FLORIDA STATE
UNIVERSITY

The Transformer Architecture

Olmo Zavala-Romero and ChatGPT

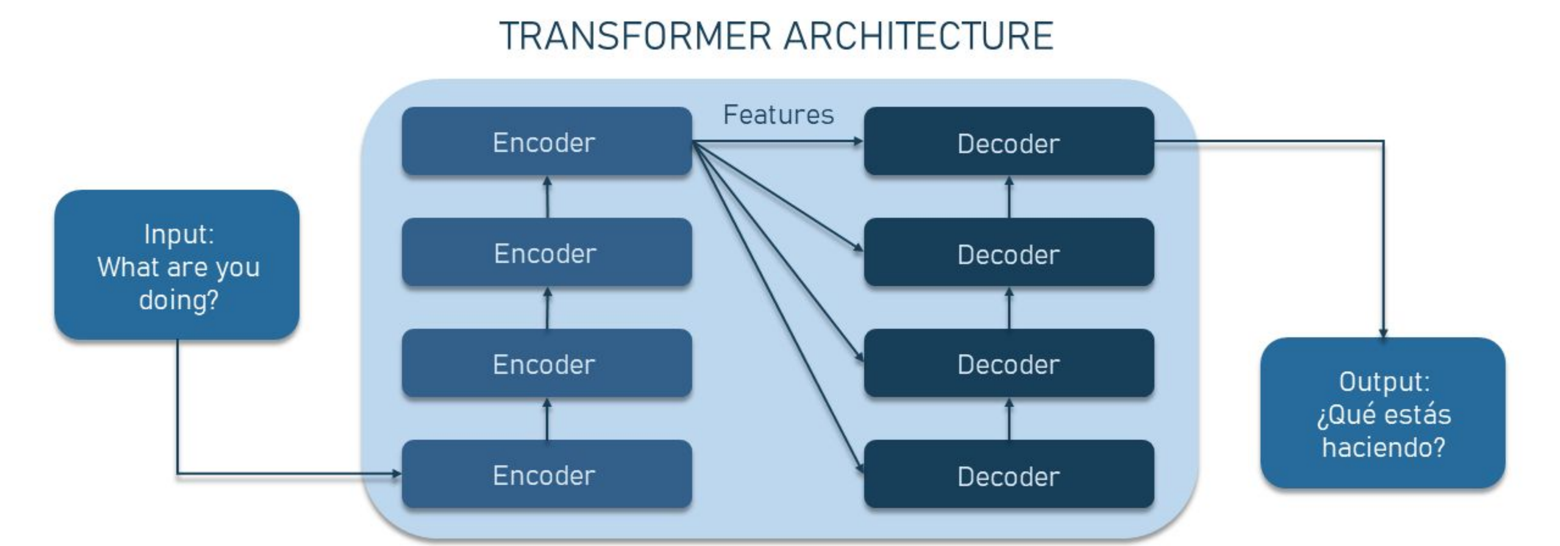
FLORIDA STATE UNIVERSITY

- Objective
  - *Good* understanding of the architecture

- Outline
  - Motivation
  - Evolution of Sequence Models
  - Need of Transformers
  - Transformers architecture components
    - Embedding
      - Positional encoding
    - Attention
      - Multi-head attention

What is the transformer architecture?

The Transformer architecture is a type of neural network architecture introduced in the paper "Attention Is All You Need" by Vaswani et al. in 2017. It's primarily designed for sequence-to-sequence tasks like machine translation but has been successfully adapted for a variety of other tasks. The architecture consists of an Encoder-Decoder framework, although variations like BERT use just the Encoder, and GPT uses just the Decoder for specific tasks.



https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

FLORIDA STATE UNIVERSITY

Motivation

Natural Language Processing (NLP):

1. **Machine Translation**: Transformers were initially proposed for this task and have become the state-of-the-art here.
2. **Text Summarization**: Automatic generation of concise summaries from long articles.
3. **Question Answering**: Models like answering questions based on a
4. **Named Entity Recognition (NER)** locations, and organizations in te
5. **Sentiment Analysis**: Classifying t data.
6. **Language Modeling**: Models like Transformer) are often used in te
7. **Speech Recognition**: Transforme converting spoken language into

Graph-based Learning:

1. **Molecular Design**: Predicting molecular p
2. **Social Network Analysis**: Analyzing relati

Time-Series Analysis:

1. **Financial Forecasting**: Using transformer predicting stock prices.
2. **Anomaly Detection**: Detecting anomalies

Bioinformatics:

1. **Protein Folding**: AlphaFold 2, which has r folding prediction, employs transformer-like architectures.

Audio and Speech:

1. **Music Generation**: Transformers have been adapted to generate musical sequences.
2. **Speech Synthesis**: Text-to-Speech systems like Tacotron 2 have started integrating transformer architectures.

Computer Vision:

1. **Image Classification**: Vision Transformer (ViT) is a notable application of the transformer architecture in computer vision.
2. **Object Detection**: Transformers have been used in frameworks like DETR for object detection tasks.
3. **Semantic Segmentation**: Segmenting parts of images based on semantic categories.

Multi-modal Learning:

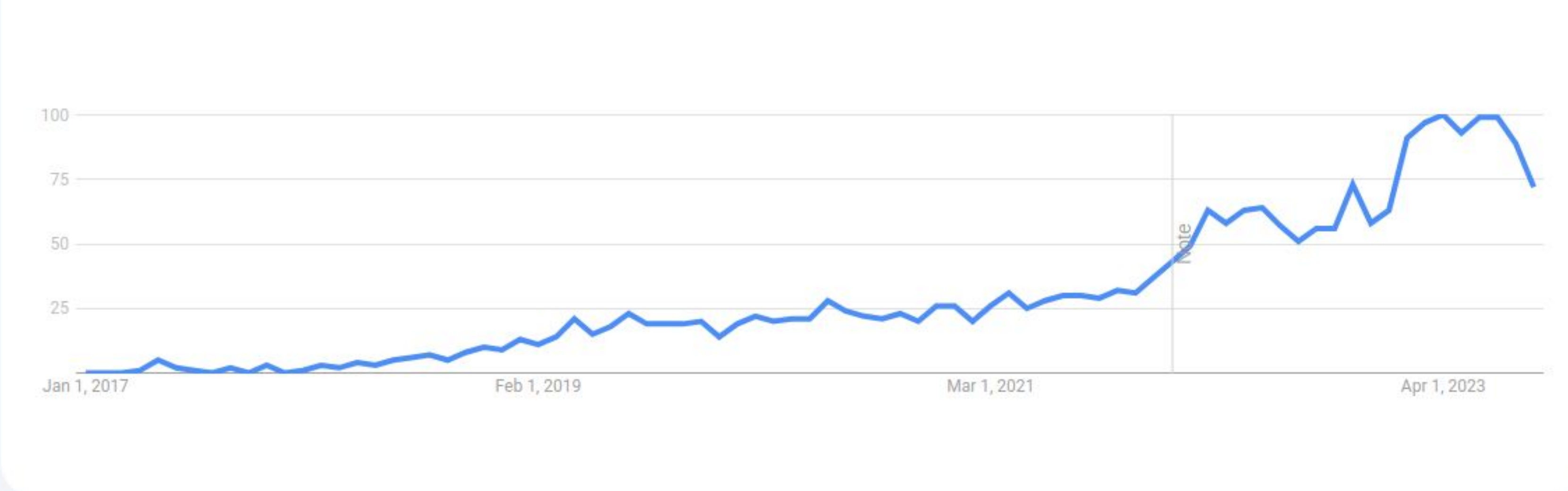
1. **Image Captioning**: Generating textual descriptions of images.
2. **Visual Question Answering**: Answering questions about an image o a series of images.

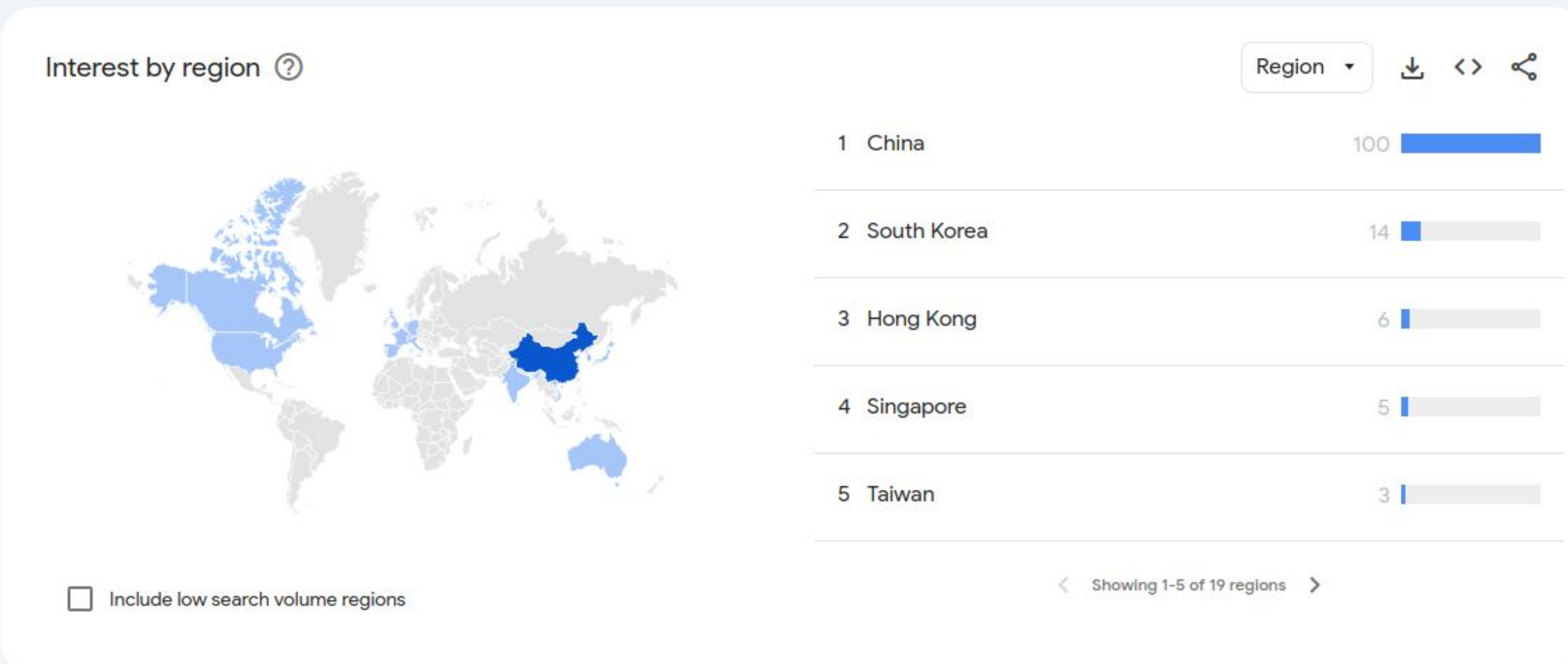
Reinforcement Learning:

1. **Game Playing**: Transformers have been used in learning policies for complex games.
2. **Multi-Agent Systems**: Utilized in coordinating the behavior of multiple agents.

FL

Trends for “Transformer attention”





Transformers (sources by Tom)

High-level, easy to read intro:
https://www.altexsoft.com/blog/language-models-gpt/

Paper introducing the Transformer Model:
https://arxiv.org/pdf/1706.03762.pdf

Detailed/annotated implementation:
http://nlp.seas.harvard.edu/annotated-transformer/

In my view accessible explanation of the workings of the model:
https://e2eml.school/transformers.html

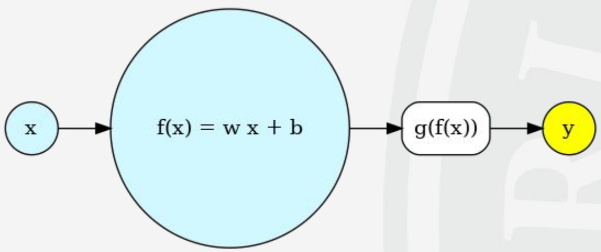
History of Sequence models

Original objectives

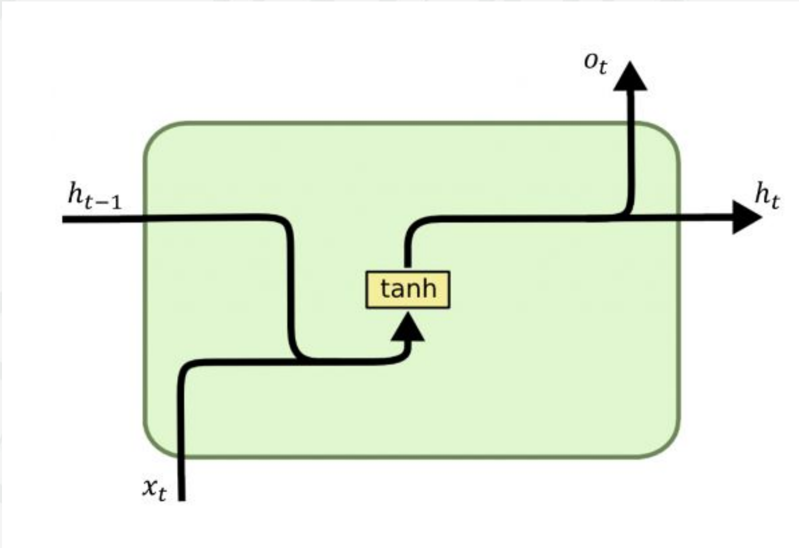
- Temporal/Order Dependencies
- Variable-Length Sequences

"Anita lava la tina" --> "la tina lava a Anita"

Recurrent Neural Networks



$$L = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - g(f(x_i))\right)^2$$



$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

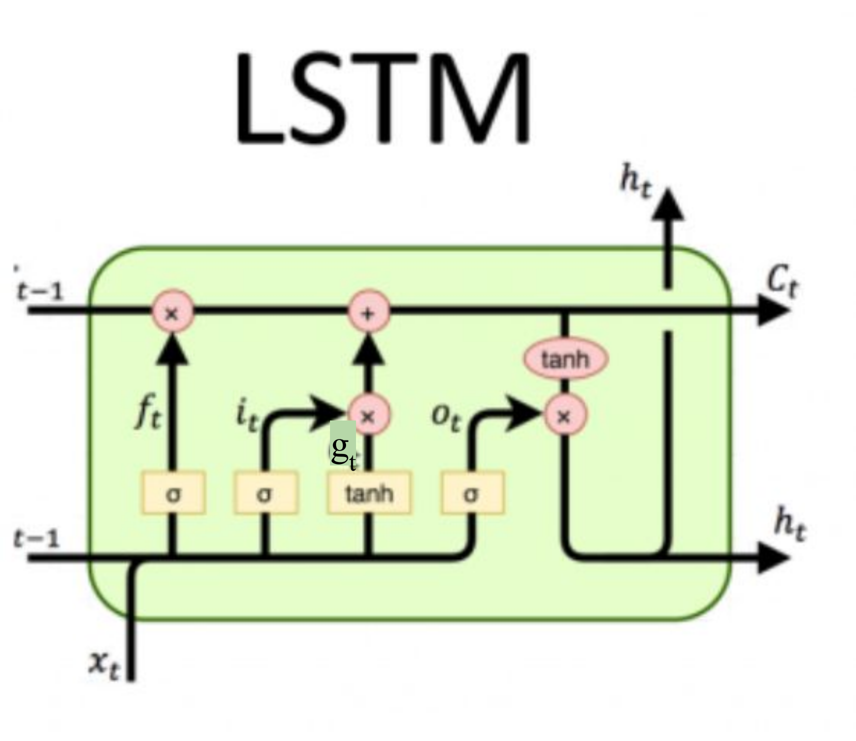
[Torch RNN]

[http://dprogrammer.org/rnn-lstm-gru]

FLORIDA STATE UNIVERSITY

LSTM

- Handling Long-Range Dependencies
- Avoid vanishing gradient



- i → input
- f → forget
- g → cell
- o → output
- c → context
- h → hidden state

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

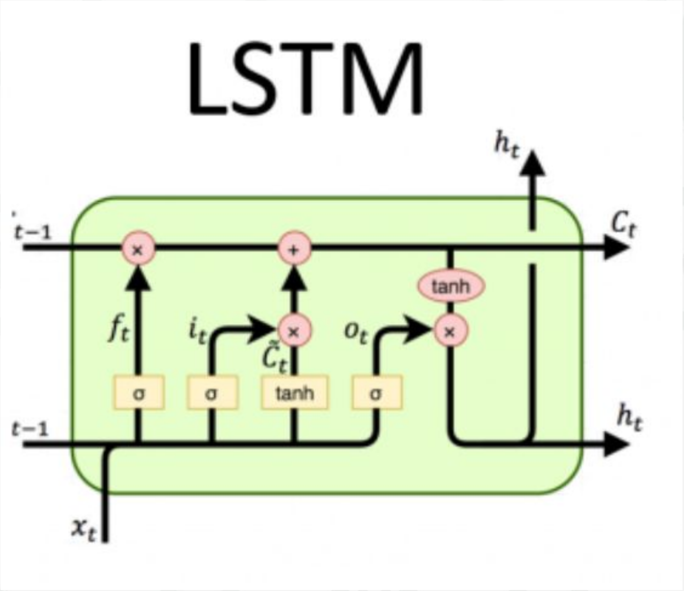
http://dprogrammer.org/rnn-lstm-gru

FLORIDA STATE UNIVERSITY

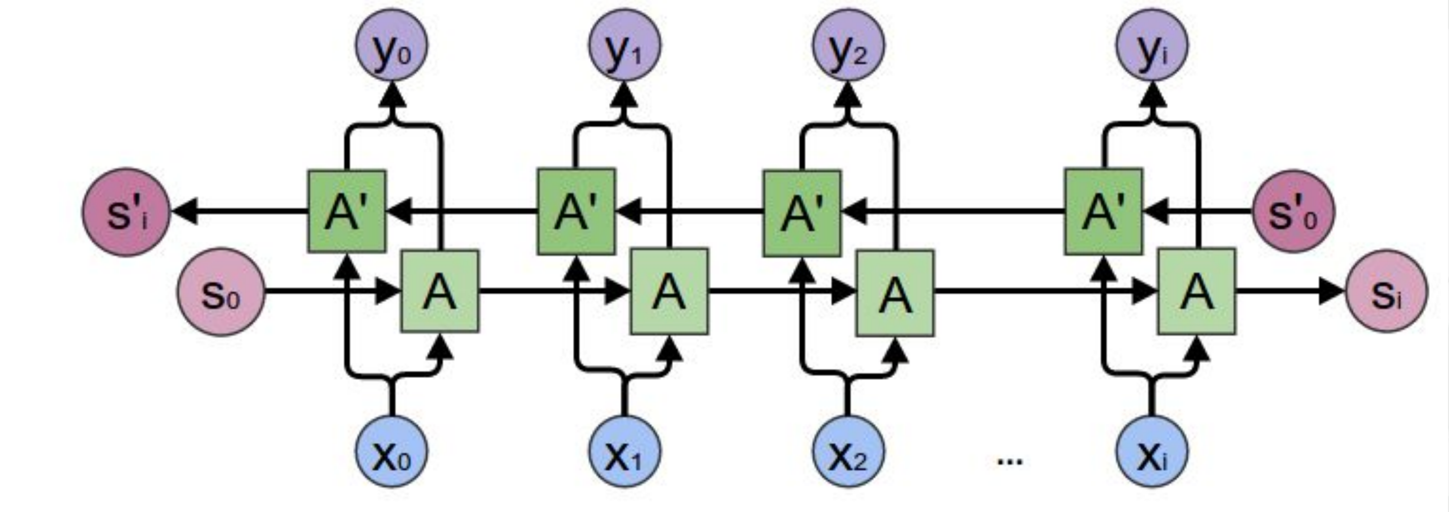
Need for Transformers

Limitations (RNNs, LSTMs, GRUs)

- Parallelization
- Handling Long Sequences
- Computational Complexity

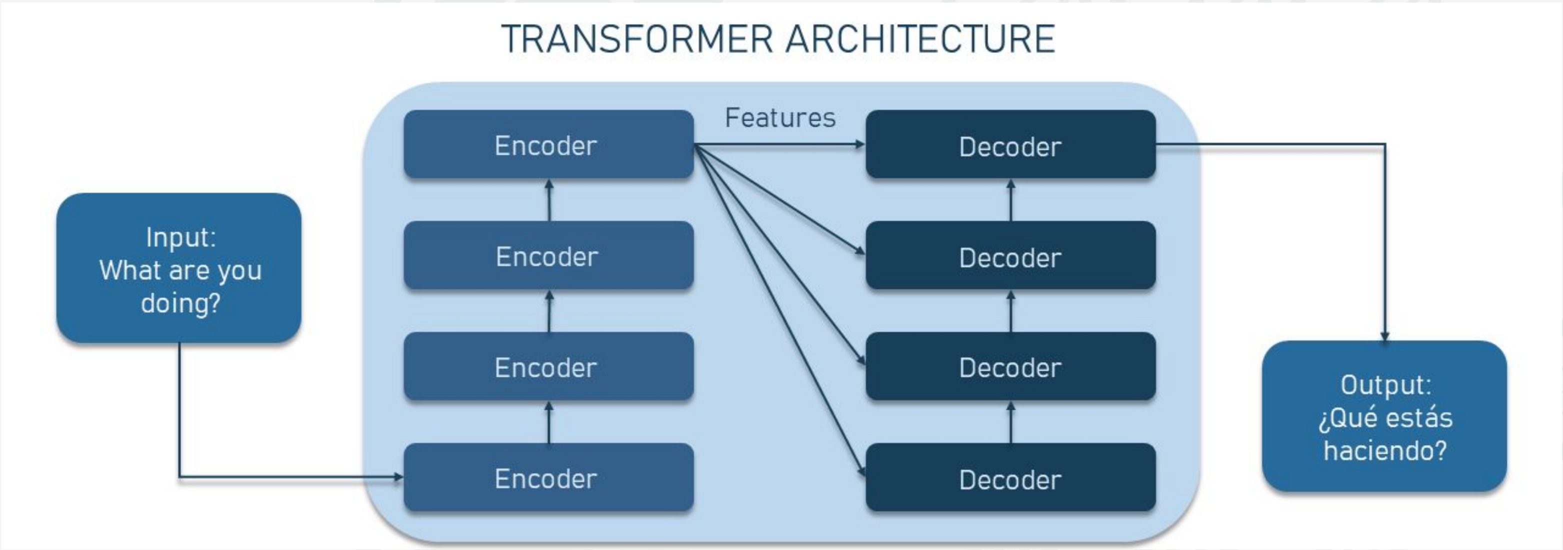


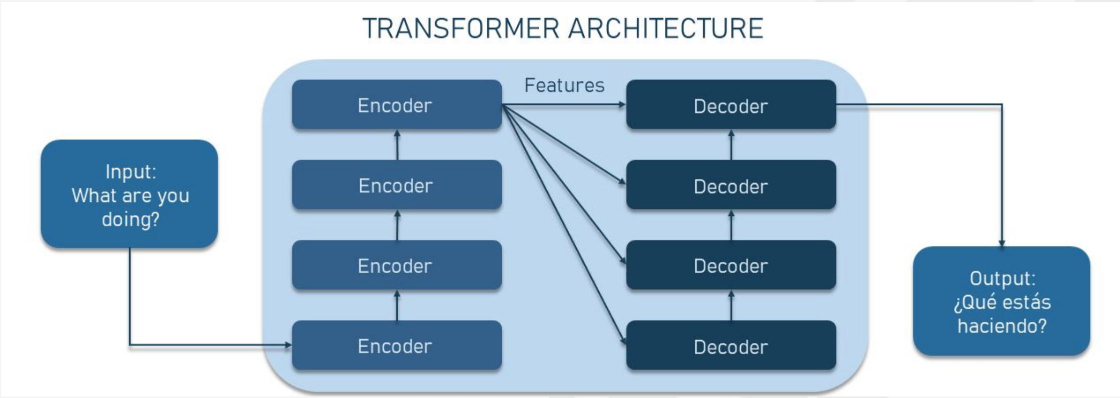
BiRNN

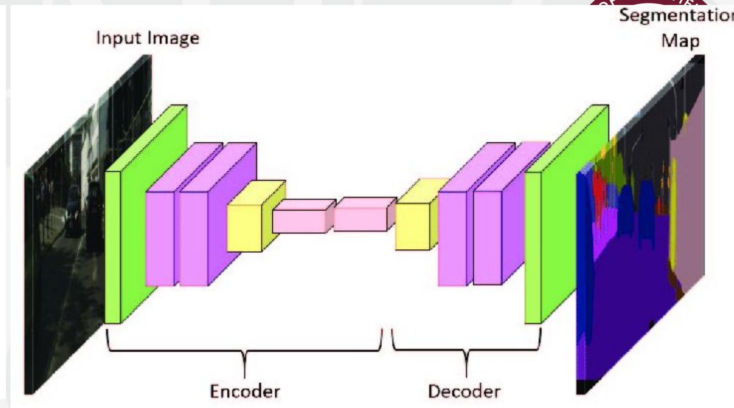


- Context awareness
- Improved accuracy

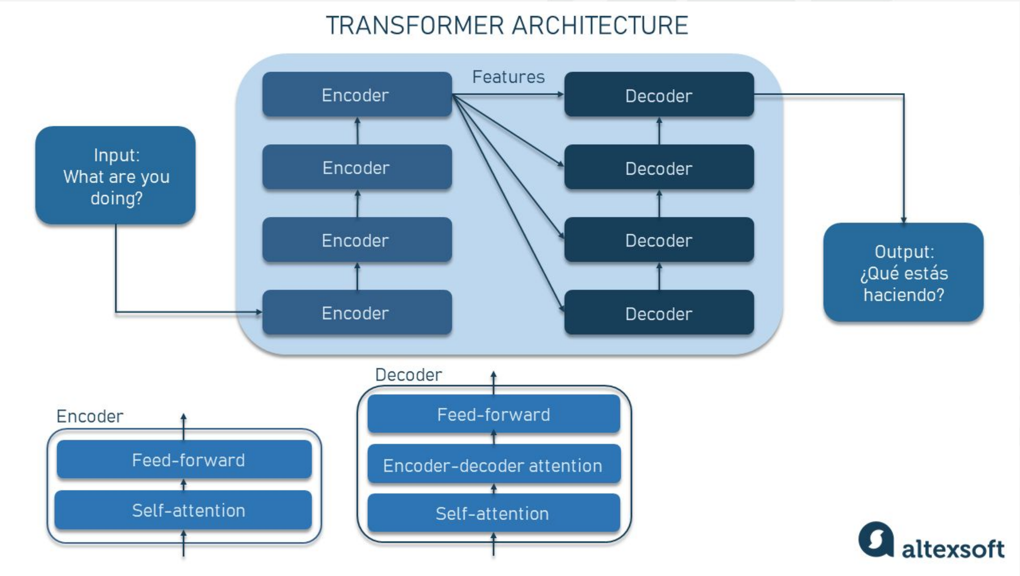
Transformer Architecture




TRANSFORMER ARCHITECTURE
Input:
What are you
doing?
Encoder
Encoder
Encoder
Encoder
Features
Decoder
Decoder
Decoder
Decoder
Output:
¿Qué estás
haciendo?


Input Image
Segmentation
Map
Encoder
Decoder

Transformer



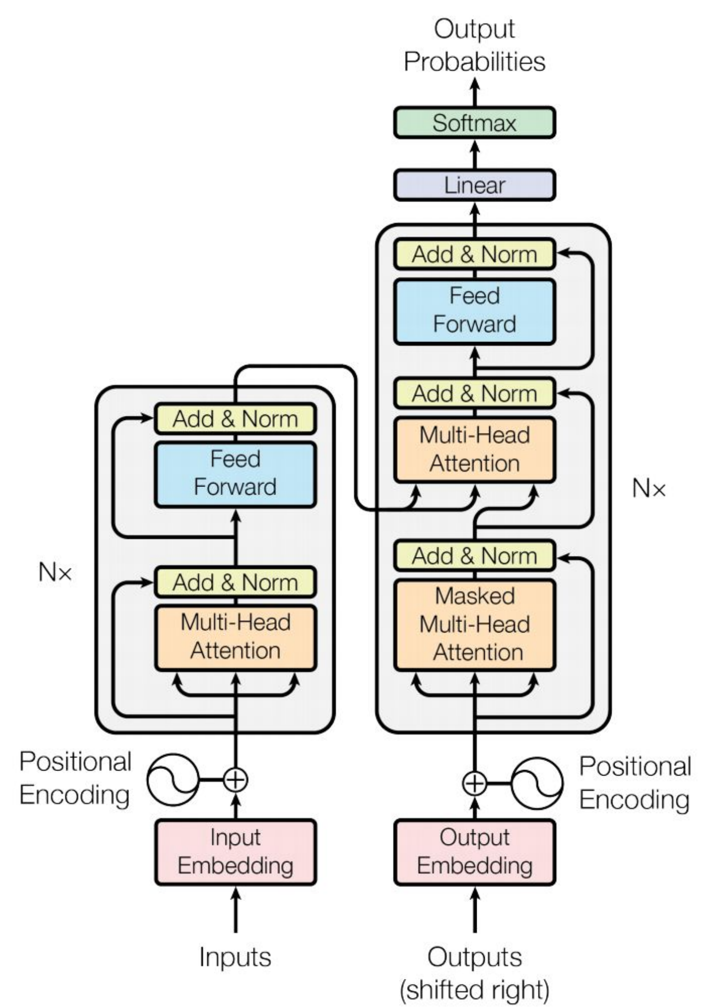


Figure 1: The Transformer - model architecture.

Main components

- Encoder - Decoder
- Feed forward networks
- Embeddings
- Positional encoding
- Attention
  - Multi-head attention
  - Masked Multi-Head Attention

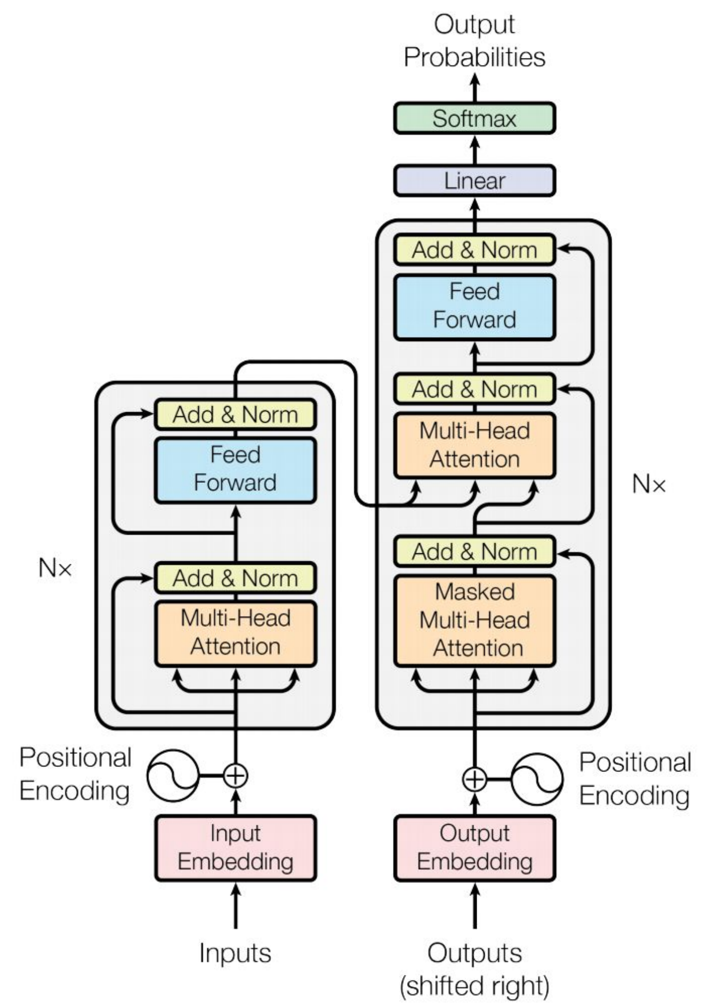


Figure 1: The Transformer - model architecture.

FLORIDA STATE UNIVERSITY

Main components

- Encoder - Decoder
- Feed forward networks
- Embedding
- Positional encoding
- Attention
  - Multi-head attention
  - Masked Multi-Head Attention

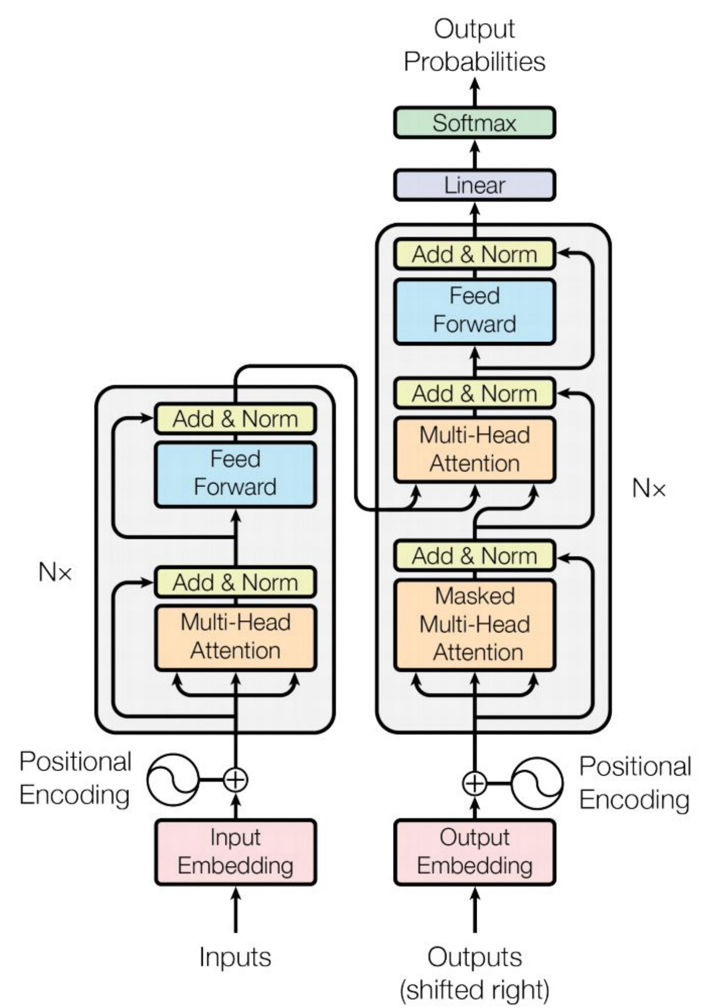


Figure 1: The Transformer - model architecture.

Word Representations

One hot encoding

Paris
Rome   word V

```
Rome   = [1, 0, 0, 0, 0, 0, …, 0]
Paris  = [0, 1, 0, 0, 0, 0, …, 0]
Italy  = [0, 0, 1, 0, 0, 0, …, 0]
France = [0, 0, 0, 1, 0, 0, …, 0]
```

Source :(Marco Bonzanini, 2017)

https://medium.com/intelligentmachines/word-embedding-and-one-hot-encoding-ad17b4bbe111

FLORIDA STATE UNIVERSITY

Word Representations

Word embedding

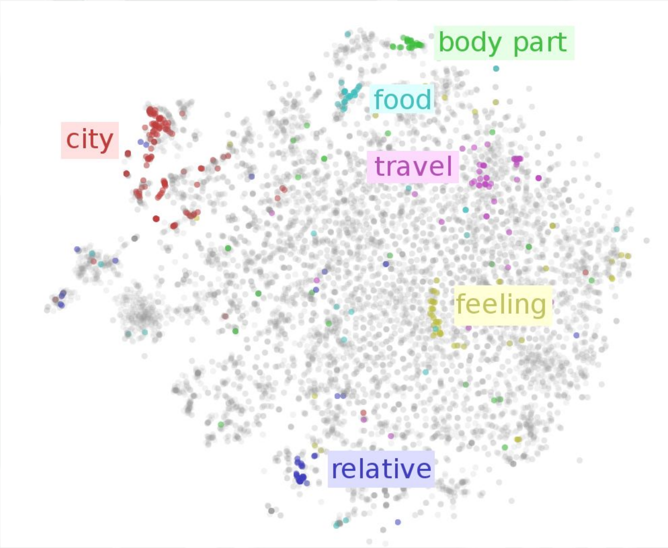
	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
⋮						

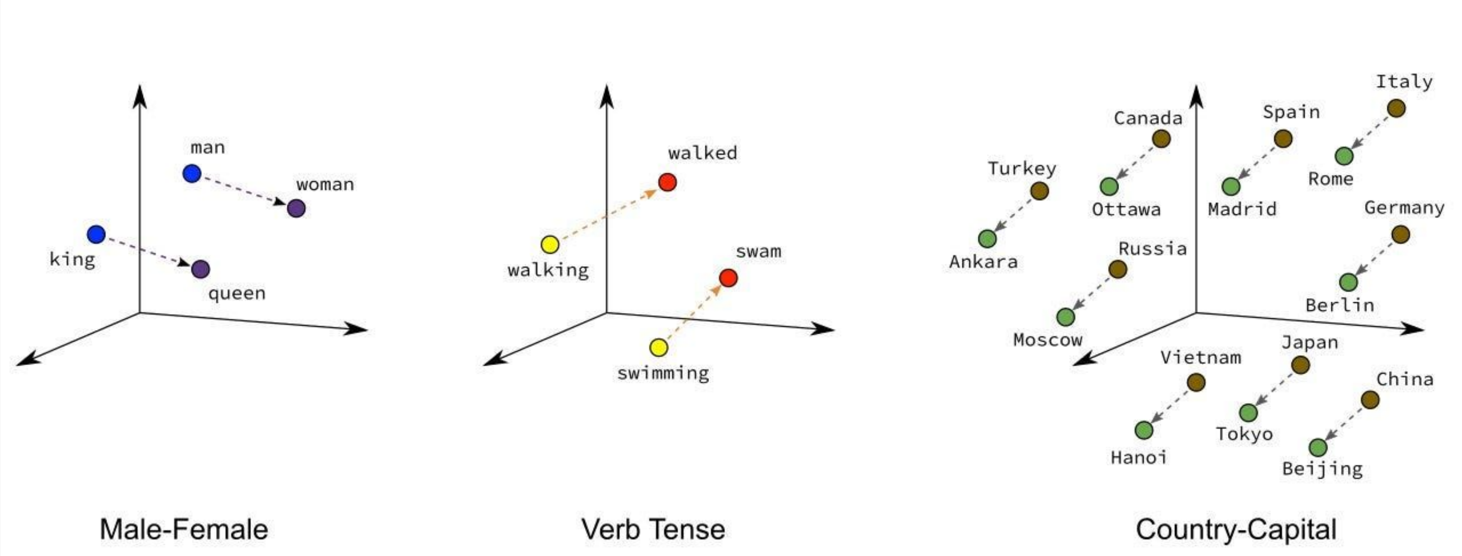
Represent your vocabulary as an N dimensional vector where ‘related’ words are ‘closer’ together.

[Example file to show the learning of embeddings]()

FLORIDA STATE UNIVERSITY

Word embeddings





How to learn embeddings?

The objective is to maximize the following likelihood function:

$$L = \sum_{t=1}^{T} \sum_{-c \le j \le c, j \ne 0} \log p(w_{t+j} | w_t)$$

Where $p(w_o | w_i)$ is defined as:

$$p(w_o | w_i) = \frac{\exp(v'_{w_o} \cdot v_{w_i})}{\sum_{w=1}^{V} \exp(v'_w \cdot v_{w_i})}$$

1. Initialize word vectors randomly.
2. Iterate through each word in the corpus and use surrounding words as context.
3. Update the word vectors by optimizing the objective function, typically using SGD or some variant.

Main components

- Encoder - Decoder
- Feed forward networks
- Embedding
- Positional encoding
- Attention
  - Multi-head attention
  - Masked Multi-Head Attention

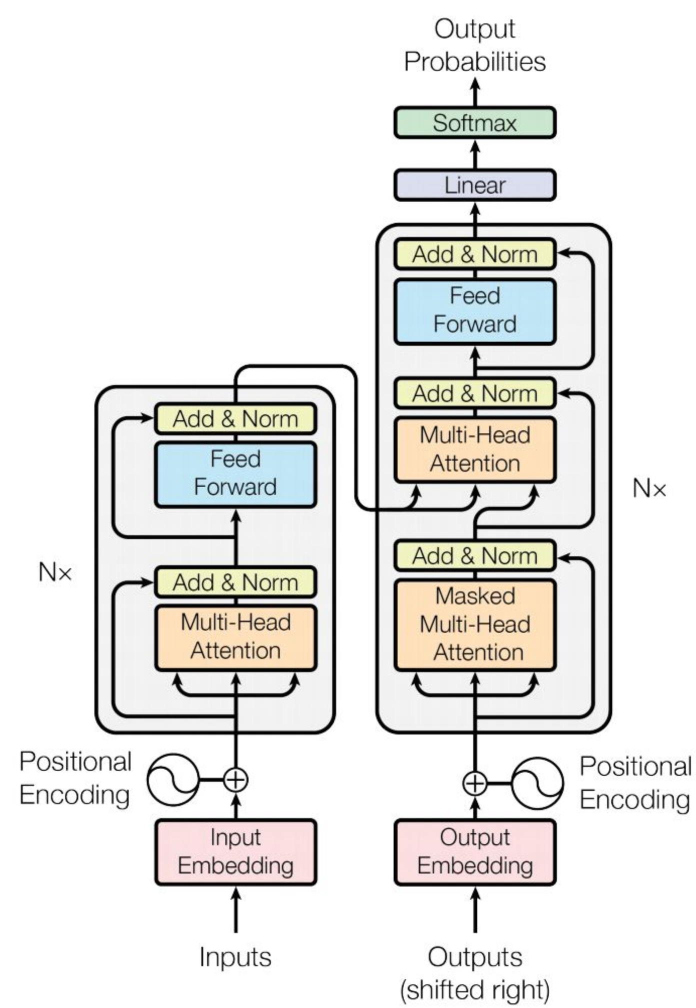


Figure 1: The Transformer - model architecture.

FLORIDA STATE UNIVERSITY

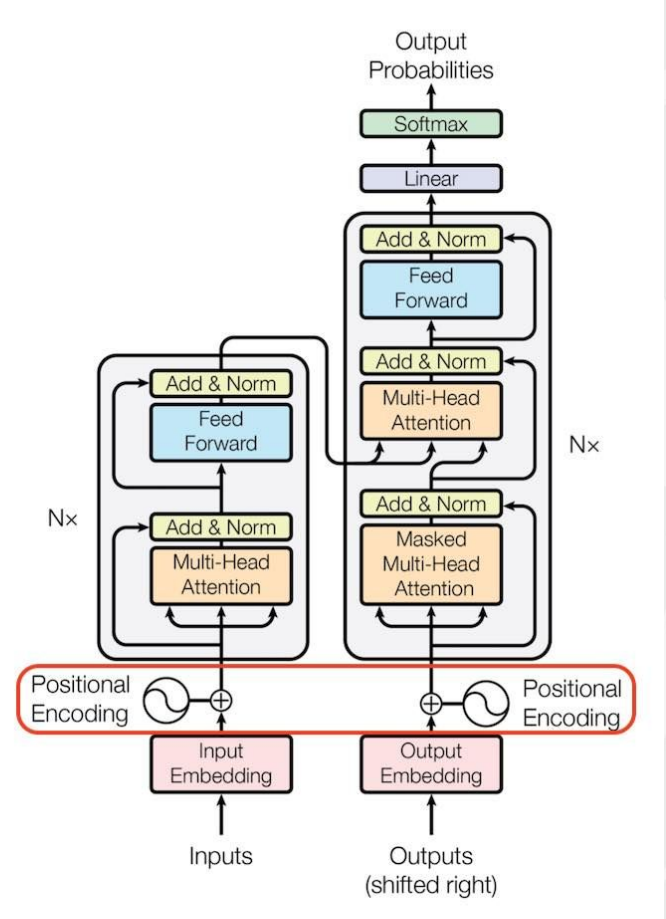
Positional Encoding

Why?

- Transformers doesn't know the order of the words (embeddings)

What do we want?

- Unique encoding for each location-word combination
- Distance *between* locations should be consistent
- Generalize to longer sentences



https://e2eml.school/transformers.html

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

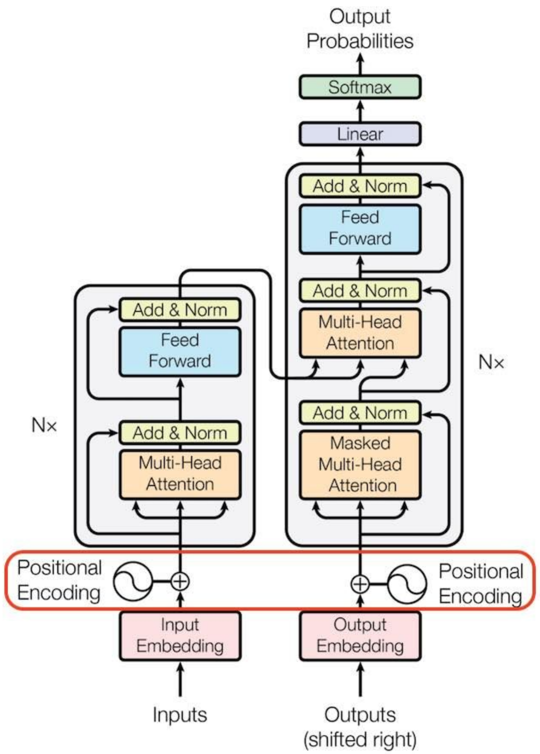
How?

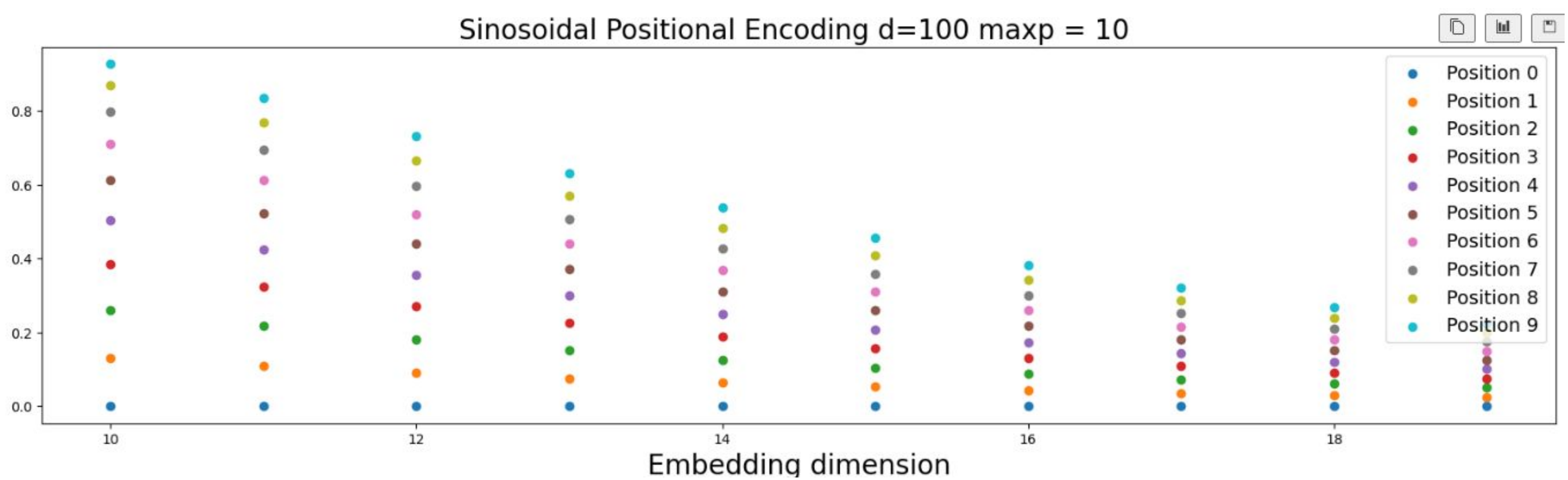
1. **Sinusoidal Positional Embeddings**: One popular method involves using sinusoidal functions to generate these embeddings. Given a position p in the sequence and a dimension i in the embedding, the sinusoidal positional encoding for that entry is computed as:

$$PE_{(p,2i)} = \sin\left(\frac{p}{10000^{2i/d}}\right)$$

$$PE_{(p,2i+1)} = \cos\left(\frac{p}{10000^{2i/d}}\right)$$

where PE is the positional encoding matrix, p is the position, i ranges over the dimensions, and d is the embedding dimension. This method is easy to compute and can be scaled for sequences of variable lengths.





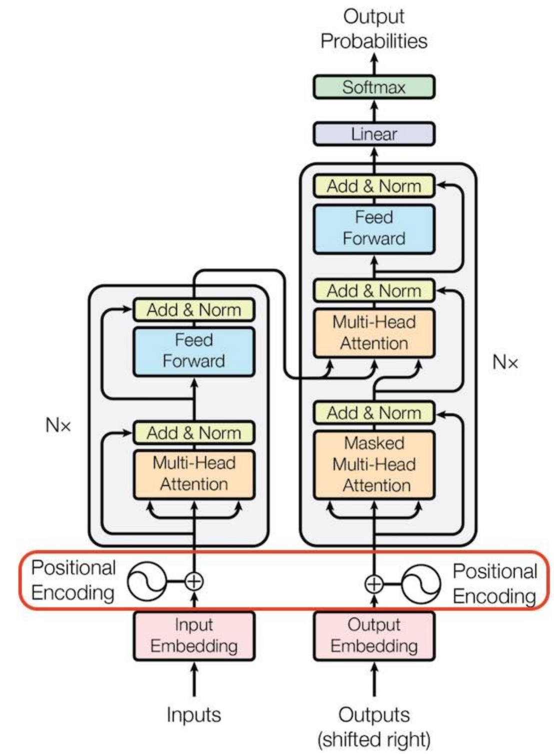
How?

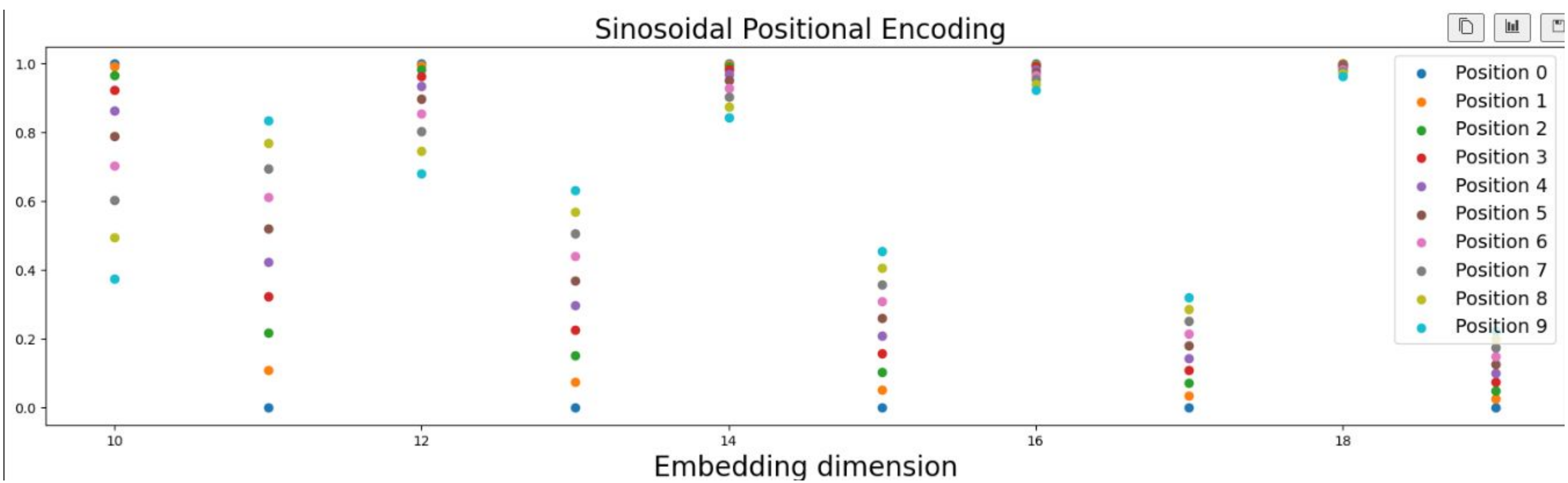
1. **Sinusoidal Positional Embeddings**: One popular method involves using sinusoidal functions to generate these embeddings. Given a position p in the sequence and a dimension i in the embedding, the sinusoidal positional encoding for that entry is computed as:

$$PE_{(p,2i)} = \sin\left(\frac{p}{10000^{2i/d}}\right)$$

$$PE_{(p,2i+1)} = \cos\left(\frac{p}{10000^{2i/d}}\right)$$

where PE is the positional encoding matrix, p is the position, i ranges over the dimensions, and d is the embedding dimension. This method is easy to compute and can be scaled for sequences of variable lengths.





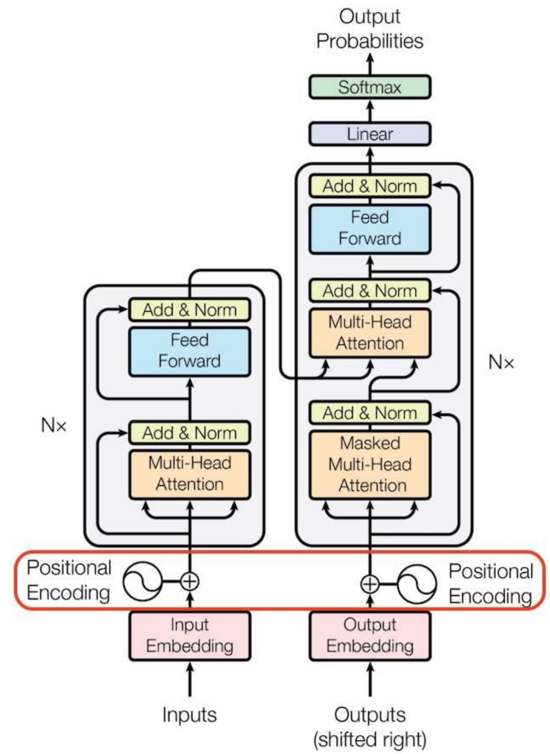
How?

1. **Sinusoidal Positional Embeddings**: One popular method involves using sinusoidal functions to generate these embeddings. Given a position p in the sequence and a dimension i in the embedding, the sinusoidal positional encoding for that entry is computed as:

$$PE_{(p,2i)} = \sin\left(\frac{p}{10000^{2i/d}}\right)$$

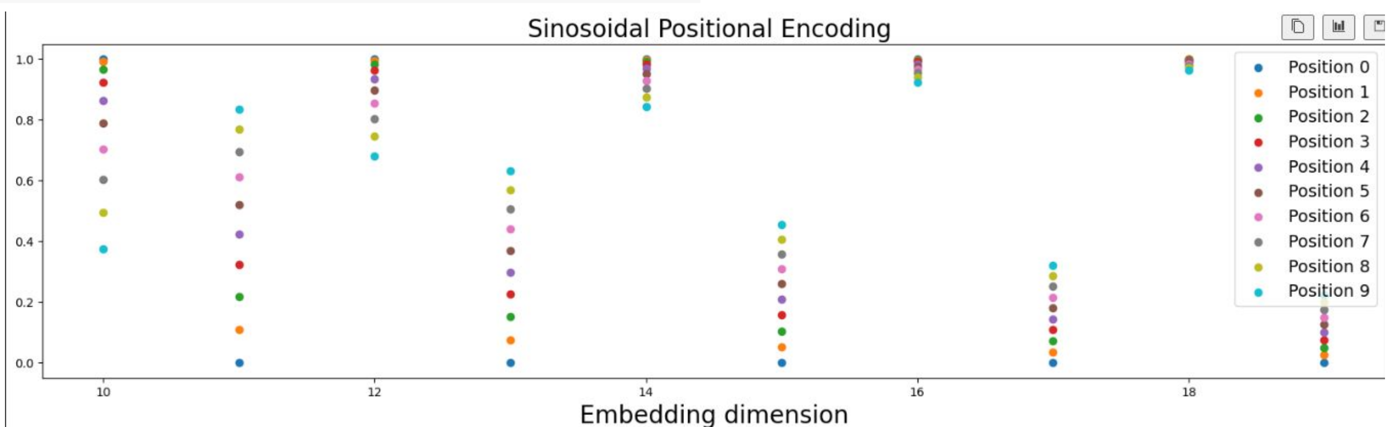
$$PE_{(p,2i+1)} = \cos\left(\frac{p}{10000^{2i/d}}\right)$$

where PE is the positional encoding matrix, p is the position, i ranges over the dimensions, and d is the embedding dimension. This method is easy to compute and can be scaled for sequences of variable lengths.



$$E' = E + PE$$

Here E' is the enhanced embedding, E is the original word embedding, and PE is the positional embedding.



FLORIDA STATE UNIVERSITY

Main components

- Encoder - Decoder
- Feed forward networks
- Embedding
- Positional encoding
- Attention
  - Self-attention
  - Multi-head attention
  - Masked Multi-Head Attention

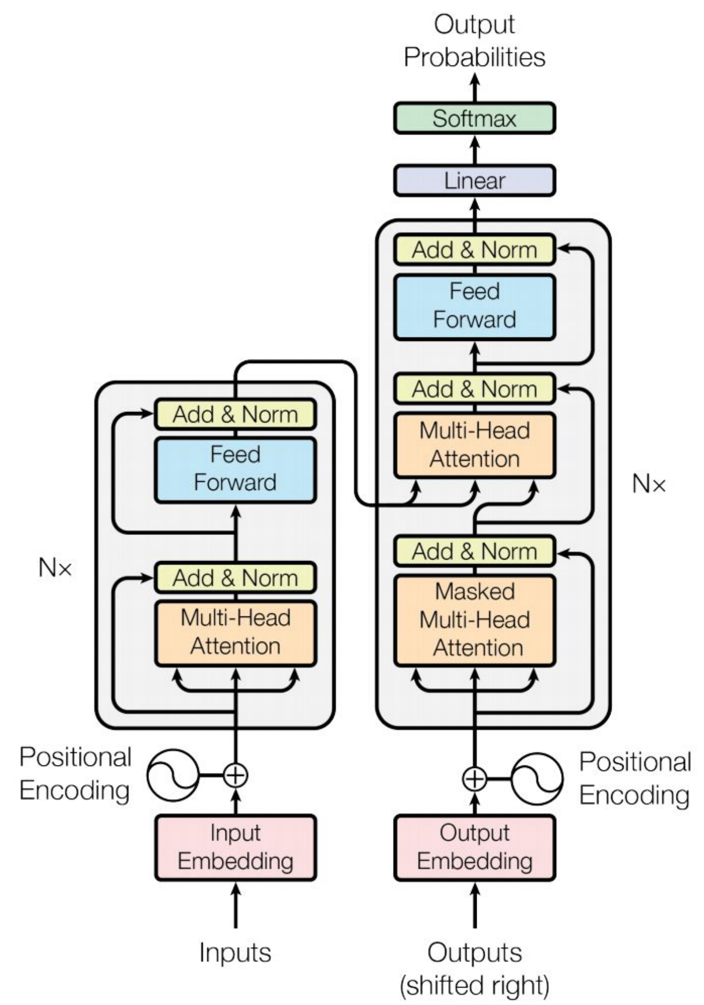


Figure 1: The Transformer - model architecture.

FLORIDA STATE UNIVERSITY

Concept of Attention

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

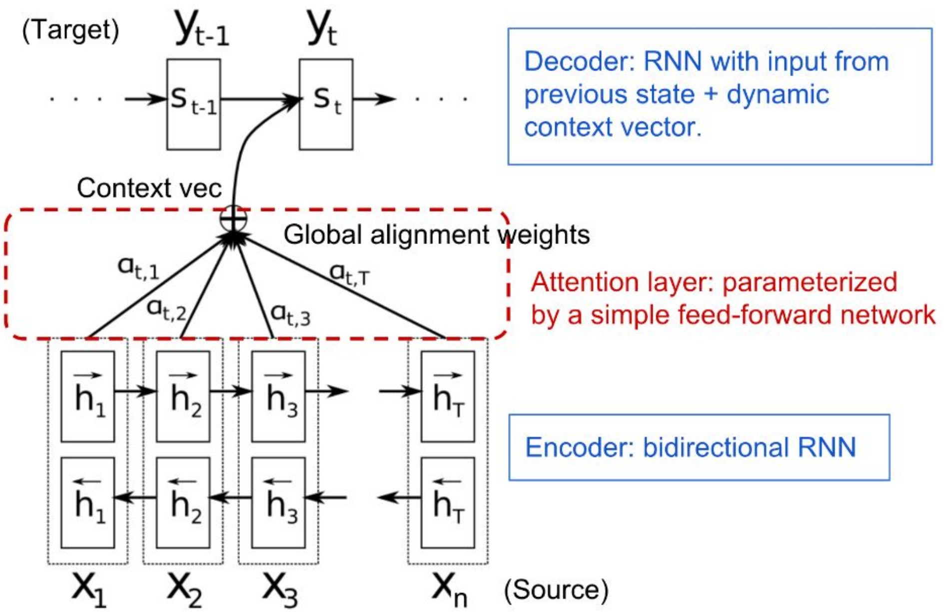
Attention

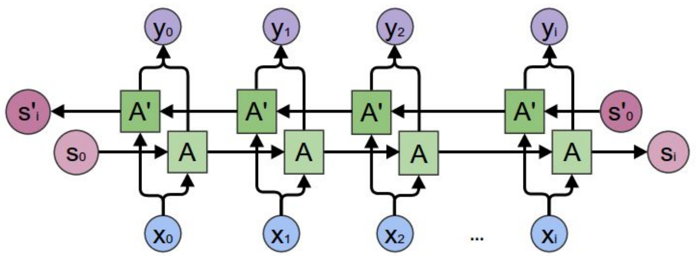
[redacted]

[redacted]

https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Attention





The decoder network has hidden state $\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}, y_{t-1}, \mathbf{c}_t)$ for the output word at position t, $t = 1, \dots, m$, where the context vector $\mathbf{c}_t$ is a sum of hidden states of the input sequence, weighted by alignment scores:

$$\mathbf{c}_t = \sum_{i=1}^{n} \alpha_{t,i} \boldsymbol{h}_i \qquad \boldsymbol{h}_i = [\overrightarrow{\boldsymbol{h}}_i^\top ; \overleftarrow{\boldsymbol{h}}_i^\top]^\top, i = 1, \dots, n \qquad \text{; Context vector for output } y_t$$

$$\alpha_{t,i} = \text{align}(y_t, x_i) \qquad \text{; How well two words } y_t \text{ and } x_i \text{ are aligned.}$$

$$= \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))} \qquad \text{; Softmax of some predefined alignment score..}$$

$$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t ; \boldsymbol{h}_i])$$

where both $\mathbf{v}_a$ and $\mathbf{W}_a$ are weight matrices to be learned in the alignment model.

https://lilianweng.github.io/posts/2018-06-24-attention/

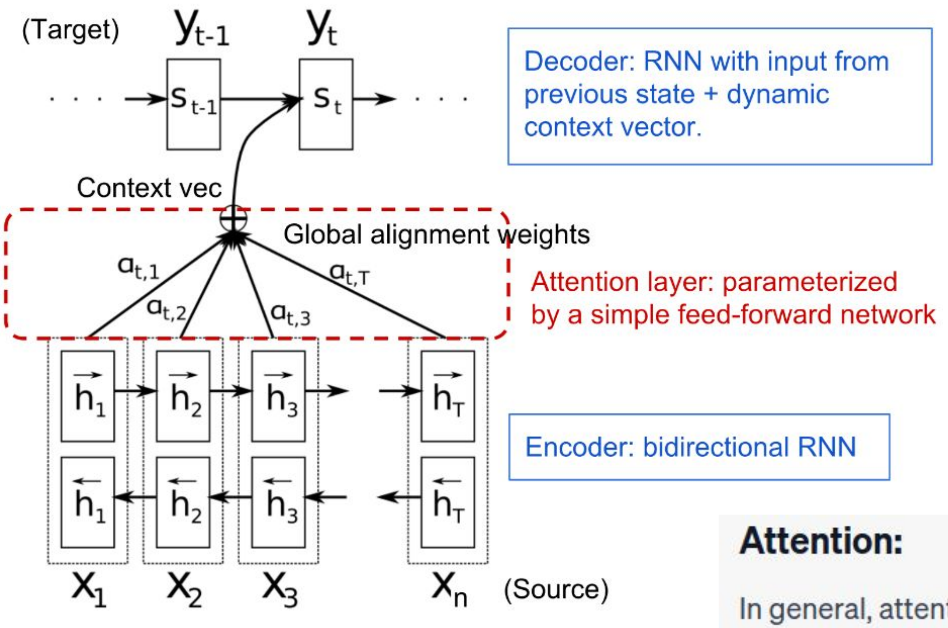
FLORIDA STATE UNIVERSITY

Attention mechanisms

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_{t-1}; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

https://lilianweng.github.io/posts/2018-06-24-attention/

Attention



$$\mathbf{c}_t = \sum_{i=1}^{n} \alpha_{t,i} \boldsymbol{h}_i$$

$$\alpha_{t,i} = \text{align}(y_t, x_i) = \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

Dot-Product $\quad \text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$

Attention:

In general, attention mechanisms allow a model to focus on certain parts of the input when producing the output. The basic idea is to take a set of queries (Q), keys (K), and values (V) as input and produce weighted combinations of the values as output.

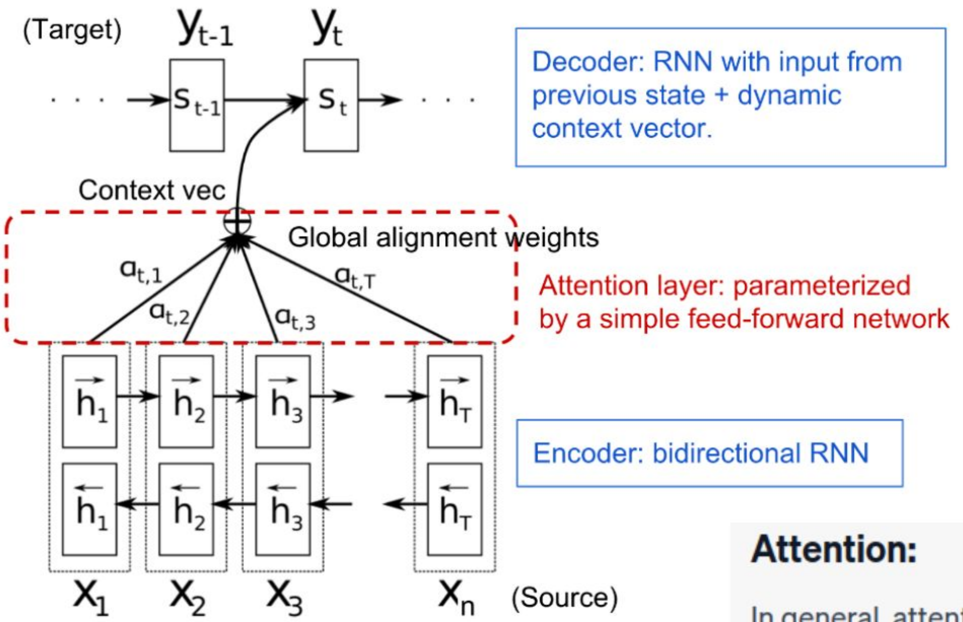
The attention function is generally formulated as:

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

In sequence-to-sequence models like those used in machine translation, "attention" often refers to the mechanism where the decoder pays attention to different parts of the encoder's output while generating each element of the decoded sequence. In this case, Q usually comes from the decoder, and K and V come from the encoder.

https://lilianweng.github.io/posts/2018-06-24-attention/

Attention



Attention:

In general, attention mechanisms allow a model to focus on certain parts of the input when producing the output. The basic idea is to take a set of queries (Q), keys (K), and values (V) as input and produce weighted combinations of the values as output.

The attention function is generally formulated as:

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

In sequence-to-sequence models like those used in machine translation, "attention" often refers to the mechanism where the decoder pays attention to different parts of the encoder's output while generating each element of the decoded sequence. In this case, Q usually comes from the decoder, and K and V come from the encoder.

$$\mathbf{c}_t = \sum_{i=1}^{n} \alpha_{t,i} \boldsymbol{h}_i$$

$$\alpha_{t,i} = \text{align}(y_t, x_i)$$

$$= \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}$$

Dot-Product $\quad \text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$

https://lilianweng.github.io/posts/2018-06-24-attention/

Main components

- Encoder - Decoder
- Feed forward networks
- Embedding
- Positional encoding
- Attention
  - Self-attention
  - Multi-head attention
  - Masked Multi-Head Attention

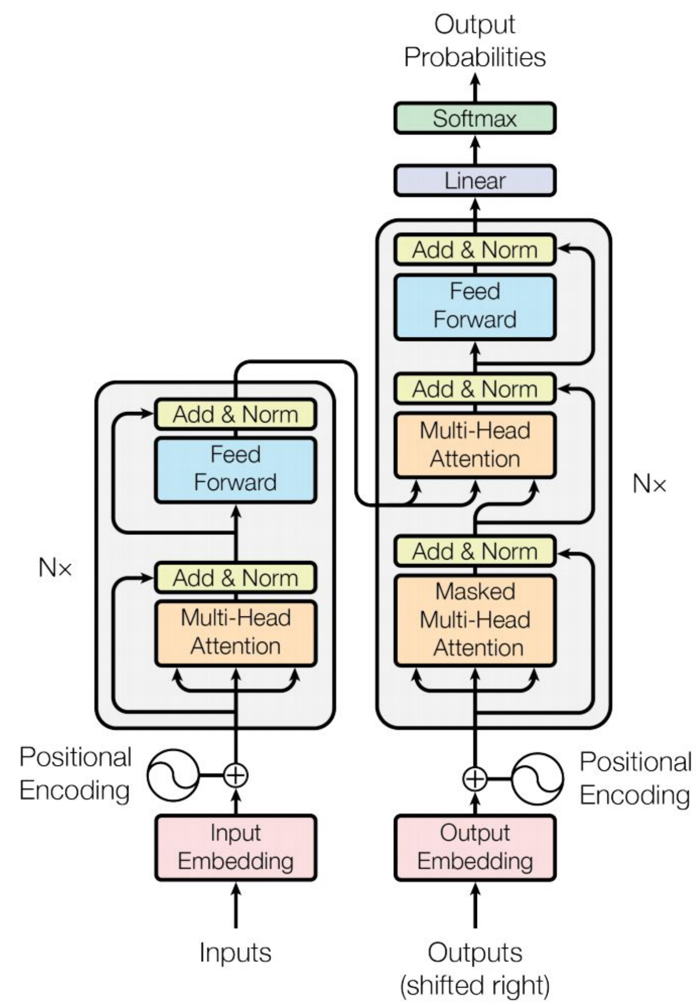


Figure 1: The Transformer - model architecture.

FLORIDA STATE UNIVERSITY

Attention vs Self-attention

Differences:

1. **Source of Q, K, V:**
   - **Attention**: Q, K, V can come from different sequences.
   - **Self-Attention**: Q, K, V are from the same sequence.
2. **Use-Cases**:
   - **Attention**: Often used in encoder-decoder architectures where the decoder attends to the encoder's output.
   - **Self-Attention**: Commonly used within either the encoder or the decoder to consider other positions in the same sequence for each position.
3. **Range of Application**:
   - **Attention**: Between different sequences (e.g., source and target in machine translation).
   - **Self-Attention**: Within the same sequence.
4. **Types of Tasks**:
   - **Attention**: Sequence-to-sequence tasks like machine translation, text summarization, etc.
   - **Self-Attention**: Tasks that require understanding the context in a single sequence, such as language modeling, text classification, etc.

Main components

- Encoder - Decoder
- Feed forward networks
- Embedding
- Positional encoding
- Attention
  - Self-attention
  - Multi-head attention
  - Masked Multi-Head Attention

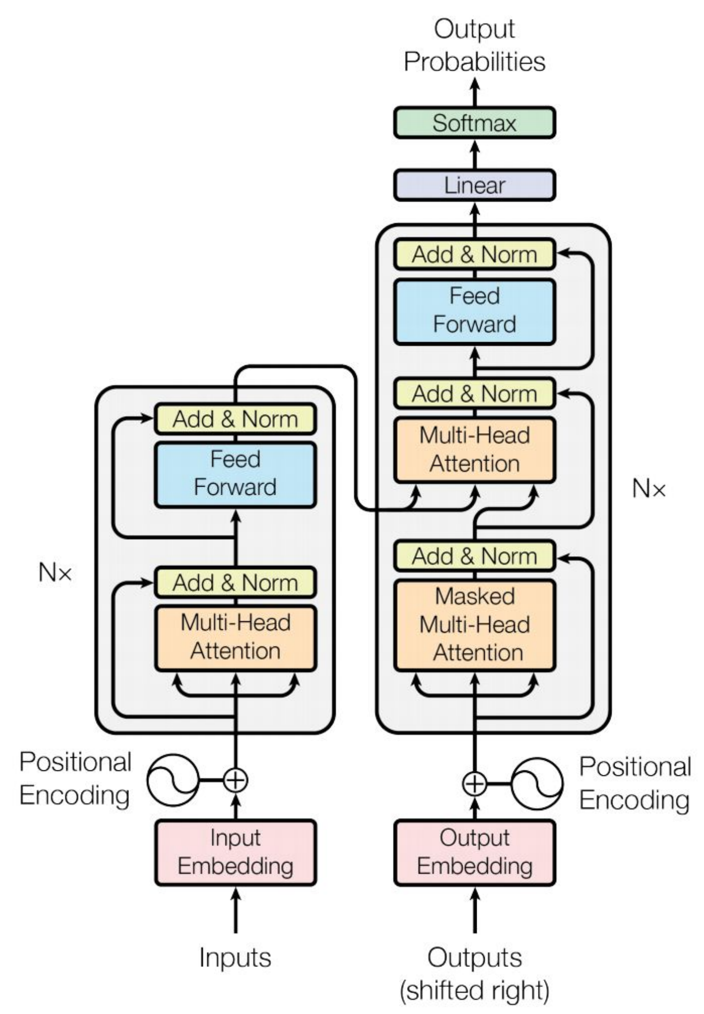


Figure 1: The Transformer - model architecture.

Multi-Head Attention

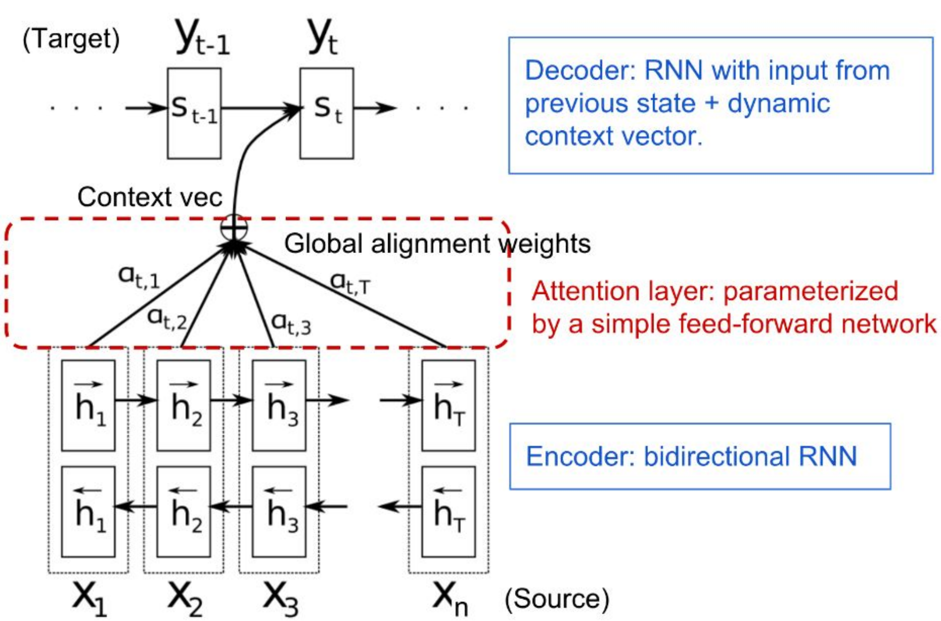
Multi-head attention is an extension of the standard attention mechanism that allows the model to focus on different parts of the input for different tasks or reasoning, all in parallel. It's a cornerstone of transformer models and has found applications in various domains like NLP, computer vision, and even scientific disciplines like your fields of oceanography and medical imaging when sequential or structured data are involved.

For ... are:

$= VW_h^V$

A ... $= H_h$

V^O



Main components

- Encoder - Decoder
- Feed forward networks
- Embedding
- Positional encoding
- Attention
  - Self-attention
  - Multi-head attention
  - Masked Multi-Head Attention

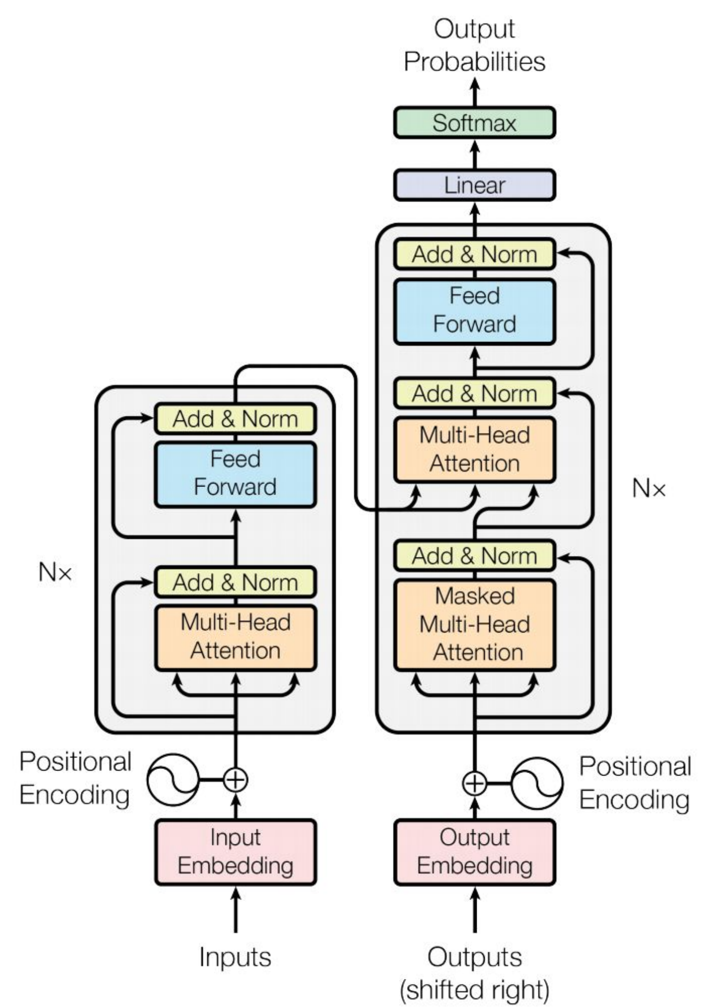


Figure 1: The Transformer - model architecture.

Didn't have more time … but

- Encoder - Decoder
- Feed forward networks
- Embedding
- Positional encoding
- Attention
    - Self-attention
    - Multi-head attention
    - Masked Multi-Head Attention
- Skip connections
- Softmax in the output

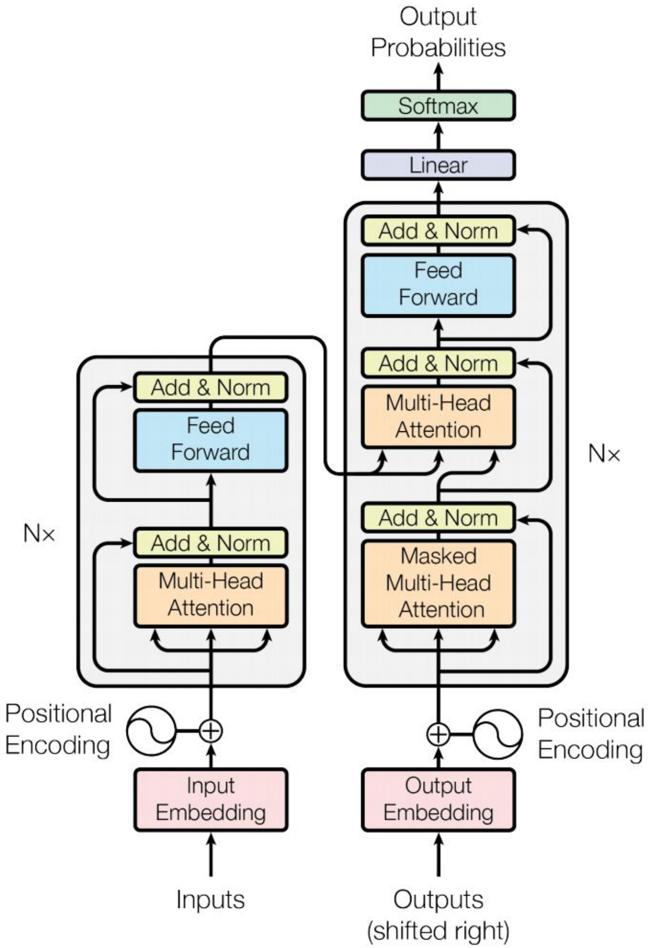


Figure 1: The Transformer - model architecture.

FLORIDA STATE UNIVERSITY

Summary

Innovations:

- Self-Attention
- Multi-Head Attention
- Positional Embeddings

Relevance:

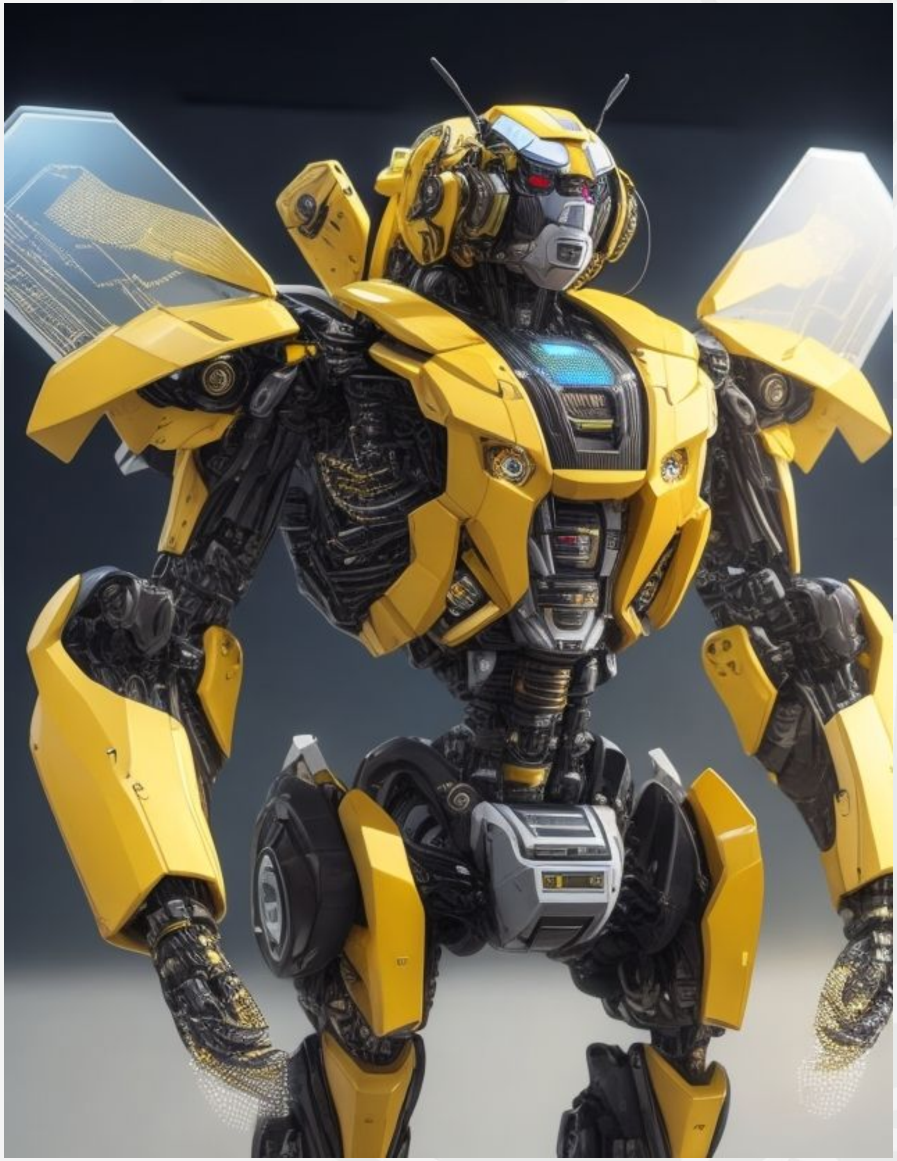
- NLP
- Computer Vision
- Etc.

Limitations:

- Computational cost
- Interpretability
- Lack of Recurrence
- Risk of Overfitting

FLORIDA STATE UNIVERSITY

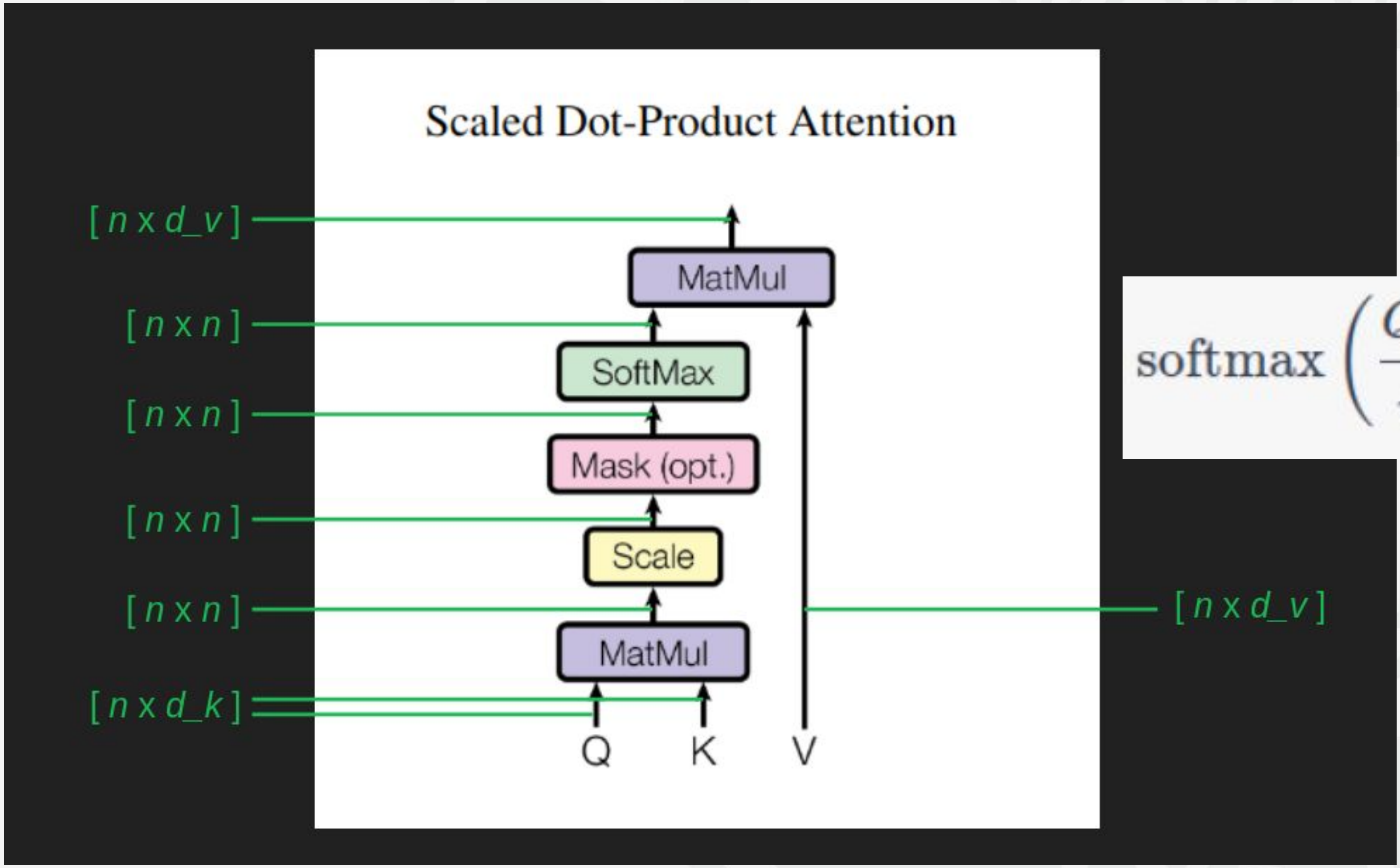
Questions?



By Leonardo ai

Scaled Dot-Product Attention

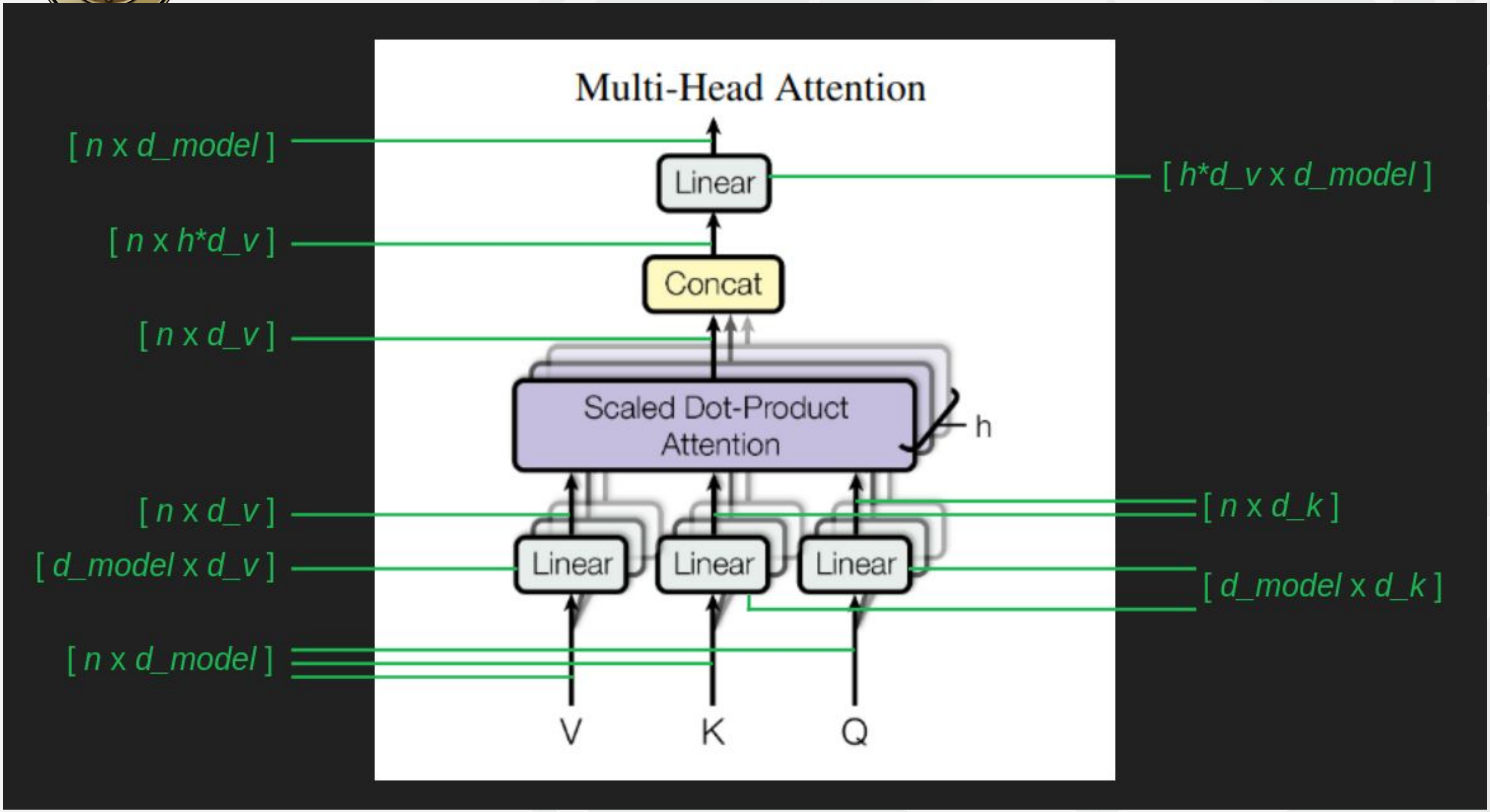
The Scaled Dot-Product Attention is a type of attention mechanism that calculates the importance of different parts in a sequence relative to each part. It's primarily used in Transformer models. The main components are Queries Q, Keys K, and Values V.



$$\mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \times V$$

FLORIDA STATE UNIVERSITY

Multi-head Self-attention



https://e2eml.school/transformers.html#multihead

FLORIDA STATE UNIVERSITY

Transformer Decoder