Outline

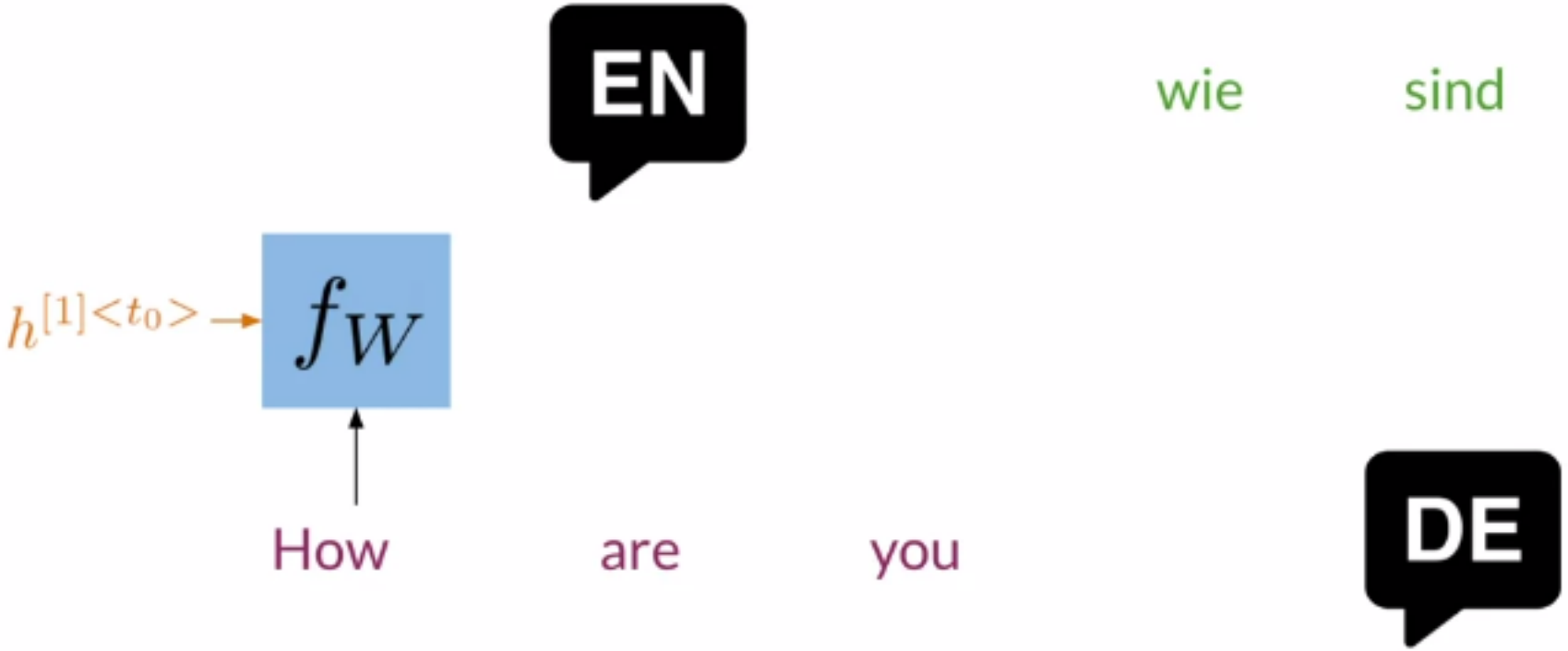
- Issues with RNNs
- Comparison with Transformers

Neural Machine Translation

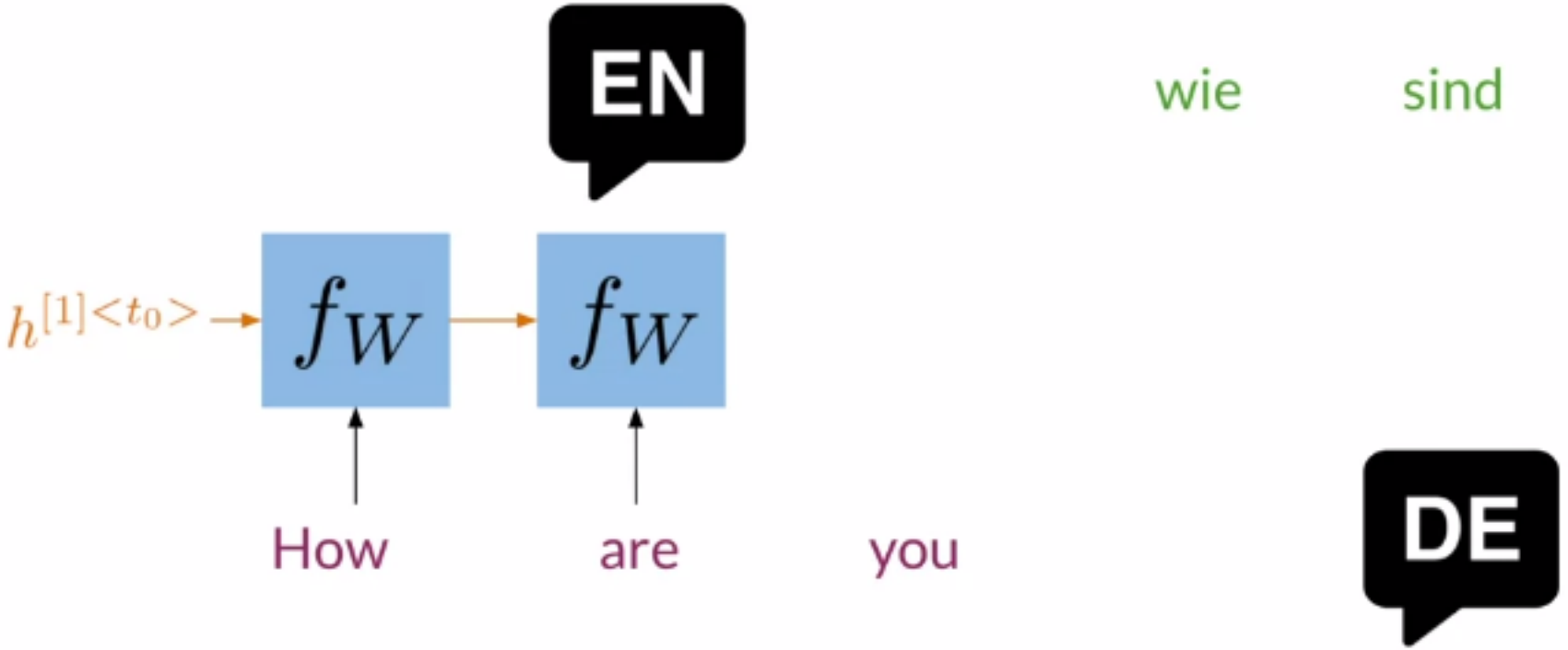
wie sind sie

How are you

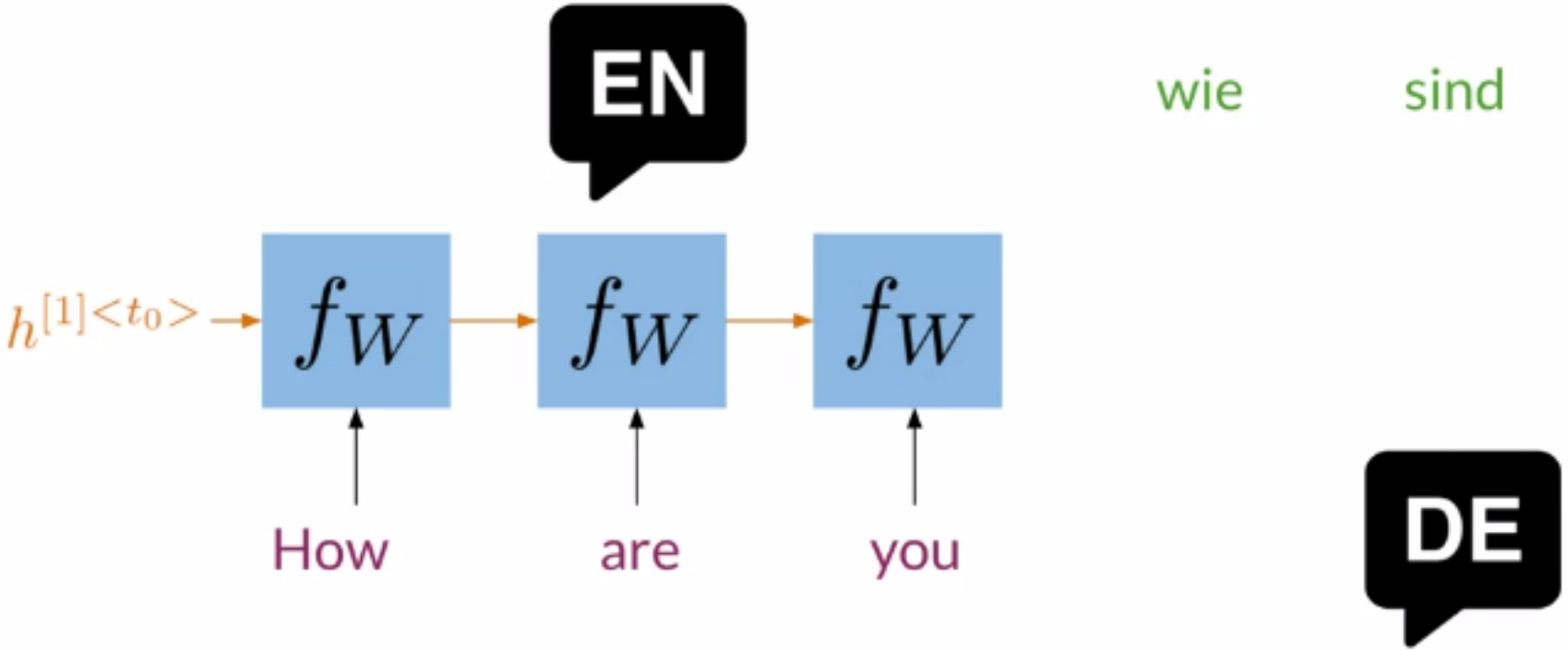
Neural Machine Translation



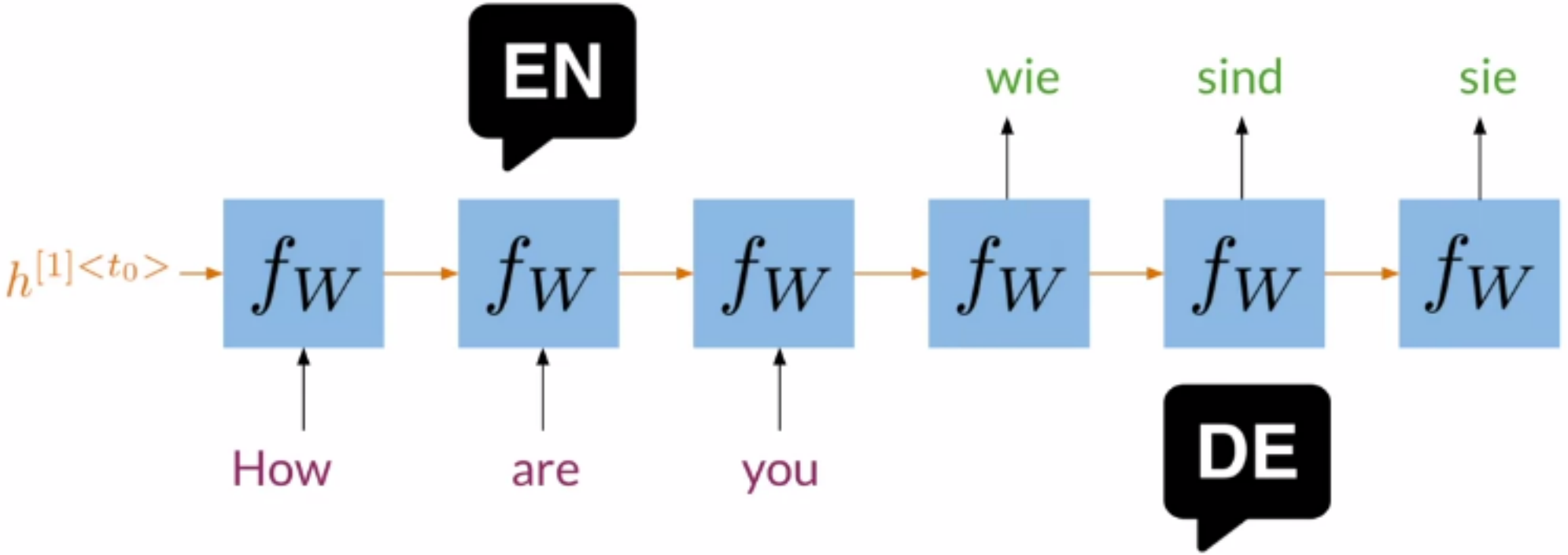
Neural Machine Translation



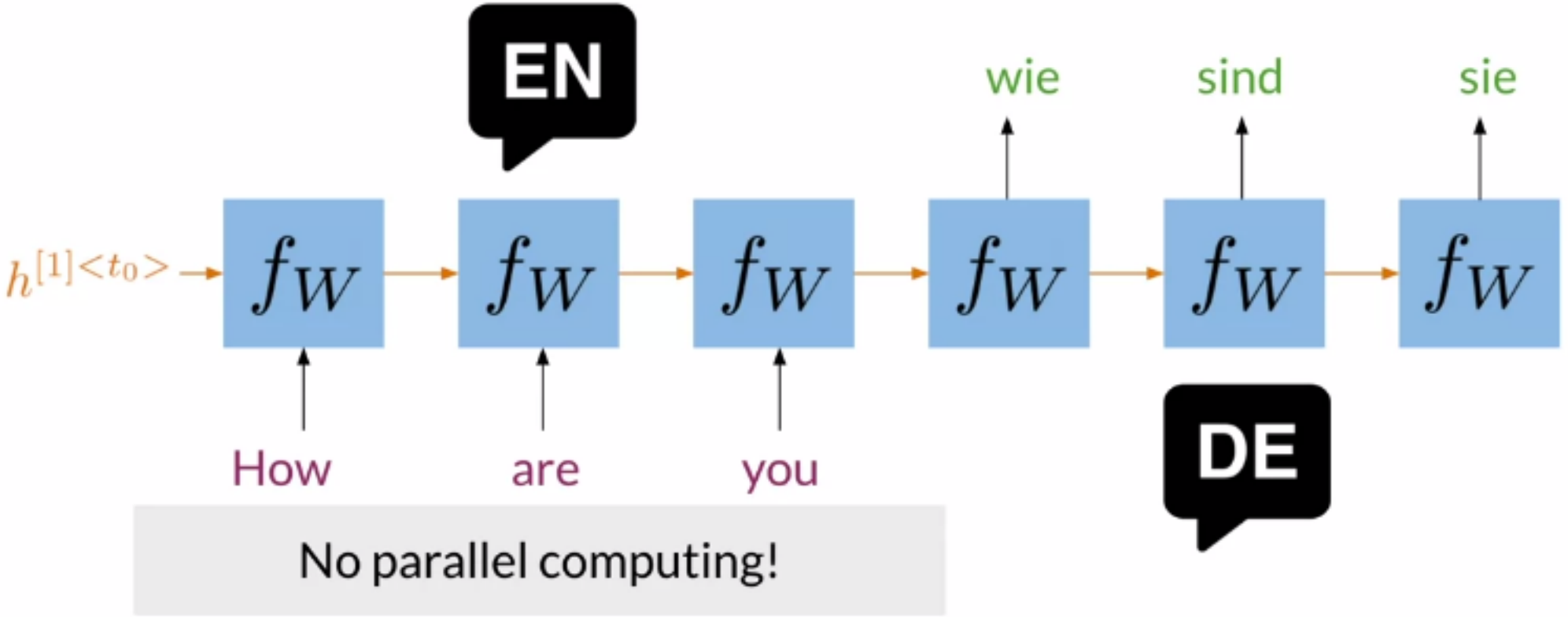
Neural Machine Translation



Neural Machine Translation

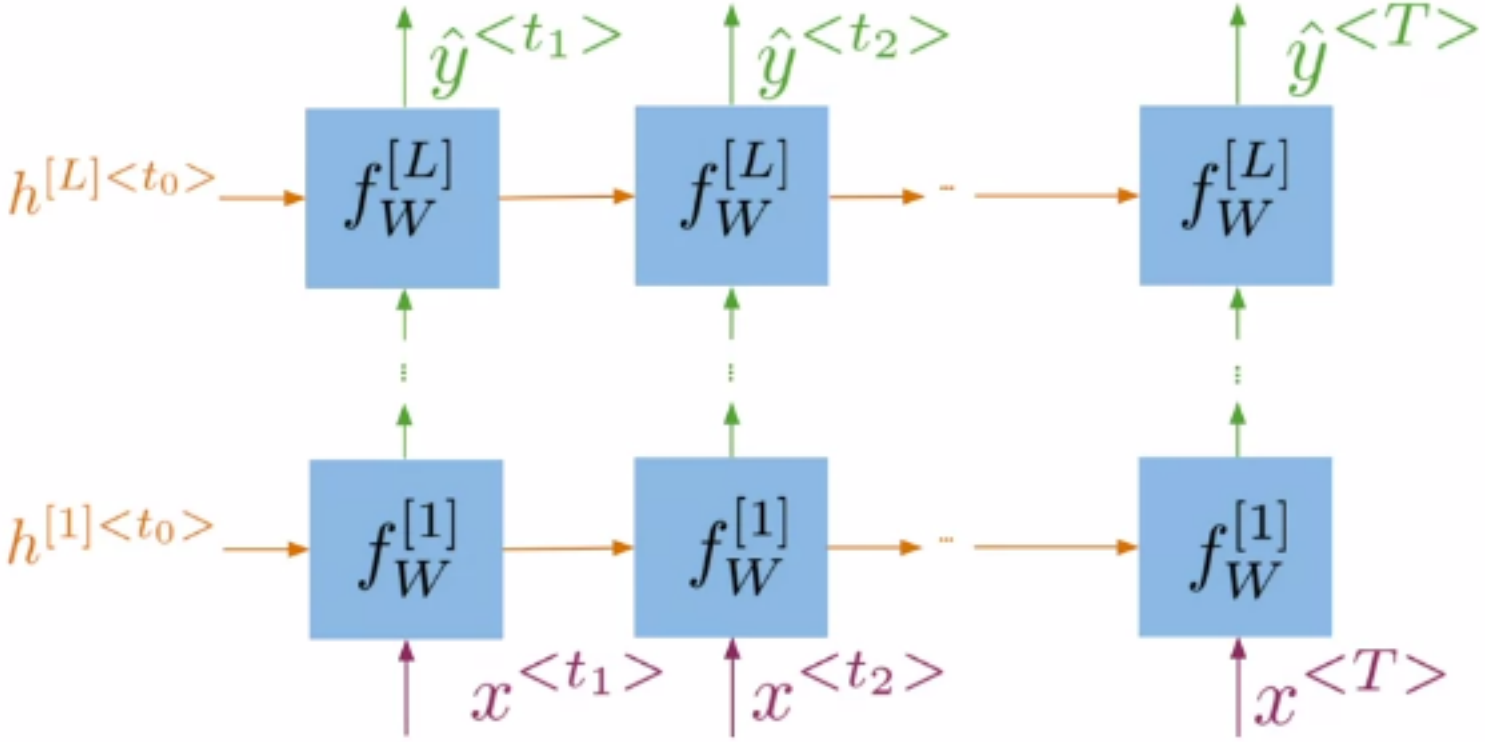


Neural Machine Translation

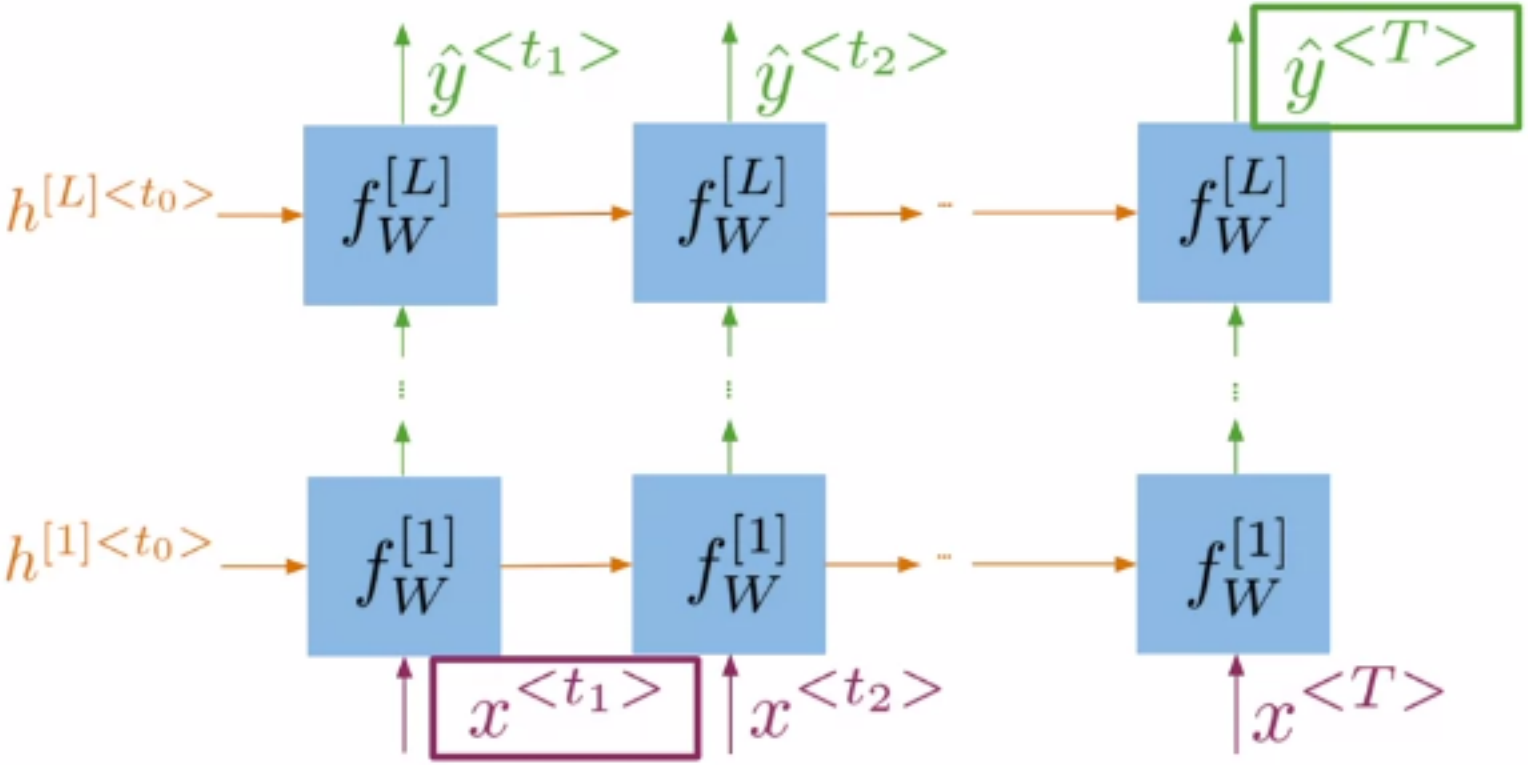


Seq2Seq Architectures

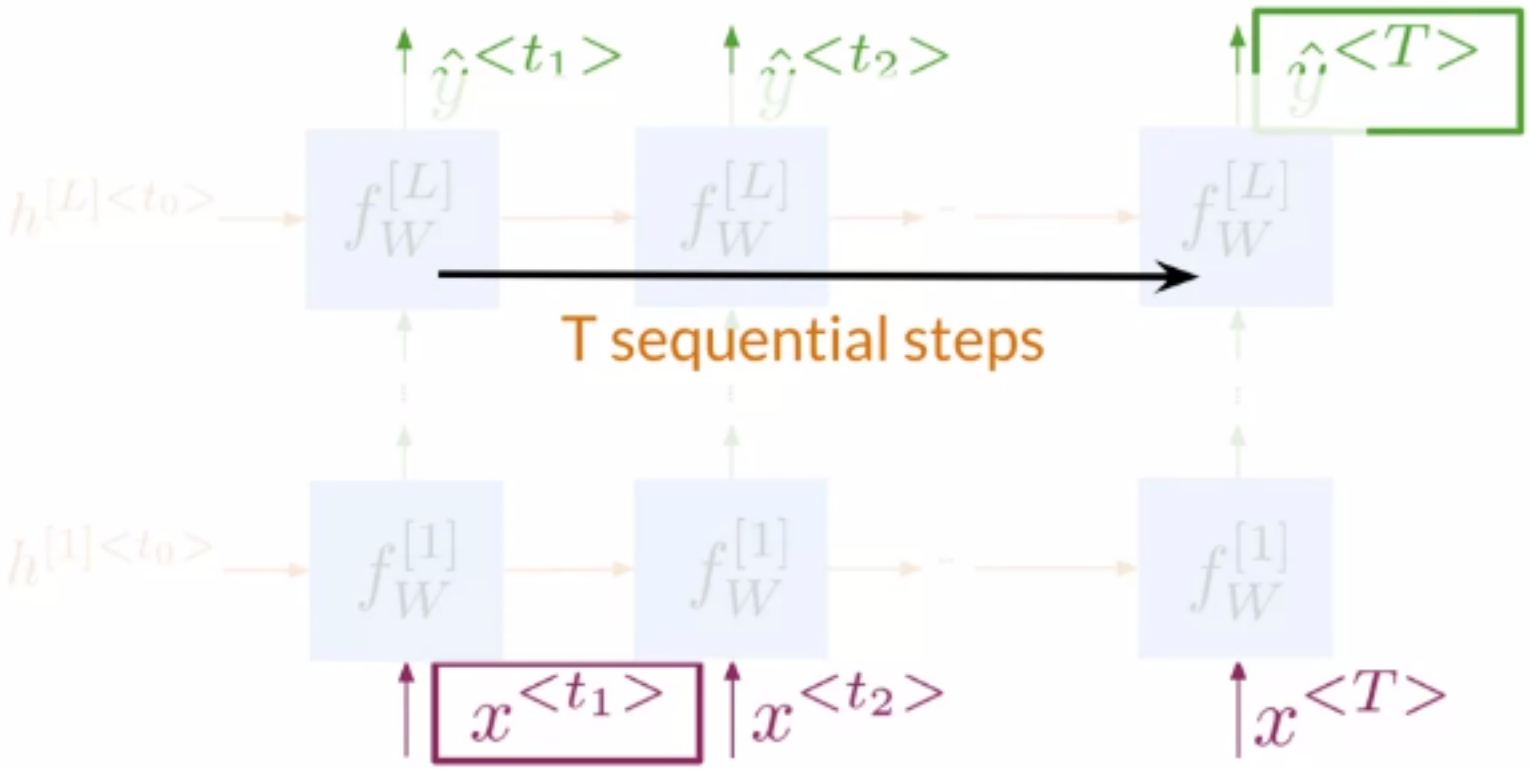
Seq2Seq Architectures



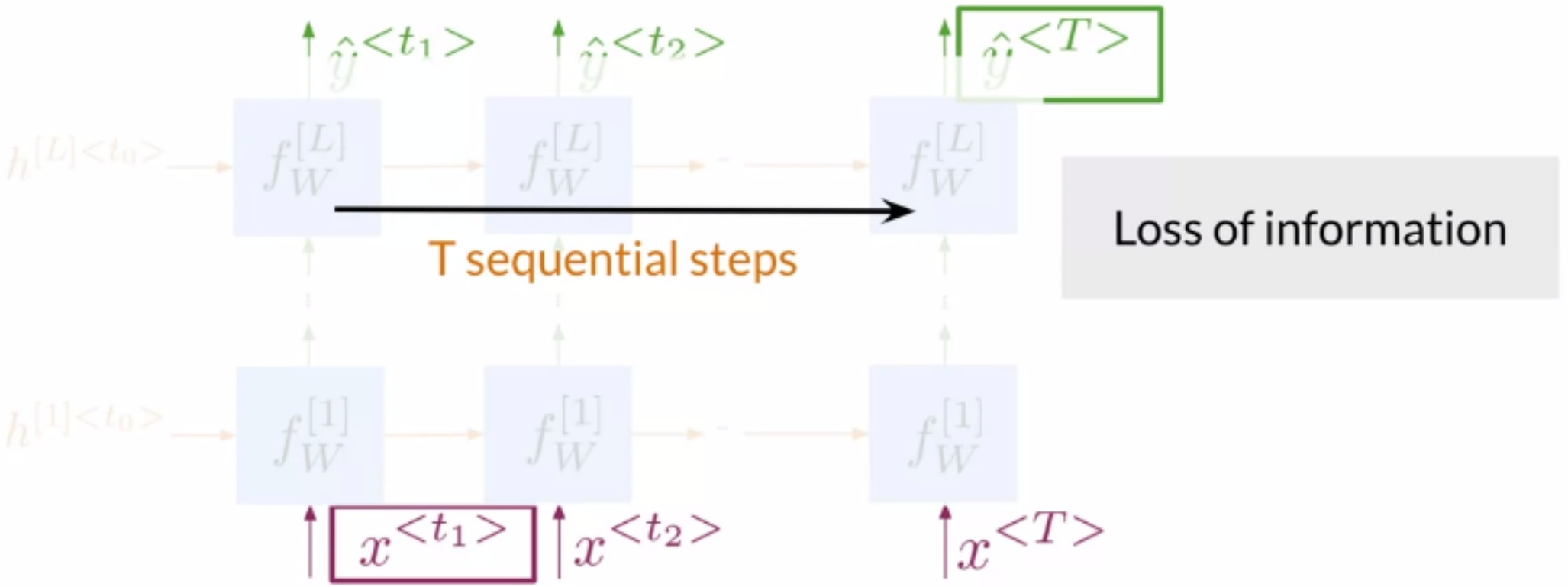
Seq2Seq Architectures



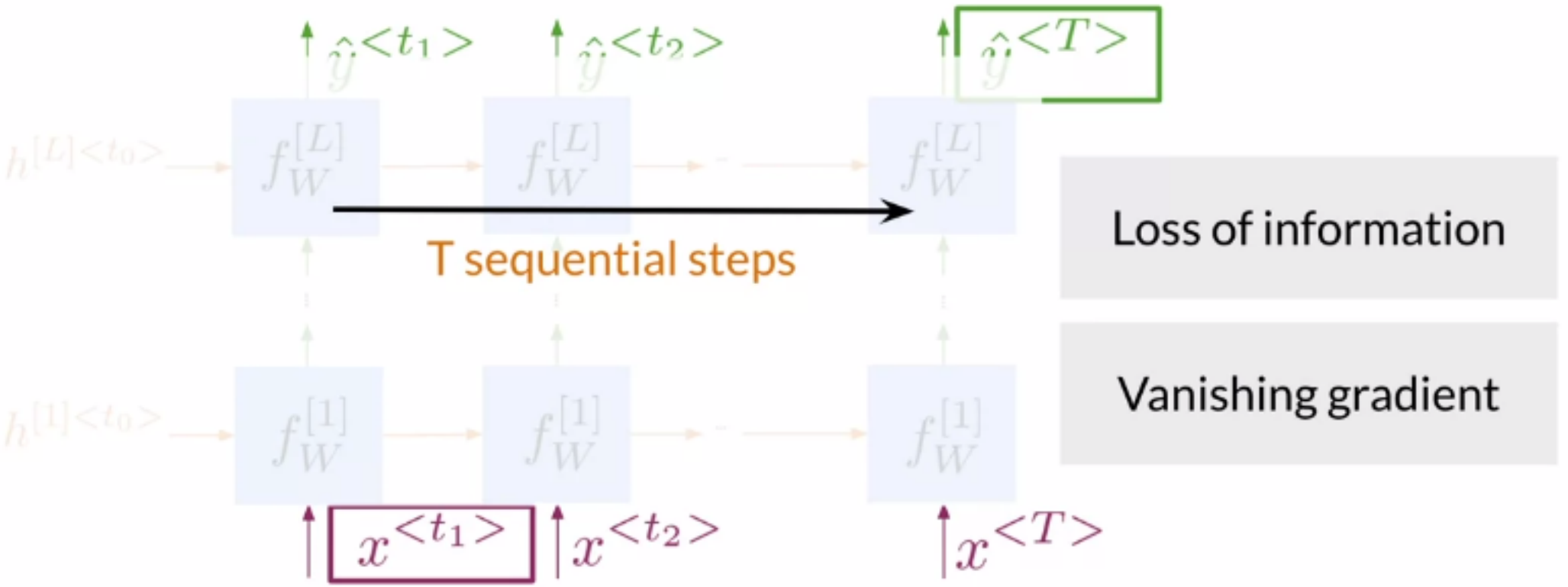
Seq2Seq Architectures



Seq2Seq Architectures

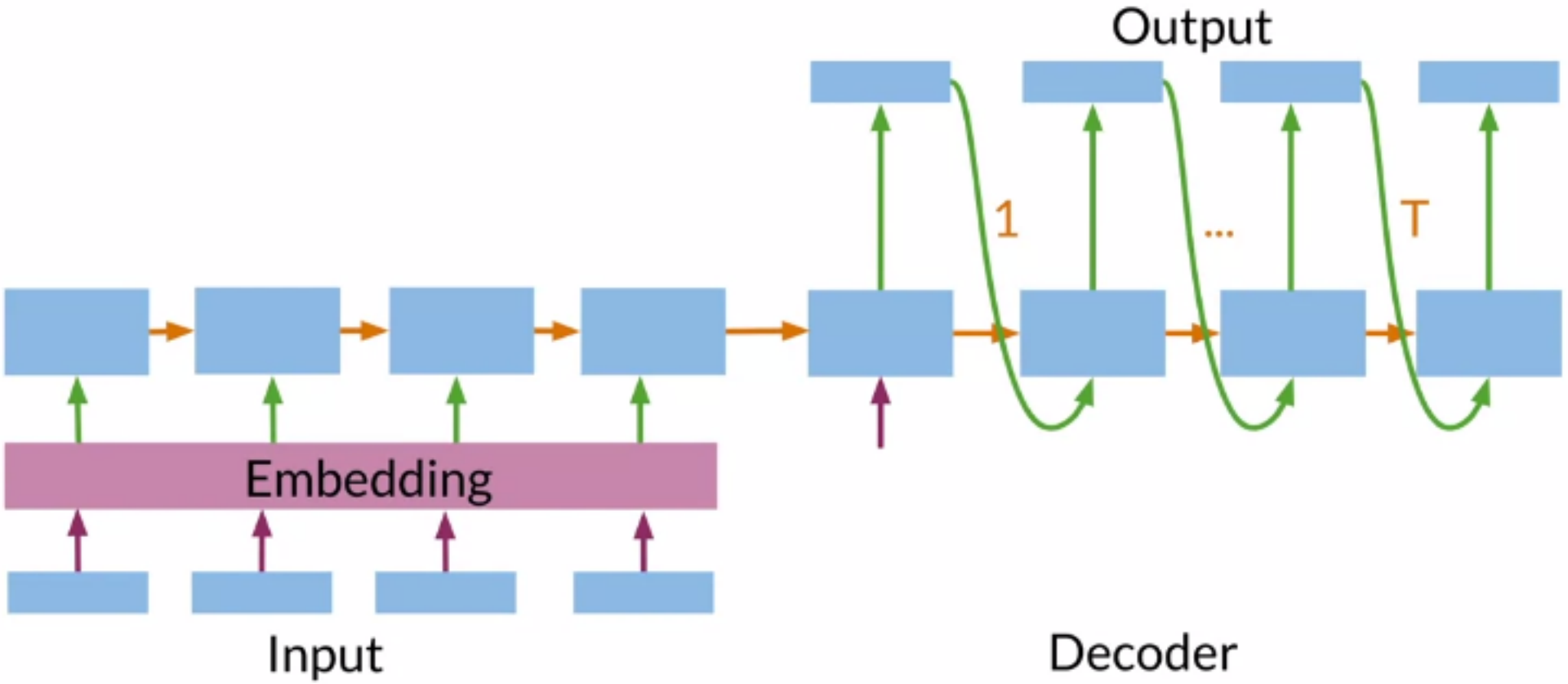


Seq2Seq Architectures

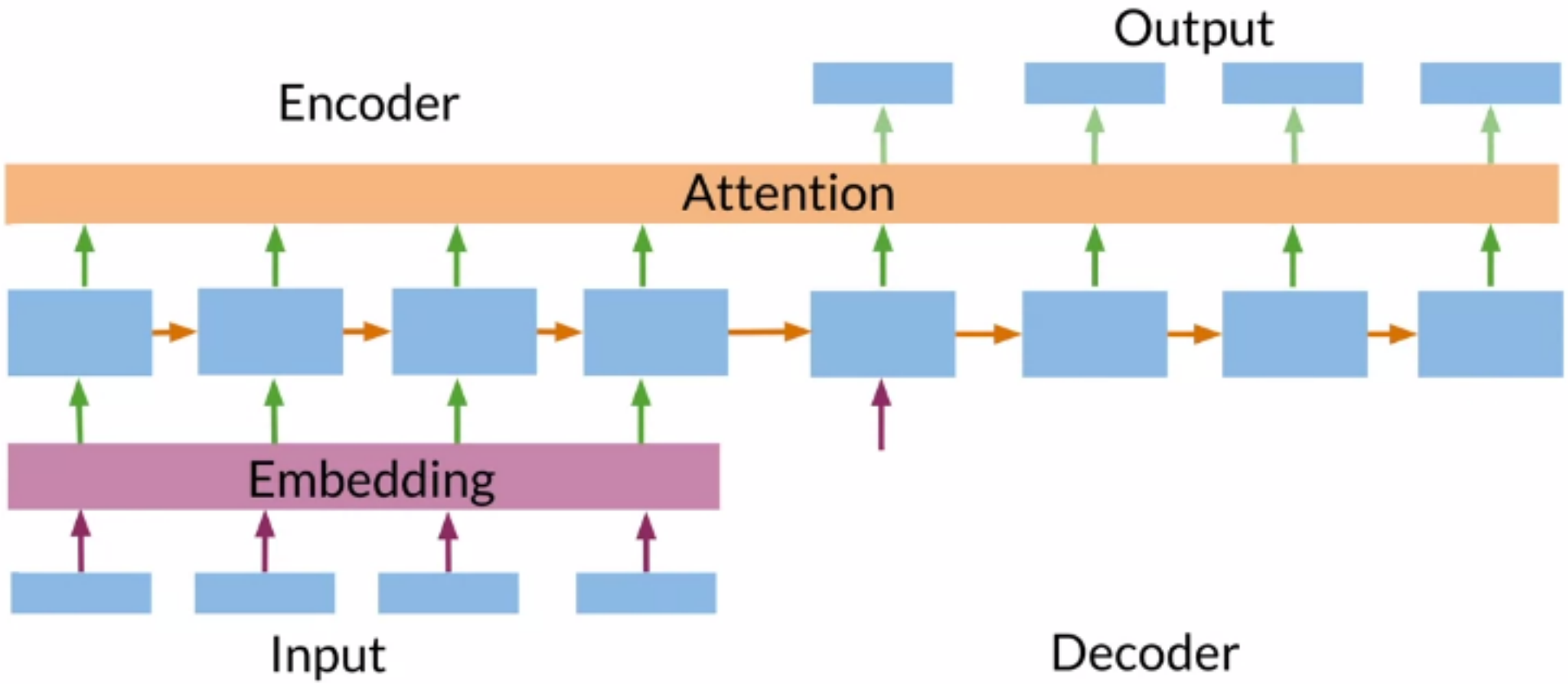


RNNs vs Transformer: Encoder-Decoder

RNNs vs Transformer: Encoder-Decoder

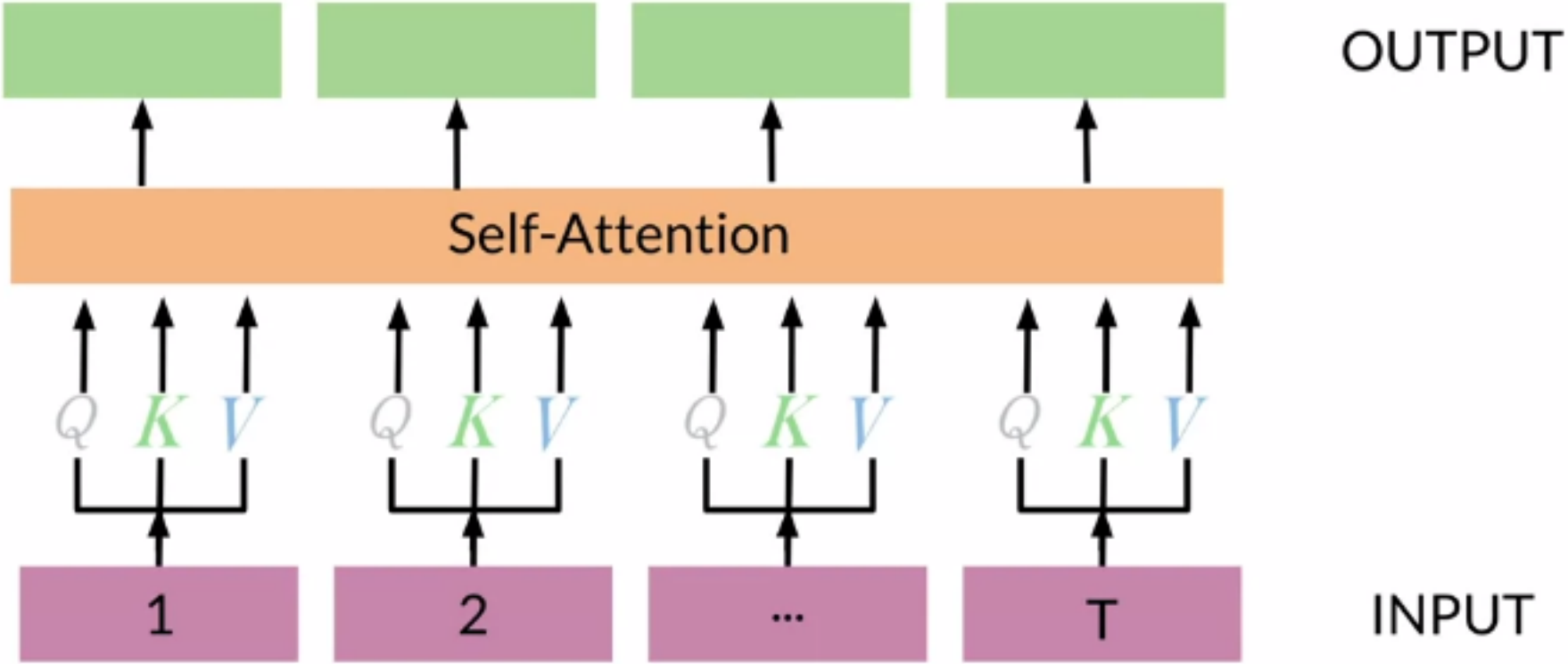


RNNs vs Transformer: Encoder-Decoder

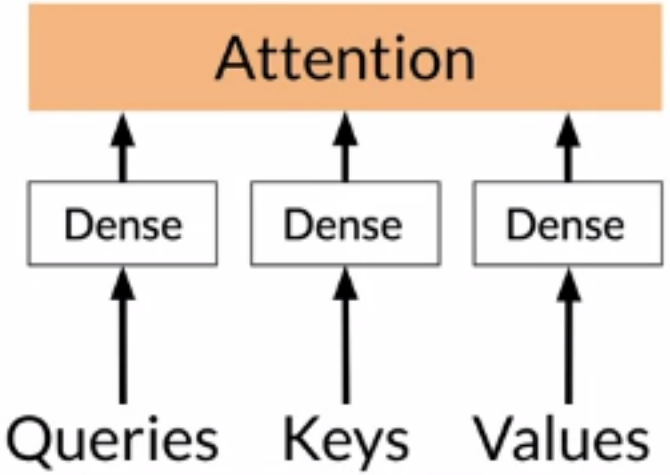


RNNs vs Transformer: Multi-headed attention

RNNs vs Transformer: Multi-headed attention

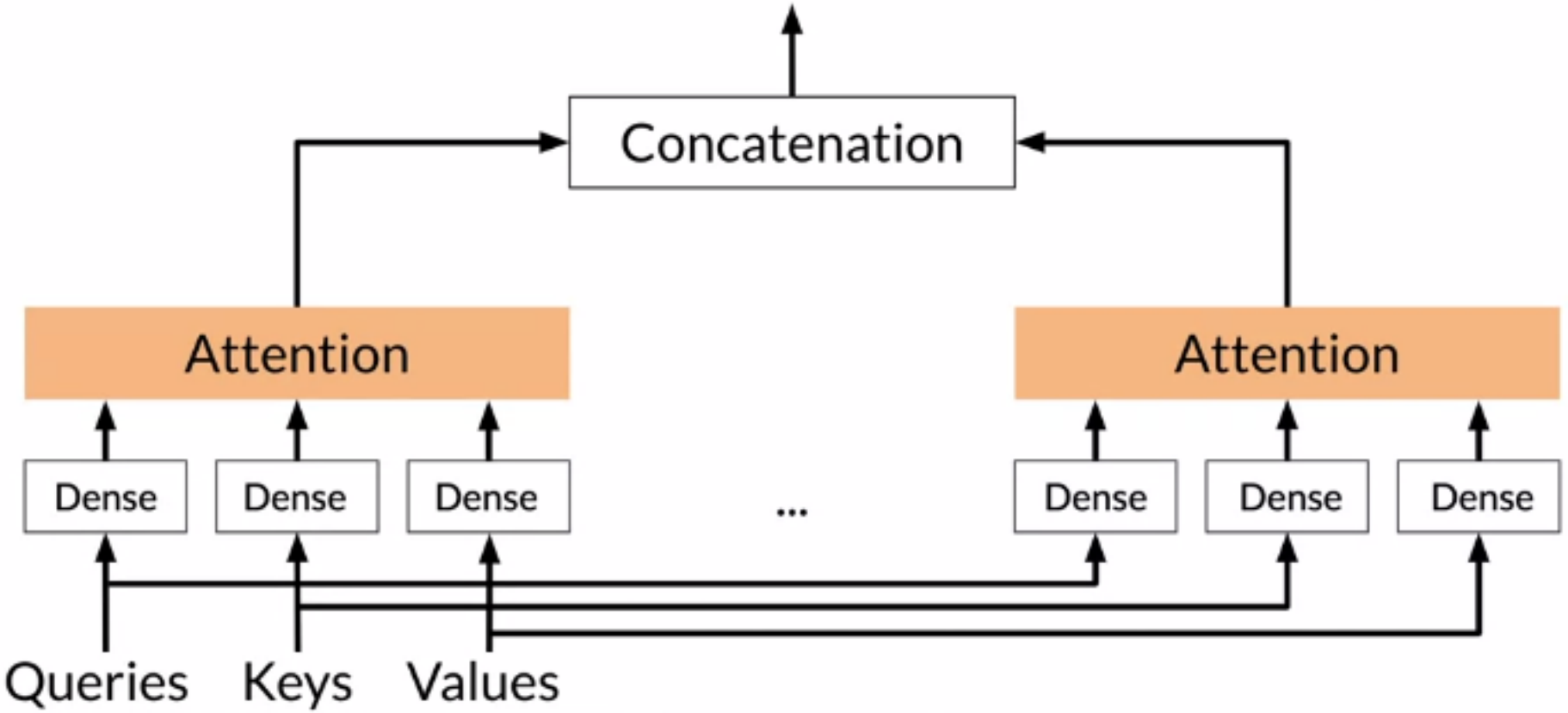


RNNs vs Transformer: Multi-headed attention



RNNs vs Transformer: Multi-headed attention

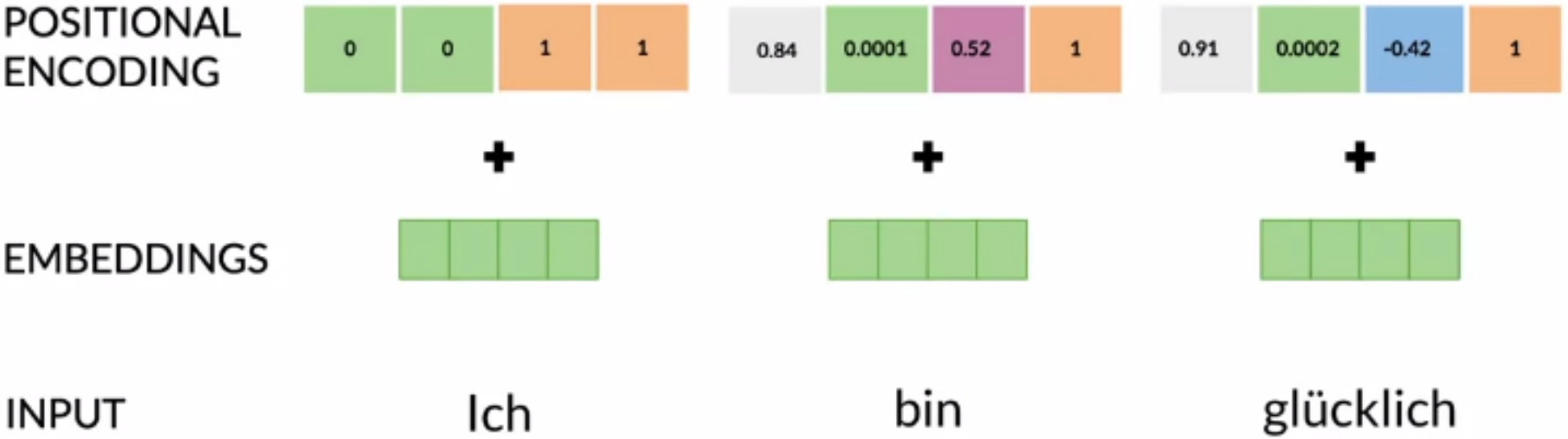
RNNs vs Transformer: Multi-headed attention



RNNs vs Transformer: Positional Encoding

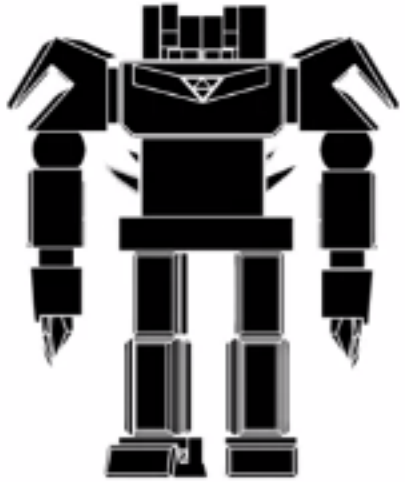
RNNs vs Transformer: Positional Encoding

RNNs vs Transformer: Positional Encoding



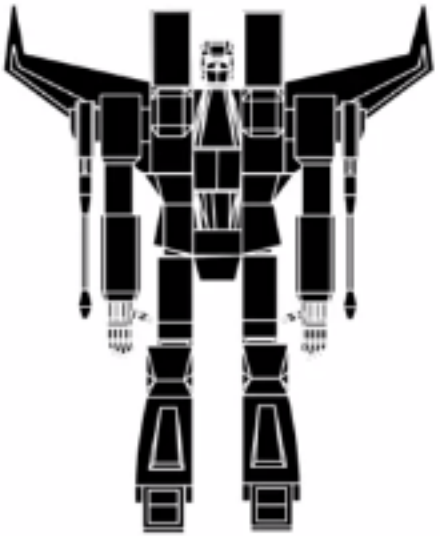
Summary

- In RNNs parallel computing is difficult to implement
- For long sequences in RNNs there is loss of information
- In RNNs there is the problem of vanishing gradient
- Transformers help with all of the above



Outline

- Transformers applications in NLP
- Some Transformers
- Introduction to T5



Transformer NLP applications

Transformer NLP applications

Text summarization

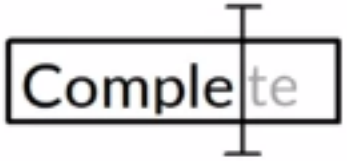
Transformer NLP applications

Text summarization

Auto-Complete

Transformer NLP applications

Text summarization

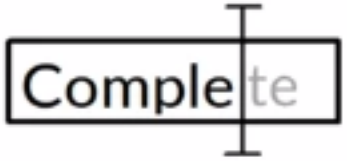


Auto-Complete

Named entity recognition (NER)

Transformer NLP applications

Text summarization



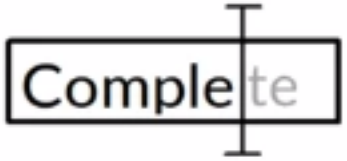
Auto-Complete

Named entity recognition (NER)

Question answering (Q&A)

Transformer NLP applications

Text summarization



Auto-Complete

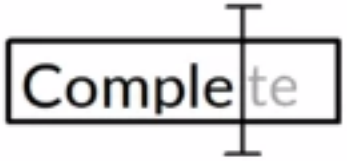
Named entity recognition (NER)

Question answering (Q&A)

Translation

Transformer NLP applications

Text summarization



Auto-Complete

Named entity recognition (NER)

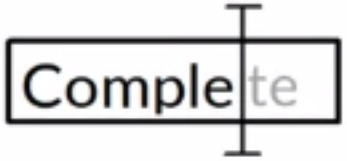
Question answering (Q&A)

Translation

Chat-bots

Transformer NLP applications

Text summarization



Auto-Complete

Named entity recognition (NER)

Question answering (Q&A)

Translation

Chat-bots

Other NLP tasks

- Sentiment Analysis
- Market Intelligence
- Text Classification
- Character Recognition
- Spell Checking

State of the Art Transformers

Radford, A., et al. (2018)
Open AI

GPT-2: Generative Pre-training for Transformer

State of the Art Transformers

Radford, A., et al. (2018)
Open AI

GPT-2: Generative Pre-training for Transformer

Devlin, J., et al. (2018)
Google AI Language

BERT:Bidirectional Encoder Representations from Transformers

State of the Art Transformers

Radford, A., et al. (2018)
Open AI

GPT-2: Generative Pre-training for Transformer

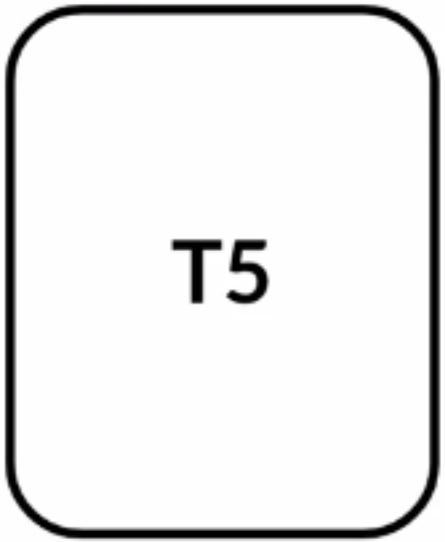
Devlin, J., et al. (2018)
Google AI Language

BERT:Bidirectional Encoder Representations from Transformers

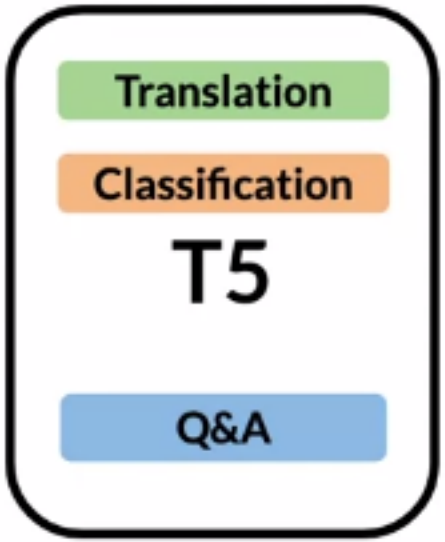
Colin, R., et al. (2019)
Google

T5: Text-to-text transfer transformer

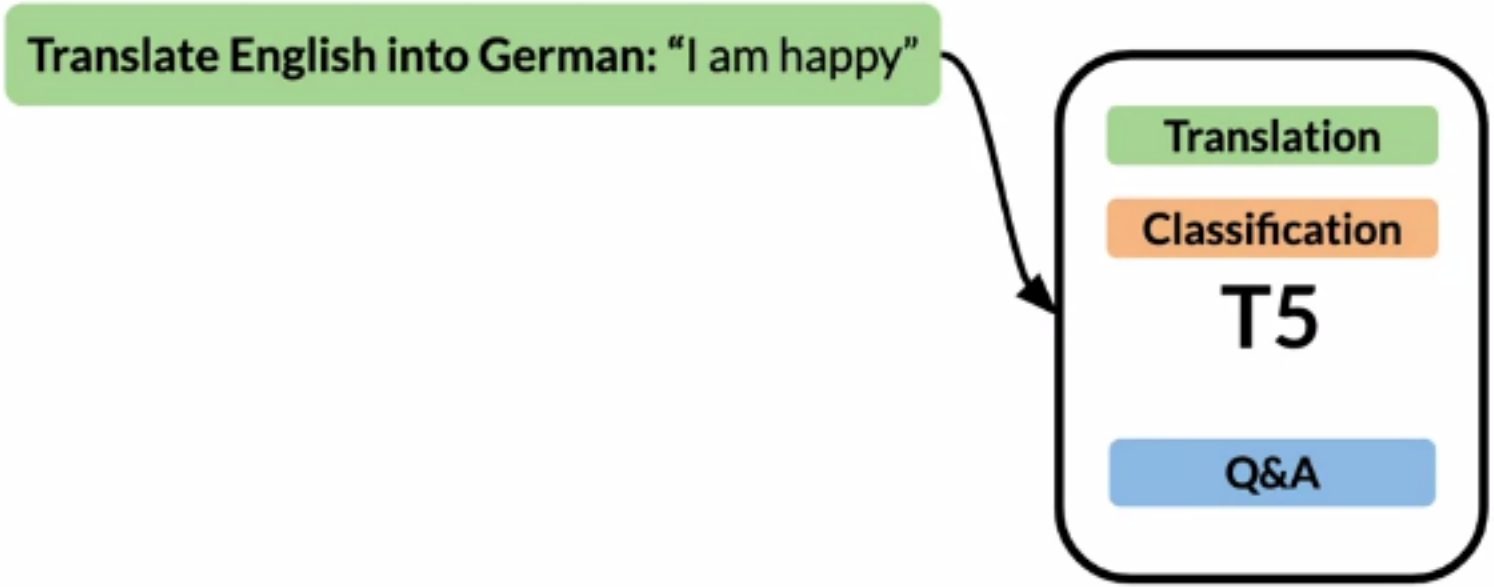
T5: Text-To-Text Transfer Transformer



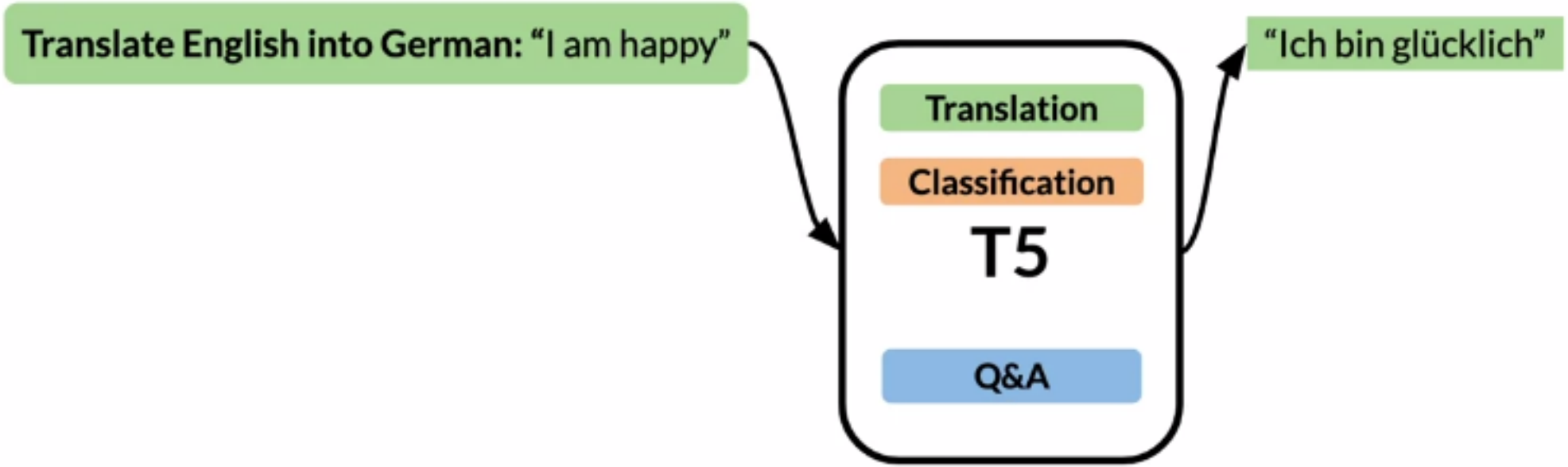
T5: Text-To-Text Transfer Transformer



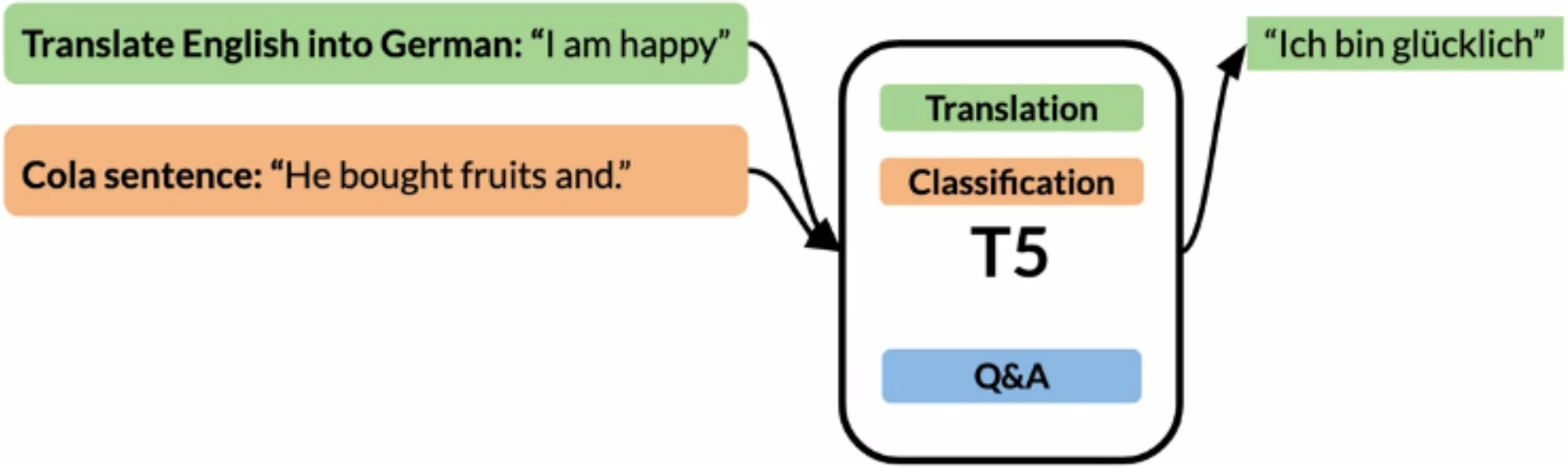
T5: Text-To-Text Transfer Transformer



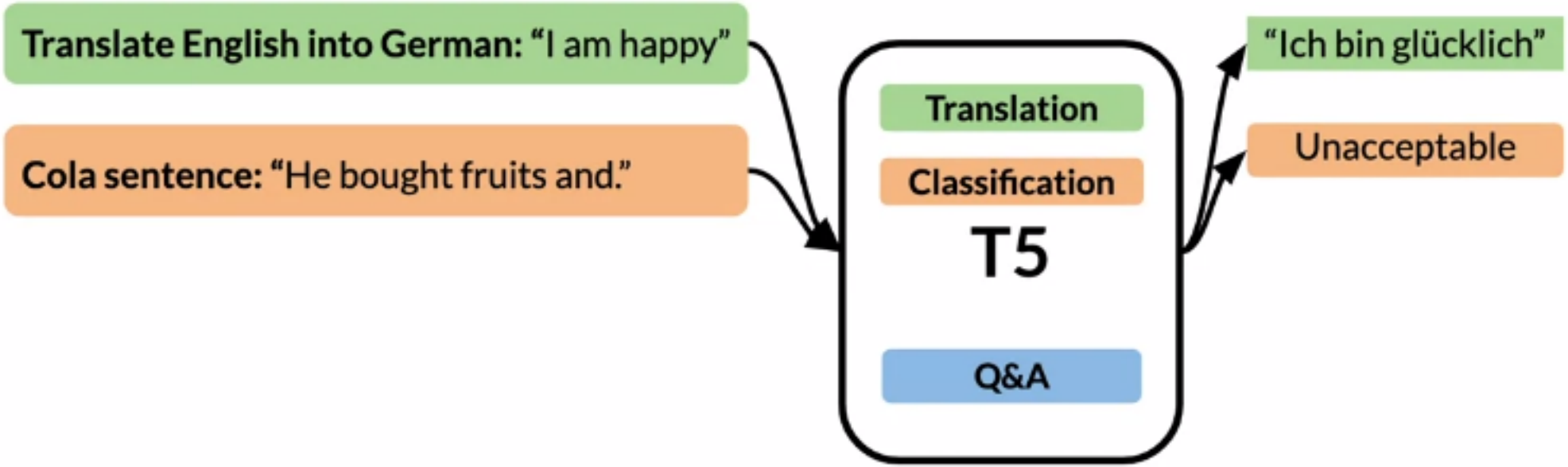
T5: Text-To-Text Transfer Transformer



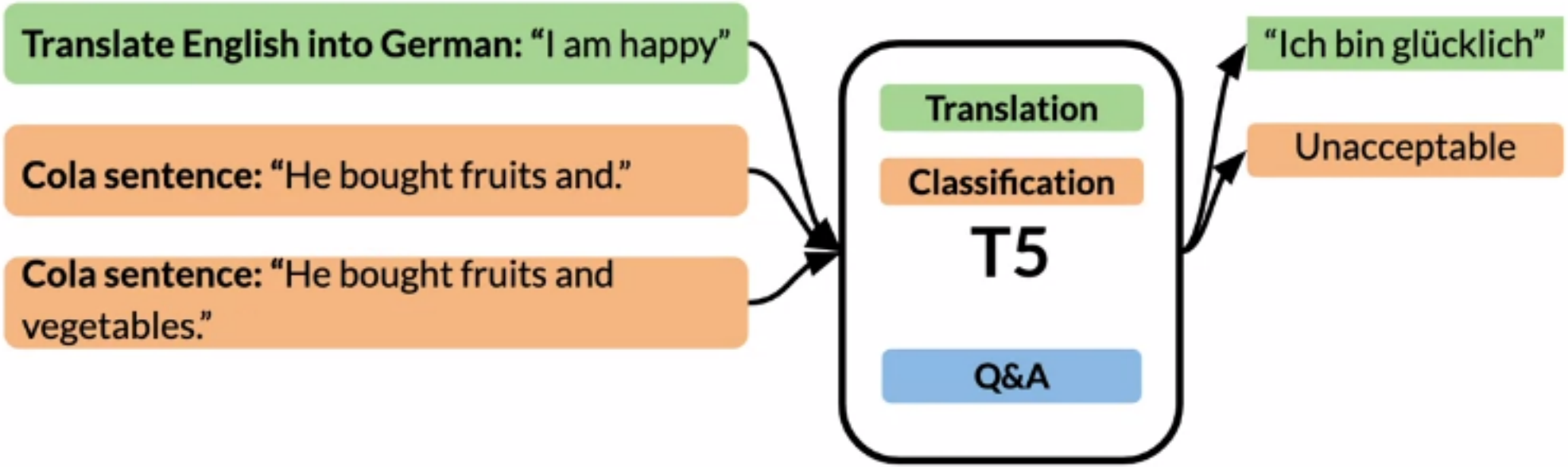
T5: Text-To-Text Transfer Transformer



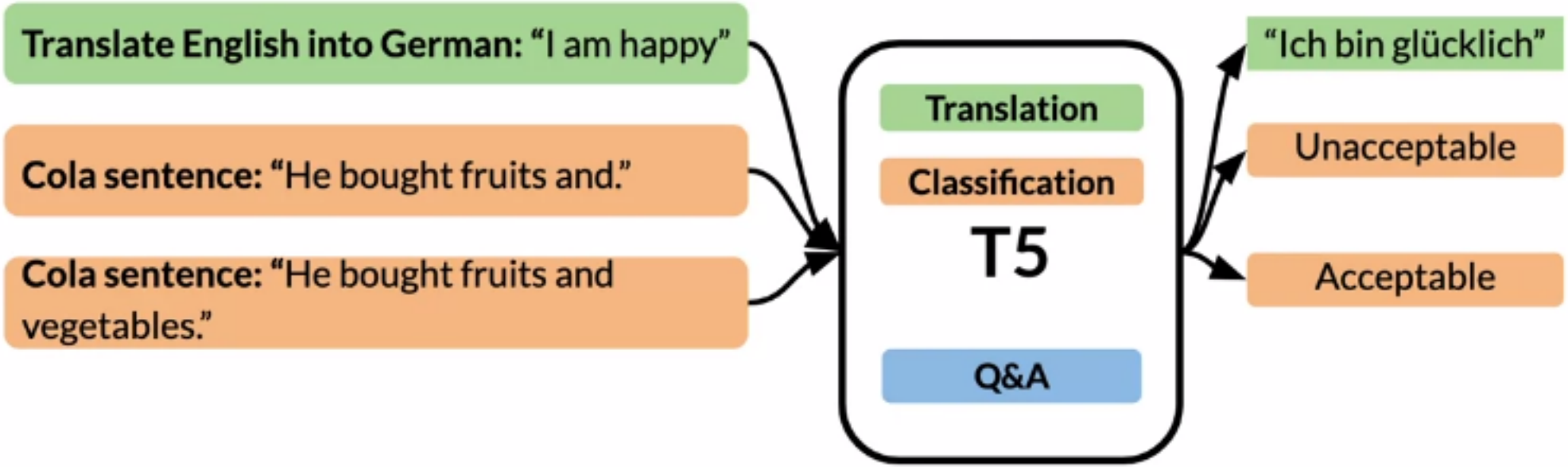
T5: Text-To-Text Transfer Transformer



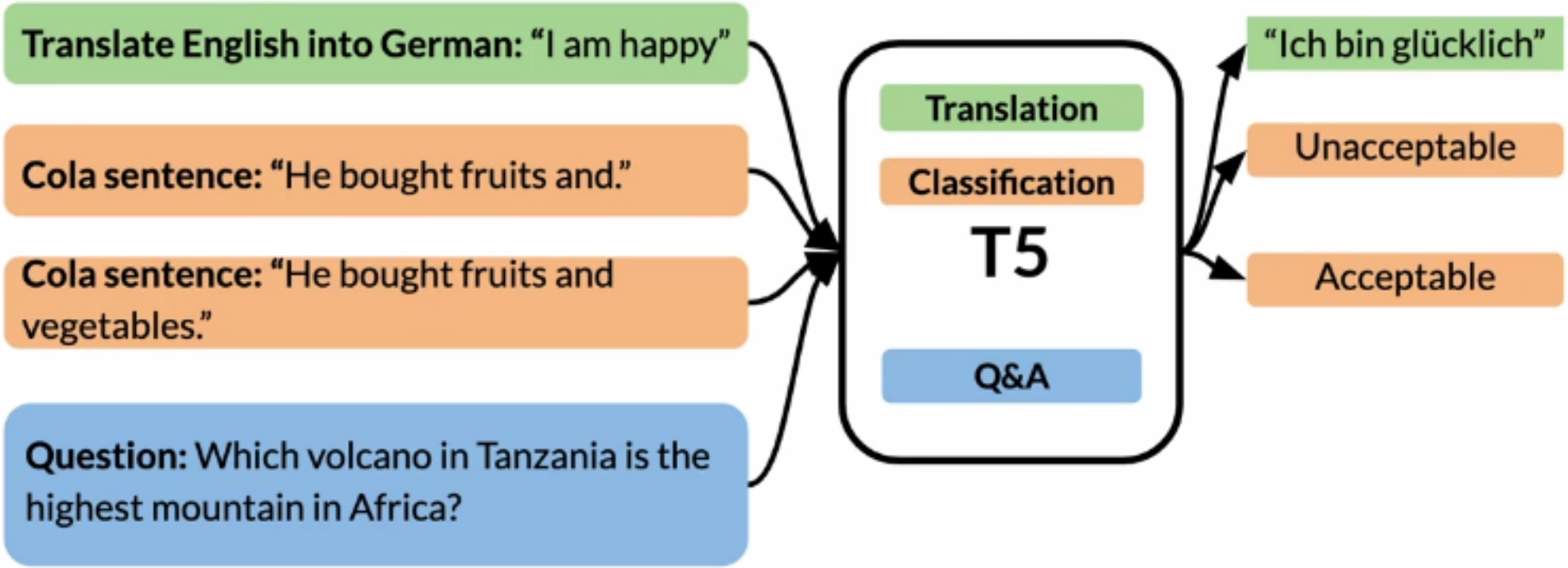
T5: Text-To-Text Transfer Transformer



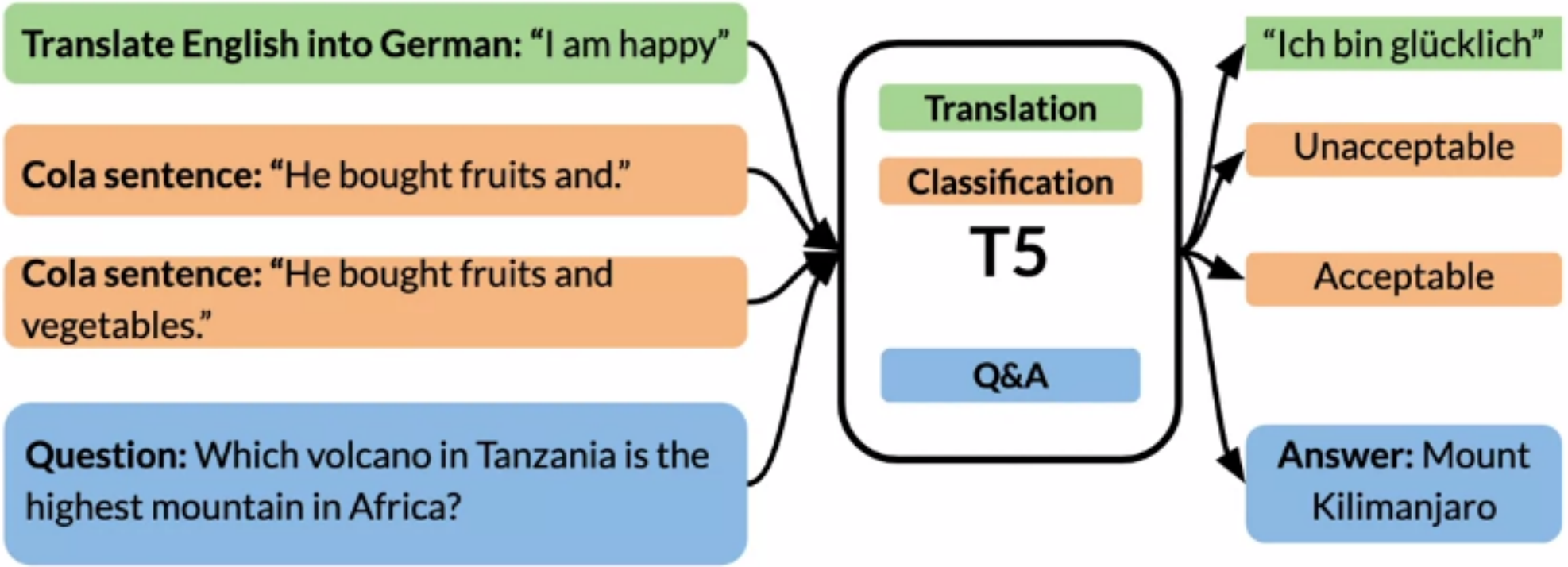
T5: Text-To-Text Transfer Transformer



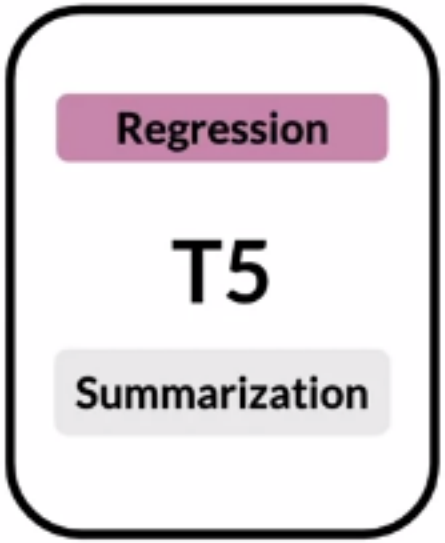
T5: Text-To-Text Transfer Transformer



T5: Text-To-Text Transfer Transformer

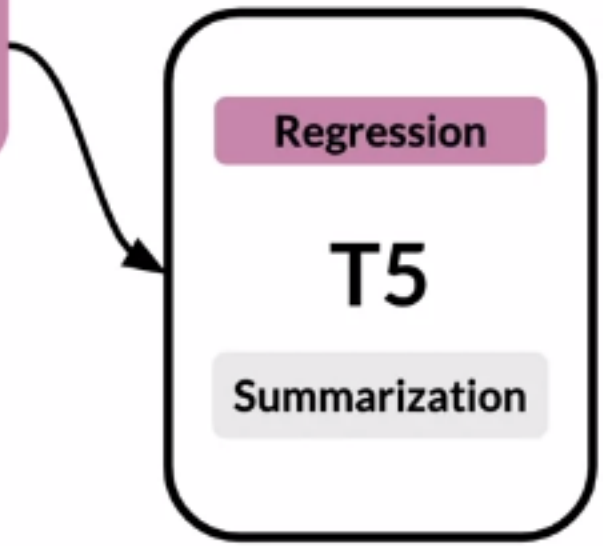


T5: Text-To-Text Transfer Transformer

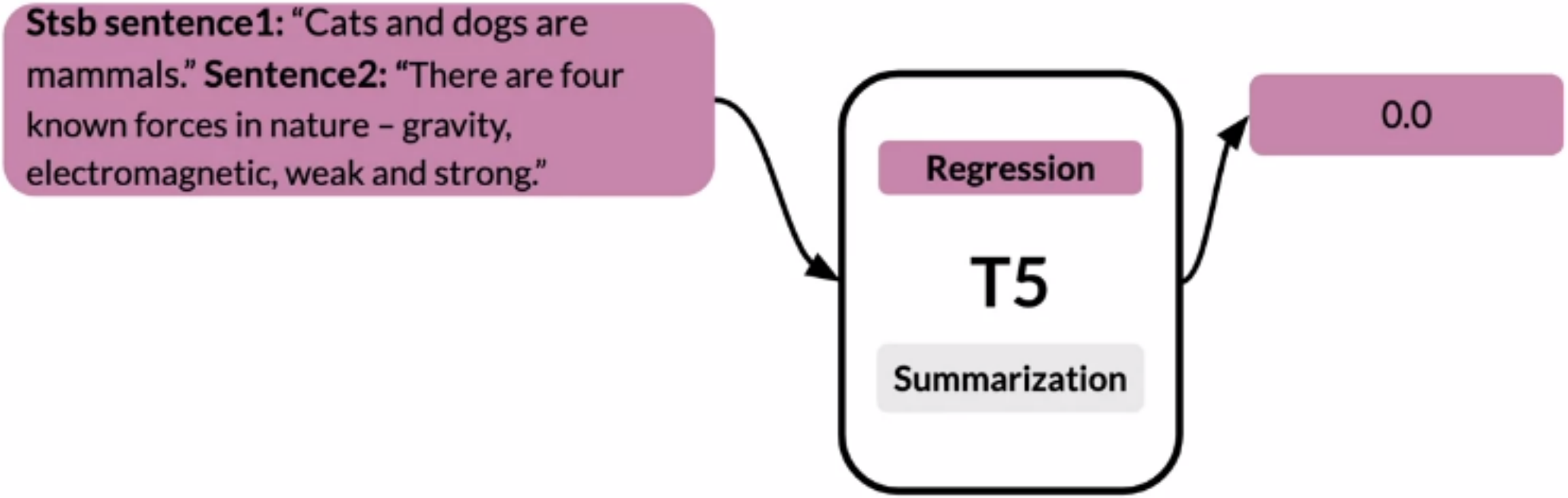


T5: Text-To-Text Transfer Transformer

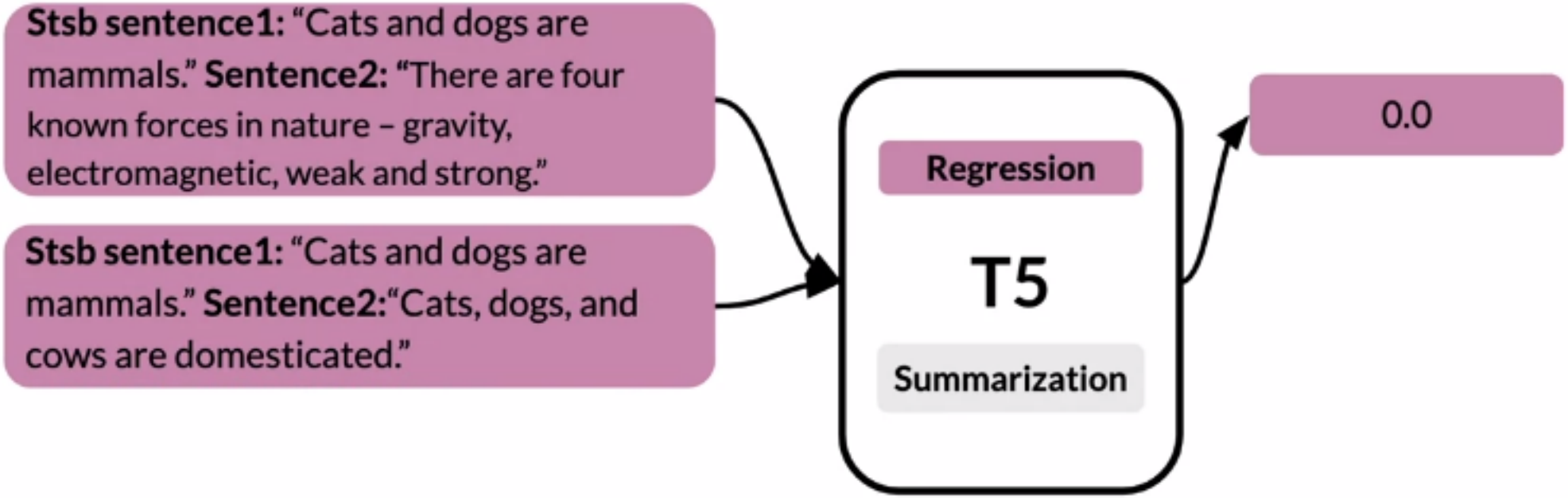
Stsb sentence1: "Cats and dogs are mammals." **Sentence2:** "There are four known forces in nature – gravity, electromagnetic, weak and strong."



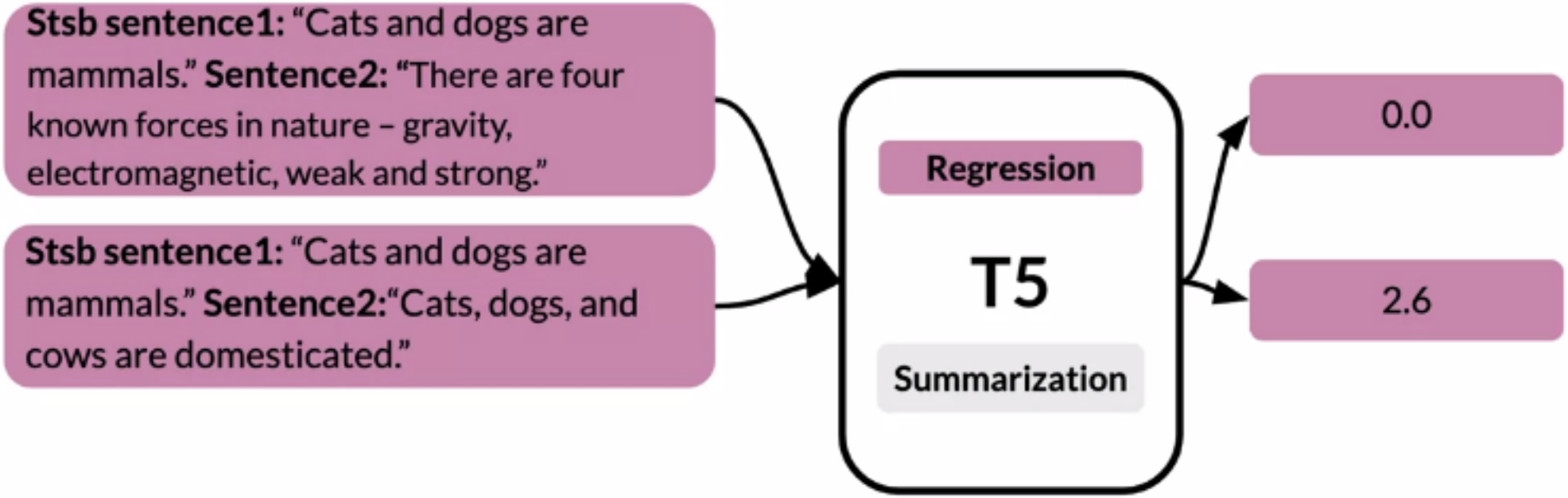
T5: Text-To-Text Transfer Transformer



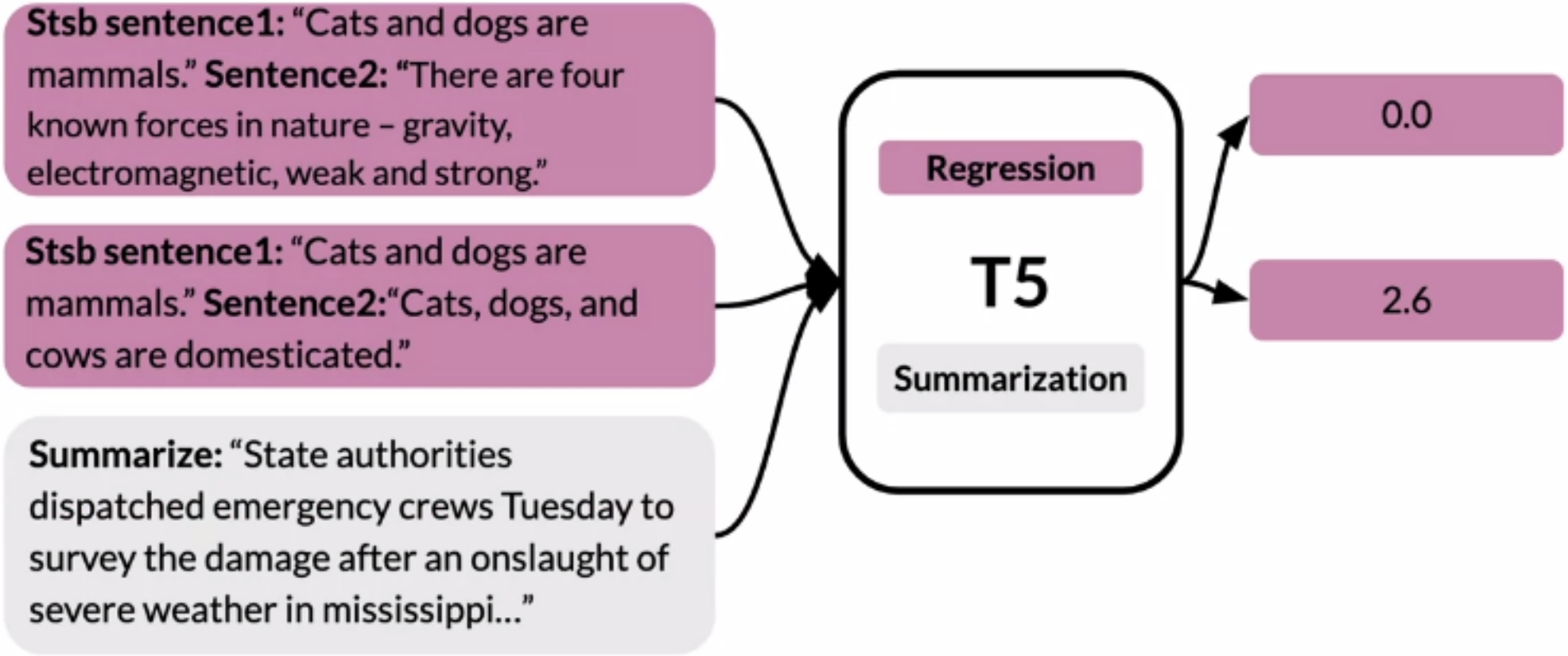
T5: Text-To-Text Transfer Transformer



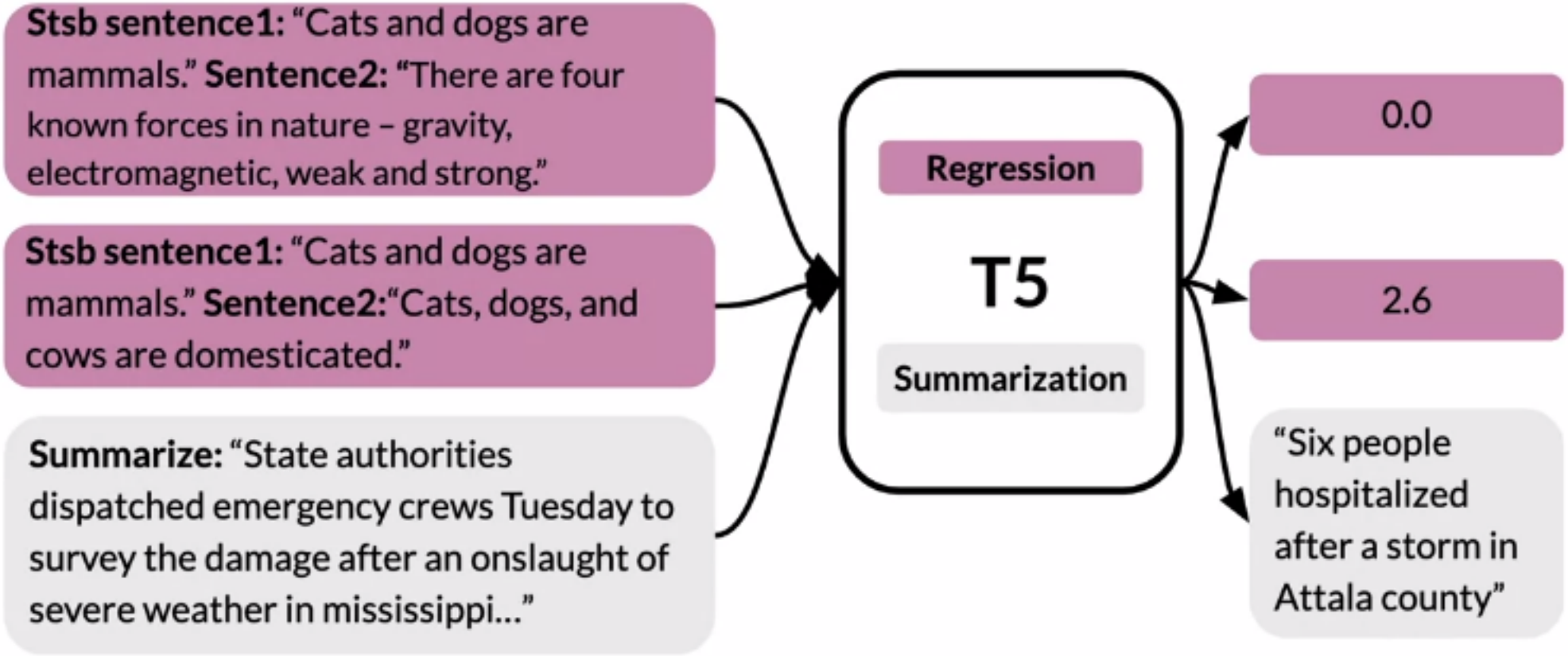
T5: Text-To-Text Transfer Transformer



T5: Text-To-Text Transfer Transformer



T5: Text-To-Text Transfer Transformer



T5 Demo


question 2
What colour hair did Charles Dickens' character David Copperfield have?
You
P
SUBMIT
1
T5
LOCKED IN!
1

T5 Demo


question 1
What country is the largest oil producer in Africa?
You
N
SUBMIT
T5
LOCKED IN!
0:05 / 0:26

T5 Demo

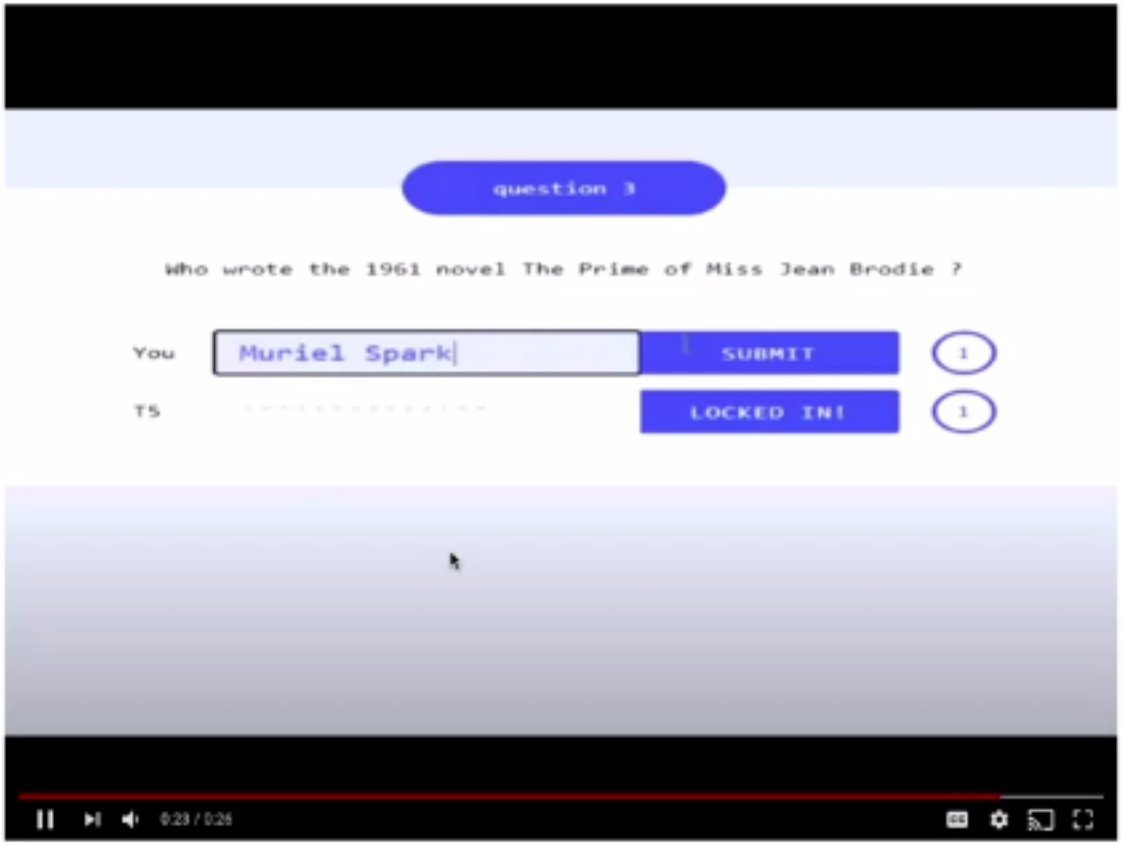

question 1
What country is the largest oil producer in Africa?
You
Nigeria
CORRECT!
1
T5
Nigeria
CORRECT!
1
NEXT QUESTION
Correct answer: nigeria
See all accepted answers
0:09 / 0:26

T5 Demo

T5 Demo

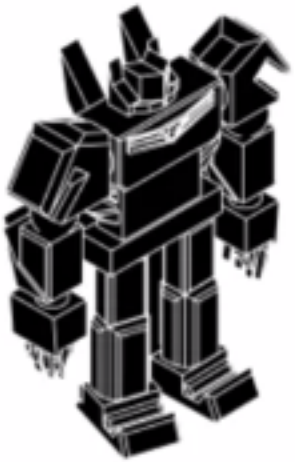

question 3
Who wrote the 1961 novel The Prime of Miss Jean Brodie ?
You
SUBMIT
1
T5
1
0:16 / 0:26

T5 Demo


question 3
Who wrote the 1961 novel The Prime of Miss Jean Brodie ?
You
Muriel Spark
SUBMIT
1
T5
LOCKED IN!
1
0:23 / 0:26

Summary

- Transformers are suitable for a wide range of NLP applications
- GPT-2, BERT and T5 are the cutting-edge Transformers
- T5 is a powerful multi-task transformer



Outline

- Introducing attention (Translation example)
- Mathematics behind Attention

Introducing attention - Translation example

Introducing attention - Translation example

- A query (German word) looks for similar keys (English words).

I am happy

Introducing attention - Translation example

- A query (German word) looks for similar keys (English words).

I am happy

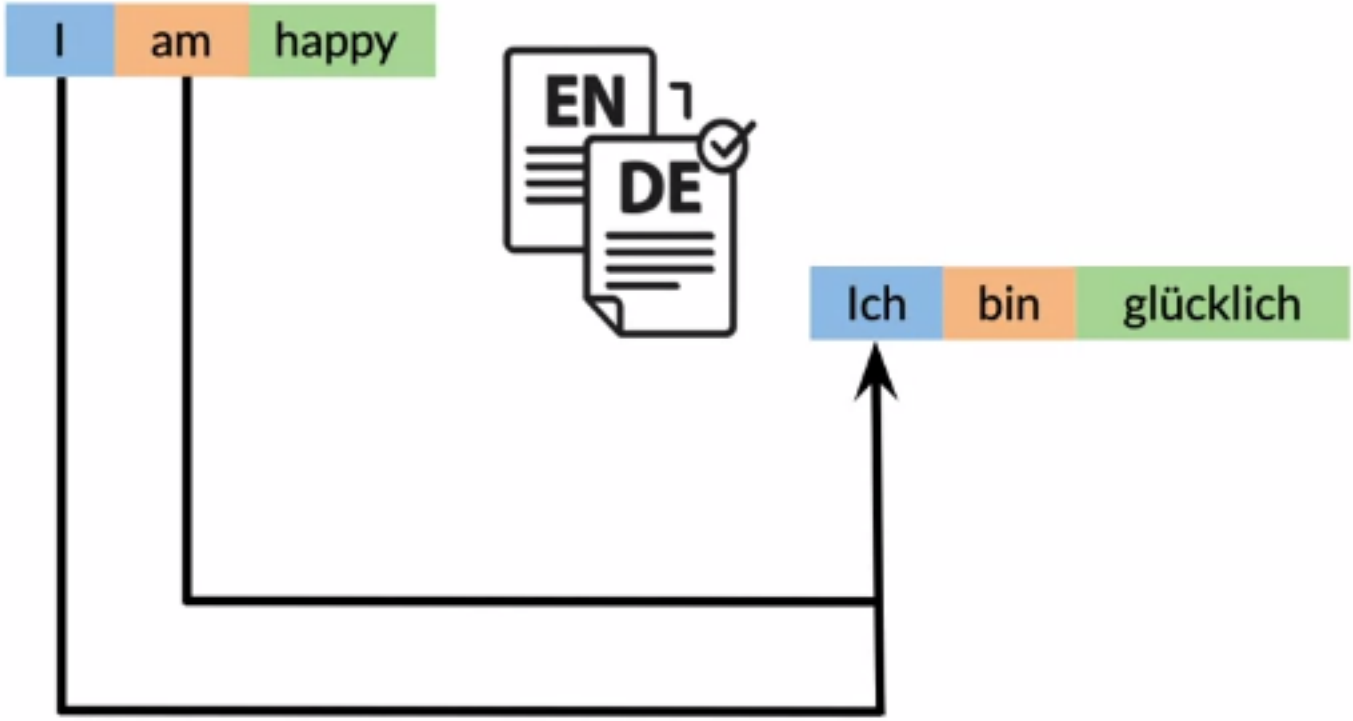
Ich bin glücklich

Introducing attention - Translation example

- A query (German word) looks for similar keys (English words).

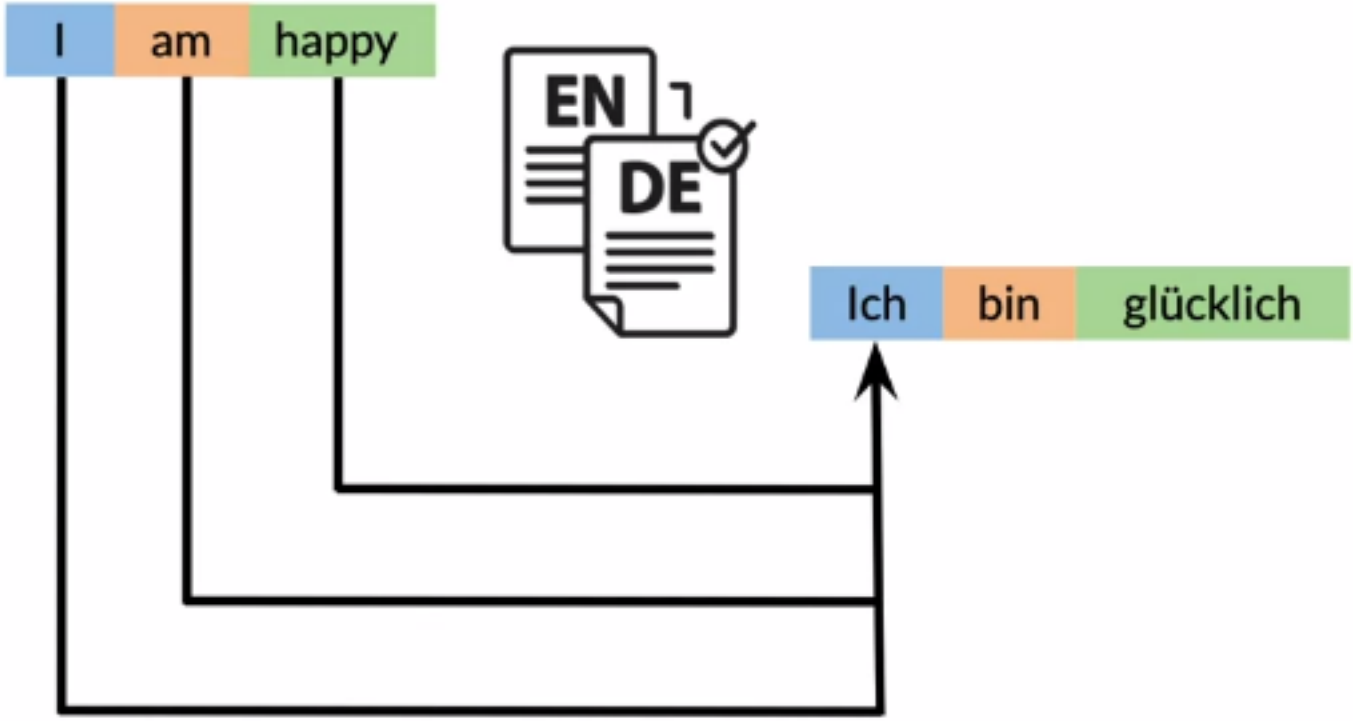
Introducing attention - Translation example

- A query (German word) looks for similar keys (English words).
- Each key has a probability of being a match for the query.



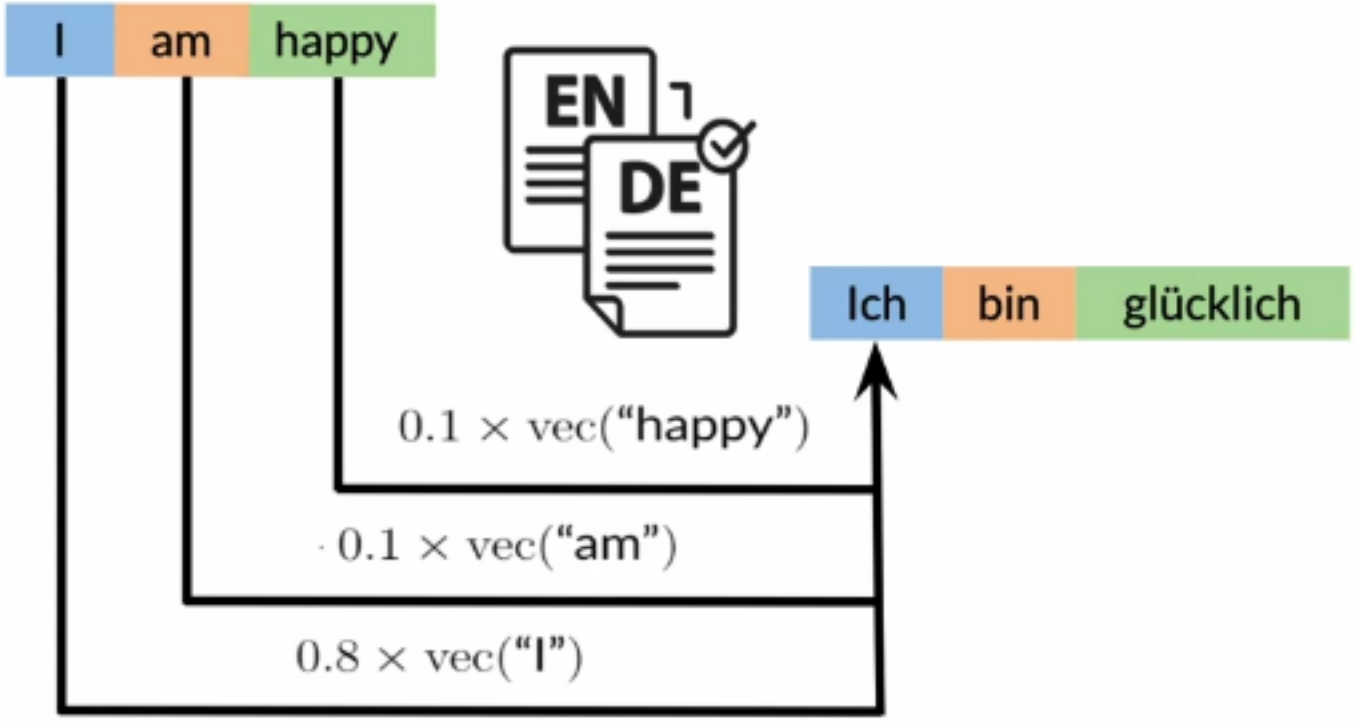
Introducing attention - Translation example

- A query (German word) looks for similar keys (English words).
- Each key has a probability of being a match for the query.



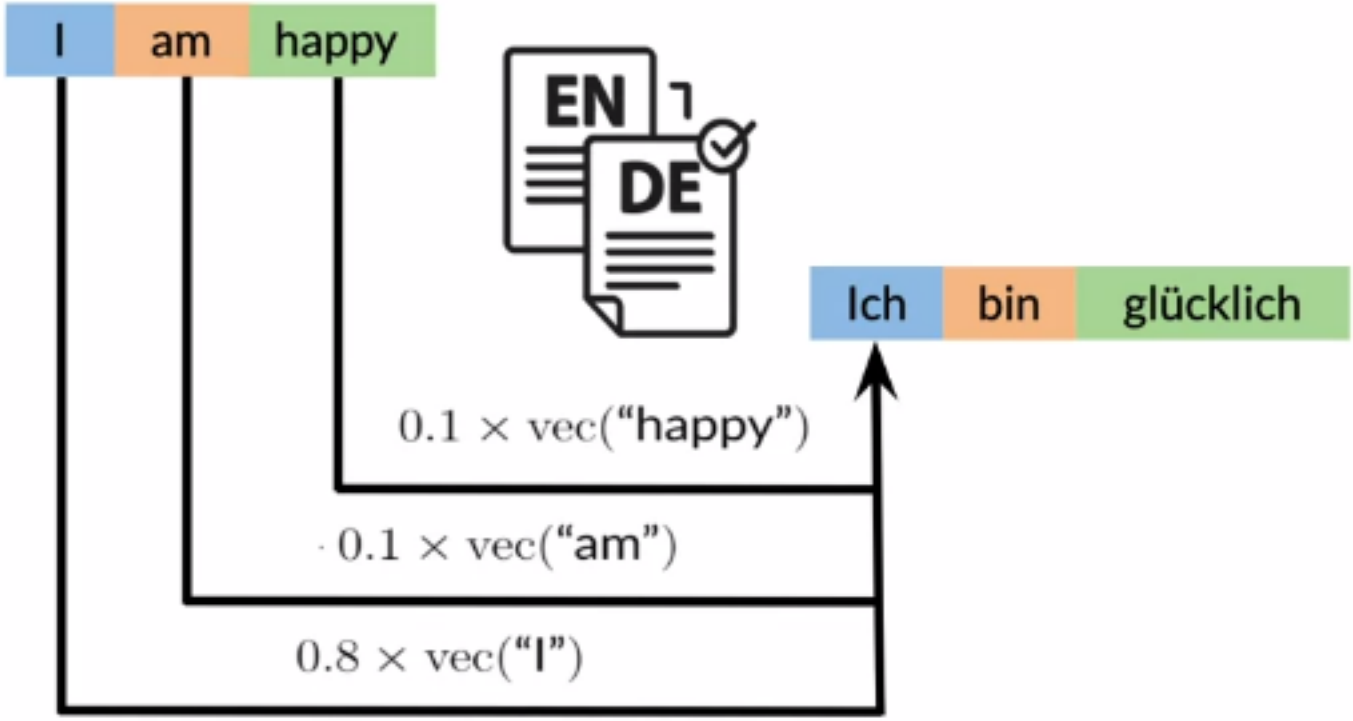
Introducing attention - Translation example

- A query (German word) looks for similar keys (English words).
- Each key has a probability of being a match for the query.



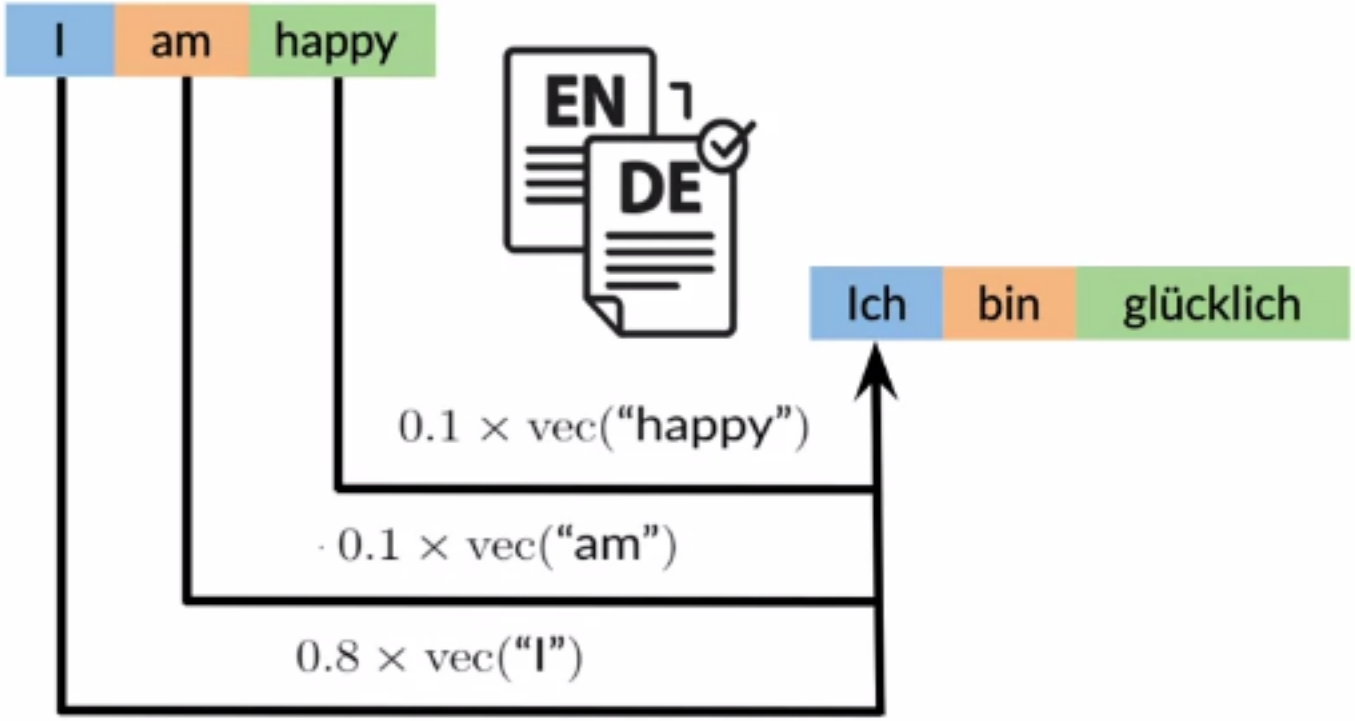
Introducing attention - Translation example

- A query (German word) looks for similar keys (English words).
- Each key has a probability of being a match for the query.
- Return sum of the keys weighted by their probabilities.



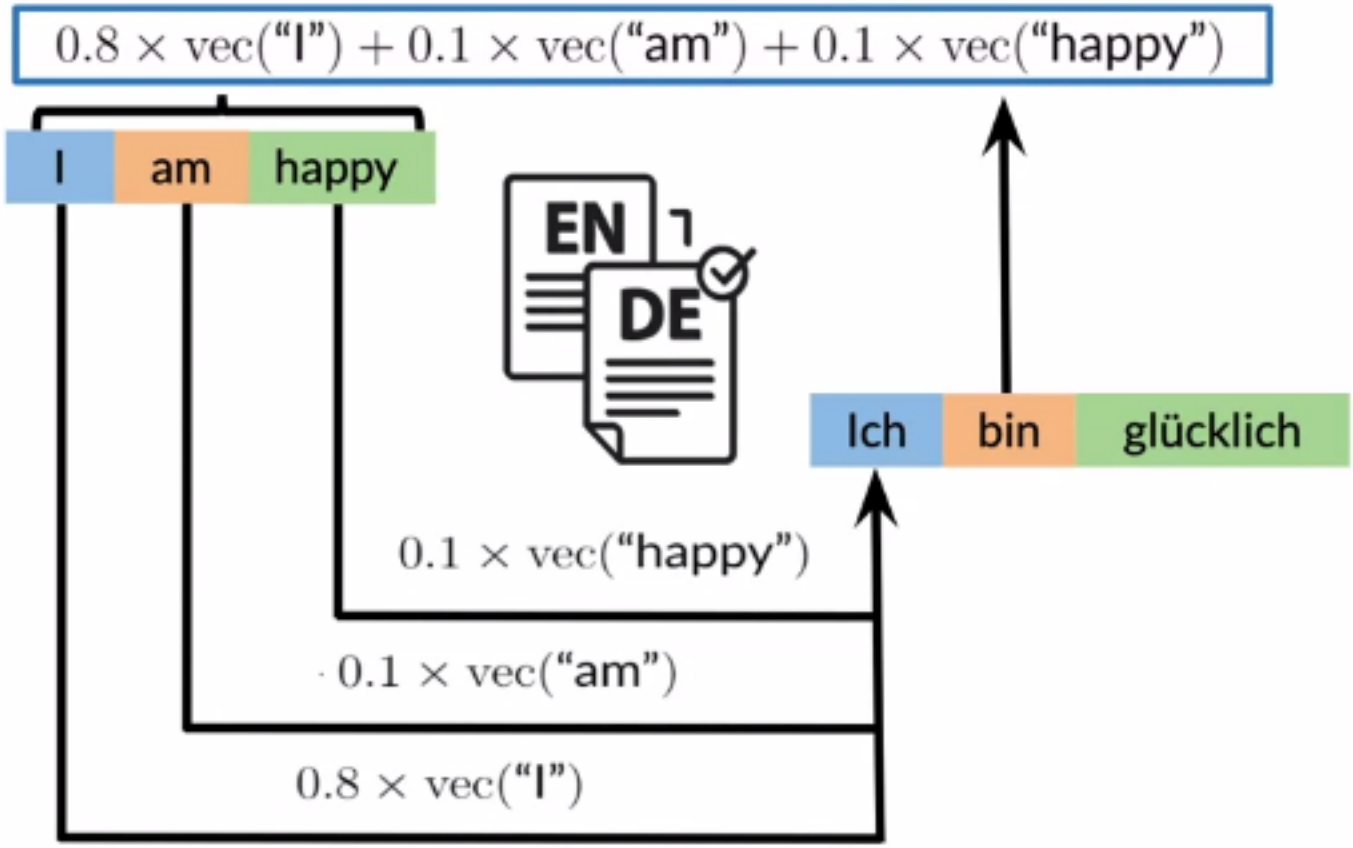
Introducing attention - Translation example

- A query (German word) looks for similar keys (English words).
- Each key has a probability of being a match for the query.
- Return sum of the keys weighted by their probabilities.



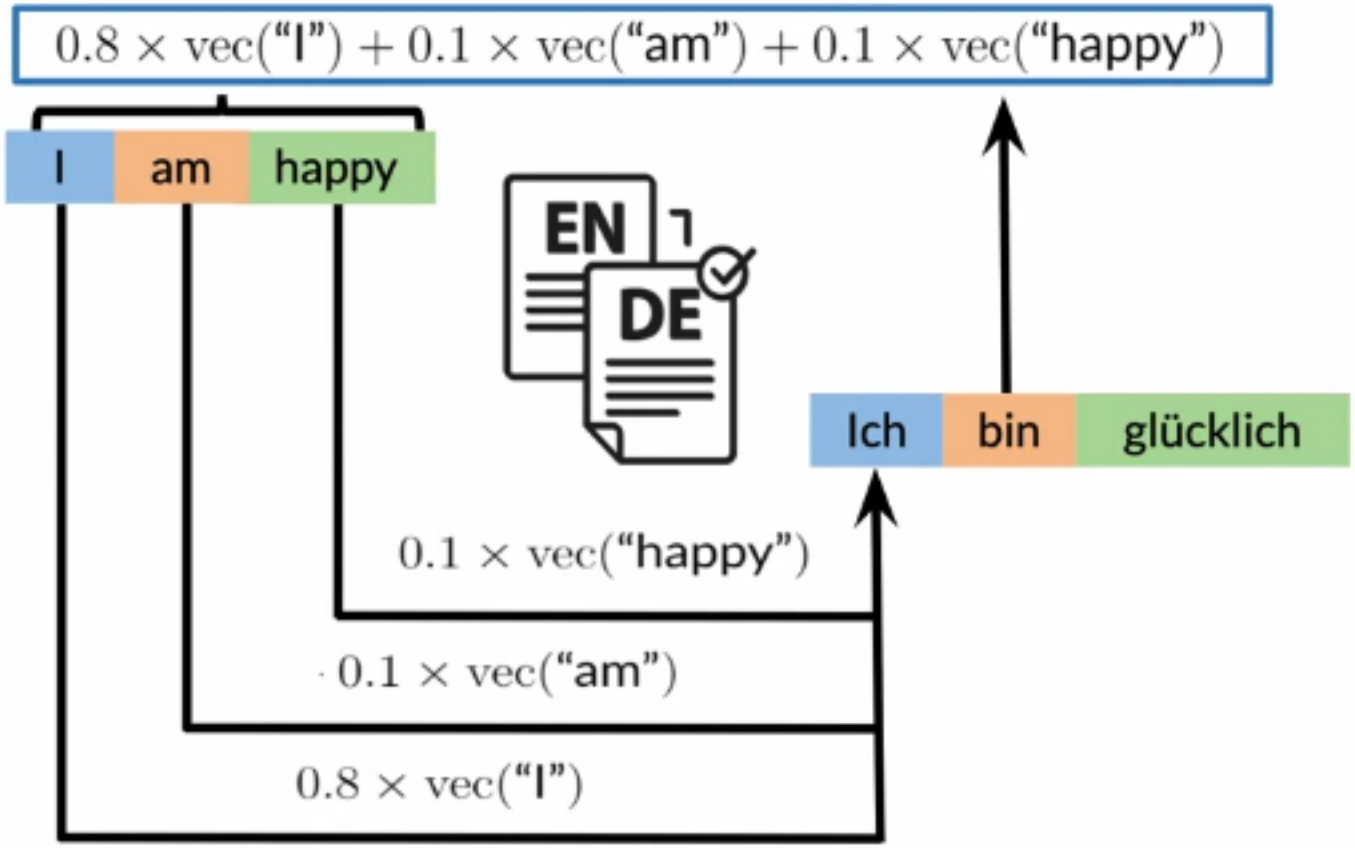
Introducing attention - Translation example

- A query (German word) looks for similar keys (English words).
- Each key has a probability of being a match for the query.
- Return sum of the keys weighted by their probabilities.



Introducing attention - Translation example

- A query (German word) looks for similar keys (English words).
- Each key has a probability of being a match for the query.
- Return sum of the keys weighted by their probabilities.



Queries, Keys and Values

Queries, Keys and Values

Q

K

V

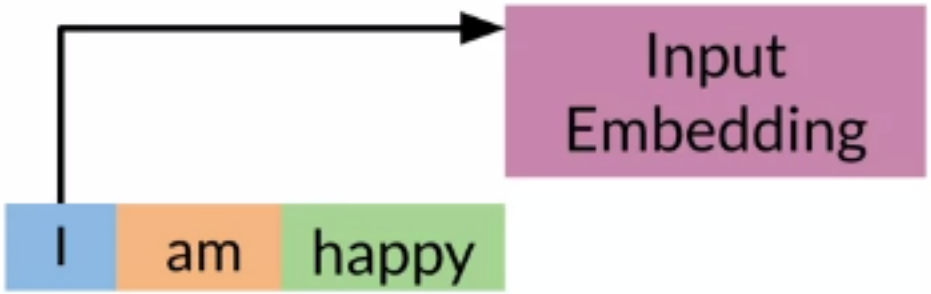
Queries, Keys and Values

Q

K

V

Queries, Keys and Values

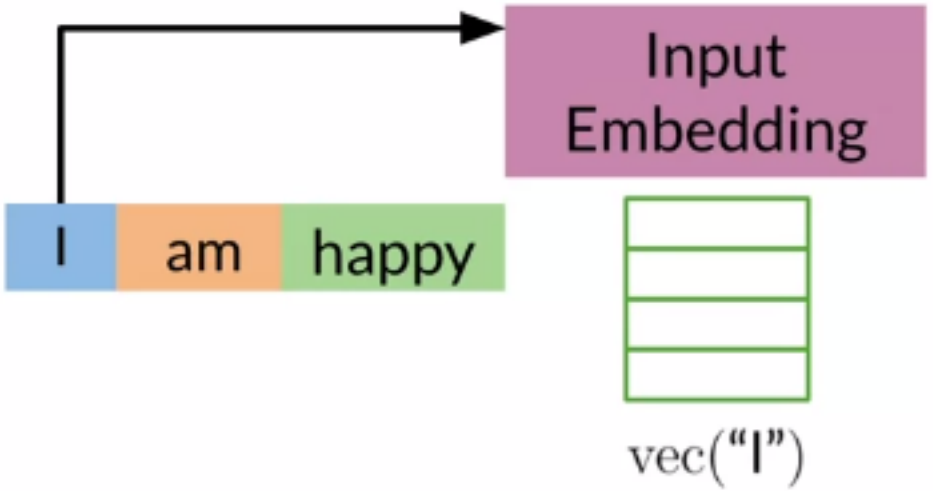


Q

K

V

Queries, Keys and Values

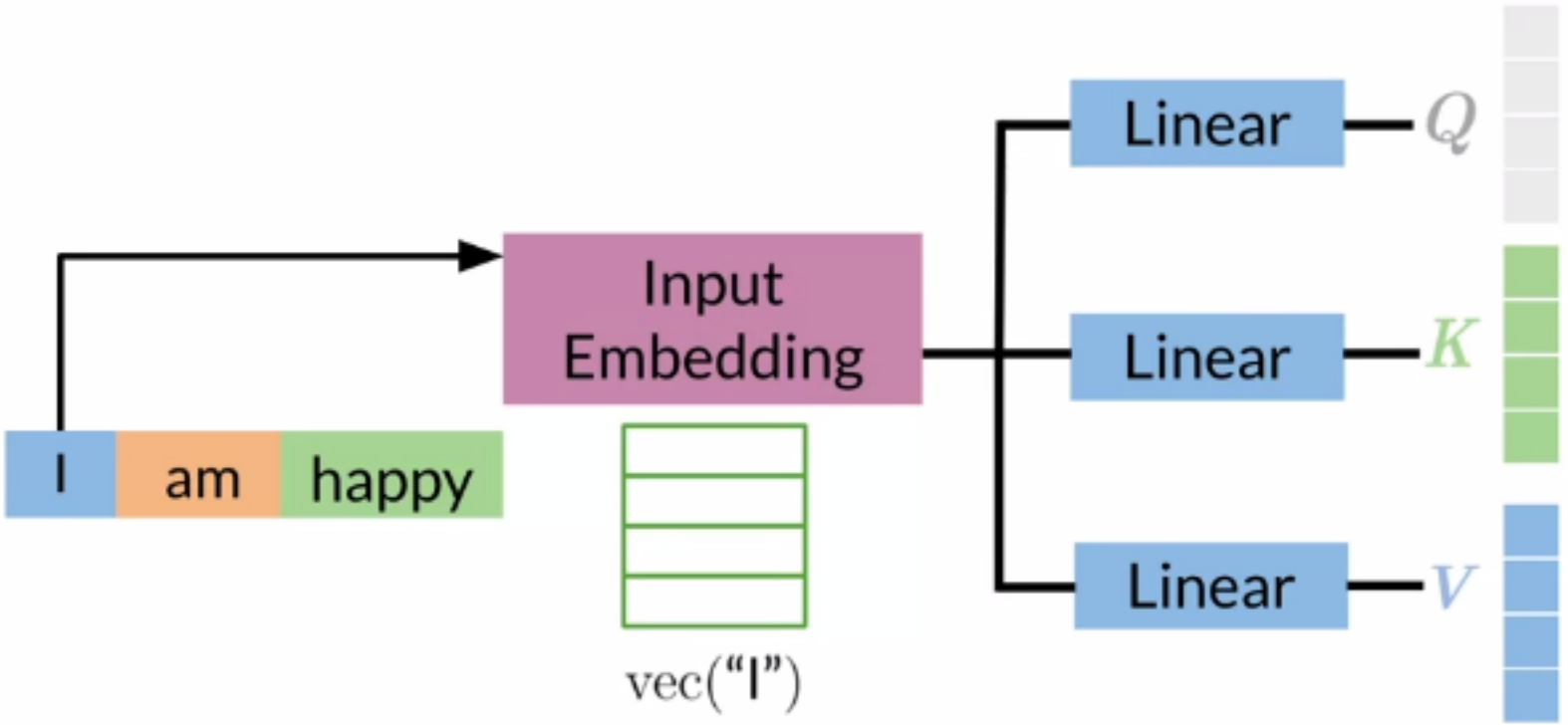


Q

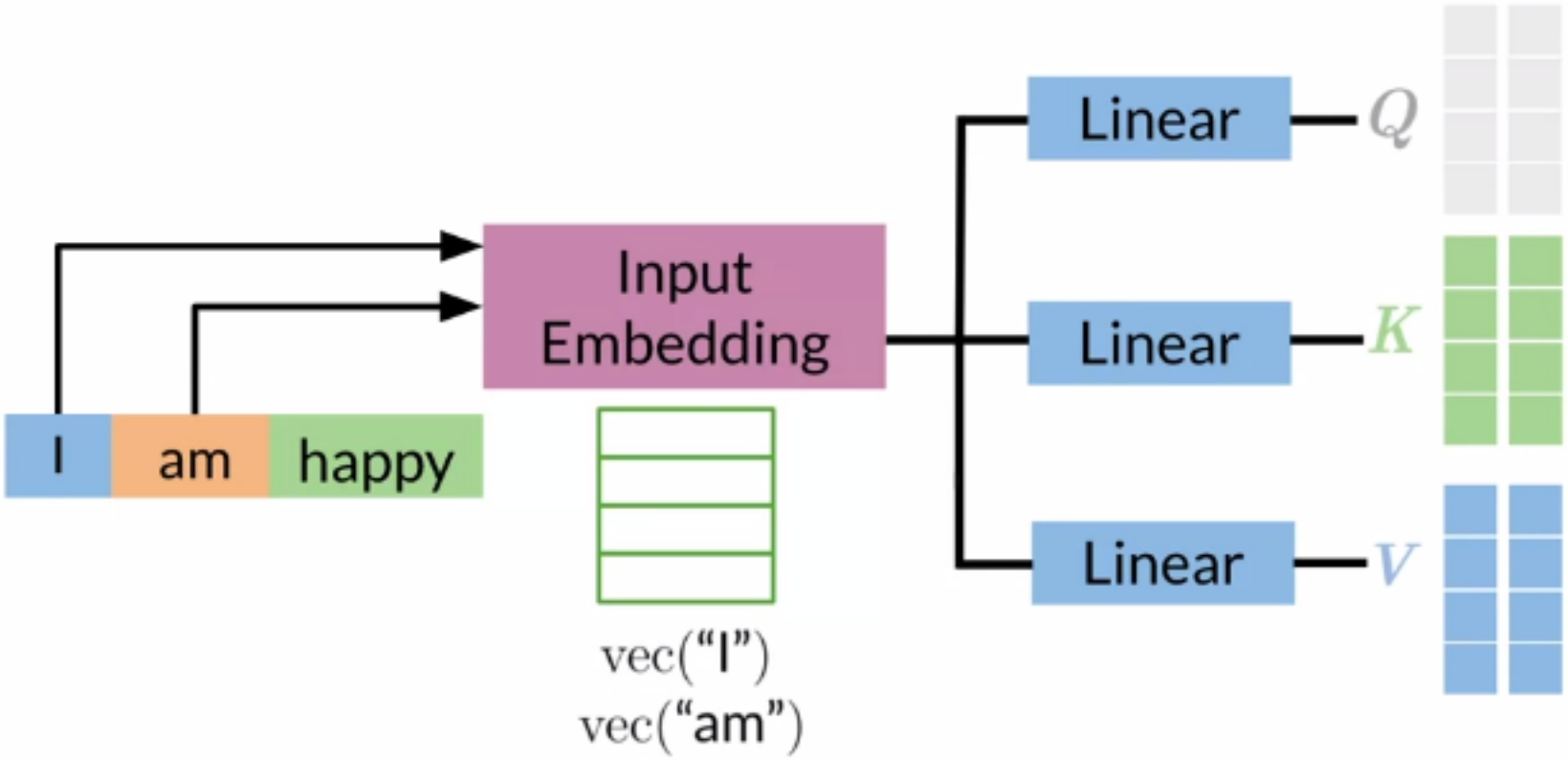
K

V

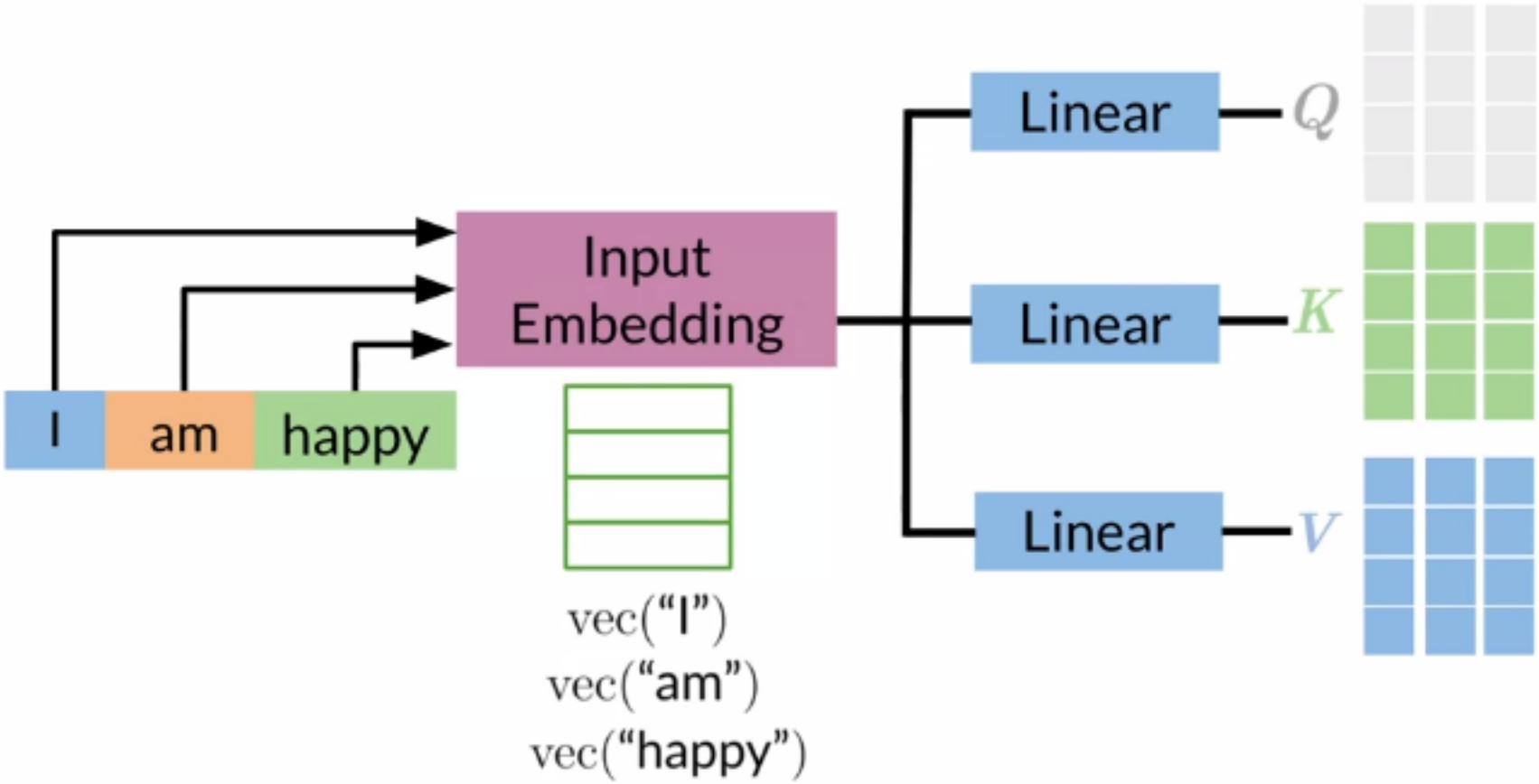
Queries, Keys and Values



Queries, Keys and Values



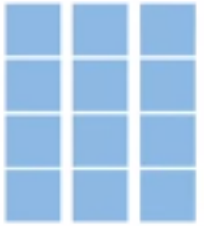
Queries, Keys and Values



Concept of attention

Q

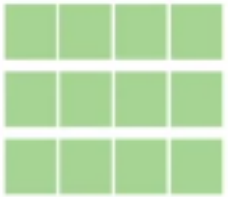
K



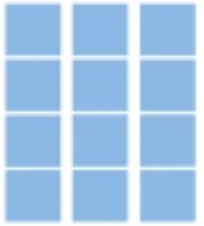
V

Concept of attention

Q

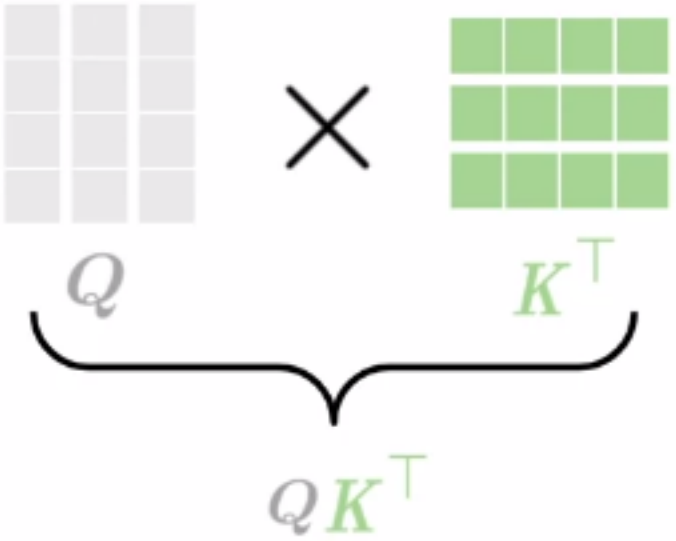


$K^\top$

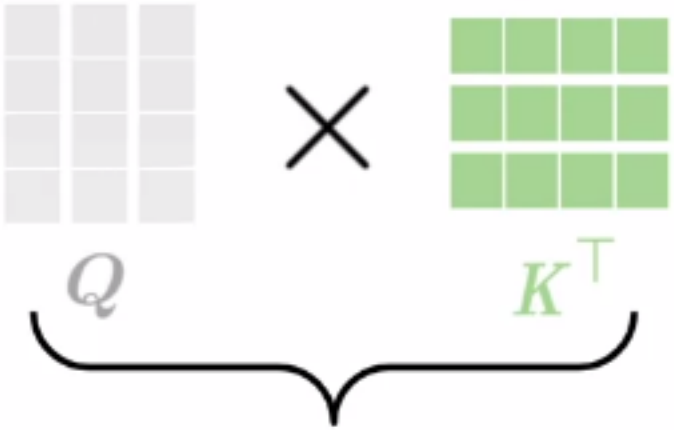


V

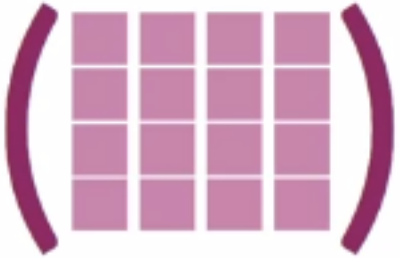
Concept of attention



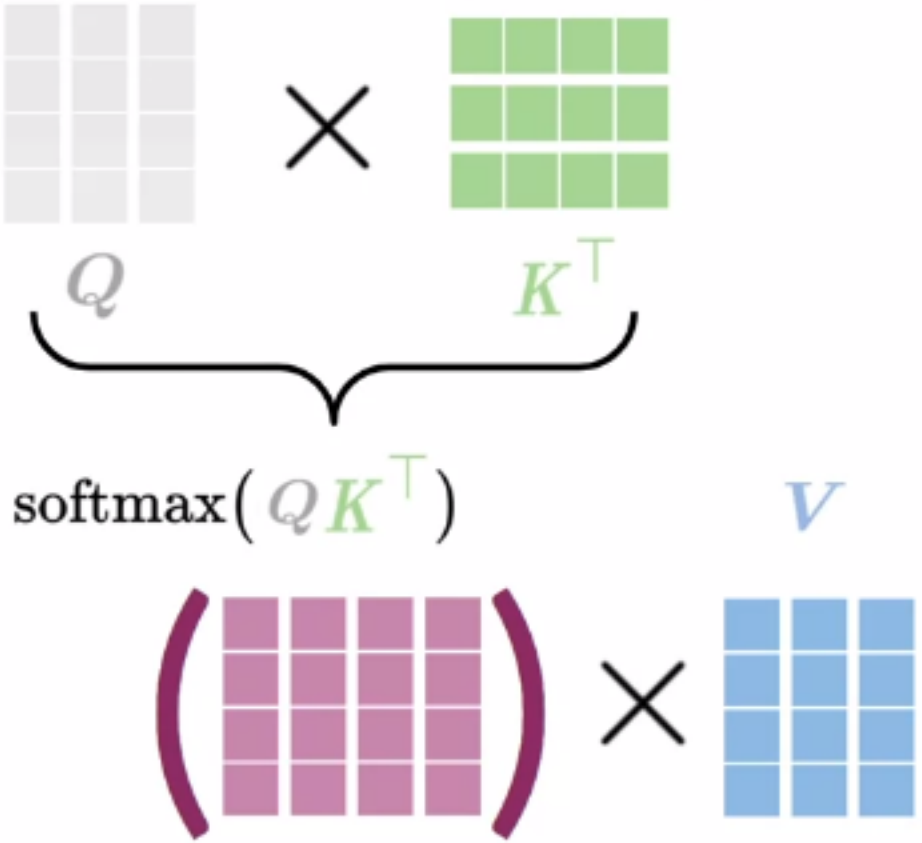
Concept of attention



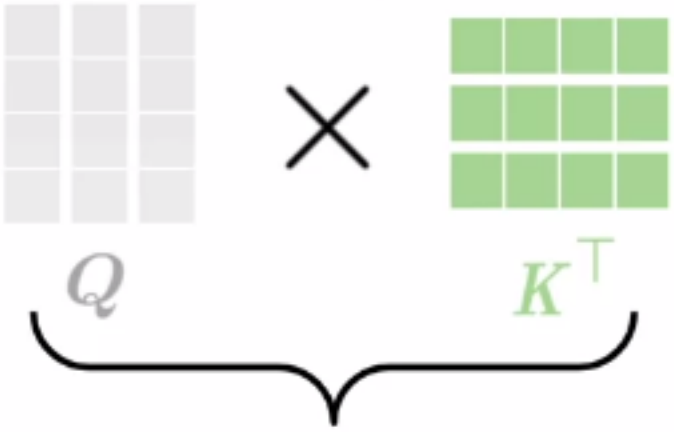
$$\text{softmax}(QK^\top)$$



Concept of attention



Concept of attention



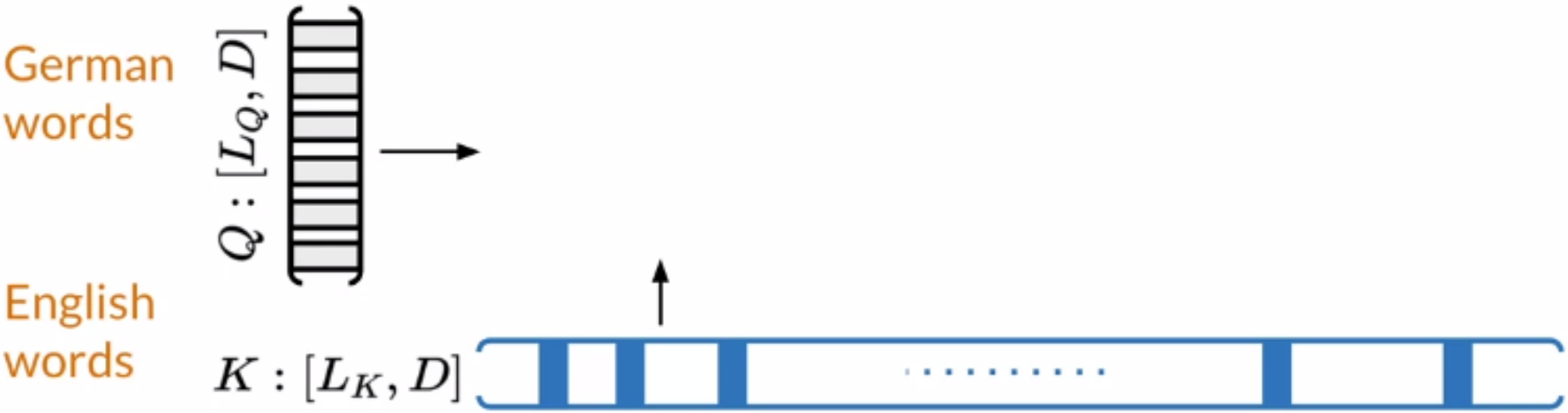
$$\text{softmax}(QK^\top)V = Z$$

Attention math

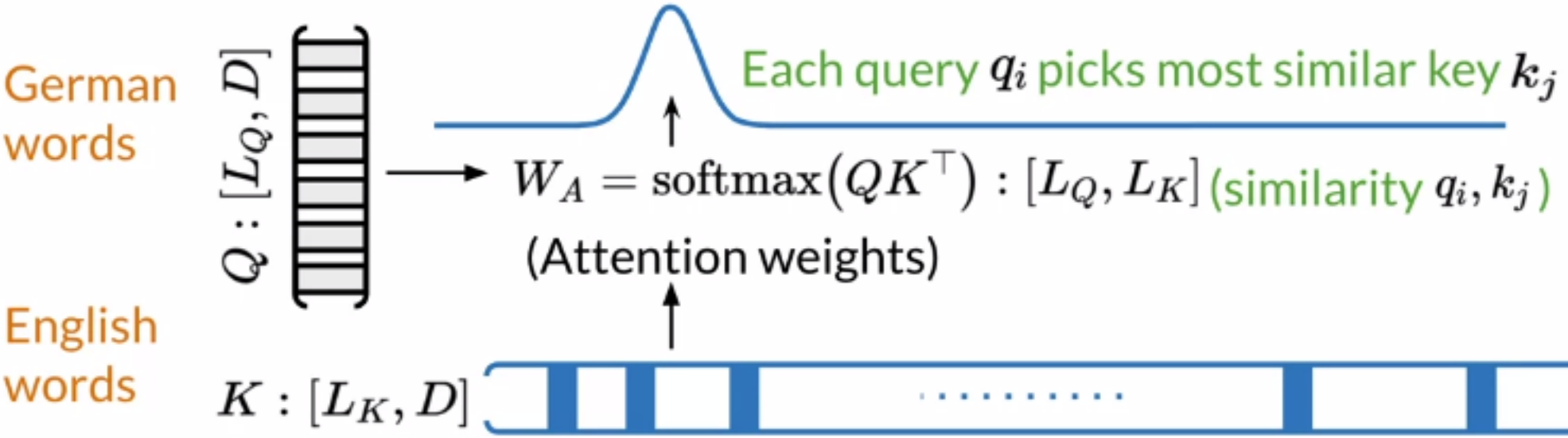
English words

$K : [L_K, D]$

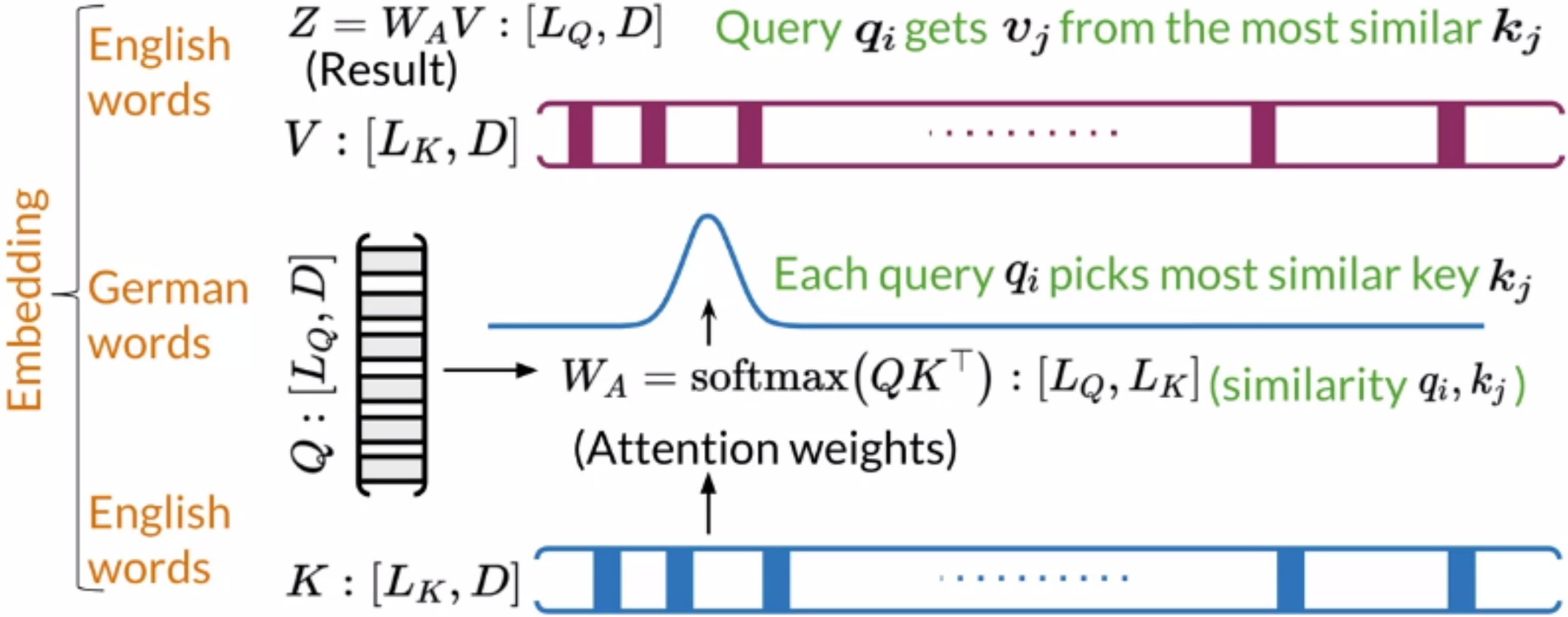
Attention math



Attention math



Attention math



Attention formula

Attention formula

Z

(Result)

Attention formula

$$Z = \text{attention}\left(Q, K, V\right)$$

(Result)

Attention formula

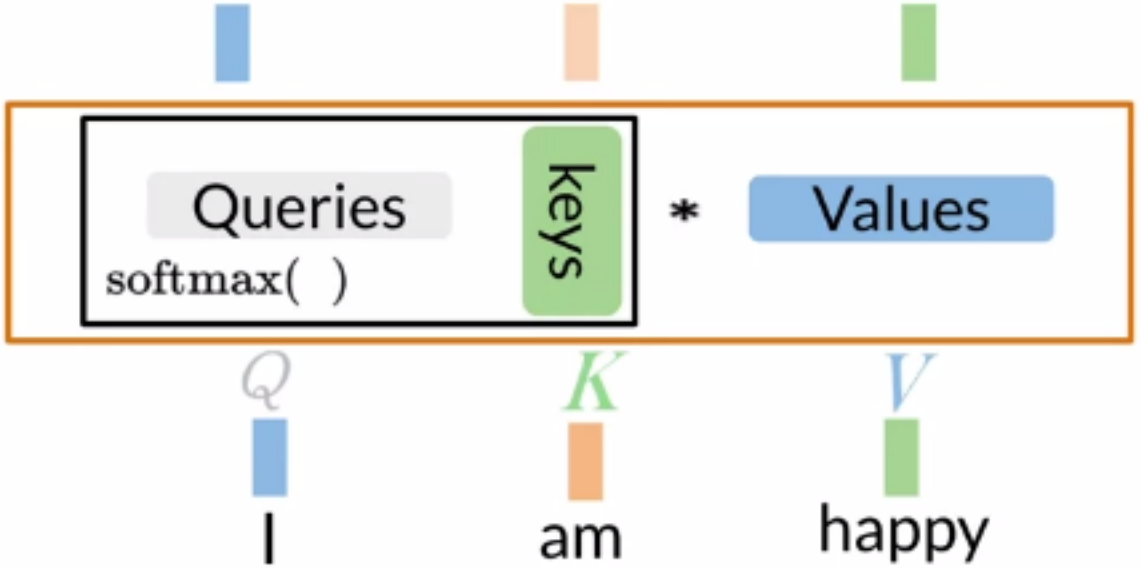
$$Z = \text{attention}(Q, K, V) = \text{softmax}\left(QK^\top\right)V$$

(Result)

Attention formula

$$Z = \text{attention}(Q, K, V) = \text{softmax}(QK^T)V = W_A V$$

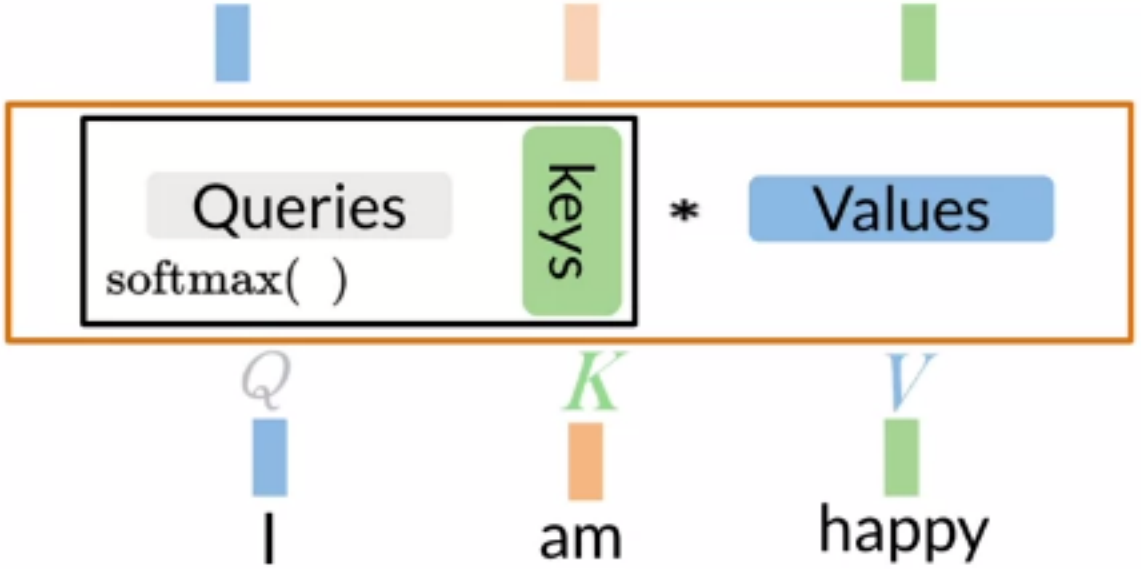
(Result)



Attention formula

$$Z = \text{attention}(Q, K, V) = \text{softmax}(QK^T)V = W_A V$$

(Result)



Summary

- Dot-product Attention is essential for Transformer
- The input to Attention are queries, keys, and values
- A softmax function makes attention more focused on best keys
- GPUs and TPUs is advisable for matrix multiplications

Outline

- Ways of Attention
- Overview of Causal Attention
- Math behind causal attention

Three ways of attention

Three ways of attention

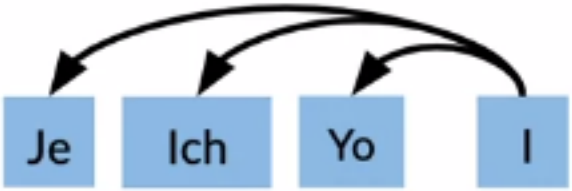
- **Encoder/decoder attention**: One sentence (decoder) looks at another one (encoder)

Three ways of attention

- **Encoder/decoder attention**: One sentence (decoder) looks at another one (encoder)

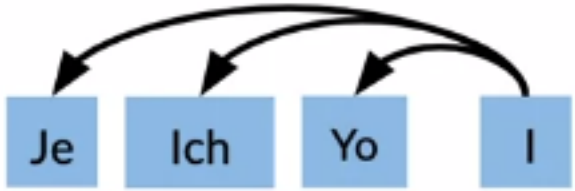
Three ways of attention

- **Encoder/decoder attention**: One sentence (decoder) looks at another one (encoder)



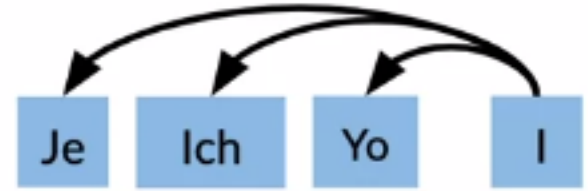
Three ways of attention

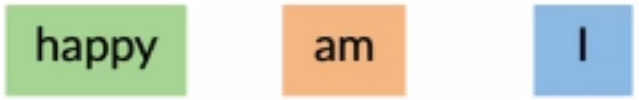
- **Encoder/decoder attention**: One sentence (decoder) looks at another one (encoder)
- **Causal (self) attention**: In one sentence, words look at previous words (used for generation)



Three ways of attention

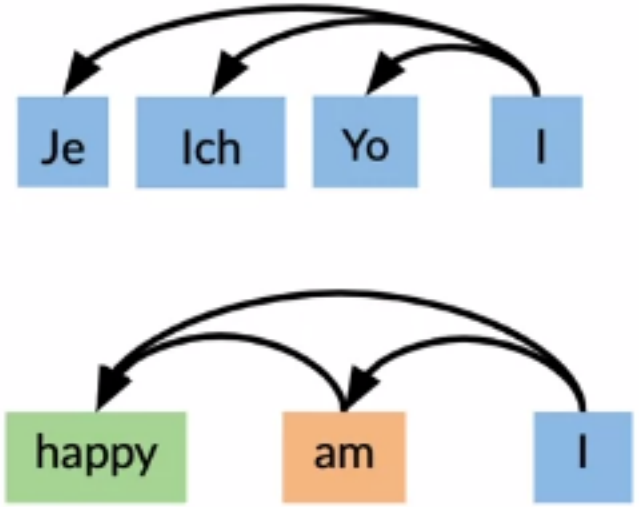
- **Encoder/decoder attention**: One sentence (decoder) looks at another one (encoder)
- **Causal (self) attention**: In one sentence, words look at previous words (used for generation)





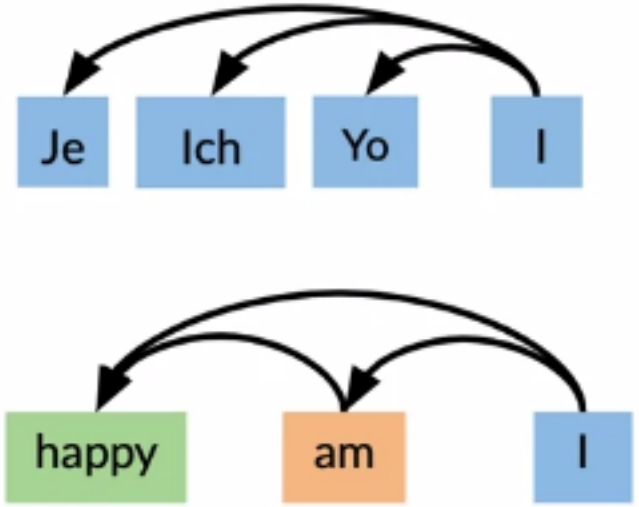
Three ways of attention

- **Encoder/decoder attention**: One sentence (decoder) looks at another one (encoder)
- **Causal (self) attention**: In one sentence, words look at previous words (used for generation)



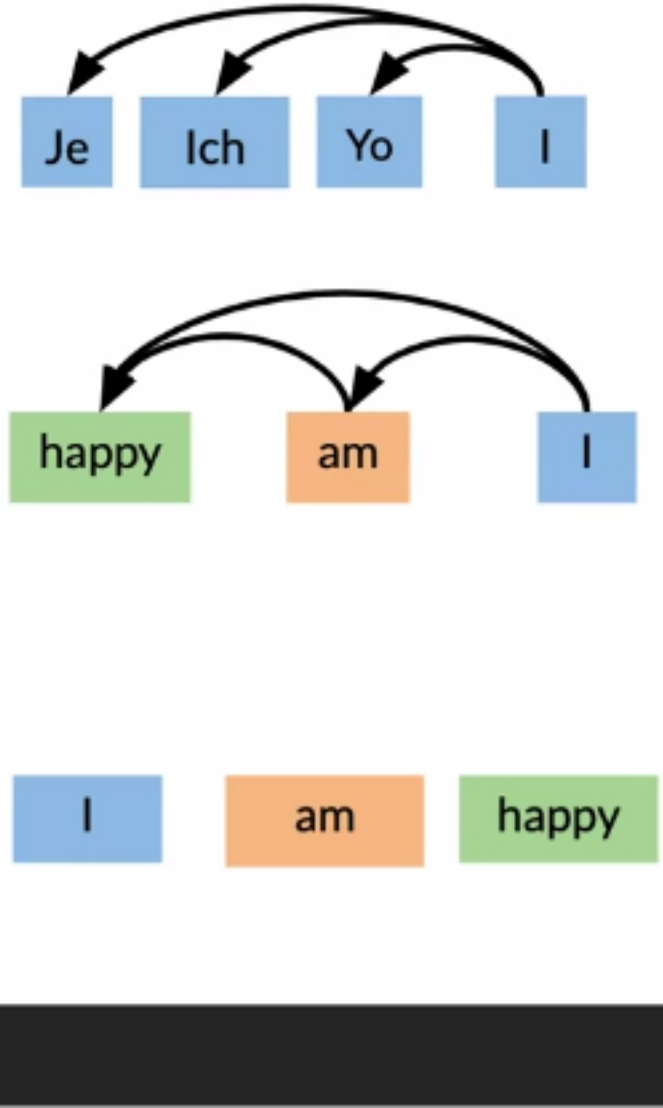
Three ways of attention

- **Encoder/decoder attention**: One sentence (decoder) looks at another one (encoder)
- **Causal (self) attention**: In one sentence, words look at previous words (used for generation)
- **Bi-directional self attention**: In one sentence, words look at both previous and future words



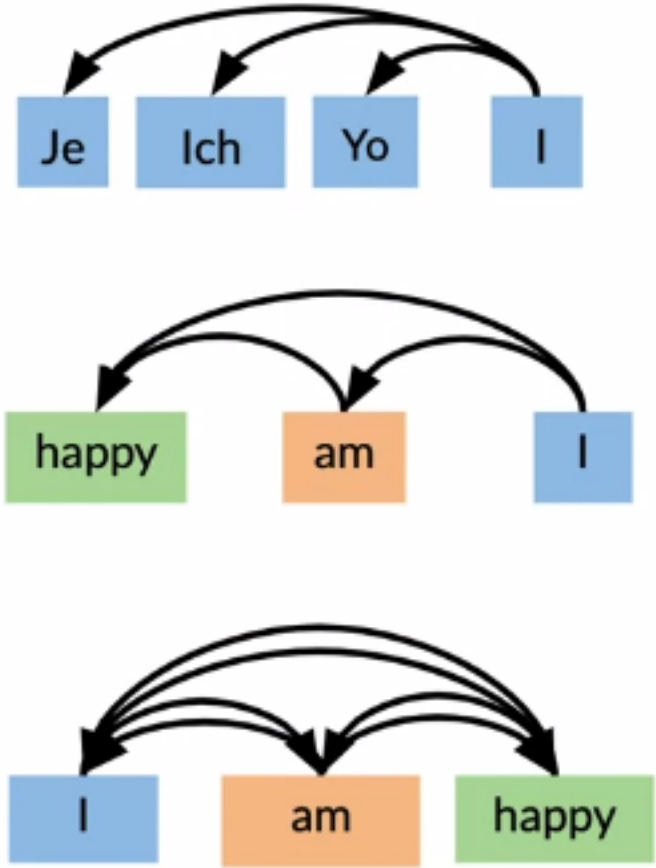
Three ways of attention

- **Encoder/decoder attention**: One sentence (decoder) looks at another one (encoder)
- **Causal (self) attention**: In one sentence, words look at previous words (used for generation)
- **Bi-directional self attention**: In one sentence, words look at both previous and future words



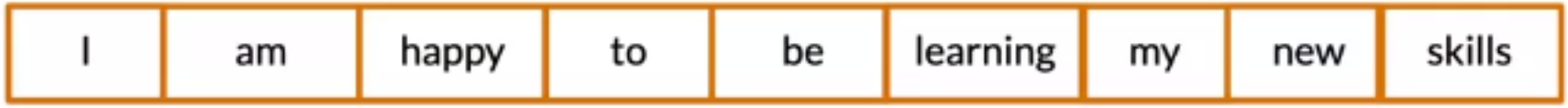
Three ways of attention

- **Encoder/decoder attention**: One sentence (decoder) looks at another one (encoder)
- **Causal (self) attention**: In one sentence, words look at previous words (used for generation)
- **Bi-directional self attention**: In one sentence, words look at both previous and future words



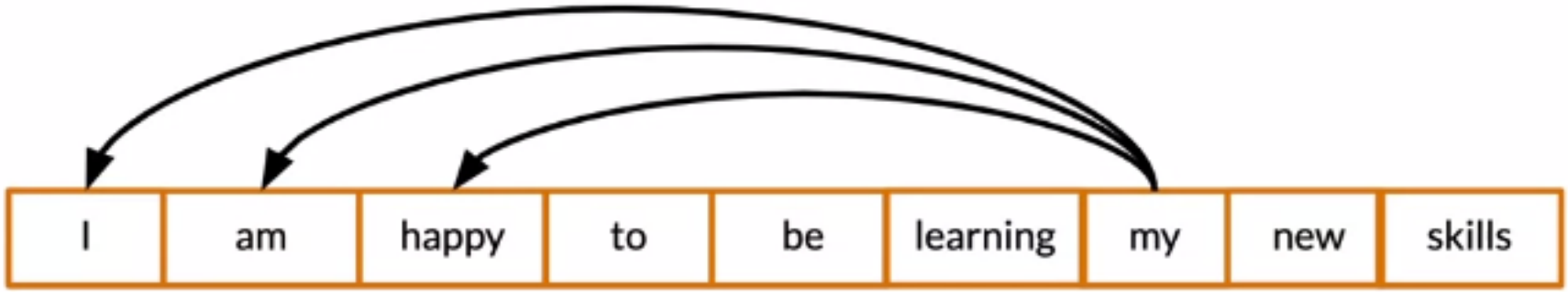
Causal attention

- Queries and keys are words from the same sentence



Causal attention

- Queries and keys are words from the same sentence
- Queries should only be allowed to look at words before

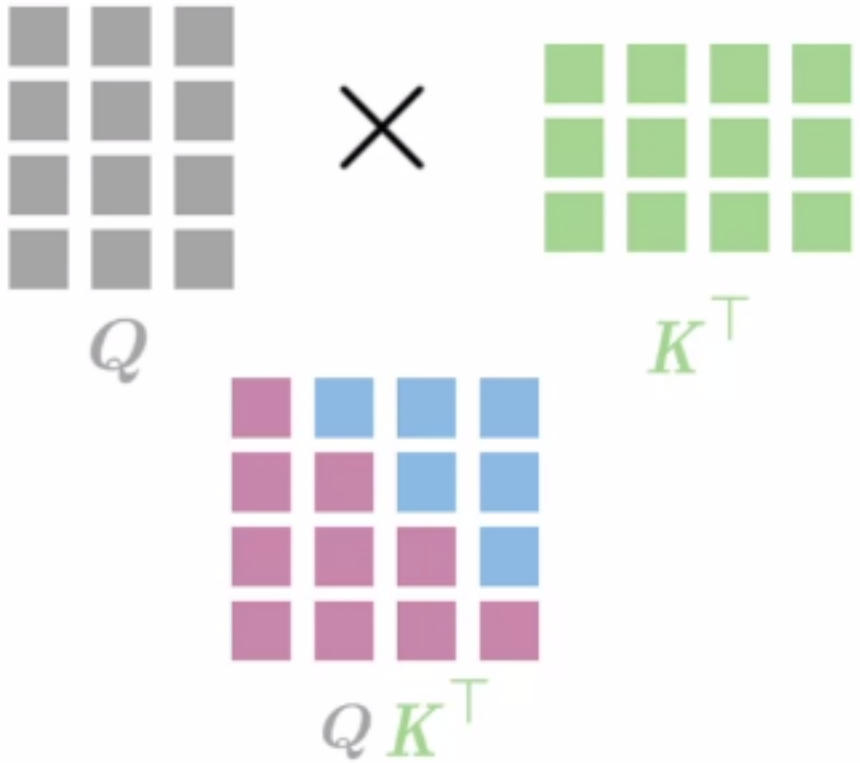


Causal attention math

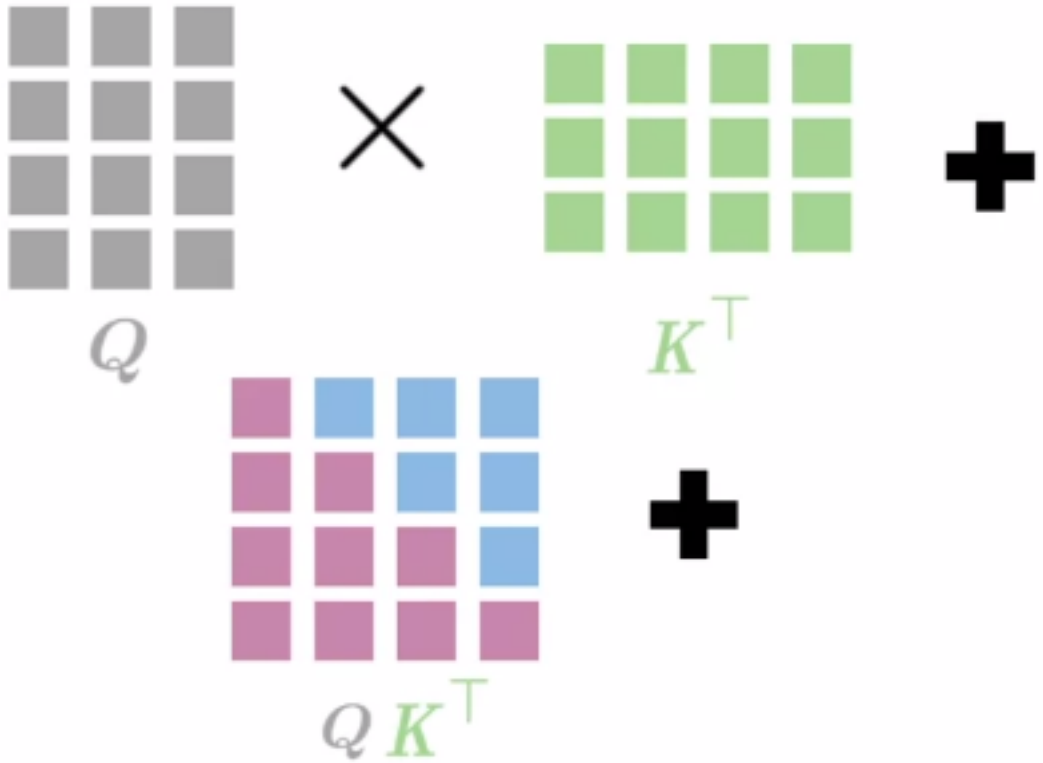
Causal attention math

Causal attention math

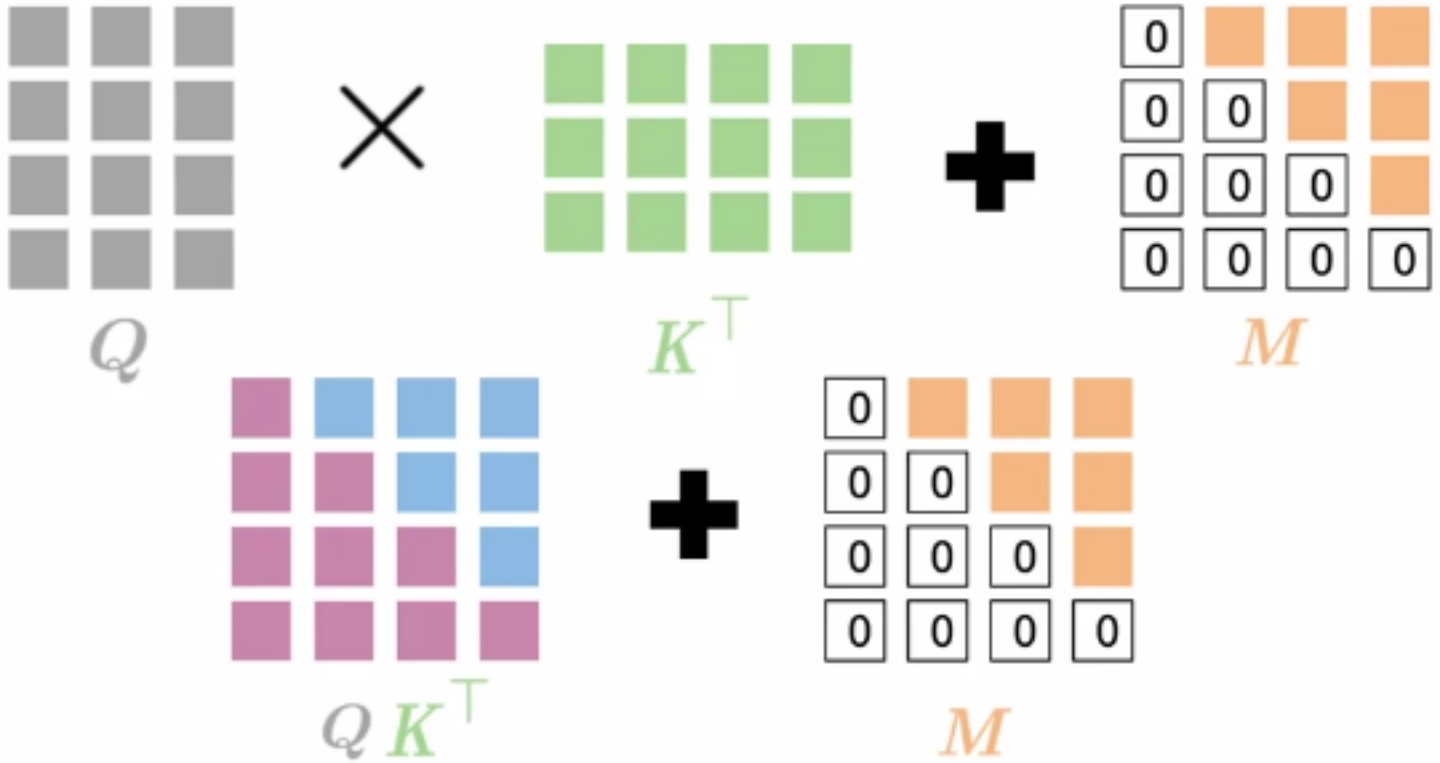
Causal attention math



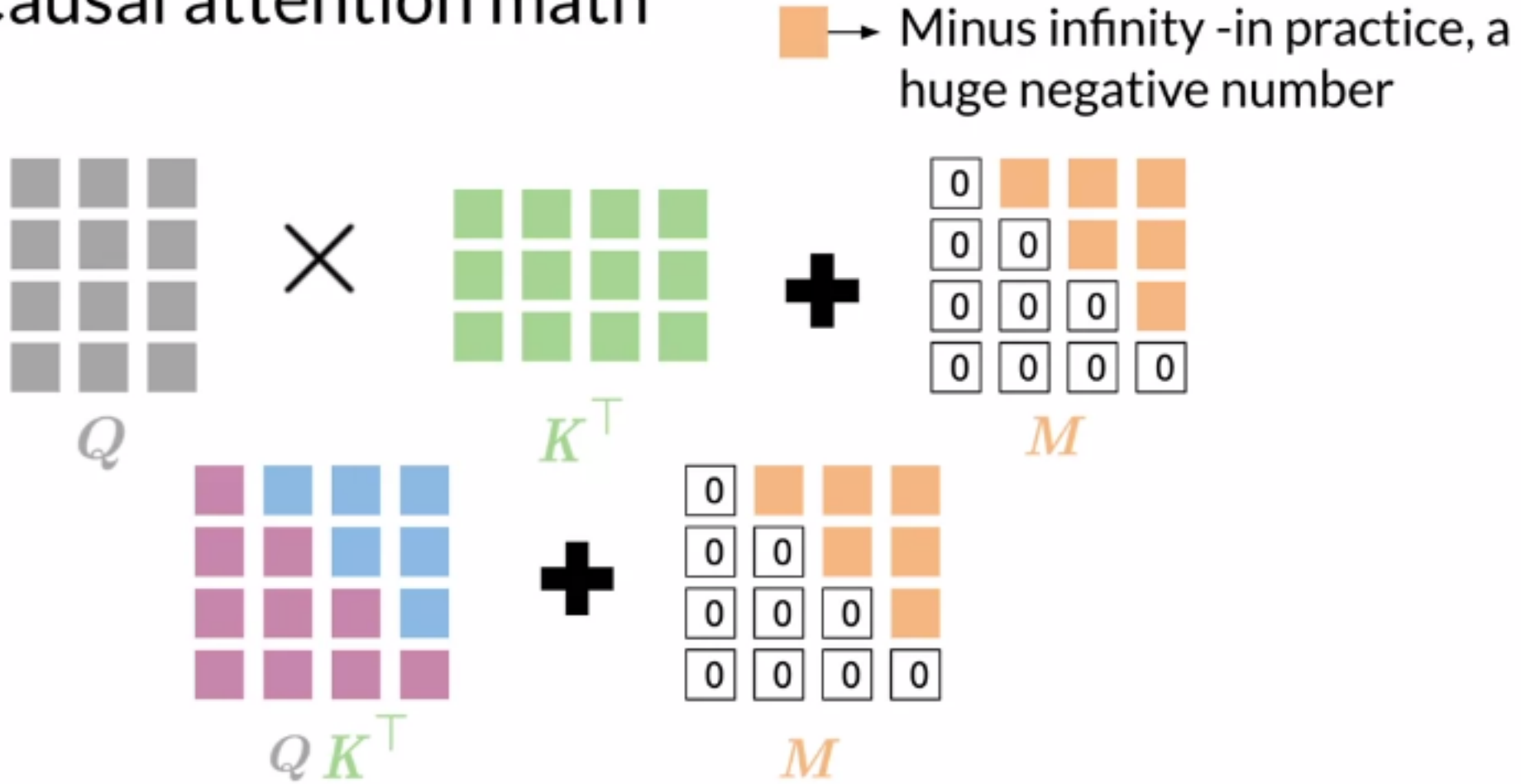
Causal attention math



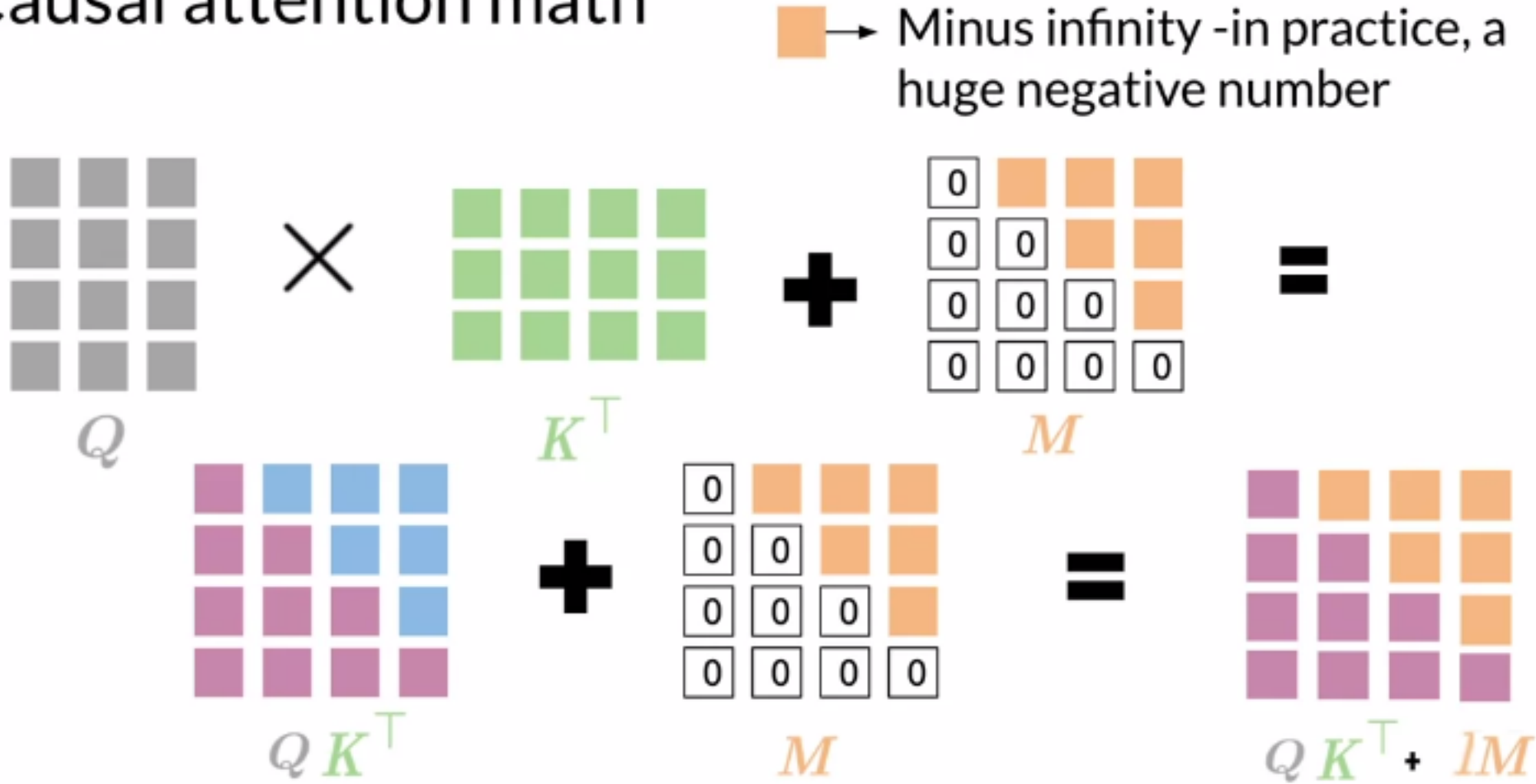
Causal attention math



Causal attention math



Causal attention math



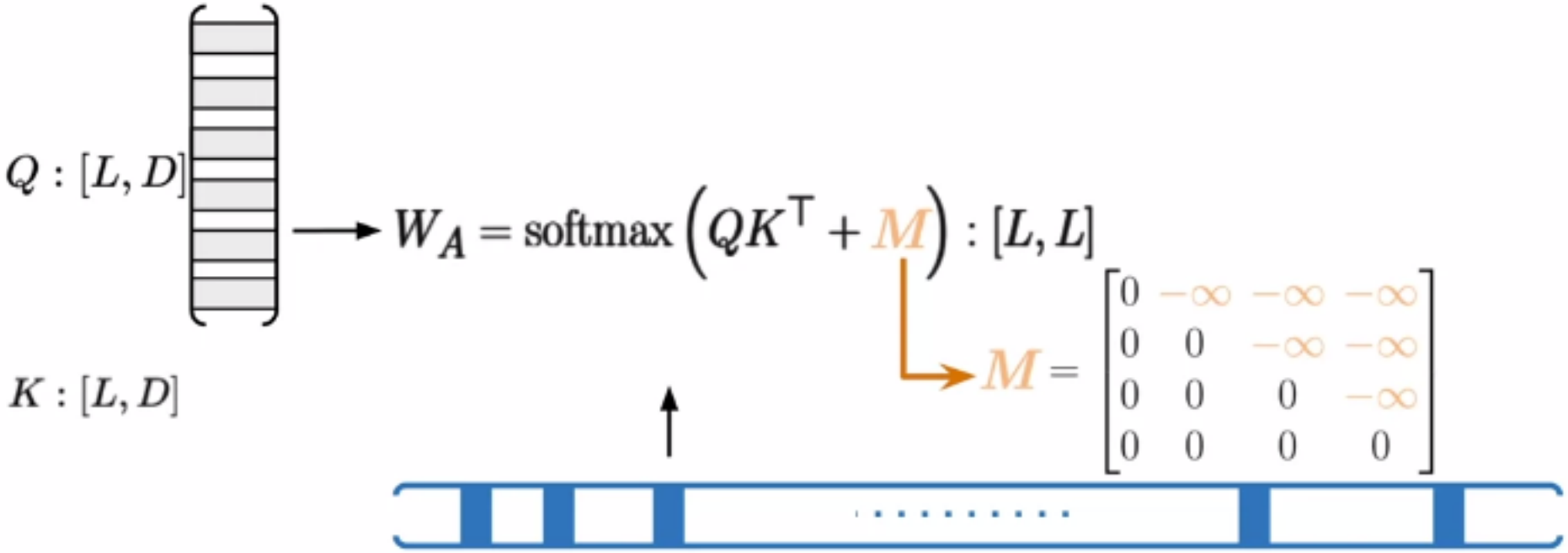
Causal attention math

$$W_A = \text{softmax}\left(QK^\top + M\right) : [L, L]$$

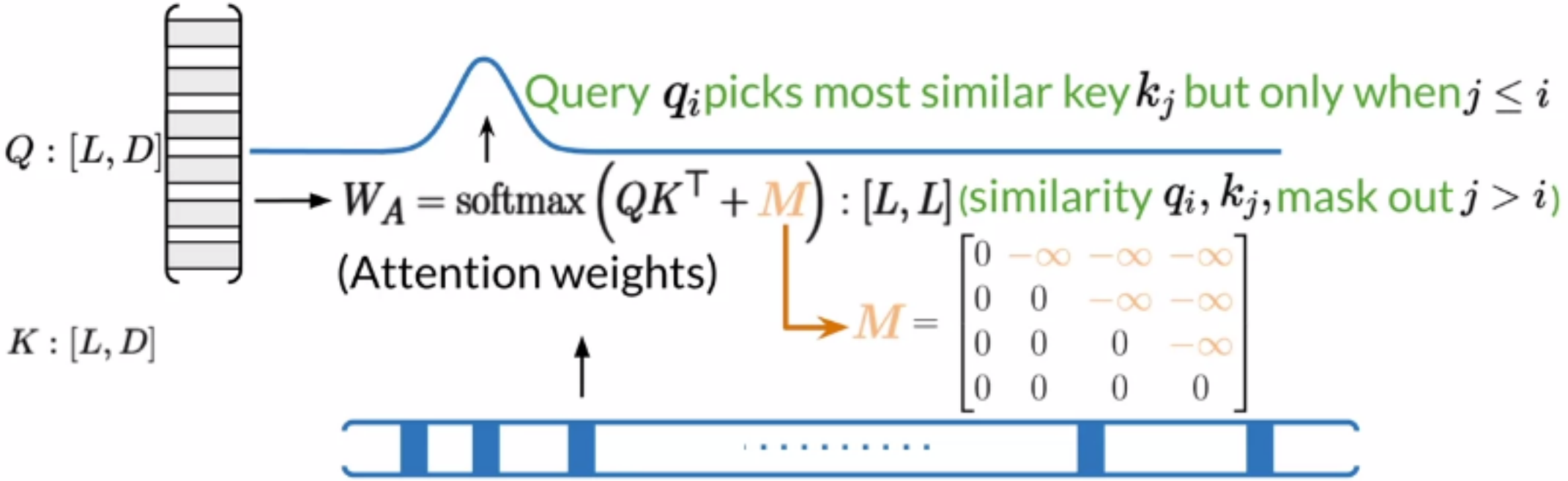
$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$K : [L, D]$

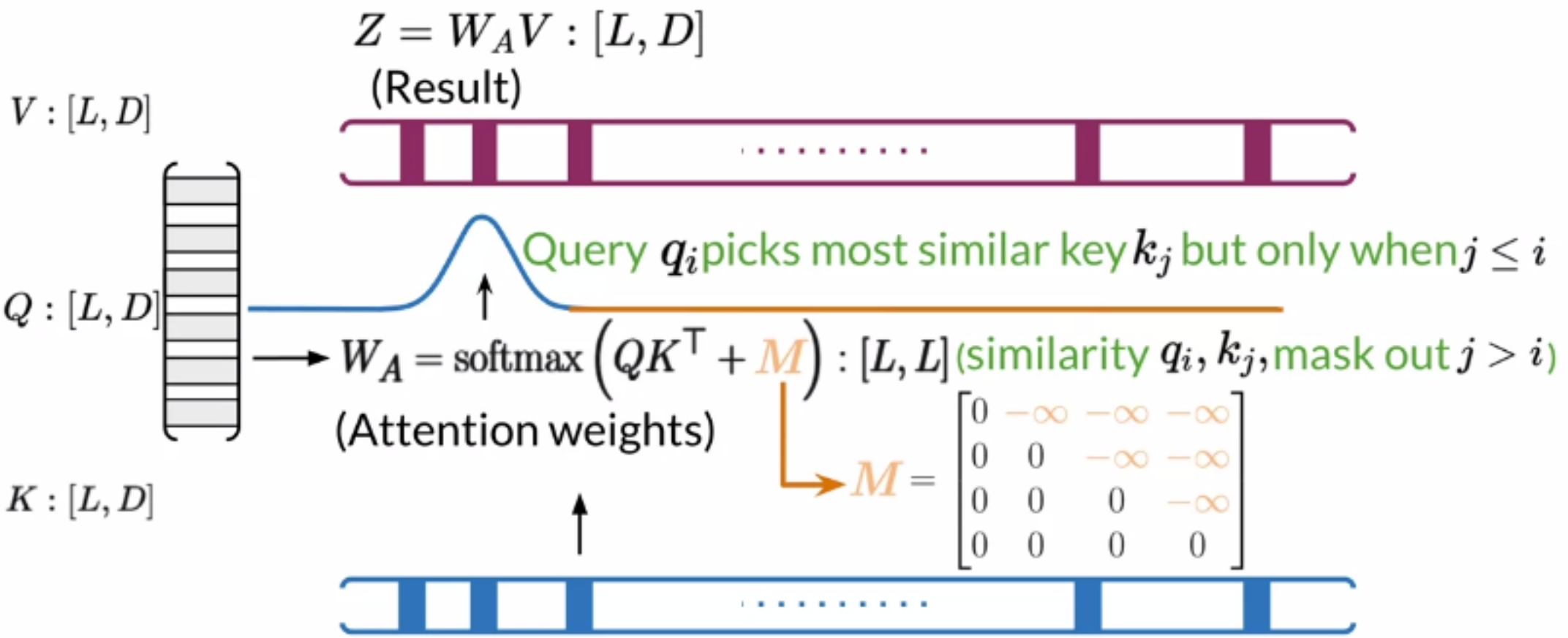
Causal attention math



Causal attention math



Causal attention math



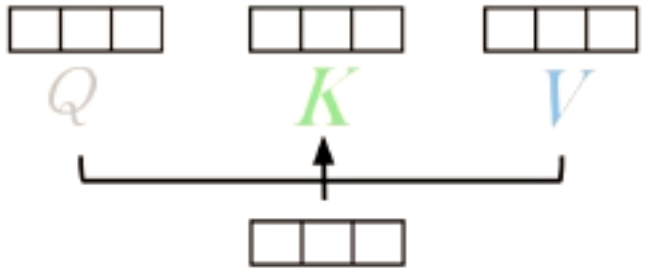
Summary

- There are three main ways of Attention: Encoder/Decoder, Causal and Bi-directional type
- In causal attention, queries and keys come from the same sentence and queries search among words before only

Outline

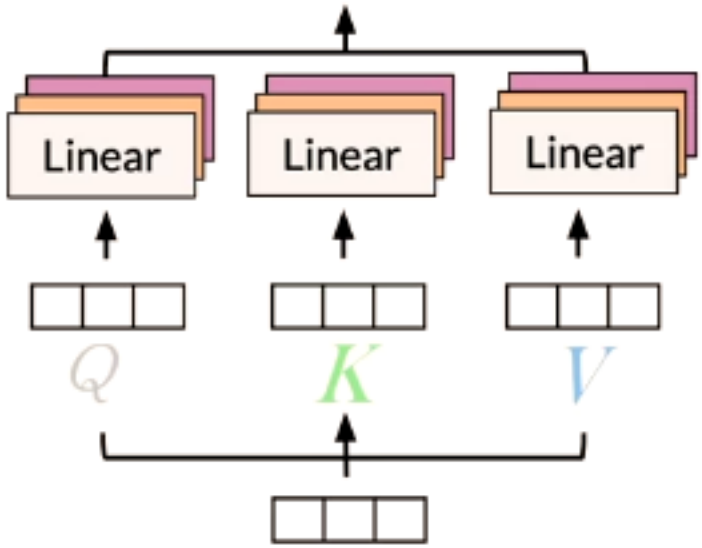
- Intuition of Multi-Head Attention
- Scaled dot-product and concatenation
- Multi-Head Attention formula

Multi-Head Attention



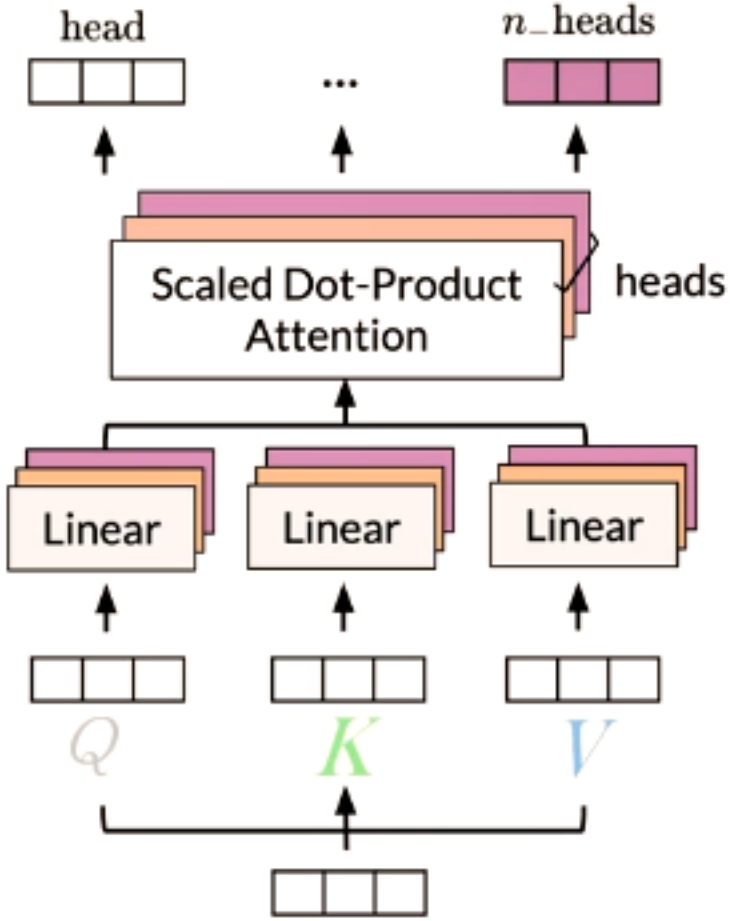
Multi-Head Attention

- Each head uses different linear transformations to represent words



Multi-Head Attention

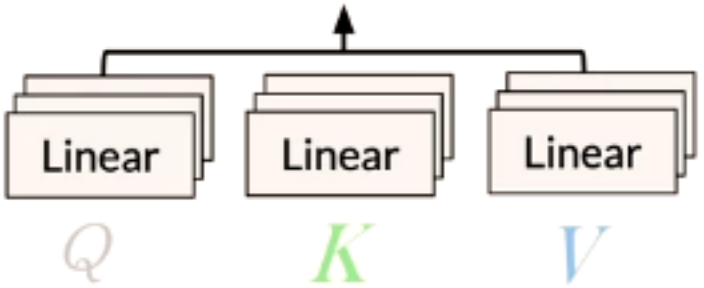
- Each head uses different linear transformations to represent words
- Different heads can learn different relationships between words



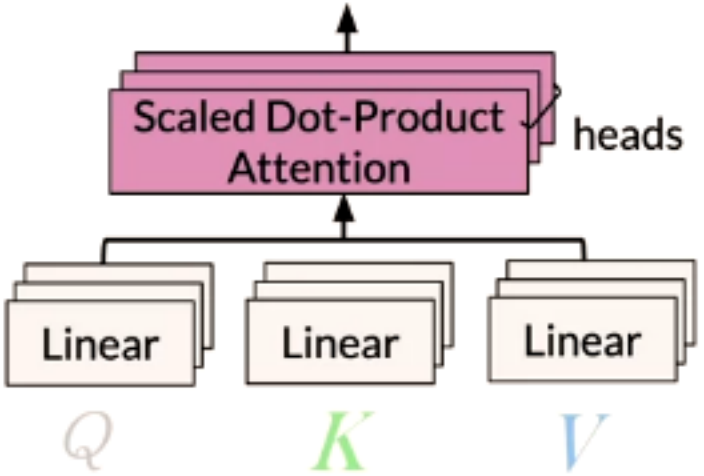
Multi-Head Attention - Overview

Q K V

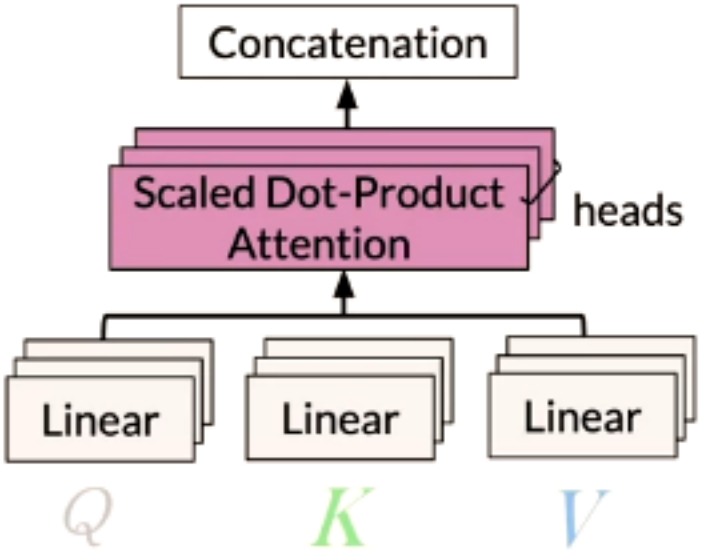
Multi-Head Attention - Overview



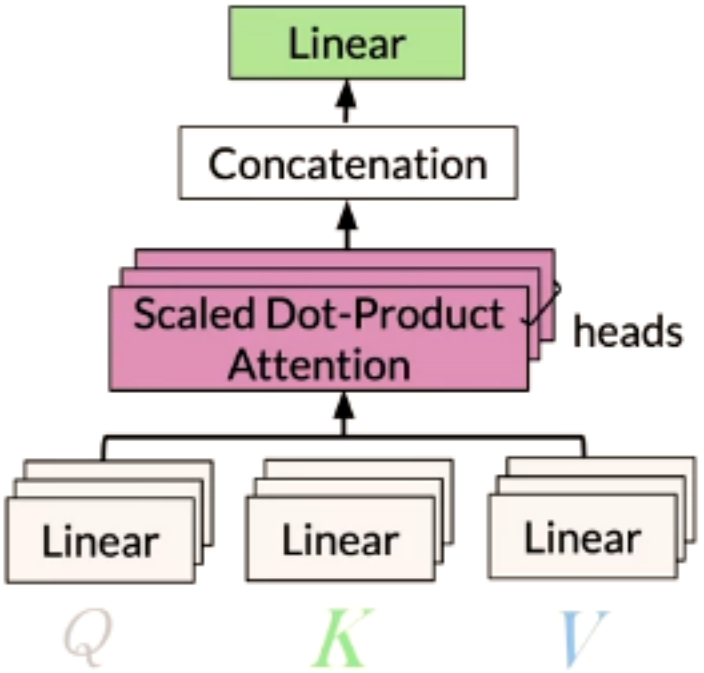
Multi-Head Attention - Overview



Multi-Head Attention - Overview



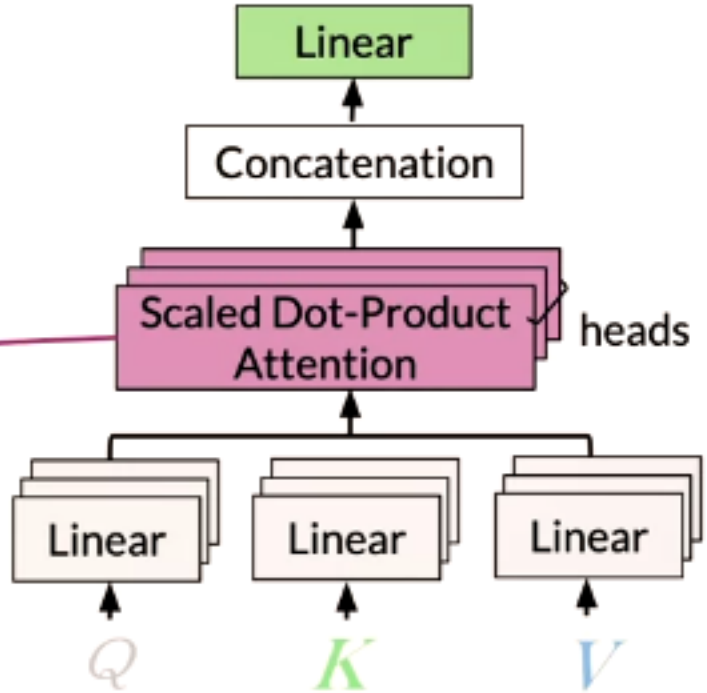
Multi-Head Attention - Overview



Multi-Head Attention - Scaled dot product

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

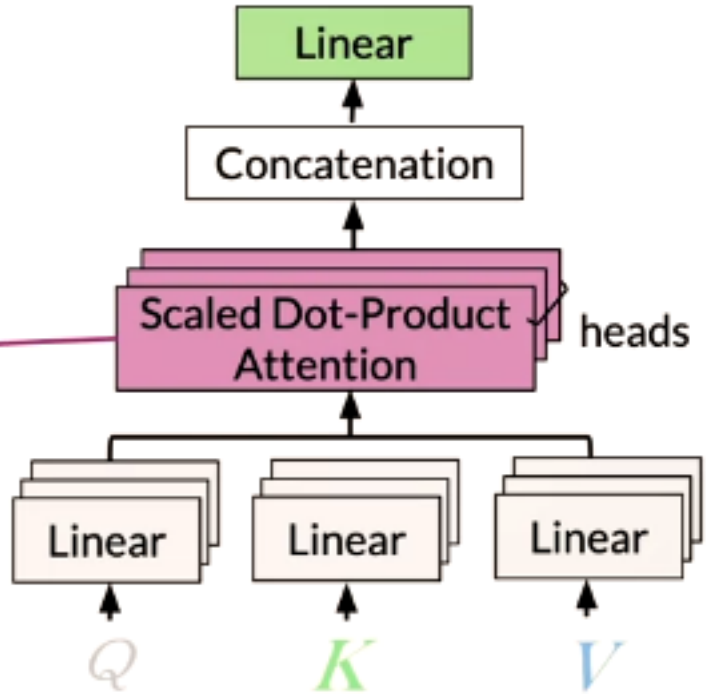
Query and key dimension



Multi-Head Attention - Scaled dot product

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

Query and key dimension



Multi-Head Attention - Concatenation

- Input(Q, K, V): `[batch, length, d_model]`

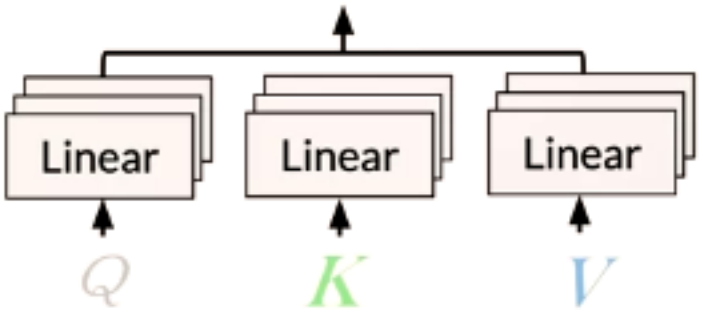
Multi-Head Attention - Concatenation

- Input(Q, K, V): `[batch, length, d_model]`

512, 1024

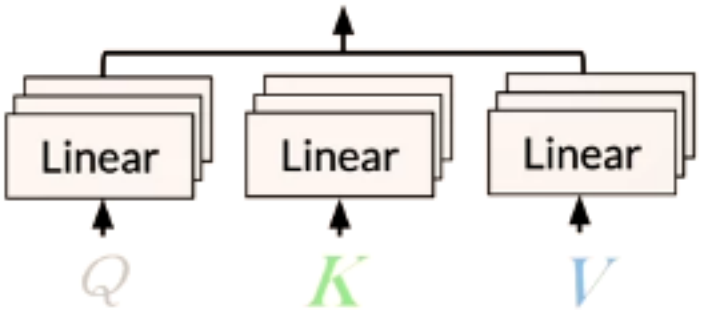
Multi-Head Attention - Concatenation

- Input(Q, K, V): `[batch, length, d_model]`
- Linear layer: `[batch, length, n_heads * d_head]`



Multi-Head Attention - Concatenation

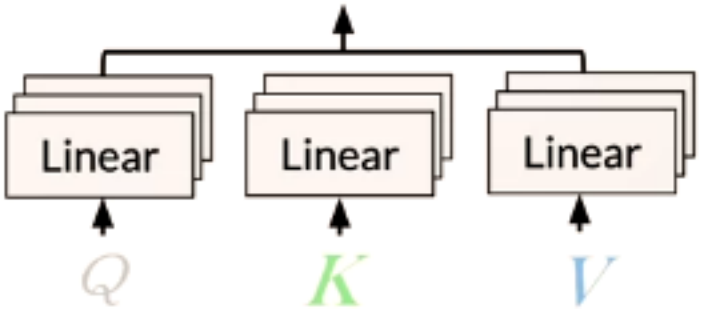
- Input(Q, K, V): `[batch, length, d_model]`
- Linear layer: `[batch, length, n_heads * d_head]`



Multi-Head Attention - Concatenation

- Input(Q, K, V): `[batch, length, d_model]`
- Linear layer: `[batch, length, n_heads * d_head]`

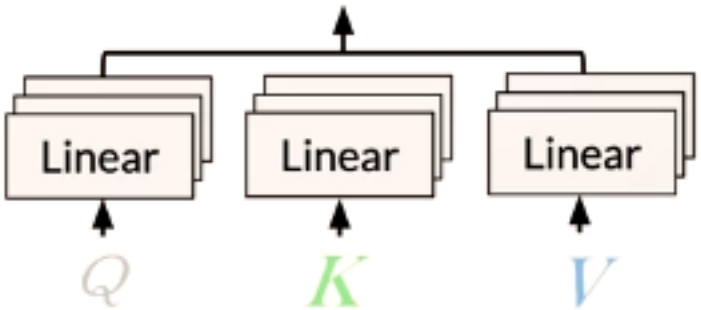
4, 6, 16, ...



Multi-Head Attention - Concatenation

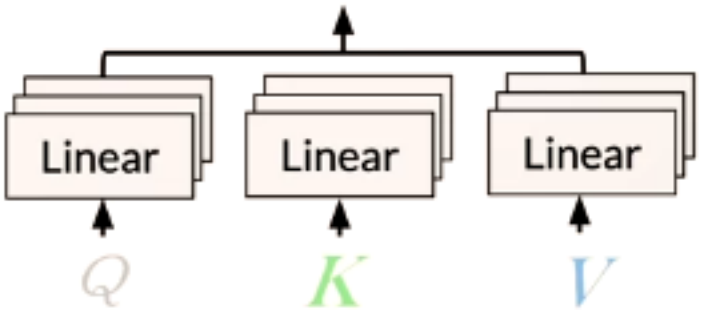
- Input(Q, K, V): `[batch, length, d_model]`
- Linear layer: `[batch, length, n_heads * d_head]`

4, 6, 16, ... 64, 128



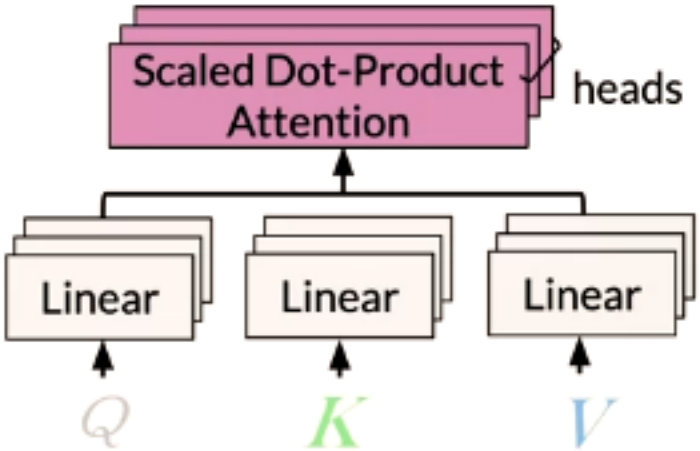
Multi-Head Attention - Concatenation

- Input(Q, K, V): `[batch, length, d_model]`
- Linear layer: `[batch, length, n_heads * d_head]`
- Transpose: `[batch, n_heads, length, d_head]`



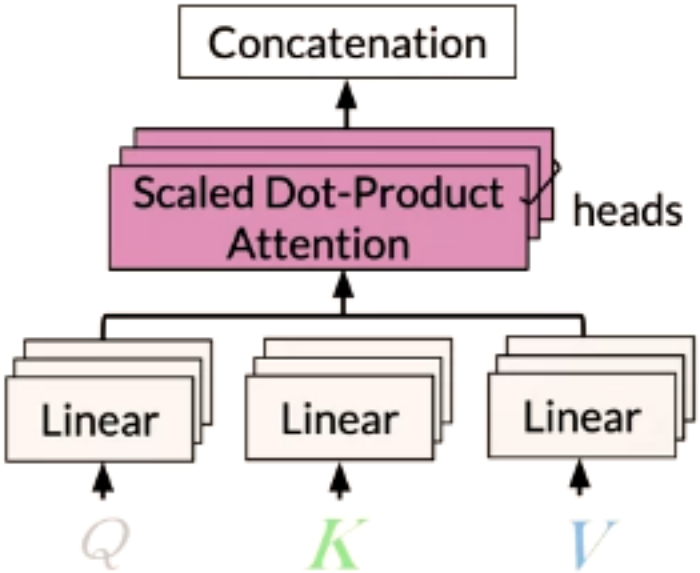
Multi-Head Attention - Concatenation

- Input(Q, K, V): `[batch, length, d_model]`
- Linear layer: `[batch, length, n_heads * d_head]`
- Transpose: `[batch, n_heads, length, d_head]`
- Apply attention treating `n_heads` like batch



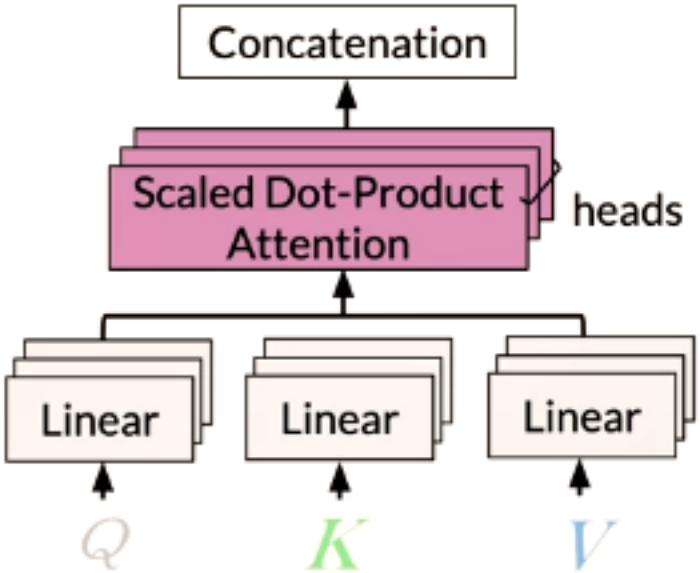
Multi-Head Attention - Concatenation

- Input(Q, K, V): `[batch, length, d_model]`
- Linear layer: `[batch, length, n_heads * d_head]`
- Transpose: `[batch, n_heads, length, d_head]`
- Apply attention treating `n_heads` like batch
- Result shape: `[batch, n_heads, length, d_head]`



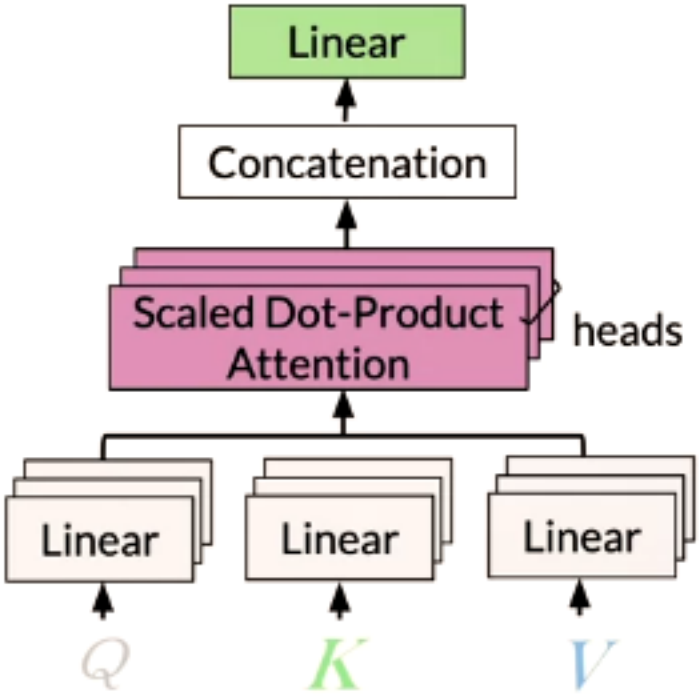
Multi-Head Attention - Concatenation

- Input(Q, K, V): `[batch, length, d_model]`
- Linear layer: `[batch, length, n_heads * d_head]`
- Transpose: `[batch, n_heads, length, d_head]`
- Apply attention treating `n_heads` like batch
- Result shape: `[batch, n_heads, length, d_head]`
- Transpose: `[batch, length, n_heads * d_head]`



Multi-Head Attention - Concatenation

- Input(Q, K, V): `[batch, length, d_model]`
- Linear layer: `[batch, length, n_heads * d_head]`
- Transpose: `[batch, n_heads, length, d_head]`
- Apply attention treating `n_heads` like batch
- Result shape: `[batch, n_heads, length, d_head]`
- Transpose: `[batch, length, n_heads * d_head]`
- Linear layer into: `[batch size, length, d_model]`



Multi-Head Attention math

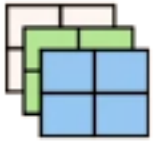
Multi-Head Attention math

Embedding

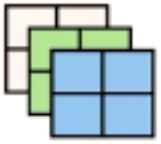
Multi-Head Attention math

Embedding

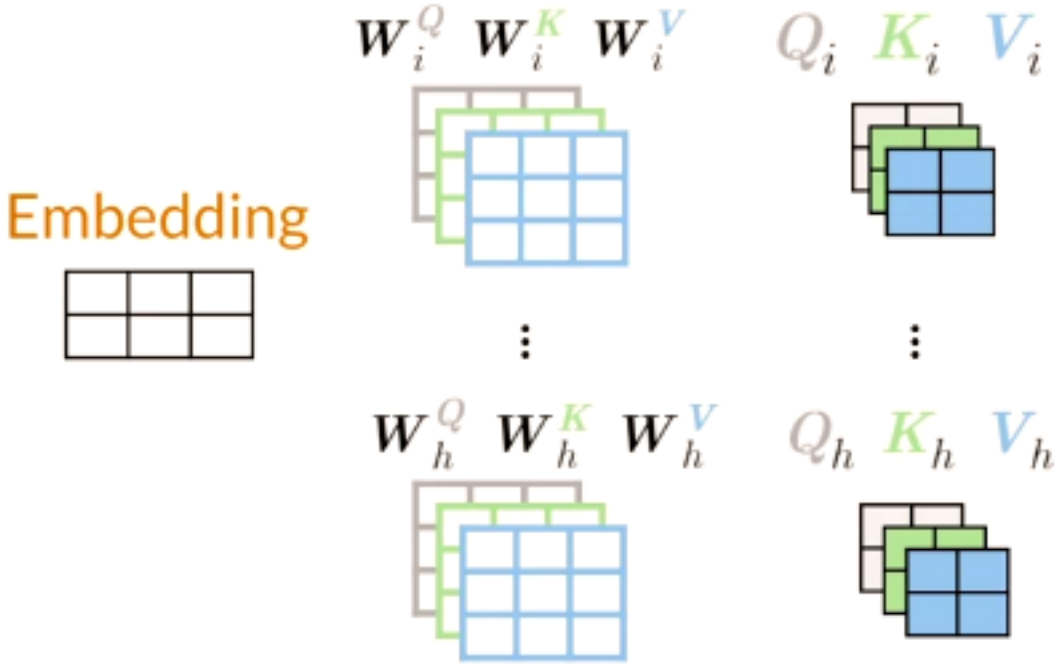
Q_i K_i V_i



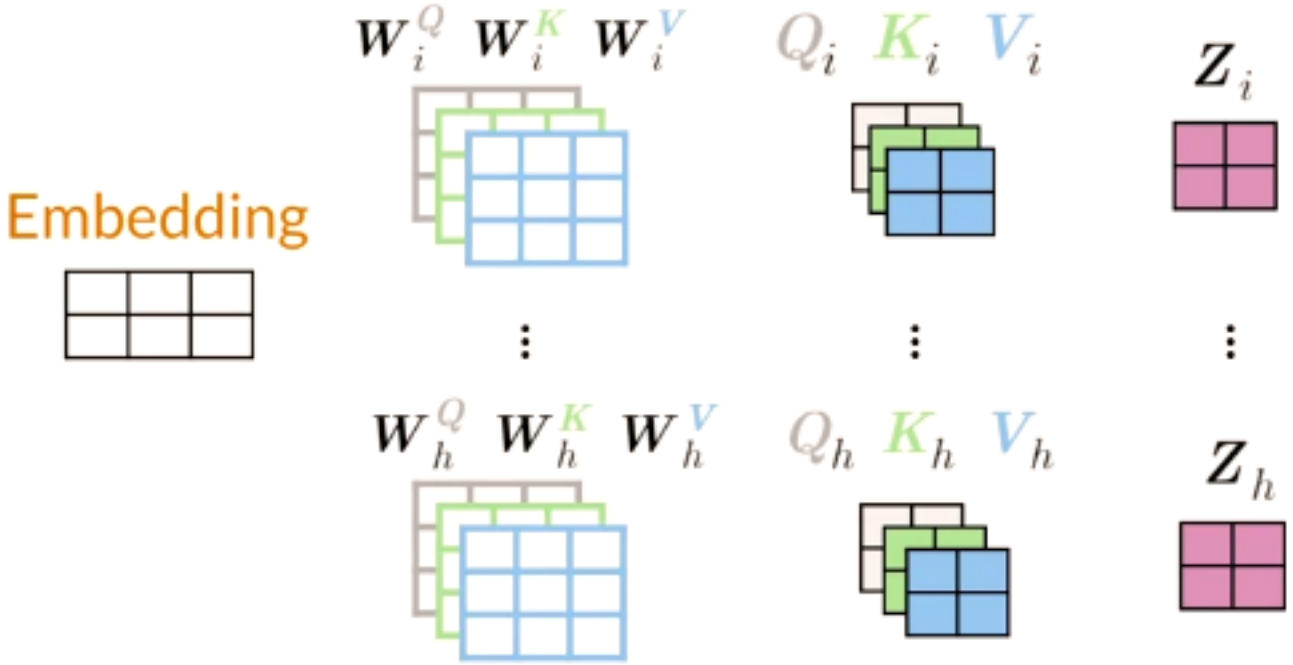
Q_h K_h V_h



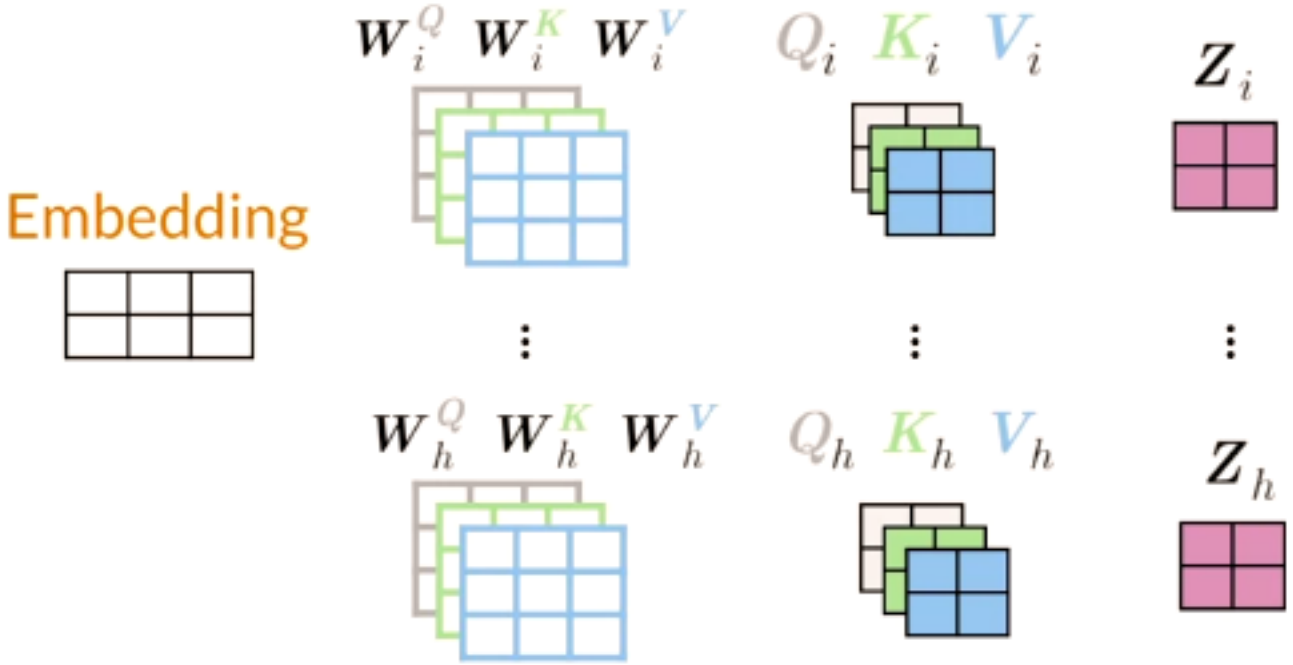
Multi-Head Attention math



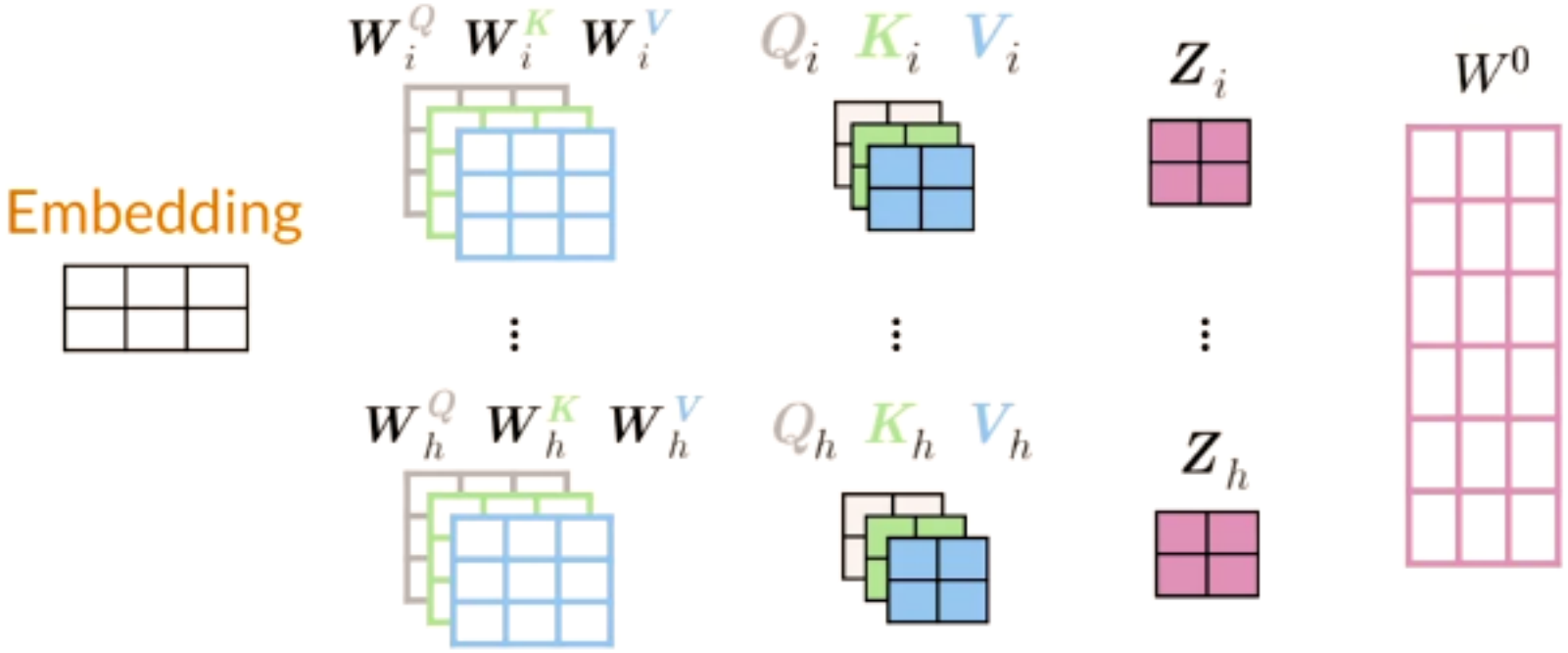
Multi-Head Attention math



Multi-Head Attention math



Multi-Head Attention math

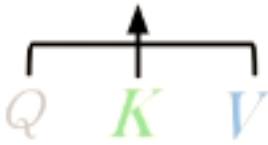


Multi-Head Attention Formula

Q K V Q K V

Multi-Head Attention Formula

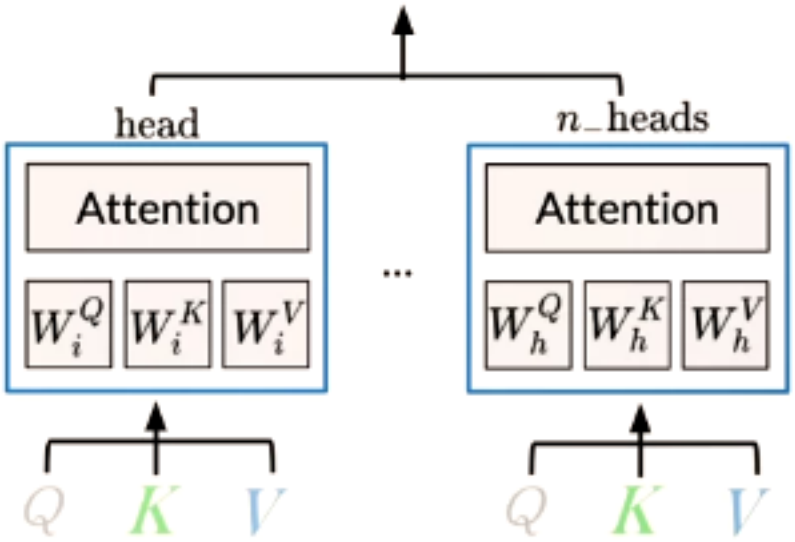
$$\text{MultiHead}(Q, K, V) = \text{Concat}(h_1, \ldots, h_h)W^0$$



Multi-Head Attention Formula

$$\text{MultiHead}(Q, K, V) = \text{Concat}(h_1, \ldots, h_h)W^0$$

where $h_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

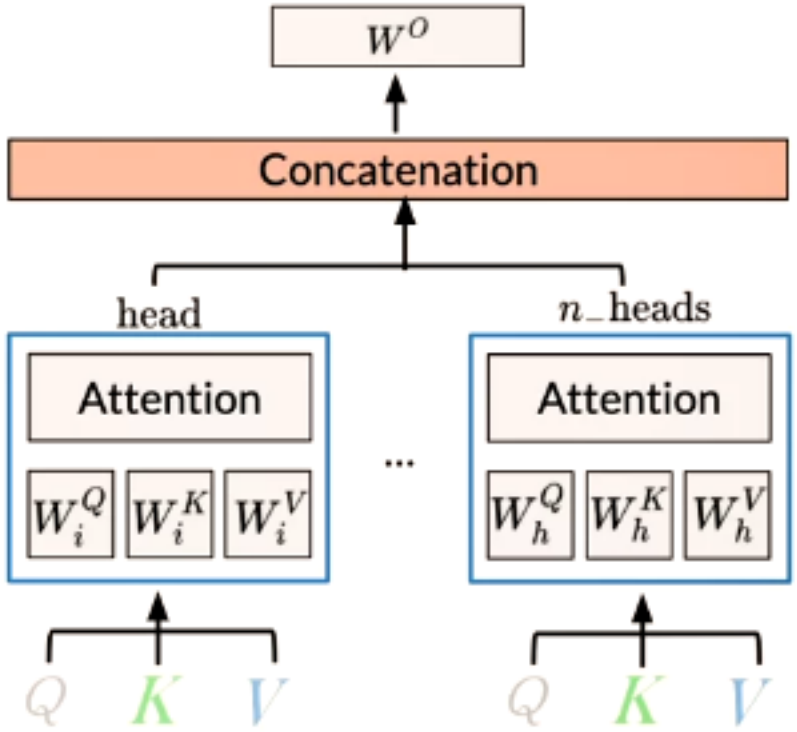


Multi-Head Attention Formula

$$\text{MultiHead}(Q, K, V) = \text{Concat}(h_1, \ldots, h_h)W^0$$

where $h_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

Each head h_i is the attention function of **Query, Key and Value** with trainable parameters (W_i^Q, W_i^K, W_i^V)

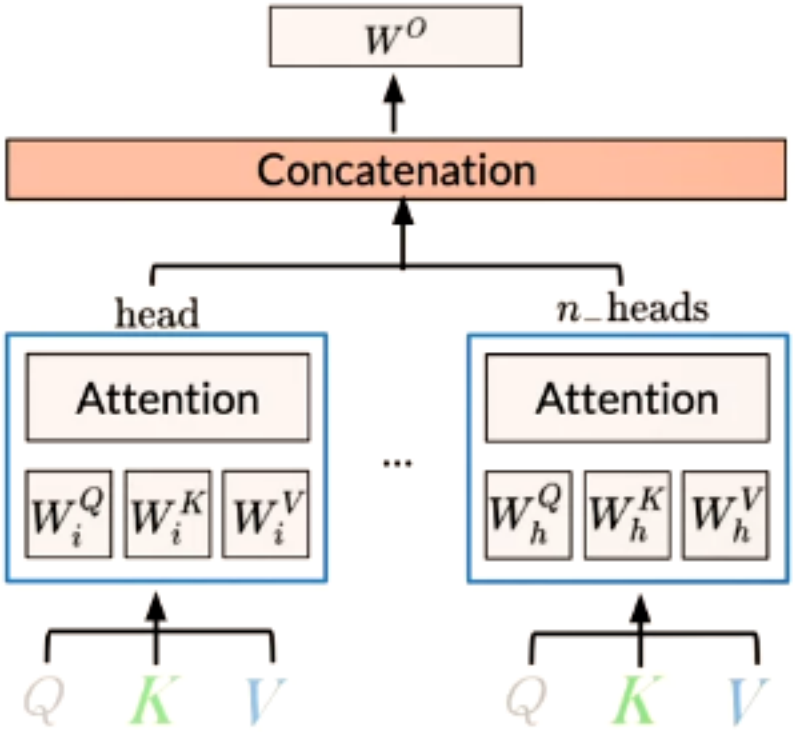


Multi-Head Attention Formula

$$\text{MultiHead}(Q, K, V) = \text{Concat}(h_1, \ldots, h_h)W^0$$

where $h_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

Each head h_i is the attention function of **Query, Key and Value** with trainable parameters (W_i^Q, W_i^K, W_i^V)



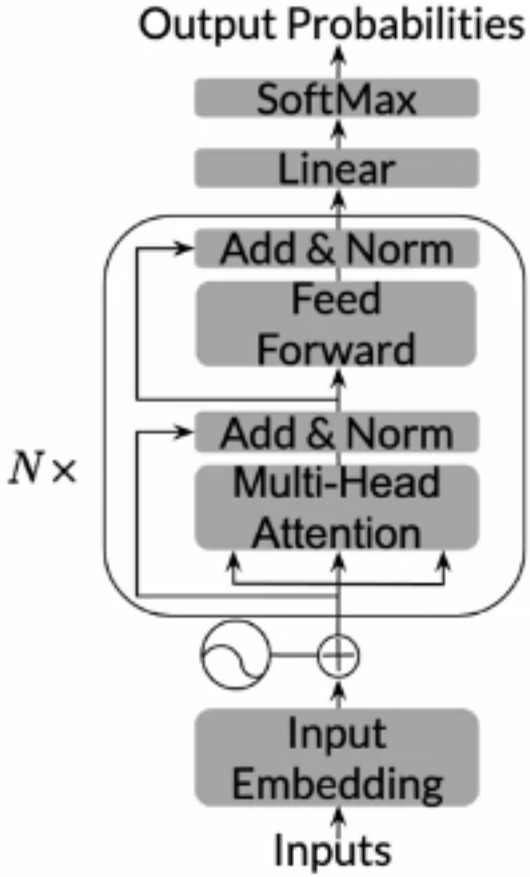
Summary

- Different heads can learn different relationship between words
- Scaled dot-product is adequate for Multi-Head Attention
- Multi-Headed models attend to information from different representations at different positions

Outline

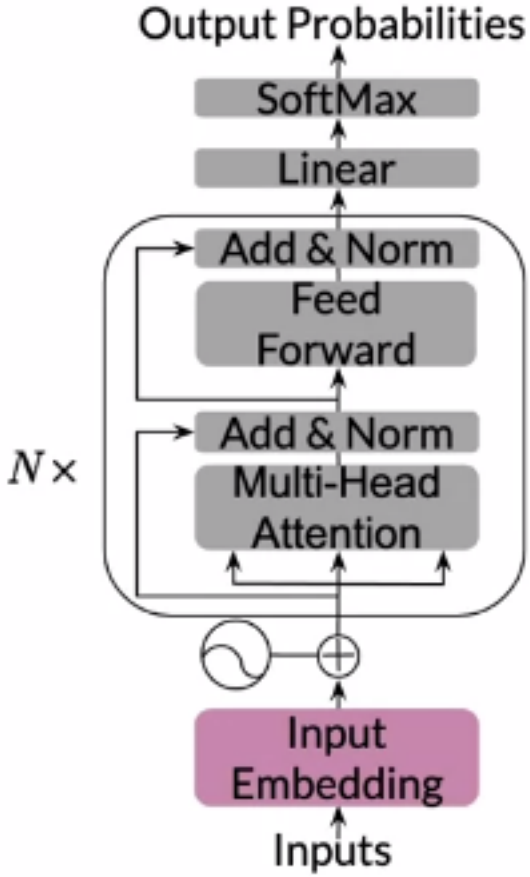
- Overview of Transformer decoder
- Implementation (decoder and feed-forward block)

Transformer decoder



Overview

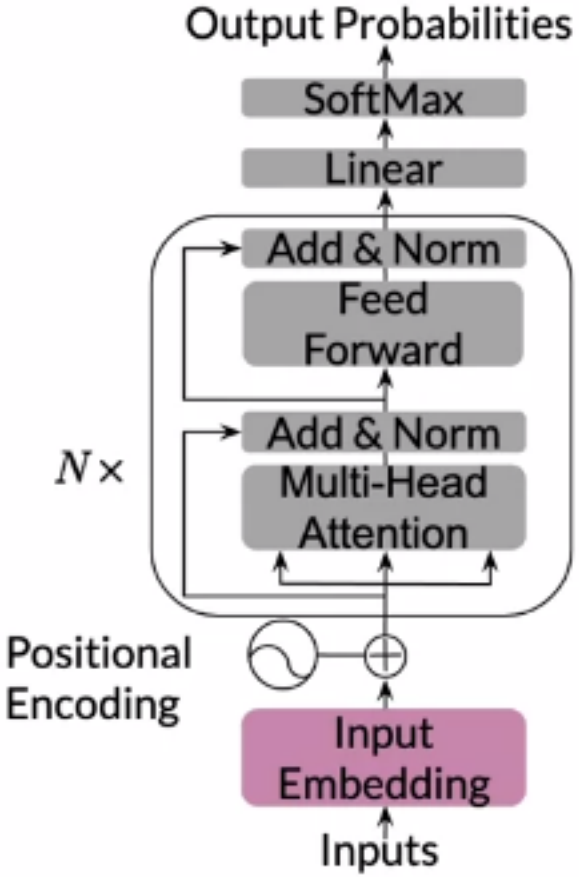
Transformer decoder



Overview

- input: sentence or paragraph
  - we predict the next word

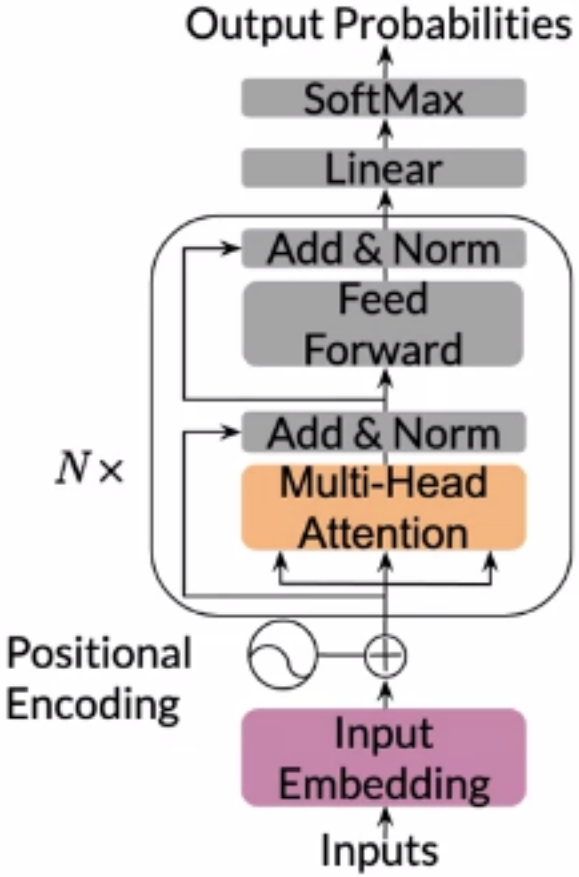
Transformer decoder



Overview

- input: sentence or paragraph
  - we predict the next word
- sentence gets embedded, add positional encoding
  - (vectors representing $\{0, 1, 2, \ldots, K\}$)

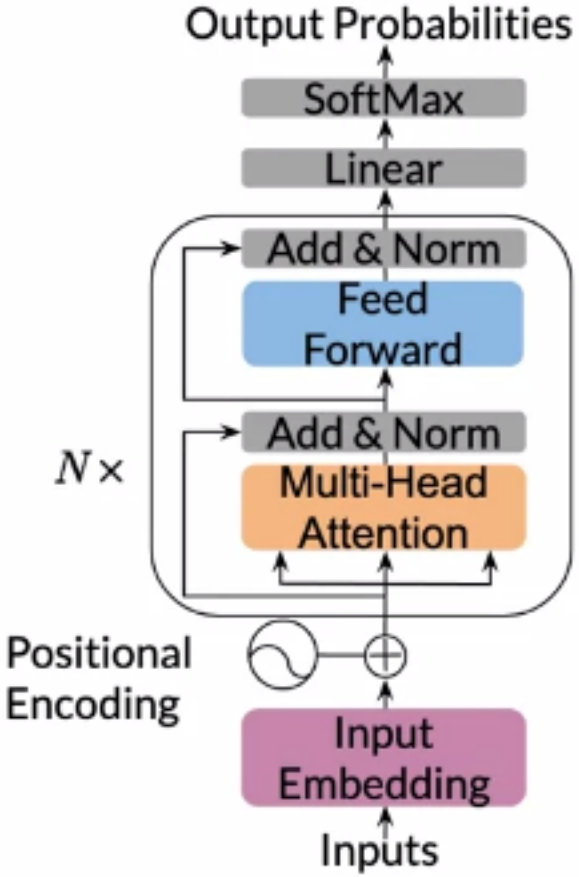
Transformer decoder



Overview

- input: sentence or paragraph
  - we predict the next word
- sentence gets embedded, add positional encoding
  - (vectors representing $\{0, 1, 2, \ldots, K\}$)
- multi-head attention looks at previous words

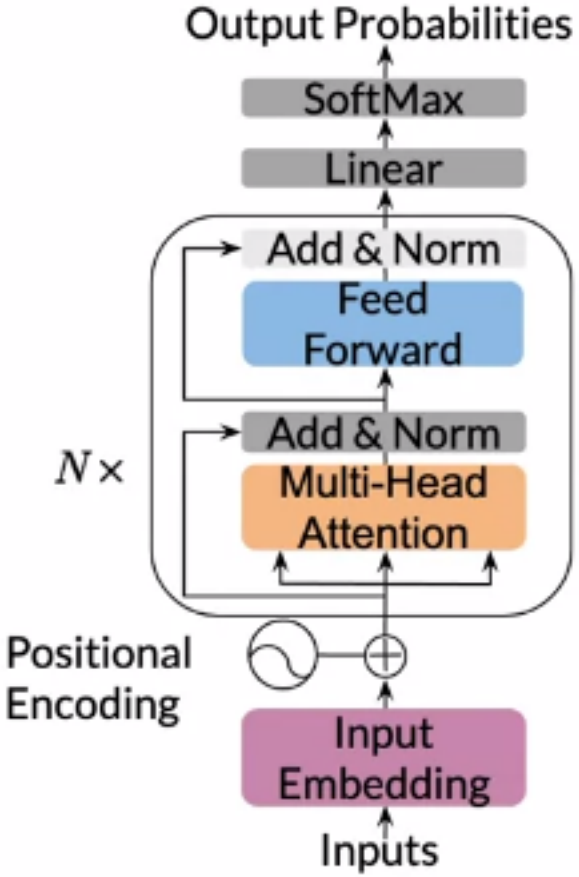
Transformer decoder



Overview

- input: sentence or paragraph
  - we predict the next word
- sentence gets embedded, add positional encoding
  - (vectors representing $\{0, 1, 2, \ldots, K\}$)
- multi-head attention looks at previous words
- feed-forward layer with ReLU
  - that's where most parameters are!

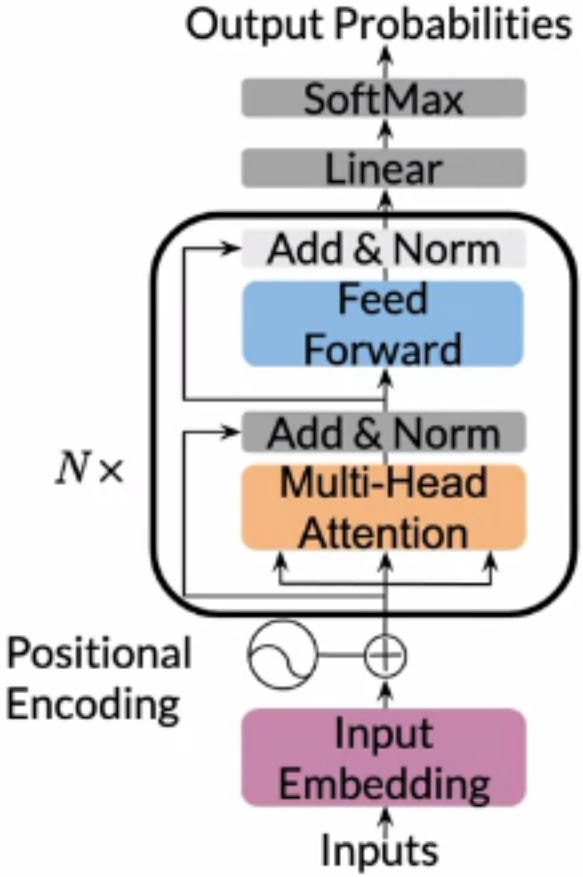
Transformer decoder



Overview

- input: sentence or paragraph
  - we predict the next word
- sentence gets embedded, add positional encoding
  - (vectors representing $\{0, 1, 2, \ldots, K\}$)
- multi-head attention looks at previous words
- feed-forward layer with ReLU
  - that's where most parameters are!
- residual connection with layer normalization

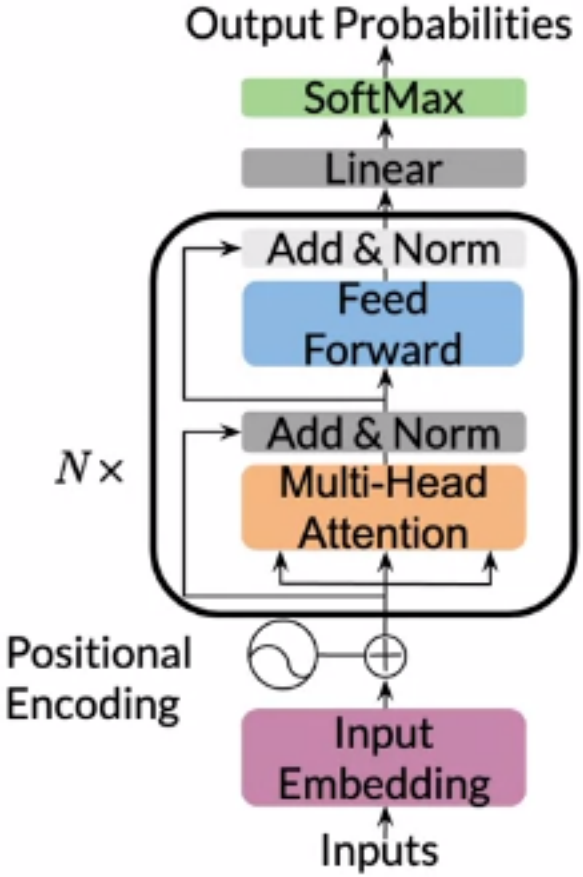
Transformer decoder



Overview

- input: sentence or paragraph
  - we predict the next word
- sentence gets embedded, add positional encoding
  - (vectors representing $\{0, 1, 2, \ldots, K\}$)
- multi-head attention looks at previous words
- feed-forward layer with ReLU
  - that's where most parameters are!
- residual connection with layer normalization
- repeat N times

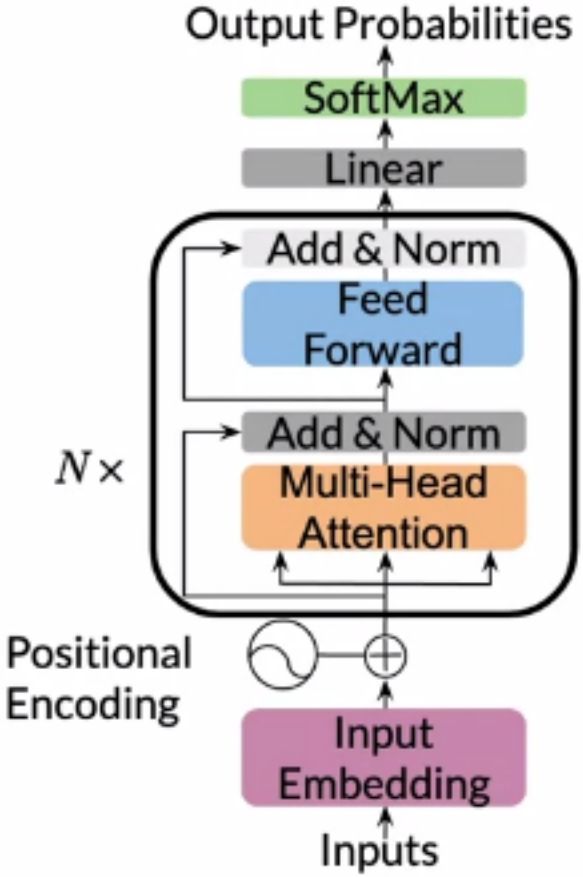
Transformer decoder



Overview

- input: sentence or paragraph
  - we predict the next word
- sentence gets embedded, add positional encoding
  - (vectors representing $\{0, 1, 2, \ldots, K\}$)
- multi-head attention looks at previous words
- feed-forward layer with ReLU
  - that's where most parameters are!
- residual connection with layer normalization
- repeat N times
- dense layer and softmax for output

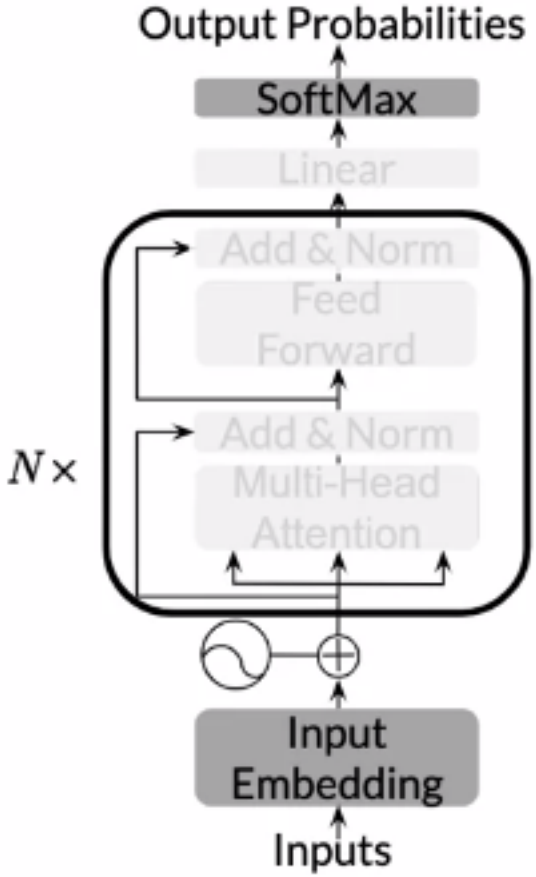
Transformer decoder



Overview

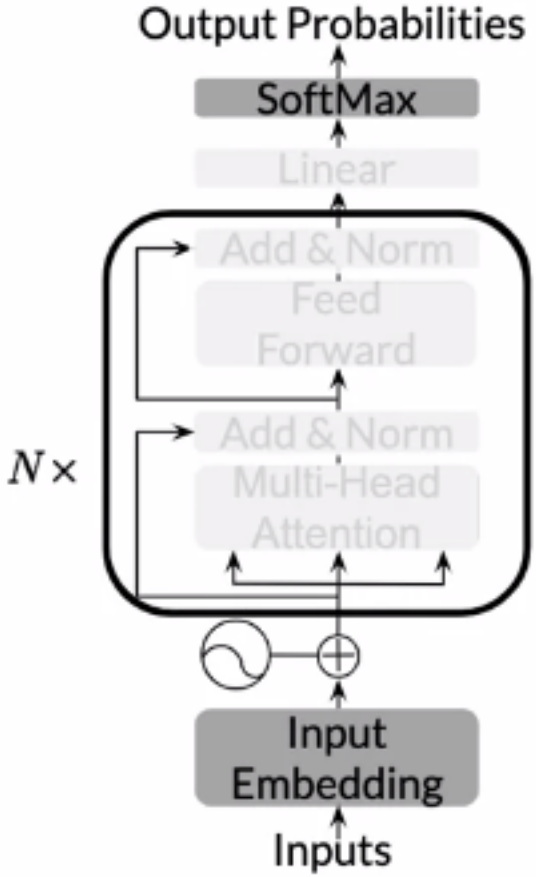
- input: sentence or paragraph
  - we predict the next word
- sentence gets embedded, add positional encoding
  - (vectors representing $\{0, 1, 2, \ldots, K\}$)
- multi-head attention looks at previous words
- feed-forward layer with ReLU
  - that's where most parameters are!
- residual connection with layer normalization
- repeat N times
- dense layer and softmax for output

Transformer decoder



Explanation

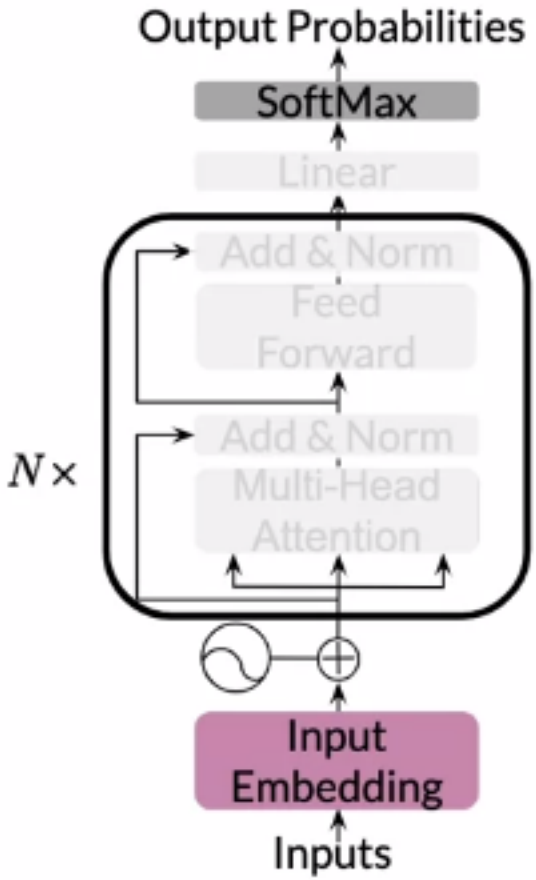
Transformer decoder



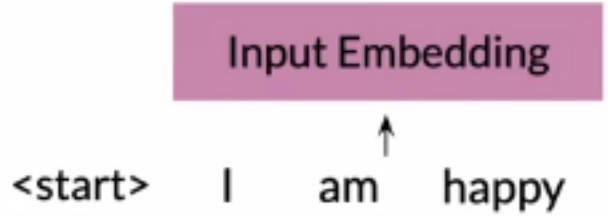
Explanation

<start> I am happy

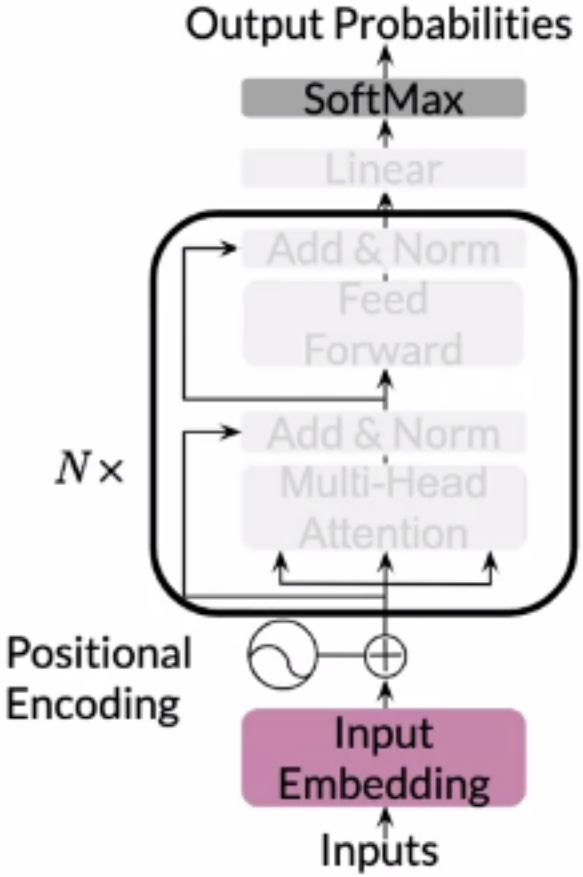
Transformer decoder



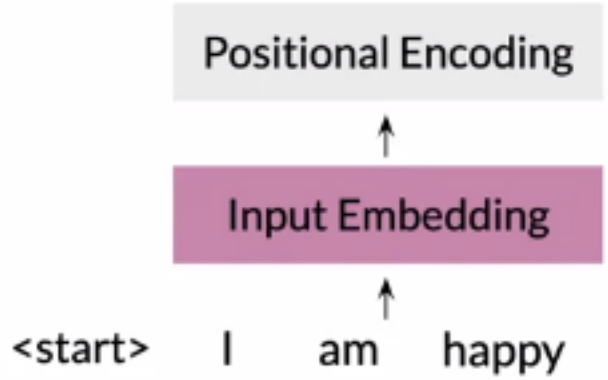
Explanation



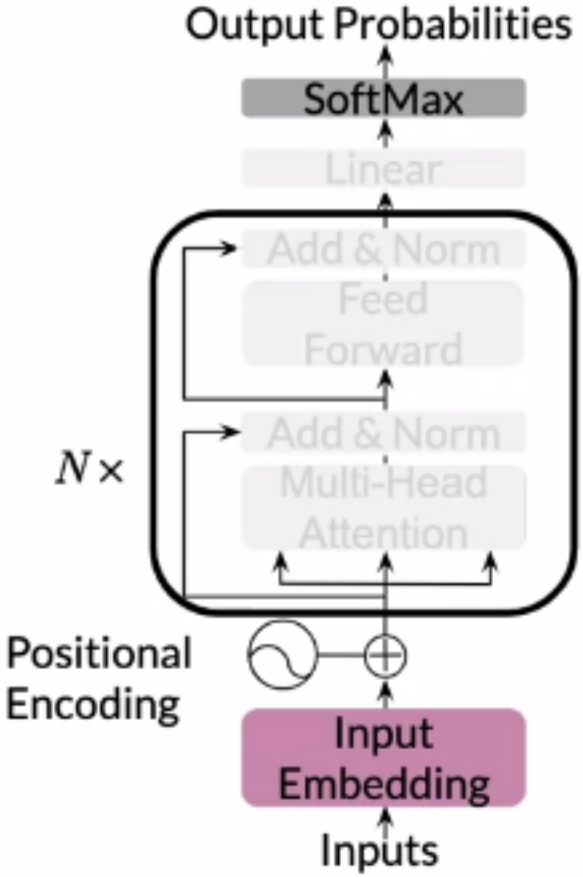
Transformer decoder



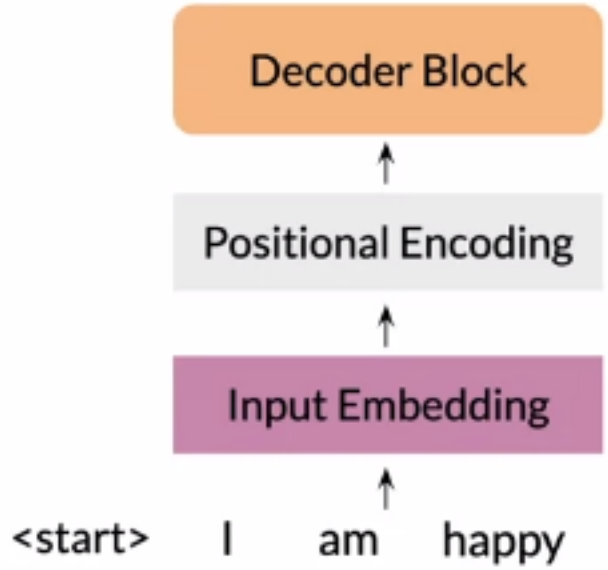
Explanation



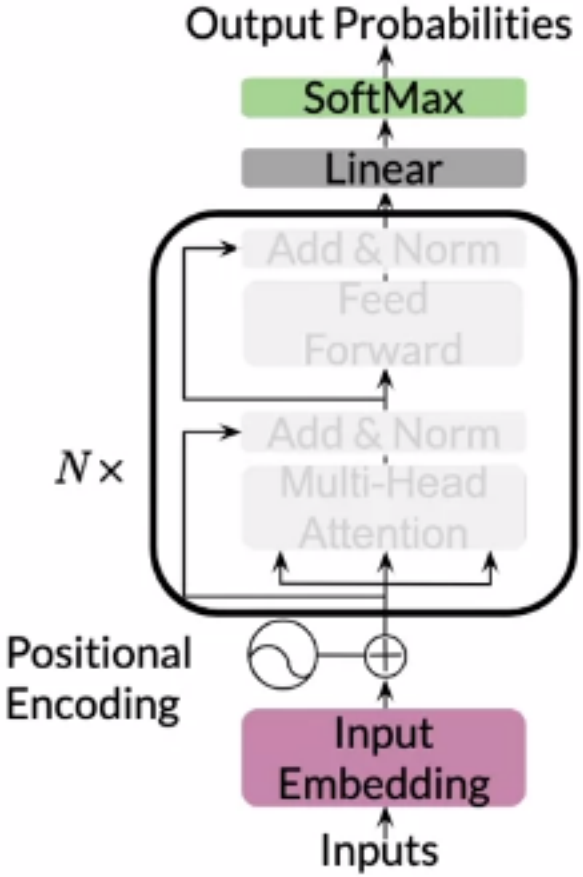
Transformer decoder



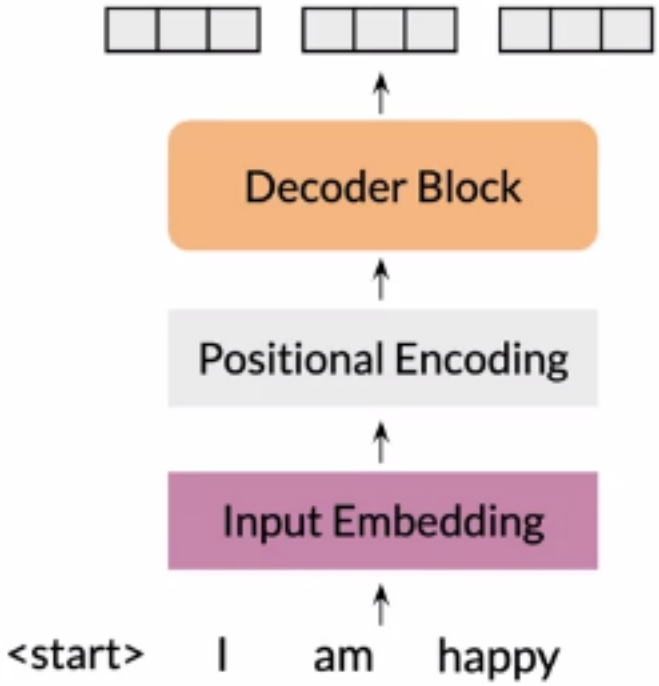
Explanation



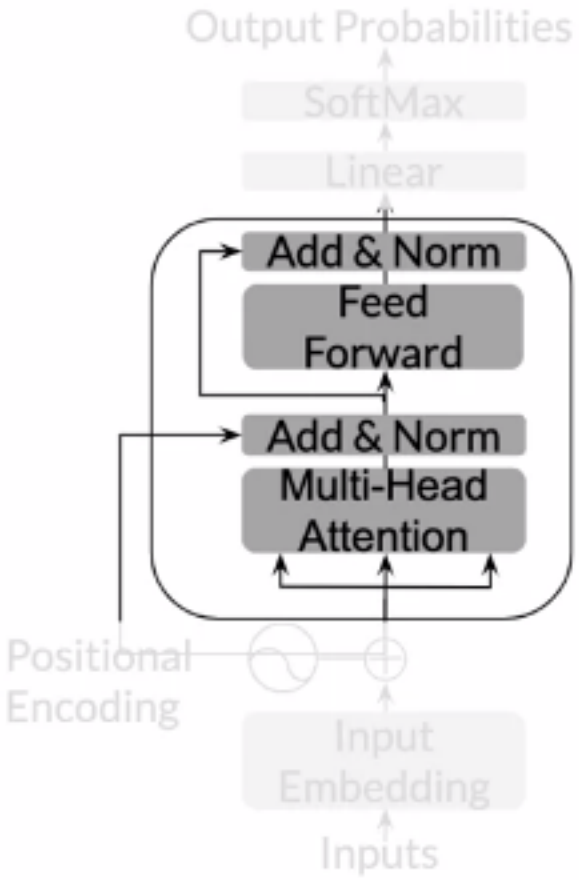
Transformer decoder



Explanation



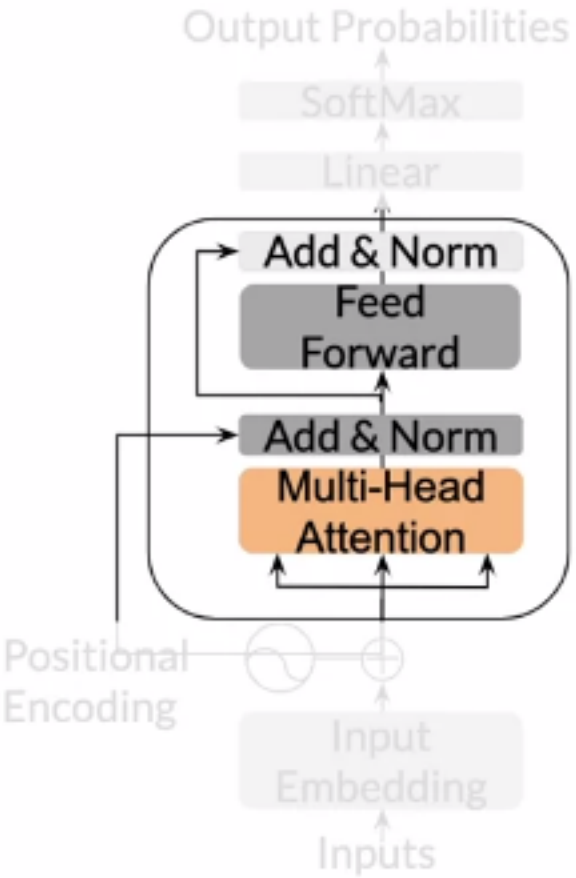
The Transformer decoder



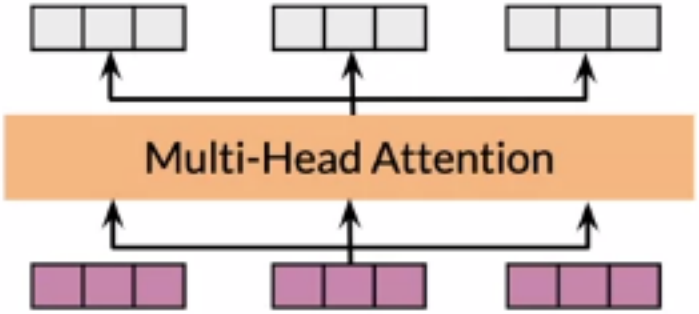
Decoder Block

Positional input embedding

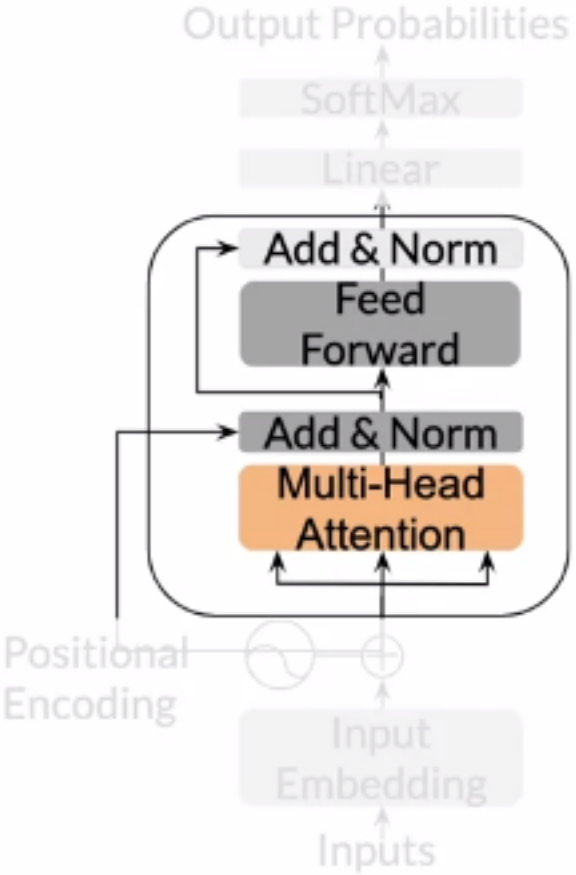
The Transformer decoder



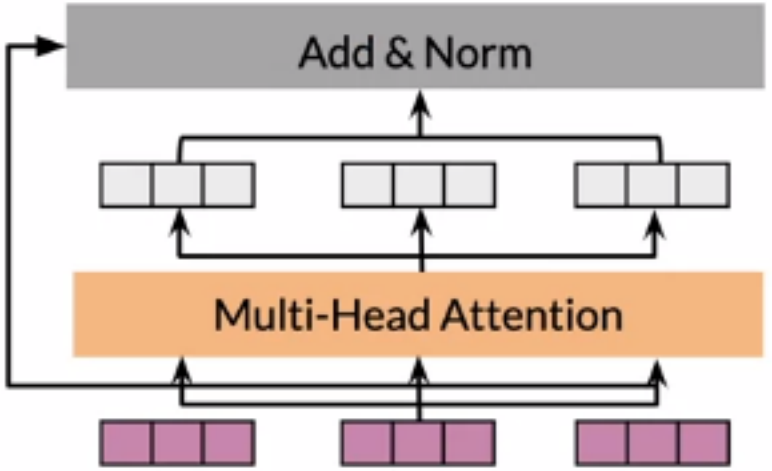
Decoder Block



The Transformer decoder



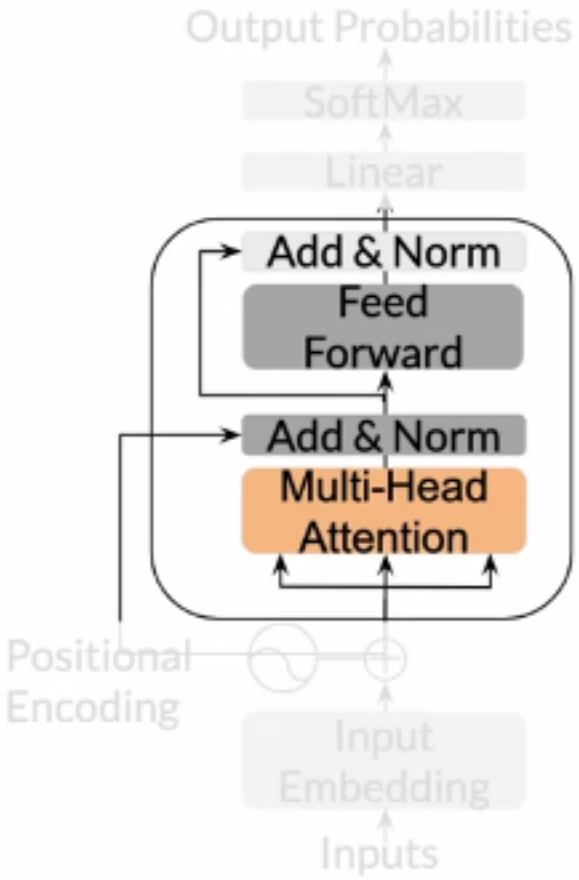
Decoder Block



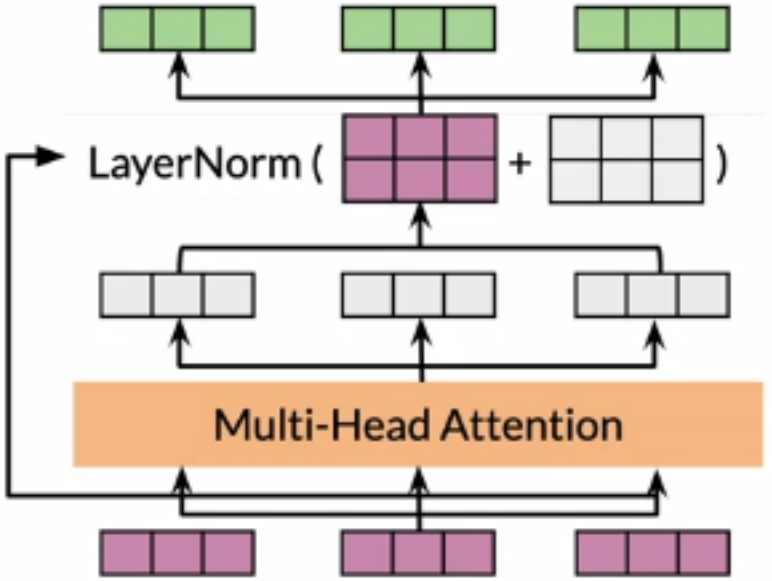
Output Vector

Positional input embedding

The Transformer decoder



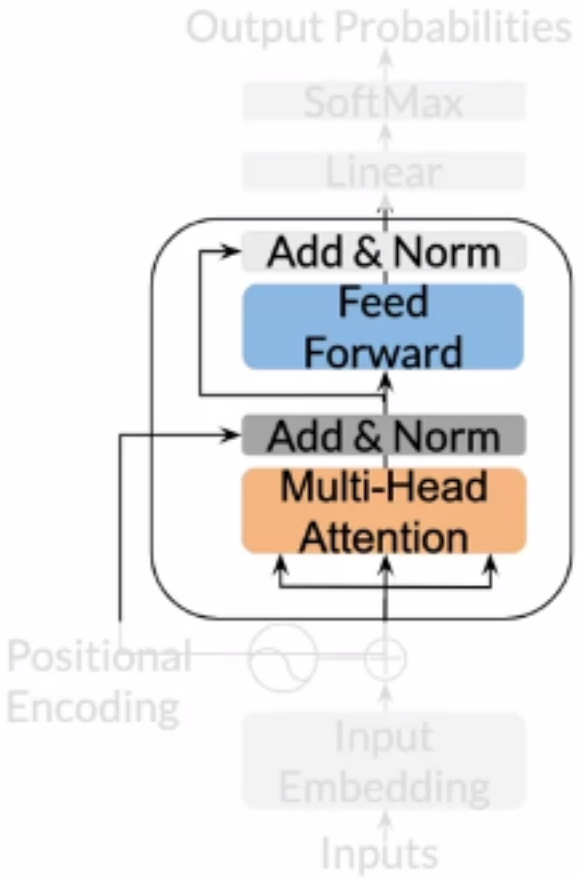
Decoder Block



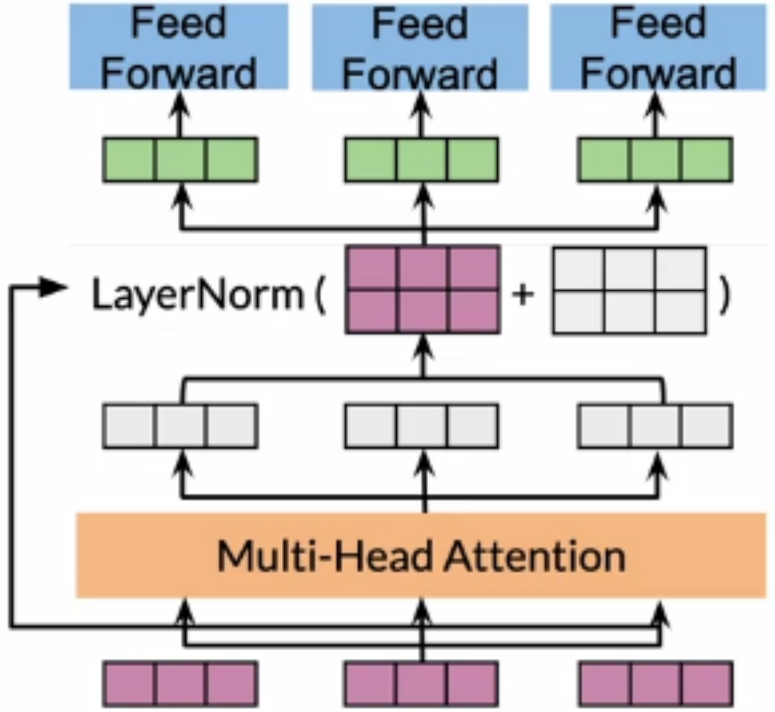
Output Vector

Positional input embedding

The Transformer decoder



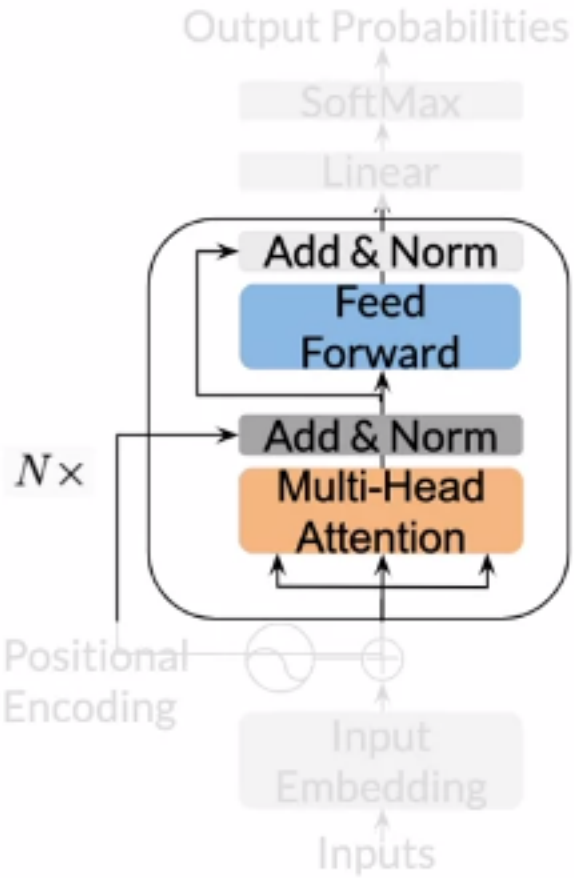
Decoder Block

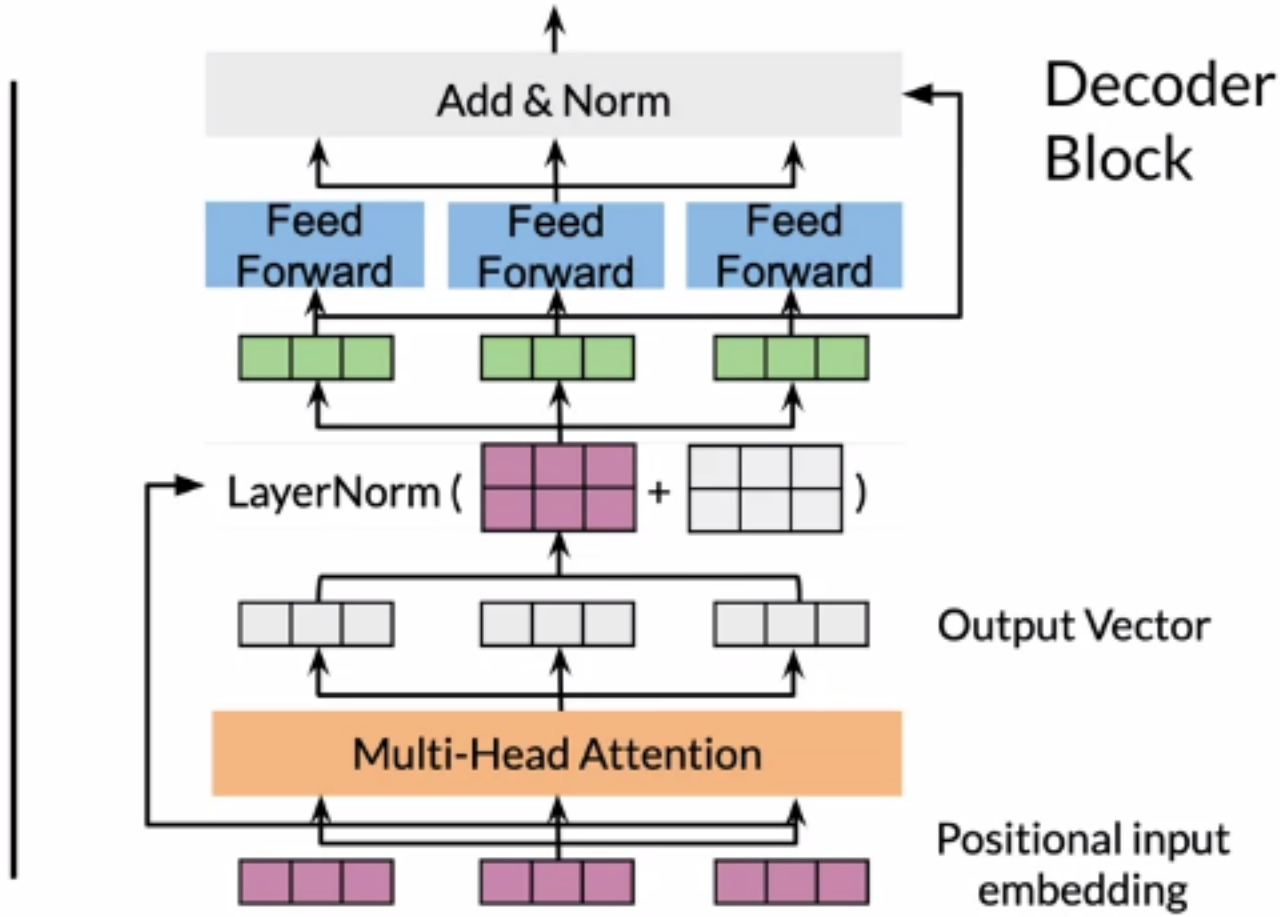


Output Vector

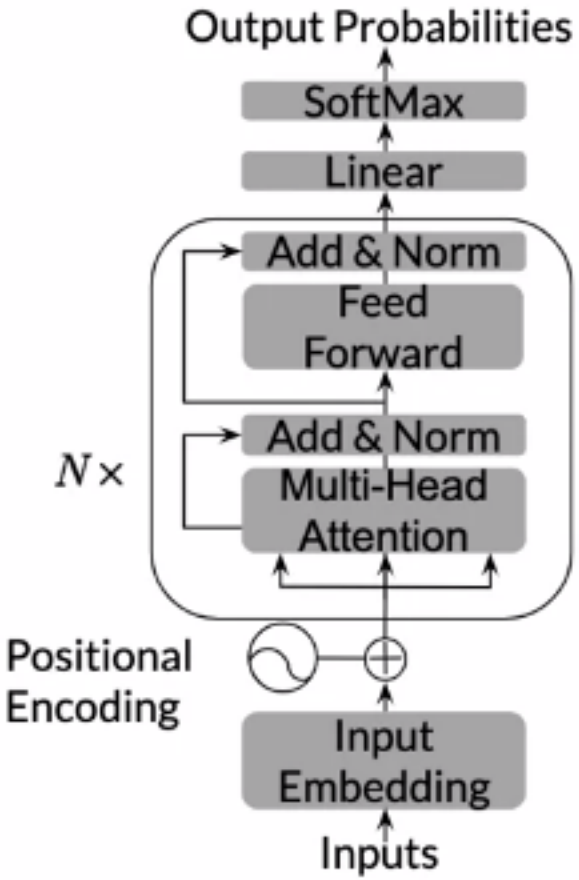
Positional input embedding

The Transformer decoder

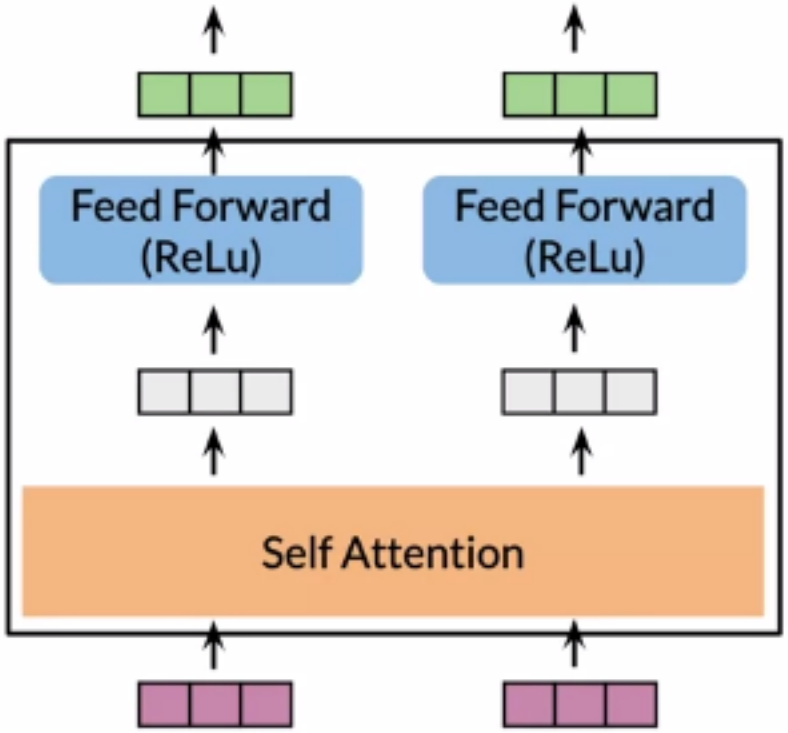




The Transformer decoder



Feed forward layer

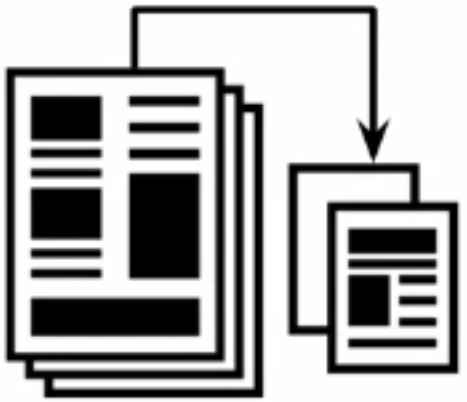


Summary

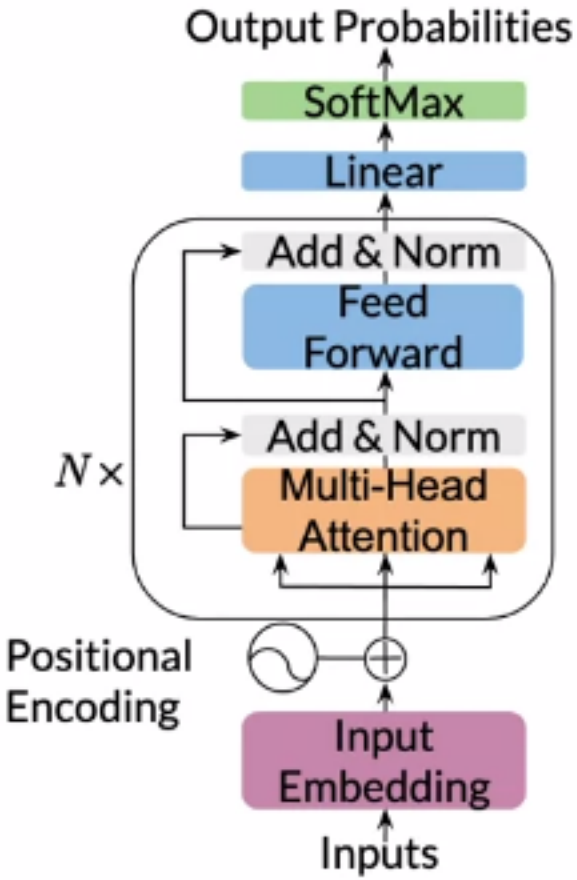
- Transformer decoder mainly consists of three layers
- Decoder and feed-forward blocks are the core of this model code
- It also includes a module to calculate the cross-entropy loss

Outline

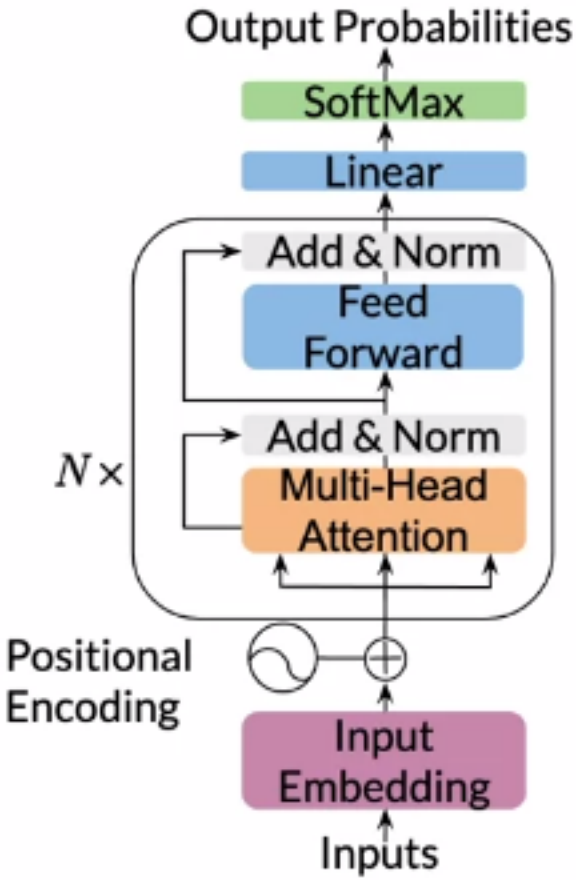
- Overview of Transformer summarizer
- Technical details for data processing
- Inference with a Language Model



Transformer for summarization

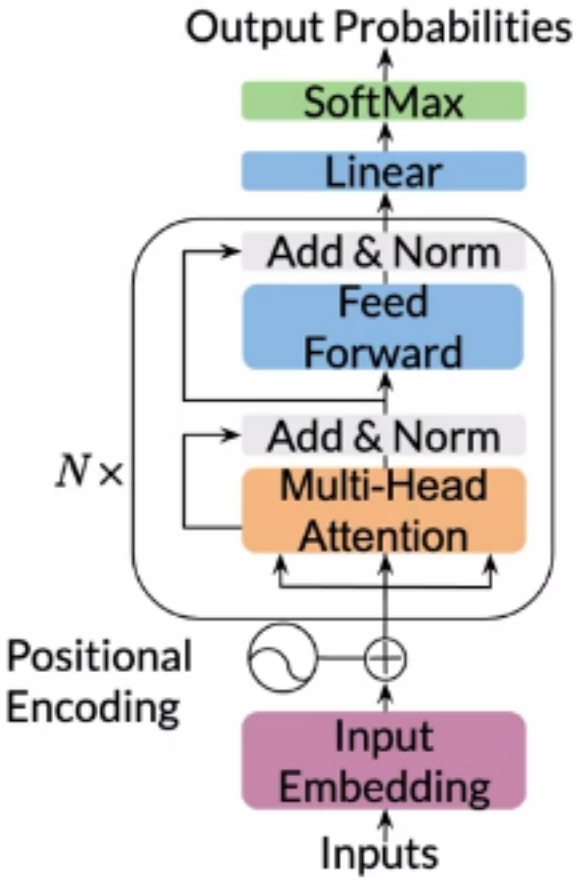


Transformer for summarization



Input

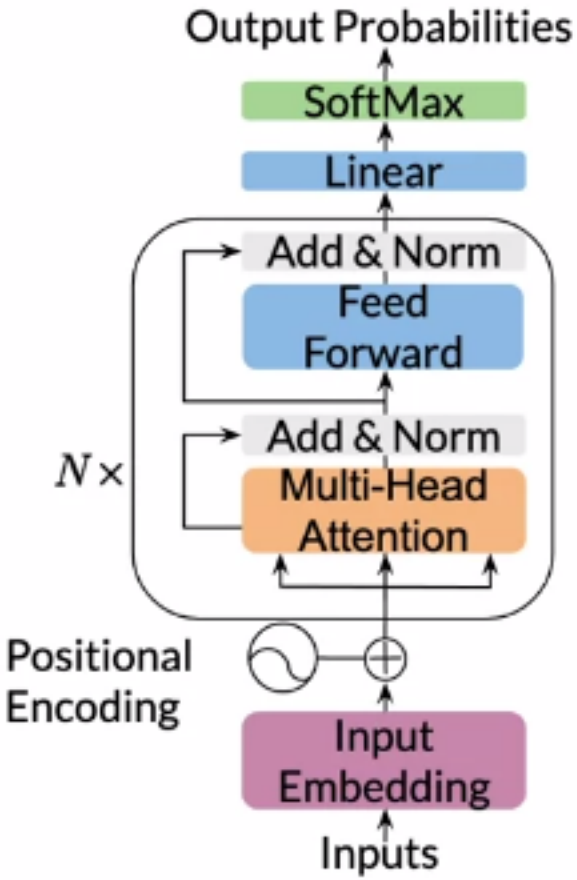
Transformer for summarization



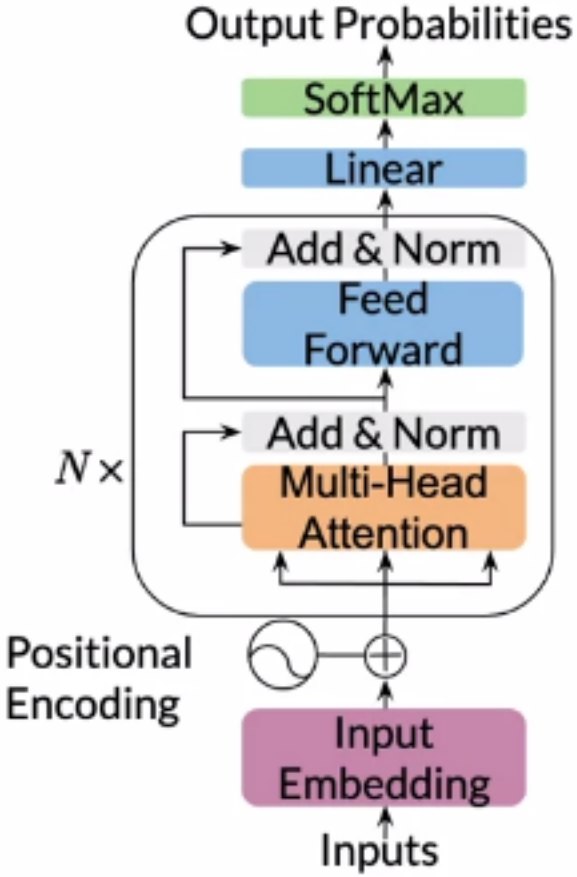
Input

Output: Summary

Technical details for data processing



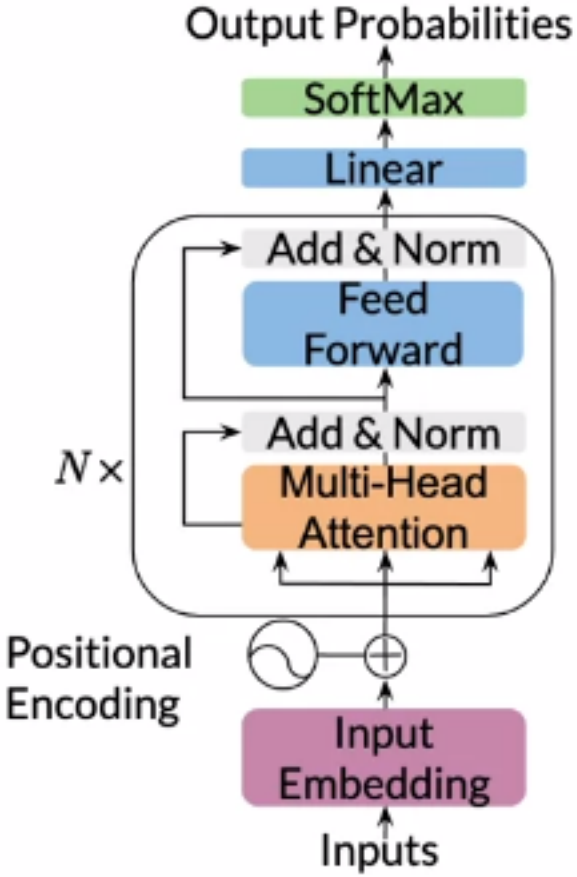
Technical details for data processing



Model Input:

```
ARTICLE TEXT <EOS> SUMMARY <EOS> <pad> ...
```

Technical details for data processing



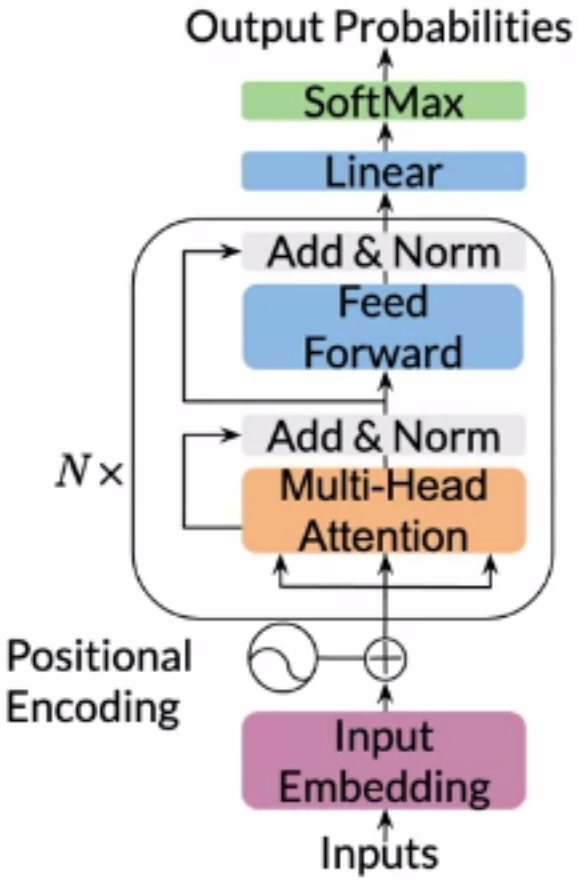
Model Input:

```
ARTICLE TEXT <EOS> SUMMARY <EOS> <pad> ...
```

Tokenized version:

```
[2,3,5,2,1,3,4,7,8,2,5,1,2,3,6,2,1,0,0]
```

Technical details for data processing



Model Input:

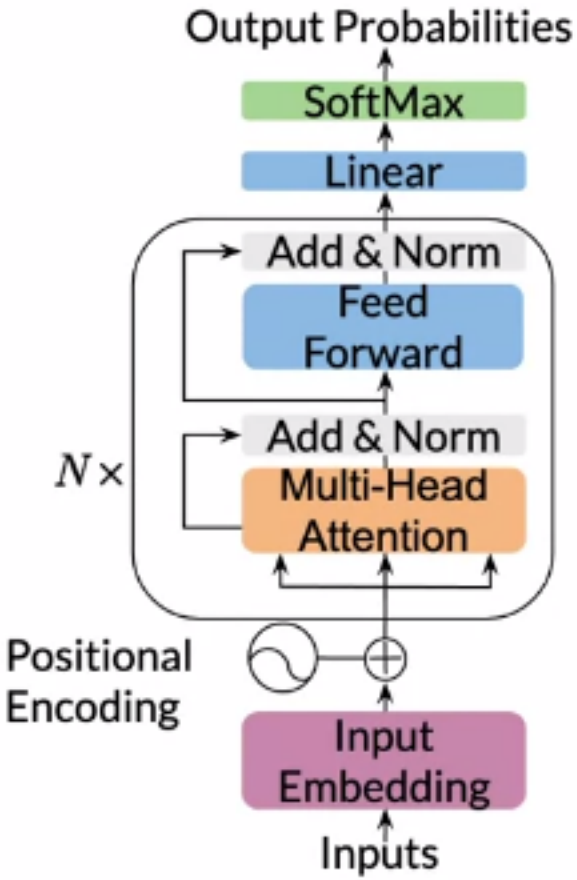
```
ARTICLE TEXT <EOS> SUMMARY <EOS> <pad> ...
```

Tokenized version:

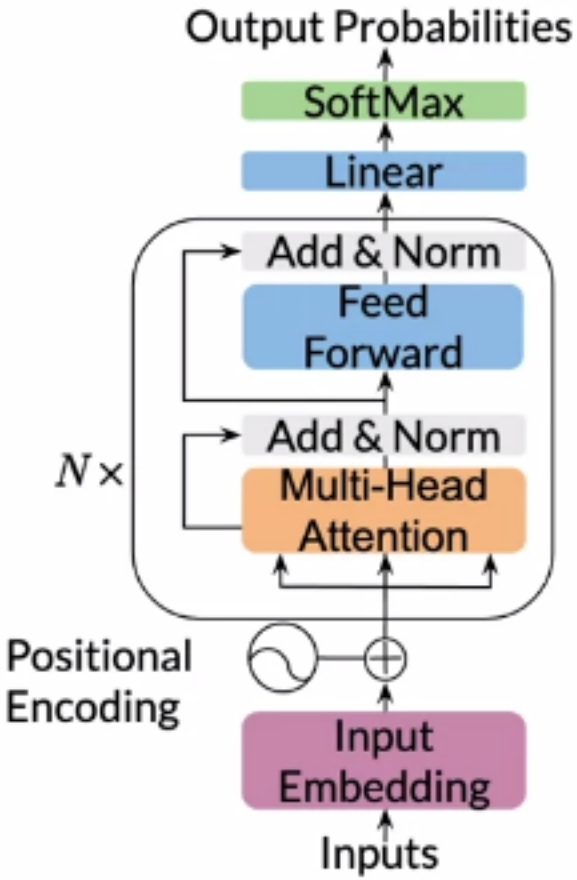
```
[2,3,5,2,1,3,4,7,8,2,5,1,2,3,6,2,1,0,0]
```

Loss weights: 0s until the first `<EOS>` and then 1 on the start of the summary.

Cost function

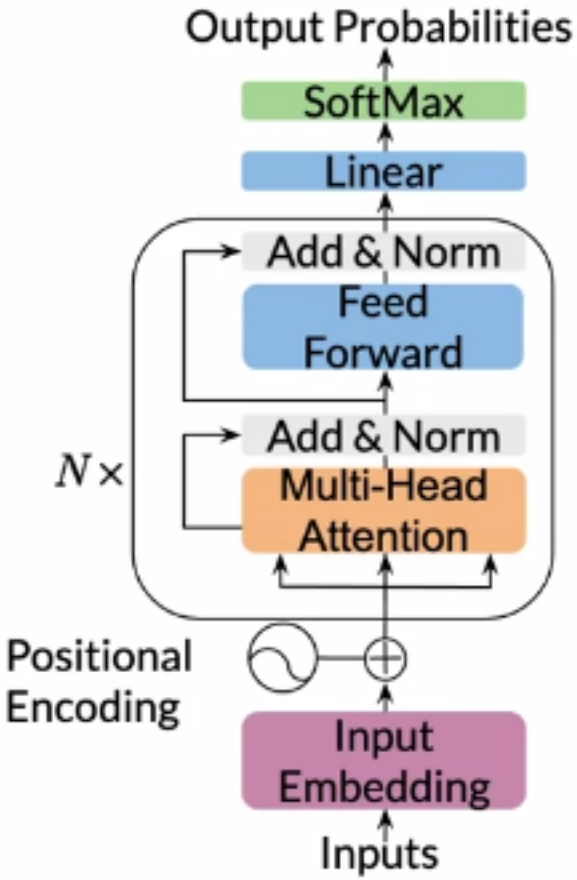


Cost function



$$J = -\frac{1}{m}\sum_{j}^{m}\sum_{i}^{K} y_j^i \log \hat{y}_j^i$$

Cost function

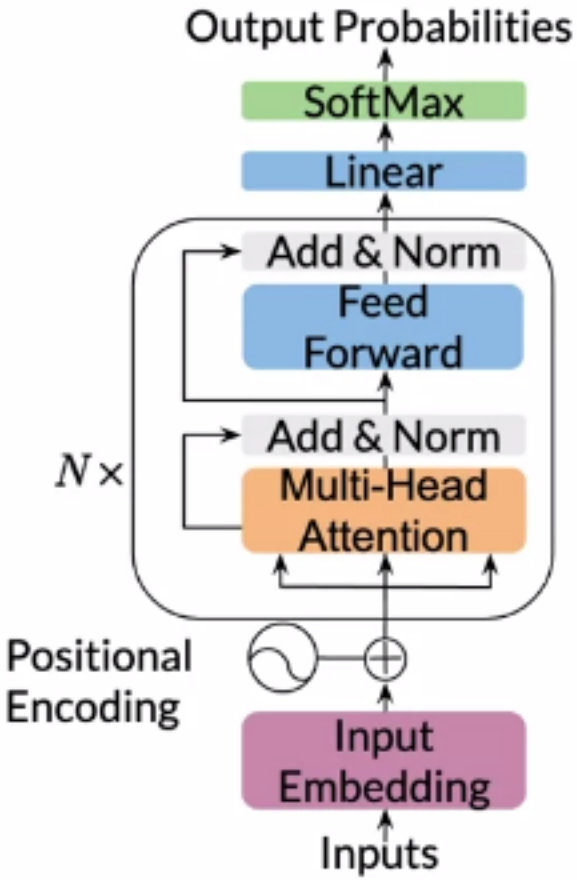


$$J = -\frac{1}{m}\sum_{j}^{m}\sum_{i}^{K} y_j^i \log \hat{y}_j^i$$

j : over summary

i : bach elements

Cost function



Cross entropy loss

$$J = -\frac{1}{m}\sum_{j}^{m}\sum_{i}^{K} y_j^i \log \hat{y}_j^i$$

j : over summary

i : bach elements

Inference with a Language Model

Inference with a Language Model

Model input:

```
[Article] <EOS>  [Summary] <EOS>
```

Inference with a Language Model

Model input:

```
[Article] <EOS>  [Summary] <EOS>
```

Inference:

Inference with a Language Model

Model input:

`[Article] <EOS>  [Summary] <EOS>`

Inference:

- Provide: `[Article] <EOS>`

Inference with a Language Model

Model input:

`[Article] <EOS> [Summary] <EOS>`

Inference:

- Provide: `[Article] <EOS>`

- Generate summary word-by-word
  - until the final `<EOS>`

Inference with a Language Model

Model input:

`[Article] <EOS> [Summary] <EOS>`

Inference:

- Provide: [Article] <EOS>

- Generate summary word-by-word
  - until the final <EOS>

- Pick the next word by random sampling
  - each time you get a different summary!

Summary

- For summarization, a weighted loss function is optimized
- Transformer Decoder summarizes predicting the next word using
- The transformer uses tokenized versions of the input