**INDEX**

[**1. Introducing the Sequence to Sequence (Seq2Seq) model 9**](#_da9oslwuqacv)

[**2. Seq2Seq Model Architecture 10**](#_z5kheg1rg3wm)

[**2.1 Training Process 10**](#_9lrnbnf352y)

[**2.2 Prediction Process 12**](#_2oofo7ef99uw)

[**3. Limitations of the Seq2Seq Model with Encoder-Decoder Architecture 14**](#_1o3zrx3s6iv)

[**3.1. Introduction to Attention 15**](#_qkeepspiqfzx)

[**3.2. Attention Mechanism and the Transformer Model 17**](#_t5dfslc95ehi)

[**3.3. Transformer Model 20**](#_9vu2h8y4xwv2)

[**4. Attention in GPT (Generative Pretrained Transformer) 22**](#_w5lge84nn14x)

[**5. Differences Between Attention Mechanisms 24**](#_xf058zw9967z)

[**Referent**](#_rkznq8l77gv) **26**

Chapter 1

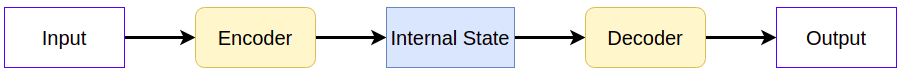
# 1*. Introducing the Sequence to Sequence (Seq2Seq) model*

The Seq2Seq model was first introduced in 2014 by Google. Its purpose is to map a fixed-length Input Sequence Data to a fixed-length Output Sequence Data. The length of 2 Sequence Data does not have to be the same. For example, when translating a sentence with 5 words What are you doing now? from English to the 7-character sentence 今天你在做什麼？ in Chinese.

The Seq2Seq model can solve the following problems:

* Text Summarization: Summarizing long texts into shorter, concise versions. Popularized by Google since 2014.
* Machine Translation: Translating text between different languages (e.g., Google Translate).
* Image/Video Captioning: Generating descriptive text for images or videos. A practical example could be describing images in the TOEIC exam.
* Speech Recognition: Also known as Speech-to-Text, converting audio into written text.
* Music Generation: A fascinating task where AI composes music.
* Recommendation Engines: Familiar systems that suggest content, often using models like Seq2Seq with Encoder-Decoder architectures.
* Chatbots: Question-and-answer systems like Siri or Alexa.
* These tasks share a common feature: they process sequential data over multiple time steps, such as text (words in a sentence), audio, or other signal data.

# 2. Seq2Seq Model Architecture

The Seq2Seq model consists of two main components: the Encoder and the Decoder. Each of these components is composed of multiple neural network (NN) layers stacked on top of each other. The type of NN layer can vary and include options like CNN, RNN, LSTM, GRU, etc. For this explanation, we'll use LSTM as an example.

* Input: The input sequence provided to the model.
* Encoder: Processes the input through stacked LSTM (or other neural network layers) to generate an Internal State (context vector).
* Internal State: A fixed-size vector representing the entire input sequence.
* Decoder: Takes the context vector as input to generate the output sequence step by step.
* Output: The final generated sequence.

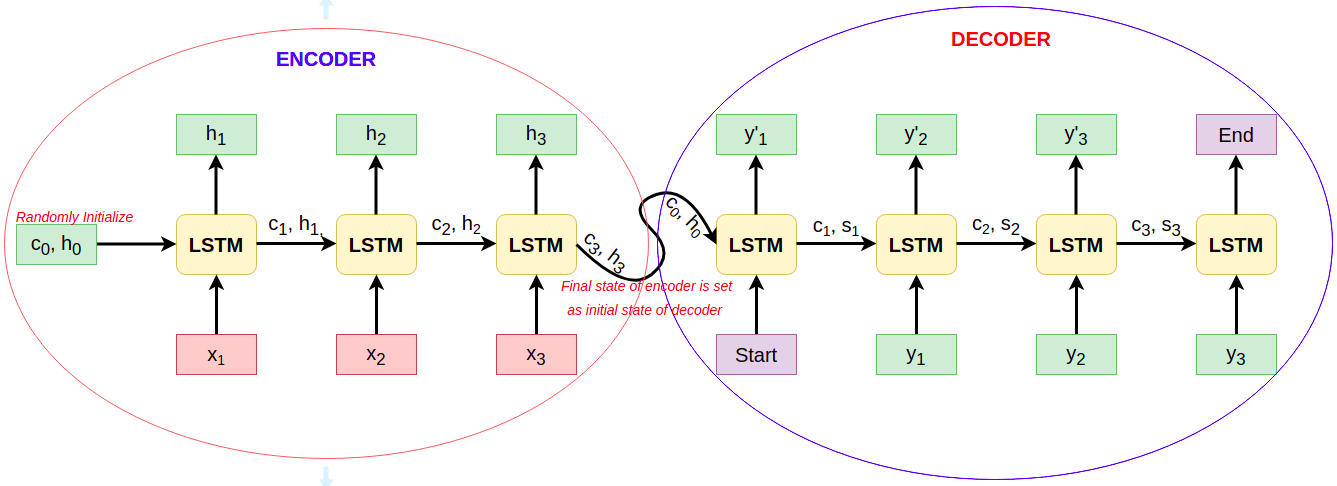
This visual complements the explanation of how Seq2Seq models function, emphasizing the flow of information from input to output through the encoder-decoder pipeline. Let me know if you want further explanation or enhancement based on this diagram!

## 2.1 Training Process

During training, each component of the Seq2Seq model performs the following tasks:

* **Encoder:** The encoder processes the entire input sequence step by step, feeding each time step into LSTM cells.
* At time step t:
* The cell outputs a **Hidden State (hth\_tht​)** and a **Cell State (CtC\_tCt​)**, collectively referred to as the **Internal State**.
* The **Internal State** from the previous time step (ht−1,Ct−1h\_{t-1}, C\_{t-1}ht−1​,Ct−1​) is used along with the input of the current time step as input to the current LSTM cell.
* The initial internal state (h0,C0h\_0, C\_0h0​,C0​) is initialized randomly.
* The **Internal State** from the last LSTM cell of the encoder (hT,CTh\_T, C\_ThT​,CT​) is used as the initial state for the decoder.
* **Decoder:**
* The decoder processes the entire **Target Sequence** step by step.
* Unlike the encoder, the target sequence is augmented with:
* A **prefix** (START\_) indicating the beginning.
* A **suffix** (\_END) indicating the end
* The decoder’s initial **Internal State** (h0,C0h\_0, C\_0h0​,C0​) is set to the final **Internal State** of the encoder (hT,CTh\_T, C\_ThT​,CT​).
* At each time step ttt, the decoder:
* Reads one word from the target sequence.
* Produces a **predicted word (yt′y\_t'yt′​)** and updates the **Internal State (ht,Cth\_t, C\_tht​,Ct​)**.
* The updated **Internal State** is used in the next time step.
* The predicted word (yt′y\_t'yt′​) is compared with the actual target word to calculate the **error**
* Backpropagation
* The error is backpropagated through the model to adjust its weights and minimize the loss.
* The **Internal State** from the decoder's last LSTM cell is discarded as it is not needed.

This training process ensures that the encoder learns to represent the input sequence effectively, and the decoder learns to generate the correct output sequence.



## 2.2 Prediction Process

The **prediction process** for the Seq2Seq model involves the following steps:

**Encoder**:

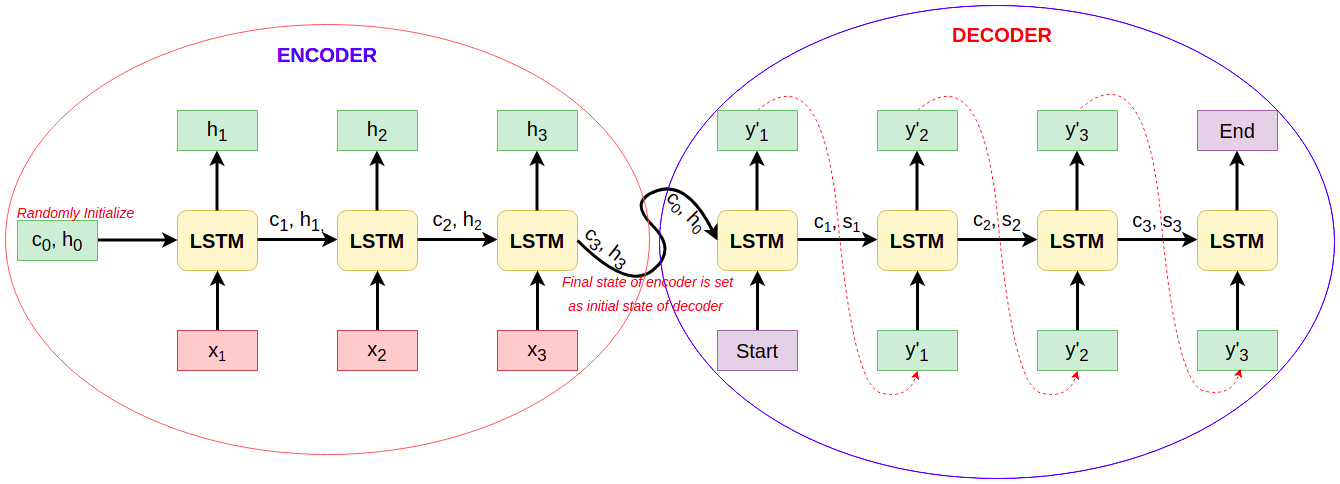
* The encoder's behavior during prediction is the same as during training:
* It processes the entire input sequence step by step.
* The final **Internal State** (hT,CTh\_T, C\_ThT​,CT​) from the encoder is passed to the decoder as its initial state.

**Decoder**:

The decoder works differently during prediction compared to training. The steps are as follows:

1. **Initialization**:
   * The initial **Internal State** (h0,C0h\_0, C\_0h0​,C0​) of the decoder is set to the final **Internal State** from the encoder (hT,CTh\_T, C\_ThT​,CT​).
2. **Starting Input**:
   * The input for the first time step of the decoder is always the special token START\_.
3. **Generating Words**:
   * At each time step:
     + The decoder's LSTM cell generates a predicted word (yt′y\_t'yt′​).
     + The **Internal State** (ht,Cth\_t, C\_tht​,Ct​) is updated and passed to the next time step.
4. **Feedback Loop**:
   * The predicted word (yt′y\_t'yt′​) from the current time step is used as the input for the next time step.
5. **Termination**:
   * The prediction process continues until the decoder generates the special token \_END, which signifies the end of the sequence.

This iterative process allows the decoder to generate an entire output sequence one word at a time based on the context provided by the encoder and its own previous predictions.



# 3. Limitations of the Seq2Seq Model with Encoder-Decoder

The Encoder-Decoder architecture performs well with short input sequences, but its performance declines as the length of the input sequence increases. Here are the key limitations:

* **Fixed-Length Context Vector:**
  + The encoder processes the input sequence x1,x2,…,xn{x\_1, x\_2, \dots, x\_n}x1​,x2​,…,xn​ and encodes it into a fixed-length vector (Context Vector), typically the final hidden state (hnh\_nhn​).
  + Only the final Context Vector (hnh\_nhn​) is passed to the decoder for generating the output sequence.
  + As a result, the entire input sequence's information must be compressed into a single vector, which can lead to information loss, especially for long sequences.
* **Information Loss:**
  + For longer input sequences, the Context Vector may not fully capture all the important details of the input. This leads to degraded performance in tasks requiring detailed understanding of the input, such as machine translation or summarization.
* **Inefficient Use of Input Information:**
  + Since only the final Context Vector (hnh\_nhn​) is used by the decoder, earlier parts of the input sequence (x1,x2,…x\_1, x\_2, \dotsx1​,x2​,…) may be underutilized or even ignored.

**Solution: Attention Mechanism**

The Attention mechanism was introduced as an effective solution to this limitation. Instead of relying solely on a single Context Vector (hnh\_nhn​):

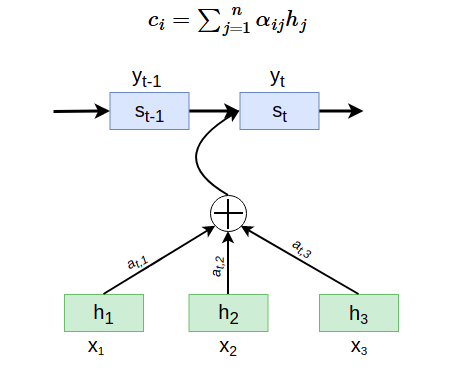
* The decoder dynamically attends to all encoder outputs (h1,h2,…,hnh\_1, h\_2, \dots, h\_nh1​,h2​,…,hn​) at each time step.
* This allows the decoder to focus on the most relevant parts of the input sequence during prediction, mitigating the problem of information loss and improving performance on longer sequences.

The integration of Attention has significantly improved the Seq2Seq model's effectiveness, especially in complex tasks like translation and summarization.

## 3.1. Introduction to Attention

Attention is a technique introduced in the papers by Bahdanau et al. (2014) and Luong et al. (2015). Its core idea is to allow the Decoder to utilize information from the entire input sequence, while dynamically focusing on the most relevant parts at each time step.

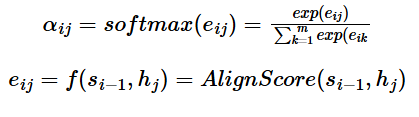
In simpler terms, Attention creates a **Context Vector** by calculating a **weighted average** of all the internal states (hidden states) of the input sequence generated by the Encoder.



**In the Attention mechanism:**

* αij represents the **weight** between the iii-th time step of the Encoder and the jjj-th time step of the Decoder. In other words, the jjj-th output of the Decoder should pay an attention weight of αij\alpha\_{ij}αij​ to the iii-th input of the Encoder.
* hi is the **Hidden State** of the Encoder at the iii-th time step..
* n is the **length of the Input Sequence**, which determines the total number of Encoder time steps.

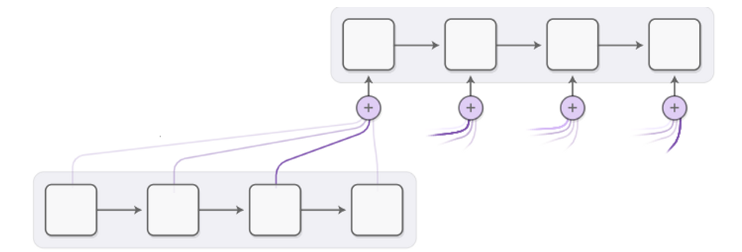
αij  is calculated by taking the Softmax of Attention Score Attention Score (eij):



**In the Attention mechanism:**

* hi−1 is the **Hidden State** at time step i−1i-1i−1 of the Decoder, representing the Decoder's context from the previous time step.
* sj is the **Hidden State** at time step jjj of the Encoder, representing the encoded information for the jjj-th input.

The **Context Vector** cic\_ici​ is then used by the Decoder to compute the output yiy\_iyi​.



## 3.2. Attention Mechanism and the Transformer Model

A major limitation of Recurrent Neural Networks (RNNs) is that all words in a sequence have the same impact on the final result. This leads to suboptimal performance in sequence-to-sequence tasks (such as Named Entity Recognition and Machine Translation) compared to LSTM-based encoder-decoder models. In reality, specific words in the input sequence often have more influence on the output sequence than others.

Let’s consider a **Sequence-to-Sequence model**, such as machine translation. This is typically performed by two Recurrent Networks: one (the **encoder**) converts the input sequence into hidden states, and the other (the **decoder**) transforms those hidden states into a sequence of translated words. The problem with this approach is that the final state of the network struggles to store information about the earlier parts of a sentence, resulting in poor quality for longer sentences.

**Attention Mechanisms:**

The **Attention mechanism** provides a way to assign context-dependent weights to each input vector based on its contribution to the output prediction of the RNN. This is achieved by creating intermediate "shortcut" weights between the hidden states of the input RNN and the output RNN.

In this way, when generating an output token yty\_tyt​, we consider the input hidden states hih\_ihi​ with different attention weights at,ia\_{t,i}at,i​. These weights allow the model to focus on more relevant parts of the input sequence for each specific output prediction.

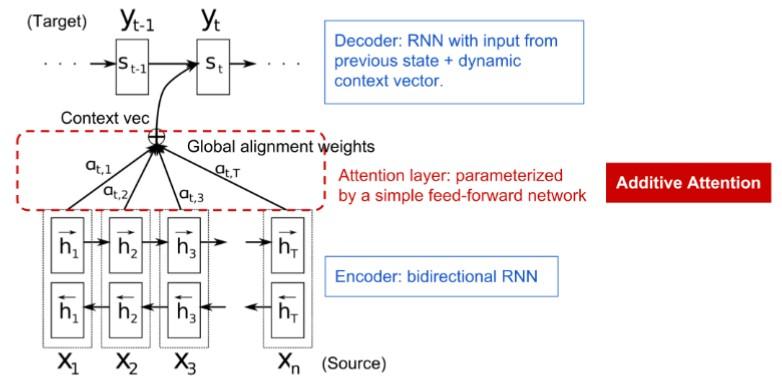
**The Transformer Model:**

The Transformer model builds on this idea of attention by using a self-attention mechanism (also called scaled dot-product attention), which allows the model to weigh the importance of each word in the input sequence relative to every other word, not just sequentially. Unlike RNNs, the Transformer does not process the sequence in a fixed order but instead uses attention mechanisms to process the entire sequence in parallel, improving efficiency and performance.

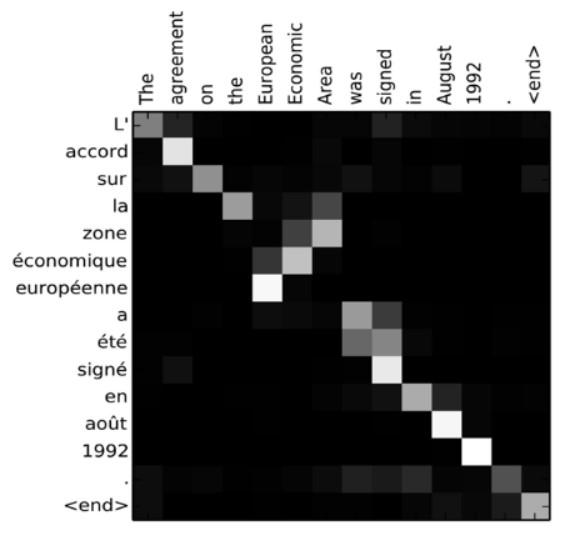
Key advantages of the Transformer:

* Parallelization: Since the Transformer processes the entire sequence at once, it can take advantage of parallel computation, making it more efficient for large datasets.
* Long-Range Dependencies: The attention mechanism allows the Transformer to capture long-range dependencies within the sequence, overcoming the limitation of RNNs that struggle to remember distant parts of the input.
* Scalability: Transformers are highly scalable and have been the backbone of state-of-the-art models in NLP (such as BERT and GPT).

Thus, the Attention mechanism improves sequence-to-sequence tasks by allowing the model to focus on relevant input parts, and the Transformer takes this a step further by using self-attention to process all elements of the sequence simultaneously.



The **attention matrix** ({αi,j}\{ \alpha\_{i,j} \}{αi,j​}) represents the degree to which certain input words contribute to generating a specific word in the output sequence. It provides a way to visualize which parts of the input the model is focusing on when generating each output token.



The **Attention mechanism** plays a crucial role in handling complex tasks in modern natural language processing (NLP). However, adding attention values significantly increases the number of parameters in the model, which leads to challenges related to scaling the model in Recurrent Neural Networks (RNNs). One major limitation in scaling RNNs is the inherent sequential nature of these models, which makes training more difficult. This is because, in an RNN, each element in a sequence must be processed in order, meaning we cannot parallelize the training process.

The application of the **Attention mechanism**, combined with this limitation, has led to the development of **Transformer models**, which have achieved very high performance. Models like **BERT** and **GPT-3** (OpenAI's Generative Pretrained Transformer 3) are examples of such high-performing Transformer models that have revolutionized the field of NLP.

The key advantage of Transformers is that they process the entire input sequence simultaneously using attention, avoiding the sequential bottleneck of RNNs. This allows for better parallelization, faster training, and the ability to handle long-range dependencies more effectively, making them ideal for large-scale NLP tasks.

## 3.3. Transformer Model

Instead of passing the context of each previous prediction to the next step, Transformer models use **Positional Encoding** and the **Attention mechanism** to capture the context of a given input dataset within a text sequence.

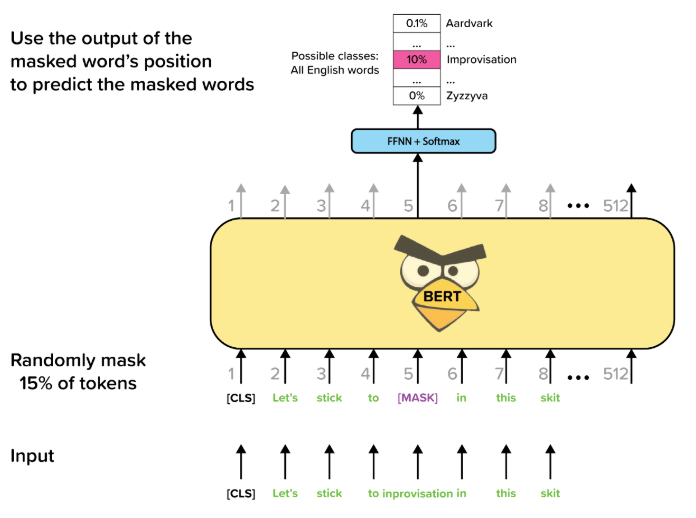
Since each input position is mapped independently to each output position, the Transformer model can perform parallel processing more efficiently than RNNs. This allows for the scaling of larger language models. Each attention head can be used to learn different relationships between words, improving various natural language processing (NLP) tasks.

**BERT (Bidirectional Encoder Representations from Transformers)**

**BERT** is a large multi-layer Transformer model with 12 layers for **BERT-base** and 24 layers for **BERT-large**. The model is pre-trained on a large text dataset (Wikipedia and books) using an unsupervised training method (predicting masked words in a sentence). During training, the model learns significant language understanding, which can later be fine-tuned for specific tasks using a process called **Transfer Learning**.

In Transfer Learning, the pre-trained model is applied to other datasets and tasks, leveraging the knowledge it has acquired during pre-training to improve performance on specialized NLP tasks such as sentiment analysis, question answering, and more.

This approach has led to substantial improvements in various NLP applications and has made Transformer-based models like BERT a powerful tool in the field.

****

There are many variations of the Transformer architecture, including **BERT**, **DistilBERT**, **BigBird**, **OpenGPT-3**, and other models that can be adapted and fine-tuned for specific tasks. These models have been designed to address various needs, such as improving efficiency, handling longer sequences, or generating more accurate predictions in language processing.

The **HuggingFace** library provides a repository for training and using these different Transformer architectures using **PyTorch**. It offers pre-trained models and tools that make it easy to fine-tune and deploy state-of-the-art NLP models on custom datasets. HuggingFace has become a popular framework in the NLP community, offering easy-to-use APIs for various Transformer-based models.

# 4. Attention in GPT (Generative Pretrained Transformer)

GPT is a model based on the Transformer architecture, but it only uses the **Decoder stack** of the Transformer. The attention mechanism in GPT is adapted to suit the autoregressive nature of text generation.

**Attention Mechanism:**

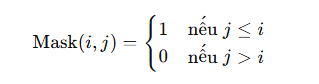
* **Masked Self-Attention**:
  + Similar to the **Self-Attention** mechanism in the Transformer, but with an added **mask** to prevent access to future tokens. This ensures that at each time step, the model only focuses on previous tokens (causal attention).
  + **Masked Self-Attention** helps GPT predict the next word in the sequence in a sequential manner.

**Details about Masked Multi-Head Attention in GPT:**

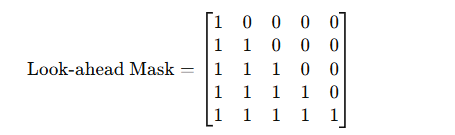
1. **Objective of Masked Attention**:
   * During text generation, at each time step, the model should not have access to future words (the words that will be generated later). This reflects the sequential nature of natural language generation and prevents the model from "cheating."
   * To achieve this, a **Look-ahead Mask** is used to hide the words to the right of each token.
2. **How the Look-ahead Mask Works**:
   * The **Look-ahead Mask** is a lower triangular matrix applied to the **Attention scores** in **Self-Attention**.
   * For an input sequence of length nnn, the Look-ahead Mask has the following form:

* The matrix ensures that each token only has access to itself and all previous tokens, preventing the model from peeking at future tokens during the prediction process.

This masked attention mechanism allows GPT to generate text in a causal, autoregressive manner, ensuring that the output at each time step depends only on the tokens generated so far, maintaining the integrity of sequential text generation.



Example vector from 5 (n=5):



This matrix is added to the **Attention scores** (after which the **Softmax** is applied), causing the masked tokens to not contribute to the Attention calculation.

1. **Masked Multi-Head Attention in GPT:**
   * GPT only uses the **Decoder stack** (unlike the full Transformer, which includes both Encoder and Decoder stacks).
   * In the Decoder of GPT, each layer consists of:
     + **Masked Multi-Head Attention**: Ensures that the model only focuses on previous tokens (or the current token).
     + **Feed-forward network**: Learns non-linear relationships from the input.
   * During training, instead of using the model's own output as input for the next step (as done during inference), the model is provided with the **ground truth output** as input at each step. This approach is known as **teacher forcing**, which helps stabilize and speed up the training process.

Teacher forcing enables faster convergence by directly guiding the model with the correct output at each time step, rather than allowing it to generate predictions from its own previous outputs.

# 5. Differences Between Attention Mechanisms

|  |  |  |  |
| --- | --- | --- | --- |
| Characteristic | Seq2Seq (LSTM) | Transformer | GPT |
| Type of Attention | Global Attention | Self-Attention and Multi-Head Attention | Masked Self-Attention |
| Direction of Attention | Encoder-Decoder | All tokens in the sequence | Only previous tokens (causal attention) |
| Computation Speed | Slower (depends on sequential processing of LSTM) | Faster due to parallel processing | Similar to Transformer, but with additional mask |
| Applications | Machine Translation, Text Summarization | General-purpose: Translation, Summarization, QA, etc. | Text Generation, Story Writing, Chatbots, etc. |
| Long-term Dependency Handling | Limited if sequence is too long | Better due to Self-Attention | Excellent for long-term dependencies in text generation |
| Order Dependency | Yes (Sequential LSTM processing) | No, uses Positional Encoding | Yes, but through mask and Positional Encoding |

# Referent

<https://pmc.ncbi.nlm.nih.gov/articles/PMC10836572/#abstract1>

<https://online-journals.org/index.php/i-joe/article/view/24499>