Deep Learning and Neural Networks

Demetrio Labate

April 14, 2025

Part 7
Neural Language Processing (NLP)

Why transformers?

The **transformer** neural network is a deep learning architecture that aims to solve sequence-to-sequence tasks while handling long-range dependencies with ease.

- It was first proposed in the paper "Attention Is All You Need" (2017) by researchers at Google.
- Transformers have the advantage of having no recurrent units, therefore requiring less training time than RNNs.
- State-of-the-art technique in the field of NLP.
- Originally introduced for language translation, they are currently used in
  - computer vision (Vision Transformers, 2019-20)
  - audio (Robust Speech Recognition)
  - reinforcement learning
  - robotics
  - playing chess
  - they also led to the development of pre-trained systems, such as generative pre-trained transformers (GPTs) and BERT (bidirectional encoder representations from transformers).

Why transformers?

One main difference with respect to RNNs is that the input sequence can be **passed in parallel** so that GPU can be used effectively and the speed of training can also be increased.

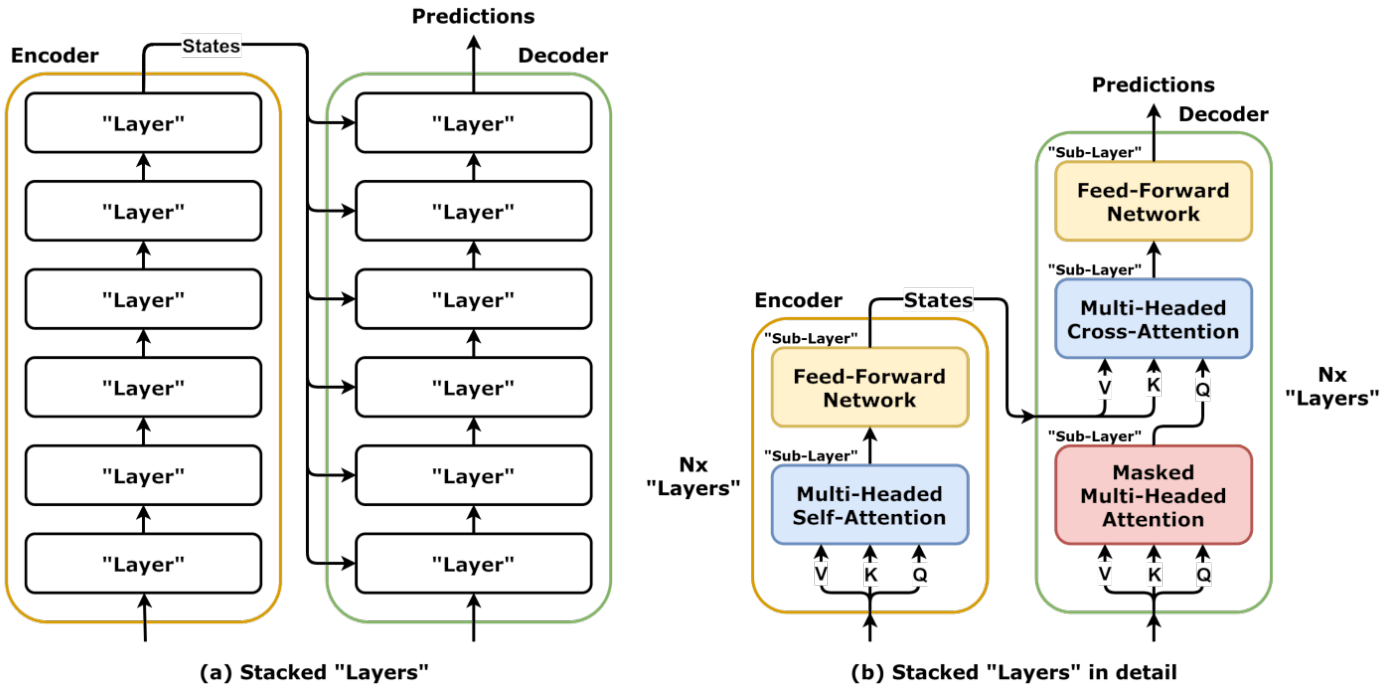
For example, in a translator made up of a simple RNN, we input a sentence in a continuous manner, one word at a time, to generate word embeddings. As every word depends on the previous word, its hidden state acts accordingly, so we have to feed it one step at a time.

In a transformer, however, we can pass all words of a sentence and determine the word embedding simultaneously.

It is also based on the multi-headed attention layer, so it easily overcomes the vanishing gradient issue of RNNs.

Transformers

The original transformer model used an **encoder-decoder architecture**.



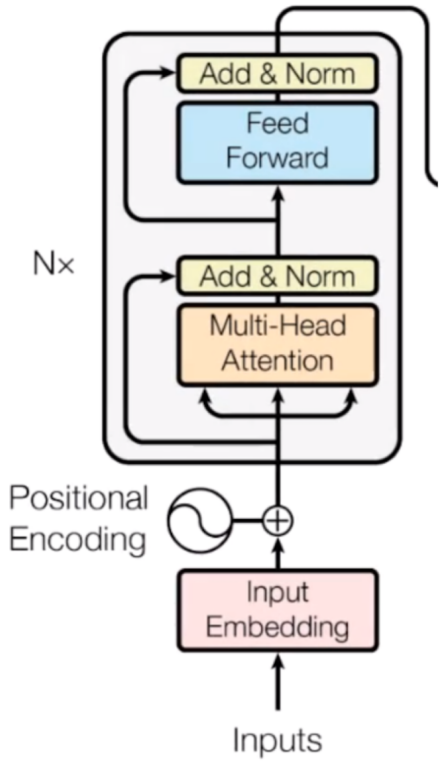
The encoder consists of encoding layers that process all the input tokens together one layer after another, while the decoder consists of decoding layers that iteratively process the encoder's output and the decoder's output tokens so far.

Transformers

All transformers have the same primary components:

- Tokenizers, which convert text into tokens.
- Embedding layer, which converts tokens and positions of the tokens into vector representations.
- Transformer layers, which carry out repeated transformations on the vector representations, extracting more and more linguistic information. These consist of alternating **attention** and **feedforward layers**. There are two types of transformer layers: encoder layers and decoder layers, with further variants.
- Un-embedding layer, which converts the final vector representations back to a probability distribution over the tokens.

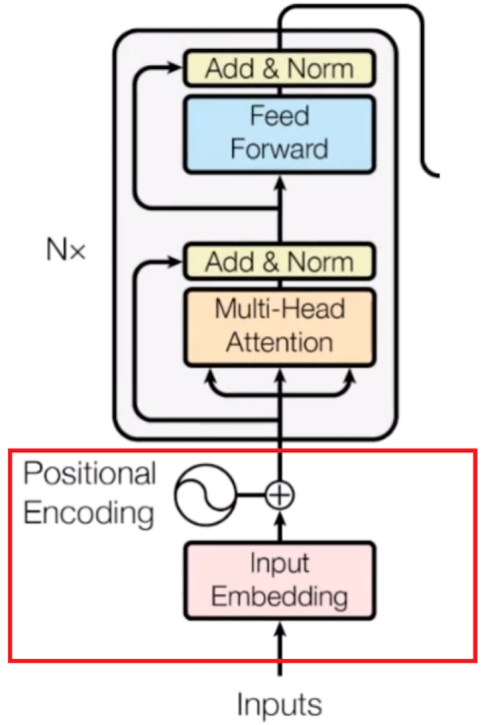
Transformers



Encoder block

The first step of the encoding block is embedding the input words into an Euclidean space - the embedding space.

Transformers



Encoder block

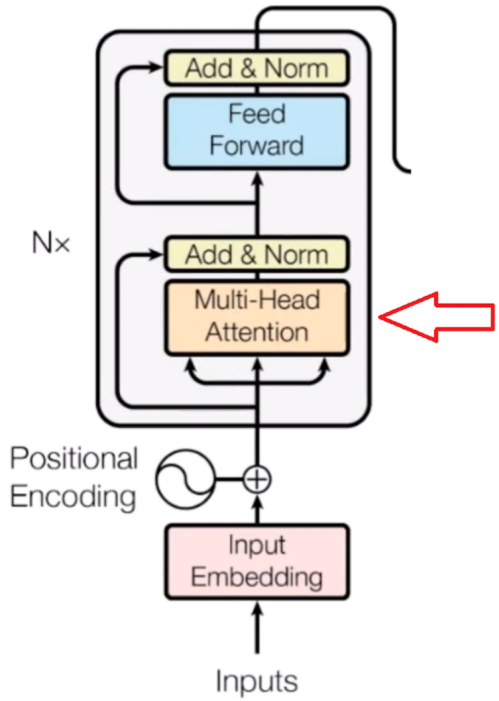
In different sentences, a word may have different meanings. To solve this issue, **positional encoders** are used, i.e., vectors that give context according to the position of the word in a sentence. Word $\rightarrow$ Embedding + Positional Embedding $\rightarrow$ Final Vector, framed as Context

Positional Encoding

Since Transformers do not have a recurrence mechanism like RNNs, they use positional encoding added to the input embeddings to provide information about the position of each token in the sequence.

To do so, they employ a combination of various sine and cosine functions to create positional vectors, enabling the use of this positional encoder for sentences of any length. In this approach, each dimension is represented by unique frequencies and offsets of the wave, with values ranging from -1 to 1, effectively representing each position.

Transformers



The **multi-head attention** is a critical network used to determine multiple attention vectors per word and take a weighted average to compute the final attention vector of every word.

Transformers

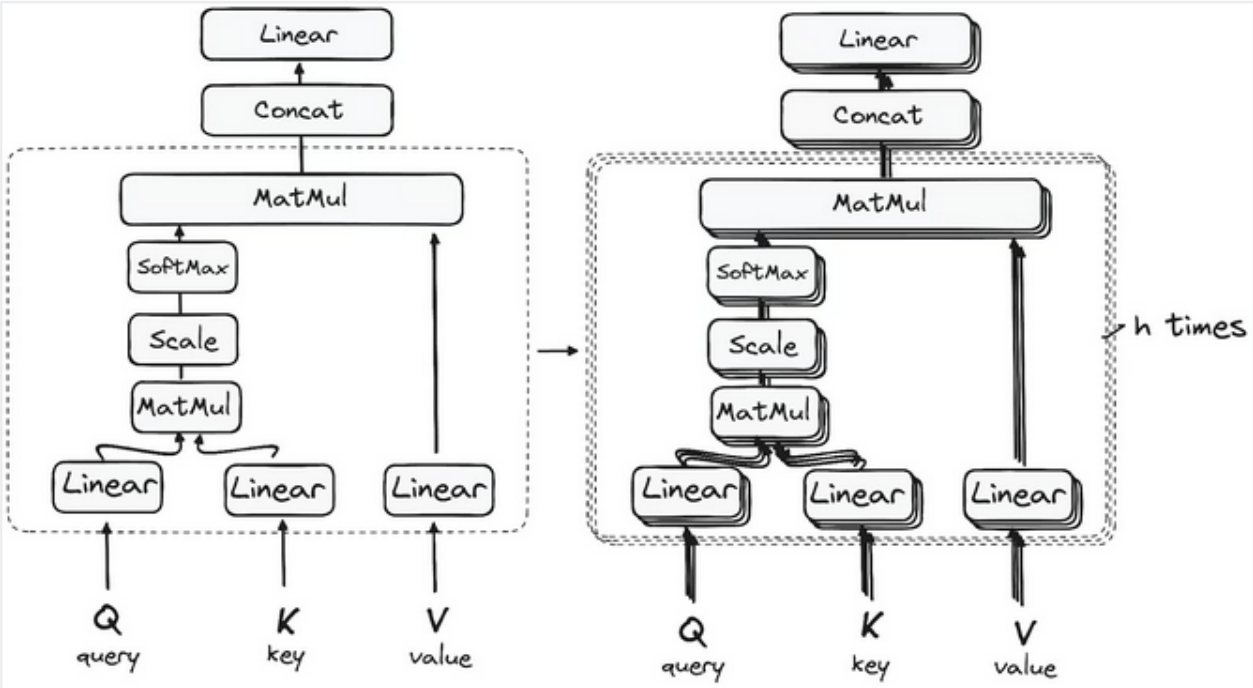
The multi-headed attention utilizes a specialized attention mechanism known as self-attention.

This approach enables the models to relate each word in the input with other words. For instance, in the sentence "how are you?", the model might learn to connect the word "are" with "you".

This mechanism allows the encoder to focus on different parts of the input sequence as it processes each token. It computes attention scores based on:

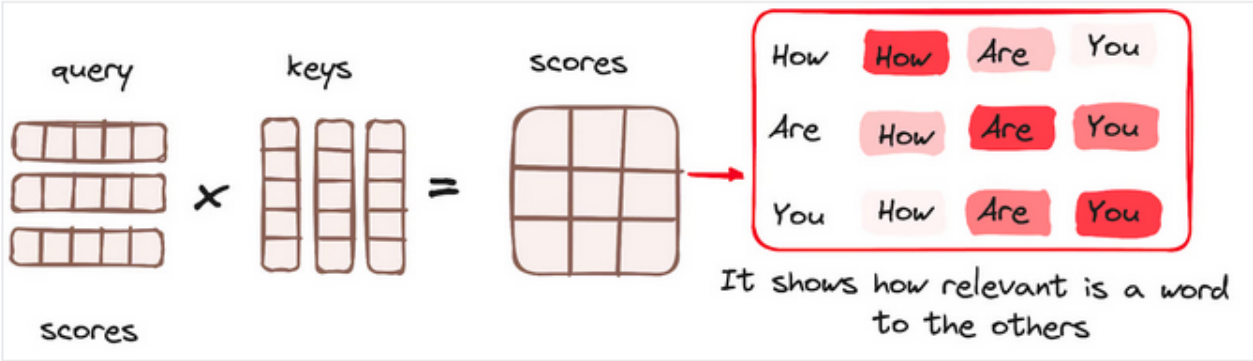
1. A **query** is a vector that represents a specific word or token from the input sequence in the attention mechanism.
2. A **key** is also a vector in the attention mechanism, corresponding to each word or token in the input sequence.
3. Each **value** is associated with a key and is used to construct the output of the attention layer. When a query and a key match well, which basically means that they have a high attention score, the corresponding value is emphasized in the output.

Transformers



This first Self-Attention module enables the model to capture contextual information from the entire sequence. Instead of performing a single attention function, queries, keys and values are linearly projected h times. On each of these projected versions of queries, keys and values the attention mechanism is performed in parallel, yielding h-dimensional output values.

Transformers

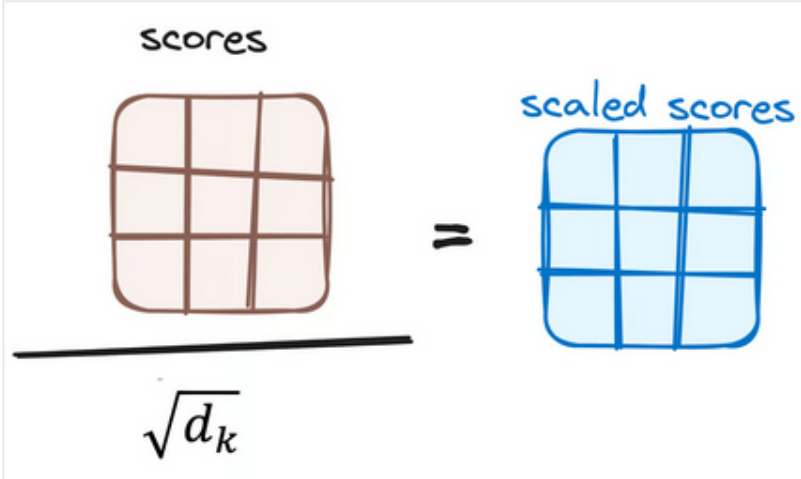


Once the query, key, and value vectors are passed through a linear layer, a dot product matrix multiplication is performed between the queries and keys, resulting in the creation of a score matrix.

The score matrix establishes the degree of emphasis each word should place on other words. Therefore, each word is assigned a score in relation to other words within the same time step. A higher score indicates greater focus.

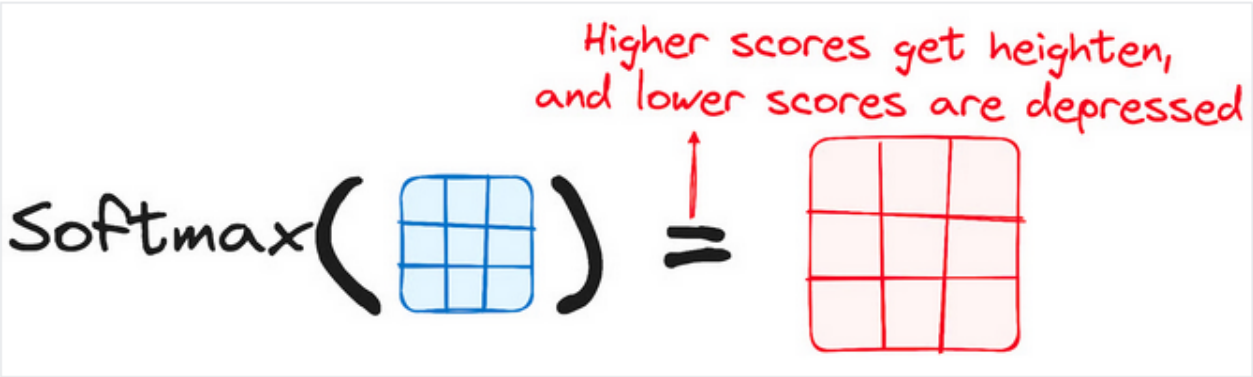
This process effectively maps the queries to their corresponding keys.

Transformers



The scores are then scaled down by dividing them by the square root of the dimension of the query and key vectors. This step is implemented to ensure more stable gradients, as the multiplication of values can lead to excessively large effects.

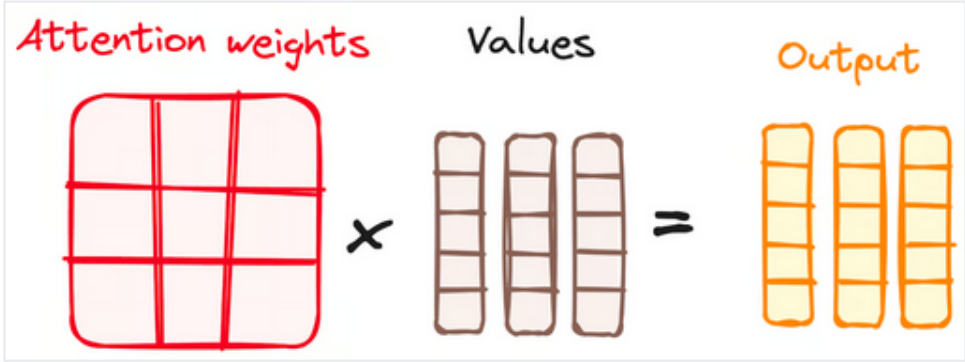
Transformers



A softmax function is applied to the adjusted scores to obtain the attention weights. This results in probability values ranging from 0 to 1.

The softmax function emphasizes higher scores while diminishing lower scores, thereby enhancing the model's ability to effectively determine which words should receive more attention.

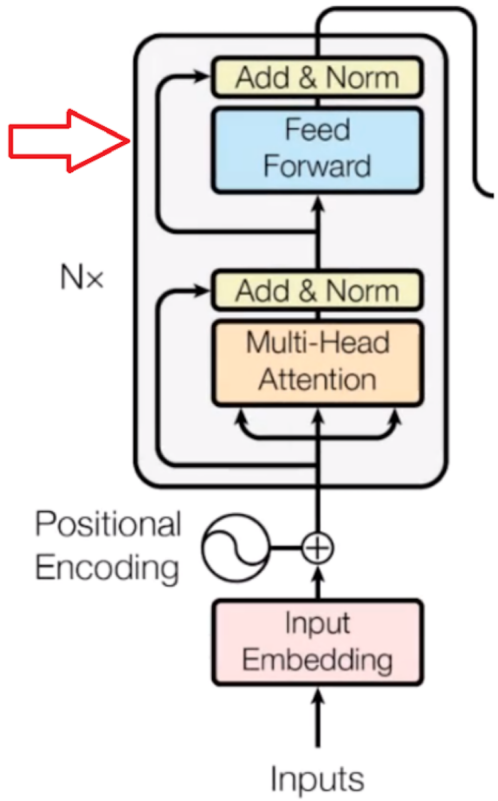
Transformers



Finally, the weights derived from the softmax function are multiplied by the value vector, resulting in an output vector.

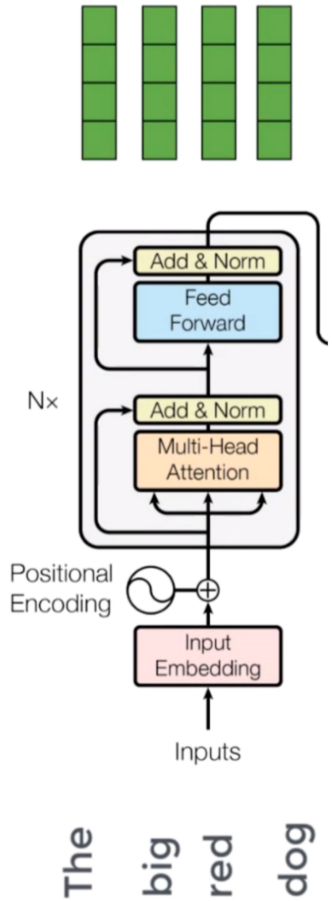
In this process, only the words that present high softmax scores are preserved. Finally, this output vector is fed into a linear layer for further processing.

Transformers



A **feed-forward neural network** is applied to every attention vector to transform the attention vectors into a form that is acceptable to the next encoder or decoder layer.

Transformers



The feed-forward network accepts attention vectors one at a time, each independently of one another. So, we can apply parallelization and pass all the words at the same time into the encoder obtaining the encoded vectors for every word simultaneously.

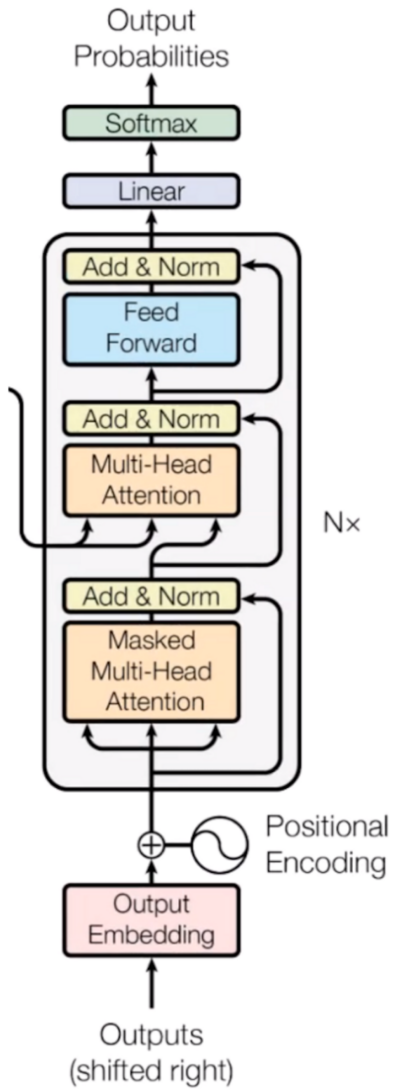
Output of the Encoder

The output of the final encoder layer is a set of vectors, each representing the input sequence with a rich contextual understanding. This output is then used as the input for the decoder in a Transformer model.

This careful encoding paves the way for the decoder, guiding it to pay attention to the right words in the input when it is time to decode.

Think of it like building a tower, where you can stack up N encoder layers. Each layer in this stack gets a chance to explore and learn different facets of attention, much like layers of knowledge. This not only diversifies the understanding but could significantly amplify the predictive capabilities of the transformer network.

Transformers



Decoder block

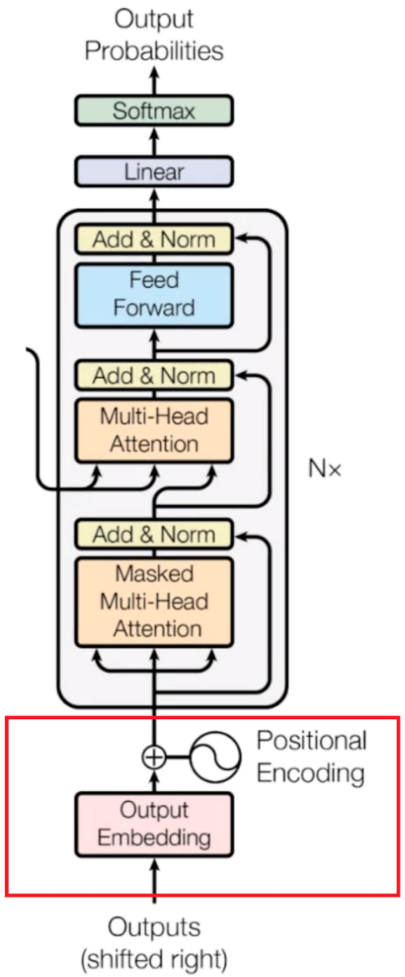
The Decoder WorkFlow

The decoder's role centers on crafting text sequences. Mirroring the encoder, the decoder is equipped with a similar set of sub-layers.

These components function in a way akin to the encoder's layers, yet with a twist: each multi-headed attention layer in the decoder has its unique mission.

The final step of the decoder's process involves a linear layer, serving as a classifier, topped off with a softmax function to calculate the probabilities of different words.
The Transformer decoder has a structure specifically designed to generate this output by decoding the encoded information step by step.

Transformers



In the decoder, embedding layer and positional encoder map the words into vectors similarly to the encoder block.

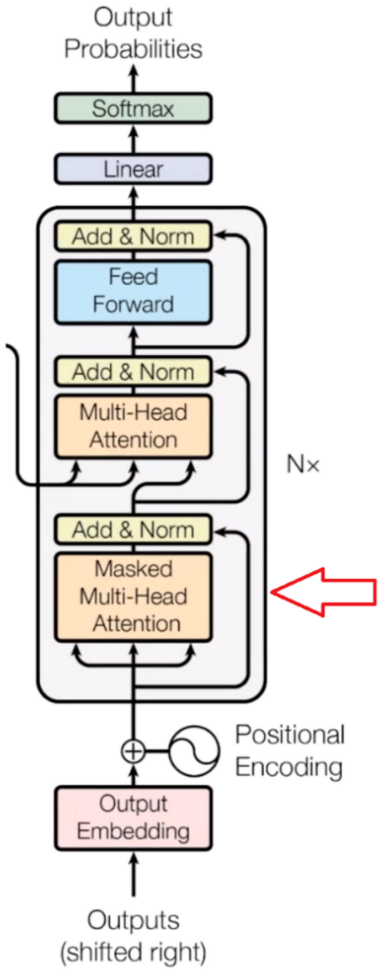
Transformers

The decoder operates in an autoregressive manner, kickstarting its process with a start token. It cleverly uses a list of previously generated outputs as inputs, in tandem with the outputs from the encoder that are rich with attention information from the initial input.

Suppose we are training a translator for English to French.

For training, we need to give an English sentence along with its translated French version for the model to learn.
So, our English sentences pass through encoder block, and French sentences pass through the decoder block.

Transformers



Encoded words pass through the self-attention block, where attention vectors are generated for every word in the French sentences to represent how much each word is related to every word in the same sentence, just like in the encoder block.

Transformers

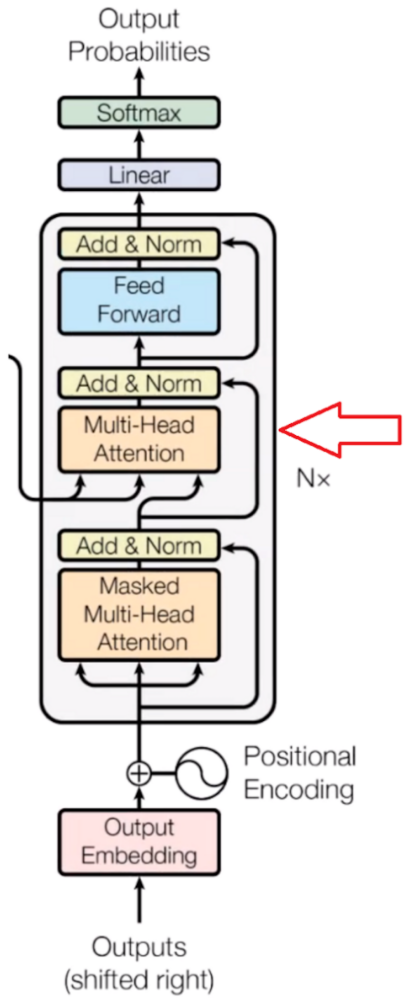
This block is called the **masked multi-head attention block**.

When we provide an English word, it is translated into its French version using previous results. It is then matched and compared to the actual French translation that we fed into the decoder block. After comparing both, it will update its matrix value. This is how it will learn after several iterations.

We observe that we need to hide the next French word so that, at first, it will predict the next word itself using previous results without knowing the real translated word. For learning to take place, it would make no sense if it already knows the next French word. Therefore, we need to hide (or mask) it.

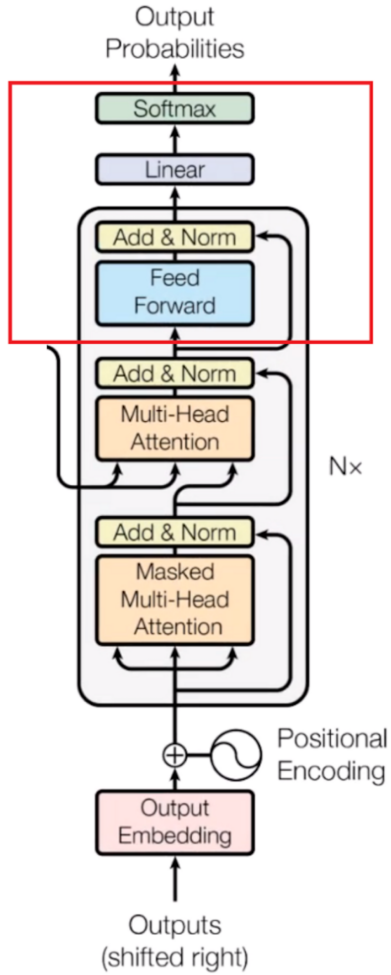
We can take any word from the English sentence, but we can only take the previous word of the French sentence for learning purposes. So, while performing parallelization with the matrix operation, we need to make sure that the matrix will mask the words appearing later by transforming them into zeroes so that the attention network can't use them.

Transformers



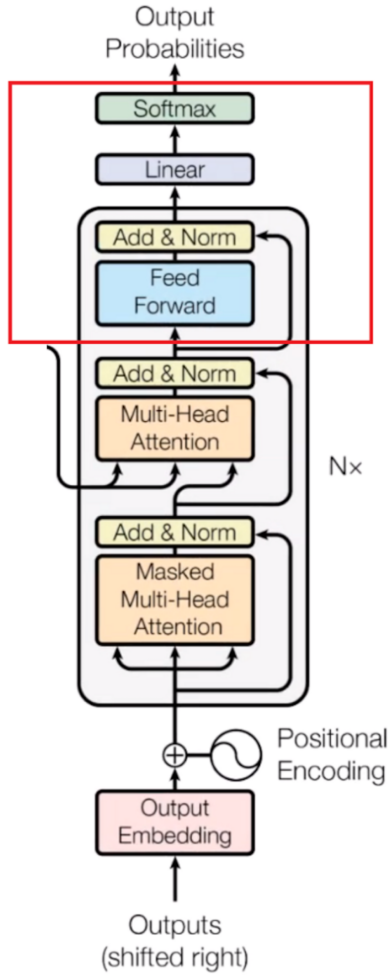
The attention vectors from the previous layer and the vectors from the encoder block are passed into another multi-head attention block. This is where the results from the encoder block also come into the picture.

Transformers



By passing each attention vector into a feed-forward unit, the output vectors is converted into a form that is easily acceptable by another decoder block or a linear layer.

Transformers



Finally the code is passed through a softmax layer that transforms the input into a probability distribution, and the resulting word is produced with the highest probability after translation.

Real-Life Transformer Models: BERT

Google's 2018 release of BERT, an open-source natural language processing framework, revolutionized NLP with its unique bidirectional training, which enables the model to have more context-informed predictions about what the next word should be.

By understanding context from all sides of a word, BERT outperformed previous models in tasks like question-answering and understanding ambiguous language. Its core uses Transformers, connecting each output and input element dynamically.

BERT, pre-trained on Wikipedia, excelled in various NLP tasks, prompting Google to integrate it into its search engine for more natural queries. This innovation sparked a race to develop advanced language models and significantly advanced the field's ability to handle complex language understanding.

Real-Life Transformer Models: LaMDA

LaMDA (Language Model for Dialogue Applications) is a Transformer-based model developed by Google, designed specifically for conversational tasks, and launched during the 2021 Google I/O keynote. They are designed to generate more natural and contextually relevant responses, enhancing user interactions in various applications.

LaMDA's design enables it to understand and respond to a wide range of topics and user intents, making it ideal for applications in chatbots, virtual assistants, and other interactive AI systems where a dynamic conversation is key.

This focus on conversational understanding and response marks LaMDA as a significant advancement in the field of natural language processing and AI-driven communication.

Real-Life Transformer Models: GPT

GPT and ChatGPT, developed by OpenAI, are advanced generative models known for their ability to produce coherent and contextually relevant text. GPT-1 was its first model launched in June 2018 and GPT-3, one of the most impactful models, was launched two years later in 2020.

These models are adept at a wide range of tasks, including content creation, conversation, language translation, and more. GPT's architecture enables it to generate text that closely resembles human writing, making it useful in applications like creative writing, customer support, and even coding assistance. ChatGPT, a variant optimized for conversational contexts, excels in generating human-like dialogue, enhancing its application in chatbots and virtual assistants.

Real-Life Transformer Models

The landscape of foundation models, particularly transformer models, is rapidly expanding. A study identified over 50 significant transformer models, while the Stanford group evaluated 30 of them, acknowledging the field's fast-paced growth. NLP Cloud, an innovative startup part of NVIDIA's Inception program, utilizes around 25 large language models commercially for various sectors like airlines and pharmacies.

There is an increasing trend towards making these models open-source, with platforms like Hugging Face's model hub leading the way. Additionally, numerous Transformer-based models have been developed, each specialized for different NLP tasks, showcasing the model's versatility and efficiency in diverse applications.

`https://www.datacamp.com/blog/what-are-foundation-models`

Vision transformers (Vit)

A **vision transformer (ViT)** is a transformer designed for computer vision.

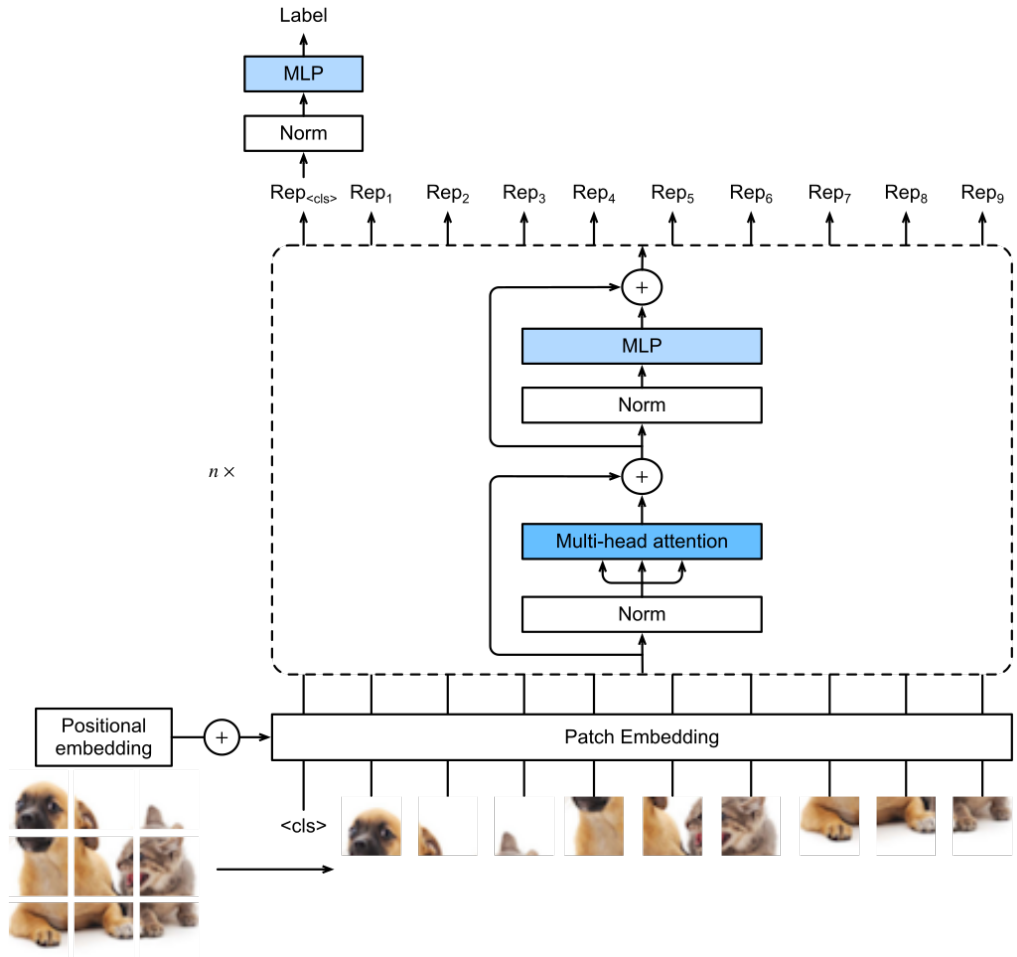
It carries out the following operations

1. it decomposes an input image into a series of patches (rather than text into tokens)
2. it serializes each patch into a vector
3. it maps the vector into a smaller dimension with a single matrix multiplication
4. the vector embeddings are then processed by a transformer encoder as if they were token embeddings.

ViTs were designed as alternatives to convolutional neural networks (CNNs) in computer vision applications. Compared to CNNs, ViTs are less data efficient, but have higher capacity.

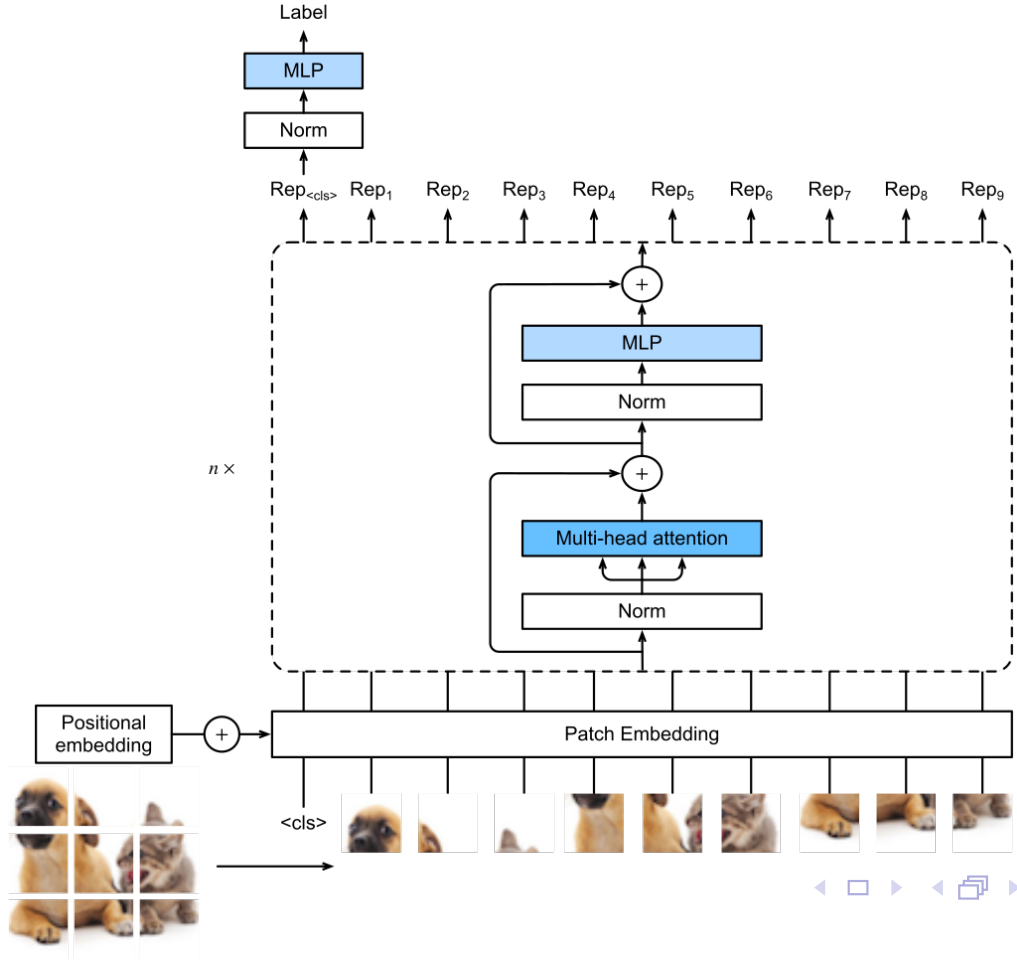
Vision transformers (ViT)

- In the basic ViT architecture, an image is first split into square-shaped patches.



Vision transformers (ViT)

- Each patch is pushed through a linear operator, to obtain "patch embedding" vector. The position of the patch is also transformed into a vector by "position encoding". The two vectors are added, then pushed through several Transformer encoders.



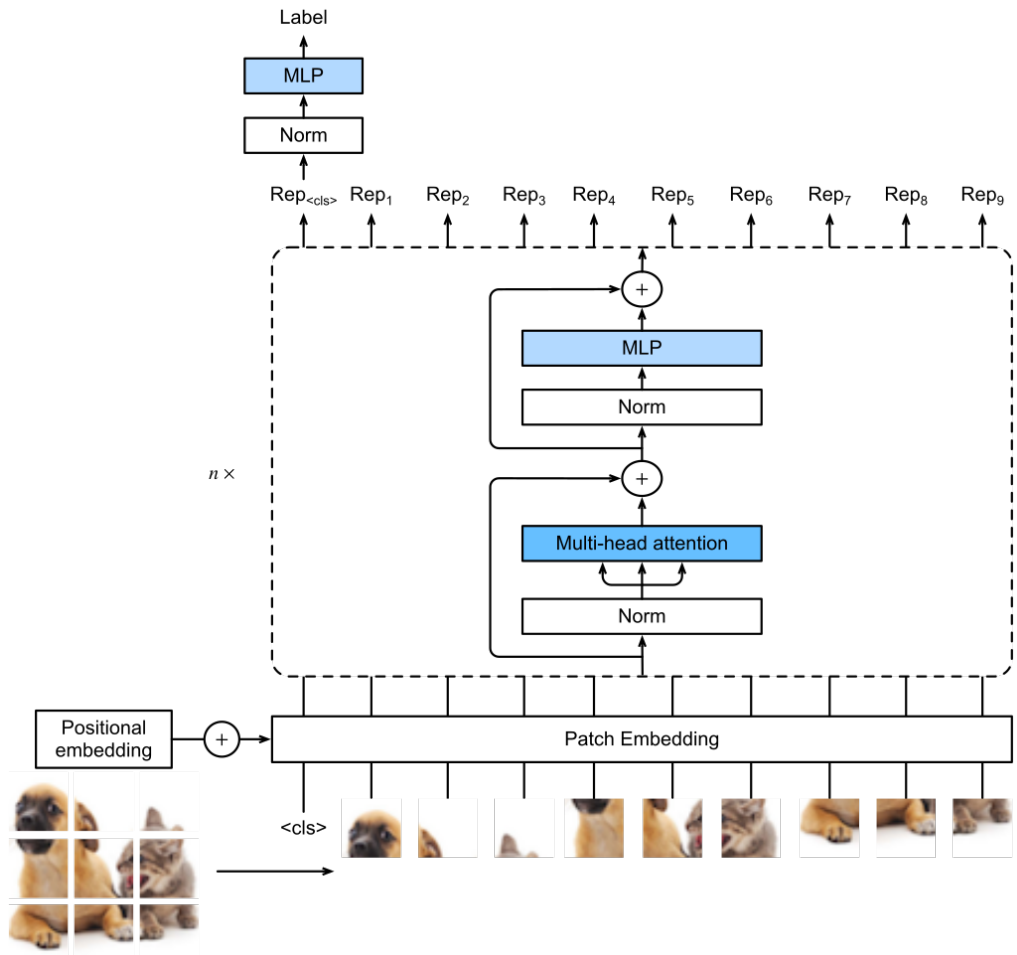
Vision transformers (ViT)

- The attention mechanism in a ViT repeatedly transforms representation vectors of image patches, incorporating more and more semantic relations between image patches in an image.
- This is analogous to how in natural language processing, as representation vectors flow through a transformer, they incorporate more and more semantic relations between words, from syntax to semantics.

The ViT turns an image into a sequence of vector representations. To use these for downstream applications, an additional head needs to be trained to interpret them.

Vision transformers (ViT)

- For example, to use the ViT for image classification, one can add a shallow MLP on top of it that outputs a probability distribution over classes.



ViT vs CNN

- Typically, ViT uses patch sizes larger than standard CNN kernels (3x3 to 7x7).
- ViT is more sensitive to the choice of the optimizer, hyperparameters, and network depth.
- ViTs require more data thah CNNs to train, but they can ingest more training data compared to CNN, which might not improve after training on a large enough training dataset.
- CNN applies the same set of filters for processing the entire image. This allows them to be more data efficient and less sensitive to local perturbations. ViT applies self-attention, allowing them to easily capture long-range relationships between patches.
- ViTs also appear more robust to input image distortions such as adversarial patches or permutations.

ViT applications

- ViTs provide state-of-the-art performance in many Computer Vision tasks such as Image Classification, Object Detection, Video Deepfake Detection, Image segmentation, Anomaly detection, Image Synthesis, Cluster analysis, Autonomous Driving.
- ViT had been used for image generation as backbones for GAN and for diffusion models (diffusion transformer, or DiT).
- DINO (self-distillation with no labels) - a refinement of the ViT proposed by researchers at META in 2021- has been demonstrated to learn useful representations for clustering images and exploring morphological profiles on biological datasets, such as images generated with the Cell Painting assay.
- In 2024, a 113 billion-parameter ViT model was proposed (the largest ViT to date) for weather and climate prediction, and trained on the Frontier supercomputer with a throughput of 1.6 exaFLOPs.

Analysis of data with temporal dependencies

The discipline of **dynamical systems** is concerned with developing tools used to model **dynamic datasets**, by which we mean **ordered data**, where often (but not always) ordering refers to the time variable.

Definition.
A **dynamical system with fixed order** consists of a function f and a sequence $(x_1, x_1, \dots)$ determined by the recursive equation

$$\begin{aligned} x_t &= \gamma_t, \qquad t = 1, \dots, W, \\ x_t &= f(x_{t-1}, \dots, x_{t-W}), \quad t > W \end{aligned}$$

where γ_t, for $t = 1, \dots, W$ are called the **initial conditions** of the system.