\Re^{T\times d} \rightarrow \Re^{T\times d}$
The "novel" part of the transformer
X
T×d matrix



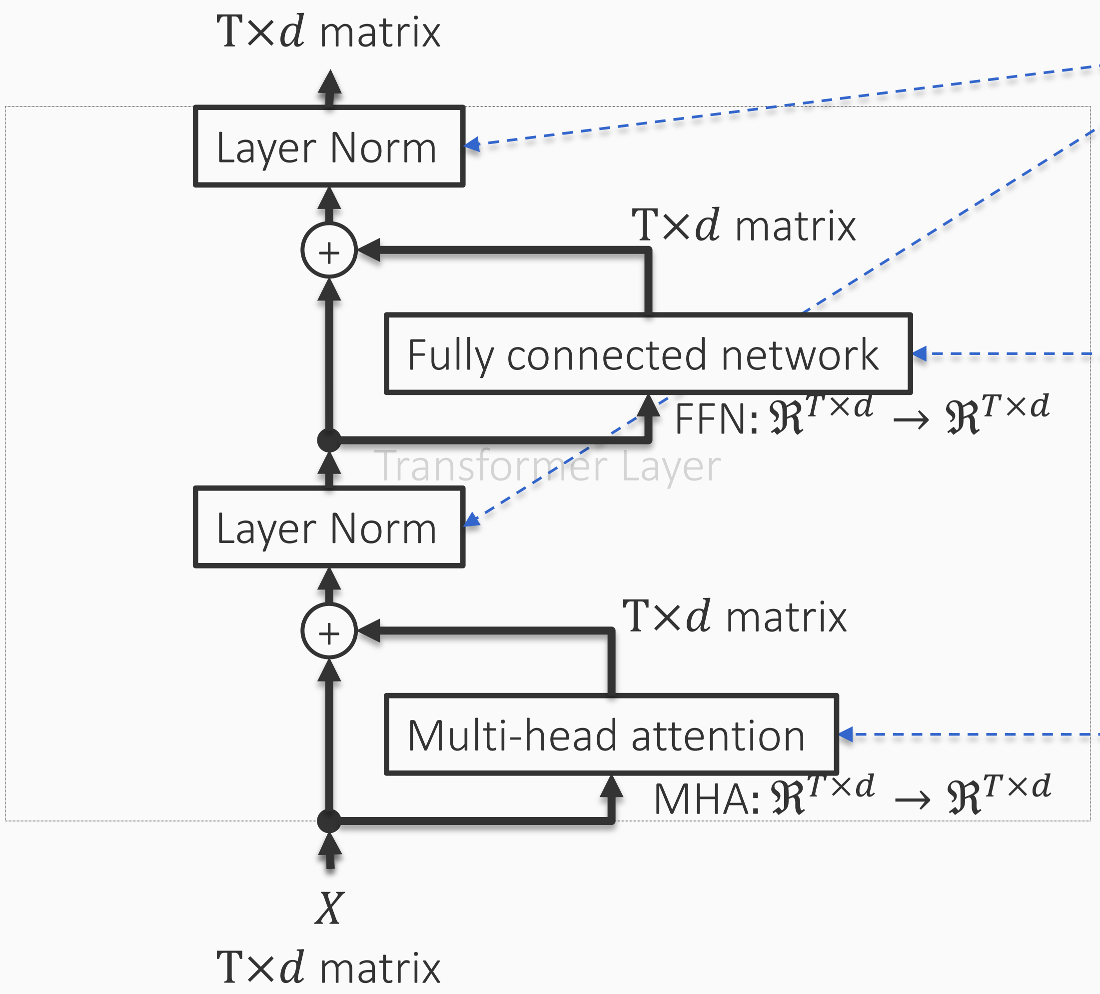
A residual connection

The "novel" part of the transformer



- A standard two layer neural network with GELU, ReLU (or sometimes other) activation, applied to each row separately
  - Trivially parallelizable over the rows
- Hidden layer can have dimensionality larger than d (e.g. $4d$)

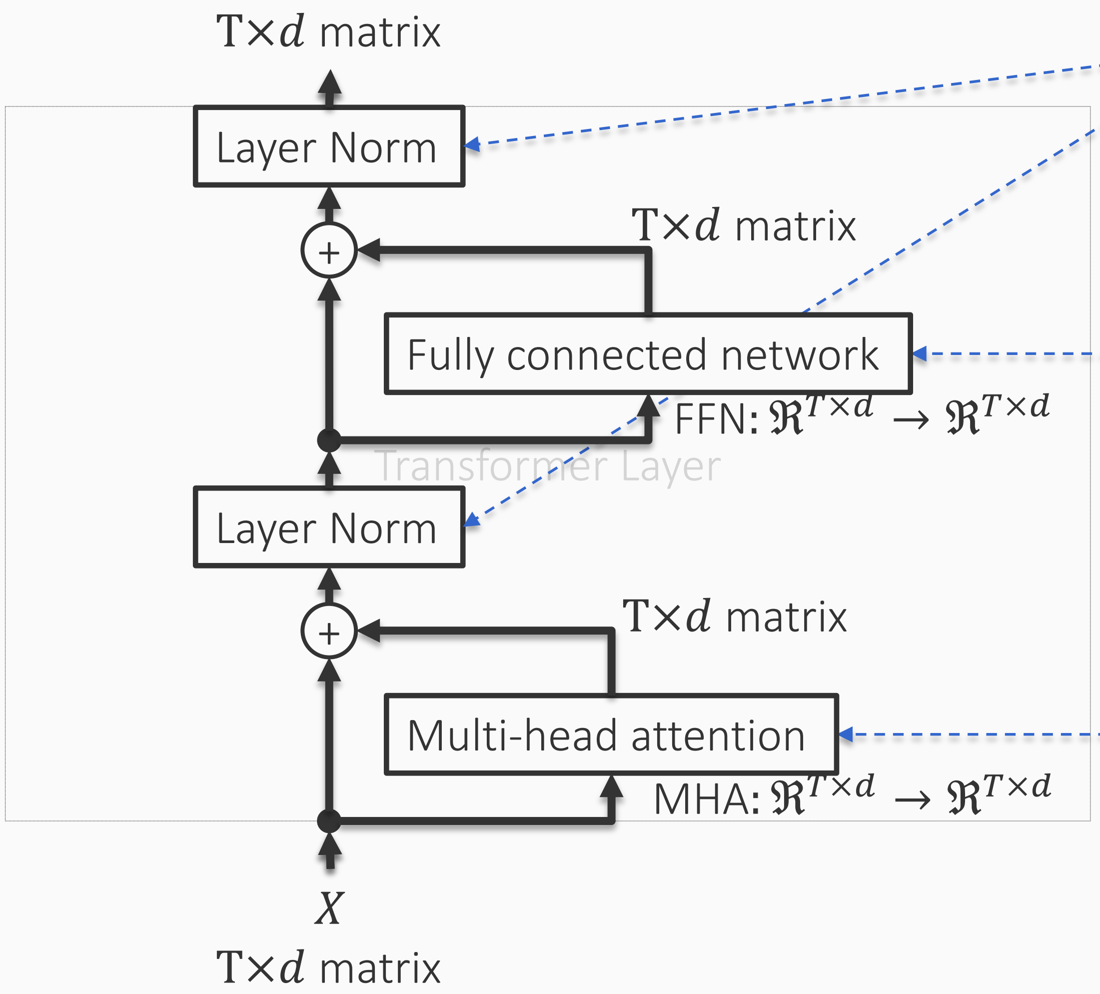
The "novel" part of the transformer



- Normalizes each column of the input matrix so that the elements have zero mean and unit variance
- Leads to more stable optimization

- A standard two layer neural network with GELU, ReLU (or sometimes other) activation, applied to each row separately
    - Trivially parallelizable over the rows
- Hidden layer can have dimensionality larger than d (e.g. $4d$)

The “novel” part of the transformer

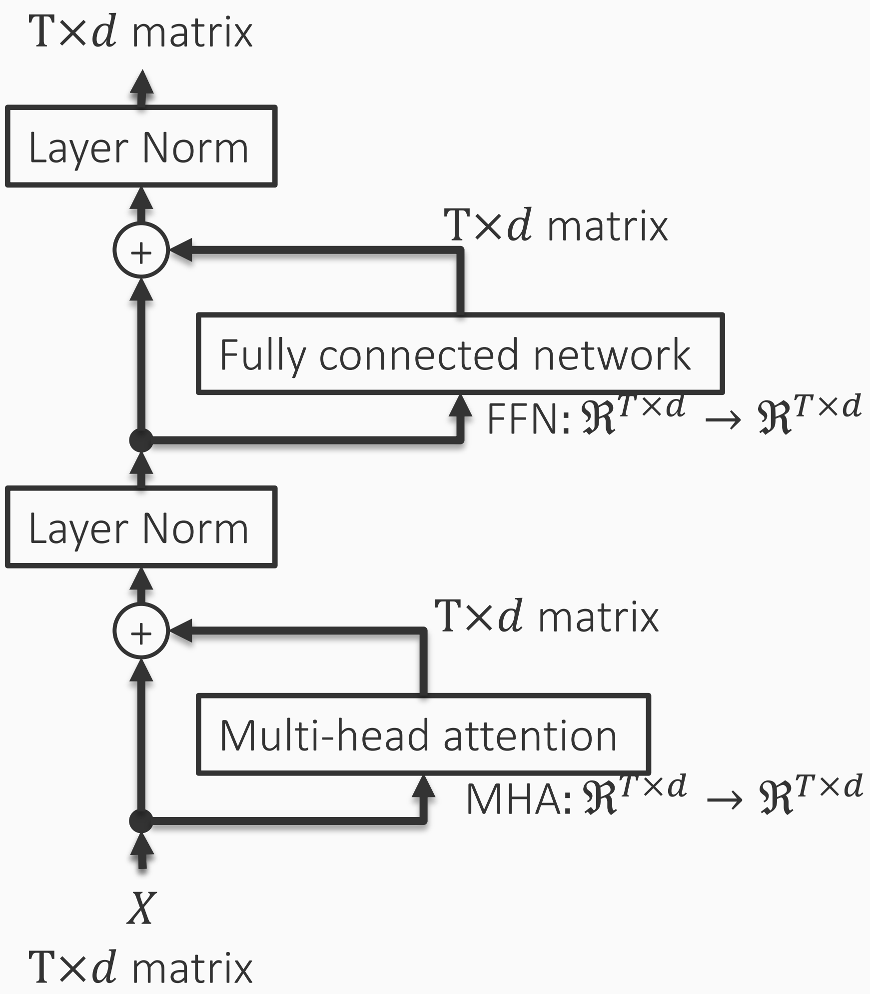


- Normalizes each column of the input matrix so that the elements have zero mean and unit variance
- Leads to more stable optimization

- A standard two layer neural network with GELU, ReLU (or sometimes other) activation, applied to each row separately
  - Trivially parallelizable over the rows
- Hidden layer can have dimensionality larger than d (e.g. $4d$)

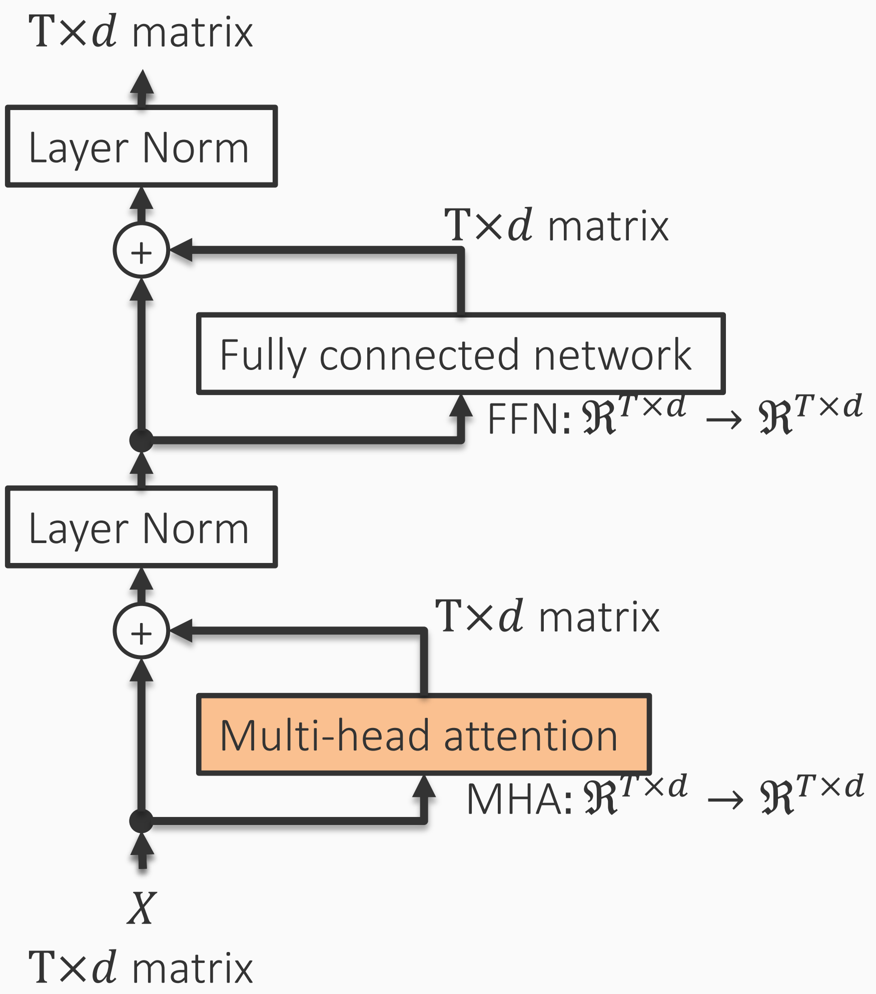
The “novel” part of the transformer

This is the transformer encoder. The decoder transformer has a bit more detail. We will encounter the details later



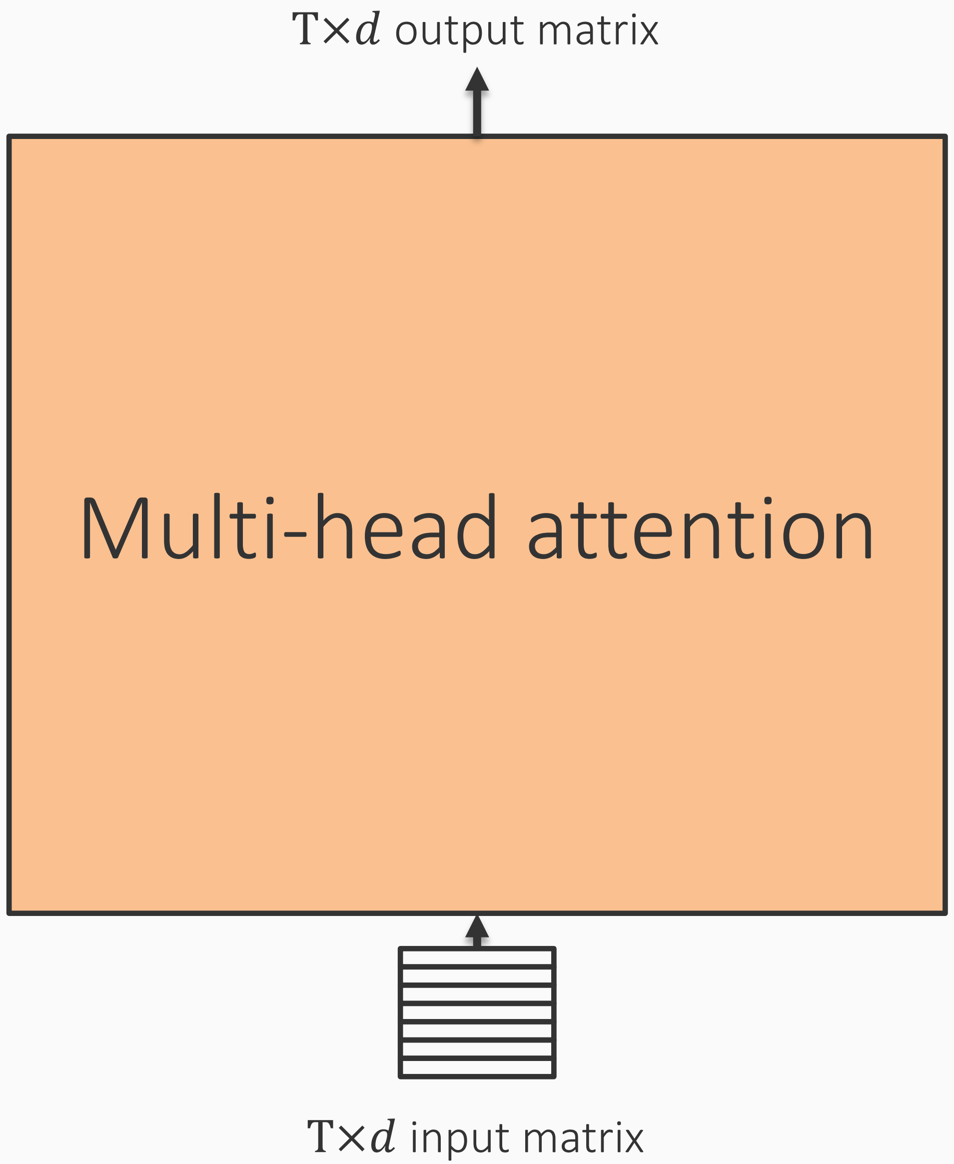

```
def transformer_layer(X):
    X1 = layer_norm1(X + multi_head_attention(X))
    X2 = layer_norm2(X1 + fully_connected(X1))
    return X2
```

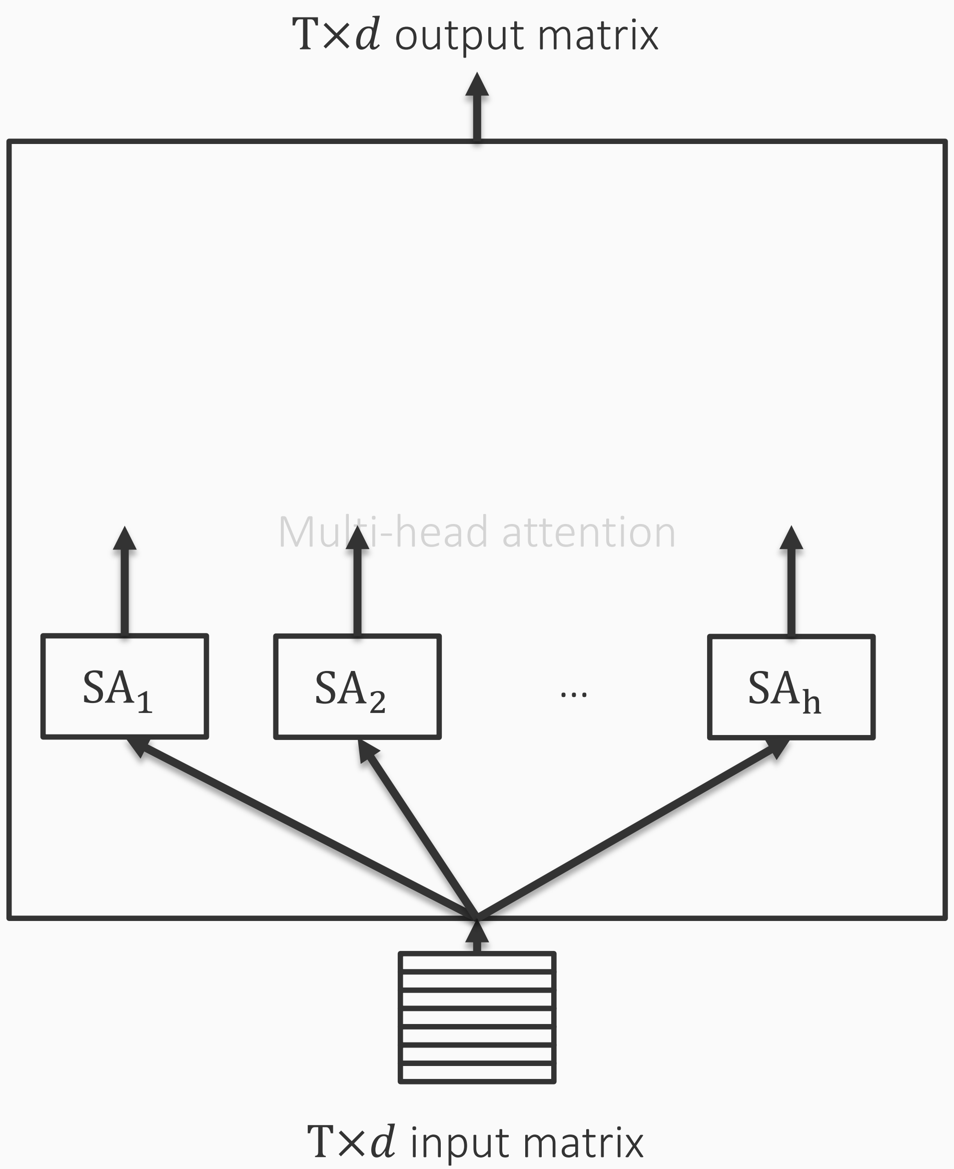
$$X_1 = \mathrm{LayerNorm}\big(X + MHA(X)\big)$$
$$\mathrm{Result} = \mathrm{LayerNorm}\big(X_1 + FFN(X_1)\big)$$



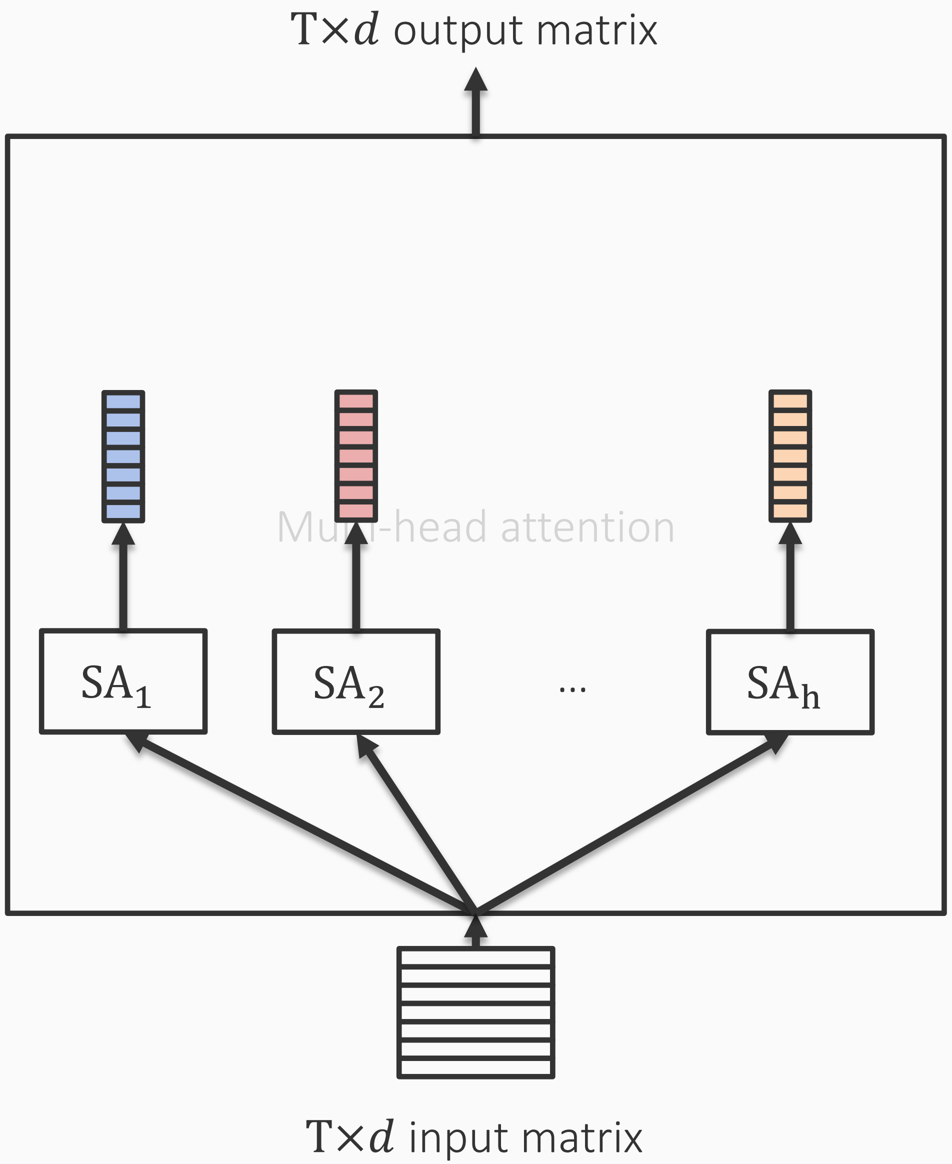

```python
def transformer_layer(X):
    X1 = layer_norm1(X + multi_head_attention(X))
    X2 = layer_norm2(X1 + fully_connected(X1))
    return X2
```

$$
\begin{aligned}
X_1 &= \text{LayerNorm}\big(X + MHA(X)\big)\\
\text{Result} &= \text{LayerNorm}\big(X_1 + FFN(X_1)\big)
\end{aligned}
$$


T×d output matrix
Multi-head attention
T×d input matrix

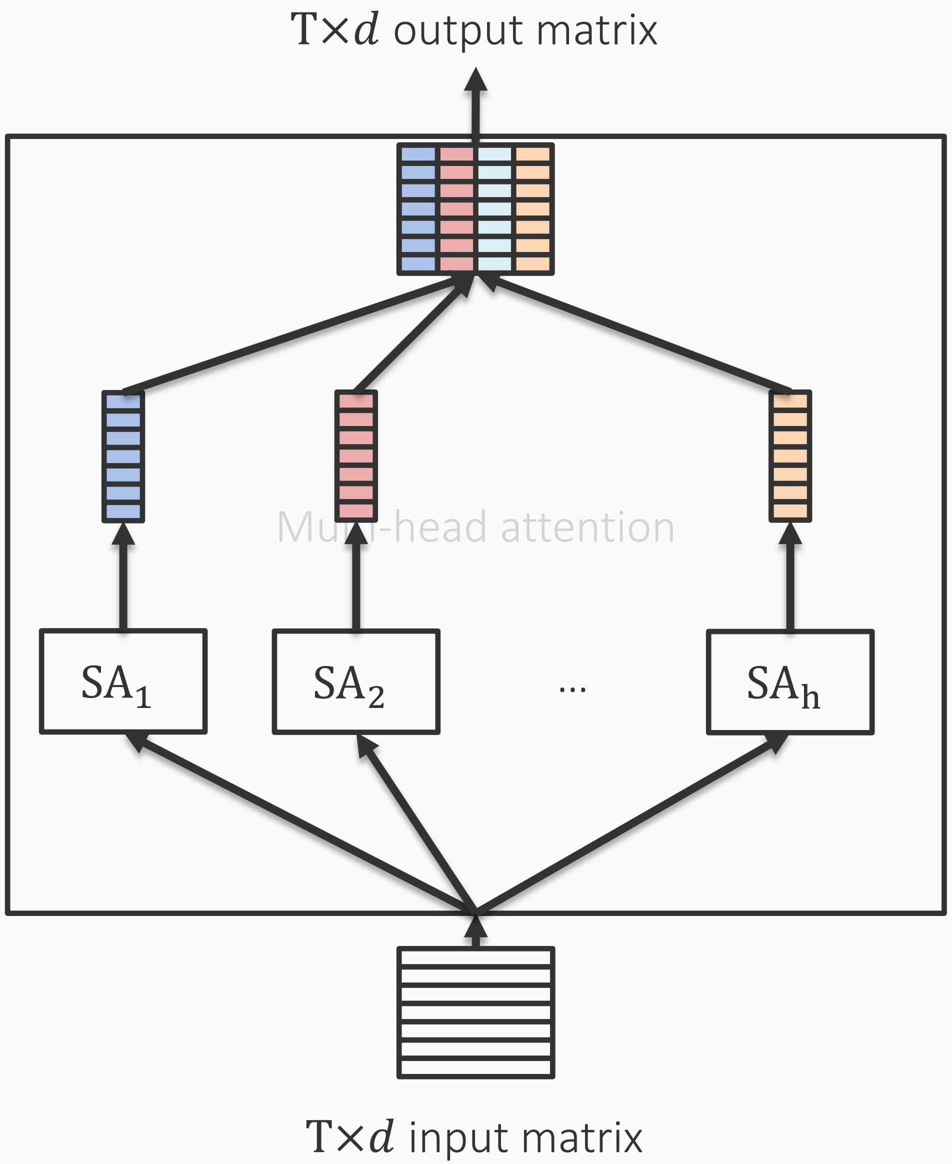


- h self-attention (SA) networks
- Analogous to channels in a CNN



Each produces a matrix of size $T \times \frac{d}{h}$

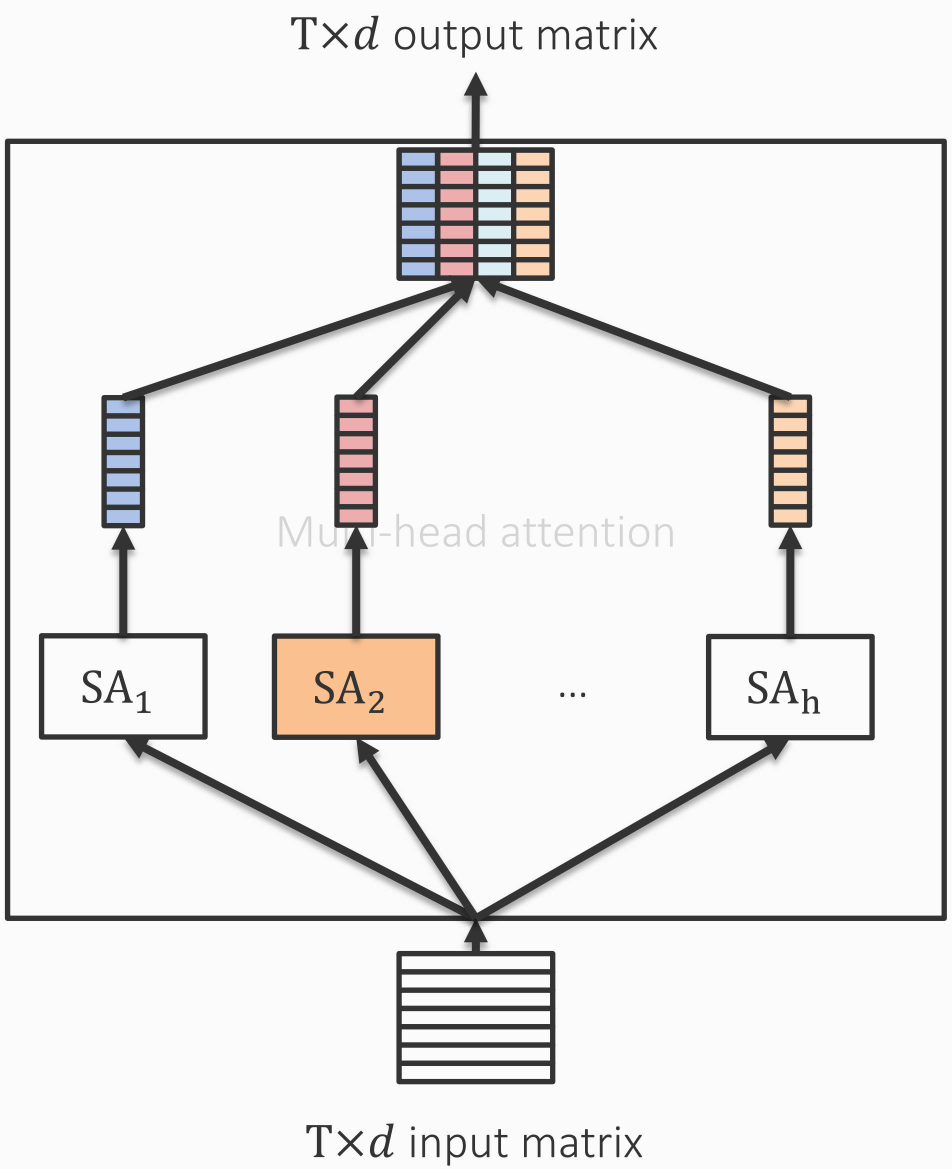
- h self-attention (SA) networks
- Analogous to channels in a CNN



These matrices are simply stacked to produce the output

Each produces a matrix of size $T \times \frac{d}{h}$

- h self-attention (SA) networks
- Analogous to channels in a CNN



These matrices are simply stacked to produce the output

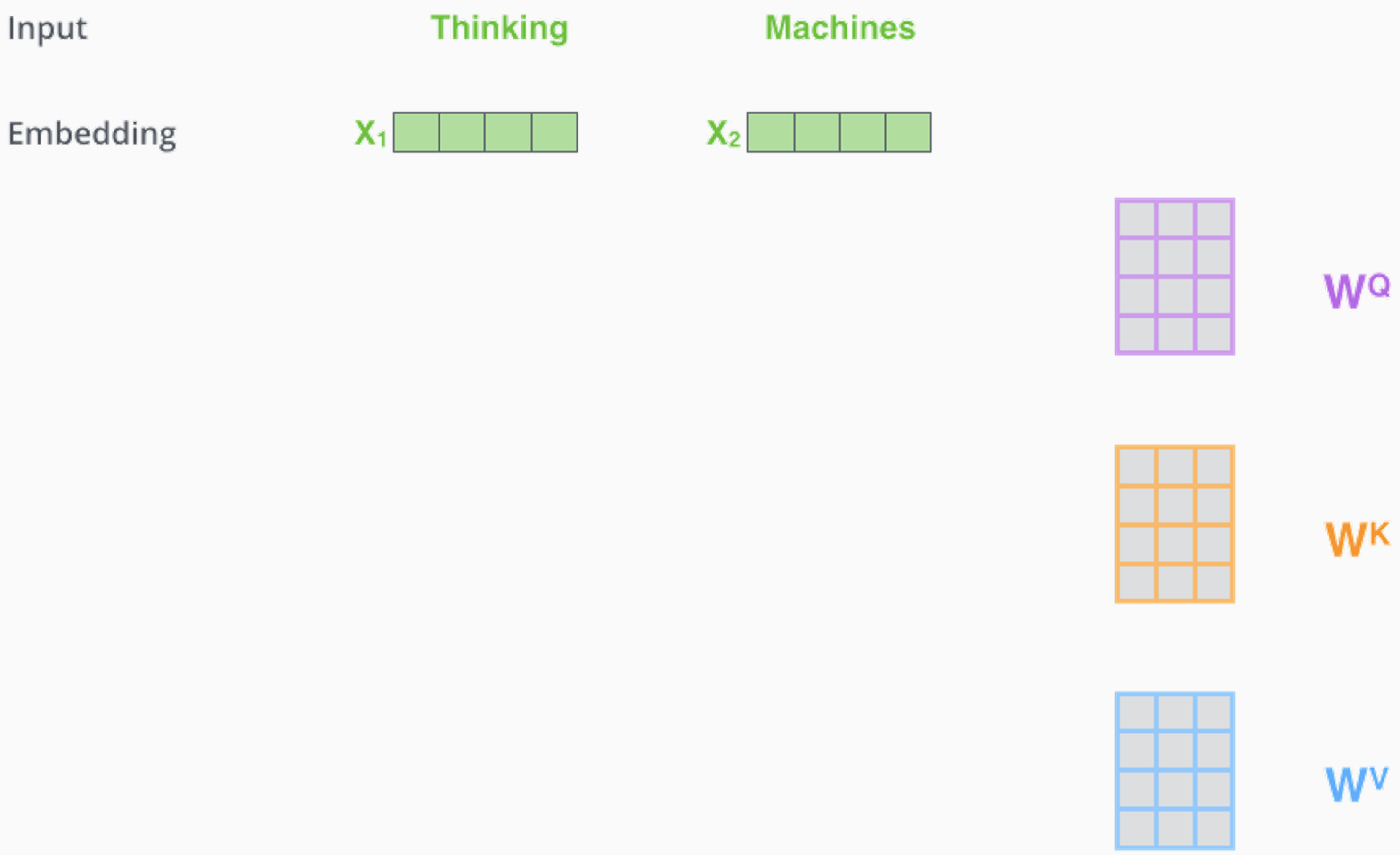
Each produces a matrix of size $T \times \frac{d}{h}$

- h self-attention (SA) networks
- Analogous to channels in a CNN

Self attention: An example

Images from https://jalammar.github.io/illustrated-transformer/

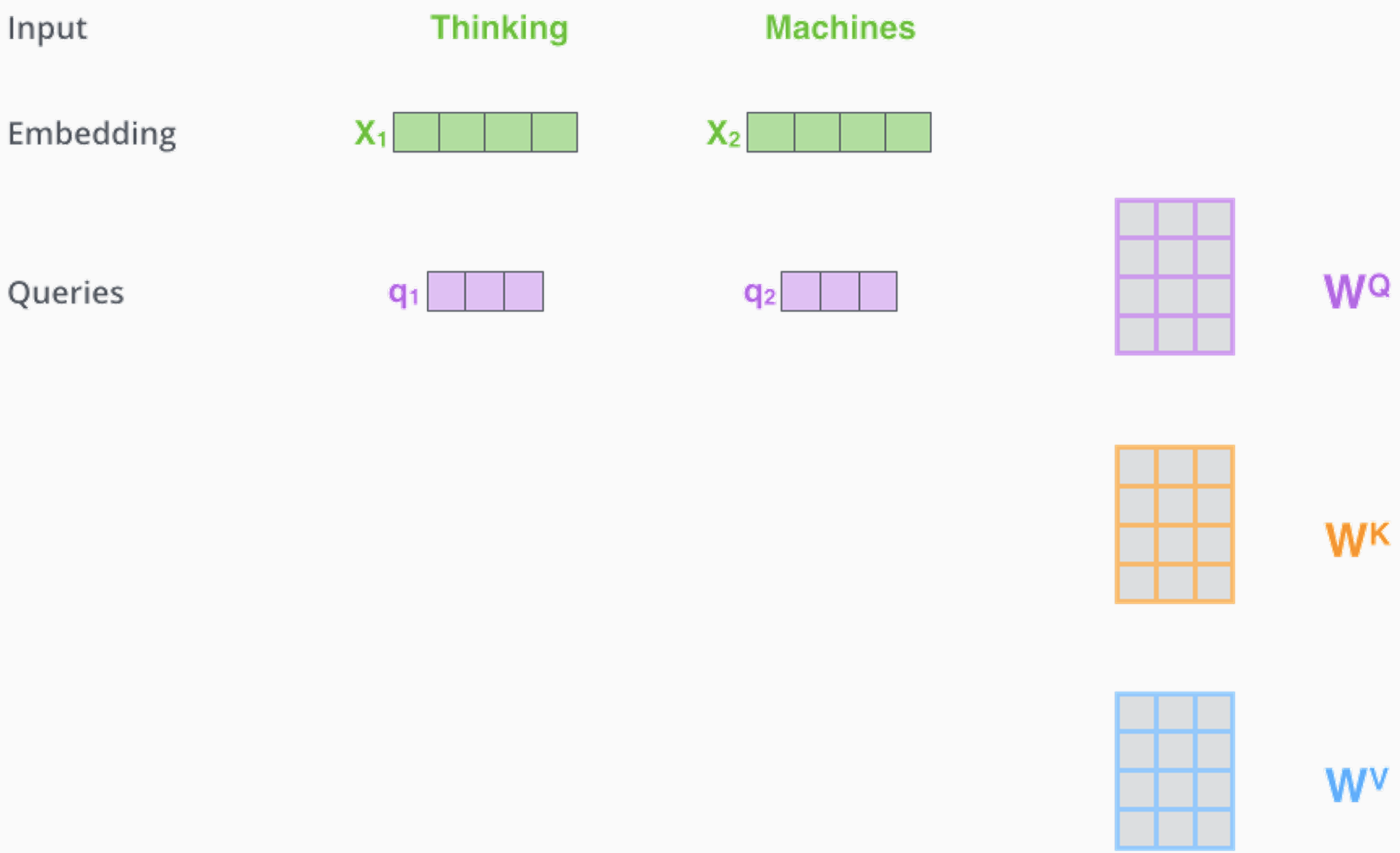
Self attention: An example



Three parameter matrices associated with this self attention block

Images from https://jalammar.github.io/illustrated-transformer/

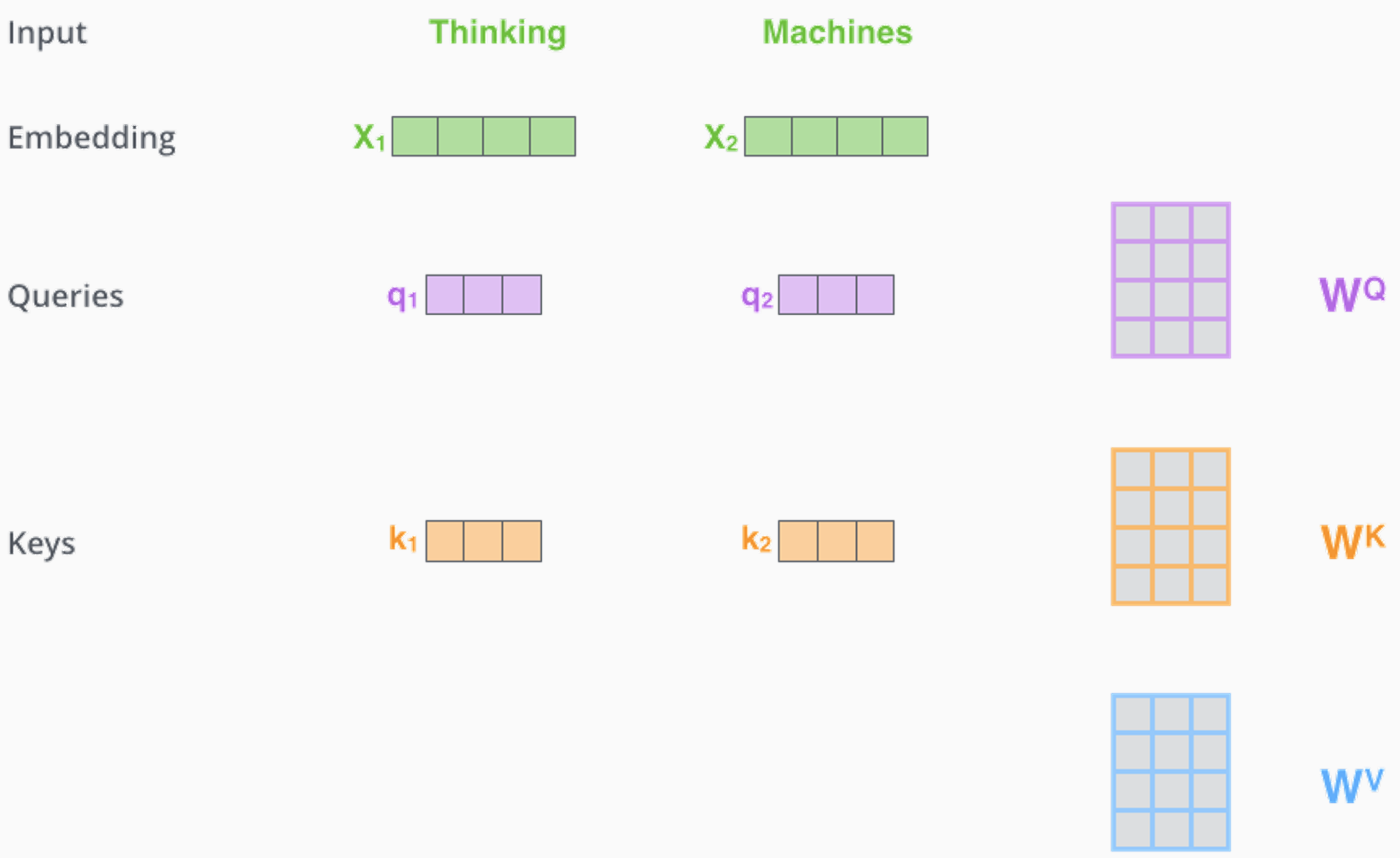
Self attention: An example



Three parameter matrices associated with this self attention block

Images from https://jalammar.github.io/illustrated-transformer/

Self attention: An example

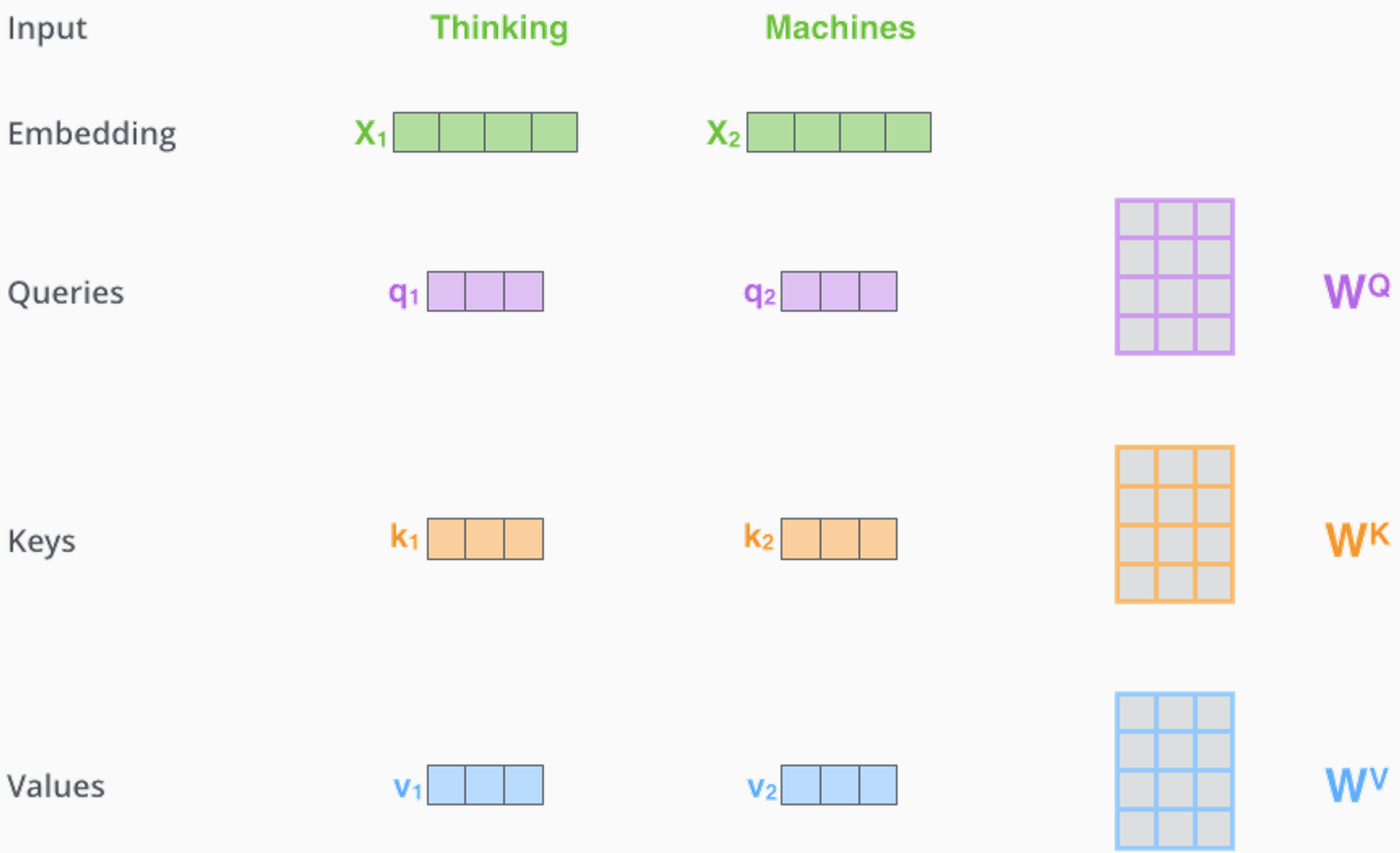


Three parameter matrices associated with this self attention block

Images from https://jalammar.github.io/illustrated-transformer/

Self attention: An example

For every element of the sequence, create three vectors that are called its *query*, *key* and *value* vectors.



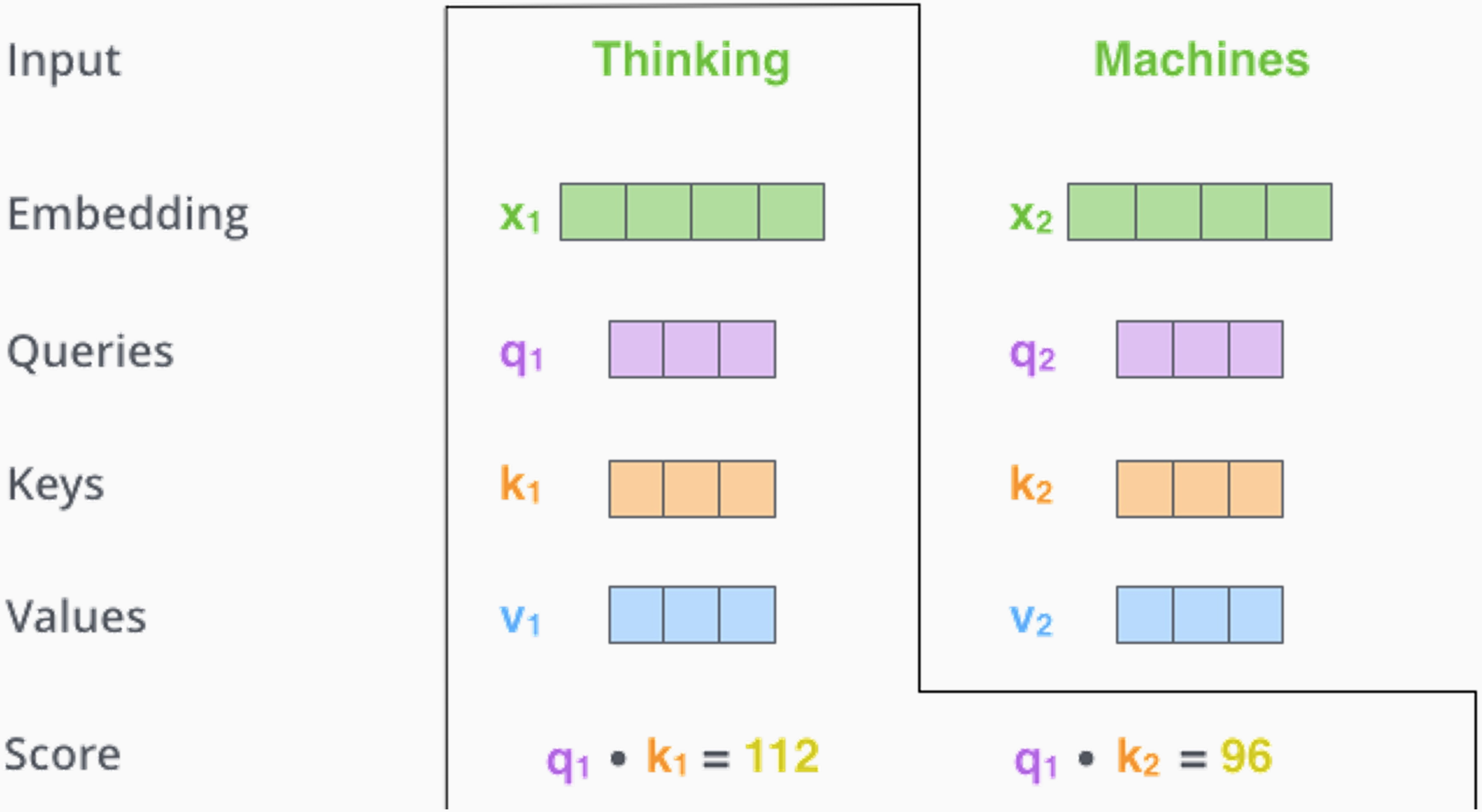
Three parameter matrices associated with this self attention block

Images from https://jalammar.github.io/illustrated-transformer/

Self attention: An example

For each word, compute the self attention.

First compute the dot product of its query vector with the key vector of all words in the sentence

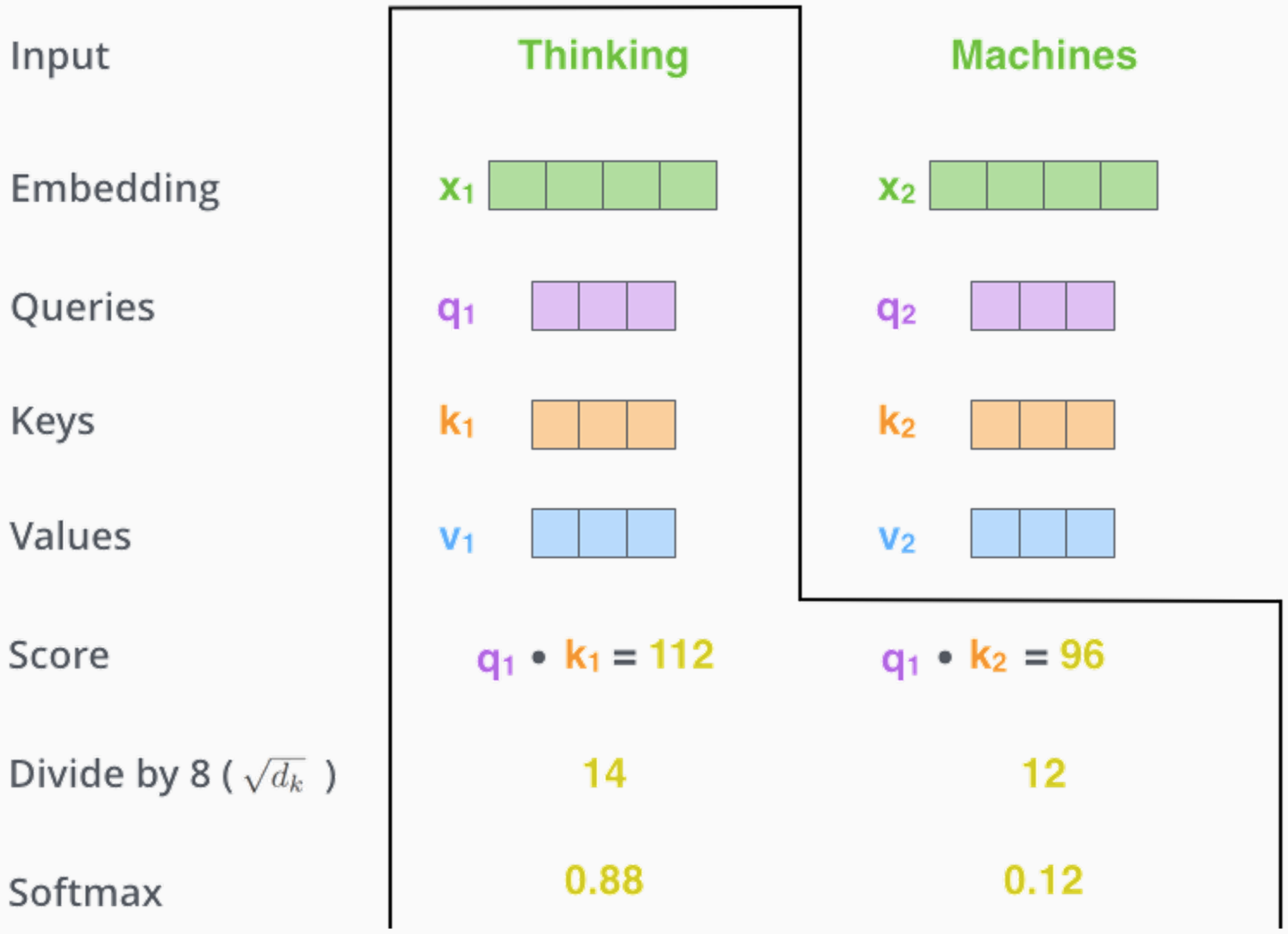


Images from https://jalammar.github.io/illustrated-transformer/

Self attention: An example

For each word, compute the self attention.

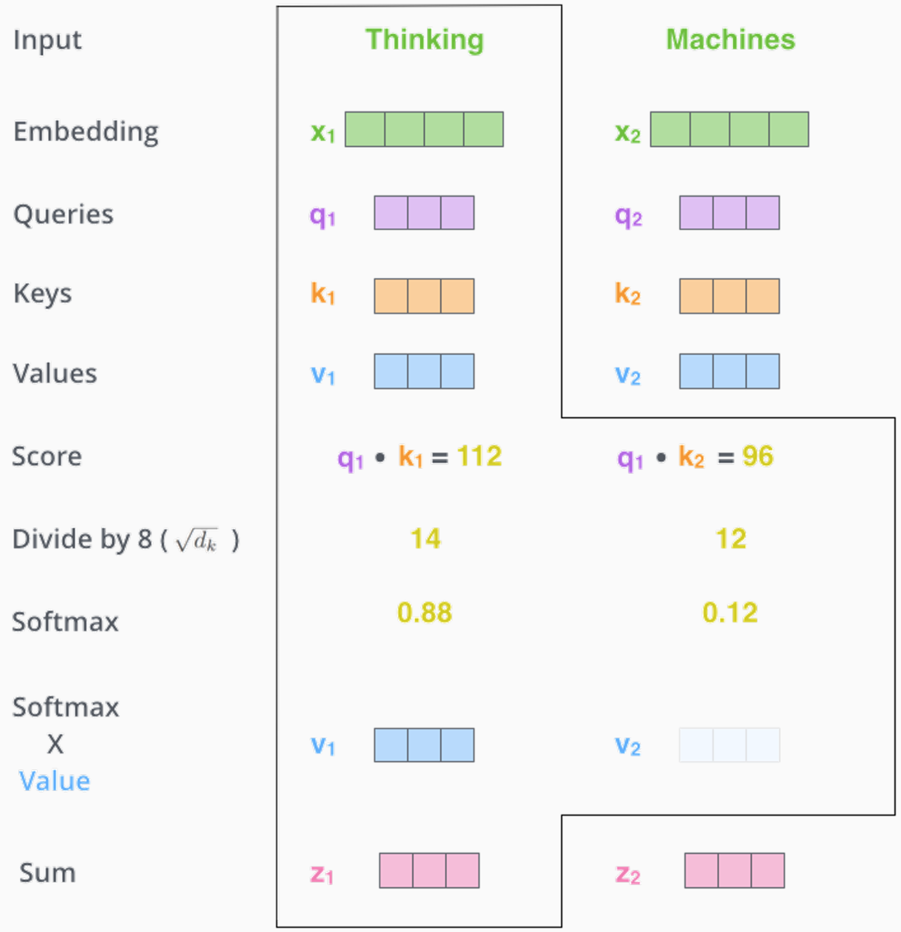
Then, normalize



Images from https://jalammar.github.io/illustrated-transformer/

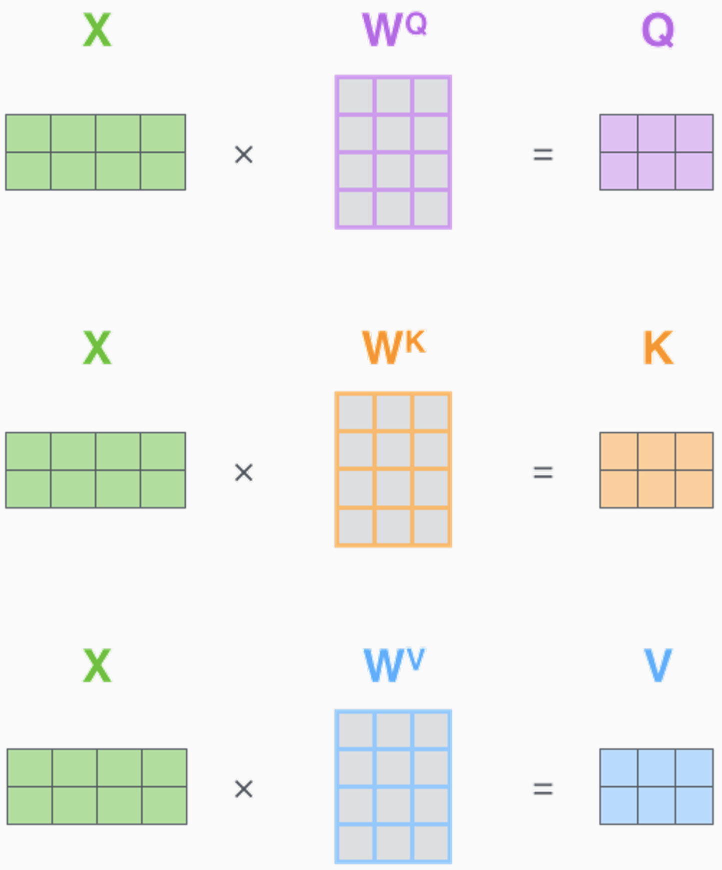
Self attention: An example

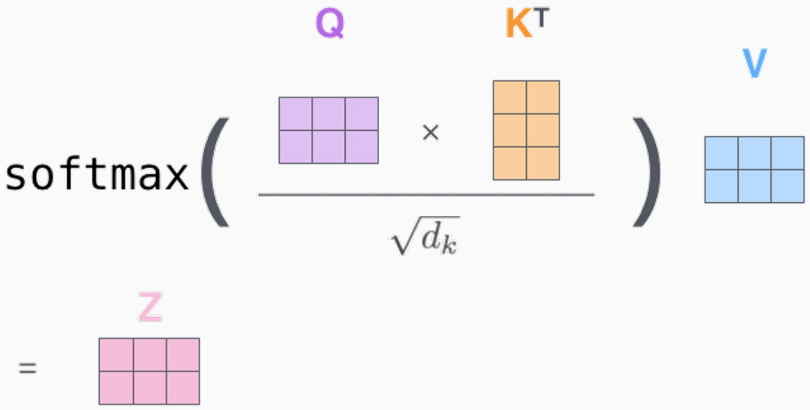
Use the attention probabilities to weigh the value vectors to produce the output vector for that word



Images from https://jalammar.github.io/illustrated-transformer/

Self attention: Illustrated





Images from https://jalammar.github.io/illustrated-transformer/

Self attention

Given:

- A $T \times d$ matrix X
- Three $d \times \frac{d}{h}$ parameter matrices called $W^{(q)}, W^{(k)}, W^{(v)}$

Self attention

Given:

- A $T \times d$ matrix X
- Three $d \times \frac{d}{h}$ parameter matrices called $W^{(q)}, W^{(k)}, W^{(v)}$

1. Compute the $T \times T$ matrix $\mathrm{A} = \frac{XW^{(q)}\left(XW^{(k)}\right)^T}{\sqrt{d}}$

Self attention

Given:

- A $T \times d$ matrix X
- Three $d \times \frac{d}{h}$ parameter matrices called $W^{(q)}, W^{(k)}, W^{(v)}$

Called the "query" in the original paper

1. Compute the $T \times T$ matrix $\mathrm{A} = \dfrac{\boxed{XW^{(q)}}\left(XW^{(k)}\right)^{T}}{\sqrt{d}}$

Self attention

Given:

- A $T \times d$ matrix X
- Three $d \times \frac{d}{h}$ parameter matrices called $W^{(q)}, W^{(k)}, W^{(v)}$

1. Compute the $T \times T$ matrix $\mathrm{A} = \frac{\boxed{XW^{(q)}}\left(\boxed{XW^{(k)}}\right)^T}{\sqrt{d}}$

Called the "query" in the original paper

Called the "key" in the original paper

Self attention

Given:

- A $T \times d$ matrix X
- Three $d \times \frac{d}{h}$ parameter matrices called $W^{(q)}, W^{(k)}, W^{(v)}$

Called the "query" in the original paper

Called the "key" in the original paper

1. Compute the $T \times T$ matrix $\mathrm{A} = \frac{\boxed{XW^{(q)}}\left(\boxed{XW^{(k)}}\right)^T}{\sqrt{d}}$

2. Return $\mathbf{softmax}(A)XW^{(v)}$

Self attention

Given:

- A $T \times d$ matrix X
- Three $d \times \frac{d}{h}$ parameter matrices called $W^{(q)}, W^{(k)}, W^{(v)}$

Called the "query" in the original paper

Called the "key" in the original paper

Called the "value" in the original paper

1. Compute the $T \times T$ matrix $\mathrm{A} = \frac{\boxed{XW^{(q)}}\left(\boxed{XW^{(k)}}\right)^T}{\sqrt{d}}$
2. Return $\mathbf{softmax}(A)\boxed{XW^{(v)}}$

Self attention

Given:

- A $T \times d$ matrix X
- Three $d \times \frac{d}{h}$ parameter matrices called $W^{(q)}, W^{(k)}, W^{(v)}$

Called the "query" in the original paper

Called the "key" in the original paper

1. Compute the $T \times T$ matrix $A = \frac{\boxed{XW^{(q)}}\left(\boxed{XW^{(k)}}\right)^T}{\sqrt{d}}$

2. Return $\boxed{\mathrm{softmax}(A)}\boxed{XW^{(v)}}$

Called the "value" in the original paper

Normalized row wise

$$\mathrm{softmax}(A)_{ij} = \frac{\exp(A_{ij})}{\sum_k \exp(A_{ik})}$$

Self attention

Given:

- A $T \times d$ matrix X
- Three $d \times \frac{d}{h}$ parameter matrices called $W^{(q)}, W^{(k)}, W^{(v)}$

Called the "query" in the original paper

1. Compute the $T \times T$ matrix $\mathrm{A} = \dfrac{\boxed{XW^{(q)}}\left(\boxed{XW^{(k)}}\right)^T}{\sqrt{d}}$

Called the "key" in the original paper

2. Return $\mathbf{softmax}(A)\boxed{XW^{(v)}}$

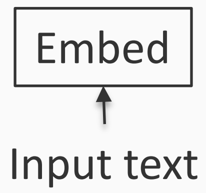
Called the "value" in the original paper

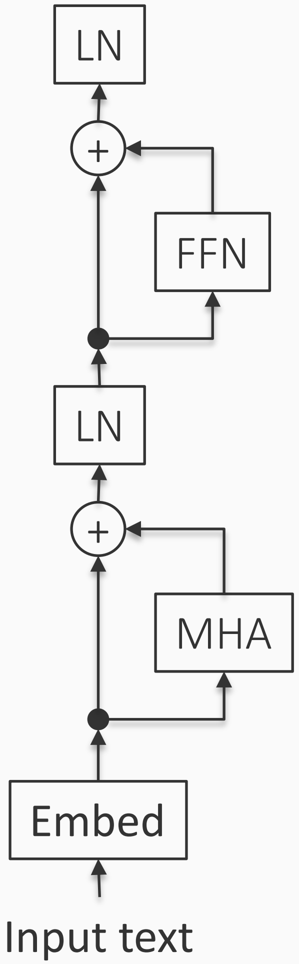
$T \times T$ $\quad T \times d \quad d \times \frac{d}{h}$

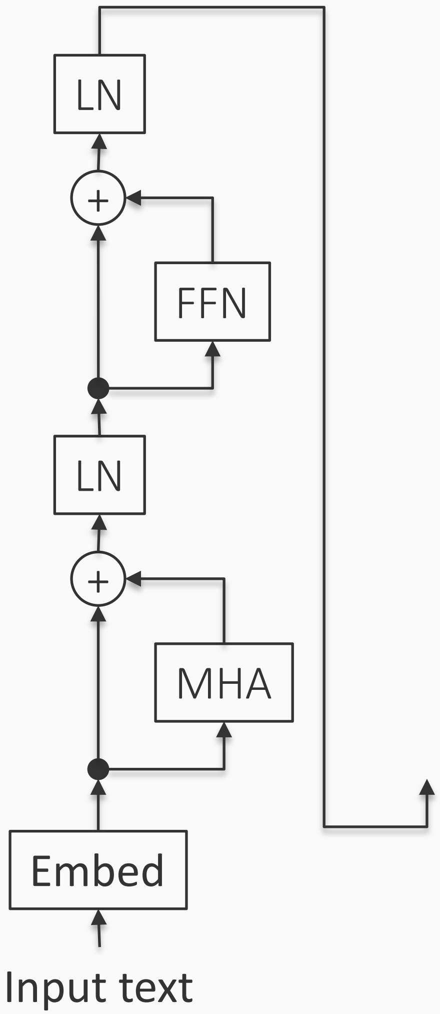
The final result is a $T \times \frac{d}{h}$ matrix, i.e., one $\frac{d}{h}$ dimensional vector per token

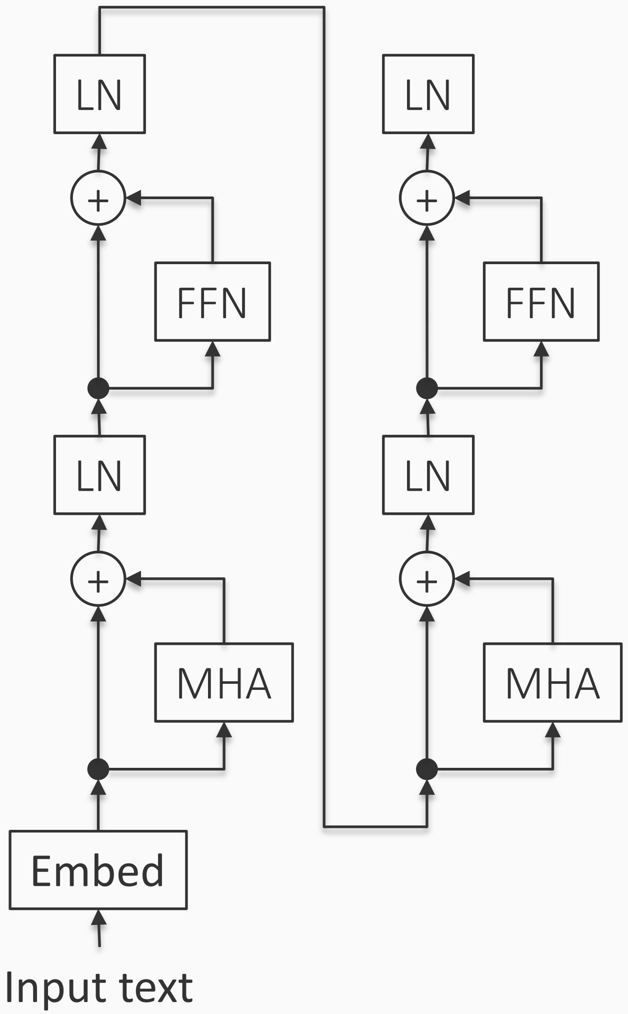
Recall that the transformer neural network is just layer after layer of the transformer block

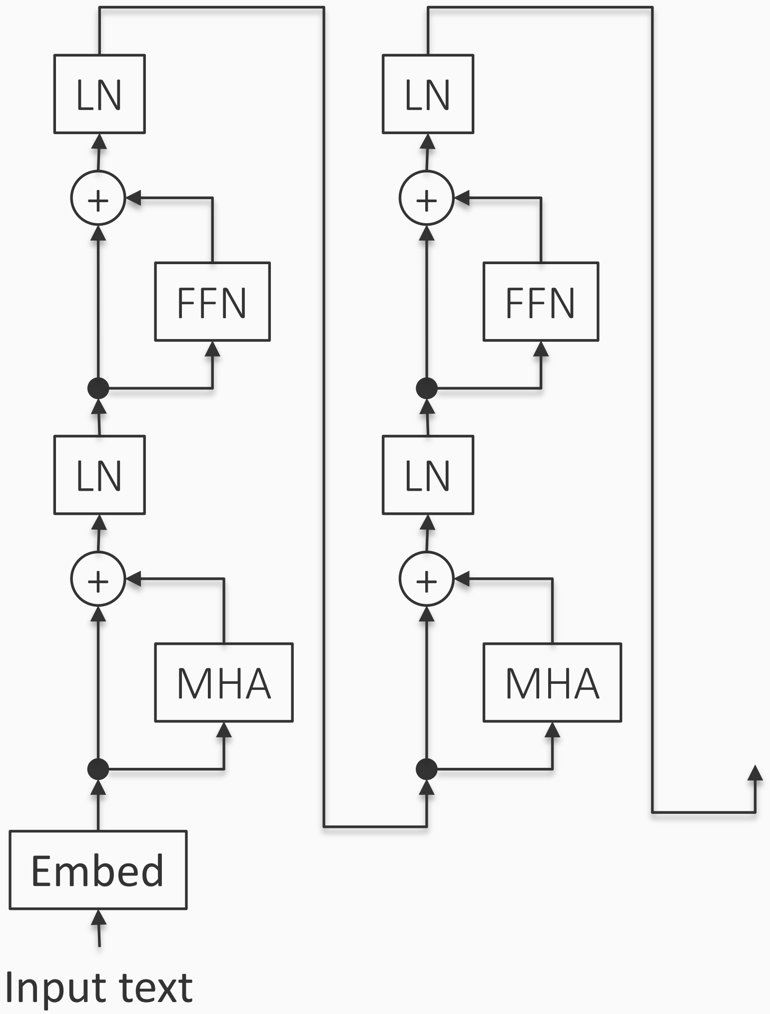
Recall that the transformer neural network is just layer after layer of the transformer block



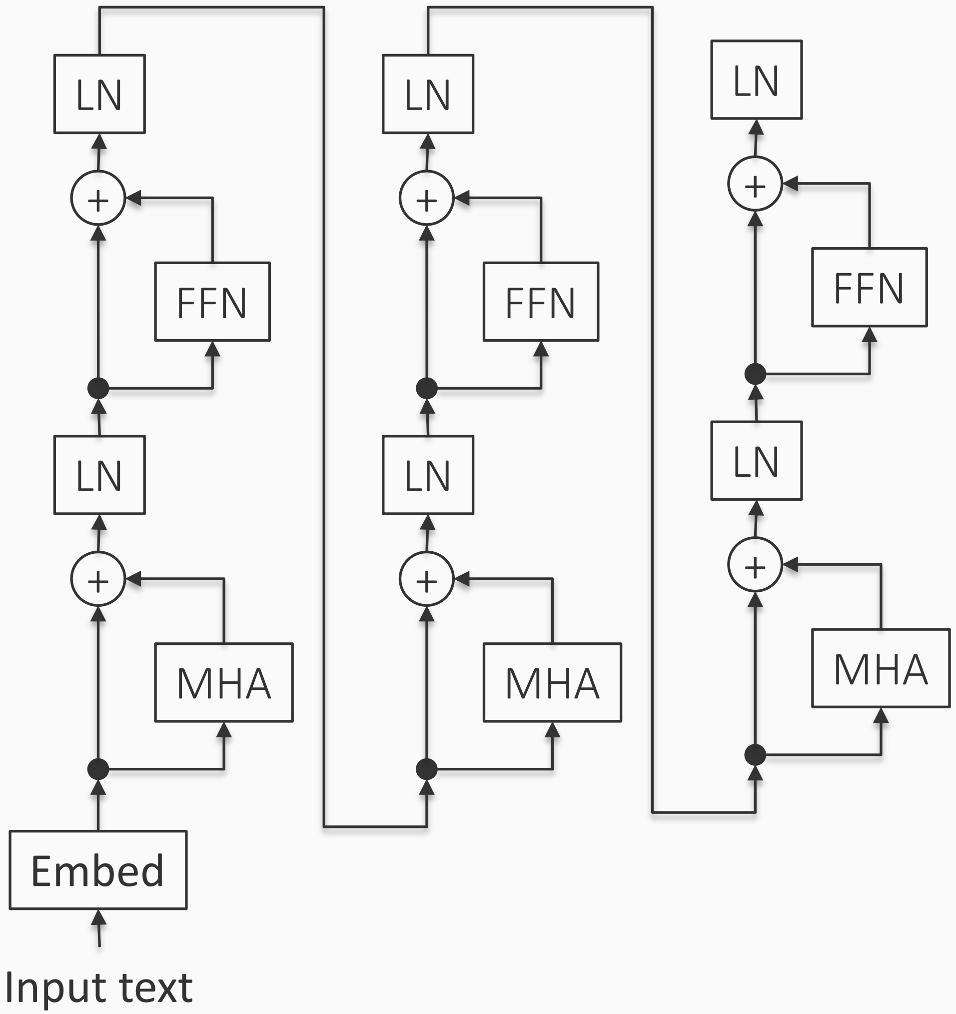

LN
+
FFN
LN
+
MHA
Embed
Input text

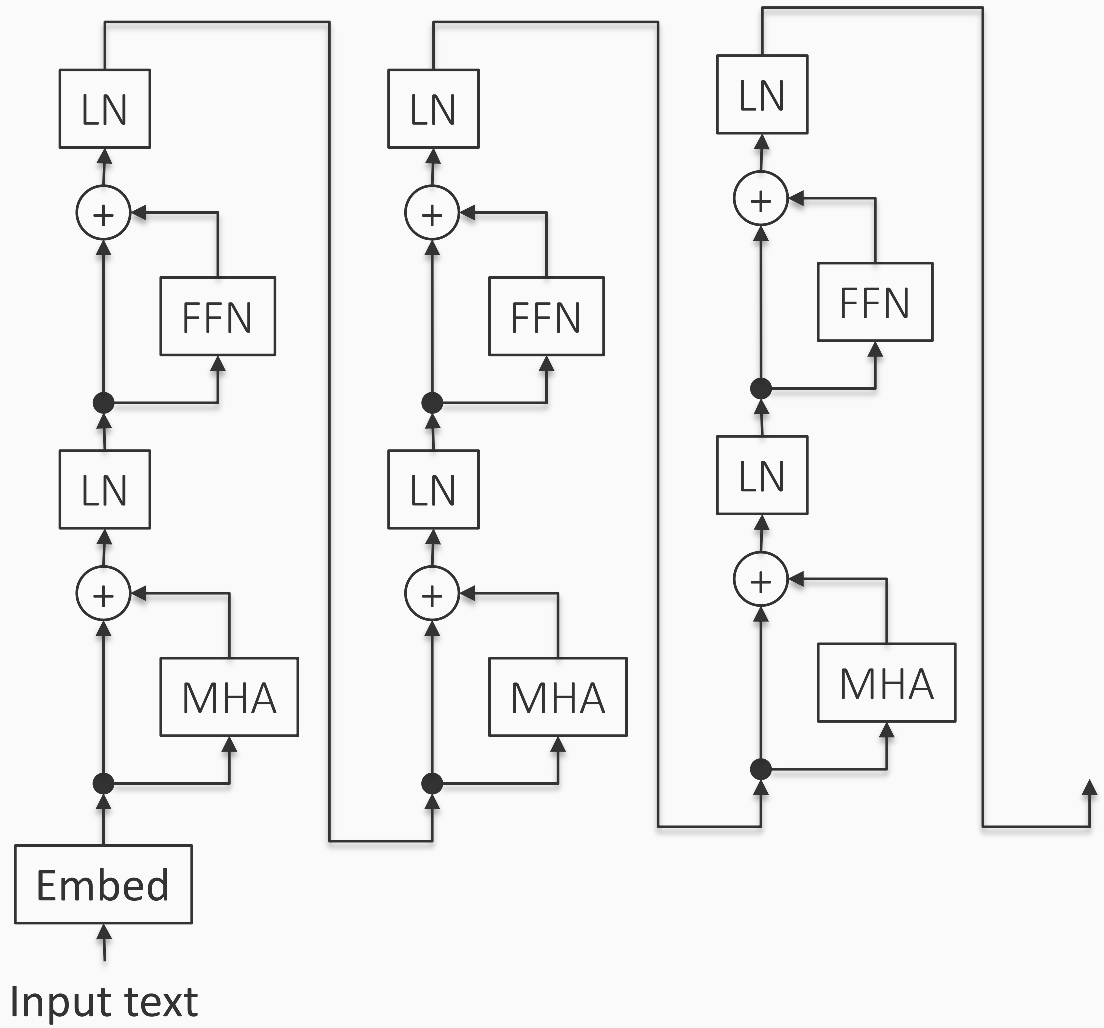

LN
+
FFN
LN
+
MHA
Embed
Input text

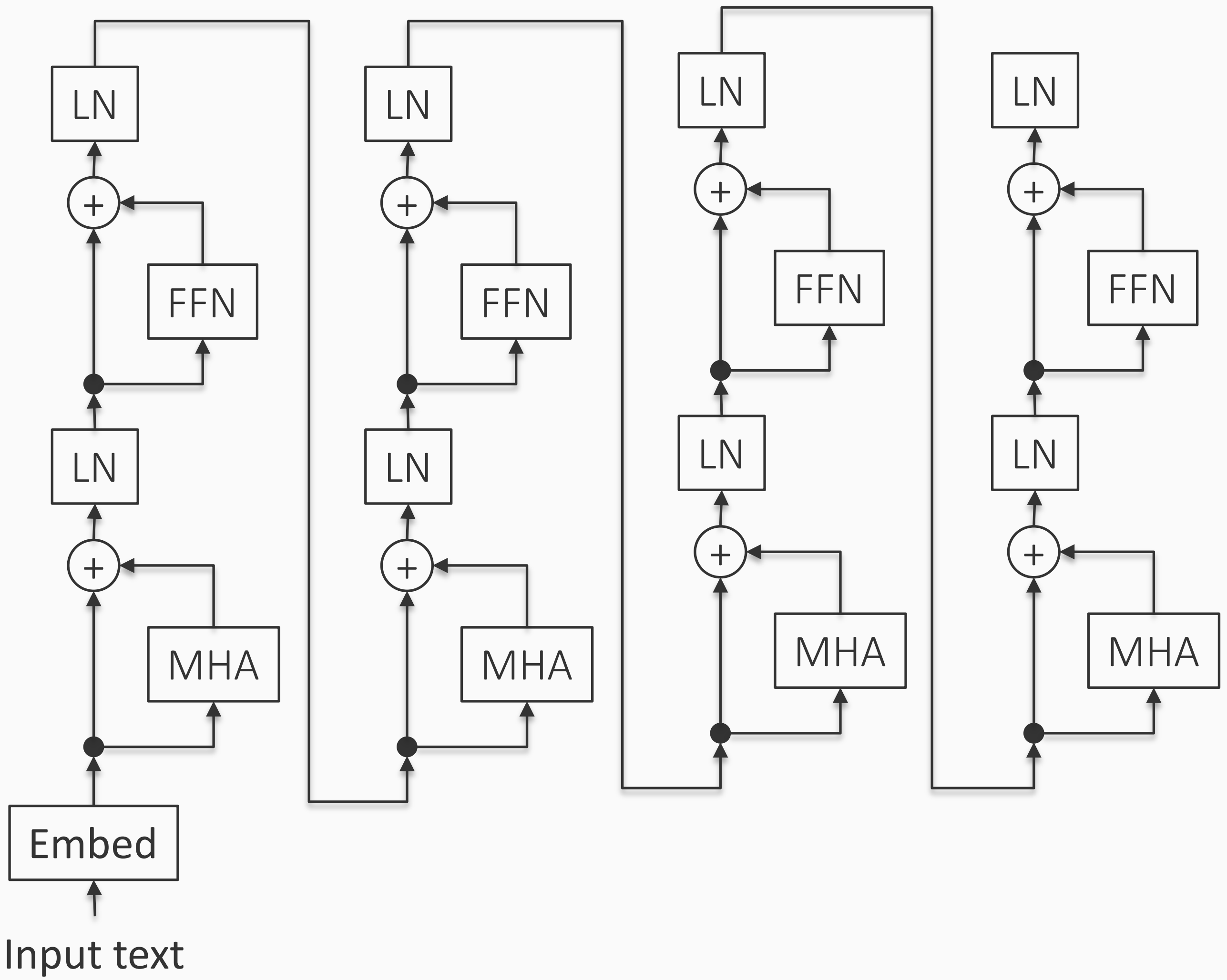

LN
LN
+
+
FFN
FFN
LN
LN
+
+
MHA
MHA
Embed
Input text

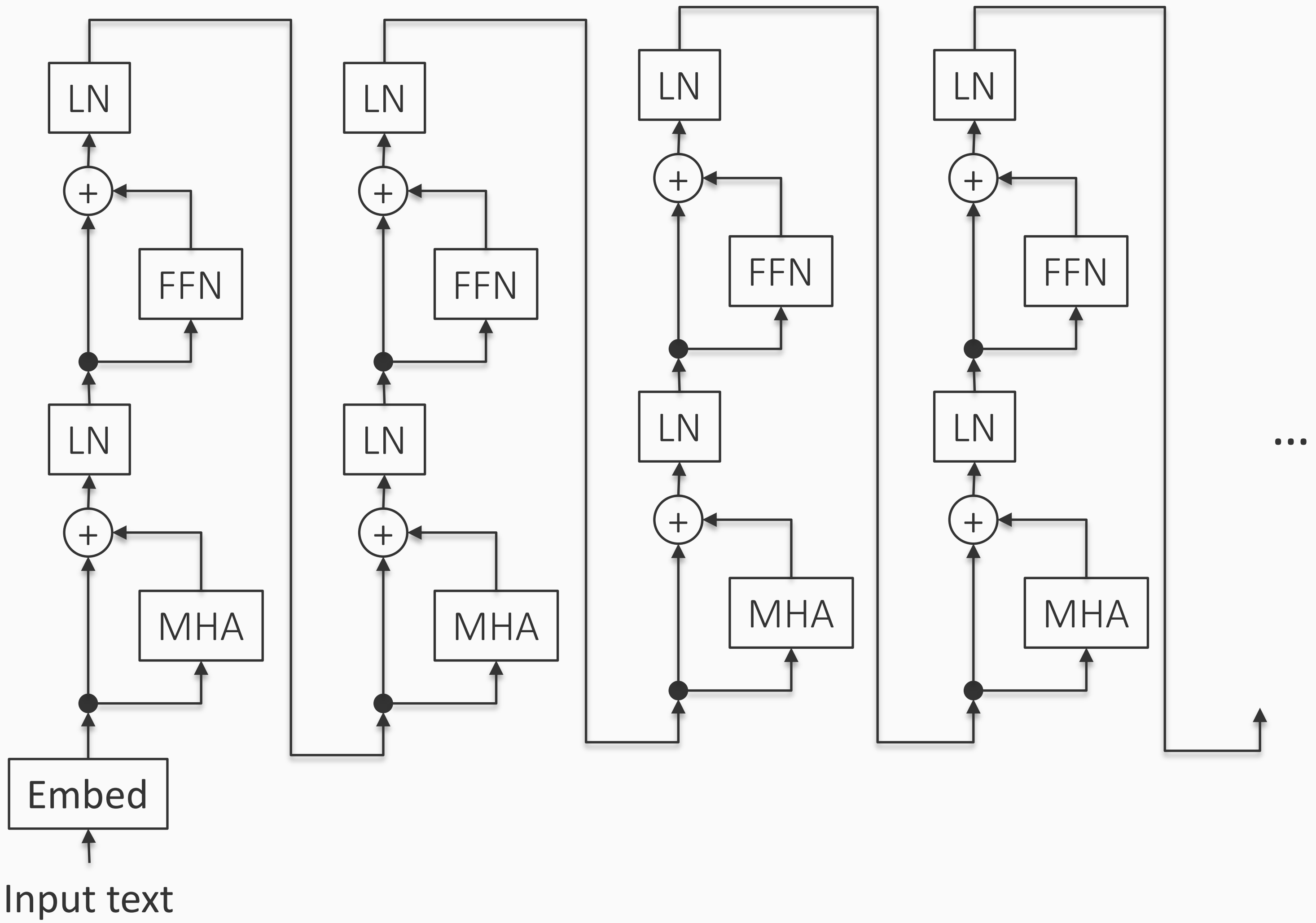


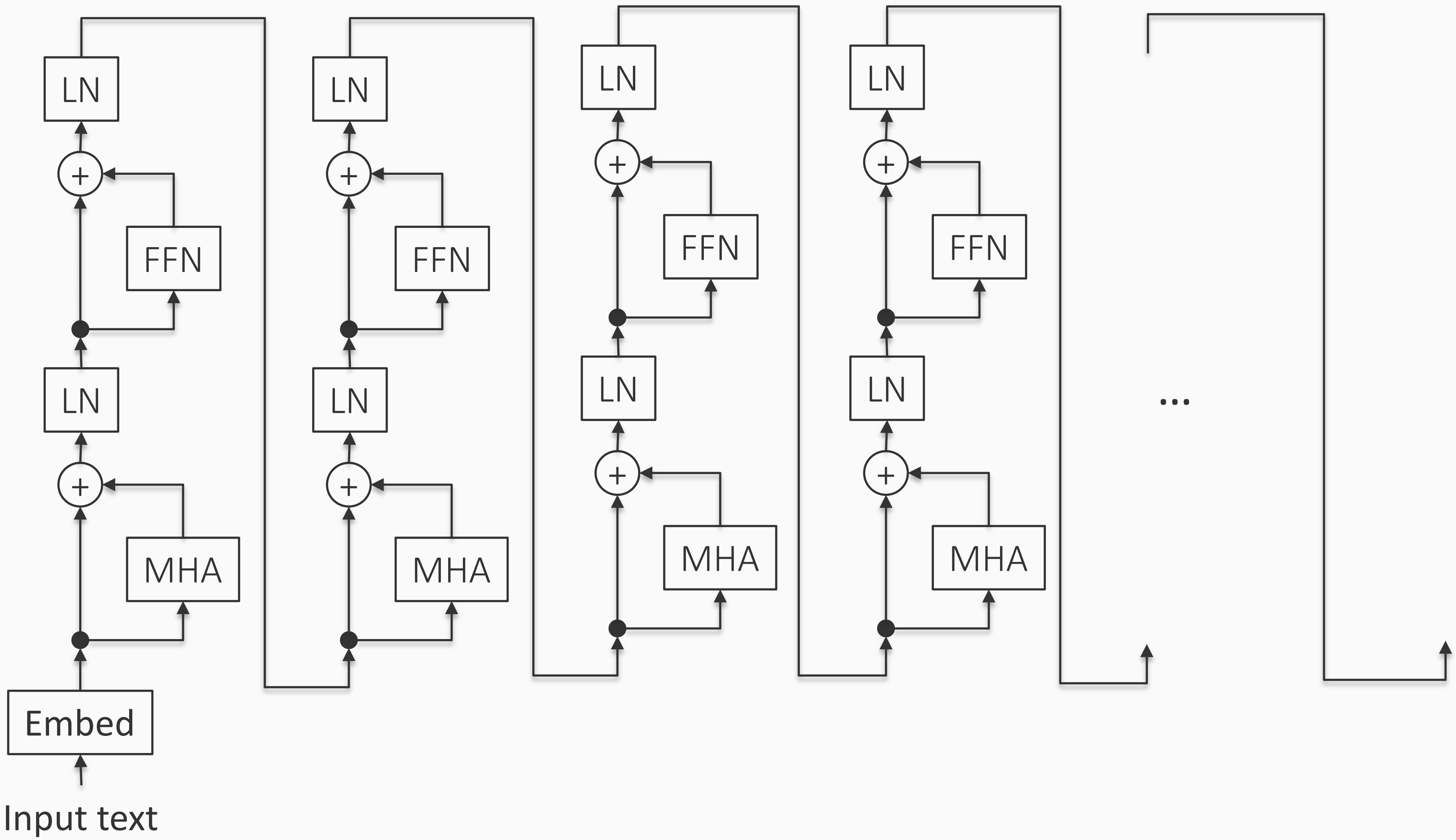
LN
+
FFN
LN
+
MHA
LN
+
FFN
LN
+
MHA
Embed
Input text

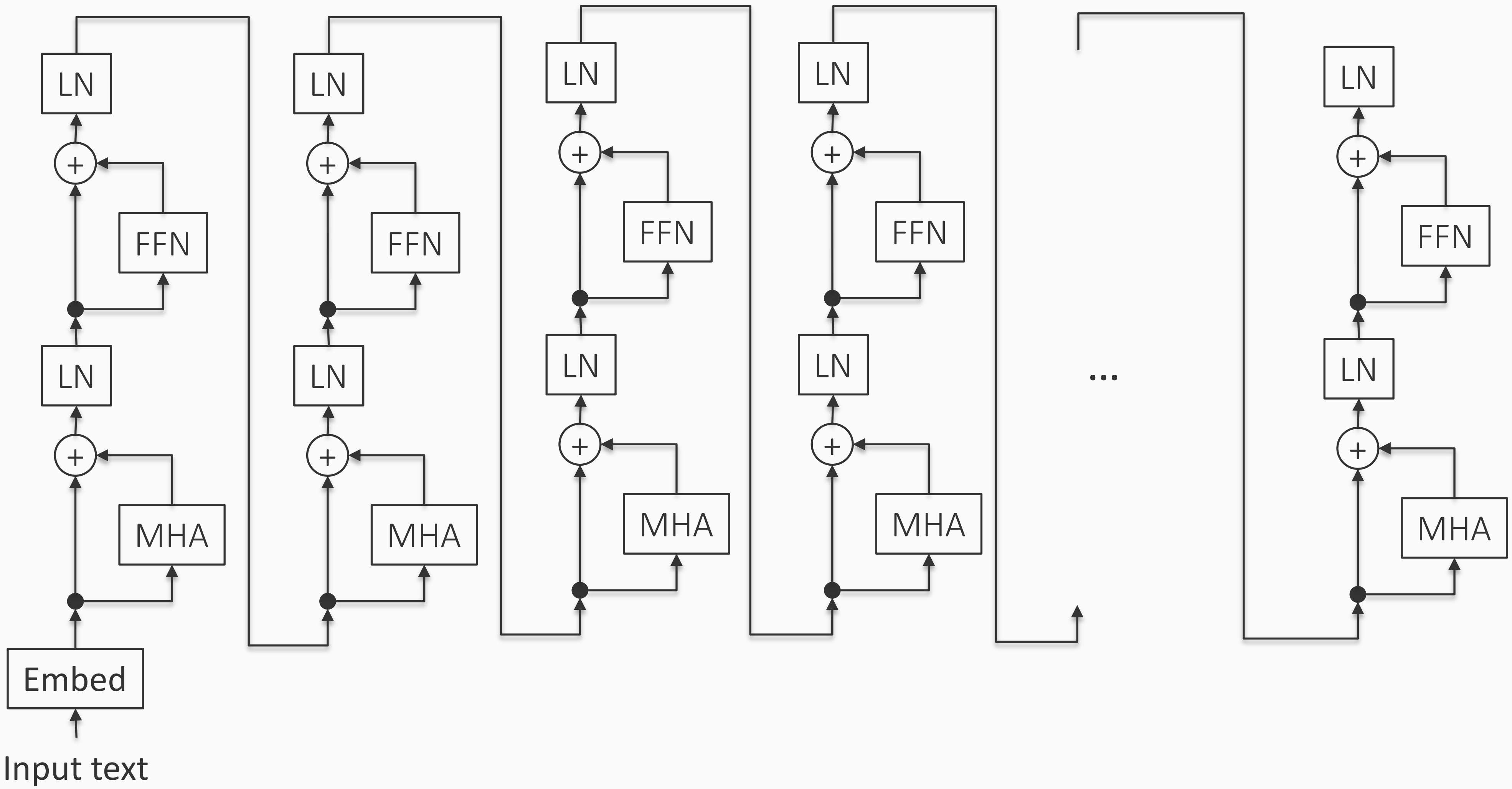

LN
FFN
LN
MHA
LN
FFN
LN
MHA
LN
FFN
LN
MHA
Embed
Input text

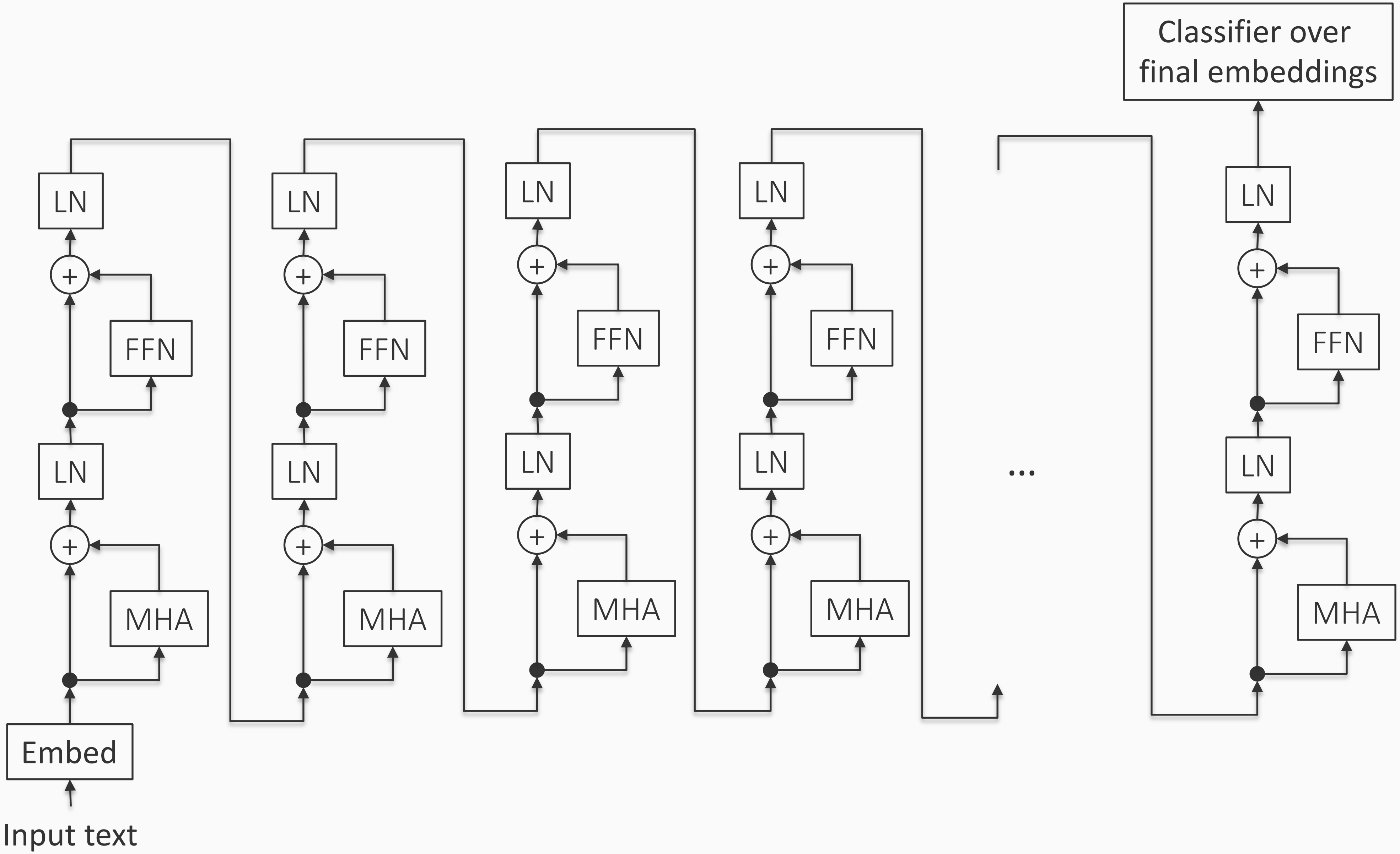

LN
FFN
LN
MHA
LN
FFN
LN
MHA
LN
FFN
LN
MHA
Embed
Input text

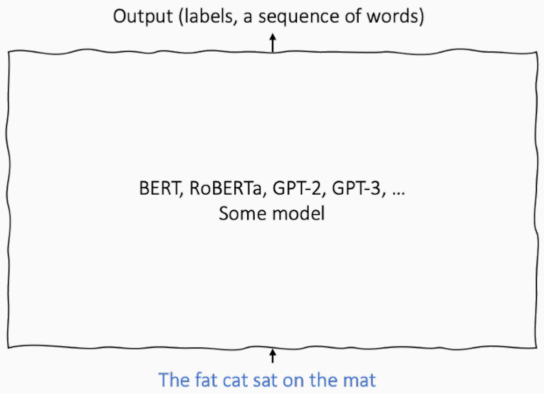

LN
+
FFN
LN
+
MHA
Embed
Input text
LN
+
FFN
LN
+
MHA
LN
+
FFN
LN
+
MHA
LN
+
FFN
LN
+
MHA

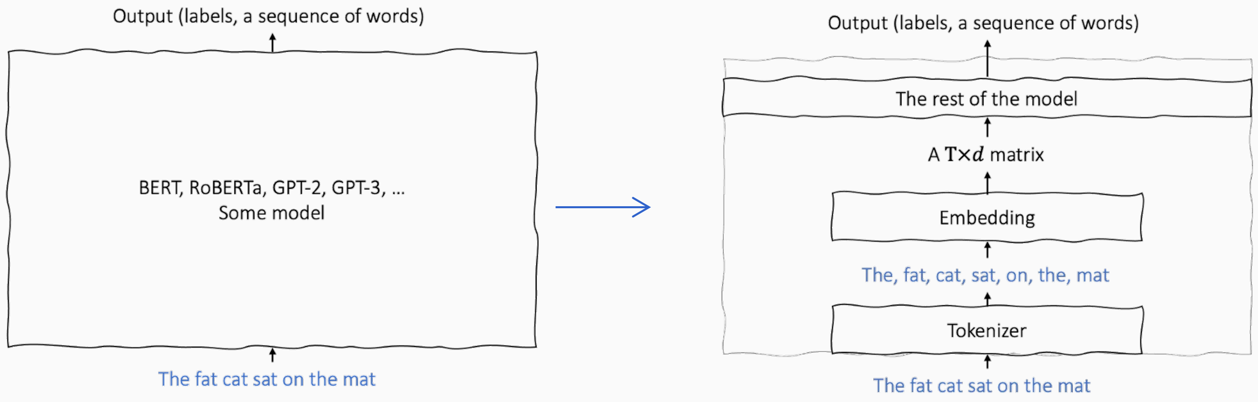

LN
FFN
LN
MHA
LN
FFN
LN
MHA
LN
FFN
LN
MHA
LN
FFN
LN
MHA
...
Embed
Input text

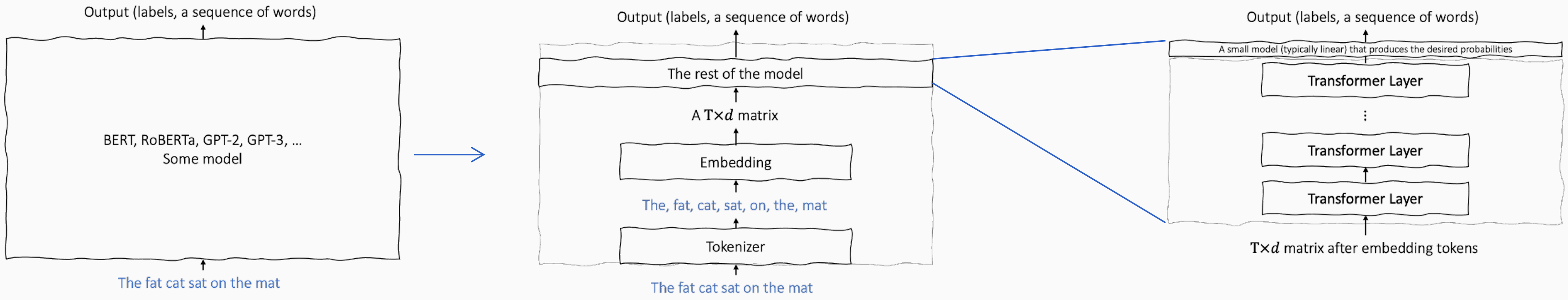

LN
FFN
LN
MHA
Embed
Input text
LN
FFN
LN
MHA
LN
FFN
LN
MHA
LN
FFN
LN
MHA
...

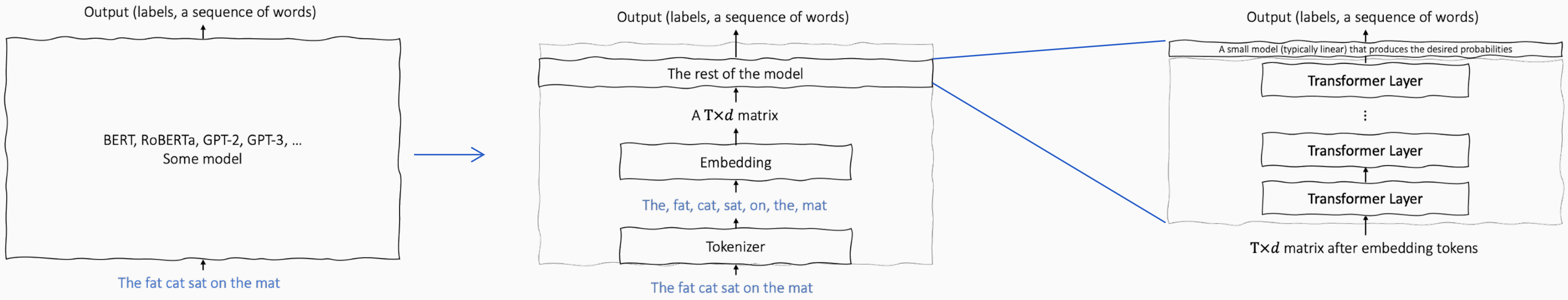

LN
FFN
LN
MHA
LN
FFN
LN
MHA
LN
FFN
LN
MHA
LN
FFN
LN
MHA
...
LN
FFN
LN
MHA
Embed
Input text


Classifier over
final embeddings
LN
FFN
LN
MHA
Embed
Input text
...

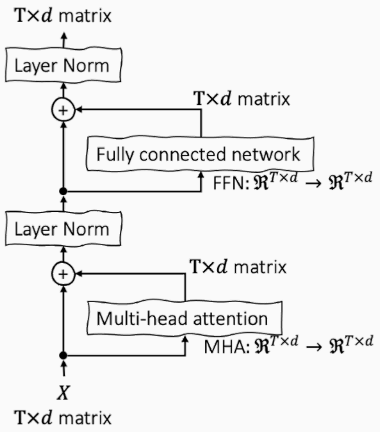

Output (labels, a sequence of words)
BERT, RoBERTa, GPT-2, GPT-3, …
Some model
The fat cat sat on the mat


Output (labels, a sequence of words)
BERT, RoBERTa, GPT-2, GPT-3, …
Some model
The fat cat sat on the mat
Output (labels, a sequence of words)
The rest of the model
A $\mathrm{T}\times d$ matrix
Embedding
The, fat, cat, sat, on, the, mat
Tokenizer
The fat cat sat on the mat

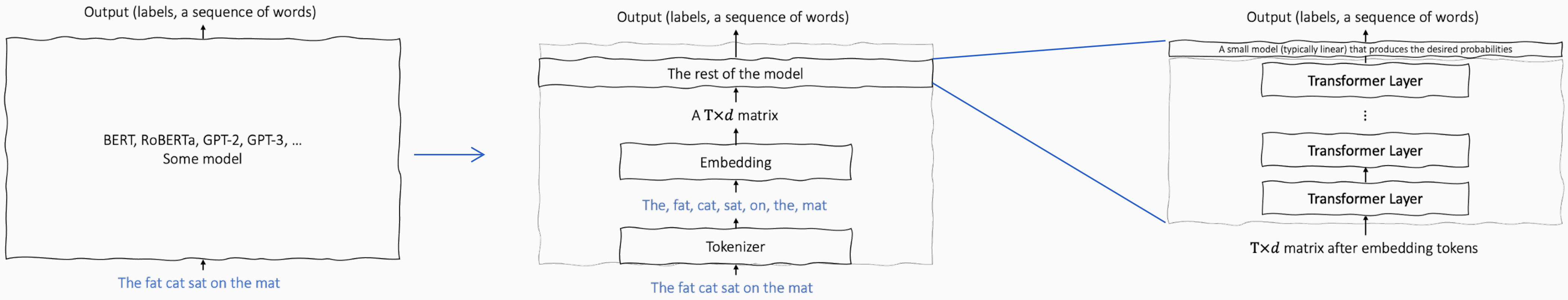

Output (labels, a sequence of words)
BERT, RoBERTa, GPT-2, GPT-3, …
Some model
The fat cat sat on the mat
Output (labels, a sequence of words)
The rest of the model
A T×d matrix
Embedding
The, fat, cat, sat, on, the, mat
Tokenizer
The fat cat sat on the mat
Output (labels, a sequence of words)
A small model (typically linear) that produces the desired probabilities
Transformer Layer
⋮
Transformer Layer
Transformer Layer
T×d matrix after embedding tokens

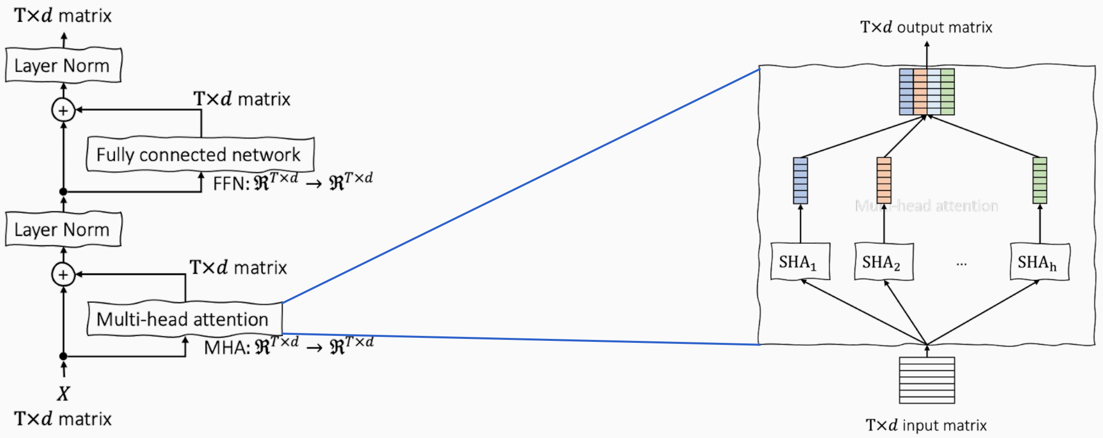


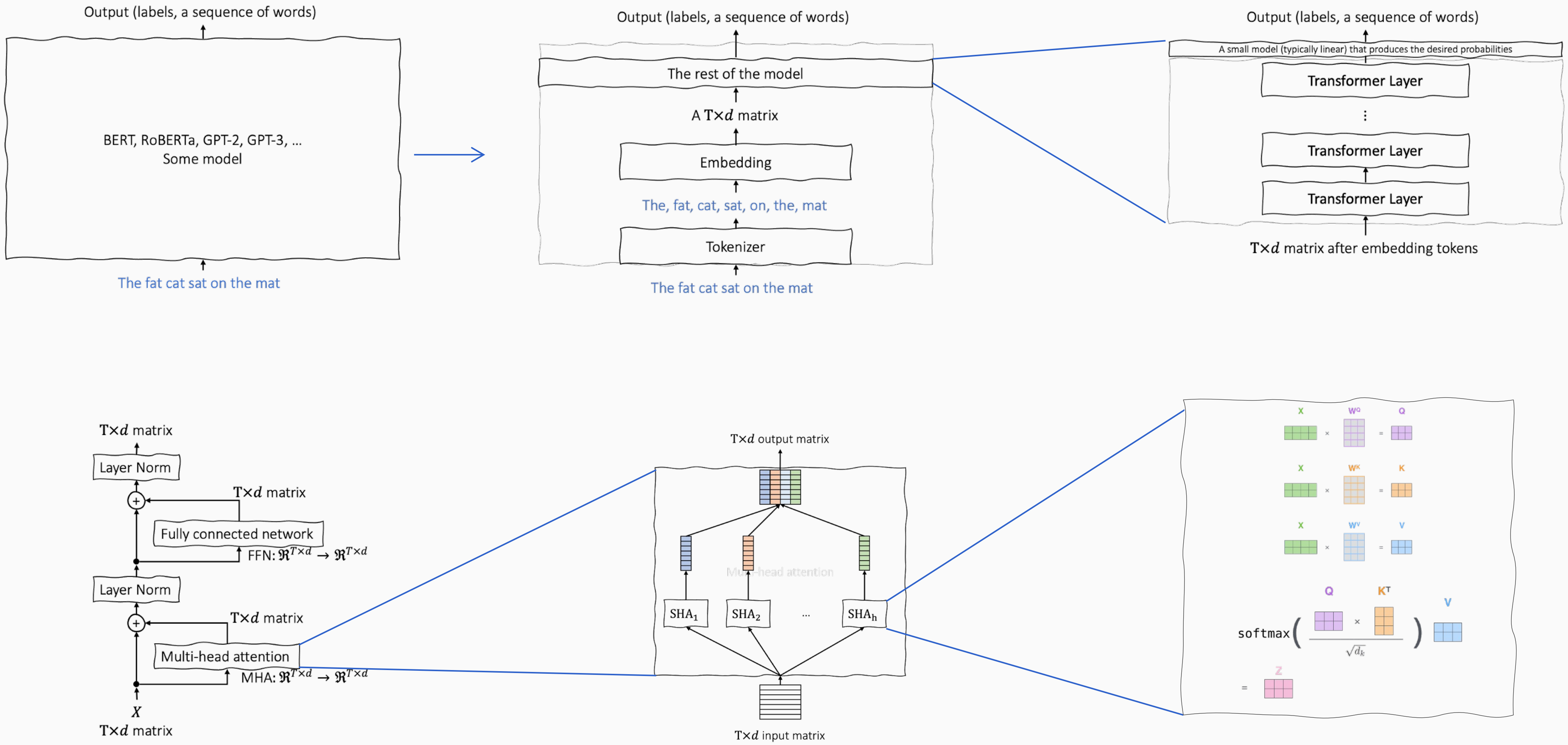
Output (labels, a sequence of words)
BERT, RoBERTa, GPT-2, GPT-3, …
Some model
The fat cat sat on the mat
Output (labels, a sequence of words)
The rest of the model
A T×d matrix
Embedding
The, fat, cat, sat, on, the, mat
Tokenizer
The fat cat sat on the mat
Output (labels, a sequence of words)
A small model (typically linear) that produces the desired probabilities
Transformer Layer
⋮
Transformer Layer
Transformer Layer
T×d matrix after embedding tokens



T×d matrix
Layer Norm
T×d matrix
Fully connected network
FFN: $\Re^{T\times d} \rightarrow \Re^{T\times d}$
Layer Norm
T×d matrix
Multi-head attention
MHA: $\Re^{T\times d} \rightarrow \Re^{T\times d}$
X
T×d matrix


Output (labels, a sequence of words)
BERT, RoBERTa, GPT-2, GPT-3, ...
Some model
The fat cat sat on the mat
Output (labels, a sequence of words)
The rest of the model
A $T \times d$ matrix
Embedding
The, fat, cat, sat, on, the, mat
Tokenizer
The fat cat sat on the mat
Output (labels, a sequence of words)
A small model (typically linear) that produces the desired probabilities
Transformer Layer
⋮
Transformer Layer
Transformer Layer
$T \times d$ matrix after embedding tokens


$T \times d$ matrix
Layer Norm
$T \times d$ matrix
Fully connected network
FFN: $\Re^{T \times d} \rightarrow \Re^{T \times d}$
Layer Norm
$T \times d$ matrix
Multi-head attention
MHA: $\Re^{T \times d} \rightarrow \Re^{T \times d}$
X
$T \times d$ matrix
$T \times d$ output matrix
SHA_1
SHA_2
...
SHA_h
$T \times d$ input matrix



Output (labels, a sequence of words)
BERT, RoBERTa, GPT-2, GPT-3, ...
Some model
The fat cat sat on the mat
Output (labels, a sequence of words)
The rest of the model
A T×d matrix
Embedding
The, fat, cat, sat, on, the, mat
Tokenizer
The fat cat sat on the mat
Output (labels, a sequence of words)
A small model (typically linear) that produces the desired probabilities
Transformer Layer
⋮
Transformer Layer
Transformer Layer
T×d matrix after embedding tokens
T×d matrix
Layer Norm
T×d matrix
Fully connected network
FFN: $\mathfrak{R}^{T\times d} \rightarrow \mathfrak{R}^{T\times d}$
Layer Norm
T×d matrix
Multi-head attention
MHA: $\mathfrak{R}^{T\times d} \rightarrow \mathfrak{R}^{T\times d}$
X
T×d matrix
T×d output matrix
Multi-head attention
SHA_1
SHA_2
...
SHA_h
T×d input matrix
X
W^Q
Q
X
W^K
K
X
W^V
V
Q
K^T
V
softmax
$\sqrt{d_k}$
Z

"The Transformer paper": NeurIPS 2017

Focus: machine translation

"We propose a new simple !! network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely."

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-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task

Outline

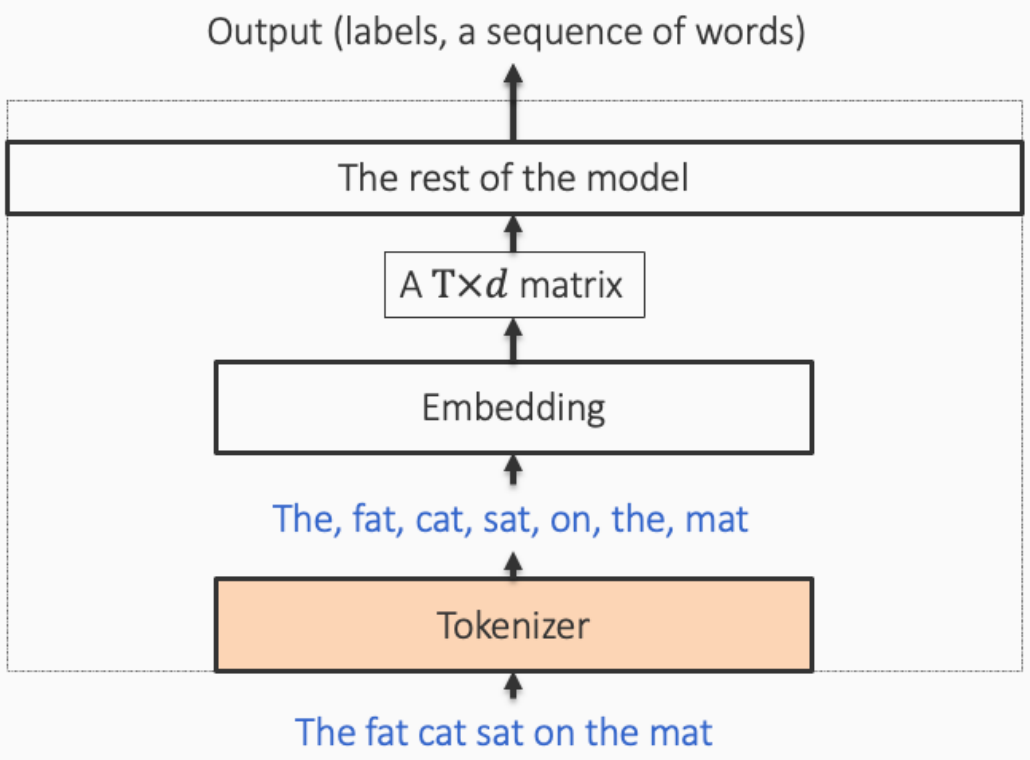
- The challenge of modeling sequences
- The transformer architecture
  - The big picture
- Details and fine print
- The impact of transformers

Details of the model

1. Tokenization
2. Position embeddings
3. Encoders, decoders and encoder-decoders

Tokenization and the vocabulary

How we break a sequence into tokens decides the vocabulary of the model, and the size of the embedding matrix

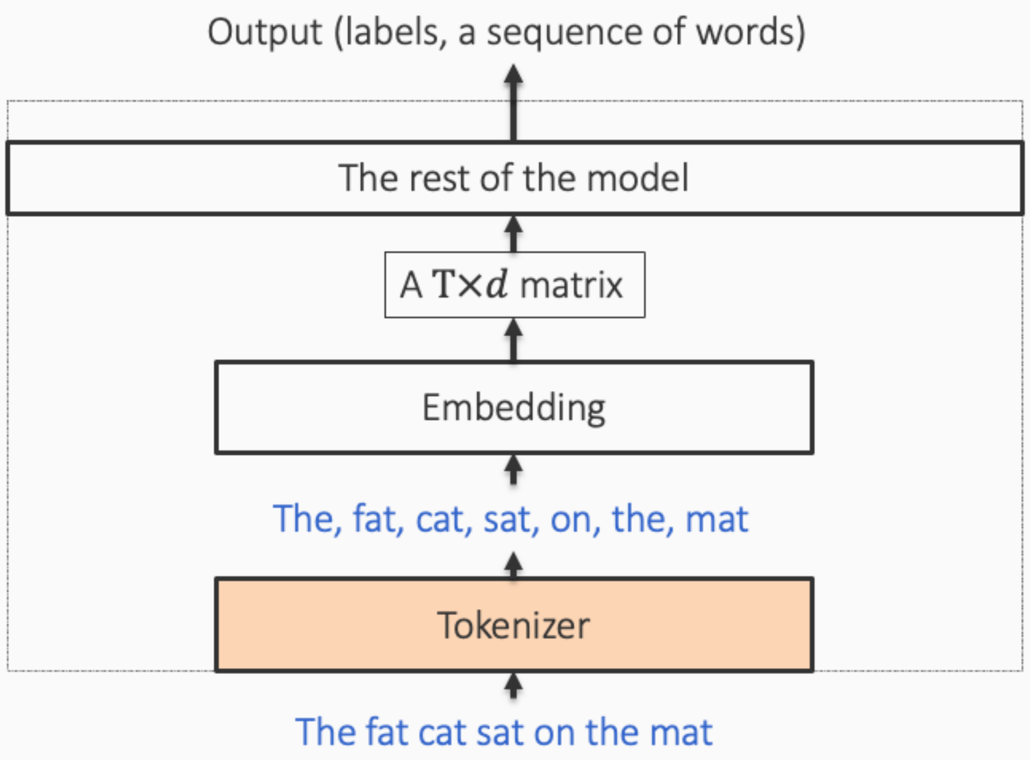


Several options possible:

- Each character is a token
- Each word is a token
  - How do we define what a "word" is?
  - What about languages that are not English?
- Whitespace tokenization
  - What could be the problems with this?
- Subword tokenization
  - Words are broken into segments
  - Example:
    - figs → fig, s
    - management → man, age, ment
  - Segments are discovered from dataset statistics using a technique called byte-pair encoding

Tokenization and the vocabulary

How we break a sequence into tokens decides the vocabulary of the model, and the size of the embedding matrix



Several options possible:

- Each character is a token
- Each word is a token
  - How do we define what a "word" is?
  - What about languages that are not English?
- Whitespace tokenization
  - What could be the problems with this?
- Subword tokenization
  - Words are broken into segments
  - Example:
    - figs → fig, s
    - management → man, age, ment
  - Segments are discovered from dataset statistics using a technique called byte-pair encoding

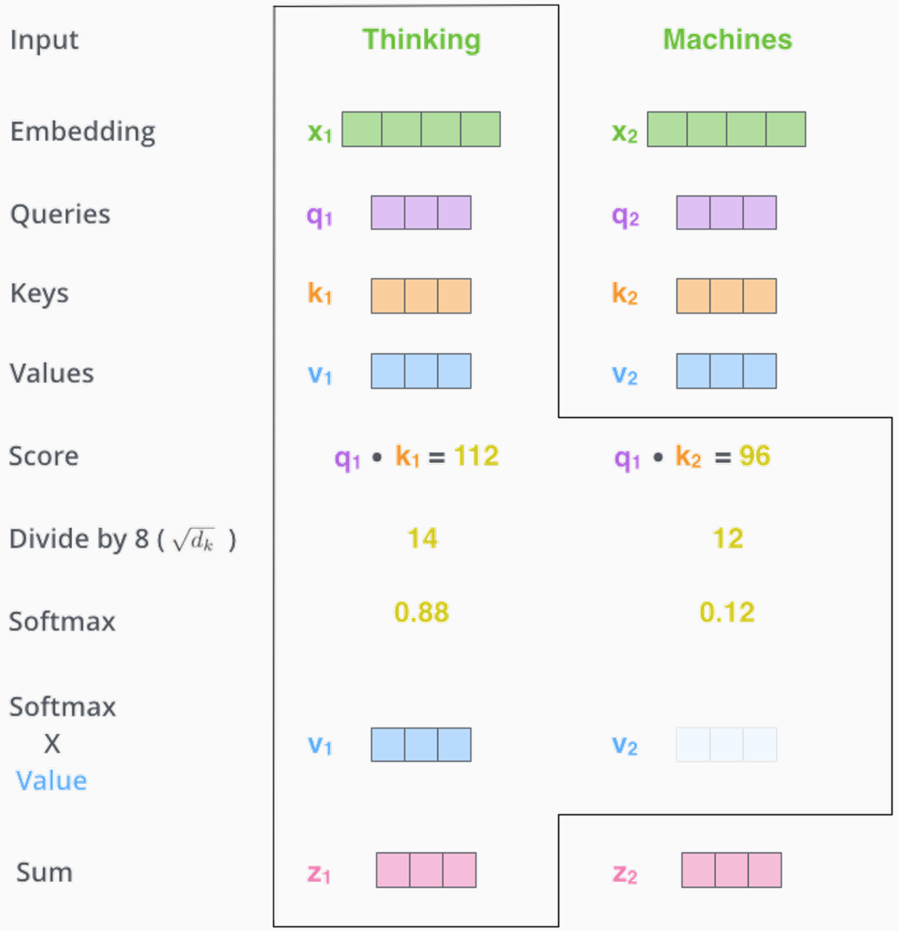
The most common approach today

Details of the model

✓ Tokenization

2. Position embeddings
3. Encoders, decoders and encoder-decoders

Position embeddings



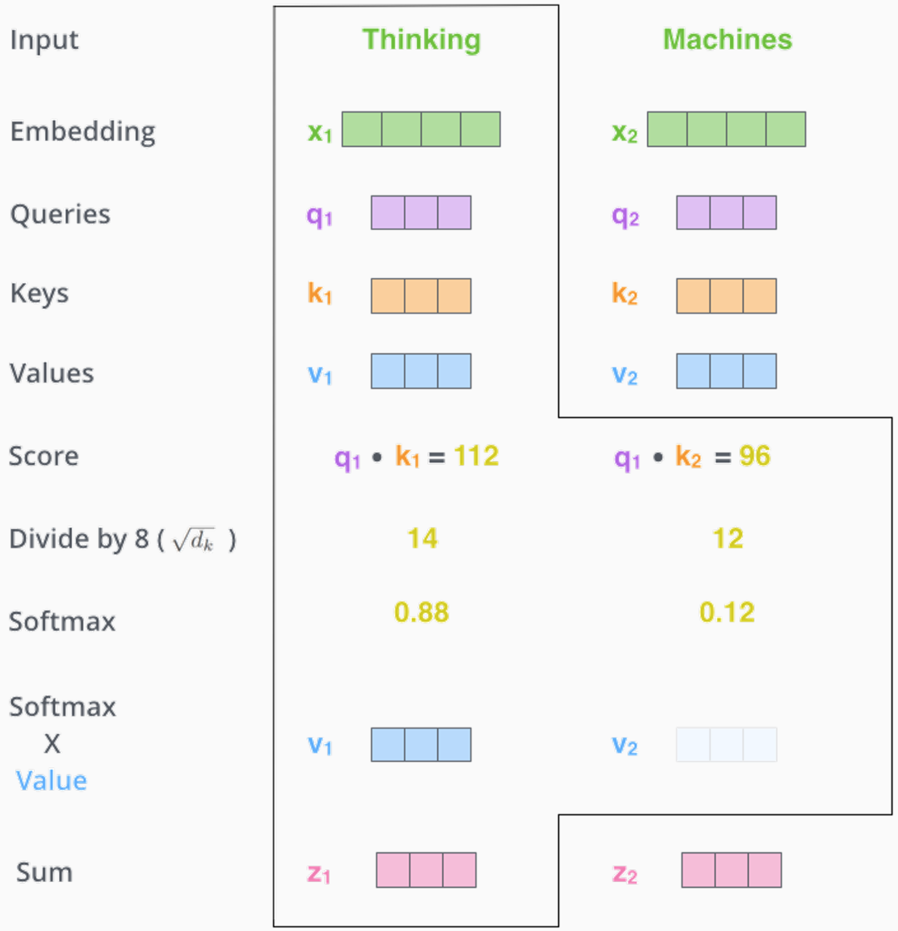
Self attention is symmetric with respect to the position.

If the order of the words were reversed, this will not change the resulting vectors

This could be a problem if we need to encode sequences

Images from https://jalammar.github.io/illustrated-transformer/

Position embeddings



Self attention is symmetric with respect to the position.

If the order of the words were reversed, this will not change the resulting vectors

This could be a problem if we need to encode sequences

The answer: The input embeddings should contain position information

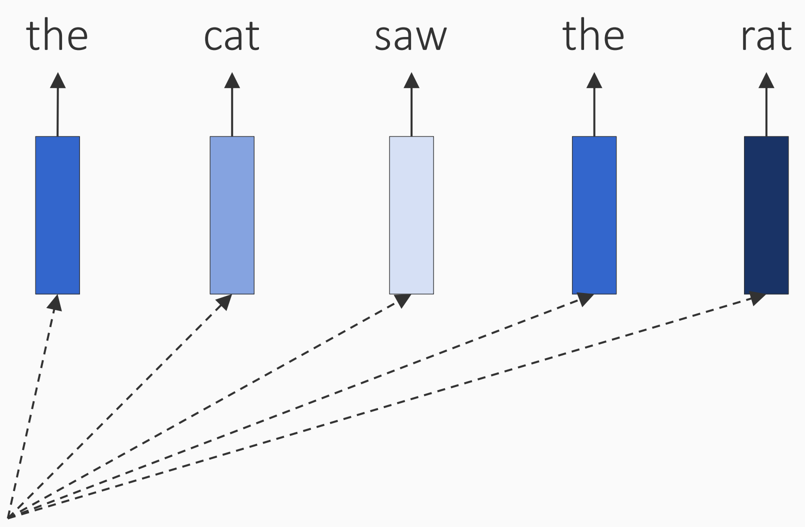
Images from https://jalammar.github.io/illustrated-transformer/

Position information into embeddings

the cat saw the rat

The goal: To design a scheme such that input embeddings contain position information

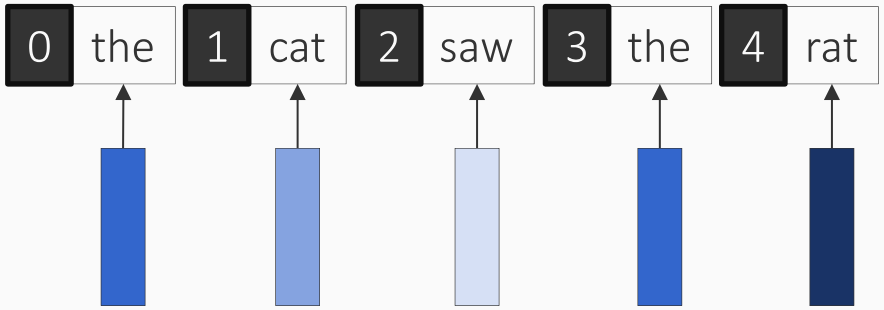
Position information into embeddings



Token embeddings

These embeddings are based on the tokens only. They do not include any information about where it occurs in the sequence

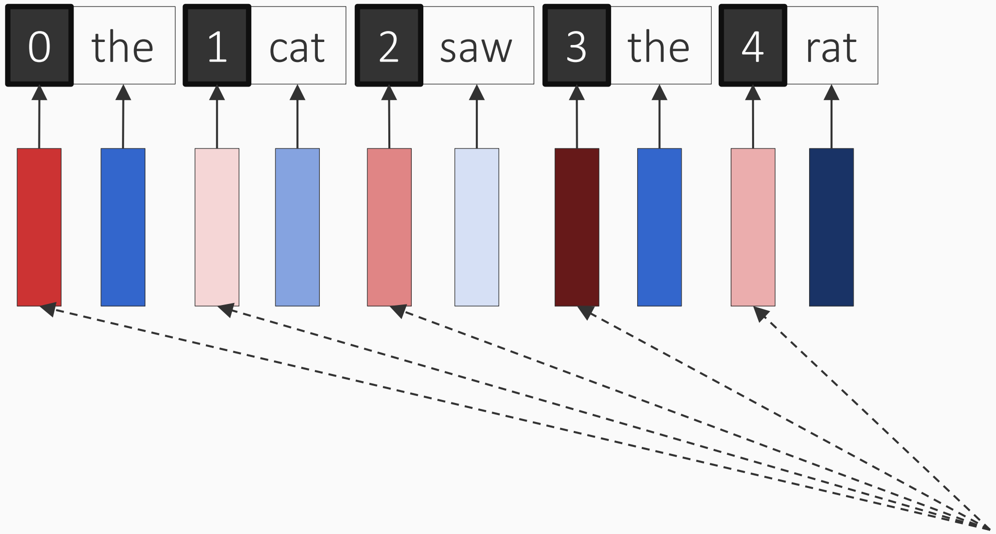
Position information into embeddings



Token embeddings

These embeddings are based on the tokens only. They do not include any information about where it occurs in the sequence

Position information into embeddings



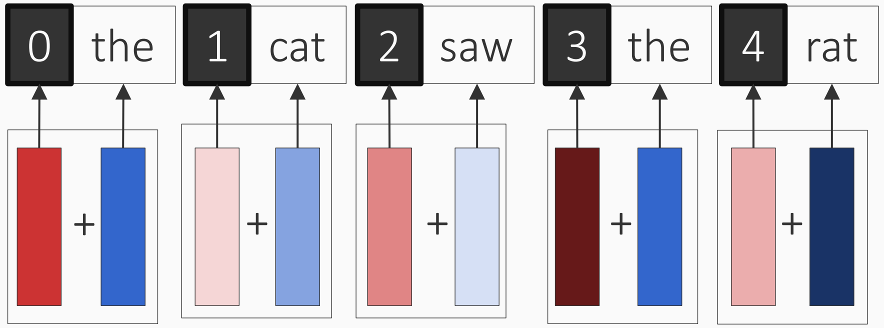
Token embeddings

These embeddings are based on the tokens only. They do not include any information about where it occurs in the sequence

Position embeddings

These embeddings are based on the position only. They do not include any information about what the token is

Position information into embeddings



Input to the transformer network is a sum of these two embeddings

Token embeddings

These embeddings are based on the tokens only. They do not include any information about where it occurs in the sequence

Position embeddings

These embeddings are based on the position only. They do not include any information about what the token is

Position embeddings

We want a vector that represents integers. Many different possibilities

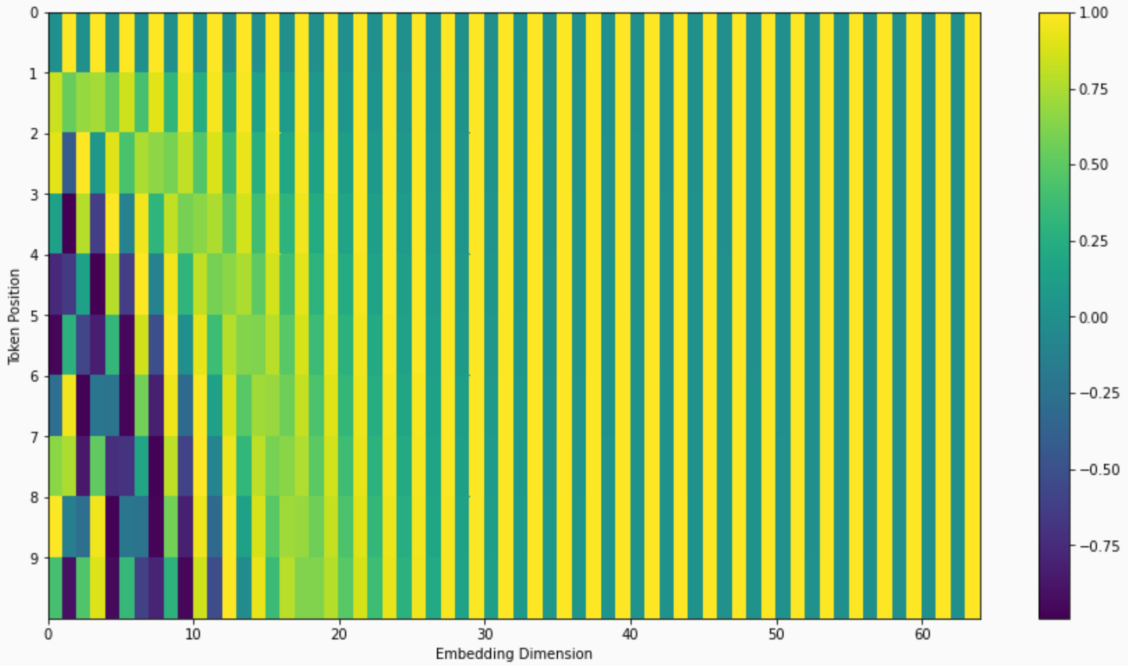
Position embeddings

We want a vector that represents integers. Many different possibilities

The original transformer paper used

$$\mathrm{PE}(t, 2i) = \sin\left(\frac{t}{N^{\frac{2i}{d}}}\right)$$

$$\mathrm{PE}(t, 2i+1) = \cos\left(\frac{t}{N^{\frac{2i}{d}}}\right)$$



Images from https://jalammar.github.io/illustrated-transformer/

Position embeddings

We want a vector that represents integers. Many different possibilities

The original transformer paper used

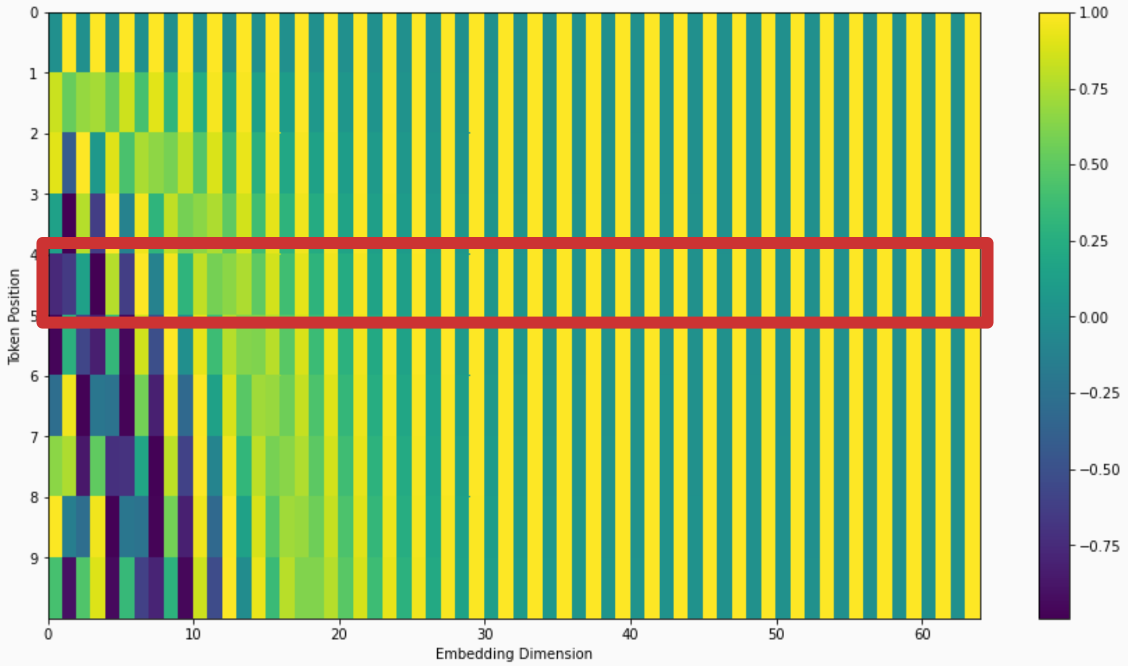
$$\mathrm{PE}(t, 2i) = \sin\left(\frac{t}{N^{\frac{2i}{d}}}\right)$$

$$\mathrm{PE}(t, 2i+1) = \cos\left(\frac{t}{N^{\frac{2i}{d}}}\right)$$

Even numbered element of the position embedding for position t

Odd numbered element of the position embedding for position t

Some position t



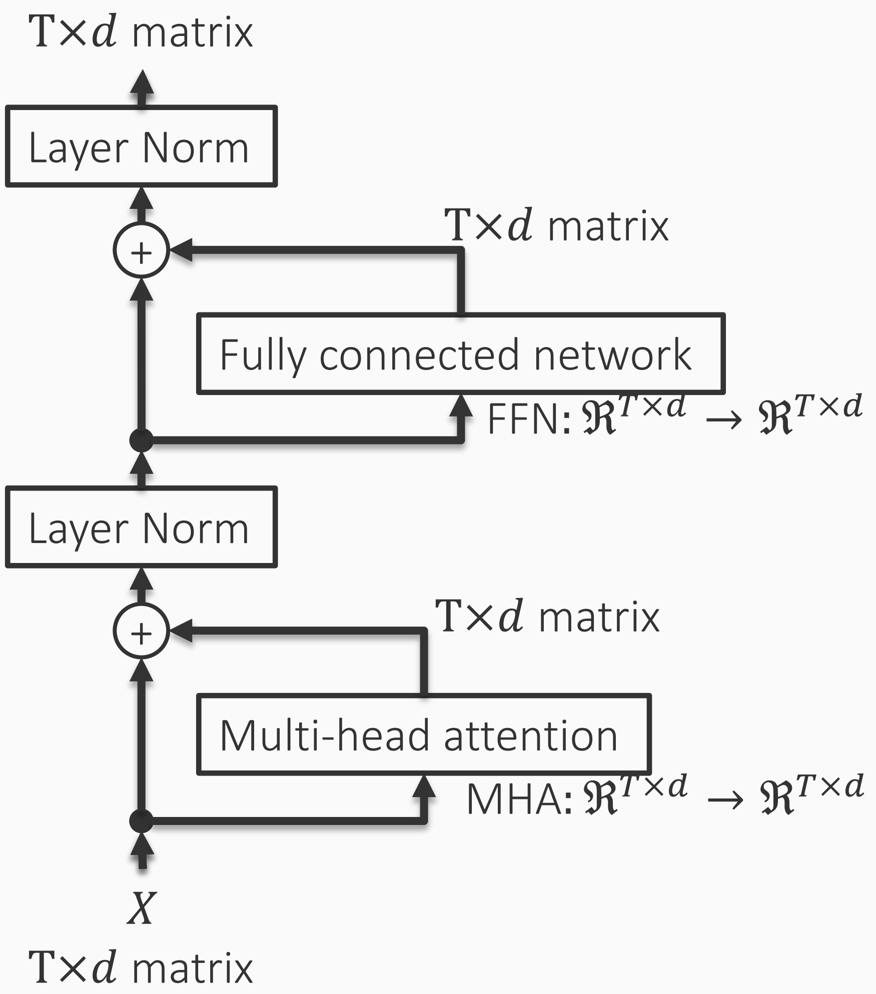
Images from https://jalammar.github.io/illustrated-transformer/

Details of the model

- ✓ Tokenization
- ✓ Position embeddings

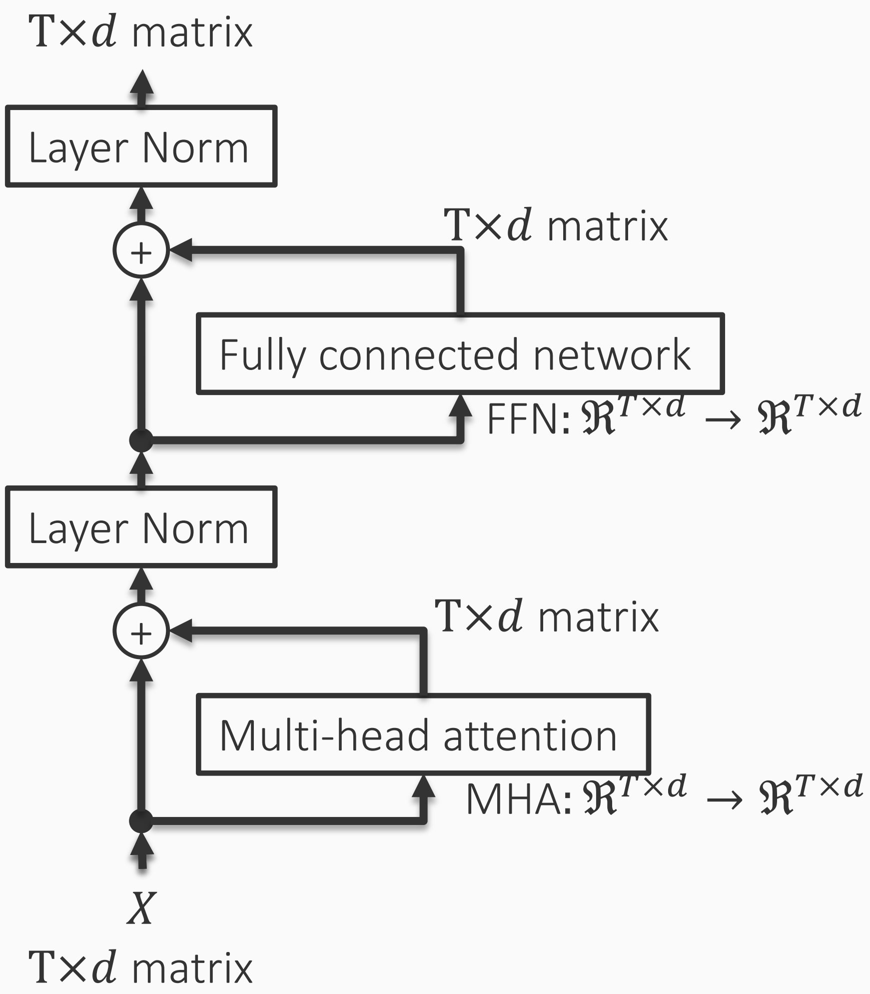
3. Encoders, decoders and encoder-decoders

Encoders, decoders and encoder-decoders



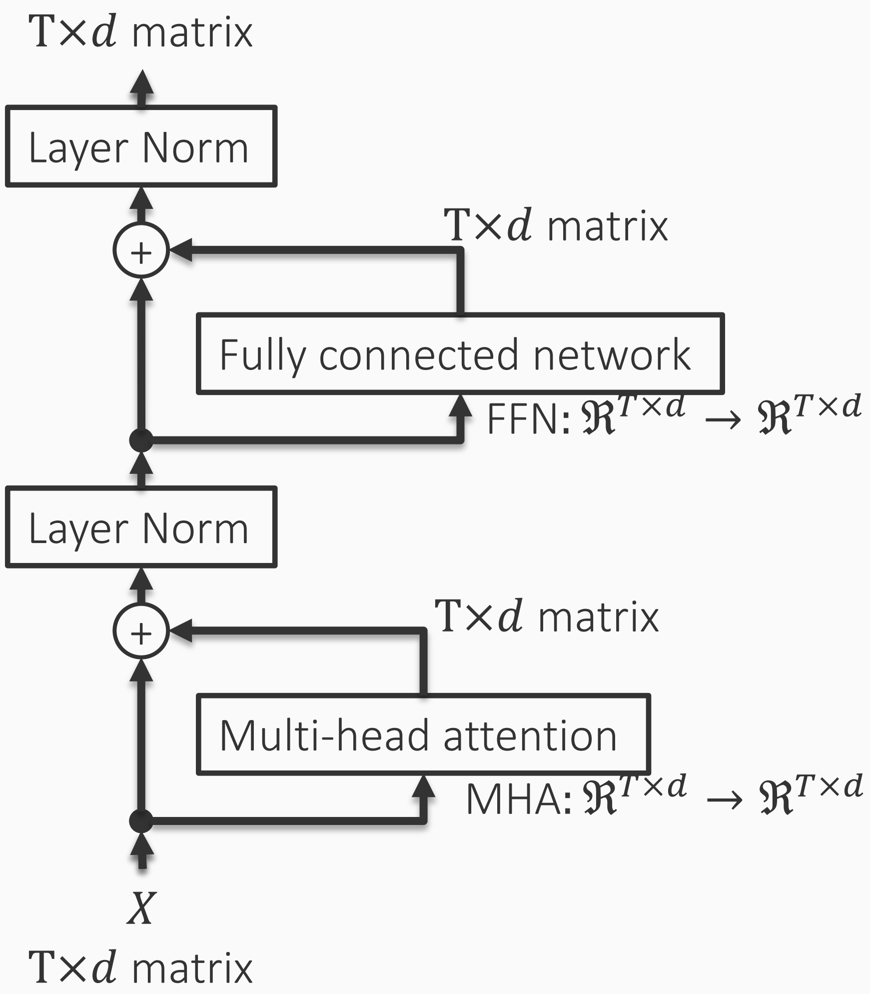
- Encoder: Given a full sequence of tokens (or vectors representing them), encode it into a sequence of vectors
- Decoder: Encode a partial sequence (we do not have access to what comes next in the sequence)
- Encoder-decoder: First encode a sequence, and then conditioned on the encoding, decode it

Encoders, decoders and encoder-decoders



- Encoder: Given a full sequence of tokens (or vectors representing them), encode it into a sequence of vectors
- Decoder: Encode a partial sequence (we do not have access to what comes next in the sequence)
- Encoder-decoder: First encode a sequence, and then conditioned on the encoding, decode it

Encoders, decoders and encoder-decoders

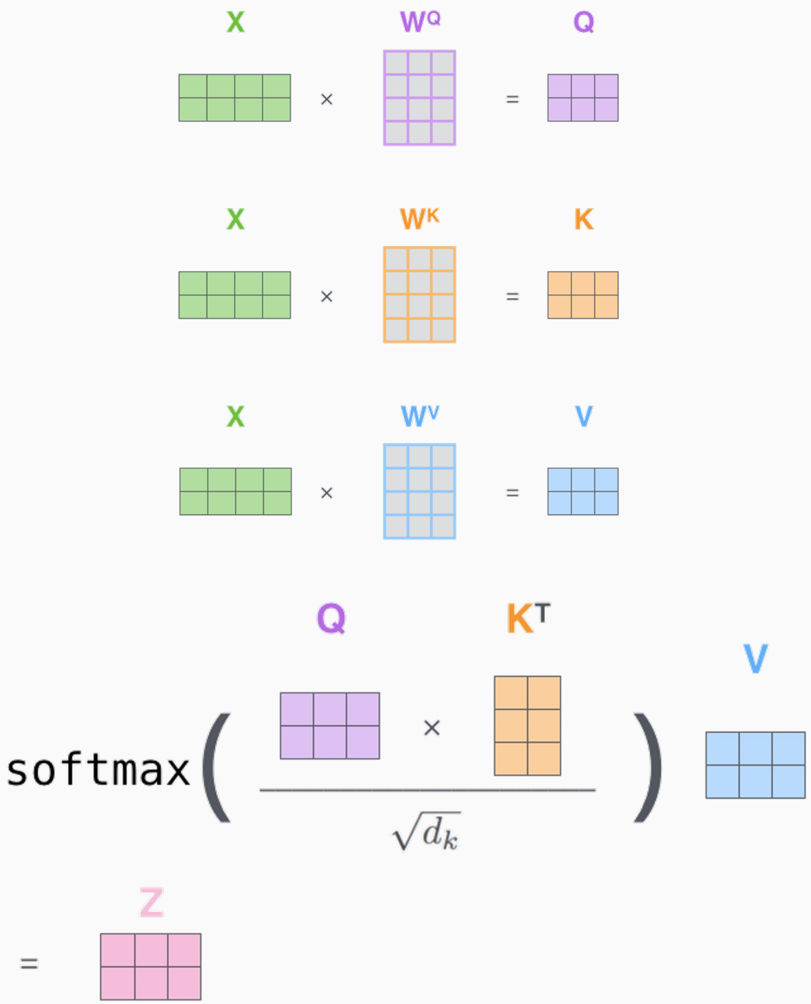


- Encoder: Given a full sequence of tokens (or vectors representing them), encode it into a sequence of vectors
- Decoder: Encode a partial sequence (we do not have access to what comes next in the sequence)
- Encoder-decoder: First encode a sequence, and then conditioned on the encoding, decode it

In the decoder mode, even if we know that the full sequence can have T tokens, we only see a prefix of the sequence. That is, the first k tokens.

How can we modify this architecture to accommodate this fact?

Decoder transformers

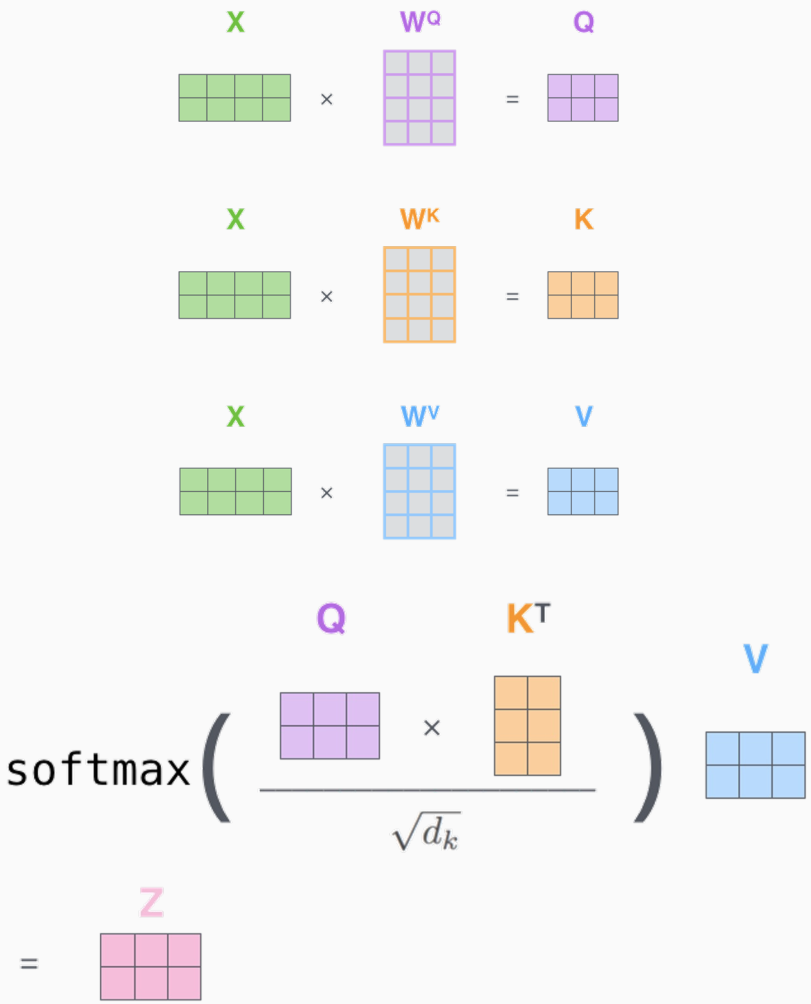


In the decoder mode, even if we know that the full sequence can have T tokens, we only see a prefix of the sequence. That is, the first k tokens.

How can we modify this architecture to accommodate this fact?

Answer: Modify the self-attention

Decoder transformers



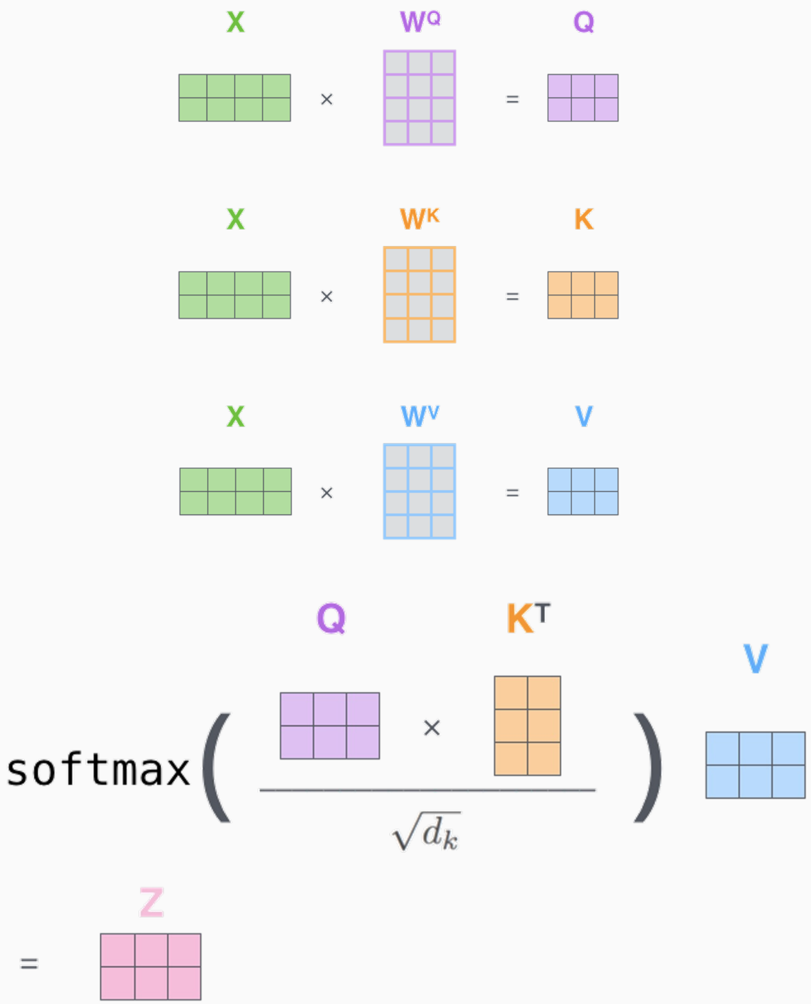
In the decoder mode, even if we know that the full sequence can have T tokens, we only see a prefix of the sequence. That is, the first k tokens.

How can we modify this architecture to accommodate this fact?

Answer: Modify the self-attention

In the decoder mode, at training time, we need to simulate the fact that only the previous tokens can affect a certain token

Decoder transformers



In the decoder mode, even if we know that the full sequence can have T tokens, we only see a prefix of the sequence. That is, the first k tokens.

How can we modify this architecture to accommodate this fact?

Answer: Modify the self-attention

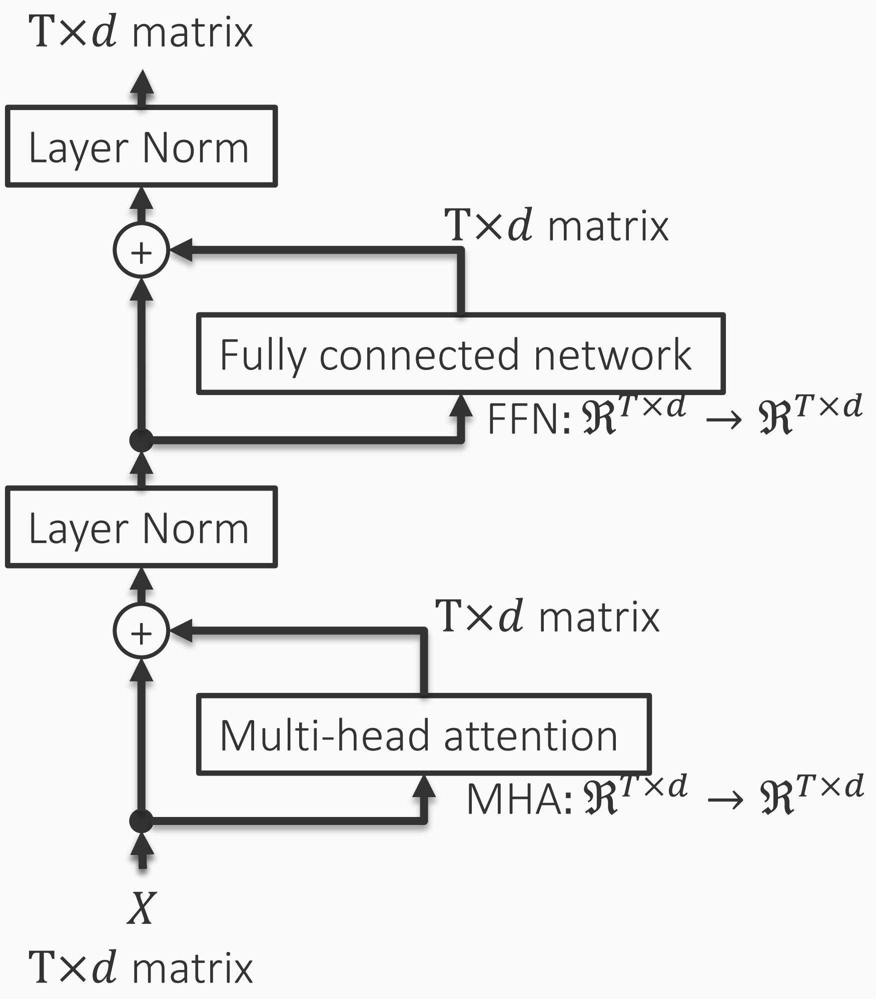
In the decoder mode, at training time, we need to simulate the fact that only the previous tokens can affect a certain token

For any token that comes after , mask future positions before the softmax step.

That is, set those values to $-\infty$

What happens to the corresponding softmaxes?

Encoders, decoders and encoder-decoders

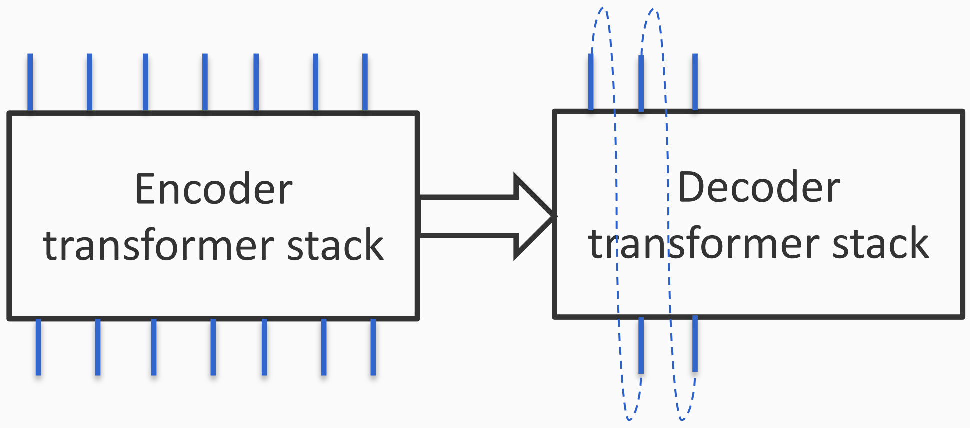


- Encoder: Given a full sequence of tokens (or vectors representing them), encode it into a sequence of vectors

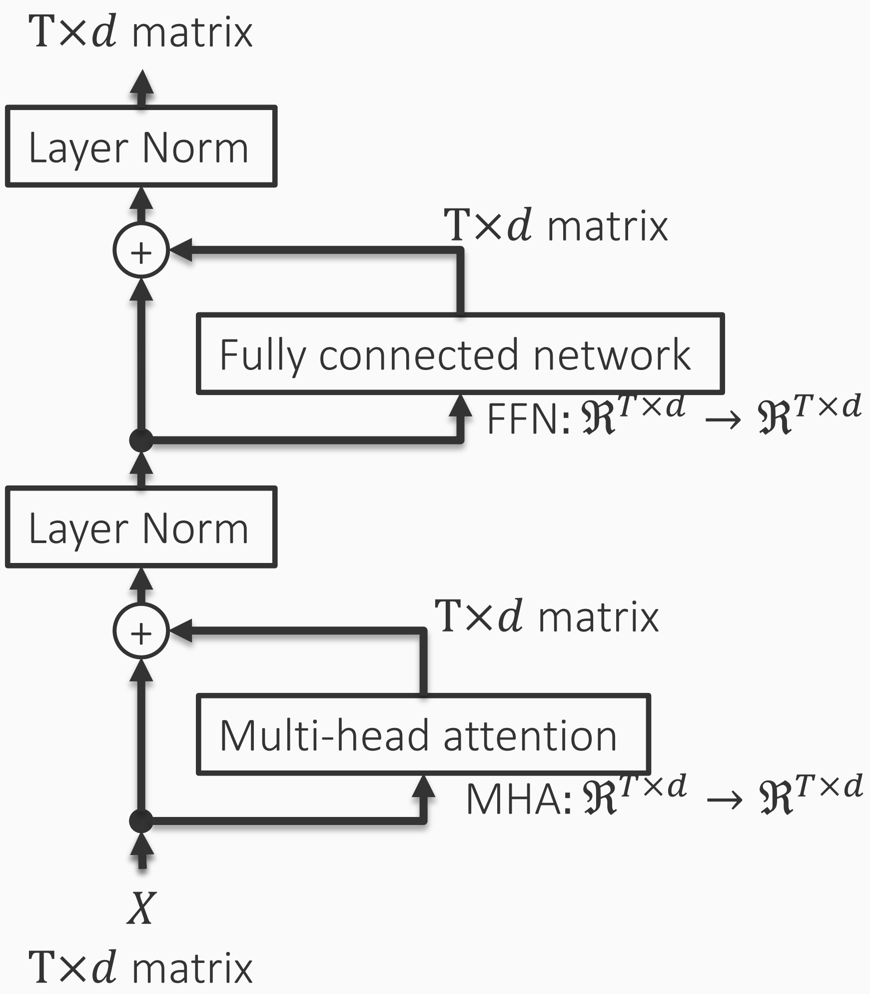
- Decoder: Encode a partial sequence (we do not have access to what comes next in the sequence)

- Encoder-decoder: First encode a sequence, and then conditioned on the encoding, decode it

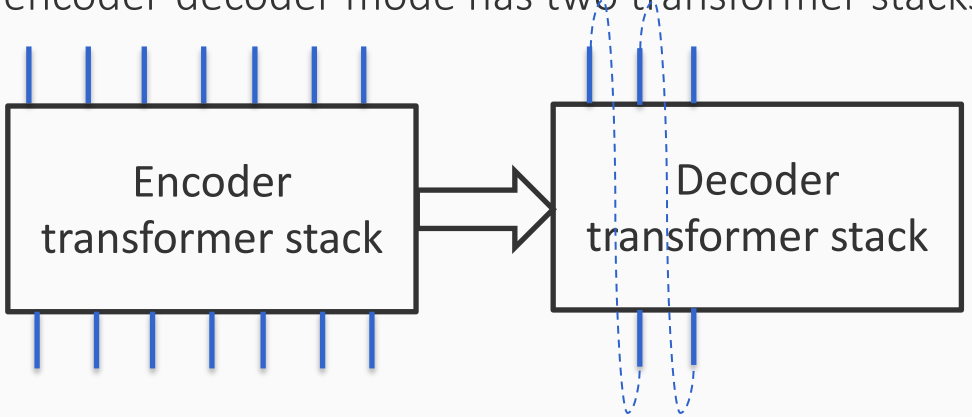
The encoder-decoder mode has two transformer stacks.



Encoders, decoders and encoder-decoders

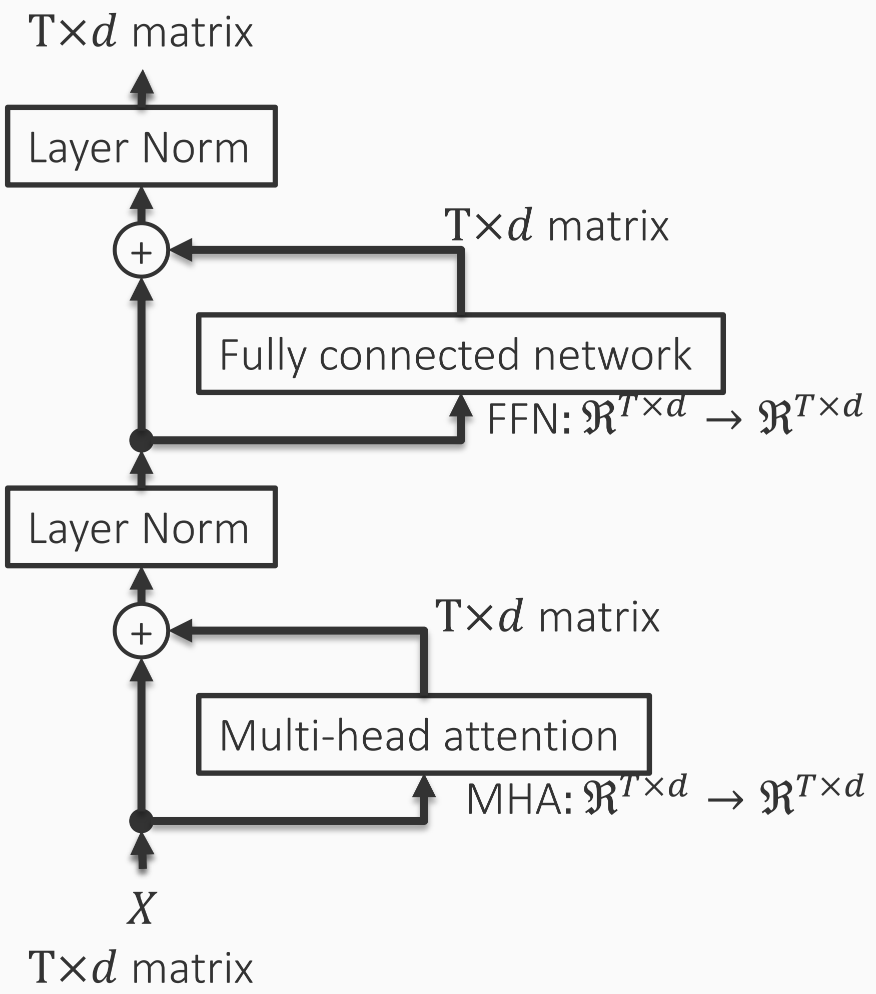


The encoder-decoder mode has two transformer stacks.

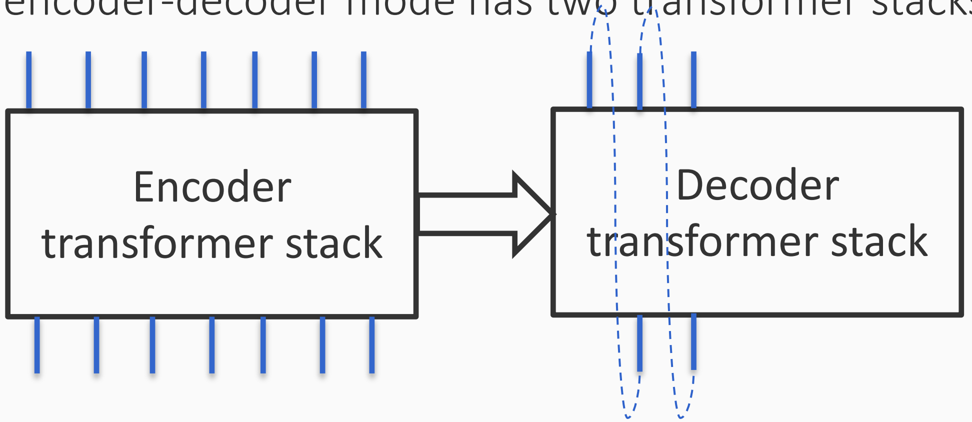


What we want: The decoder output should depend on the tokens of the encoder

Encoders, decoders and encoder-decoders



The encoder-decoder mode has two transformer stacks.



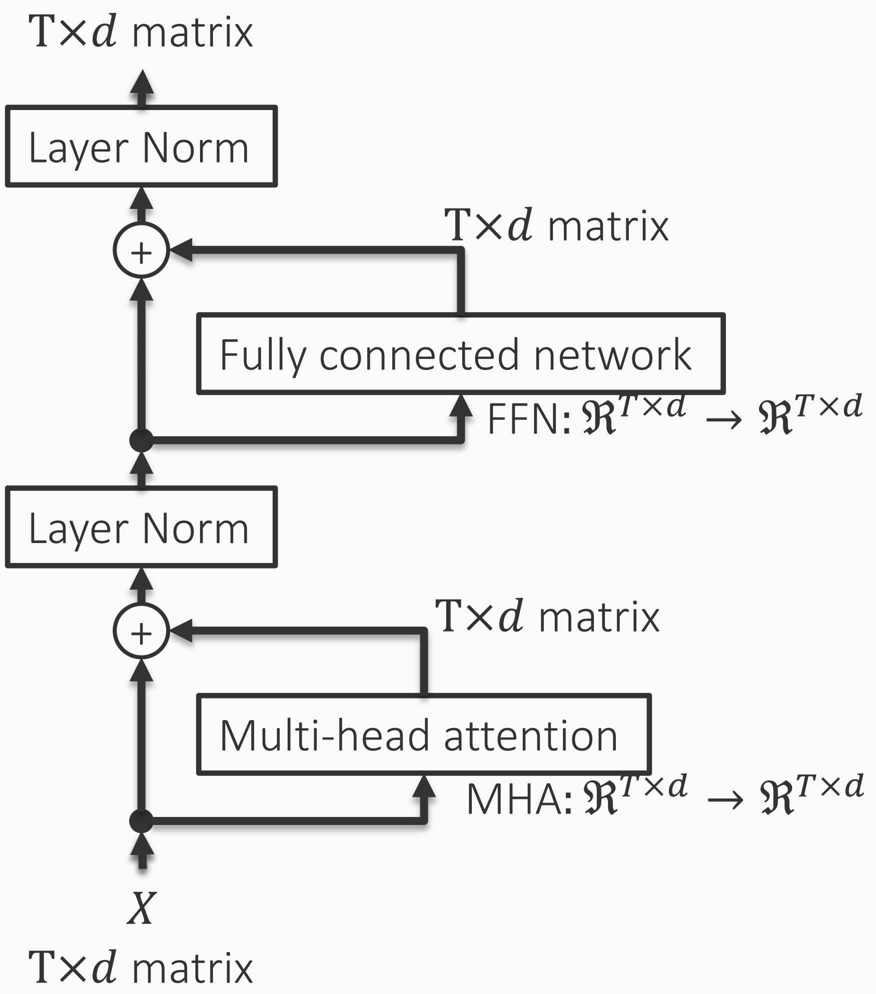
What we want: The decoder output should depend on the tokens of the encoder

The solution: Encoder-decoder attention

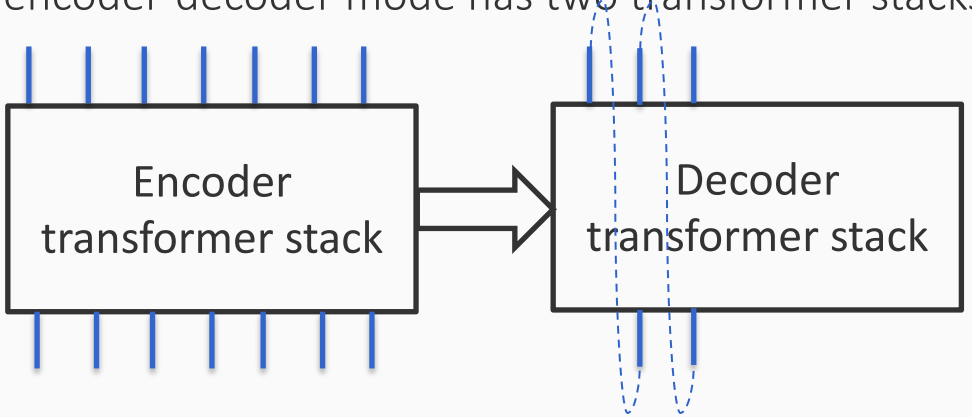
Queries come from the decoder layer below

Keys and values come from the encoder output

Encoders, decoders and encoder-decoders



The encoder-decoder mode has two transformer stacks.



What we want: The decoder output should depend on the tokens of the encoder

The solution: Encoder-decoder attention

- Queries come from the decoder layer below
- Keys and values come from the encoder output

Few different interpretations of this in the literature

- Either replace the MHA, or add a second MHA layer after the existing one
- Adds more layer norms

Outline

- The challenge of modeling sequences
- The transformer architecture
  - The big picture
- Details and fine print
- The impact of transformers

Impact of transformer models

- The biggest state-of-the-art increase in NLP research in the last decade is from these models
  - NLP research = BERTology?
  - NLP research = Transformer-powered LLMs?
  - Default toolset for NLP research and products today
- More mainstream adoption of human language technology
- Growing use in computer vision as well
  - Vision Transformers are comparable/better than CNN models
- Transformers power…
  - …search and translation engines
  - …language models
  - …products like Github Copilot that convert textual description to code
  - …the NLP part of systems like DALL-E that create images based on prompts

Language Modeling: 2018-today

The goal: Recursively keep generating the next word in text given words so far

Improving Language Understanding by Generative Pre-Training

Alec Radford, OpenAI, alec@openai.com
Karthik Narasimhan, OpenAI, karthikn@openai.com
Tim Salimans, OpenAI, tim@openai.com
Ilya Sutskever, OpenAI, ilyasu@openai.com

GPT (2018), 117 million parameters

Language Models are Unsupervised Multitask Learners

Alec Radford [*1] Jeffrey Wu [*1] Rewon Child [1] David Luan [1] Dario Amodei [**1] Ilya Sutskever [**1]

GPT-2 (2019), 1.5 billion parameters

Language Models are Few-Shot Learners

Tom B. Brown[*] Benjamin Mann[*] Nick Ryder[*] Melanie Subbiah[*]

GPT-3 (2020), 175 billion parameters
NeurIPS 2020 best paper

The BERT family (2019-today)

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

- A family of models that were trained by masking words in a sentence and asking the transformer model to fill in the blank
  Along the way, it learns what words mean!

- Original 2019 paper followed by numerous variants like DistillBERT, RoBERTa-large/base, DeBERTa, multilingual BERT (mBERT), XLM,…

The BERT family (2019-today)

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

- A family of models that were trained by masking words in a sentence and asking the transformer model to fill in the blank
  Along the way, it learns what words mean!

- Original 2019 paper followed by numerous variants like DistillBERT, RoBERTa-large/base, DeBERTa, multilingual BERT (mBERT), XLM,…

A Primer in BERTology: What We Know About How BERT Works

Anna Rogers
Center for Social Data Science
University of Copenhagen
arogers@sodas.ku.dk

Olga Kovaleva
Dept. of Computer Science
University of Massachusetts Lowell
okovalev@cs.uml.edu

Anna Rumshisky
Dept. of Computer Science
University of Massachusetts Lowell
arum@cs.uml.edu

A good survey of hundreds of papers

Wrapping up

Transformers are a neural network architecture designed to handle sequences

- But operate in parallel over the entire sequence

The architecture has many different building blocks

- Multi-head attention that “mixes” the input tokens
- Fully-connected layers that operate in parallel over the input tokens
- Residual connections all around
- Layer norms all around

Encoders versus decoders versus encoder-decoder blocks

Many details. A clean overview of the entire stack from the ground up is in

Mary Phuong and Marcus Hutter. 2022. Formal Algorithms for Transformers. arXiv:2207.09238 [cs].

Massive impact on the current NLP practice

- All the state-of-the-art NLP models today are built with transformers