Lesson 11

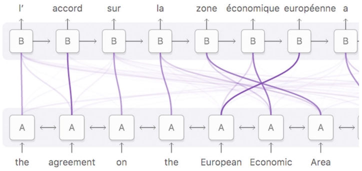
Transformers

The state of the art before Transformers

Tansformers have been introduced in Attention is All You Need, one of the most influential works of recent years.

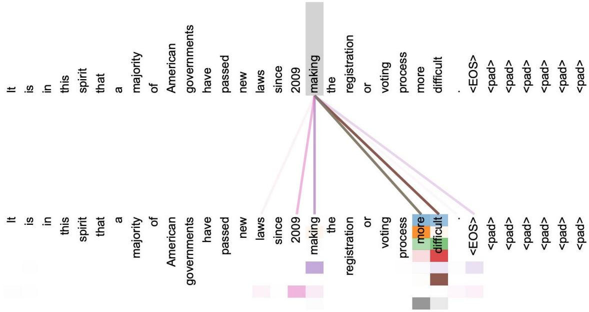
Recall the state of the art with RNNs.
The source sequence is fully visible, and we use attention to dynamically focus on relevant part.

We can do a similar operation for the target sequence. Instead of keeping a summary of the past target sequence in the local memory of a LSTM unit, let us give full access through attention.



Transformers

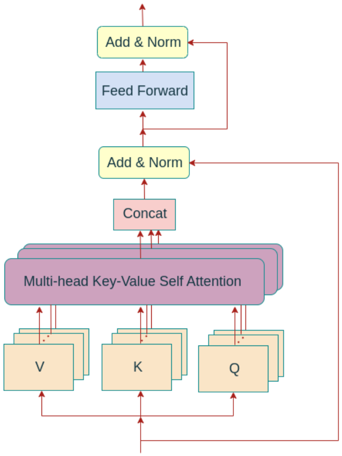
Transfomers are a feed-forward architecture meant to process sequences through a sequence of **transformers blocks**, exploiting at the maximum extent the attention mechanims.


It is in this spirit that a majority of American governments have passed new laws since 2009 making the registration or voting process more difficult . <EOS> <pad> <pad> <pad> <pad> <pad> <pad>
It is in this spirit that a majority of American governments have passed new laws since 2009 making the registration or voting process more difficult . <EOS> <pad> <pad> <pad> <pad> <pad> <pad>

A stack of Transformer blocks

A transformer is neither an encoder, nor a decoder: as the name suggests, it is a network transforming an input sequence into an output sequence, according to the training objectives.

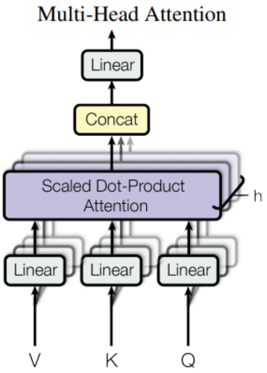
E.g. it can trained to act as an encoder, or a decoder.



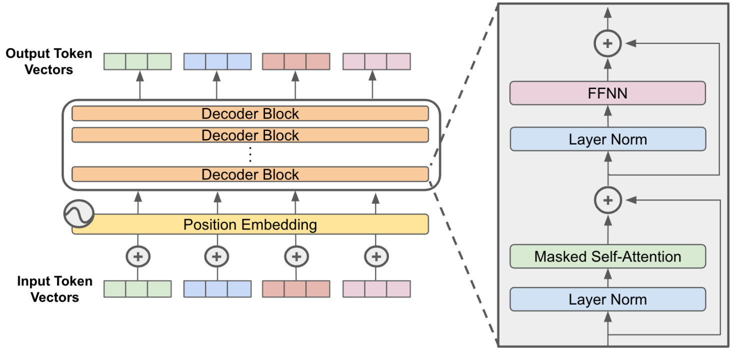
Multi head attention

Using multiple heads for attention expands the model's ability to focus on different positions, for different purposes.

As a result, multiple "representation subspaces" are created, focusing on potentially different aspects of the input sequence.



A typical architecture



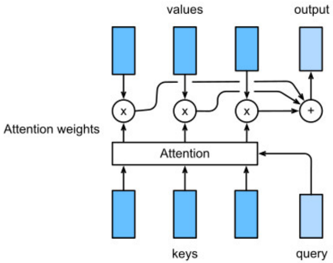
Positional encoding

Positional encoding is added to word embeddings to give the model some information about the **relative position** of the words in the sentence.

Why it matters?

The attention mechanims is agnostic to the position of key-value pairs: they are a set.

The order of return-tokens only depends on queries, so it reflects the input order.



This is a bit strange, since we are processing information with a strong sequential structure!

Positional encoding

The definition of the encoding is a bit technical, and not so important (see a common definition in the next slide).

The main property of the encoding is that for any fixed offset k, PE_{pos+k} can be represented as a **linear function** of PE_{pos}, allowing heads to focus on entities at different offsets.

Sinusoid encoding

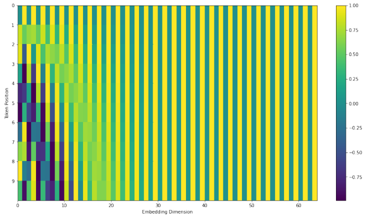
The positional information is a vector of the same dimensions d_{model}, of the word embedding.

The authors use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i+1/d_{model}})$$

According to the previous encoding, each dimension i of the PE vector corresponds to a sinusoid, where the wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$.



The Learning task

For language modeling, the learning task consists in predicting the next token in a sequence.

Example:

Input: The - cat - sat - on - the
Target: cat - sat - on - the - mat

Both sequences have the same length and the model processes the entire input sequence in parallel

A Problem: Transformers See Everything

- In self-attention, each token attends to all others, including future ones
- But to predict token $t+1$, the model must not see it.
- We must enforce **causality**

Masking the future

In **Masked self-attention** we manipulate attention scores to exclude future tokens within the sequence.

Technically if works in the following way. Once the query and key matrices have been multiplied, we have an attention matrix of size [T, T] with each token's attention scores across the full sequence. Prior to performing the softmax operation across each row of this matrix, we **set all values above the diagonal of the attention matrix to negative infinity**.

By doing this, we ensure that, for each token, all tokens that follow this token in the sequence are given an attention score of zero after the softmax operation has been applied.

Next Argument

Conditioning

Conditional Generation

LLMs are next-token predictors by default
But we often want:

- Translate: English $\rightarrow$ French
- Summarize a paragraph
- Answer a question
- ...

LLMs do not do these operations natively.
They just generate text - but we can condition them to behave differently depending on what we input.

Different ways for conditioning generation

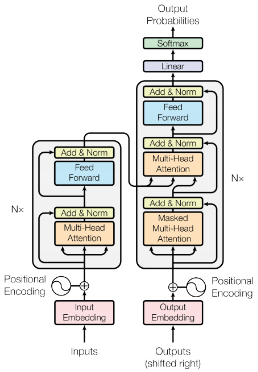
Two main approaches: **Prompting** vs **Cross-Attention**.

- **Prompting** treats the condition as more tokens.
- **Cross-attention** treats it as a separate representation.

Feature	Prompting	Cross-Attention
Input form	Concatenated text	Separate encoder representation
Architecture	Decoder only	Encoder-Decoder
Used in	GPT, LLaMA, ChatGPT	T5, BART, mT5, Flamingo
Architectural?	No	Yes

Encoder-decoder (T5, BART,. . .)

- The encoder transforms the input into a latent representation before the decoder even starts generating.
- The decoder attends to these representations via cross-attention.
- The training objective is typically sequence-to-sequence (e.g., translating a sentence, summarizing a document).

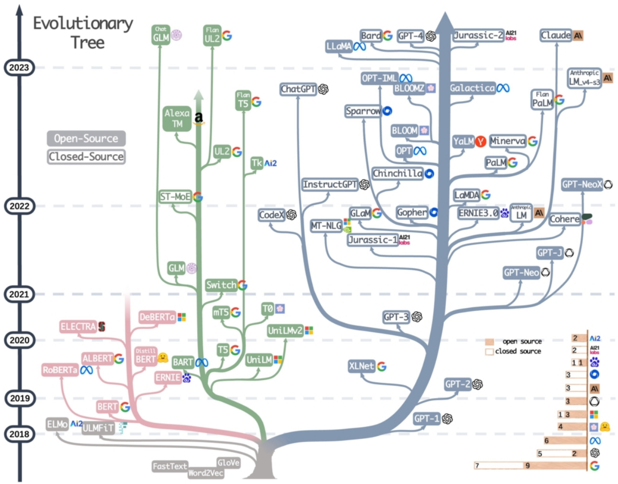


Encoder-decoder transformer

Decoder-only (GPT, LLaMA, Falcon, . . .)

- Instead of a separate encoding step, all embeddings (tokens, conditioning inputs, etc.) are directly inserted into the same sequence.
- The model must learn to interpret conditioning signals entirely within self-attention.
- The training objective is causal language modeling - predicting the next token autoregressively.

LLMs family tree



Next Argument

Fine-Tuning vs

Lightweight Adapters

Lightweight Adapters: efficient and modular fine-tuning

Instead of fine-tuning the whole model, we inject small, trainable components while freezing the rest.

Popular Adapter Techniques:

Method	**Description**
Adapters	Add small neural modules between layers
LoRA	Add low-rank matrices to key projections
QLoRA	Quantized LoRA for low-resource fine-tuning

Lightweight Adapters (e.g., LoRA)

Only tiny trainable modules are inserted in select sub-layers.

A×B decomposition: express a matrix with dimension $N \times N$ as the product of two matrices of dimension $N \times m$ and $m \times N$, with m small.

Where LoRA is applied: In attention projections (e.g., Q, V matrices)

So inside the attention layer:

$$Q = (\underset{\substack{\uparrow \\ \llcorner\text{ Base model (frozen)}}}{W_q} + \underset{\substack{\hookleftarrow \text{ LoRA A}\times\text{ B (learnable)}}}{\Delta W_q}) \cdot x$$

Other layers remain frozen, so we save memory, preserve general knowledge, and only learn task-specific deltas.

Benefits of Lightweight Adapters

- Parameter-efficient: Train $< 1\%$ of total model
- Faster training: Works on consumer GPUs
- Modular: One base model, many adapters
- Open source friendly: you can freely share the adapter's weights

Adapters make LLMs customizable and efficient. You can fine-tune a 7B model on a laptop, and keep the base model intact for other tasks.

Example Use Cases

- Alpaca-LoRA (fine-tuned LLaMA using LoRA)
- Personal assistants (train a private adapter on your data)
- Multitask inference (swap adapters at runtime)

Comparison Summary

In today's ecosystem, lightweight adapters are becoming the default for fine-tuning, especially in open-source and multi-task systems.

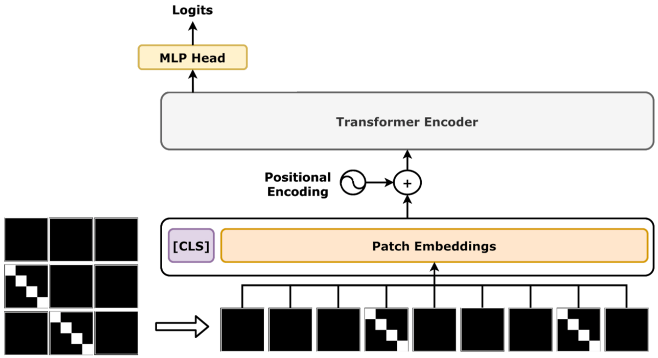
Feature	**Full Fine-Tuning**	**Lightweight Adapters**
Parameters updated	All (100%)	Small subset (0.1%)
Computational cost	High	Low
Memory footprint	High per task	One base + small deltas
Modularity	No	Yes
Deployment flexibility	single-purpose models	Plug-and-play
Use case	Deep domain adaptation	Quick task adaptation, sharing

Next argument

Visual Transformers (ViT)

Vision Transformer

The input image is decomposed into a series of patches (rather than text into tokens). Each patch is embedded and serialized into a sequence, that is theen processed by a traditional transformer.


Logits
MLP Head
Transformer Encoder
Positional Encoding
+
[CLS]
Patch Embeddings