IEEE FCI - Helwan Student Branch

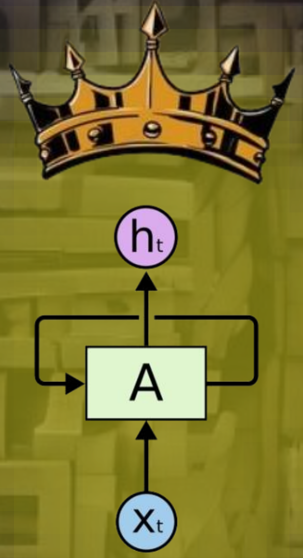
Transformers

Hossam Ahmed

Ziad Waleed

Mario Mamdouh

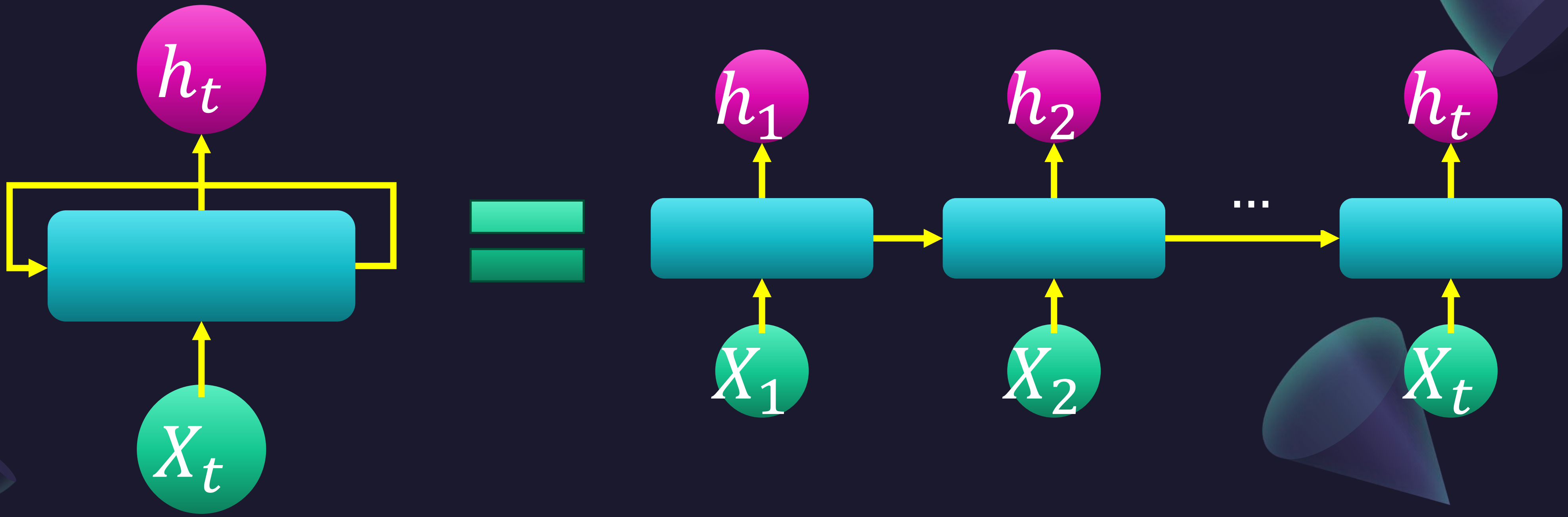
In the old days of language modeling, the throne was long held by Recurrent Neural Networks and convolution-based architectures



Everything changed in 2017 with the rise of transformers that leverages attention mechanism

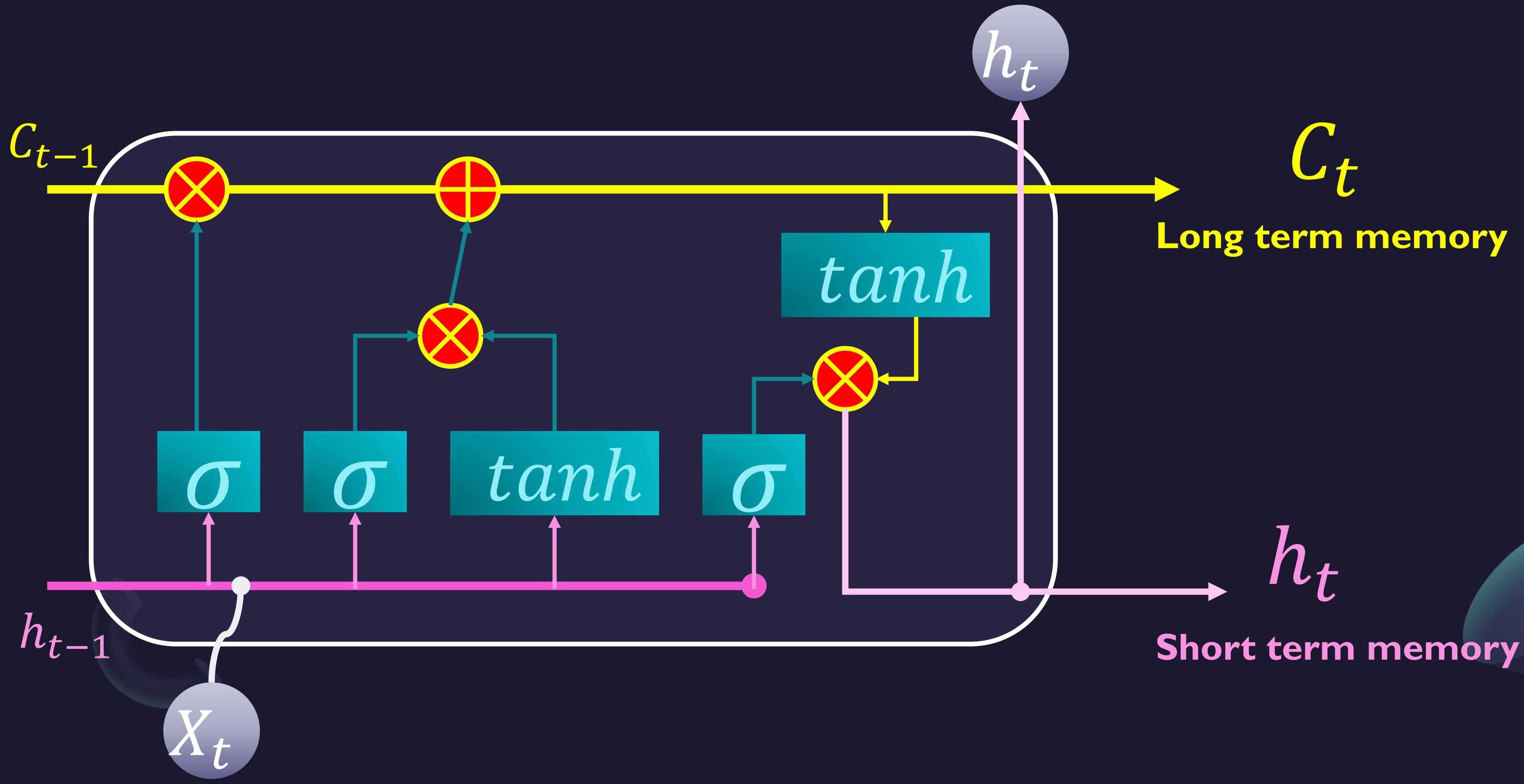
Introduction

- RNN architectures were the most used model for dealing with **sequential data**
- RNNs function similarly to a feed-forward neural network but process the input sequentially, **one element at a time**.



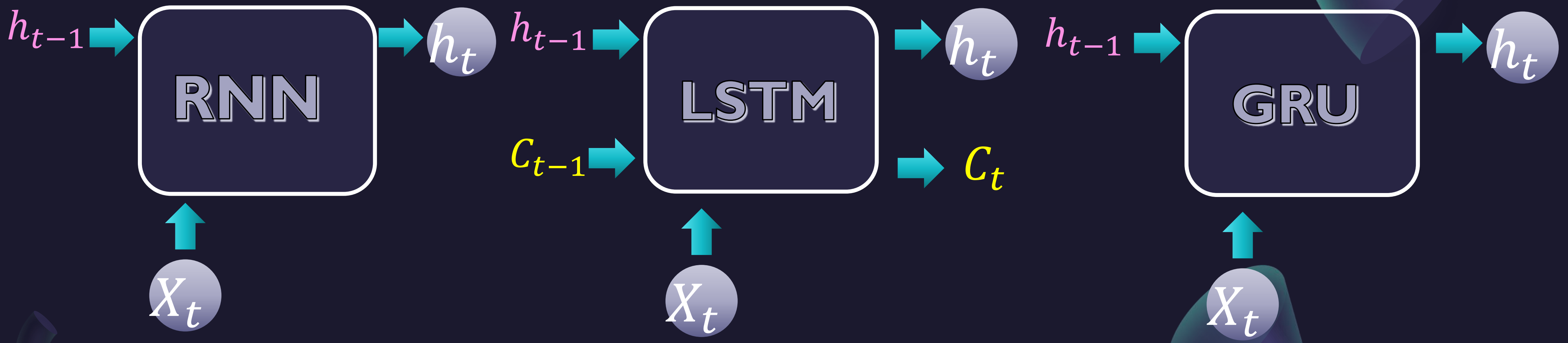
Introduction

- LSTM was introduced to solve the **vanishing gradients** problem to be able to train more deep RNN and to alleviate the loss of old information in the sequence.



Introduction

- GRU introduced in 2014, omitting context vector, resulting in a fewer parameters
- **Making us able to create deeper models with fewer parameters and faster training**



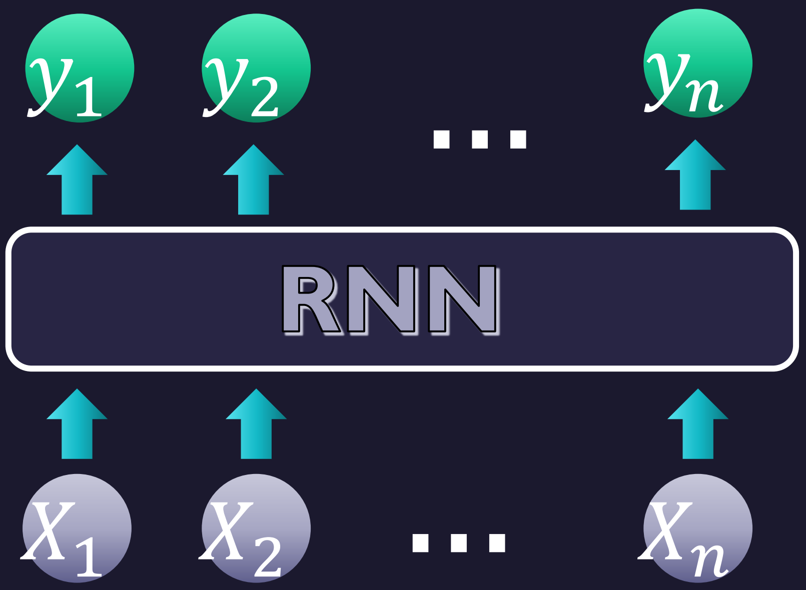
Introduction **problems with RNNs**

- **Hard to parallelize** as they process the data sequentially, one input after the other so doesn't make use of modern GPUs.

- **Difficulty with Long-Term Dependencies** this is due to the vanishing gradients problem that can cause loss of information when the chain of RNN units grows.

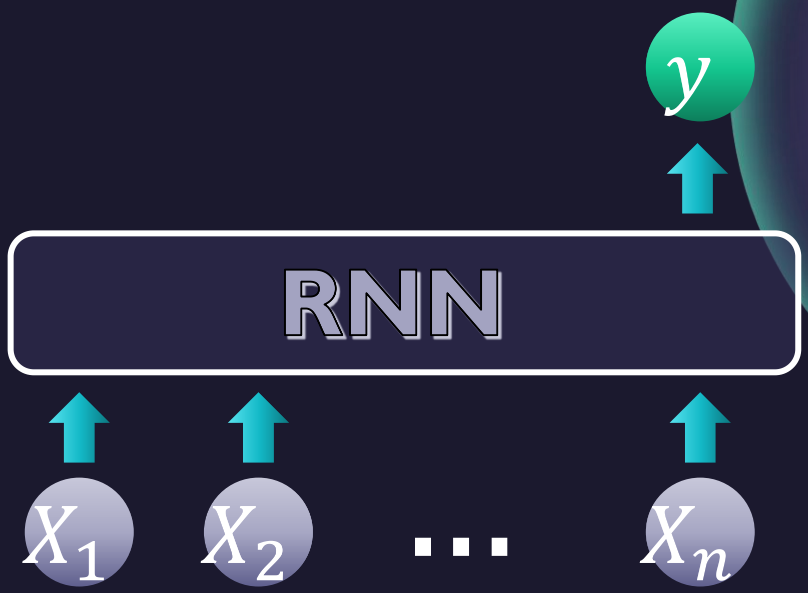
- **Limited Context Understanding** RNNs have a fixed-size context window determined by the length of the sequence they process.

Introduction **Common RNN NLP Architectures**



Sequence Labeling

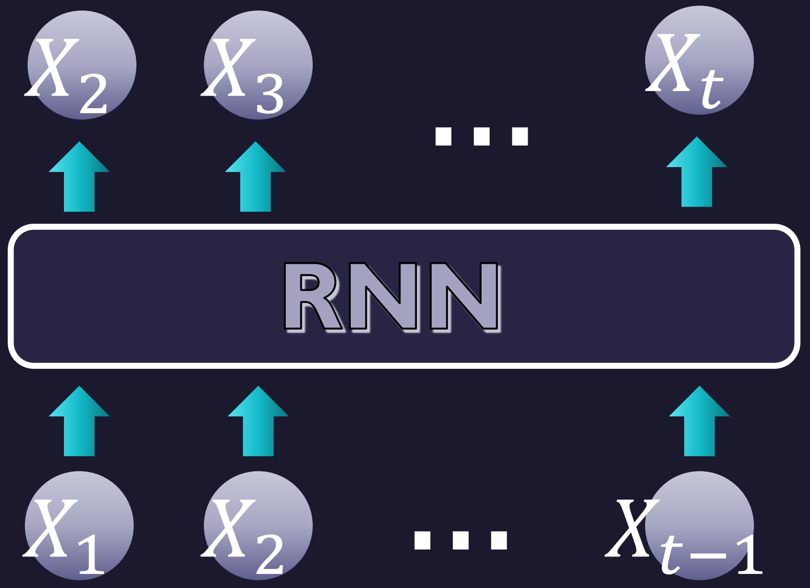
Named Entity Tagging



Sequence Classification

Sentiment analysis

Introduction **Common RNN NLP Architectures**



Language modeling

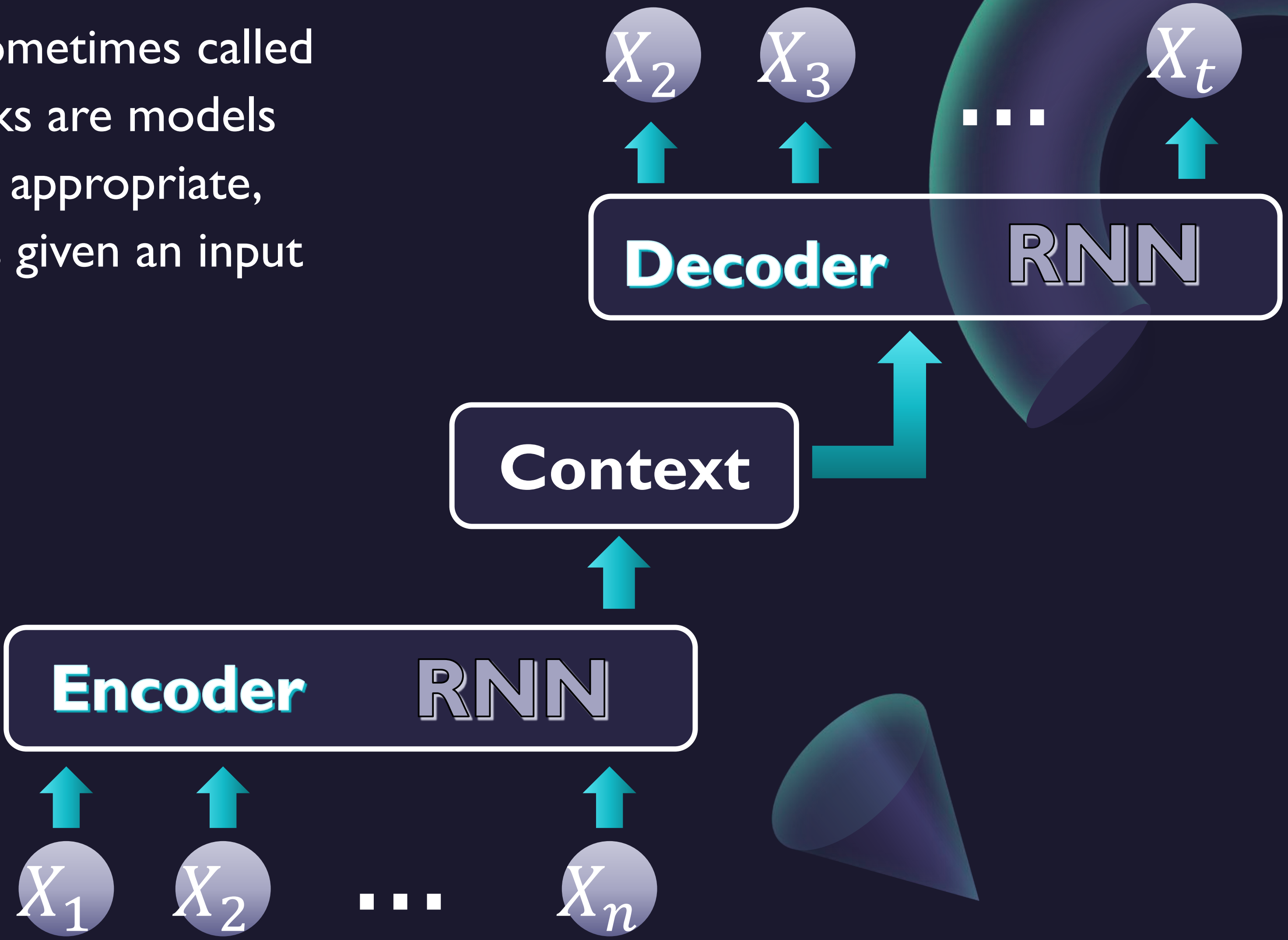
What is the next word...?

Introduction **Common RNN NLP Architectures**

- **Encoder-decoder networks**, sometimes called **sequence-to-sequence** networks are models capable of generating contextually appropriate, arbitrary length, output sequences given an input sequence.

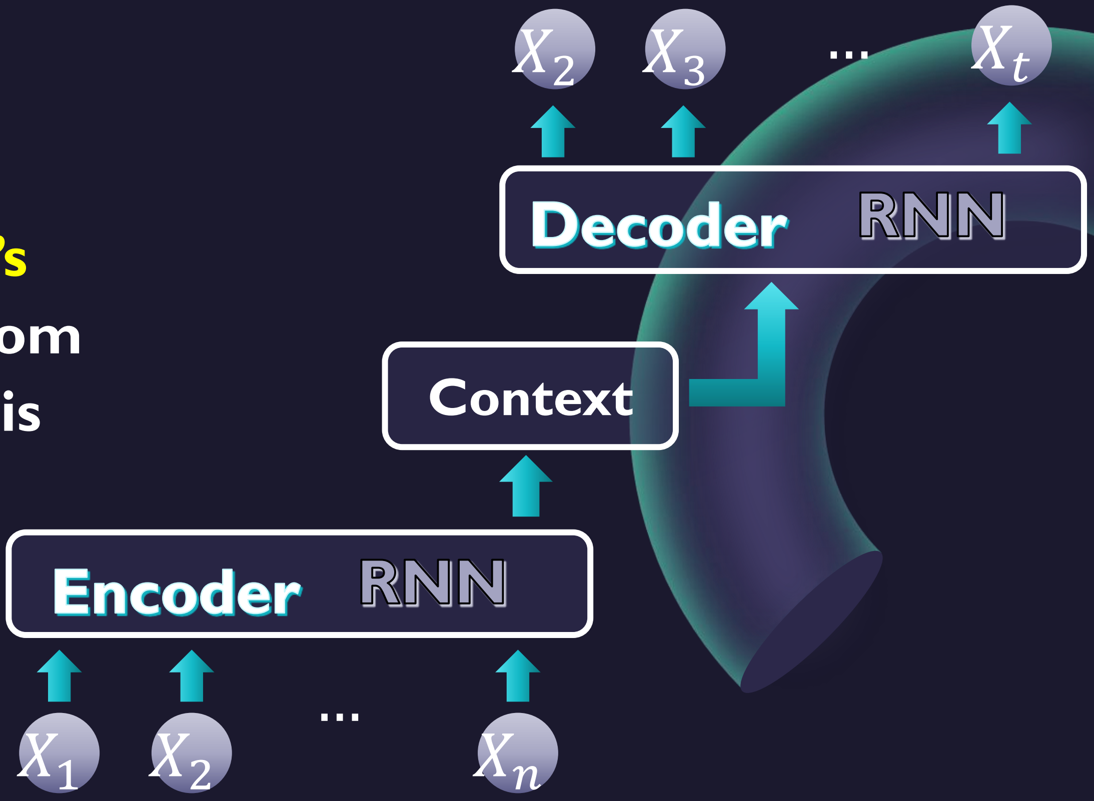
Encoder-Decoder

Translation



Introduction **context bottleneck**

- **Requiring the context to be the only encoder's final hidden state forces all the information from the entire source sentence to pass through this representation bottleneck.**



- **Bottleneck because**
  - it must represent absolutely everything about the meaning of the source text
  - since the **Decoder knows only the context vector** in this bottleneck

Attention is all you need **attention**

- **attention mechanism** is a solution to the bottleneck problem, a way of **allowing the decoder to get information from all the hidden states** of the **encoder**, not just the last hidden state.

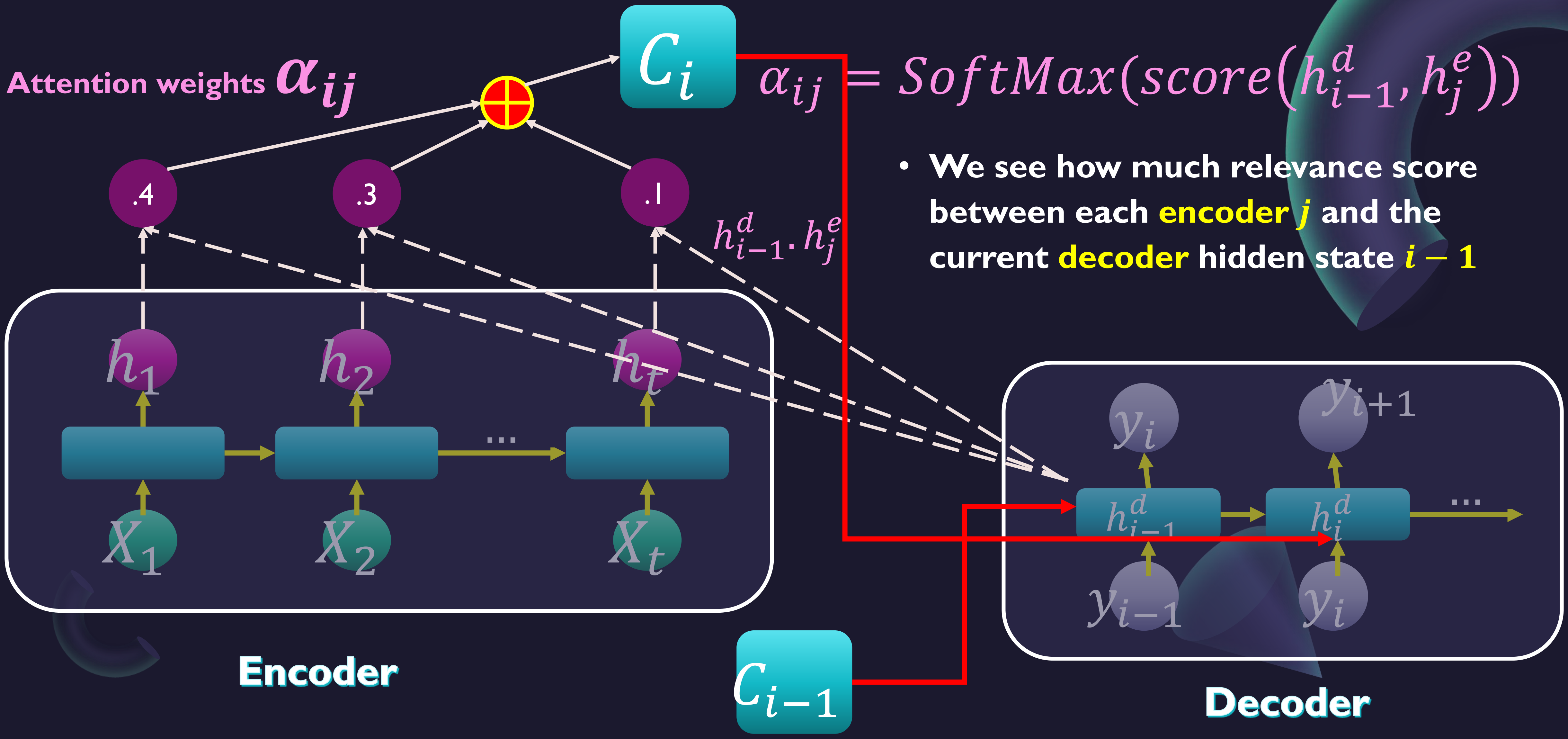
- The **idea of attention** is instead to **create the single fixed-length vector** c by taking a weighted sum of all the encoder hidden states.
  - The weights focus on ('attend to') a particular part of the source text that is relevant for the token the decoder is currently producing.
  - Attention thus **replaces the static context vector** with one that is **dynamically derived from the encoder hidden states, different for each token in decoding**

Attention is all you need **attention**

$$C_i = \sum \alpha_{ij} . h_j^e$$

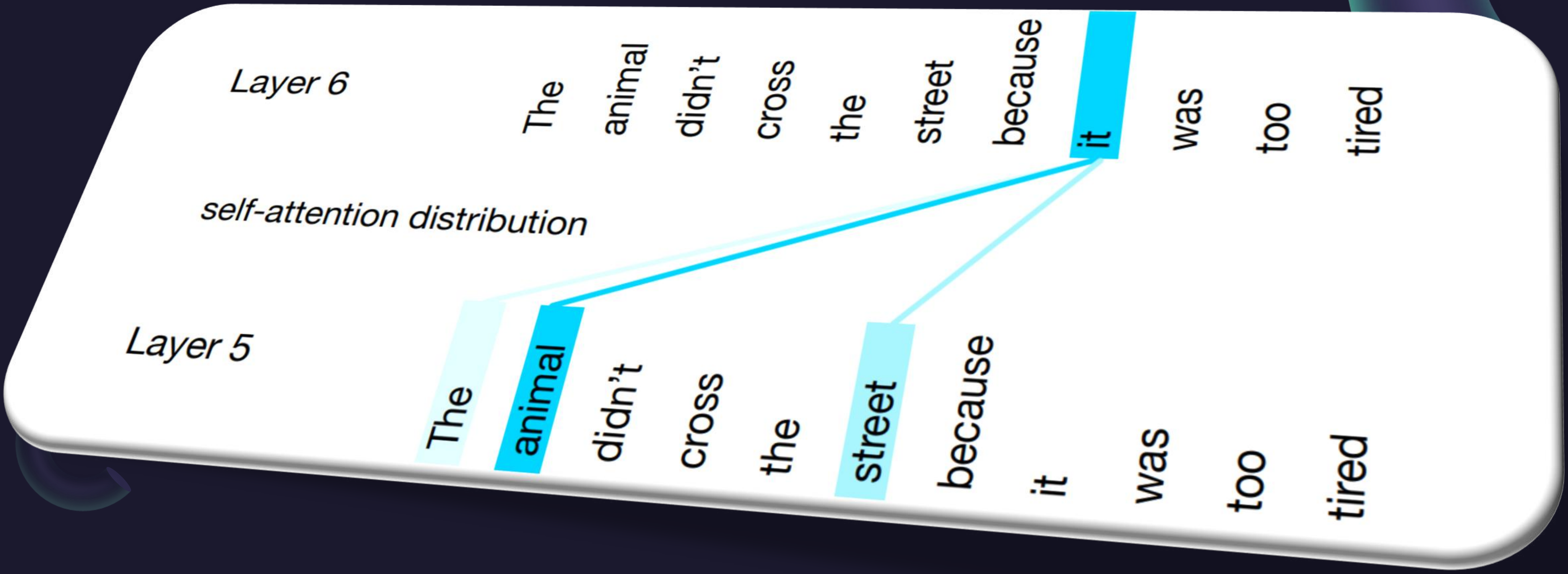
$$\alpha_{ij} = SoftMax(score(h_{i-1}^d, h_j^e))$$

- **We see how much relevance score between each encoder j and the current decoder hidden state $i-1$**



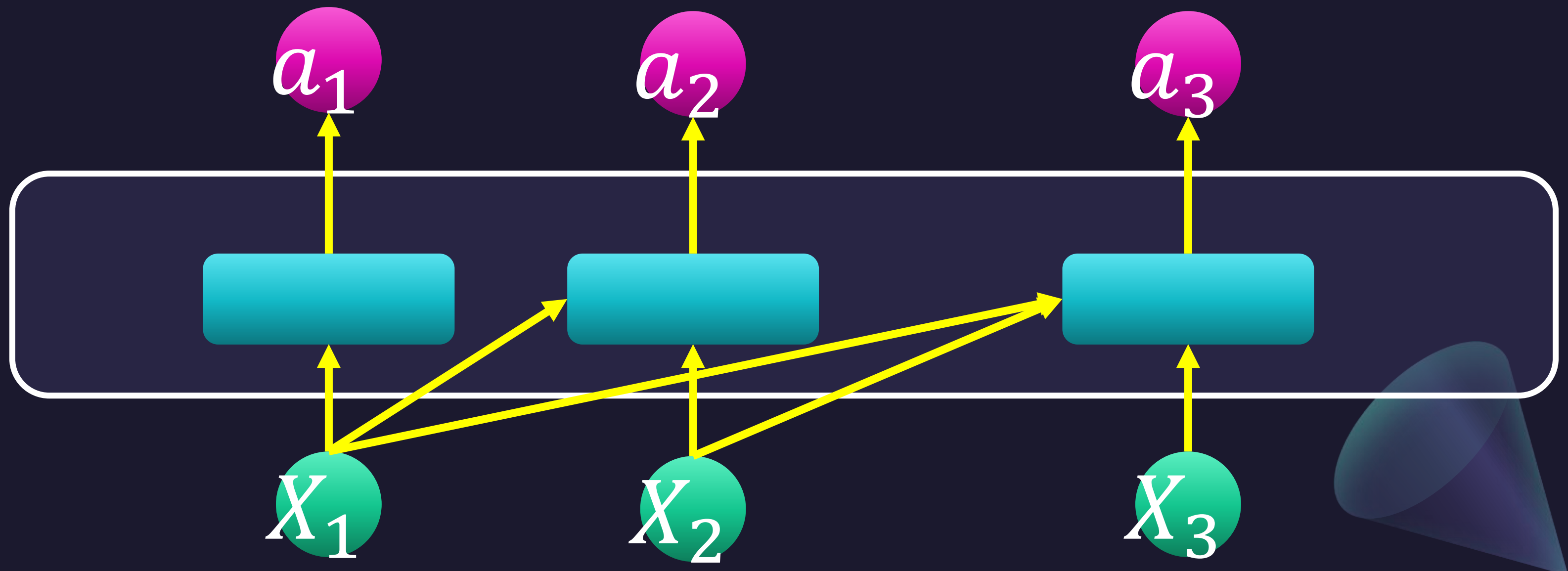
Attention is all you need **Self-Attention**

- **Self-Attention** can be thought of a way to **build contextual representations of a word's meaning** that **integrate information from surrounding words**, helping the model learn how words relate to each other over large spans of text.



Attention is all you need **Causal self-attention**

- Causal the model has **access to all of the inputs up to and including the one under consideration**
- In general **bidirectional self-attention**, the context can include future words
  - Bidirectional attention was used by BERT model.



Attention is all you need **Query, Key and Value**

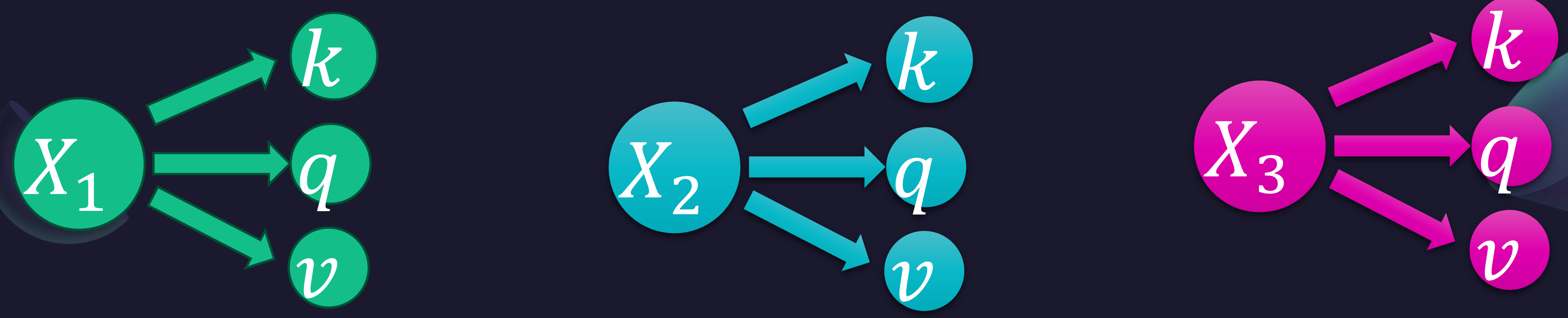
- Consider the three different roles that **each input embedding plays**
- Query
  - As the **current focus** of attention when **being compared to all** of the other **preceding** inputs
- Key
  - In its role as a **preceding input** being compared to the current focus of attention
- Value
  - used to **compute the output for** the current focus of attention.

Attention is all you need **Query, Key and Value**

$$SelfAttention(Q,K,V) = SoftMax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Calculate the value a_3 the third element in a sequence using causal self attention

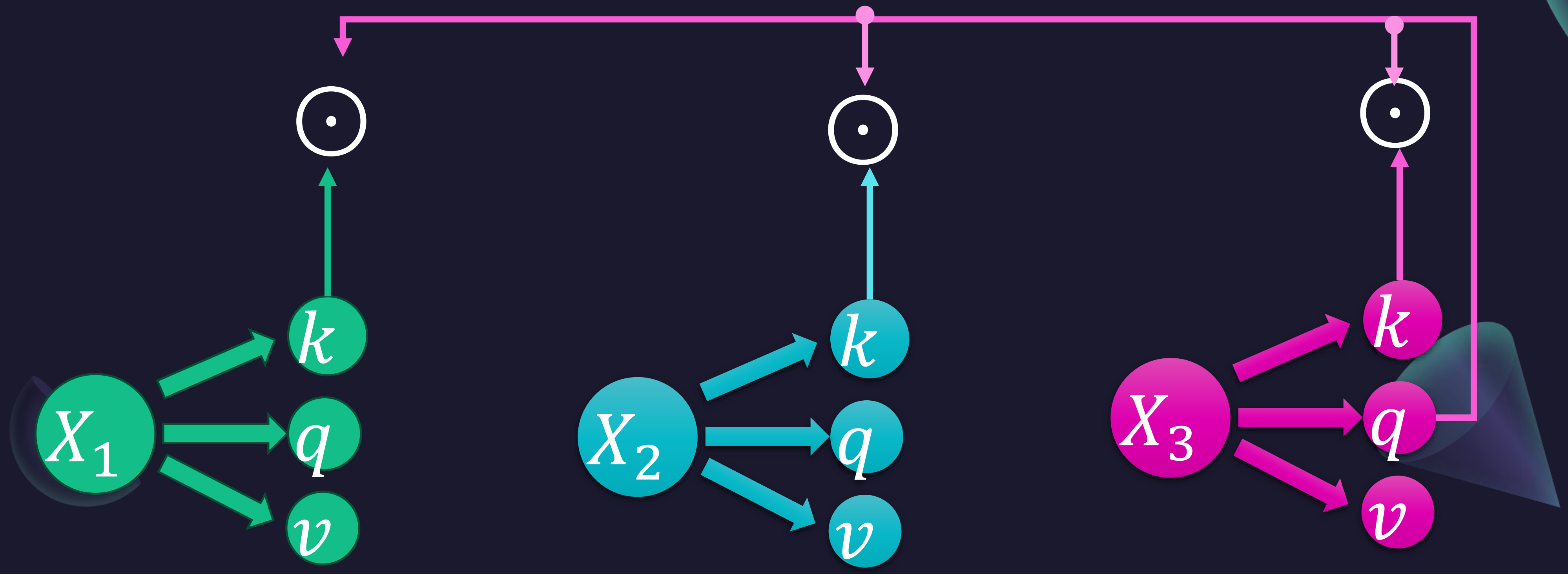
1. Generate key, query, value vectors



Attention is all you need **Query, Key and Value**

- Calculate the value a_3 the third element in a sequence using causal self attention

2. Compare X_3's **query** with all the other **keys**



Attention is all you need **Query, Key and Value**

- Calculate the value a_3 the third element in a sequence using causal self attention

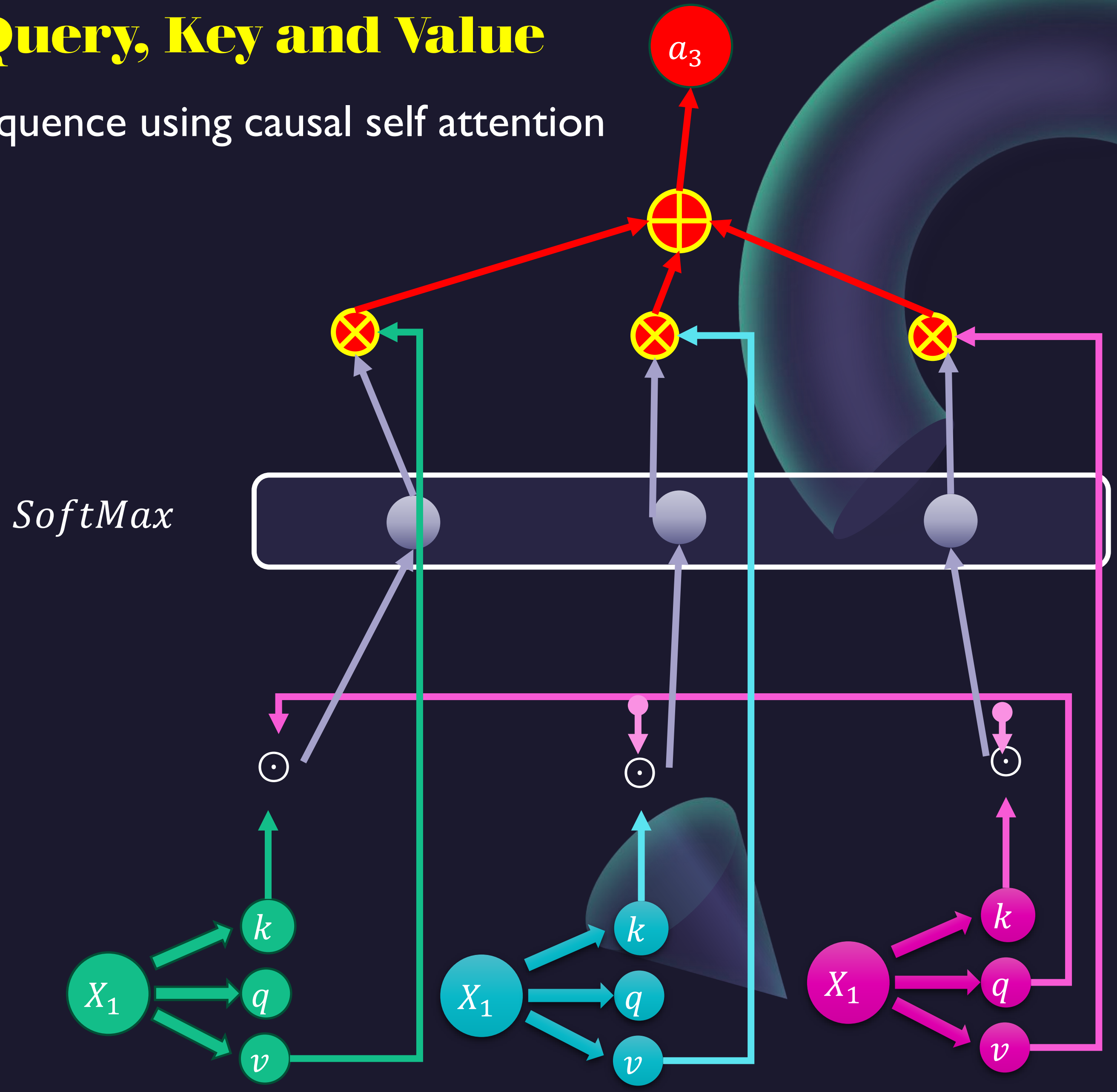
3.Divide the score by d_k

4.Apply softmax to turn it into weights

5.Weight each value

6. Sum the weighted value vectors

Output a_3



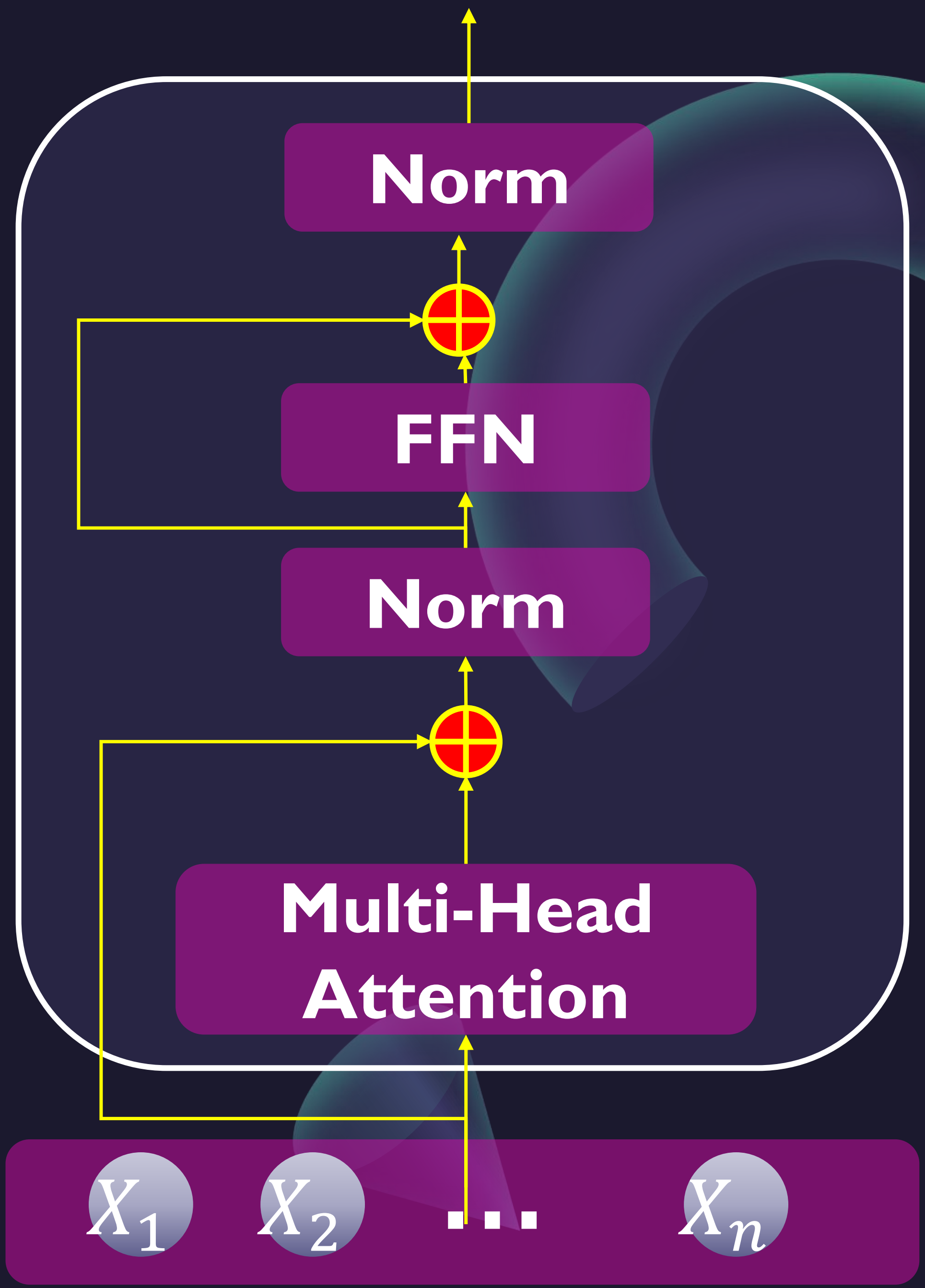
Transformers **Multi-head Attention**

- Transformers actually compute a more complex kind of attention than the single self-attention
- This is because the different words in a sentence can relate to each other in many different ways simultaneously
- It would be difficult for a single self-attention model to learn to capture all of the different kinds of parallel relations among its inputs
- **multihead self-attention** : sets of self-attention layers, called heads, that reside **in parallel layers at the same depth** in a model, each with its **own set of parameters**. By using these distinct sets of parameters, each head can **learn different aspects of the relationships** among inputs **at the same level of abstraction**.

Transformers **Blocks**

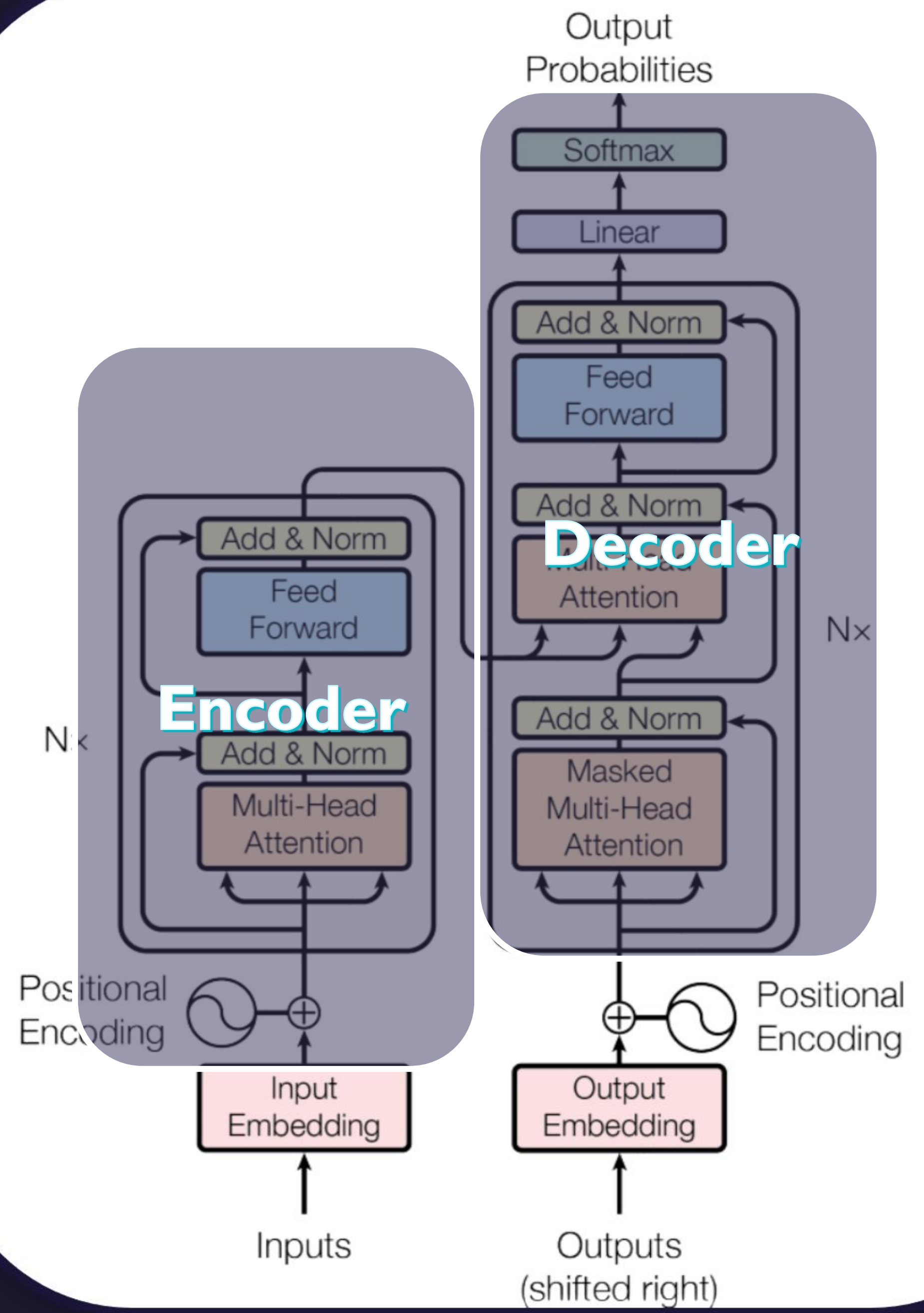
- includes three other kinds of layers:
  - a feedforward layer
  - residual connections
  - normalizing layers

- $O = \mathit{LayerNorm}(X \oplus \mathit{SelfAttention}(X))$
- $H = \mathit{LayerNorm}(O \oplus \mathit{FFN}(O))$



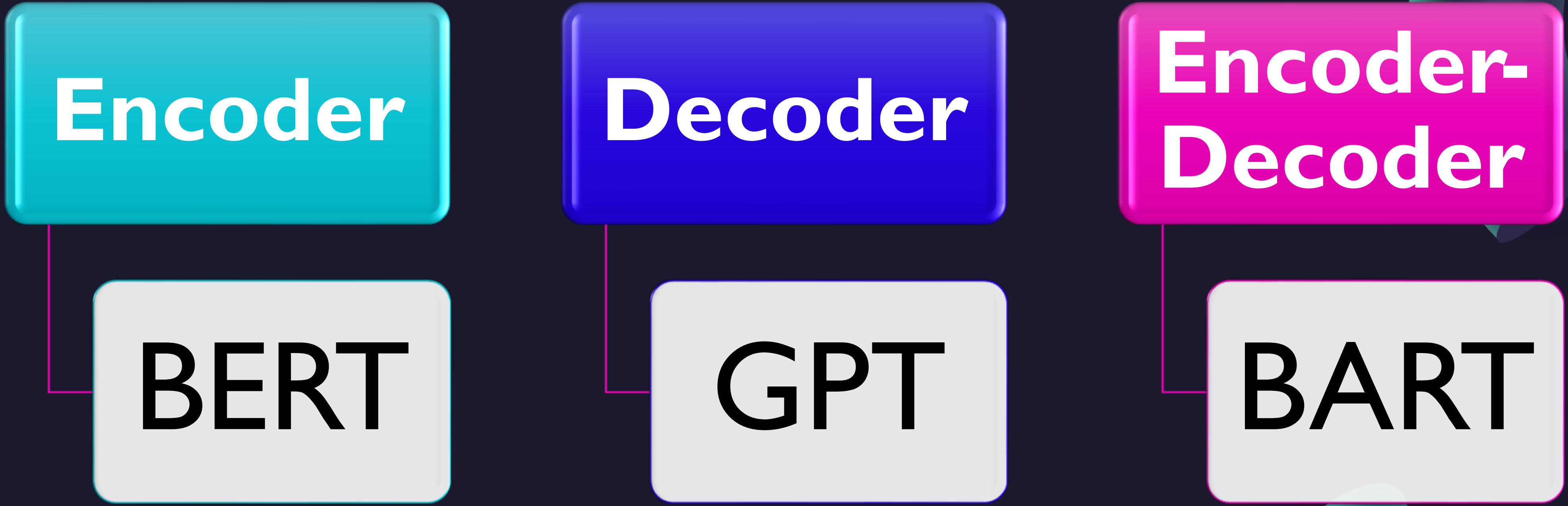
Transformers
in "Attention is all you need paper"

- Has two parts "Encoder-Decoder"
- The encoder is composed of a stack of N = 6 identical layers.
- The decoder is composed of a stack of N = 6 identical layers, plus a third sub-layer of multi-head attention that works over the output of the encoder stack



Transformers

Encoder/Decoder models

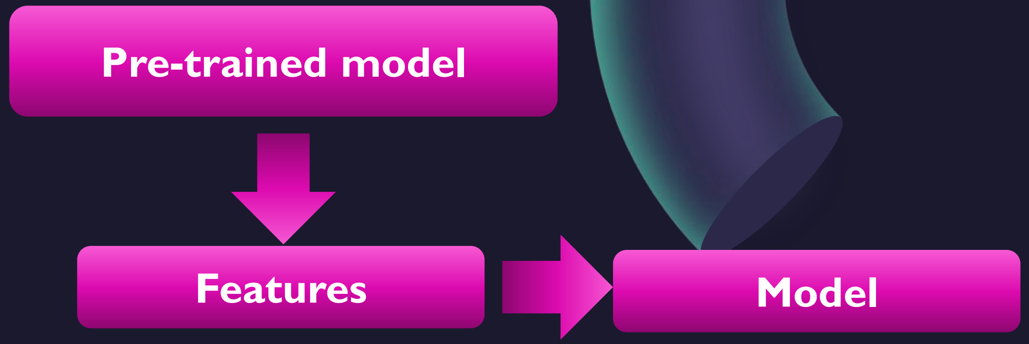


BERT

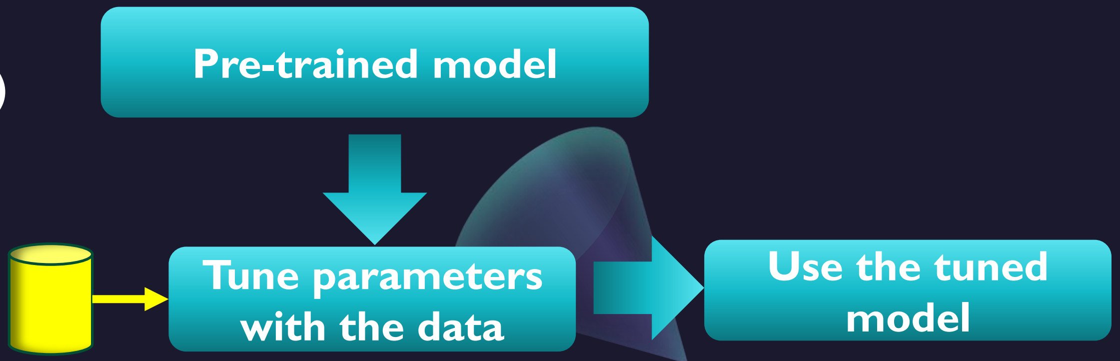
Bidirectional Encoder Representations from Transformers

- There are **two** existing strategies for **applying pre-trained** language representations to downstream tasks

- Feature-based
  - Embeddings from Language Models (ELMo)



- Fine-tuning
  - the Generative Pre-trained Transformer (OpenAI GPT)
  - introduces minimal task-specific parameters
  - trained on the downstream tasks by simply fine-tuning all pretrained parameters

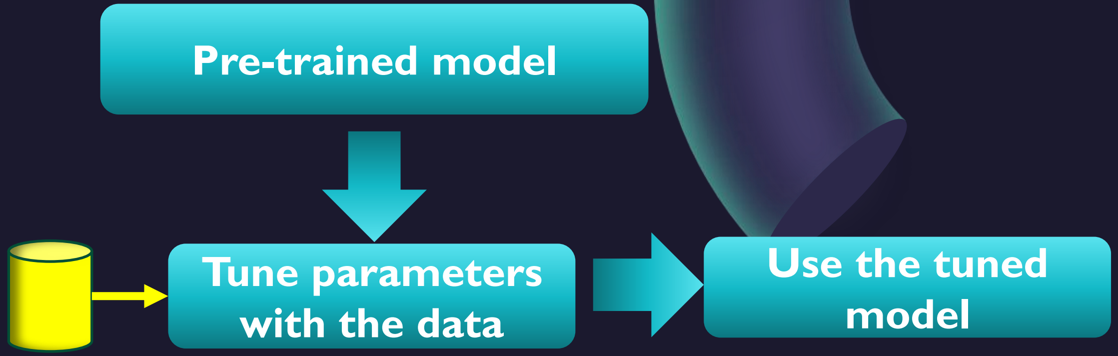


BERT

Bidirectional Encoder Representations from Transformers

- BERT paper claimed that "The major limitation is that standard language models are unidirectional"

- and this limits the choice of architectures that can be used during pre-training



- in OpenAI GPT
  - the authors use a left-to-right architecture
  - where every token can only attend to previous tokens in the self-attention layers of the Transformer

- Such restrictions are sub-optimal for sentence-level tasks

BERT

Bidirectional Encoder Representations from Transformers

- BERT alleviates the previously mentioned unidirectionality constraint by using
  - “masked language model” (MLM) pre-training objective
  - The masked language model randomly masks some of the tokens from the input
  - and the objective is to predict the original vocabulary id of the masked word based only on its context
  - Unlike left-toright language model pre-training
  - the MLM objective enables the representation to fuse the left and the right context
  - which allows to **pretrain a deep bidirectional Transformer**

BERT

Bidirectional **E**ncoder **R**epresentations from **T**ransformers

- BERT use also in the pretrain phase "next sentence prediction" task

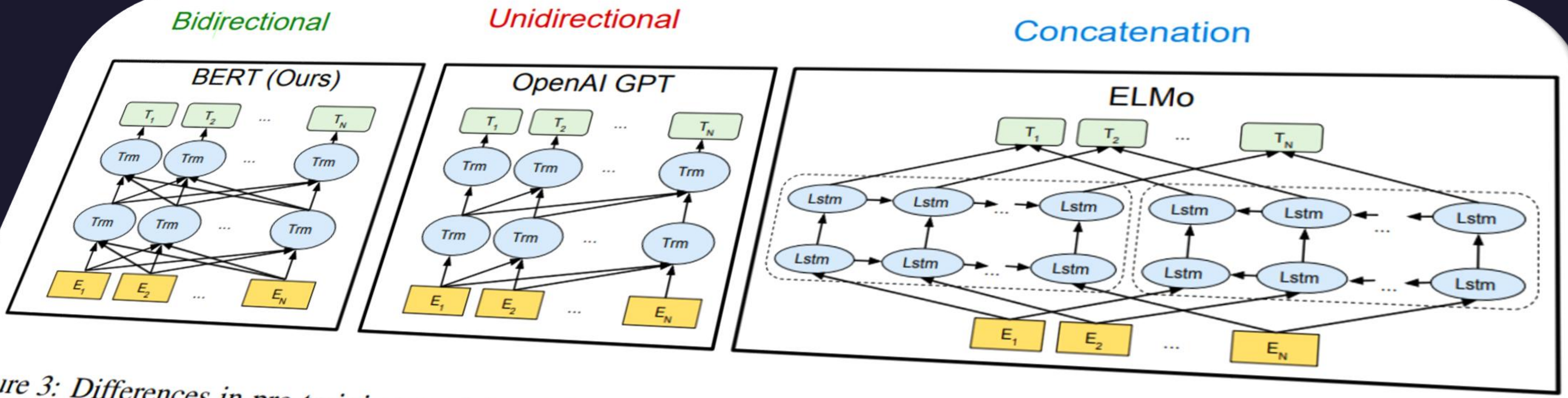


Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

BERT

Bidirectional Encoder Representations from Transformers

- A distinctive feature of BERT is its **unified architecture across different tasks**

- BERT Base
  - $L = 12, H = 768, A = 12$
  - **110 M total parameter**
- BERT Large
  - $L = 24,\ H = 1024,\ A = 16$
  - **340 M total parameter**

BERT

Input representation

- For a given token, its input representation is constructed by summing the corresponding **token**, **segment**, and **position embeddings**

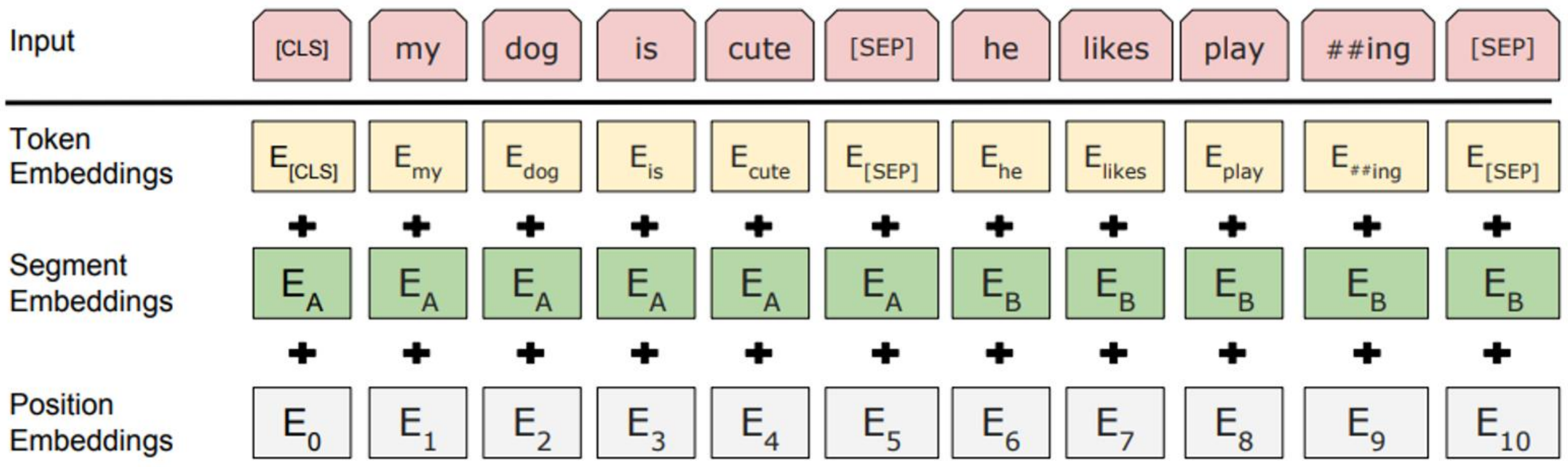


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

BERT

Feature-based Approach with BERT

- Not all tasks can be easily represented by a **Transformer encoder architecture**

- There are computational benefits to pre-compute an expensive representation of the training data once and then run many experiments with cheaper models on top of this representation
  - **Fine-tuning all the 110 million or 340 million parameter of a model for every task you want to test the model on is somehow expensive**
  - **In some cases, freezing the model and use it's understanding of language representation on other tasks by simply adding a simple network on top can be so useful and cheaper.**

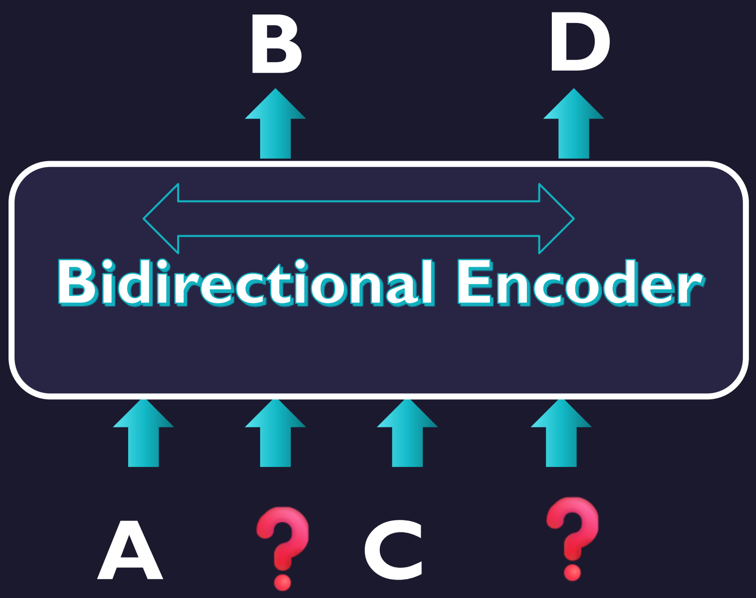
BART

Bidirectional and Auto-Regressive Transformers.

- Not all tasks can be easily represented by a **Transformer encoder architecture**
- a denoising autoencoder for pretraining sequence-to-sequence models.
- BART is trained by
  1. corrupting text with an arbitrary **noising** function
  2. learning a model to **reconstruct** the original text
- It use Encoder-Decoder Transformer based architecture
  - can be seen as generalizing to BERT and GPT
  - Bidirectional Encoder & Left-to-Right Decoder

BART

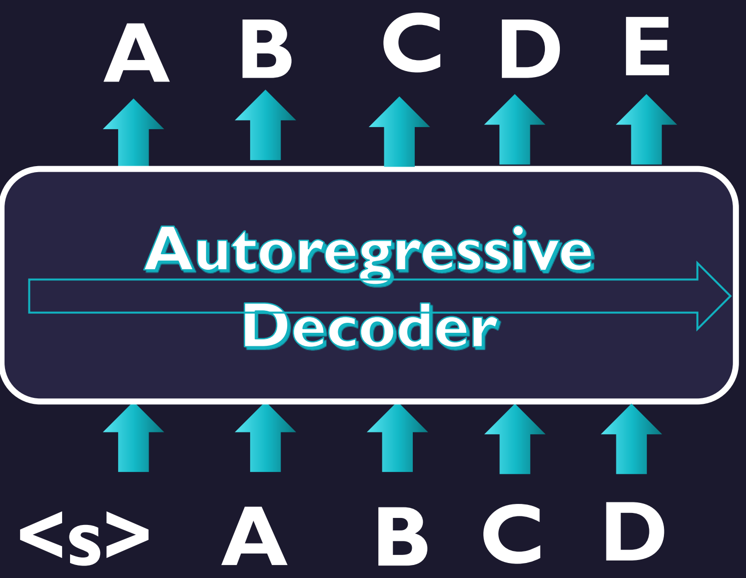
Bidirectional and Auto-Regressive Transformers.



- Random tokens are replaced with masks about 15%
- Missing tokens predicted independently
- BERT **can not easily** be used for generation

BART

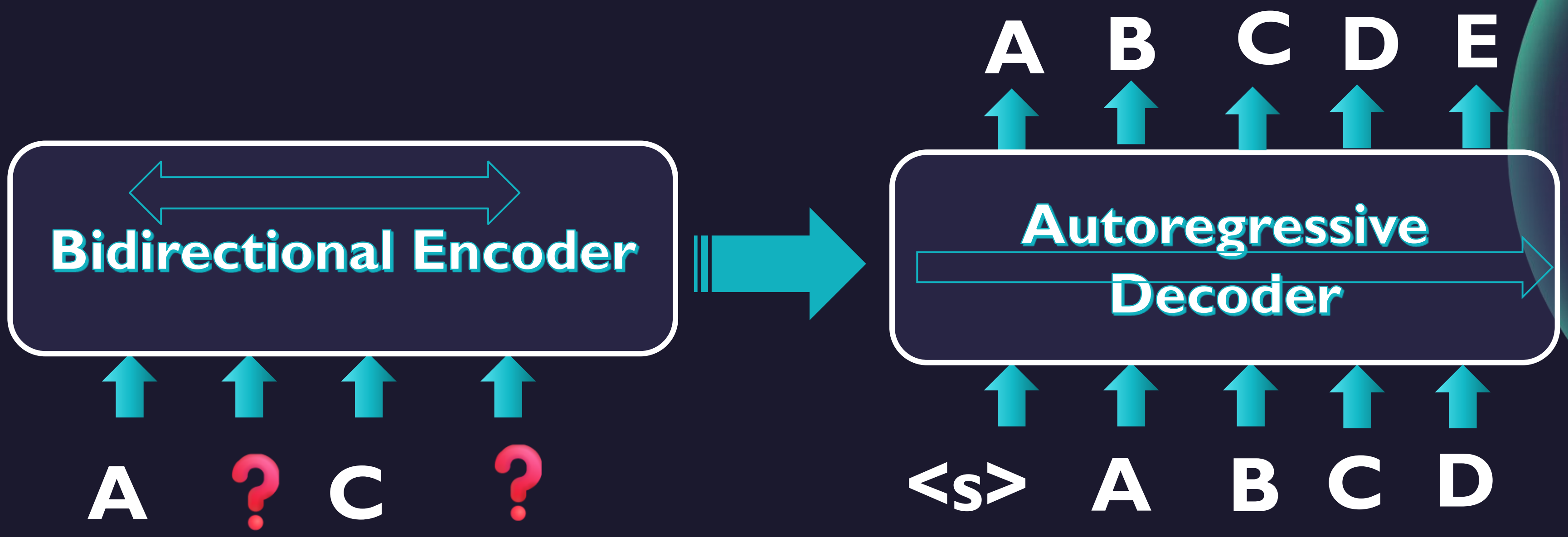
Bidirectional and Auto-Regressive Transformers.



- GPT tokens are predicted auto-regressively
- GPT can be used for generation
- But words can only condition on leftward context , and can't learn bidirectional interactions

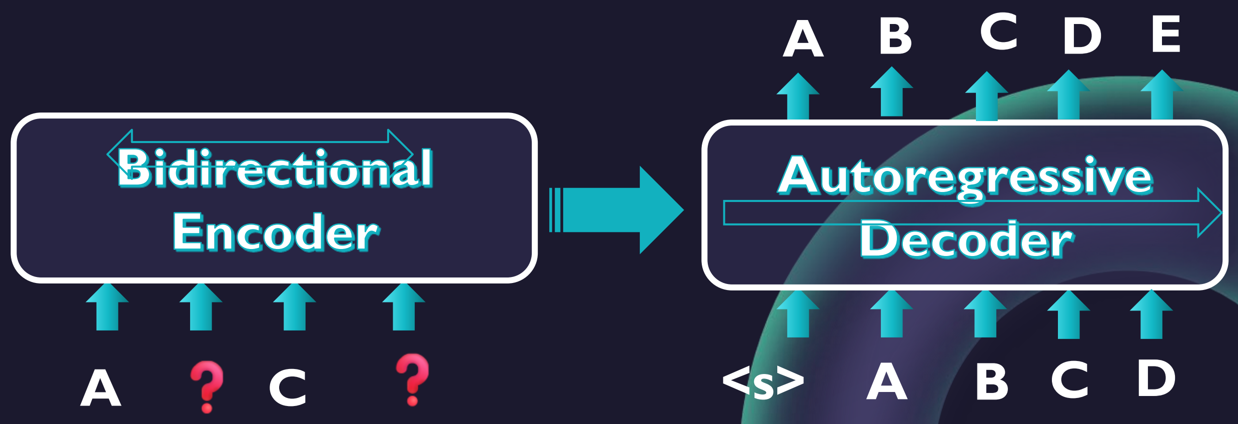
BART

Bidirectional and Auto-Regressive Transformers.



- BART , inputs to the encoder don't need to be aligned with the decoder , allowing for arbitrary noise functions
- BART is particularly effective when fine tuned for text generation but also works well for comprehension tasks.

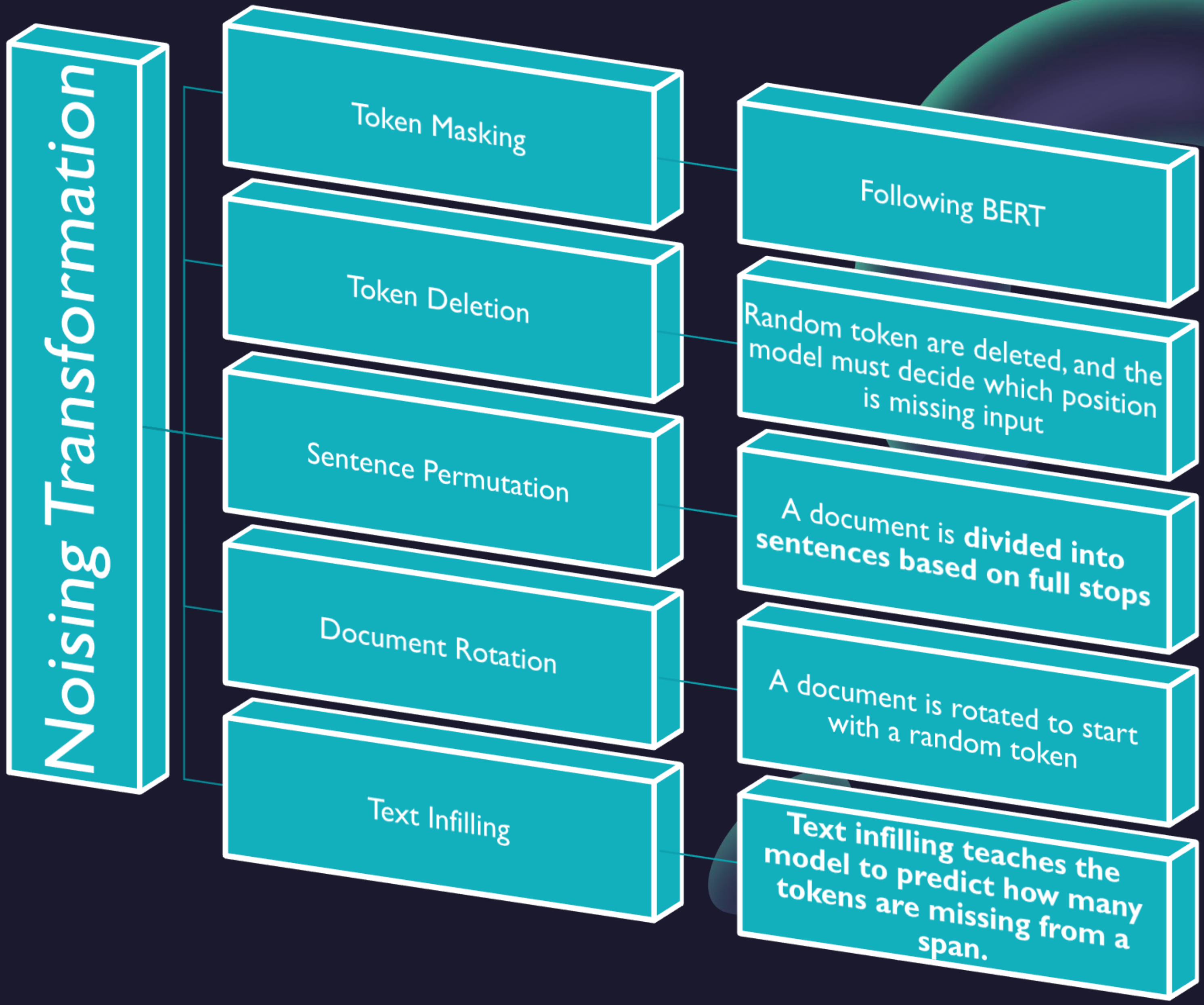
BART
Architecture



- BART uses the standard sequence-to-sequence Transformer architecture from "***attention is all you need***"
  - except, following GPT, that they modifed ReLU activation functions to GeLUs
  - initialized parameters from N (0, 0.02).
  - The architecture is closely related to that used in BERT with the following differences
    - BERT uses an additional feed-forward network before word prediction , which BART does not
    - Having a decoder that each layer in it preform cross attention over the final hidden encoder
    - BART contains roughly 10% more parameters than the equivalently sized BERT model.

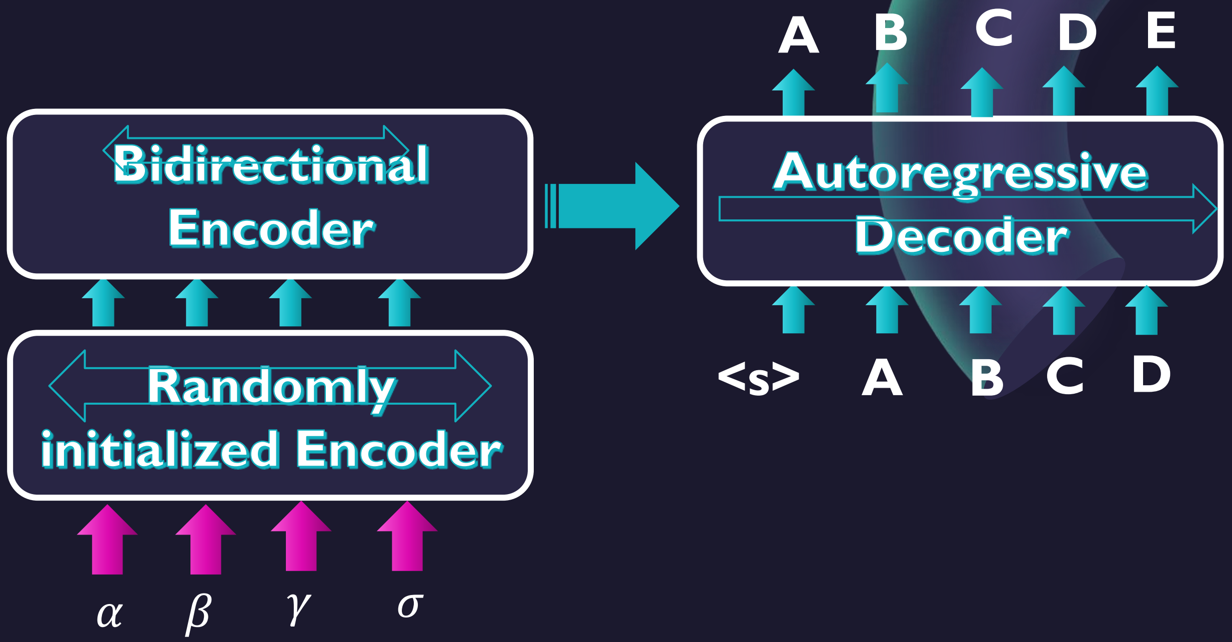
BART
Pre-training

- BART is trained by corrupting documents and then optimizing a reconstruction loss—the cross-entropy between the decoder's output and the original document



BART
In Machine Translation

- It is possible to use the entire BART model (both encoder and decoder) in Machine translation
  - by adding a new set of encoder
- The model is trained end-to-end
- which trains the new encoder to map foreign words into an input that BART can de-noise to English



See 👀

- https://poloclub.github.io/transformer-explainer/ [✨ visual article]
- https://bbycroft.net/llm [✨ visual article]
- https://jalammar.github.io/illustrated-transformer/
- https://github.com/jessevig/bertviz/tree/master
- https://colab.research.google.com/drive/1hXIQ77A4TYS4y3UthWF-Ci7V7vVUoxmQ?usp=sharing#scrollTo=T3H0qUZvPOP4
- https://youtu.be/LPZh9BOjkQs?si=PP4FIKDsoyw0MrOd
- https://youtu.be/wjZofJX0v4M?si=RNAHF-USY88pGb4o
- https://youtu.be/eMlx5fFNoYc?si=b5HhVEWYZZHofRSJ
- https://youtu.be/9-Jl0dxWQs8?si=AqMRg793N9GaviCl
- https://colab.research.google.com/drive/1JMLa53HDuA-i7ZBmqV7ZnA3c_fvtXnx-?usp=sharing#scrollTo=Q3k1Czf7LuA9
- https://www.kaggle.com/code/alejopaullier/introduction-to-transformers#Introduction-to-Transformers
- https://youtu.be/kCc8FmEb1nY?si=NSKmXDyaab-hH8xT
- https://youtu.be/C9QSpl5nmrY?si=Bu5zrbxzFs6aRpeG

Thank You