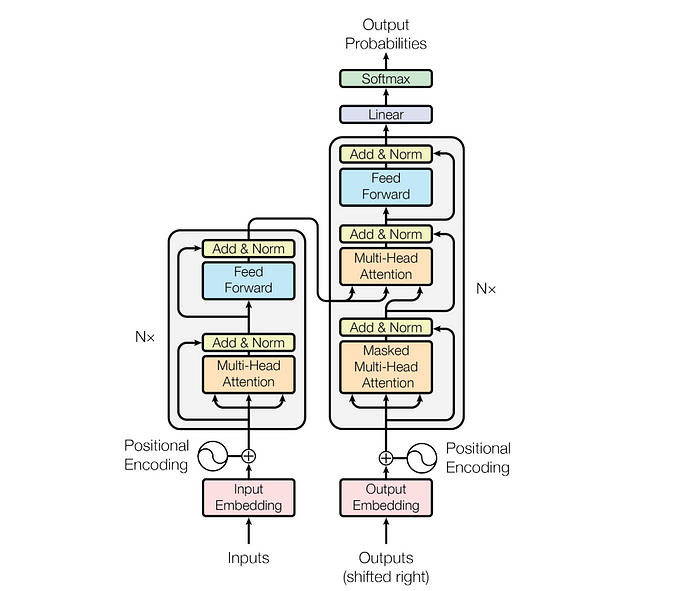
**TRANSFORMERS – Learning**

**Introduction:**

In recent years, artificial intelligence (AI) has undergone a major transformation with the introduction of **transformer-based models**. These models have played a key role in advancing **Natural Language Processing (NLP)**, while also finding applications in other areas of machine learning.

The transformer architecture is built around the idea of **attention**. Attention is a technique that helps the model focus on the most important parts of an input sequence, making it easier to understand the context. This has made transformers highly effective in tasks like **machine translation**, **text generation**, and many other NLP applications.

Another key advantage of transformers is their ability to process inputs in **parallel**, unlike older models like **RNNs** and **LSTMs**, which process data sequentially. This makes transformers faster, more efficient, and better suited for large-scale tasks.



To address the issue of **parallelization**, transformers use a combination of **encoders**, **decoders**, and **attention mechanisms**. The attention mechanism helps the model translate one sequence into another much faster. More specifically, it uses a technique called **self-attention**, which allows the model to focus on different parts of the input to better understand its meaning. A white rectangular card with red flames

Description automatically generated

Now that we have a basic understanding of the Transformer architecture, let’s take a closer look at the **Encoders** and **Decoders** to better understand how they function and work together.

**Embedding:**

Embeddings are a way to convert discrete data (like words or tokens) into continuous numerical vectors that capture their **semantic meaning** and **relationships**. Instead of representing words with sparse one-hot vectors (don't capture relationships), embeddings create dense, low-dimensional representations where words with similar meanings are placed closer together in the vector space.

Text input is divided into smaller units called **tokens**, which can be words or sub words. These tokens are converted into numerical vectors called **embeddings**, which capture the semantic meaning of words.

Let’s say you want to generate text using a Transformer model. You add the prompt like this one: **“Data visualization empowers users to”**. This input needs to be converted into a format that the model can understand and process. That is where embedding comes in: it transforms the text into a numerical representation that the model can work with. To convert a prompt into embedding, we need to

1. **Tokenization,**
2. **Token Embedding,**
3. **Positional Encoding,**
4. **Add up token and position encodings to get the final embedding.**

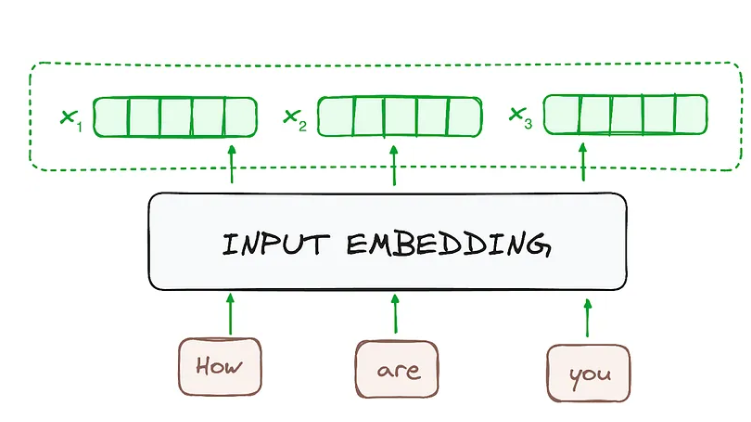


1. **Tokenization:**

Tokenization is the process of breaking down the input text into smaller, more manageable pieces called tokens. These tokens can be a word or a sub word. The words "Data" and "visualization" correspond to unique tokens.

1. **Token Embedding:**

The embedding only happens in the bottom-most encoder. The encoder begins by converting input tokens — words or sub words — into vectors using embedding layers. These embeddings capture the semantic meaning of the tokens and convert them into numerical vectors.

****

1. **Positional Encoding:**

Positional encoding is a technique used in models like Transformers to provide information about the order of words in a sequence to a model.

The self-attention module cannot capture the order of words in a sentence. For example, consider these two sentences:

1. Tiger killed the lion.
2. The lion killed Tiger.

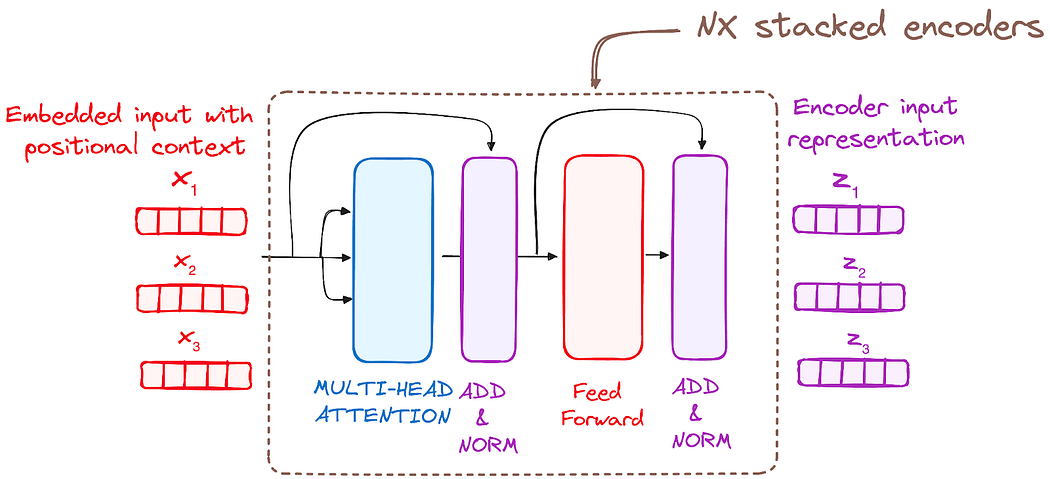
If you pass both sentences through a self-attention block, it won’t be able to distinguish between them, as it doesn’t capture the order of the words.

This limitation can lead to misunderstandings in meaning, as the module treats sentences with the same words in different orders as identical. Positional encodings are introduced in the transformer architecture to address this issue. **Positional encodings provide information about the order of words in a sentence, ensuring that the model understands the sequence and maintains the correct context.**

**A diagram of a bar

Description automatically generated with medium confidence**

**Encoders:**

The Transformer Encoder is a neural network architecture designed to process sequential data. It consists of a stack of N identical layers, with each layer built to encode the input sequence into a contextualized representation that captures relationships and dependencies between all tokens in the sequence.

The Transformer encoder consists of a stack of identical layers (6 in the original Transformer model). This number was achieved through experimentation, giving the best results for various tasks.

The encoder layer serves to transform all input sequences into a continuous, abstract representation that encapsulates the learned information from the entire sequence. This layer comprises two sub-modules:

* A multi-headed attention mechanism.
* Feed forward network.

The actual architecture of an encoder block includes additional components like **Add & Norm** **layers** and **Residual connections**. These ensure the flow of information remains smooth as it passes through each block.

The input data, typically a batch of sentences, enters the first encoder block, undergoes processing, and the output moves to the next encoder block. This process continues across all six encoder blocks, with the final output being passed to the decoder.

A diagram of a diagram of a program

Description automatically generated with medium confidence

* **Self-Attention Mechanism:**

The attention mechanism lays the foundation of the transformer model. Attention is a mechanism used to accurately evaluate how important different sections of an input sequence are to a global context.

Self-attention allows to model to weigh the importance of different tokens in the input sequences relative to each other. It also known as scaled dot product attention.

A diagram of a product

Description automatically generatedA diagram of a software algorithm

Description automatically generated

Each token’s embedding vector is transformed into three vectors: Query (Q), Key (K), and Value (V). These vectors are derived by multiplying the input embedding matrix with learned weight matrices for Q, K, and V. Here’s a web search analogy to help us build some intuition behind these matrices:

* **Query (Q)** is represent token for which we are calculate attention. This is the token you want to “find more information about”.
* **Key (K)** is represent all token in sequence and used to compare with query vectors to calculate attention score. It represents the possible tokens query can attend to.
* **Value (V)** is the actual content that will be aggregated to form the output of the attention mechanism. Preserve and aggregates relevant content from sequence.

A black text on a white background

Description automatically generatedBy using these QKV values, the model can calculate attention scores, which determine how much focus each token should receive when generating predictions.

* **Multi-Head Attention:**

performing a single attention operation with dₘ-dimensional keys, Queries and Values could be done, where dₘ is the dimensionality of the model. However, a more effective method exists.

Multi-Head Attention linearly projects the initial **Queries**, **Keys** and **Values** **n times** to matrices with **dₖ**, **dₖ** and **dᵥ** dimensionality, respectively. Following this, the standard attention mechanism is applied to the **n** unique **Q**, **K** and **V** projections in **parallel**.

By having each attention head compute its own attention weights and context vector **independently**, the transformer is able focus on different aspects of the input sequence simultaneously while capturing complex patterns and relationships more effectively.

Subsequently, the **n** resulting outputs are then concatenated and projected via **W**ₒ. The purpose of **Wₒ** is to combine the results from each individual attention head into one final output. With a single attention head, such results would not be possible.

A close-up of a text

Description automatically generated

* **Normalization and Residual Connections:**

A diagram of a multi-head attention

Description automatically generatedEach sub-layer in an encoder layer is followed by a normalization step. Also, each sub-layer output is added to its input (residual connection) to help mitigate the vanishing gradient problem, allowing deeper models. This process will be repeated after the Feed-Forward Neural Network too.

* **Feed-Forward Neural Network:**

In addition to the attention sub-layers, both the encoder and decoder implement another sub-layer called the fully connected feed-forward layer. This layer is applied independently and parallelly to each position of the input sequence.

A fully connected network applied independently to each token. FFN consists of two linear transformations with a non-linear activation (ReLU) in between:

FFN(x) = max(0, xW_1 + b_1)W_2 + b_2.

A diagram of a graph

Description automatically generated

**Decoder:**

The transformer decoder is responsible for generating the output sequence one token at time, using the encoder’s output and previously generated tokens.

The Transformer Decoder has **6 identical layers**, like the encoder, but includes an extra sub-layer for multi-head attention on the encoder's output. Residual connections and layer normalization are used throughout. To ensure the model is **auto regressive**, the first sub-layer applies masking and offsets the output embeddings, so each position depends only on the previous ones.

The Transformer Decoder generates text sequences step by step. It has two multi-head attention layers, a feed-forward layer, and uses residual connections with layer normalization. One attention layer focuses on the target sequence, while the other attends to the encoder's output.

The process starts with a **start token** and uses previously generated tokens along with encoder outputs. A final linear layer with a **SoftMax** function predicts word probabilities. The decoder works **autoregressively**, generating tokens until it produces an **end token**.

A diagram of a flowchart

Description automatically generated

* **Components:**
  1. Masked Multi-head Self Attention
  2. Multi-head Attention (Encoder-Decoder Attention)
  3. Feed Forward Neural Network
  4. Normalization and Residual Connections

**1. Masked Self-Attention Mechanism:**

This is similar to the self-attention mechanism in the encoder but with a crucial difference is, it prevents positions from attending to subsequent positions, which means that each word in the sequence isn’t influenced by future tokens.

* **Steps:**

1. I/P embedding + postional embedding
2. Linear Projection for Q, K, V
3. Scaled dot product attention
4. **Mask Application**
5. Multi-head attention
6. Concatination & Final projection
7. Normalization and Residual Connections

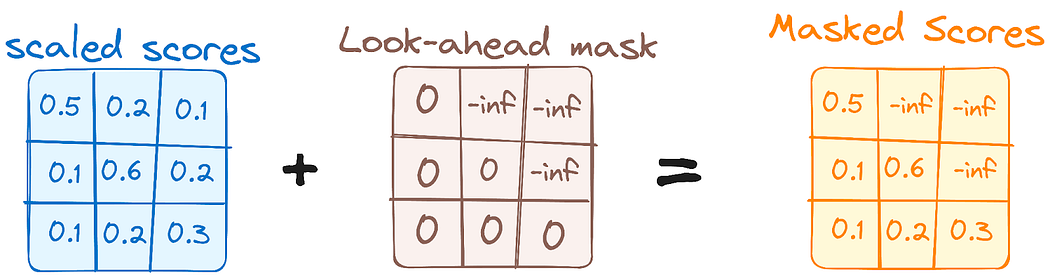
**2. Mask Application:**

It helps manage the structure of sequences being processed and ensure the models behave correctly during training and inferencing.

* Handling Variable length sequence with padding mask
* To Handle the padding tokens, which are added to make sequences of uniform length, so do not affect model prediction

Types of mask:

* **Padding Mask**
* Tokens are ingnored
* **Look ahead mask**
* Maintain Auto-regressive property
* To ensure each position in the decoder output sequence can only attend to the previous position

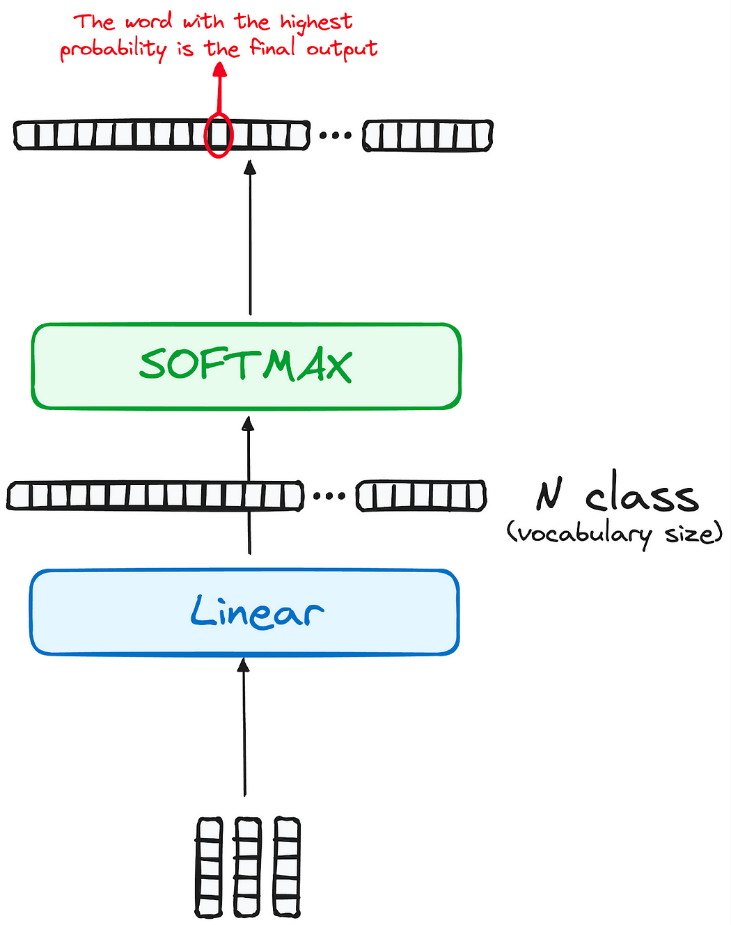


**3. Linear Classifier and Softmax for Generating Output Probabilities:**

It is the process of Concatination & Final projection. The journey of data through the transformer model culminates in its passage through a final linear layer, which functions as a classifier.

The size of this classifier corresponds to the total number of classes involved (number of words contained in the vocabulary). For instance, in a scenario with 1000 distinct classes representing 1000 different words, the classifier’s output will be an array with 1000 elements.

This output is then introduced to a softmax layer, which transforms it into a range of probability scores, each lying between 0 and 1. The highest of these probability scores is key,its corresponding index directly points to the word that the model predicts as the next in the sequence.



**4. Normalization and Residual Connections:**

Each sub-layer (masked self-attention, encoder-decoder attention, feed-forward network) is followed by a normalization step, and each also includes a residual connection around it.

**All Together: The Transformer**

Now combine all these previously discussed components and we have a transformer

The transformer’s primary objective is to perform sequence-to-sequence tasks such as machine translation, text summarisation and other NLP tasks. It does this by capturing complex relationships and long-range dependencies, something previous models struggled to achieve.

The attention mechanism is fundamental to the transformer, by focusing on several areas of a sequence in parallel, it is able to surpass other traditional models such as RNNs in both efficiency and accuracy.

The transformer follows an encoder-decoder structure. At first, word embedding vectors of the input sequence are created. These are then added element-wise to the positional encoding vectors. This resulting matrix is then fed into the encoder and decoder, where it will then under go multi-head self-attention, fully connected feed-forward networks and layer normalisation. Residual connections are additionally implemented in order to help preserve information and make the model more efficient. In order for the encoder to communicate with the decoder, the encoder’s output is used as two of the inputs in the second multi-head attention layer of the decoder.

A diagram of a software system

Description automatically generatedAfter the encoder-decoder mechanism, the outputs are linearly transformed and fed through a softmax function in order to obtain probabilities. In tasks such as text generation, these probabilities would represent the likelihood of a particular token or word appearing next in the generated output sequence.

**Example:**

The model architecture is based on a **Neural Machine Translation (NMT)** framework, using an **Encoder-Decoder Long Short-Term Memory network** with an **embedding layer**. It leverages the **sequence-to-sequence** approach, which is particularly effective for tasks like language translation. The model is enhanced with **attention mechanisms**, allowing it to focus on relevant parts of the input sequence during the decoding process, thus improving the accuracy of the translation. This architecture is a common implementation for neural network-based language translation tasks, combining the strengths of LSTMs for sequential data processing and attention mechanisms for better alignment between input and output sequences.

