Natural Language Processing (CS-472)
Spring-2023

Muhammad Naseer Bajwa
Assistant Professor,
Department of Computing, SEECS
Co-Principal Investigator,
Deep Learning Lab, NCAI
NUST, Islamabad
naseer.bajwa@seecs.edu.pk

NUST
Defining futures
School of Electrical Engineering
& Computer Science

Overview of this week's lecture

Transformers

- Rationale
- Positional Encoding
- Self Attention
- Encoder-Decoder Architecture

What architectures have we studied so far?

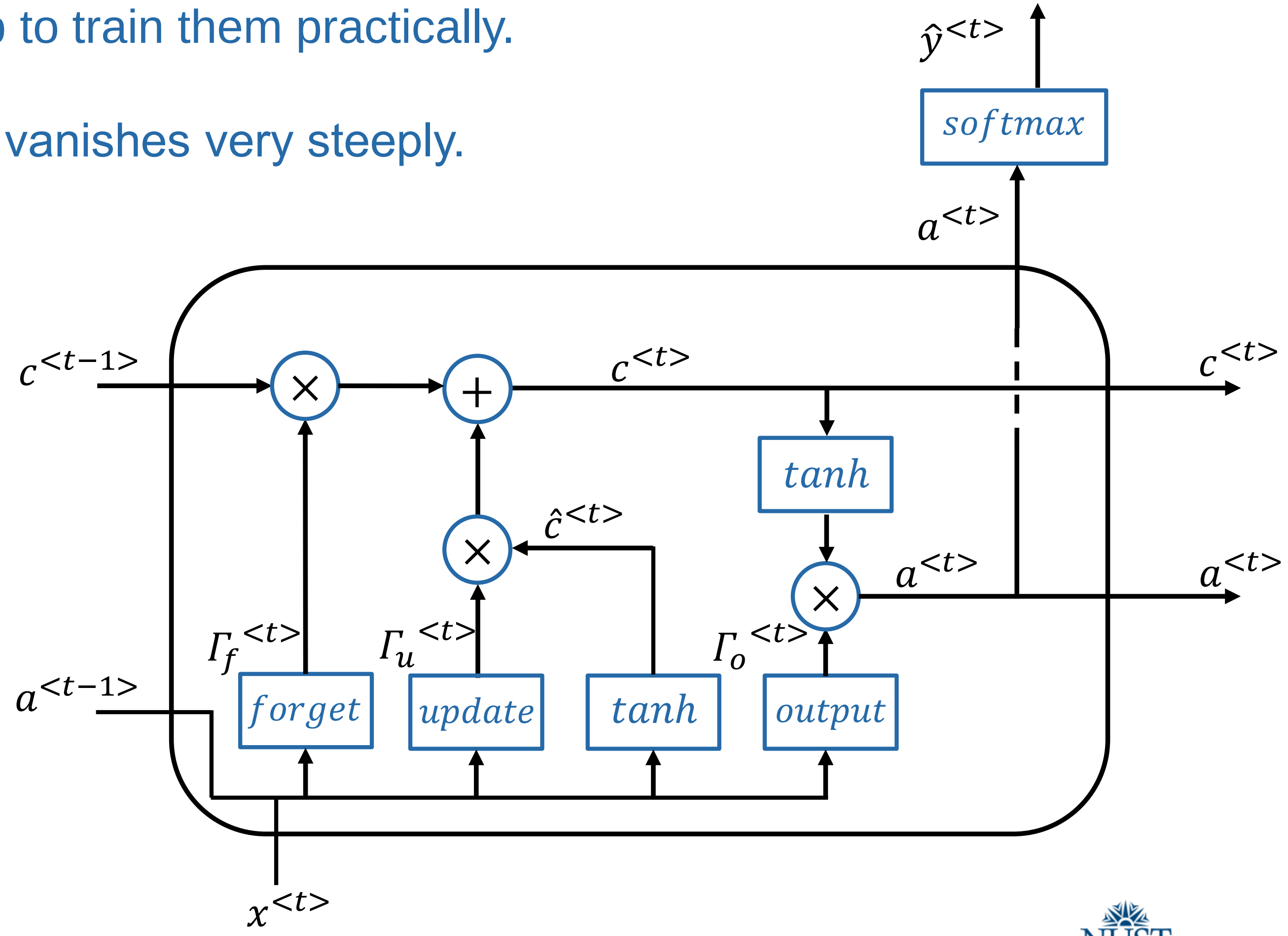
- RNNs are great at handling temporal dependencies but they have some disadvantages.

What architectures have we studied so far?

- RNNs are great at handling temporal dependencies but they have some disadvantages.
    - Slower to train. We need truncated backprop to train them practically.
    - Can’t handle very long sequences. Gradient vanishes very steeply.

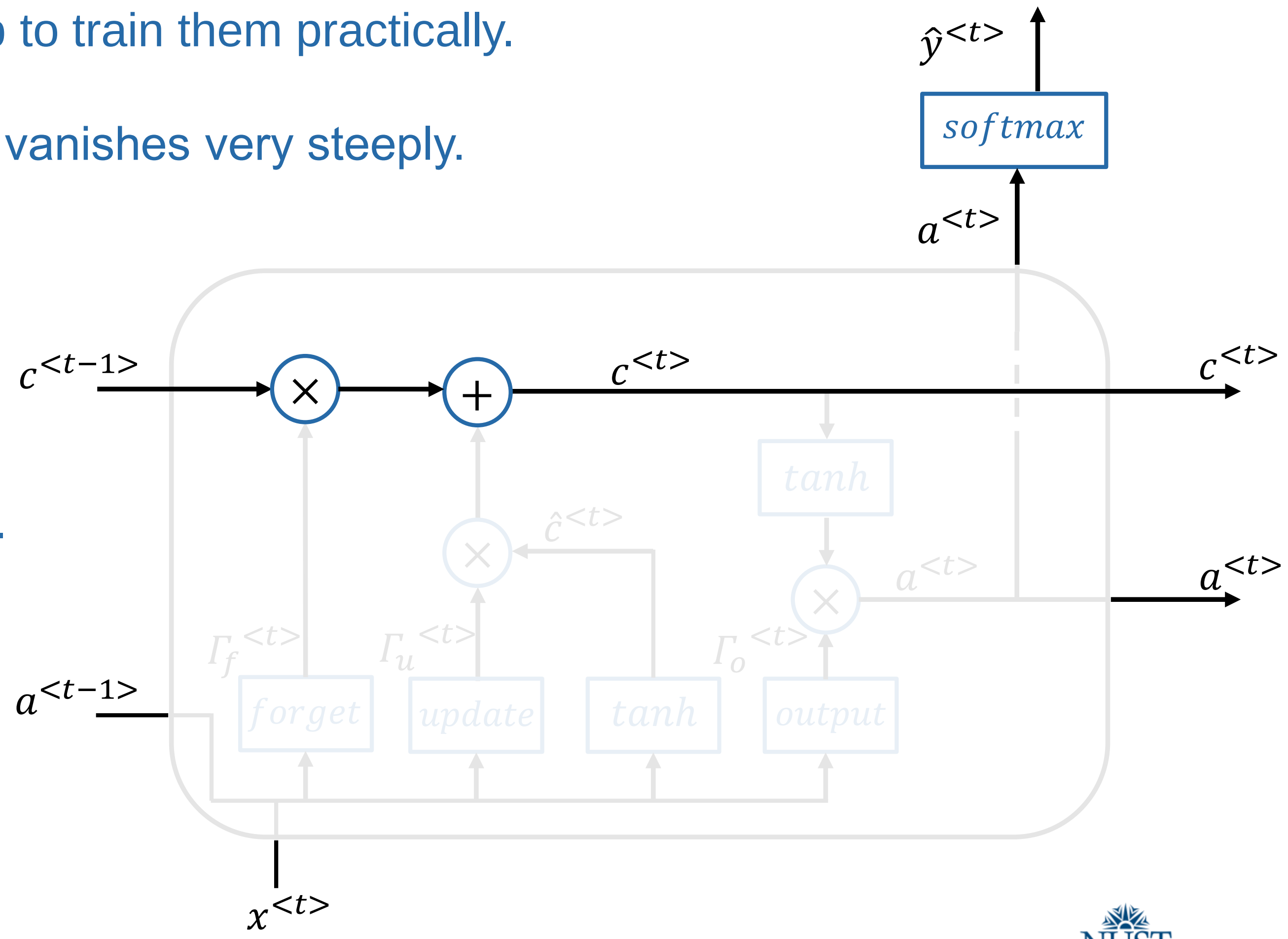
What architectures have we studied so far?

- RNNs are great at handling temporal dependencies but they have some disadvantages.
  - Slower to train. We need truncated backprop to train them practically.
  - Can't handle very long sequences. Gradient vanishes very steeply.
- LSTMs can handle longer sequences better.

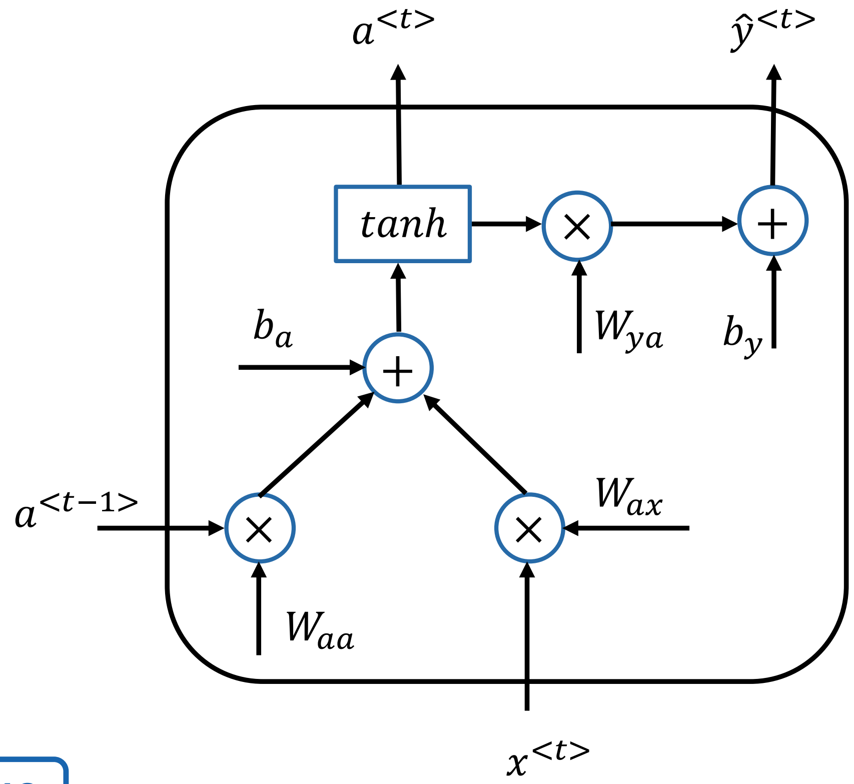


What architectures have we studied so far?

- RNNs are great at handling temporal dependencies but they have some disadvantages.
  - Slower to train. We need truncated backprop to train them practically.
  - Can't handle very long sequences. Gradient vanishes very steeply.
- LSTMs can handle longer sequences better.
  - Past information can pass through the cell unhindered via a special branch.
  - But they are even slower and more complex.

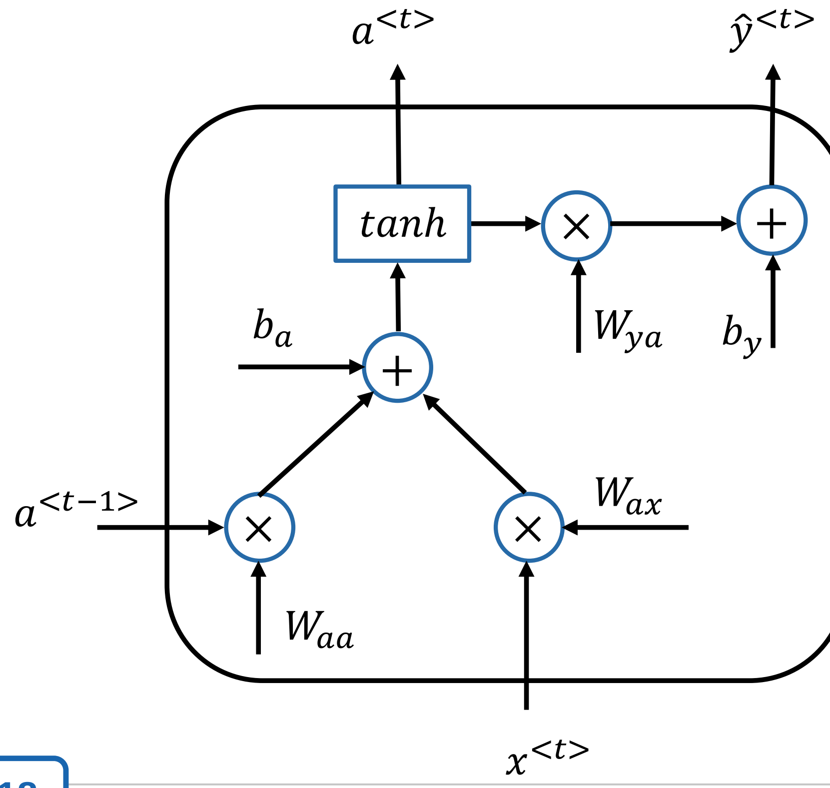


- RNNs are great at handling temporal dependencies but they have some disadvantages.
    - Slower to train. We need truncated backprop to train them practically.
    - Can’t handle very long sequences. Gradient vanishes very steeply.
- LSTMs can handle longer sequences better.
    - Past information can pass through the cell unhindered via a special branch.
    - But they are even slower and more complex.
- GRUs are faster and can handle longer sequences.
    - Still not enough.



What architectures have we studied so far?

- RNNs are great at handling temporal dependencies but they have some disadvantages.
    - Slower to train. We need truncated backprop to train them practically.
    - Can’t handle very long sequences. Gradient vanishes very steeply.
- LSTMs can handle longer sequences better.
    - Past information can pass through the cell unhindered via a special branch.
    - But they are even slower and more complex.
- GRUs are faster and can handle longer sequences.
    - Still not enough. **Need parallelisation.**



How can we achieve parallelisation for sequential data?

How can we achieve parallelisation for sequential data?

How can we achieve parallelisation for sequential data?

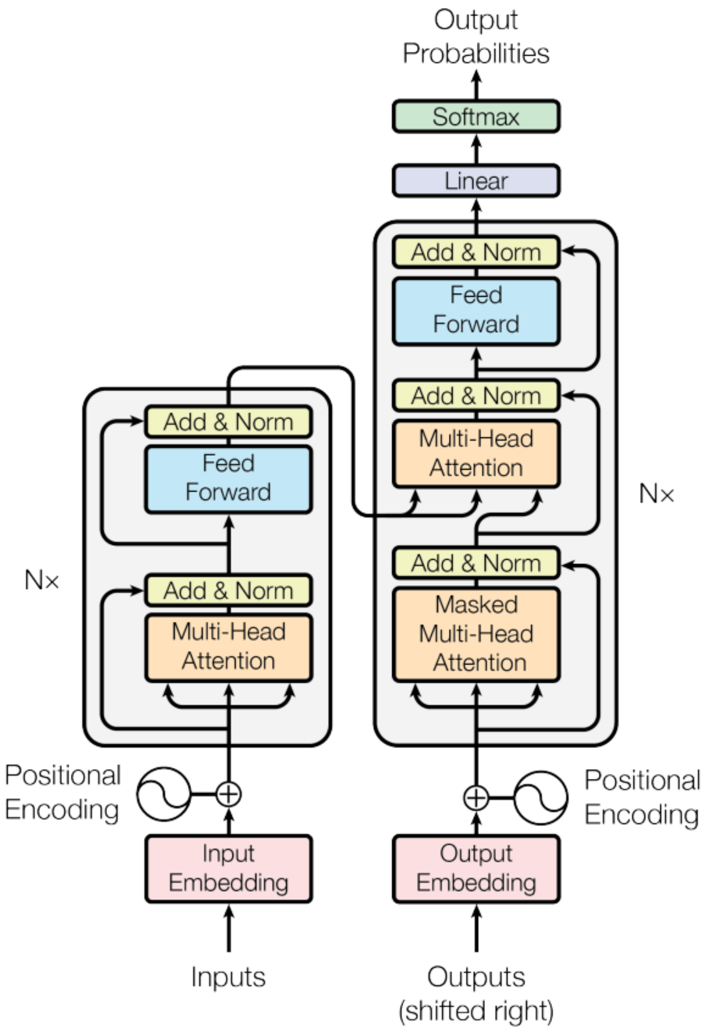


Figure 1: The Transformer - model architecture.

Transformers use encoder-decoder architecture without RNNs

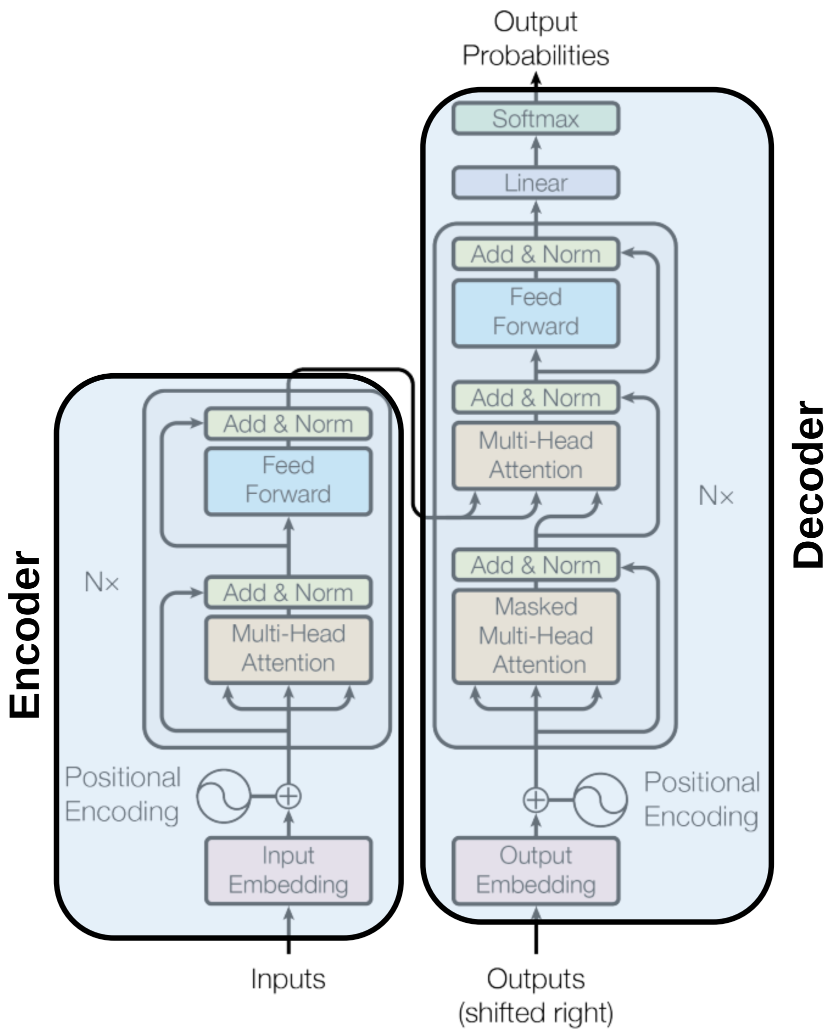


Figure 1: The Transformer - model architecture.

Transformers use encoder-decoder architecture without RNNs

- **The Encoder**
  - Accepts input as text.
  - Gives output as word embeddings.
  - Uses Self-Attention.
  - Is bidirectional.

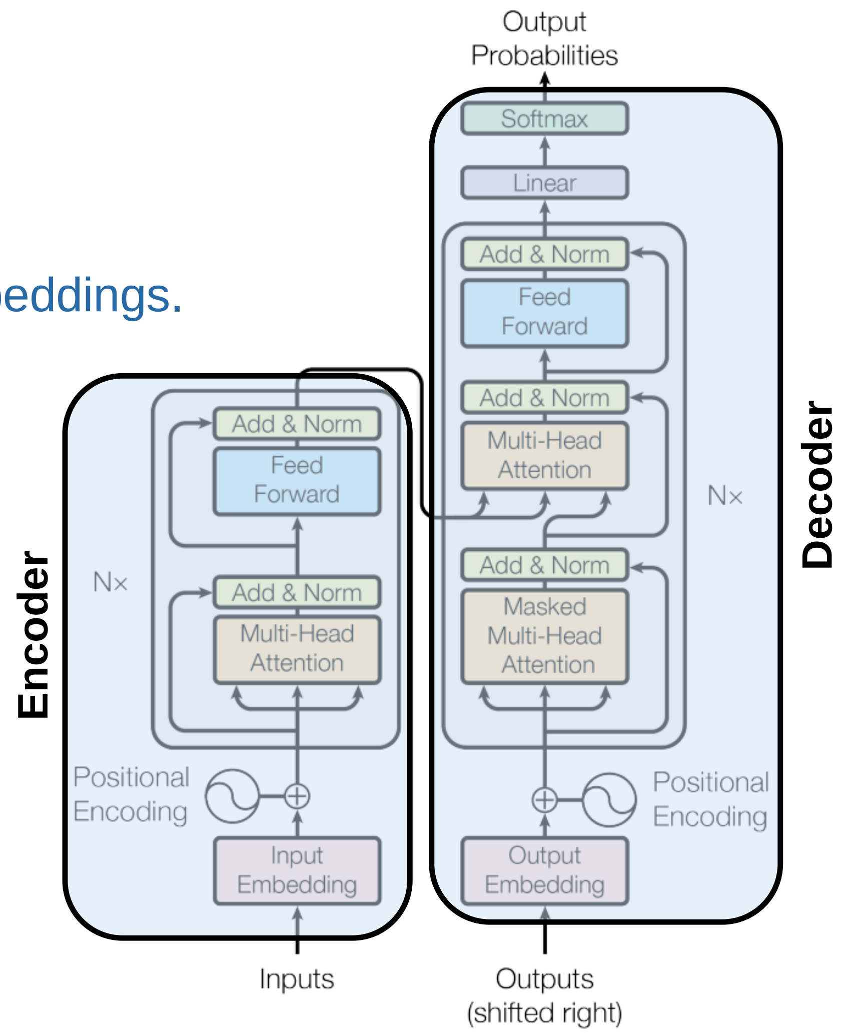


Figure 1: The Transformer - model architecture.

Transformers use encoder-decoder architecture without RNNs

- **The Encoder**
  - Accepts input as text.
  - Gives output as word embeddings.
  - Uses Self-Attention.
  - Is bidirectional.

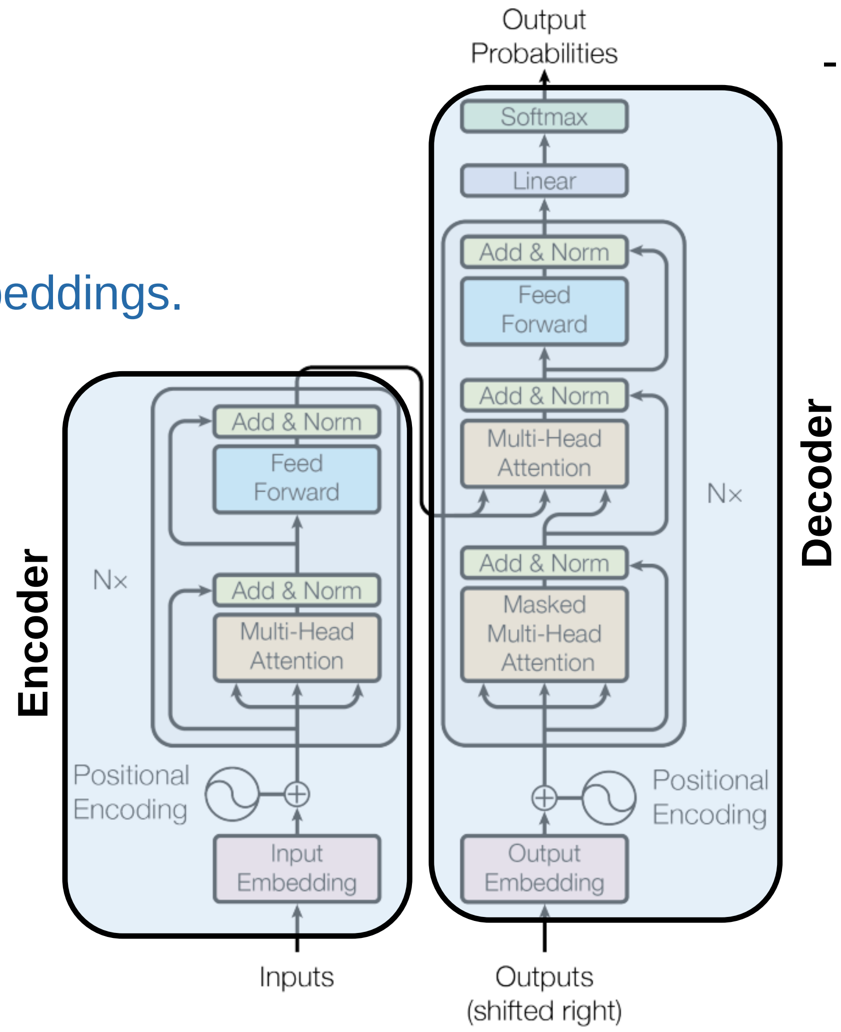
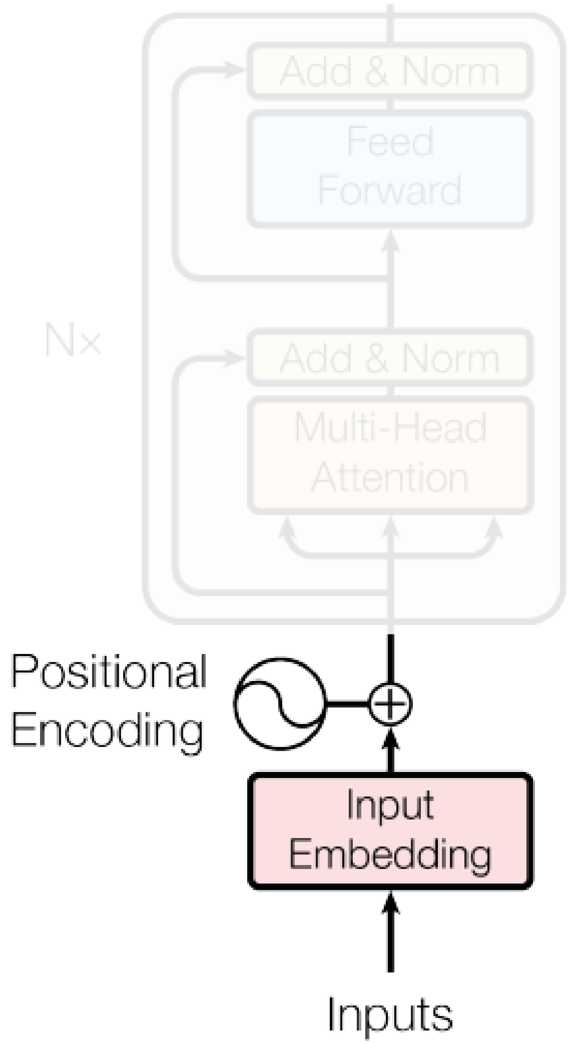


Figure 1: The Transformer - model architecture.

- **The Decoder**
  - Accepts input as word.
  - Gives output as sequence of words.
  - Uses Masked Self-Attention.
  - Is unidirectional.

Transformers use positional encoders to get position-aware embeddings

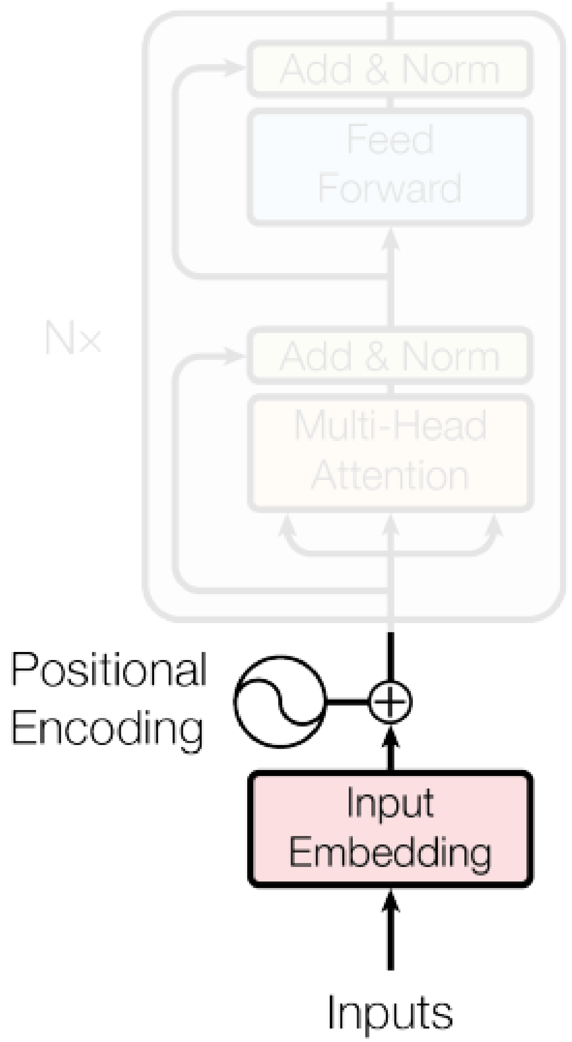
- Positional Encoder uses a vector that gives context based on position of words in a sentence.



Transformers use positional encoders to get position-aware embeddings

- Positional Encoder uses a vector that gives context based on position of words in a sentence.

- The vector has the same length as the embeddings to allow summation.



Transformers use positional encoders to get position-aware embeddings

- Positional Encoder uses a vector that gives context based on position of words in a sentence.

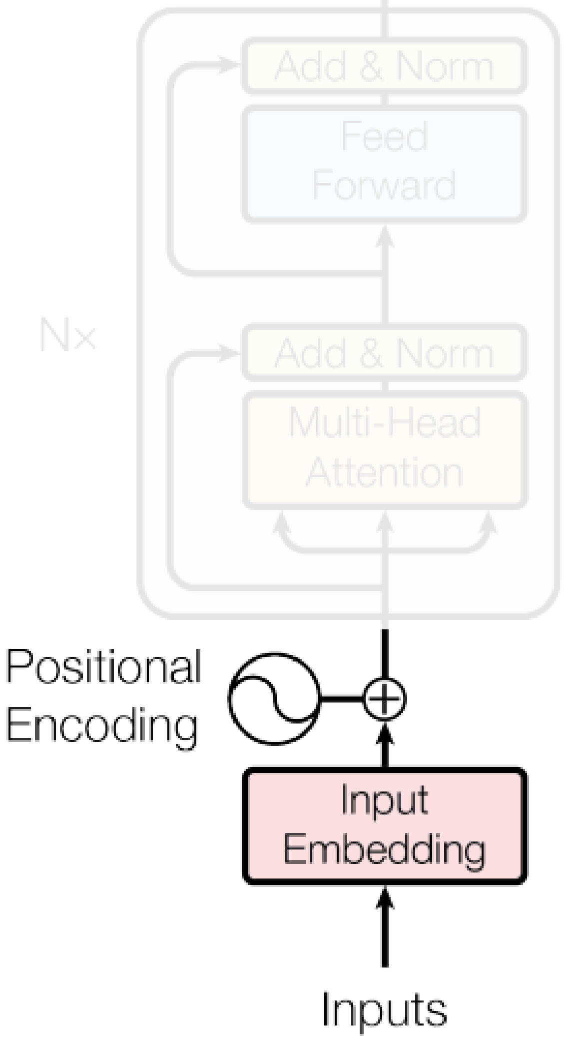
- The vector has the same length as the embeddings to allow summation.

- Positional encodings can be fixed or learned. In the original transformer paper, they used $sine$ and $cosine$ functions of different frequencies.

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

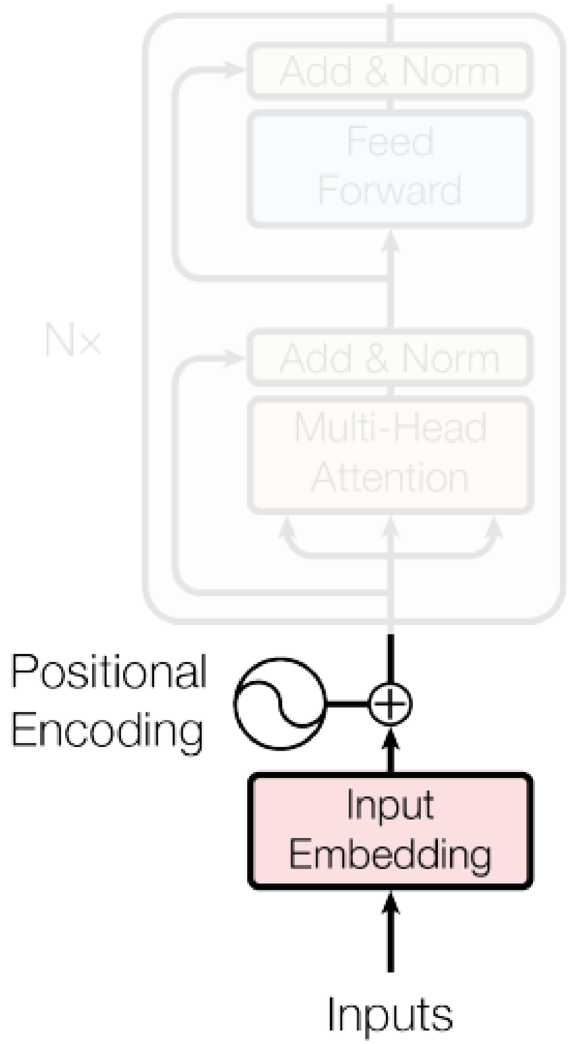
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\left(\frac{2i}{d_{model}}\right)}}\right)$$

  - Here pos is position, i represents dimension and d_{model} corresponds to the total length of input.



How Positional Encodings are calculated?

The		p0	Large		p1	Red		p2	Dog		p3
0.2		?	-0.9		?	0.5		?	0.6		?
-0.8		?	0.3		?	0.6		?	0.0		?
-0.6	+	?	0.4	+	?	-0.8	+	?	0.8	+	?
0.9		?	0.6		?	0.7		?	0.9		?
0.1		?	-0.1		?	0.7		?	0.4		?

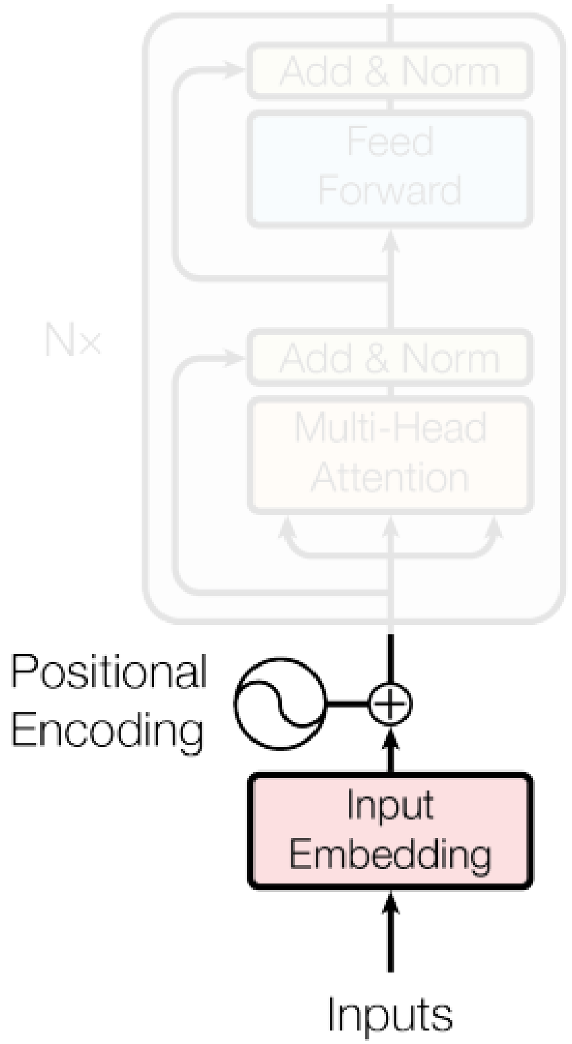


https://kazemnejad.com/blog/transformer_architecture_positional_encoding/
https://www.youtube.com/watch?v=dichIcUZfOw

How Positional Encodings are calculated?

The		p0	Large		p1	Red		p2	Dog		p3
0.2		?	-0.9		?	0.5		?	0.6		?
-0.8		?	0.3		?	0.6		?	0.0		?
-0.6	+	?	0.4	+	?	-0.8	+	?	0.8	+	?
0.9		?	0.6		?	0.7		?	0.9		?
0.1		?	-0.1		?	0.7		?	0.4		?

The-p	Large-p	Red-p	Dog-p
0.2+?	-0.9+?	0.5+?	0.6+?
-0.8+?	0.3+?	0.6+?	0.0+?
-0.6+?	0.4+?	-0.8+?	0.8+?
0.9+?	0.6+?	0.7+?	0.9+?
0.1+?	-0.1+?	0.7+?	0.4+?



How Positional Encodings are calculated?

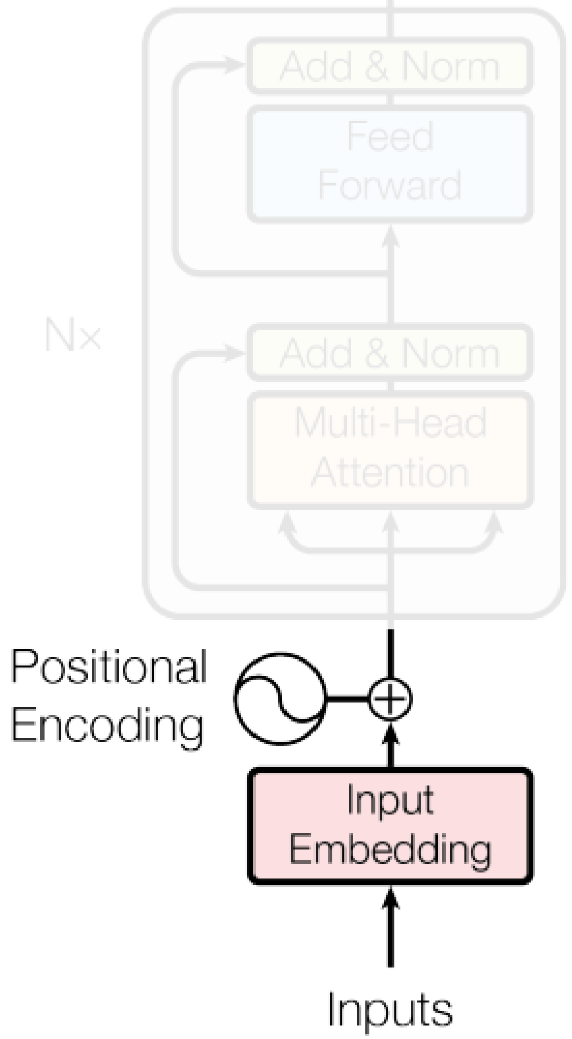
The		p0	Large		p1	Red		p2	Dog		p3
0.2	+	?	-0.9	+	?	0.5	+	?	0.6	+	?
-0.8		?	0.3		?	0.6		?	0.0		?
-0.6		?	0.4		?	-0.8		?	0.8		?
0.9		?	0.6		?	0.7		?	0.9		?
0.1		?	-0.1		?	0.7		?	0.4		?

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

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

p0

$$\begin{bmatrix} PE_{(0,0)} \\ PE_{(0,1)} \\ PE_{(0,2)} \\ PE_{(0,3)} \\ PE_{(0,4)} \end{bmatrix} \quad \begin{bmatrix} \sin\left(\frac{0}{10000^{\left(\frac{2\times 0}{5}\right)}}\right) \\ \cos\left(\frac{0}{10000^{\left(\frac{2\times 0}{5}\right)}}\right) \\ \sin\left(\frac{0}{10000^{\left(\frac{2\times 1}{5}\right)}}\right) \\ \cos\left(\frac{0}{10000^{\left(\frac{2\times 1}{5}\right)}}\right) \\ \cos\left(\frac{0}{10000^{\left(\frac{2\times 1}{5}\right)}}\right) \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$



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

How Positional Encodings are calculated?

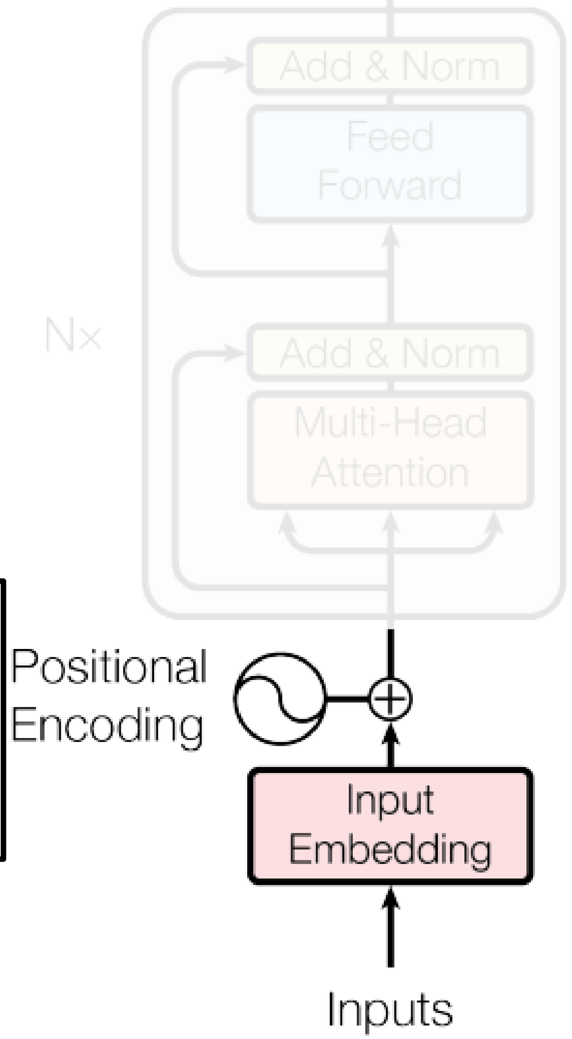
The		p0		Large		p1		Red		p2		Dog		p3
0.2		?		-0.9		?		0.5		?		0.6		?
-0.8		?		0.3		?		0.6		?		0.0		?
-0.6	+	?		0.4	+	?		-0.8	+	?		0.8	+	?
0.9		?		0.6		?		0.7		?		0.9		?
0.1		?		-0.1		?		0.7		?		0.4		?

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

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

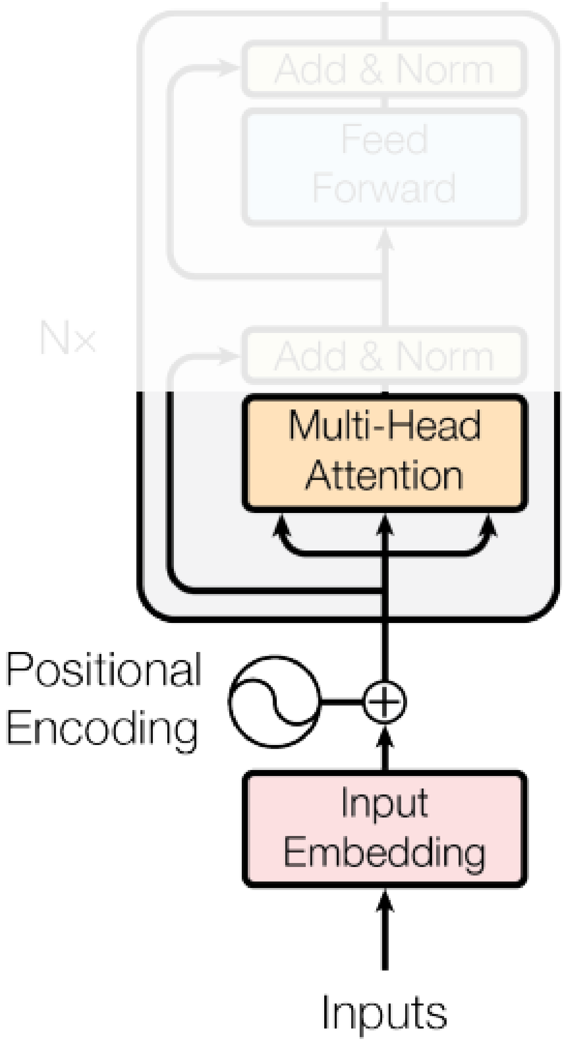
p1

$$\begin{bmatrix} PE_{(1,0)} \\ PE_{(1,1)} \\ PE_{(1,2)} \\ PE_{(1,3)} \\ PE_{(1,4)} \end{bmatrix} = \begin{bmatrix} \sin\left(\frac{1}{10000^{\left(\frac{2\times 0}{5}\right)}}\right) \\ \cos\left(\frac{1}{10000^{\left(\frac{2\times 0}{5}\right)}}\right) \\ \sin\left(\frac{1}{10000^{\left(\frac{2\times 1}{5}\right)}}\right) \\ \cos\left(\frac{1}{10000^{\left(\frac{2\times 1}{5}\right)}}\right) \\ \sin\left(\frac{1}{10000^{\left(\frac{2\times 2}{5}\right)}}\right) \end{bmatrix} \rightarrow \begin{bmatrix} 0.174000 \\ 0.999800 \\ 0.000174 \\ 0.999900 \\ 0.000631 \end{bmatrix}$$



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

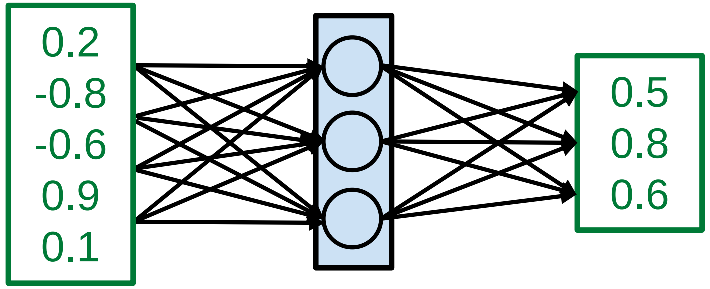
How Self Attention is calculated?

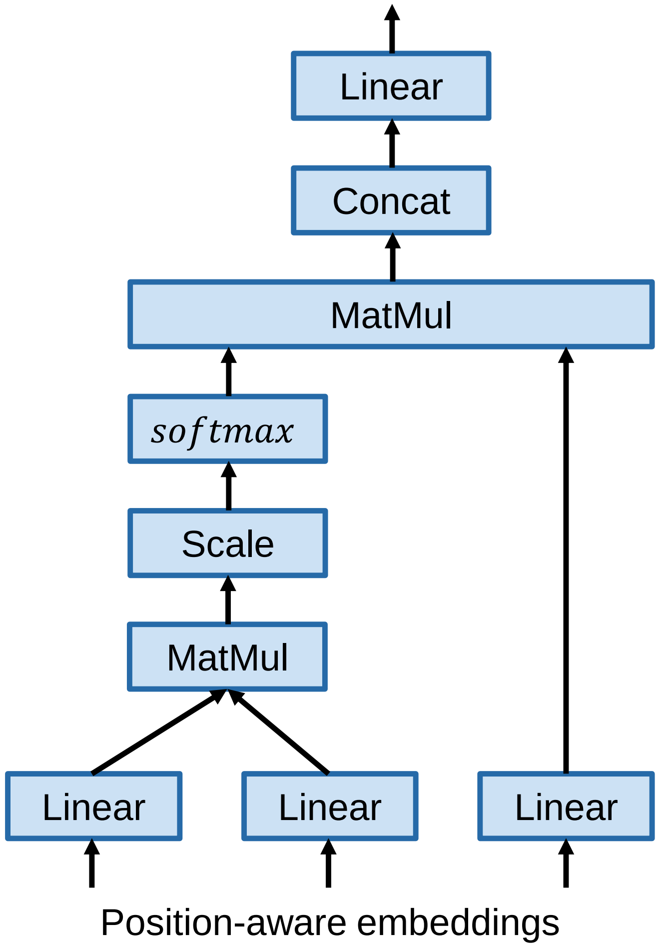


https://www.youtube.com/watch?v=mMa2PmYJlCo

How Self Attention is calculated?

- A Linear layer is a fully connected layer without activation function.
  - Used to map input to the output.
  - Change dimensionality.

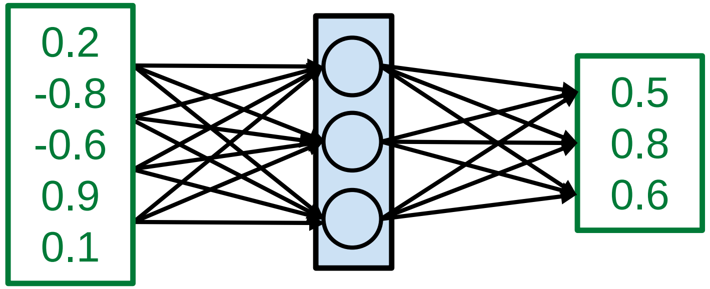




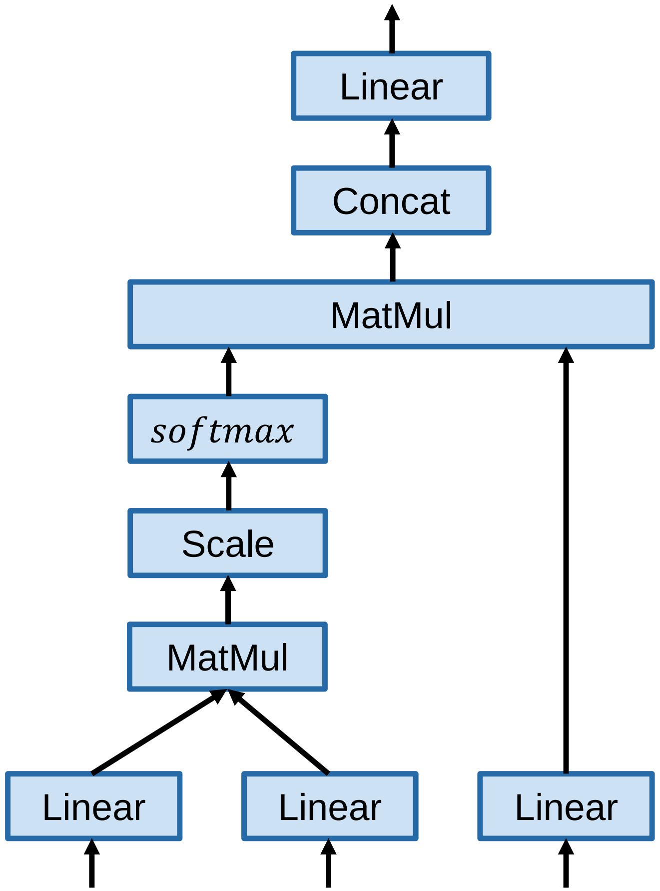
https://www.youtube.com/watch?v=mMa2PmYJlCo

How Self Attention is calculated?

- A Linear layer is a fully connected layer without activation function.
  - Used to map input to the output.
  - Change dimensionality.



- Query, Key, Value Formulation is used to calculate Attention.

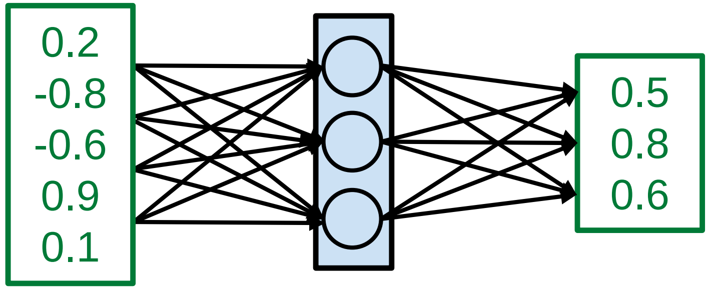


Position-aware embeddings

https://www.youtube.com/watch?v=mMa2PmYJlCo

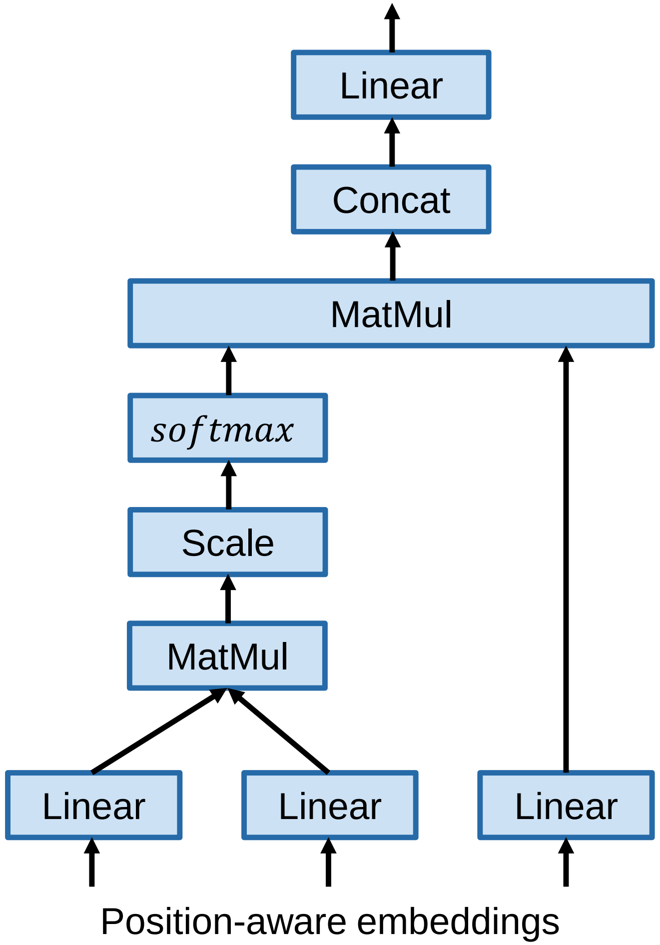
How Self Attention is calculated?

- A Linear layer is a fully connected layer without activation function.
    - Used to map input to the output.
    - Change dimensionality.



- Query, Key, Value Formulation is used to calculate Attention.
- Similarity between Query and Key can be considered as proxy to attention.
    - The similarity can be found using Cosine Similarity. (-1 to +1)

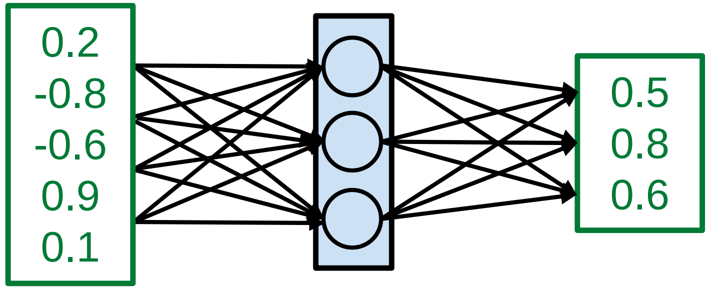
$$\cos(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a} \, . \, \boldsymbol{b}}{|\boldsymbol{a}||\boldsymbol{b}|}$$



https://www.youtube.com/watch?v=mMa2PmYJlCo

How Self Attention is calculated?

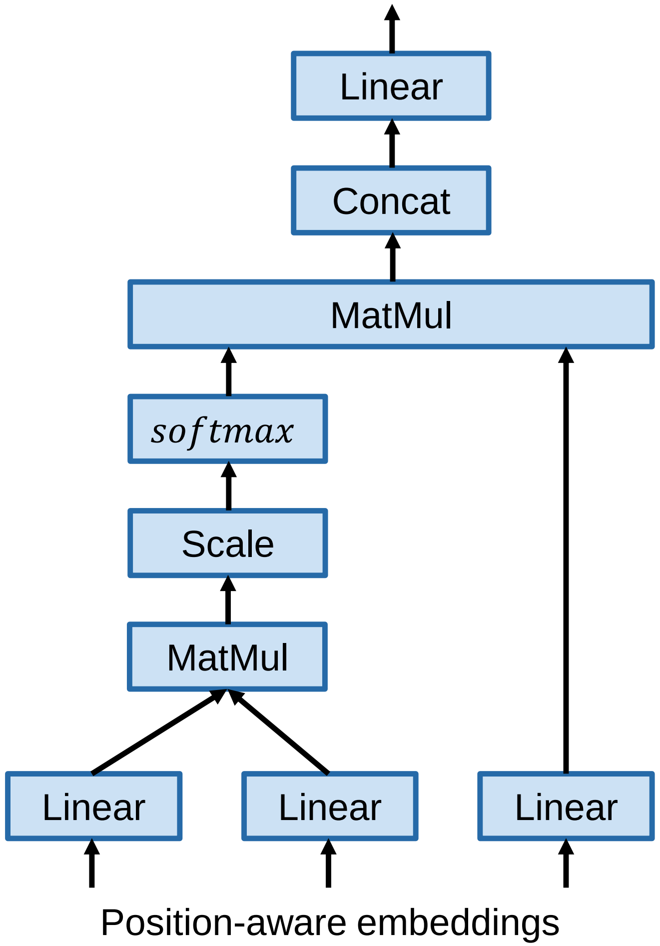
- A Linear layer is a fully connected layer without activation function.
    - Used to map input to the output.
    - Change dimensionality.



- Query, Key, Value Formulation is used to calculate Attention.
- Similarity between Query and Key can be considered as proxy to attention.
    - The similarity can be found using Cosine Similarity. (-1 to +1)

$$\cos(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{|\boldsymbol{a}||\boldsymbol{b}|}$$

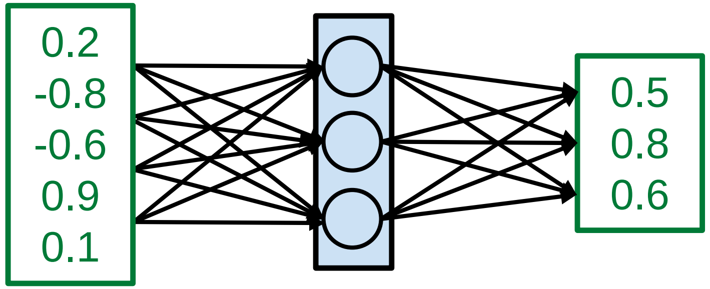
- What do Query, Key and Values consist of?



https://www.youtube.com/watch?v=mMa2PmYJlCo

How Self Attention is calculated?

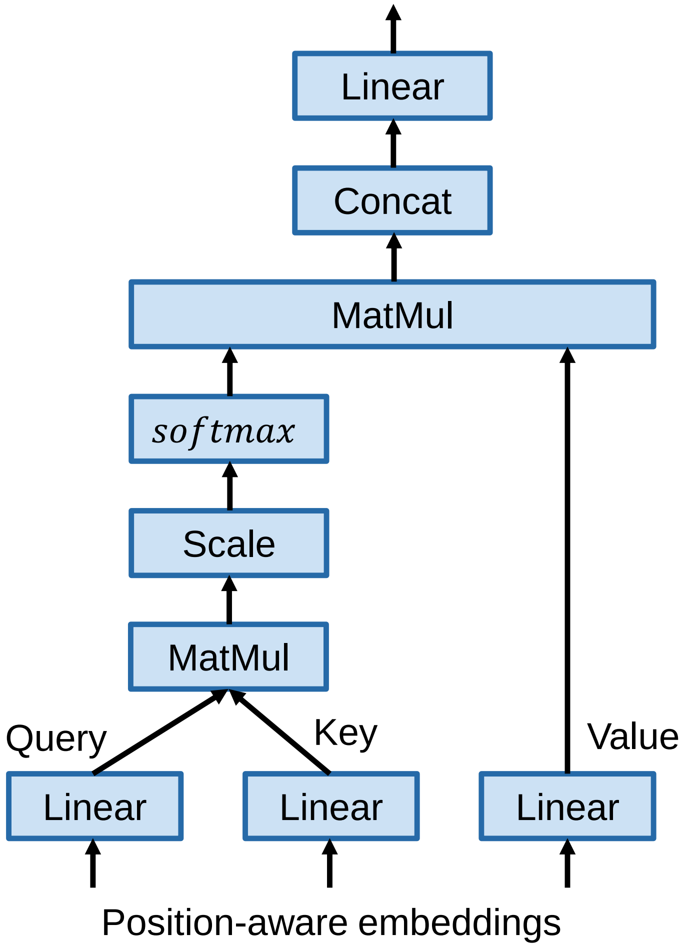
- A Linear layer is a fully connected layer without activation function.
  - Used to map input to the output.
  - Change dimensionality.



- Query, Key, Value Formulation is used to calculate Attention.
- Similarity between Query and Key can be considered as proxy to attention.
  - The similarity can be found using Cosine Similarity. (-1 to +1)

$$\cos(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a} \, . \, \boldsymbol{b}}{|\boldsymbol{a}||\boldsymbol{b}|}$$

- What do Query, Key and Values consist of?
  - Position-aware embeddings processed by relevant linear layers.



https://www.youtube.com/watch?v=mMa2PmYJlCo

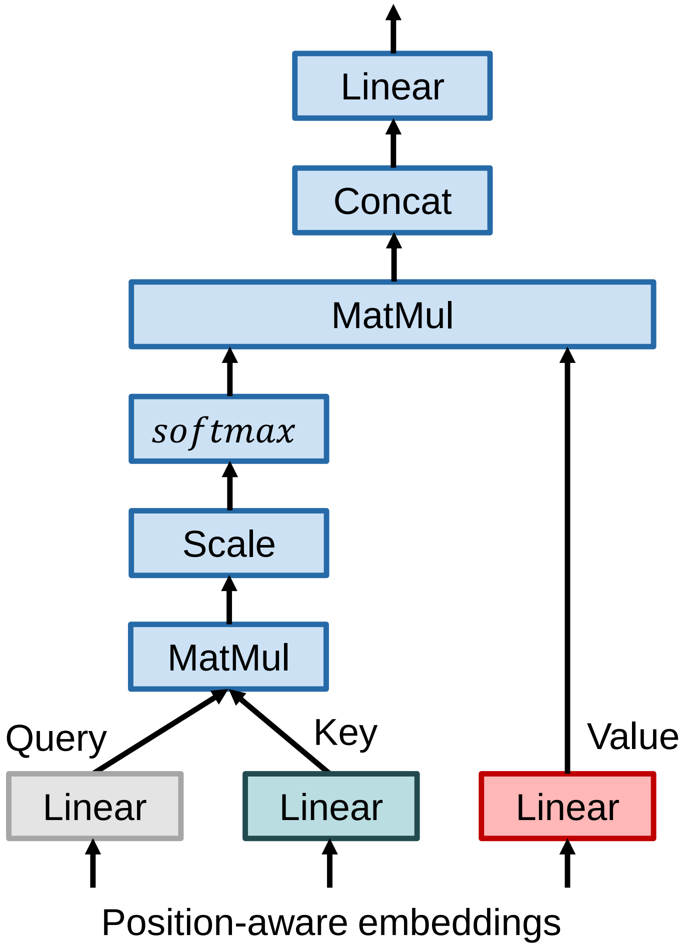
NUST
Defining futures
School of Electrical Engineering & Computer Science

Values of Query, Kay and Value matrices are learnt

4x5 positional embeddings $\boldsymbol{W} \in \mathbb{R}^{5\times3}$ 4x3 Query

The	0.2	-0.8	-0.6	0.9	0.1
Large	-0.9	0.3	0.4	0.6	-0.1
Red	0.5	0.6	-0.8	0.7	0.7
Dog	0.6	0.0	0.8	0.9	0.4

0.7	0.5	0.2
-0.3	0.6	-0.1
0.1	0.2	-0.9
0.4	0.8	0.1



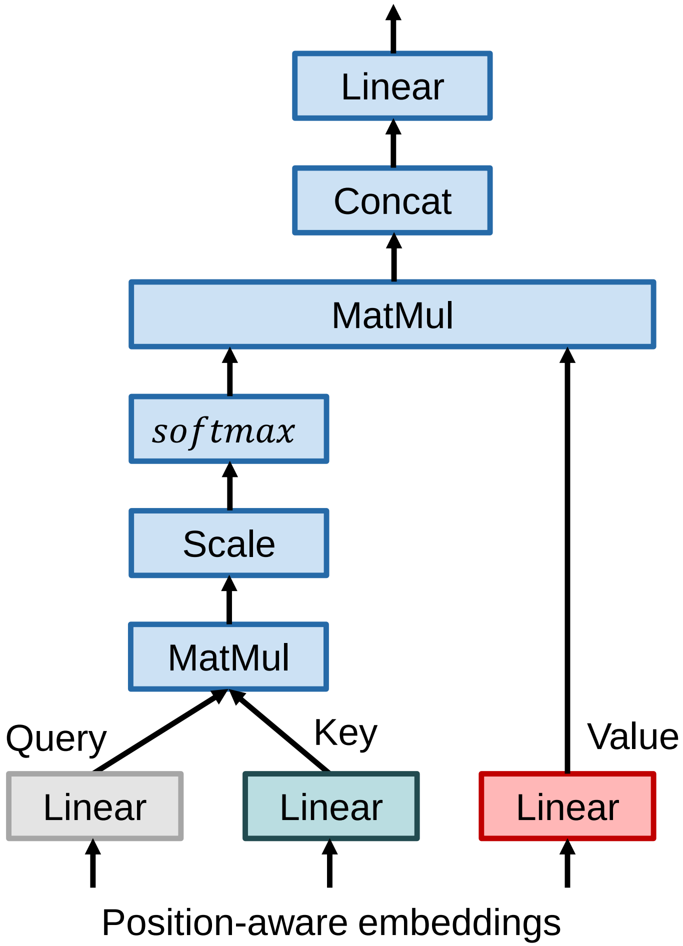
Values of Query, Kay and Value matrices are learnt

4x5 positional embeddings $\boldsymbol{W} \in \mathbb{R}^{5\times3}$ 4x3 Query

The	0.2	-0.8	-0.6	0.9	0.1	0.7	0.5	0.2
Large	-0.9	0.3	0.4	0.6	-0.1	-0.3	0.6	-0.1
Red	0.5	0.6	-0.8	0.7	0.7	0.1	0.2	-0.9
Dog	0.6	0.0	0.8	0.9	0.4	0.4	0.8	0.1

4x5 positional embeddings $\boldsymbol{W} \in \mathbb{R}^{5\times3}$ 4x3 Key

The	0.2	-0.8	-0.6	0.9	0.1	0.7	0.5	0.2
Large	-0.9	0.3	0.4	0.6	-0.1	-0.3	0.6	-0.1
Red	0.5	0.6	-0.8	0.7	0.7	0.1	0.2	-0.9
Dog	0.6	0.0	0.8	0.9	0.4	0.4	0.8	0.1



School of Electrical Engineering & Computer Science

Values of Query, Kay and Value matrices are learnt

4x5 positional embeddings $\boldsymbol{W} \in \mathbb{R}^{5\times3}$

The	0.2	-0.8	-0.6	0.9	0.1
Large	-0.9	0.3	0.4	0.6	-0.1
Red	0.5	0.6	-0.8	0.7	0.7
Dog	0.6	0.0	0.8	0.9	0.4

4x3 Query

0.7	0.5	0.2
-0.3	0.6	-0.1
0.1	0.2	-0.9
0.4	0.8	0.1

4x5 positional embeddings $\boldsymbol{W} \in \mathbb{R}^{5\times3}$

The	0.2	-0.8	-0.6	0.9	0.1
Large	-0.9	0.3	0.4	0.6	-0.1
Red	0.5	0.6	-0.8	0.7	0.7
Dog	0.6	0.0	0.8	0.9	0.4

4x3 Key

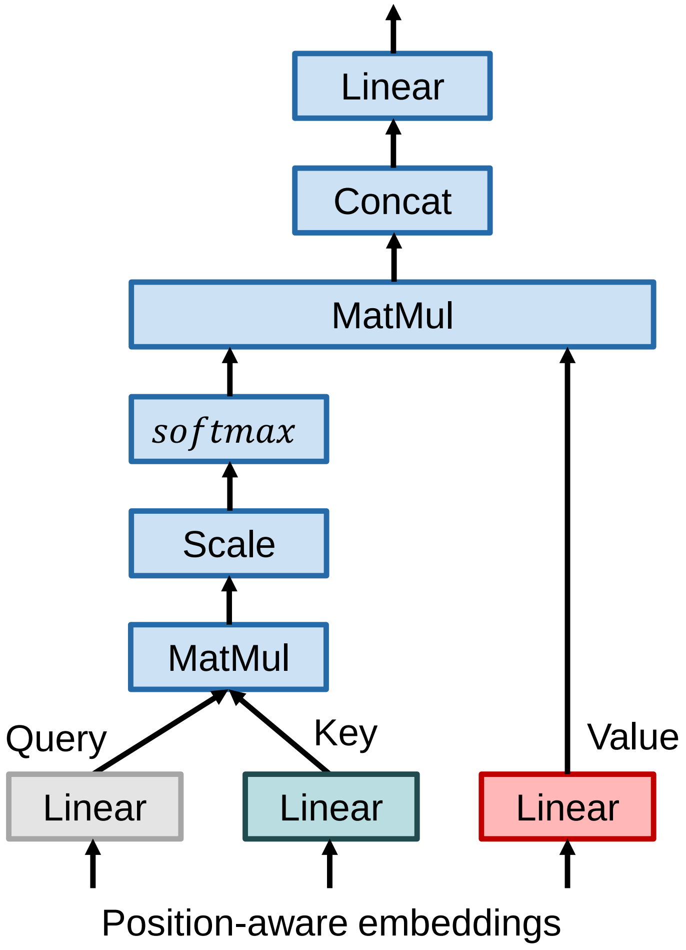
0.7	0.5	0.2
-0.3	0.6	-0.1
0.1	0.2	-0.9
0.4	0.8	0.1

4x5 positional embeddings $\boldsymbol{W} \in \mathbb{R}^{5\times3}$

The	0.2	-0.8	-0.6	0.9	0.1
Large	-0.9	0.3	0.4	0.6	-0.1
Red	0.5	0.6	-0.8	0.7	0.7
Dog	0.6	0.0	0.8	0.9	0.4

4x3 Value

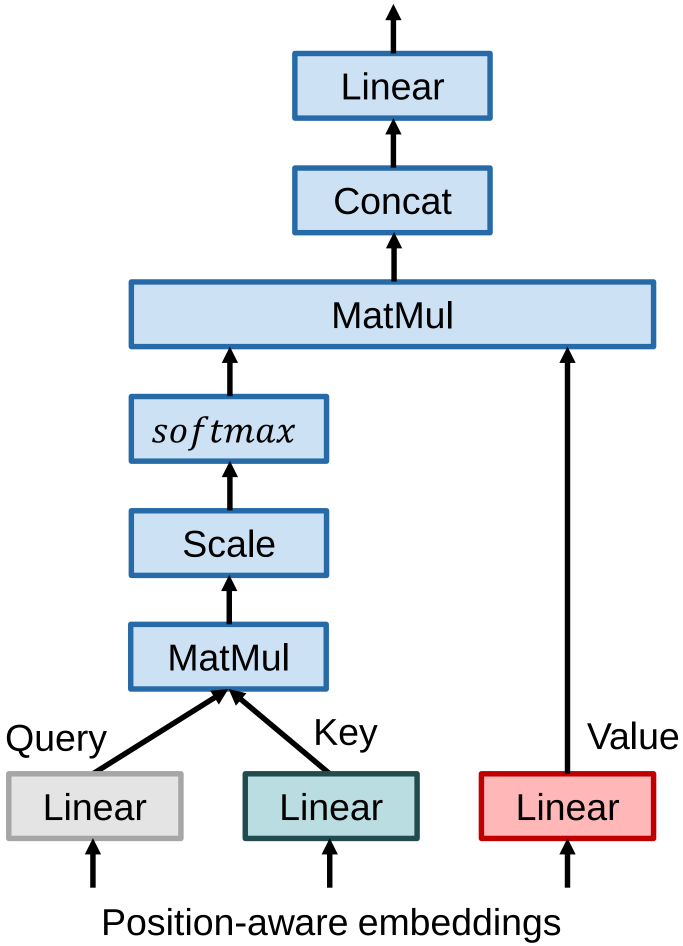
0.7	0.5	0.2
-0.3	0.6	-0.1
0.1	0.2	-0.9
0.4	0.8	0.1



Self Attention takes query and key from the same sequence

- Dot product of Query and Key gives attention scores.

$$\underset{4 \times 3\ Query}{\begin{bmatrix} 0.7 & 0.5 & 0.2 \\ -0.3 & 0.6 & -0.1 \\ 0.1 & 0.2 & -0.9 \\ 0.4 & 0.8 & 0.1 \end{bmatrix}} \times \underset{3 \times 4\ Key^T}{\begin{bmatrix} -0.1 & 0.8 & 0.3 & -0.3 \\ 0.2 & -0.4 & 0.5 & 0.9 \\ 0.1 & -0.8 & -0.7 & -0.6 \end{bmatrix}} = \underset{4 \times 4\ Attention\ Filter}{\begin{bmatrix} 0.5 & 0.2 & 0.3 & 0.1 \\ 0.1 & -0.4 & 0.3 & 0.7 \\ -0.1 & 0.7 & 0.8 & 0.7 \\ 0.1 & -0.1 & 0.4 & 0.5 \end{bmatrix}}$$



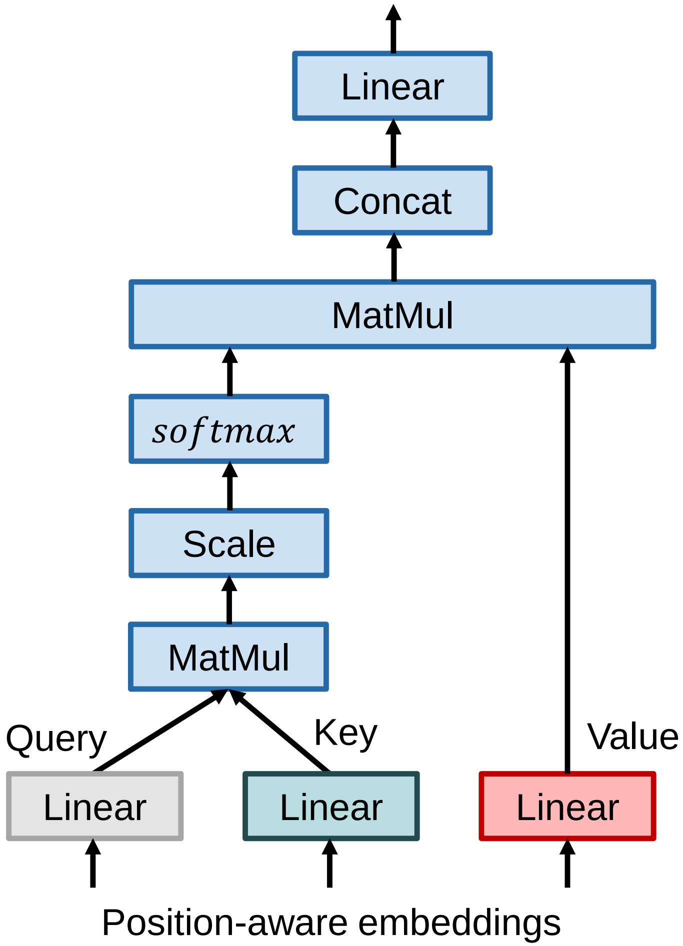
Self Attention takes query and key from the same sequence

- Dot product of Query and Key gives attention scores.

$$\underset{4\times 3\ Query}{\begin{bmatrix} 0.7 & 0.5 & 0.2 \\ -0.3 & 0.6 & -0.1 \\ 0.1 & 0.2 & -0.9 \\ 0.4 & 0.8 & 0.1 \end{bmatrix}} \times \underset{3\times 4\ Key^T}{\begin{bmatrix} -0.1 & 0.8 & 0.3 & -0.3 \\ 0.2 & -0.4 & 0.5 & 0.9 \\ 0.1 & -0.8 & -0.7 & -0.6 \end{bmatrix}} = \underset{4\times 4\ Attention\ Filter}{\begin{bmatrix} 0.5 & 0.2 & 0.3 & 0.1 \\ 0.1 & -0.4 & 0.3 & 0.7 \\ -0.1 & 0.7 & 0.8 & 0.7 \\ 0.1 & -0.1 & 0.4 & 0.5 \end{bmatrix}}$$

- Scale the attention scores by $\frac{1}{\sqrt{5}}$ and apply $softmax$.

$$\underset{4\times 4\ Attention\ Filter}{\begin{bmatrix} 0.5 & 0.2 & 0.3 & 0.1 \\ 0.1 & -0.4 & 0.3 & 0.7 \\ -0.1 & 0.7 & 0.8 & 0.7 \\ 0.1 & -0.1 & 0.4 & 0.5 \end{bmatrix}} \times \underset{4\times 3\ Value}{\begin{bmatrix} -0.1 & 0.4 & 0.5 \\ 0.6 & 0.7 & 0.8 \\ -0.2 & 0.3 & 0.9 \\ 0.3 & 0.9 & -0.1 \end{bmatrix}} = \underset{4\times 3\ Filtered\ Value}{\begin{bmatrix} -0.1 & 0.4 & 0.5 \\ 0.6 & 0.7 & 0.8 \\ -0.2 & 0.3 & 0.9 \\ 0.3 & 0.9 & -0.1 \end{bmatrix}}$$



NUST
Defining futures
School of Electrical Engineering & Computer Science

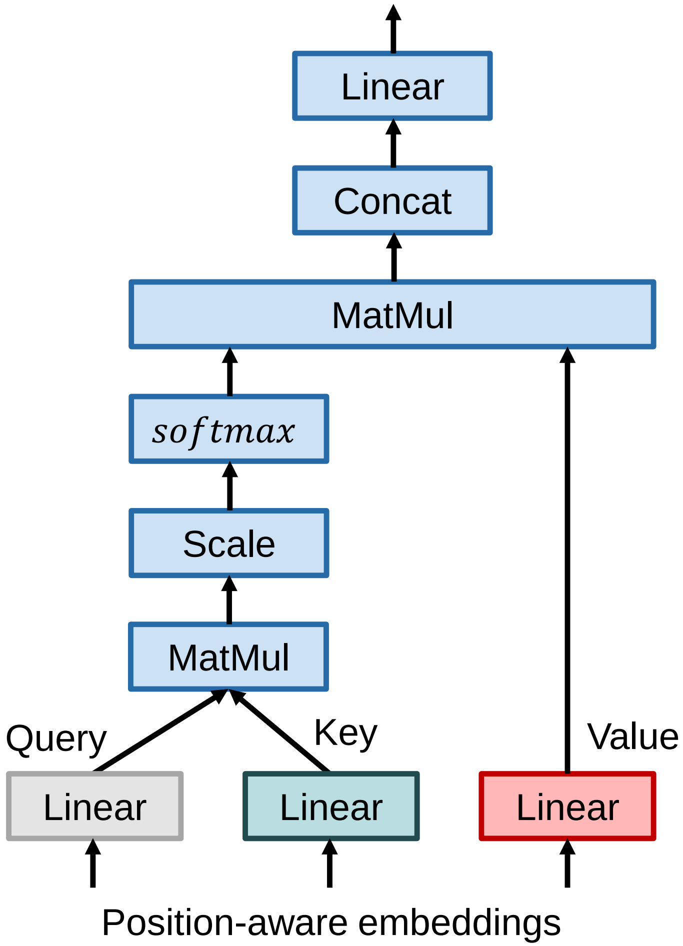
Self Attention takes query and key from the same sequence

- Dot product of Query and Key gives attention scores.

$$\underset{4 \times 3\ Query}{\begin{bmatrix} 0.7 & 0.5 & 0.2 \\ -0.3 & 0.6 & -0.1 \\ 0.1 & 0.2 & -0.9 \\ 0.4 & 0.8 & 0.1 \end{bmatrix}} \times \underset{3 \times 4\ Key^T}{\begin{bmatrix} -0.1 & 0.8 & 0.3 & -0.3 \\ 0.2 & -0.4 & 0.5 & 0.9 \\ 0.1 & -0.8 & -0.7 & -0.6 \end{bmatrix}} = \underset{4 \times 4\ Attention\ Filter}{\begin{bmatrix} 0.5 & 0.2 & 0.3 & 0.1 \\ 0.1 & -0.4 & 0.3 & 0.7 \\ -0.1 & 0.7 & 0.8 & 0.7 \\ 0.1 & -0.1 & 0.4 & 0.5 \end{bmatrix}}$$

- Scale the attention scores by $\frac{1}{\sqrt{5}}$ and apply $softmax$.

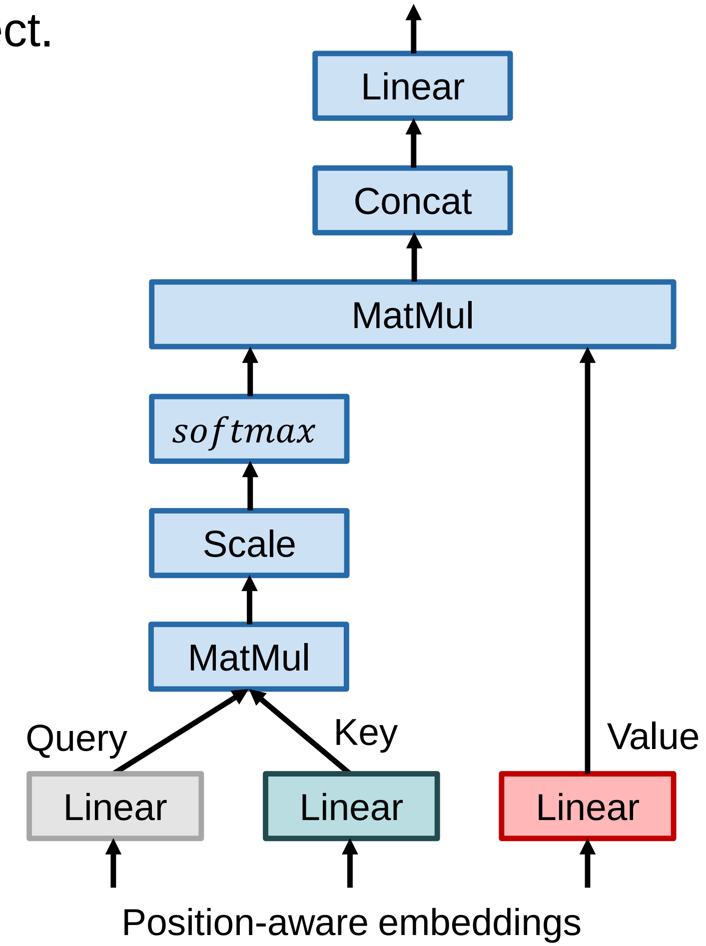
$$\underset{4 \times 4\ Attention\ Filter}{\begin{bmatrix} 0.5 & 0.2 & 0.3 & 0.1 \\ 0.1 & -0.4 & 0.3 & 0.7 \\ -0.1 & 0.7 & 0.8 & 0.7 \\ 0.1 & -0.1 & 0.4 & 0.5 \end{bmatrix}} \times \underset{4 \times 3\ Value}{\begin{bmatrix} -0.1 & 0.4 & 0.5 \\ 0.6 & 0.7 & 0.8 \\ -0.2 & 0.3 & 0.9 \\ 0.3 & 0.9 & -0.1 \end{bmatrix}} = \underset{4 \times 3\ Filtered\ Value}{\begin{bmatrix} -0.1 & 0.4 & 0.5 \\ 0.6 & 0.7 & 0.8 \\ -0.2 & 0.3 & 0.9 \\ 0.3 & 0.9 & -0.1 \end{bmatrix}}$$



$$Attension\ (Q, K, V) = softmax\left(\frac{Q.K^T}{\sqrt{d_k}}\right).V$$

Transformers encoder uses a Multi-Headed Attention block

- Multiple attention filters are learnt each focusing on a particular linguistic aspect.



Transformers encoder uses a Multi-Headed Attention block

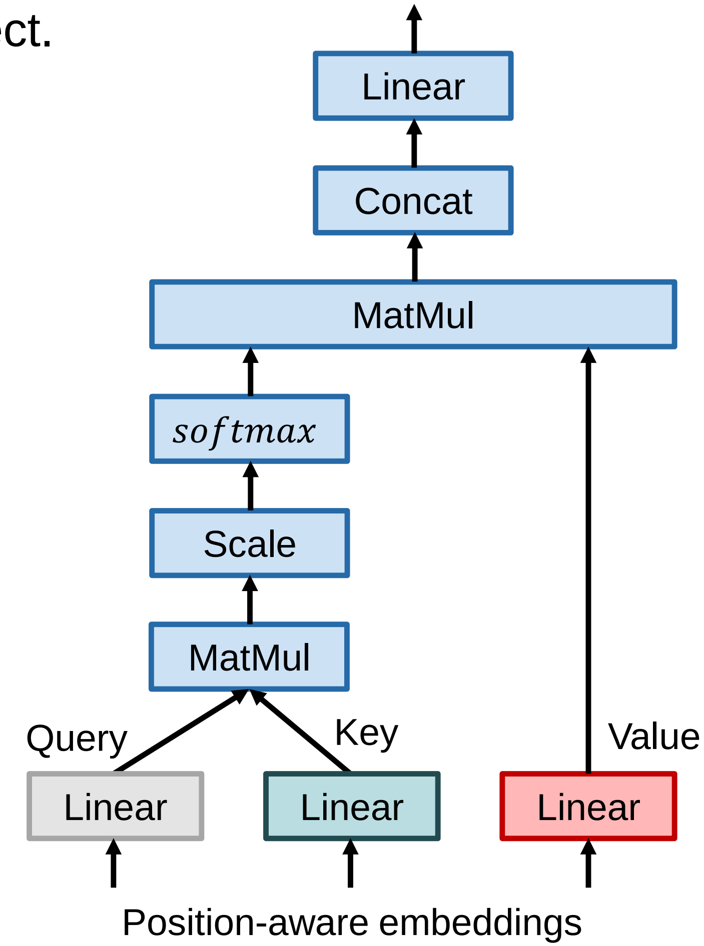
- Multiple attention filters are learnt each focusing on a particular linguistic aspect.

- Value matrices filtered by these multiple filters are then concatenated and passed through another linear layer.

$$MultiHead\ (Q, K, V) = Concat\ (head_1, head_2, \dots, head_h)W^O$$

The original Transformer paper uses $h = 8$ attention heads and

$$d_k = d_v = \frac{d_{model}}{8} = 64$$



Transformers encoder uses a Multi-Headed Attention block

- Multiple attention filters are learnt each focusing on a particular linguistic aspect.

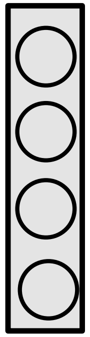
- Value matrices filtered by these multiple filters are then concatenated and passed through another linear layer.

$$MultiHead\ (Q, K, V) = Concat\ (head_1, head_2, \dots, head_h)W^O$$

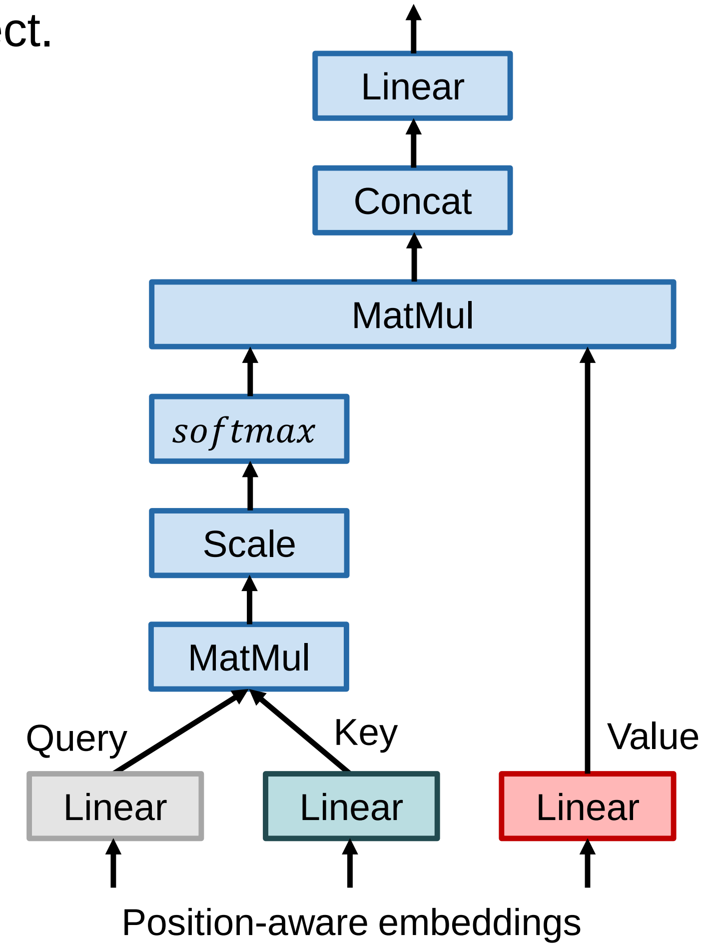
The original Transformer paper uses $h = 8$ attention heads and

$$d_k = d_v = \frac{d_{model}}{8} = 64$$

-0.1	0.4	0.5
0.6	0.7	0.8
-0.2	0.3	0.9
0.3	0.9	-0.1
-0.1	0.4	0.5
0.6	0.7	0.8
-0.2	0.3	0.9
0.3	0.9	-0.1
-0.1	0.4	0.5
0.6	0.7	0.8
-0.2	0.3	0.9
0.3	0.9	-0.1



0.2	-0.6	0.9	0.1
-0.9	0.3	0.4	-0.1
0.5	0.6	-0.8	0.7
0.6	0.8	0.9	0.4
-0.8	0.6	0.1	-0.4



Add position-aware embedding to attention output and normalise

- Add Position-aware Embeddings and the output of Multi-head Attention Layer.

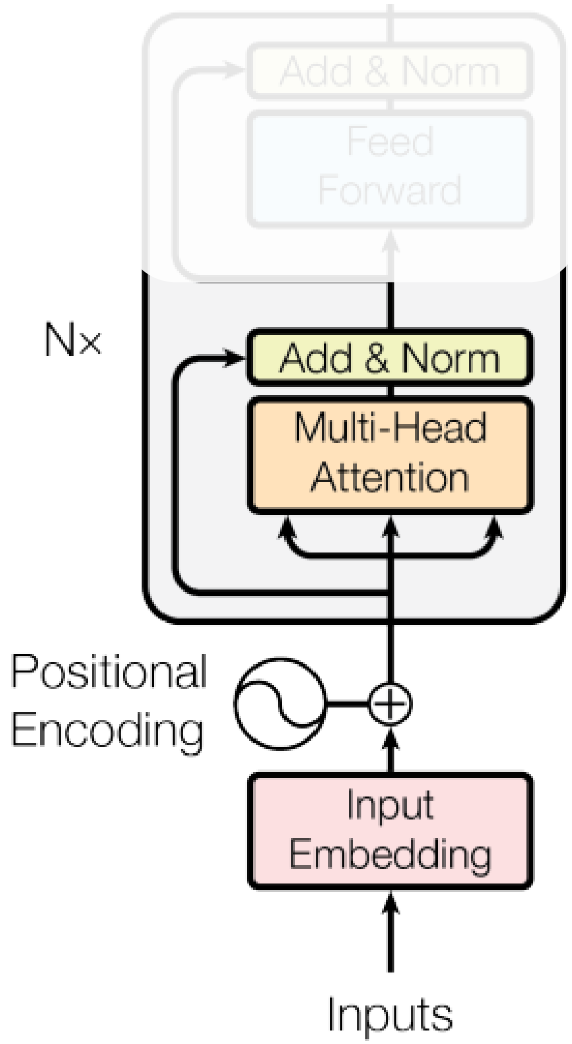
Skip Connection

0.1	-0.1	-0.3	0.8
0.8	-0.2	0.7	-0.9
-0.5	0.5	-0.6	0.3
0.7	-0.3	0.5	0.2
0.0	-0.3	0.4	0.1

+

MultiHead Attention Output

0.2	-0.6	0.9	0.1
-0.9	0.3	0.4	-0.1
0.5	0.6	-0.8	0.7
0.6	0.8	0.9	0.4
-0.8	0.6	0.1	-0.4



Add position-aware embedding to attention output and normalise

- Add Position-aware Embeddings and the output of Multi-head Attention Layer.

Skip Connection

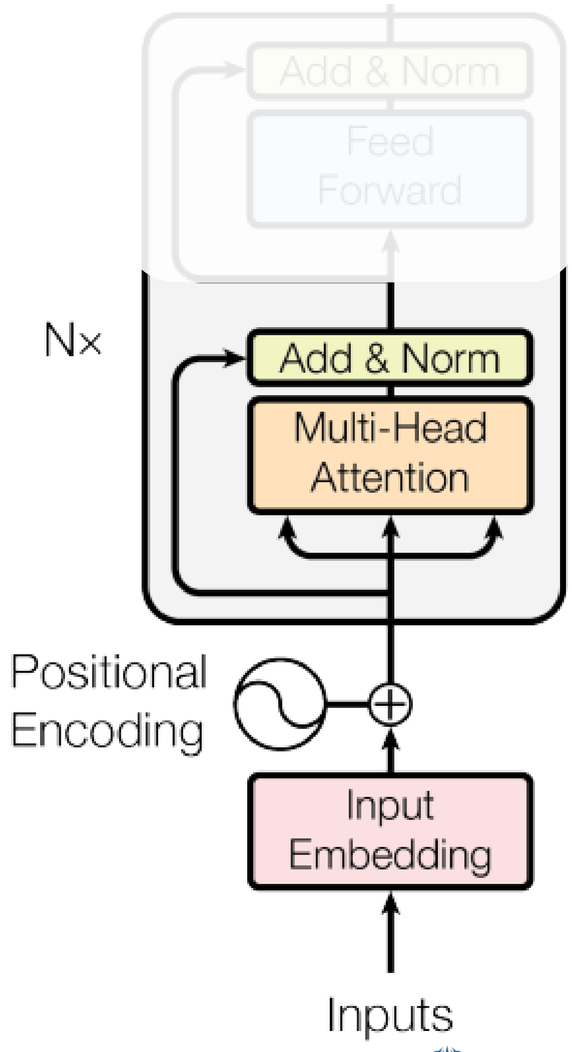
0.1	-0.1	-0.3	0.8
0.8	-0.2	0.7	-0.9
-0.5	0.5	-0.6	0.3
0.7	-0.3	0.5	0.2
0.0	-0.3	0.4	0.1

$+$

MultiHead Attention Output

0.2	-0.6	0.9	0.1
-0.9	0.3	0.4	-0.1
0.5	0.6	-0.8	0.7
0.6	0.8	0.9	0.4
-0.8	0.6	0.1	-0.4

- Perform z-score standardisation across features.



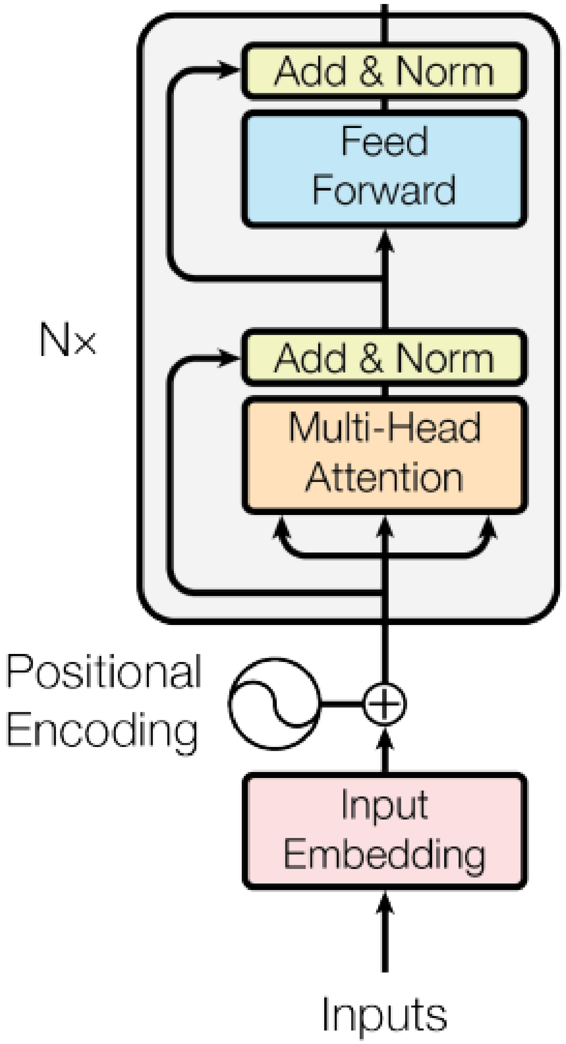
Add position-aware embedding to attention output and normalise

- Add Position-aware Embeddings and the output of Multi-head Attention Layer.

$$\underset{\text{Skip Connection}}{\begin{bmatrix} 0.1 & -0.1 & -0.3 & 0.8 \\ 0.8 & -0.2 & 0.7 & -0.9 \\ -0.5 & 0.5 & -0.6 & 0.3 \\ 0.7 & -0.3 & 0.5 & 0.2 \\ 0.0 & -0.3 & 0.4 & 0.1 \end{bmatrix}} + \underset{\text{MultiHead Attention Output}}{\begin{bmatrix} 0.2 & -0.6 & 0.9 & 0.1 \\ -0.9 & 0.3 & 0.4 & -0.1 \\ 0.5 & 0.6 & -0.8 & 0.7 \\ 0.6 & 0.8 & 0.9 & 0.4 \\ -0.8 & 0.6 & 0.1 & -0.4 \end{bmatrix}}$$

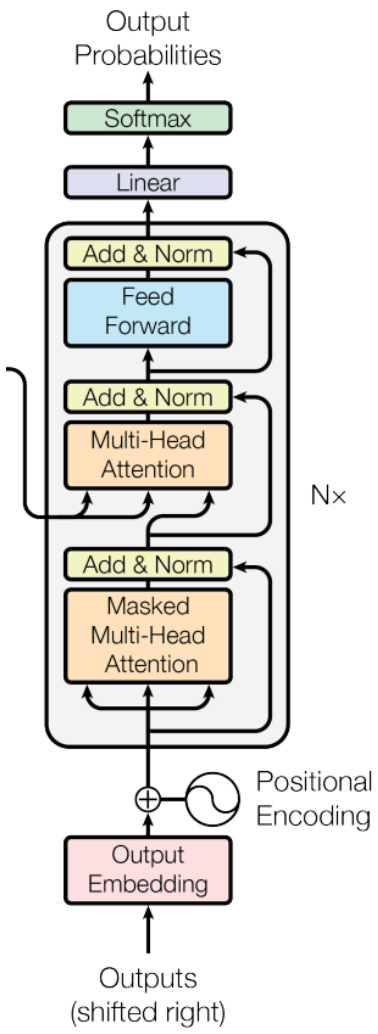
- Perform z-score standardisation across features.

- Pass through feedforward network and that's it.



Decoder uses masked self attention

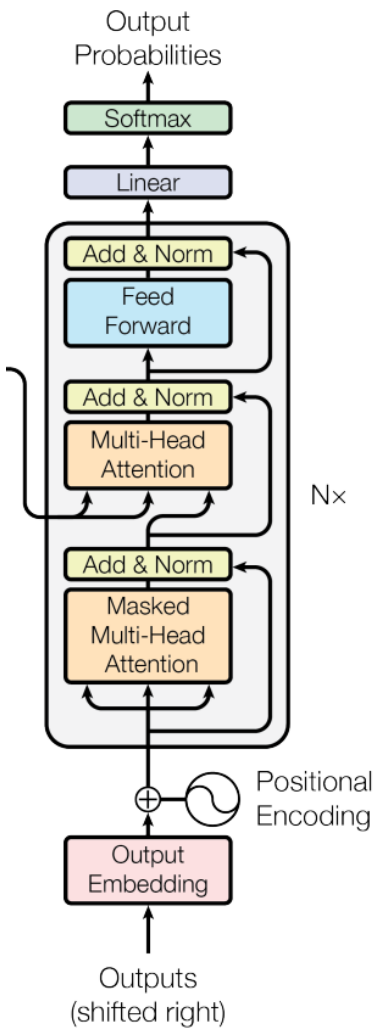
- The output of encoder is given to the decoder as Q and K matrices.



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

Decoder uses masked self attention

- The output of encoder is given to the decoder as Q and K matrices.
- The V Matrix is generated by the decoder.



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

Decoder uses masked self attention

- The output of encoder is given to the decoder as Q and K matrices.

- The V Matrix is generated by the decoder.

$Attention\ Filter$

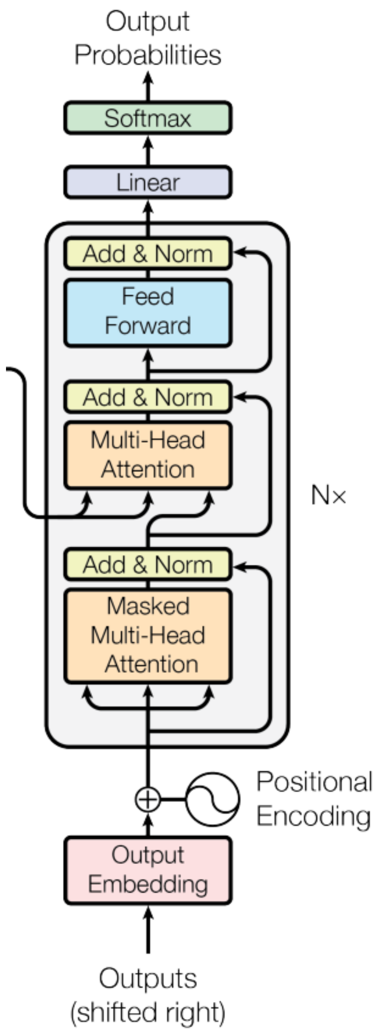
0.5	0.2	0.3	0.1
0.1	-0.4	0.3	0.7
-0.1	0.7	0.8	0.7
0.1	-0.1	0.4	0.5

$Masked\ Attention\ Filter$

0.5	-inf	-inf	-inf
0.1	-0.4	-inf	-inf
-0.1	0.7	0.8	-inf
0.1	-0.1	0.4	0.5

$Softmax\ of$
$Masked\ Attention\ Filter$

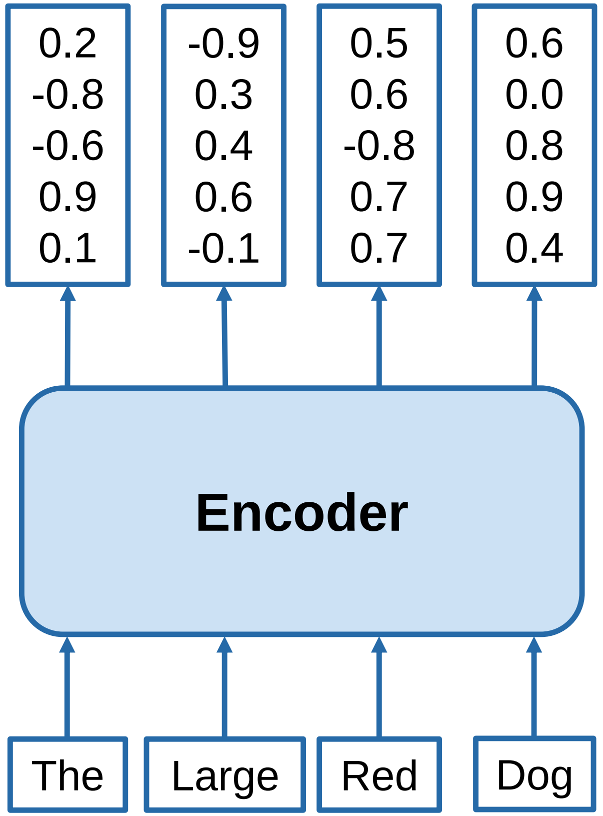
0.5	0	0	0
0.1	-0.4	0	0
-0.1	0.7	0.8	0
0.1	-0.1	0.4	0.5



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

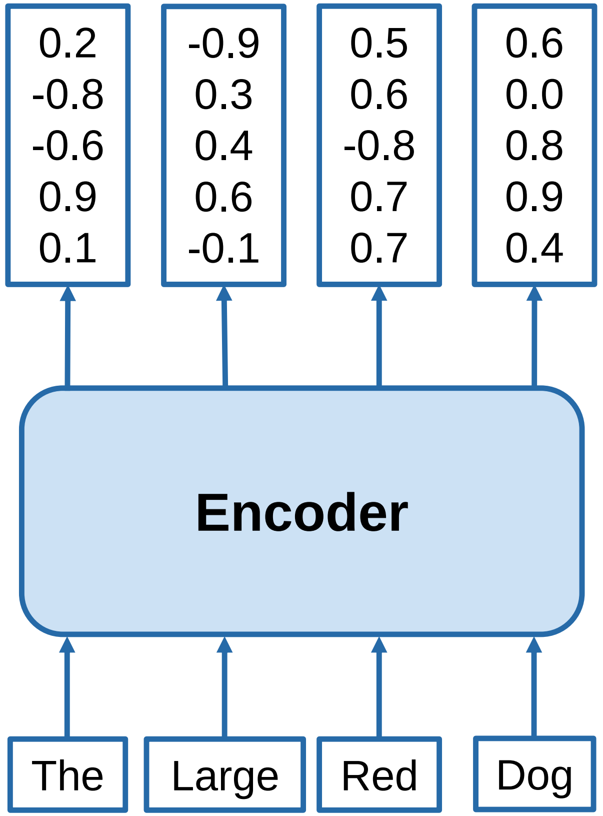
Encoder converts words into embeddings

- Dimension of word embeddings is defined by the model.
    - Base BERT used 768 dimensional embeddings.



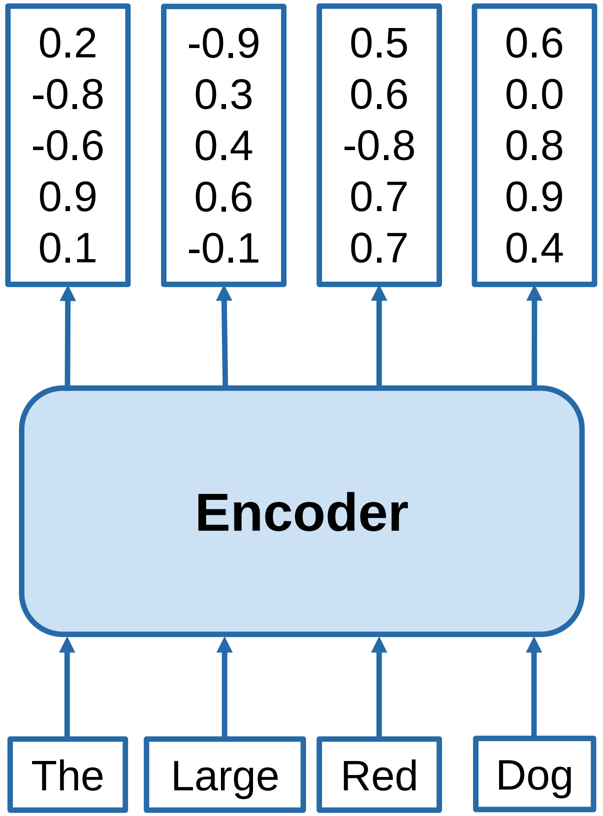
Encoder converts words into embeddings

- Dimension of word embeddings is defined by the model.
    - Base BERT used 768 dimensional embeddings.
- Transformer Encoder allows to process the whole input sequence in parallel.



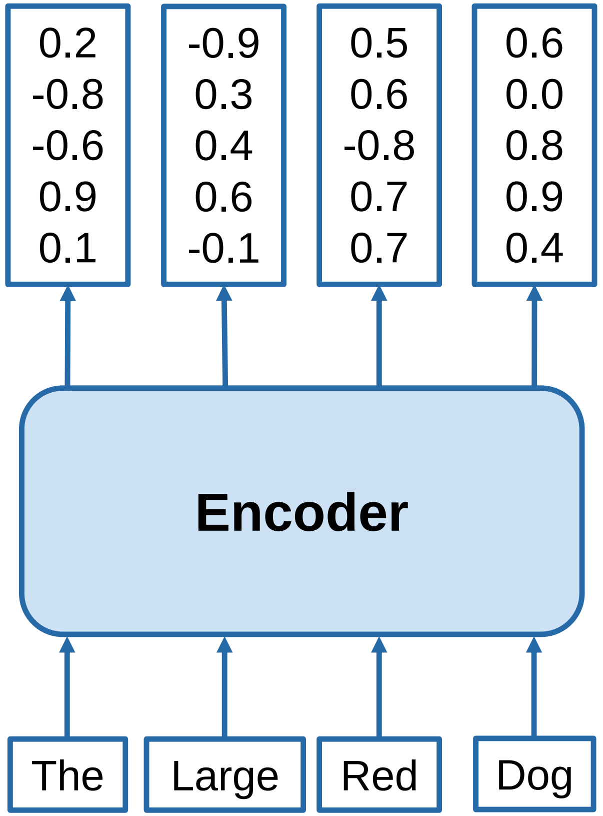
Encoder converts words into embeddings

- Dimension of word embeddings is defined by the model.
  - Base BERT used 768 dimensional embeddings.
- Transformer Encoder allows to process the whole input sequence in parallel.
- Transformer's Encoder module learns positional embeddings.
  - Meaning of a word considering its position in the text.



Encoder converts words into embeddings

- Dimension of word embeddings is defined by the model.
  - Base BERT used 768 dimensional embeddings.
- Transformer Encoder allows to process the whole input sequence in parallel.
- Transformer's Encoder module learns positional embeddings.
  - Meaning of a word considering its position in the text.
- Uses Self-Attention mechanism, which consults other words in the sequence to learn the meaning of a given word.



Applications of Encoders

- Encoders can be used as stand alone models.
    - Very good at learning meaningful representations.

Applications of Encoders

- Encoders can be used as stand alone models.
    - Very good at learning meaningful representations.
- May also be used for sequence classification, question answering, masked language modelling etc.

Applications of Encoders

- Encoders can be used as stand alone models.
  - Very good at learning meaningful representations.
- May also be used for sequence classification, question answering, masked language modelling etc.
  - Masked Language Modelling. Makes use of bi-directional context.

You	Know	Nothing		Snow

Applications of Encoders

- Encoders can be used as stand alone models.
    - Very good at learning meaningful representations.
- May also be used for sequence classification, question answering, masked language modelling etc.
    - Masked Language Modelling. Makes use of bi-directional context.

You	Know	Nothing		Snow
			Jon	

Applications of Encoders

- Encoders can be used as stand alone models.
  - Very good at learning meaningful representations.
- May also be used for sequence classification, question answering, masked language modelling etc.
  - Masked Language Modelling. Makes use of bi-directional context.

About

You Know Nothing [ ] Snow

Jon

Applications of Encoders

- Encoders can be used as stand alone models.
  - Very good at learning meaningful representations.
- May also be used for sequence classification, question answering, masked language modelling etc.
  - Masked Language Modelling. Makes use of bi-directional context.
  - Sentiment Analysis

Even though she did not win the award, she was satisfied	**Positive**
Even though she did win the award, she was not satisfied	**Negative**

Applications of Encoders

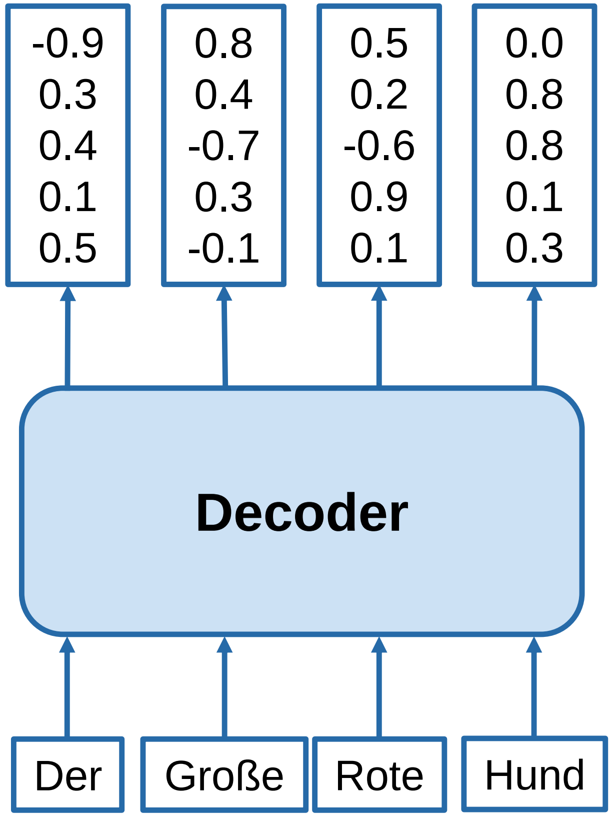
- Encoders can be used as stand alone models.
  - Very good at learning meaningful representations.
- May also be used for sequence classification, question answering, masked language modelling etc.
  - Masked Language Modelling. Makes use of bi-directional context.
  - Sentiment Analysis

Even though she did not win the award, she was satisfied	**Positive**
Even though she did win the award, she was not satisfied	**Negative**

- Common examples are BERT (SOTA of its time) and its variants RoERTa and ALBERT.

Decoders are very similar in architecture to encoders

- Can be used to perform the same tasks as encoders (with reduced performance)

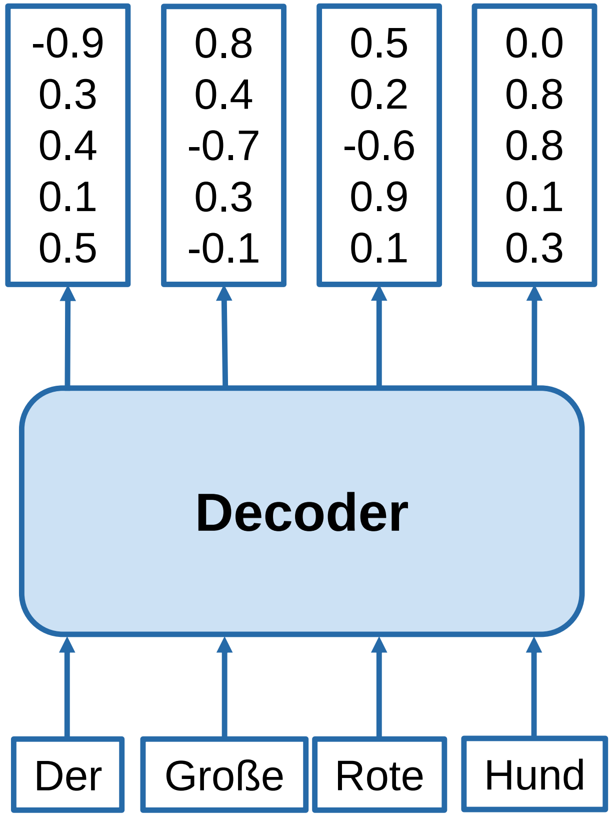


https://www.youtube.com/watch?v=d_ixlCubqQw

Decoders are very similar in architecture to encoders

- Can be used to perform the same tasks as encoders (with reduced performance)

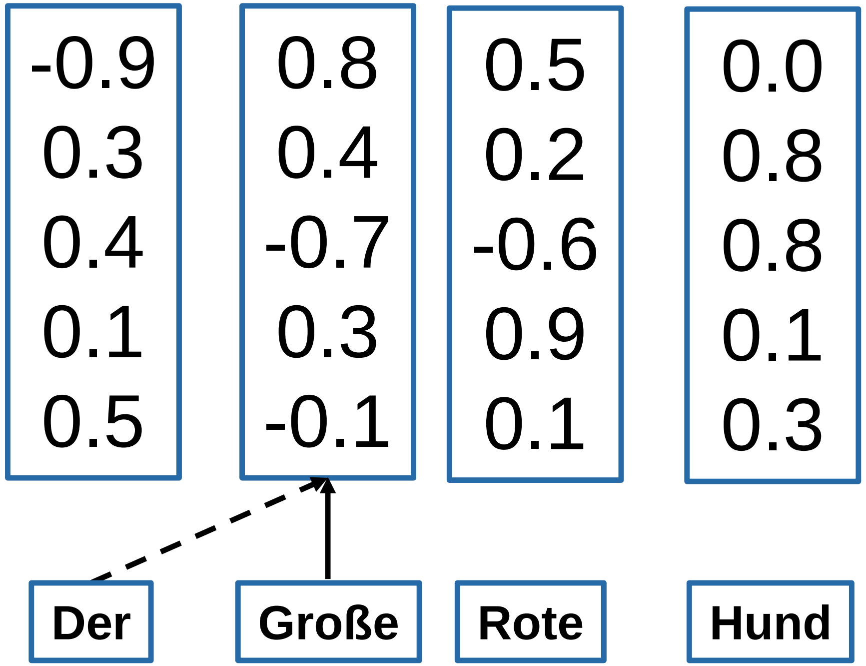
- Transformer's Decoder module also learns positional embeddings.



https://www.youtube.com/watch?v=d_ixlCubqQw

Decoders are very similar in architecture to encoders

- Can be used to perform the same tasks as encoders (with reduced performance)
- Transformer’s Decoder module also learns positional embeddings.
- Uses Masked Self-Attention mechanism, which consults only the previous or the following words in the sequence to learn the meaning of a given word.



https://www.youtube.com/watch?v=d_ixlCubqQw

Applications of Decoders

- Can be used as stand alone models

Applications of Decoders

- Can be used as stand alone models

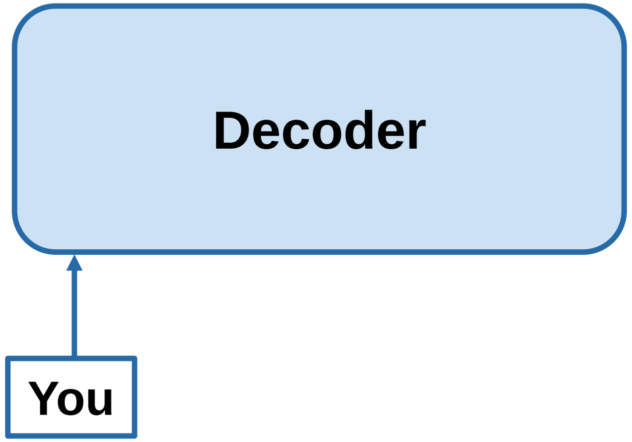
- Great at Natural Language Generation due to unidirectional context.

Applications of Decoders

- Can be used as stand alone models

- Great at Natural Language Generation due to unidirectional context.

    - Input an initial word, <You>.

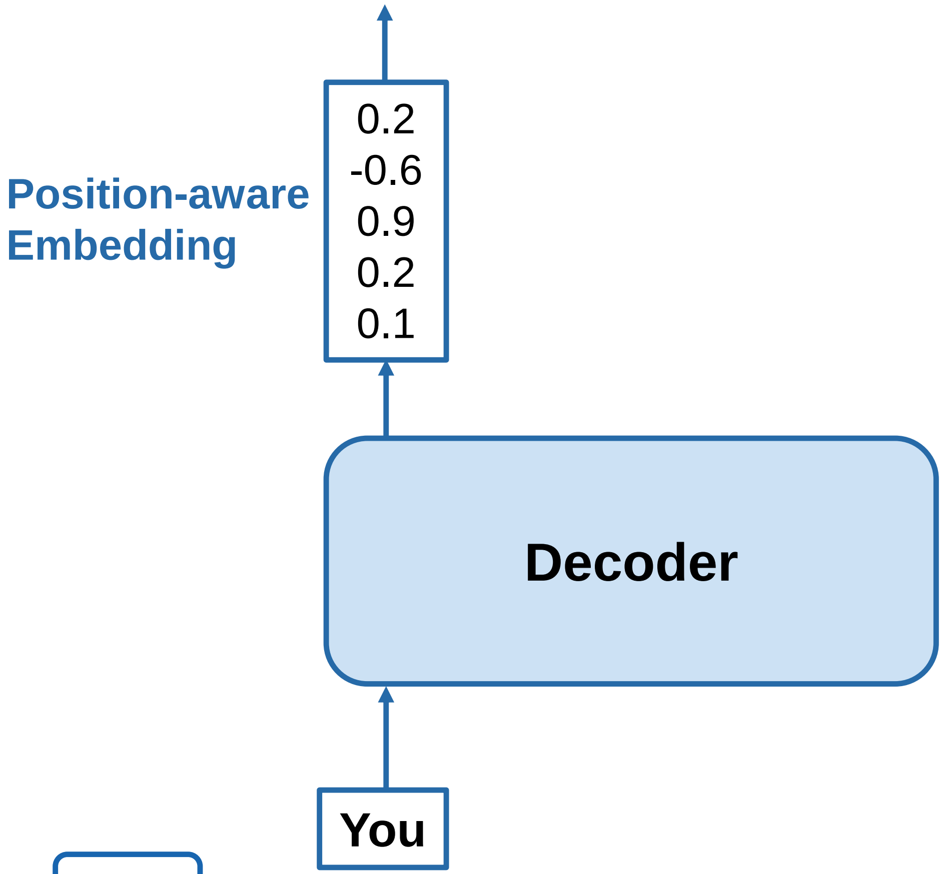


Applications of Decoders

- Can be used as stand alone models

- Great at Natural Language Generation due to unidirectional context.

    - Input an initial word, <You>.
    - Decoder determines contextual embedding based on unidirectional context (so far).



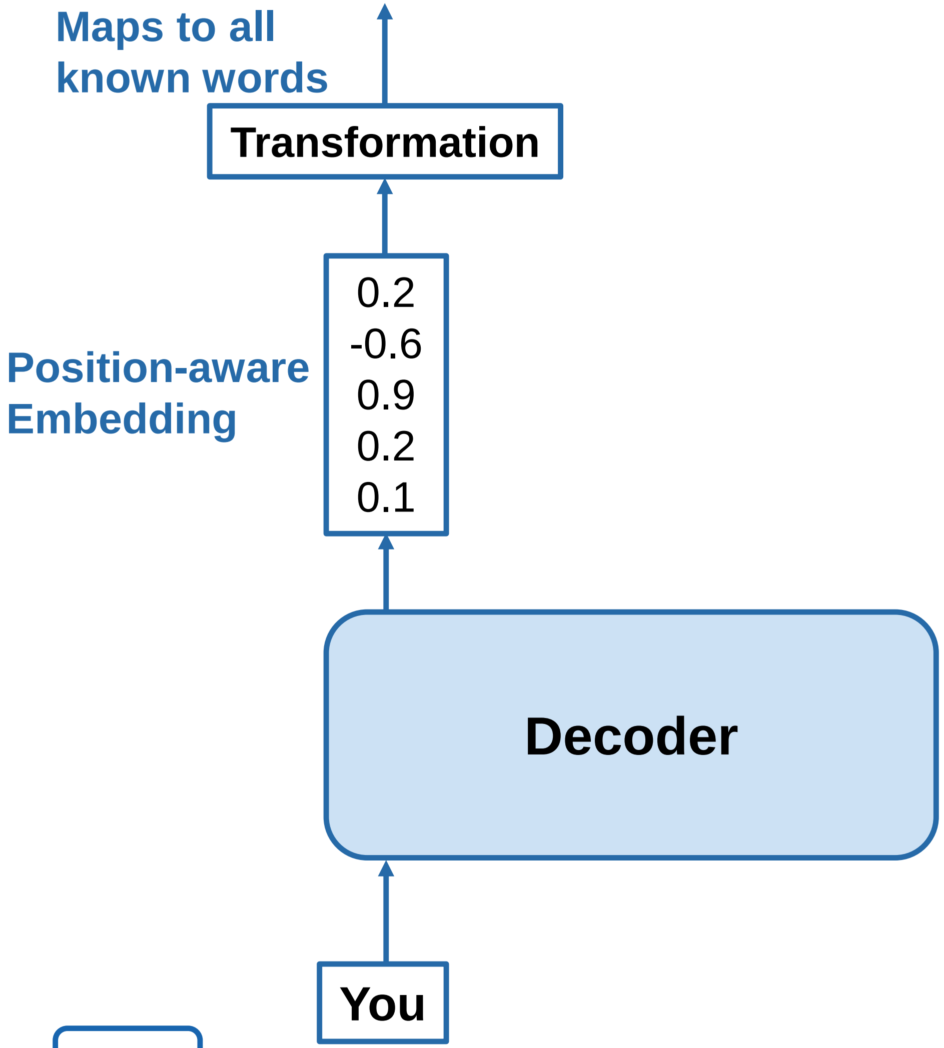
NUST
Defining futures
School of Electrical Engineering & Computer Science

Applications of Decoders

- Can be used as stand alone models

- Great at Natural Language Generation due to unidirectional context.

    - Input an initial word, <You>.
    - Decoder determines contextual embedding based on unidirectional context (so far).
    - A transformation maps the embedding to all known words.

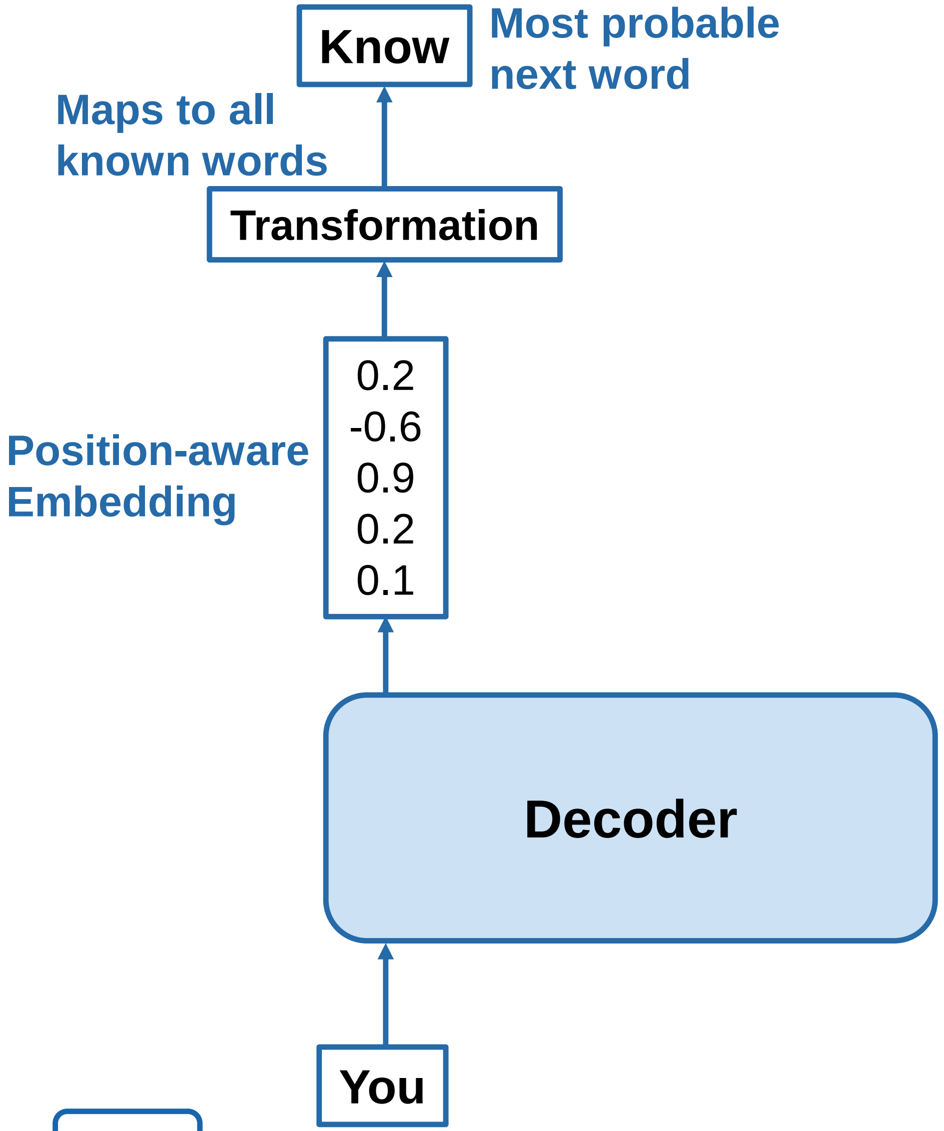


Applications of Decoders

- Can be used as stand alone models

- Great at Natural Language Generation due to unidirectional context.

    - Input an initial word, <You>.
    - Decoder determines contextual embedding based on unidirectional context (so far).
    - A transformation maps the embedding to all known words.
    - The most probable next word is given as output.

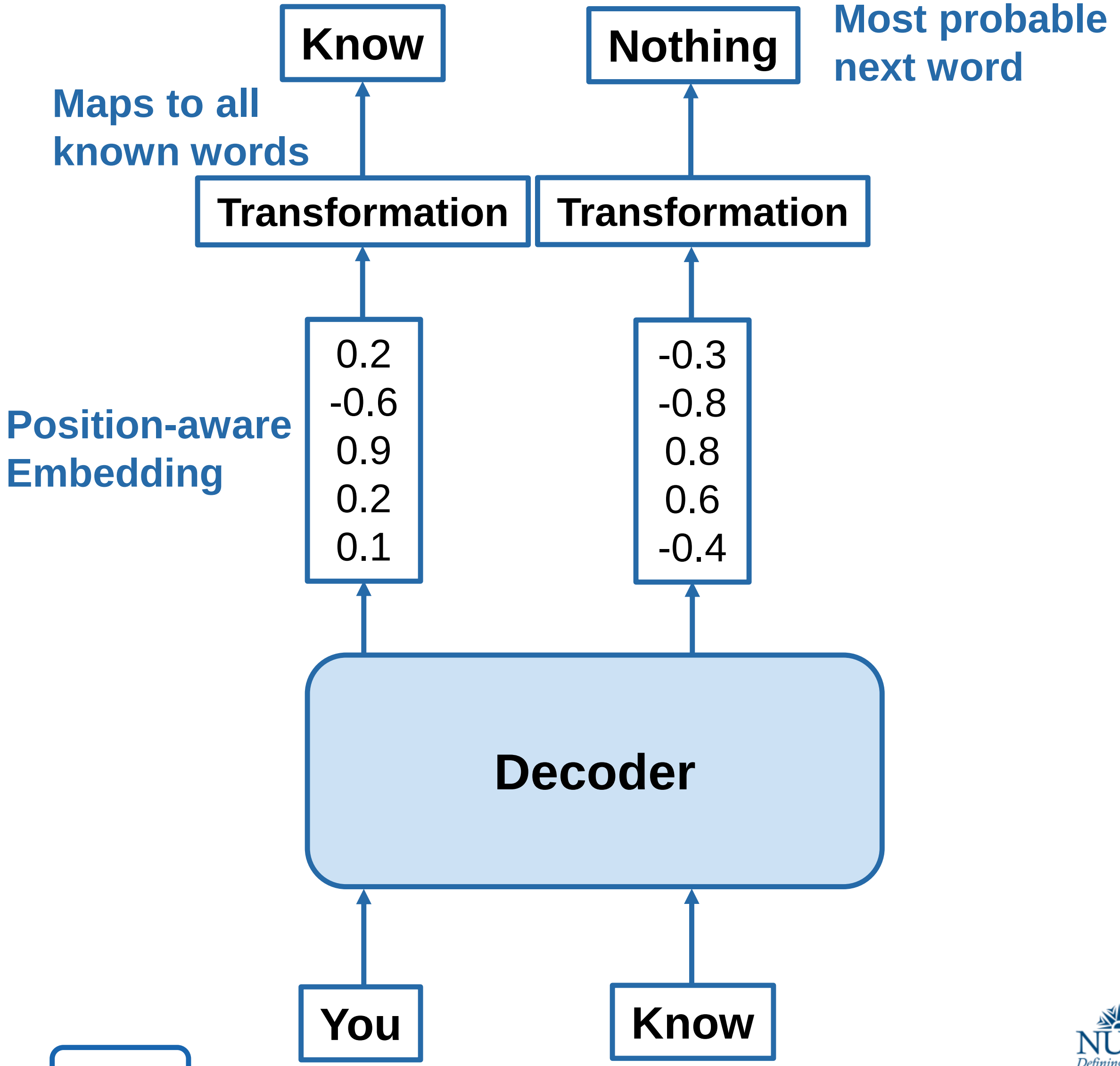


Applications of Decoders

- Can be used as stand alone models

- Great at Natural Language Generation due to unidirectional context.

    - Input an initial word, <You>.
    - Decoder determines contextual embedding based on unidirectional context (so far).
    - A transformation maps the embedding to all known words.
    - The most probable next word is given as output.
    - Current output is used as next input.



Applications of Decoders

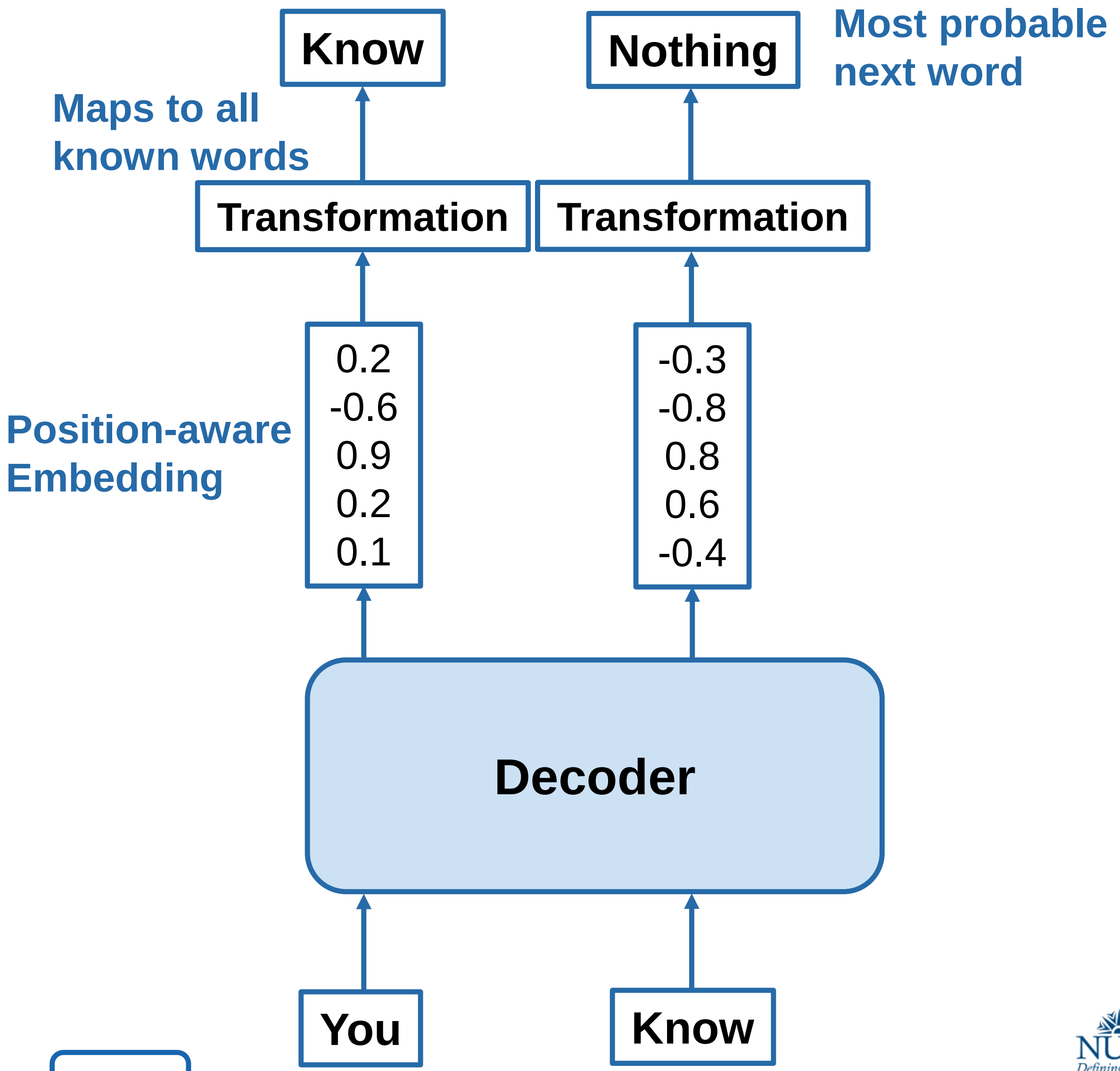
- Can be used as stand alone models

- Great at Natural Language Generation due to unidirectional context.

    - Input an initial word, <You>.
    - Decoder determines contextual embedding based on unidirectional context (so far).
    - A transformation maps the embedding to all known words.
    - The most probable next word is given as output.
    - Current output is used as next input.

- Examples include GPT and GPT-2.

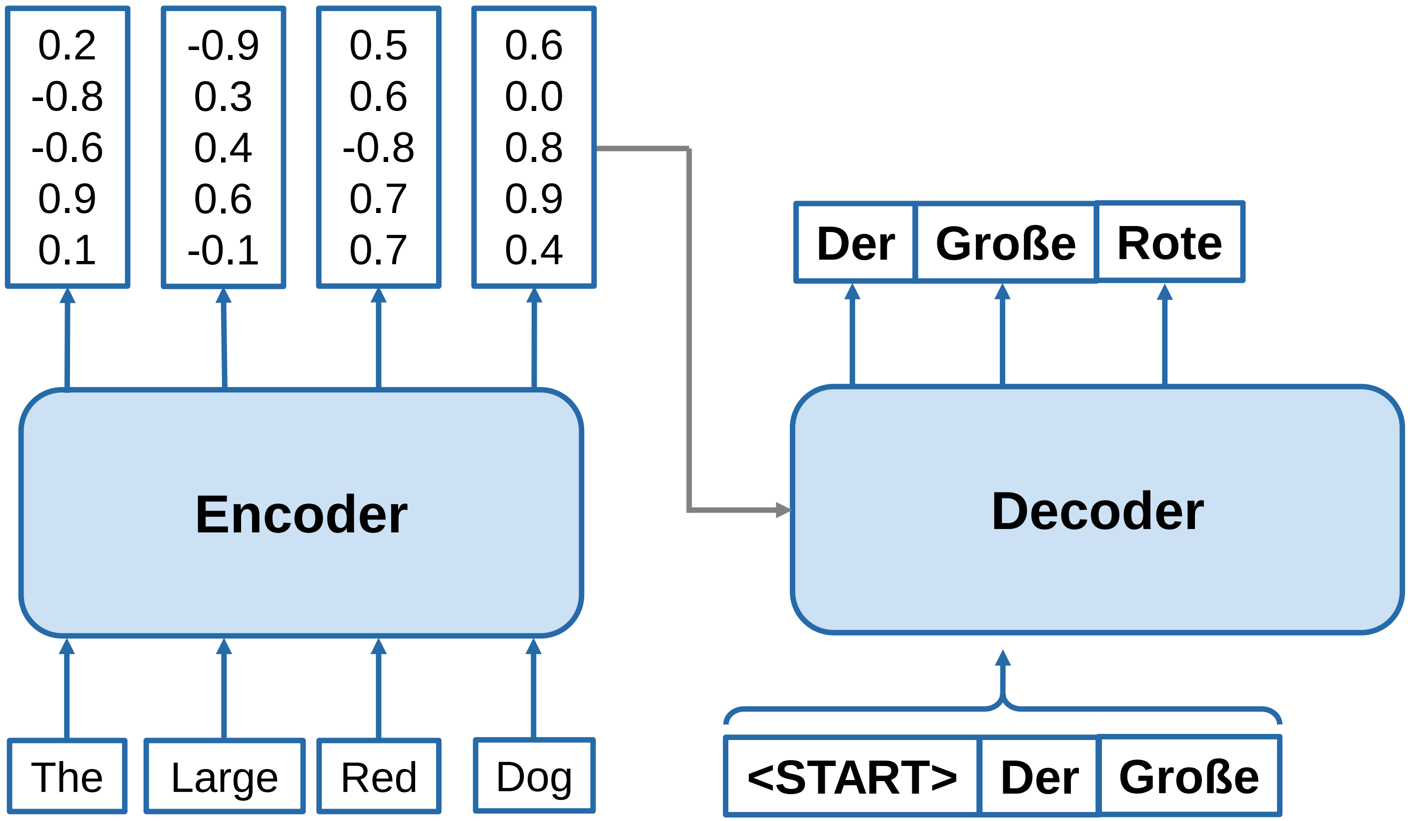
    - GPT-2 has a context of size 1024.



NUST Defining futures
School of Electrical Engineering & Computer Science

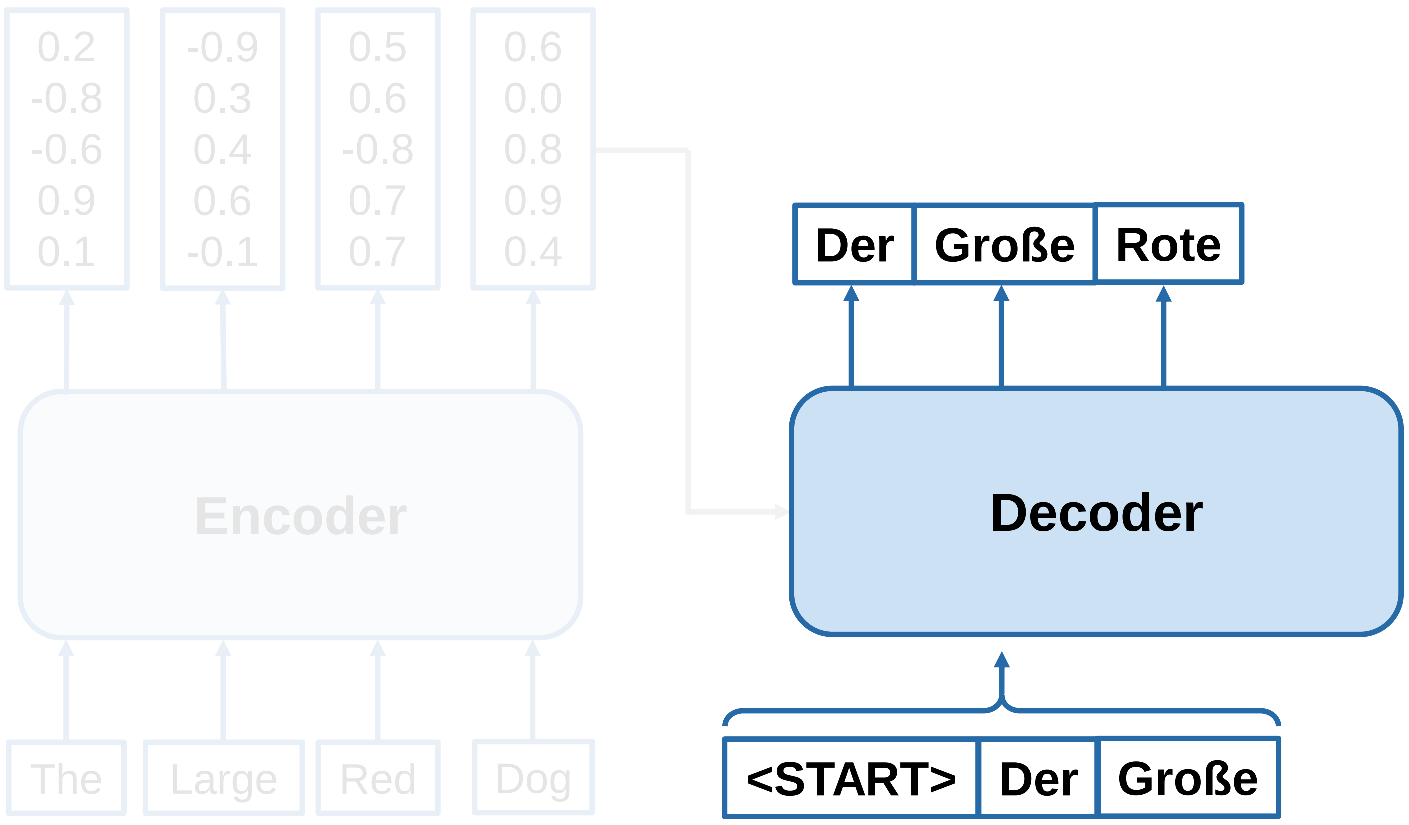
Encoder-Decoder architecture addresses *seq2seq* tasks

- Useful for translation and summarisation.



https://www.youtube.com/watch?v=0_4KEb08xrE

- Useful for translation and summarisation.

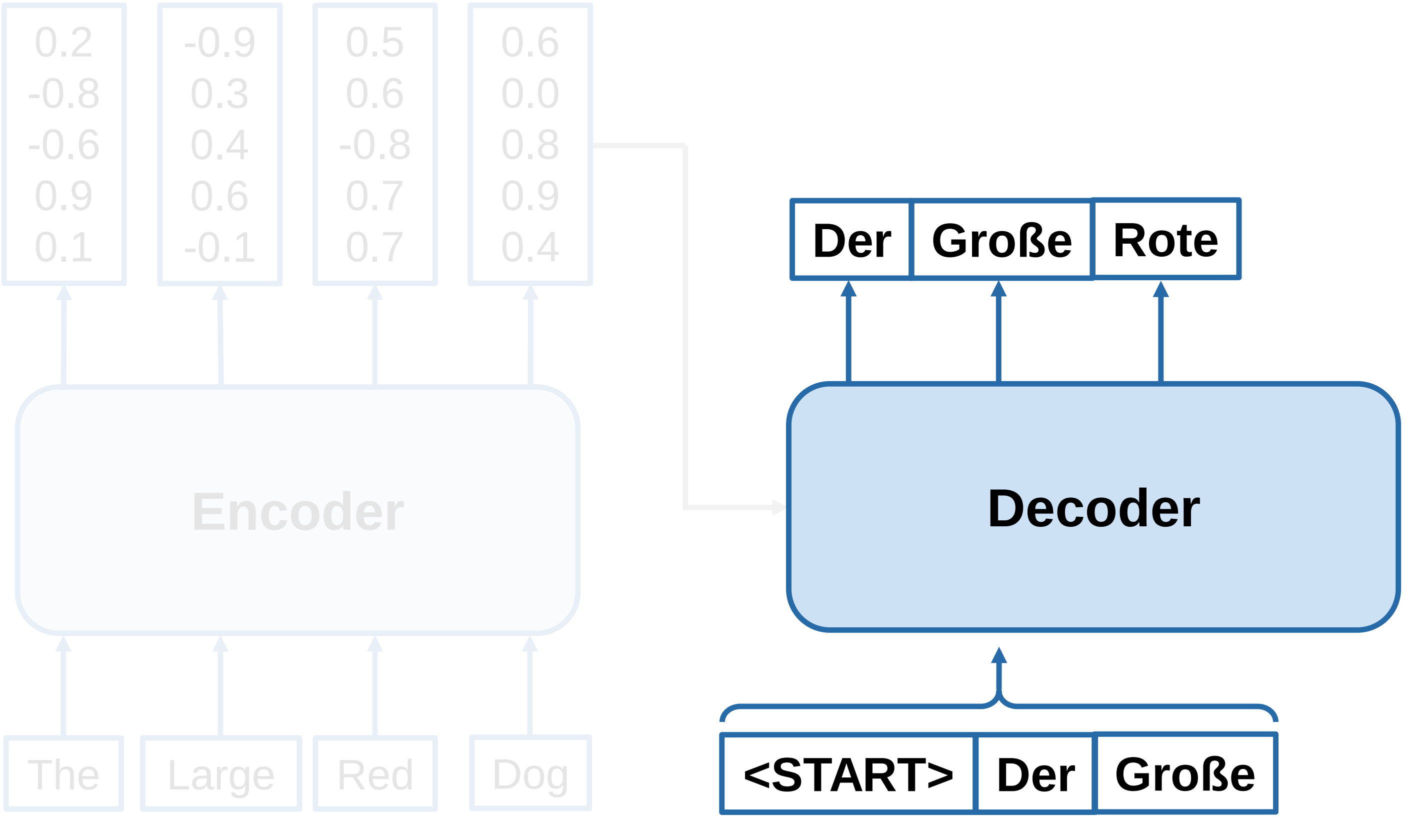


https://www.youtube.com/watch?v=0_4KEb08xrE

Encoder-Decoder architecture addresses *seq2seq* tasks

- Useful for translation and summarisation.

- Encoder and Decoder models do not necessarily share weights.



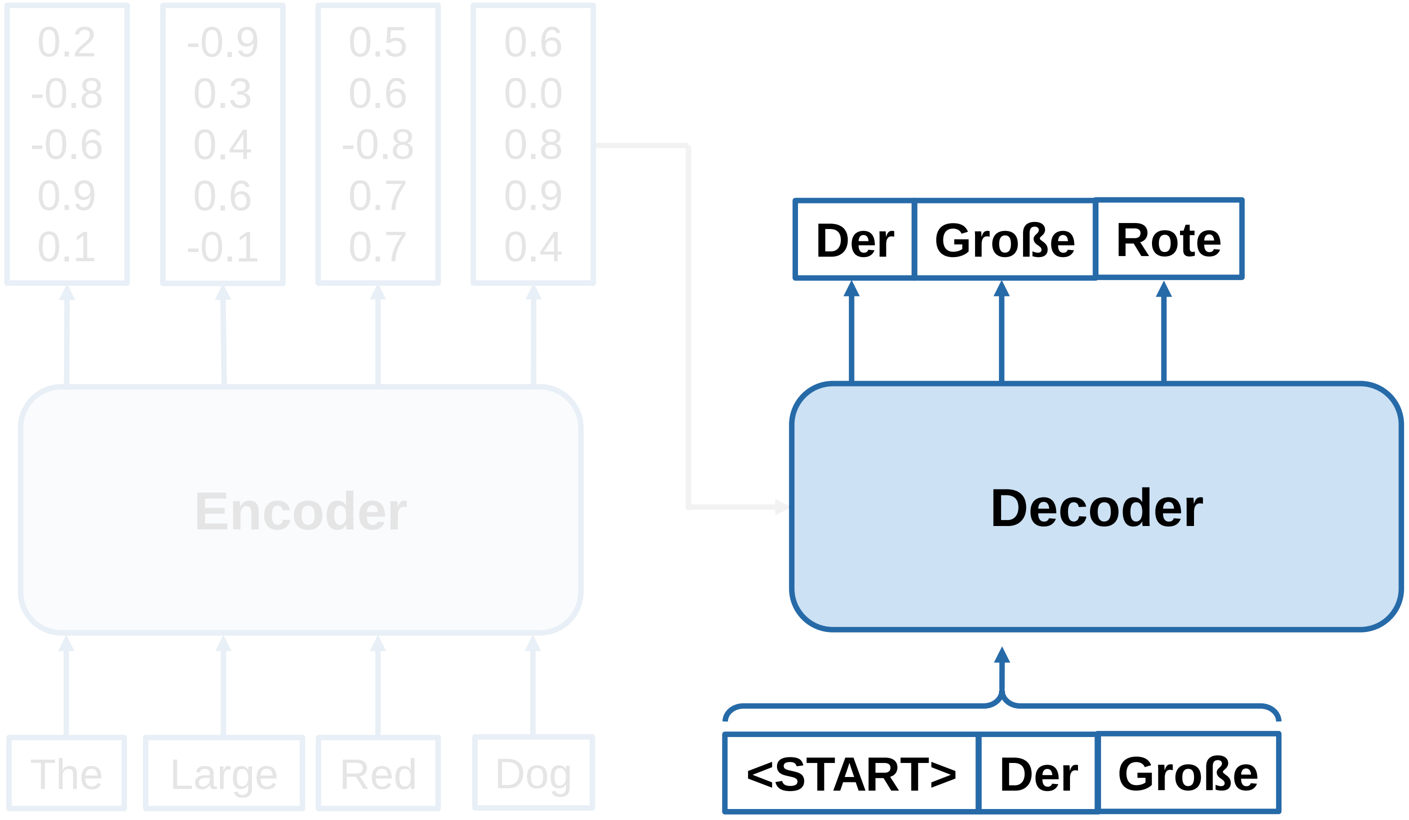
https://www.youtube.com/watch?v=0_4KEb08xrE

Encoder-Decoder architecture addresses $seq2seq$ tasks

- Useful for translation and summarisation.

- Encoder and Decoder models do not necessarily share weights.

- Input distribution differs from output distribution.



https://www.youtube.com/watch?v=0_4KEb08xrE

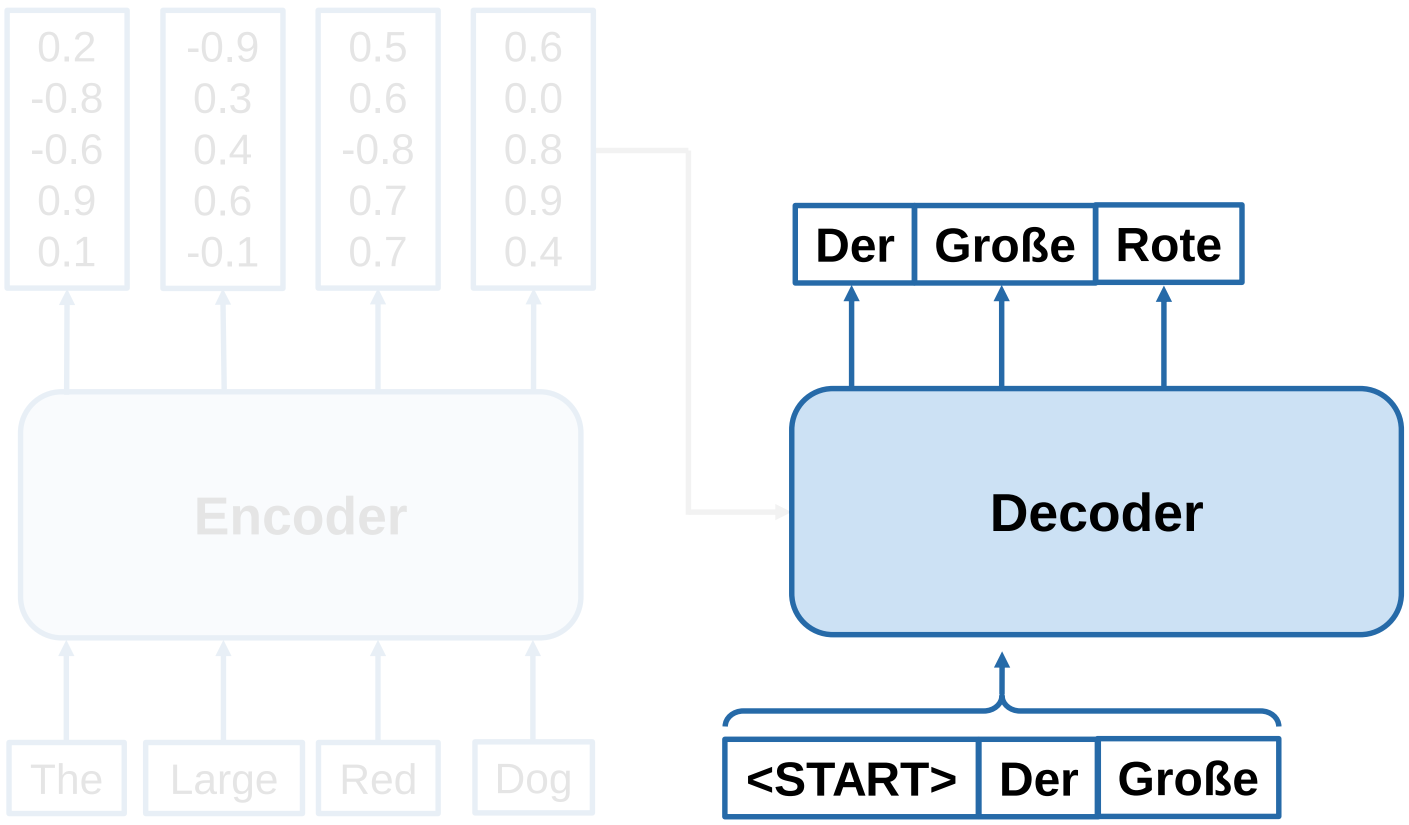
Encoder-Decoder architecture addresses *seq2seq* tasks

- Useful for translation and summarisation.

- Encoder and Decoder models do not necessarily share weights.

- Input distribution differs from output distribution.

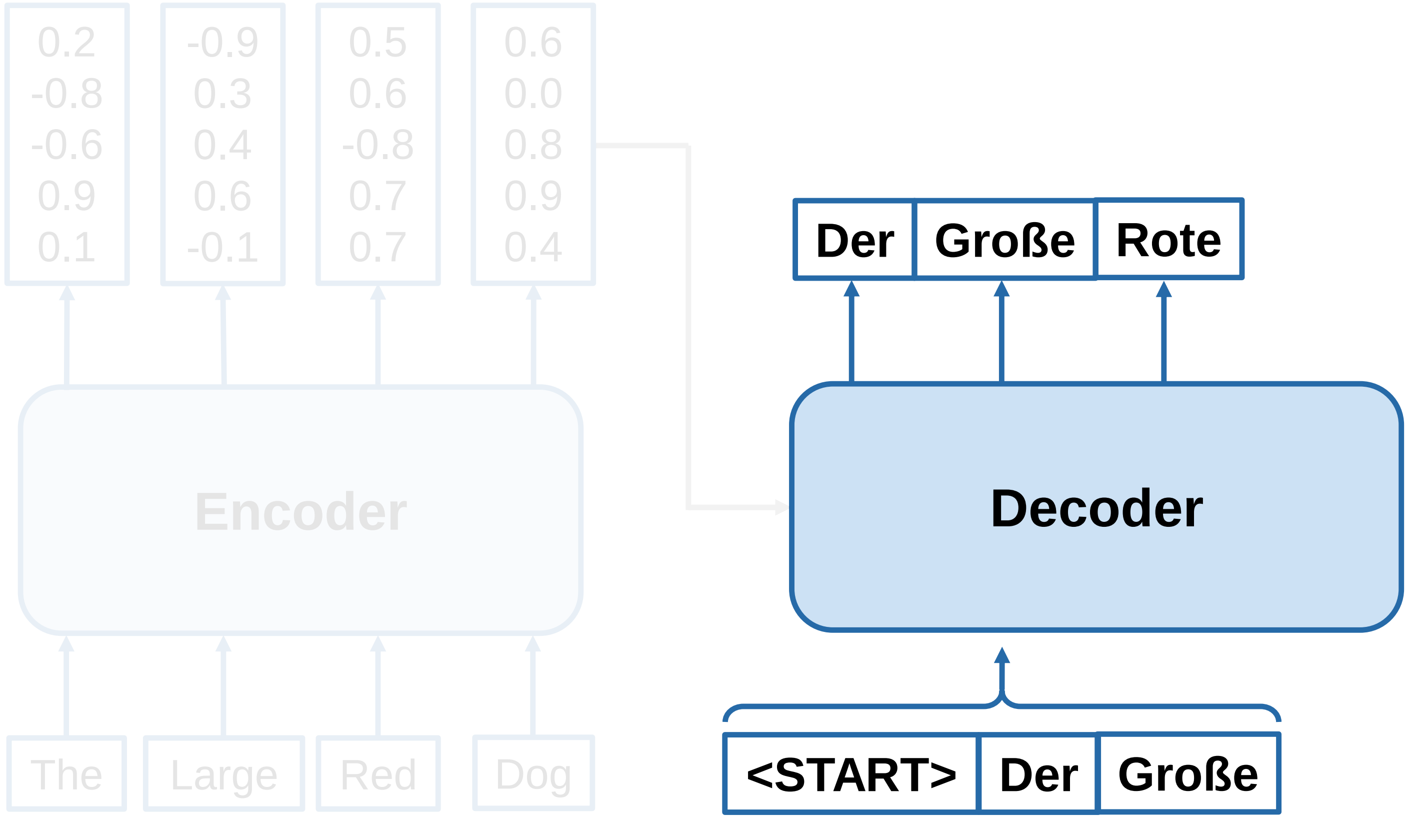
- Popular Enc-Dec models are T5, BART, M2M100, and Pegasus etc.



https://www.youtube.com/watch?v=0_4KEb08xrE

Encoder-Decoder architecture addresses *seq2seq* tasks

- Useful for translation and summarisation.
- Encoder and Decoder models do not necessarily share weights.
- Input distribution differs from output distribution.
- Popular Enc-Dec models are T5, BART, M2M100, and Pegasus etc.
- Can mix-and-match different stand-along encoders and decoders.



https://www.youtube.com/watch?v=0_4KEb08xrE

Do you have any problem?

Some material (images, tables, text etc.) in this presentation has been borrowed from different books, lecture notes, and the web. The original contents solely belong to their owners, and are used in this presentation only for clarifying various educational concepts. Any copyright infringement is ***not at all*** intended.