VIETNAM GENERAL CONFEDERATION OF LABOUR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

Lê Trần Nhật Quang - 521H0413

Nguyễn Gia Mỹ - 521H0272

Nguyễn Đình Việt Hoàng - 522H0120

FINAL REPORT
DEEP LEARNING

HO CHI MINH CITY, YEAR 2024

VIETNAM GENERAL CONFEDERATION OF LABOUR

TON DUC THANG UNIVERSITY

FACULTY OF INFORMATION TECHNOLOGY

Lê Trần Nhật Quang - 521H0413

Nguyễn Gia Mỹ - 521H0272

Nguyễn Đình Việt Hoàng - 522H0120

FINAL REPORT
DEEP LEARNING

Instructor

Mr. Lê Anh Cường

HO CHI MINH CITY, YEAR 2024

ACKNOWLEDGEMENT

We sincerely thank Mr. Lê Anh Cường for teaching us the Deep Learning course with great enthusiasm. We want to express our deep appreciation for the dedication and professional knowledge that you shared with us. Through your classes, we gained a better understanding of the fundamental aspects of the Deep Learning, thanks to your detailed explanations and practical applications. You helped us grasp the knowledge and apply it effectively. Finally, we extend our heartfelt gratitude to Mr. Lê Anh Cường for your commitment and invaluable support throughout our learning journey in this course. The skills and knowledge we acquired will continue to impact our future development. We sincerely thank you and wish your health, success, and happiness.

Ho Chi Minh City, April 30, 2024

Authors:

Lê Trần Nhật Quang

Nguyễn Gia Mỹ

Nguyễn Đình Việt Hoàng

THE COMPLETION REPORT HAS BEEN SUBMITTED AT TON DUC THANG UNIVERSITY

Our group assures that this is our own report and was guided by Mr. Lê Anh Cường. The research content and results in this report are honest and have not been published in any form before. The figures in the tables used for analysis, comments, and evaluations were collected by the authors from various sources clearly stated in the reference section.

Additionally, the report includes some comments, evaluations, and data from other authors and organizations, all of which are cited and noted for their origin.

If any fraud is detected, we fully take responsibility for the content of our final report for the second semester of the 2023-2024 academic year. Ton Duc Thang University is not involved in any copyright or intellectual property violations that we may cause during the process (if any).

Ho Chi Minh City, April 30, 2024

Authors:

Lê Trần Nhật Quang

Nguyễn Gia Mỹ

Nguyễn Đình Việt Hoàng

INSTRUCTOR RUBRIC

Supervisor’s Name: …………………………………………………………………..............

Comments: ……………………………………………………………………………………...

Total Score Based on Rubric Evaluation: ………………………………………………

Ho Chi Minh City, date … month … year …

Supervisor

(sign and write your full name)

TABLE OF CONTENTS

LIST OF FIGURES

PART 1: TRANSFORMER BASED ENCODER MODEL

1. Overview of the Transformer Model:

+ The Transformer serves as the foundation for other popular models such as Bidirectional Encoder Representations from Transformers (BERT), which is used to solve various language and speech processing tasks, including machine translation, language generation, classification, entity recognition, speech recognition, and text-to-speech conversion. Unlike RNN models, the Transformer does not process input elements in a sequential manner. Instead, it leverages parallel computation capabilities (i.e., multitasking) of GPUs to process elements non-sequentially in a natural language sentence. This significantly reduces the training time for the model and addresses the Gradient Vanishing problem caused by RNNs.

+ Instead of using the recurrent architecture like RNNs, the Transformer employs self-attention. In the self-attention mechanism, the Transformer consists of 6 encoders and 6 decoders. Each encoder contains 2 layers: self-attention and feed-forward neural networks (FNN).

+ Self-attention is a mechanism that allows the encoder to consider other words while encoding a particular word. This enables Transformers to understand the relationship between words in a sentence, even when they are far apart, unlike the case of Gradient Vanishing in RNNs. The decoders have a similar architecture, but they include an attention layer to focus on relevant parts of the input, such as determining the relationship between the two sides of a comma in a sentence, and then provide a suitable output for users in tasks like machine translation.

+ Self-attention consists of 4 steps:

1. Creating a set of 3 vectors from the input vectors of the encoder. In the first encoder, the input vector is the word embedding of the word. Therefore, for

each word, we have 3 vectors: Query, Key, and Value. These vectors are created by matrix multiplication between the input vector and 3 weight matrices corresponding to Query, Key, and Value, which are used during the training process. These 3 vectors play different roles and are all important for attention.

2. Calculating scores. For each word, we need to calculate the scores of other words in the sentence with respect to this word. These scores determine which words need to be attended to and how much attention should be given when encoding a word. The scores are calculated by taking the dot product between the Query vector of the current word and the Key vectors of the other words in the sentence. For example, when calculating self-attention for the word at position 1, its score with itself is q1.k1, its score with the second word is q1.k2, and so on.
3. Normalizing the scores. In the original paper, the scores are divided by 8 (the square root of 64, the dimension of the Key vector). This helps stabilize the gradients. Next, these values are passed through the softmax function to ensure that the score values are positive and their sum does not exceed 1.
4. Multiplying the Value vector by each calculated score and then summing them up. The purpose of this step is to preserve the value of the vectors for the words that need attention and eliminate the vectors for unrelated words (by multiplying them by a very small number, such as 0.001).

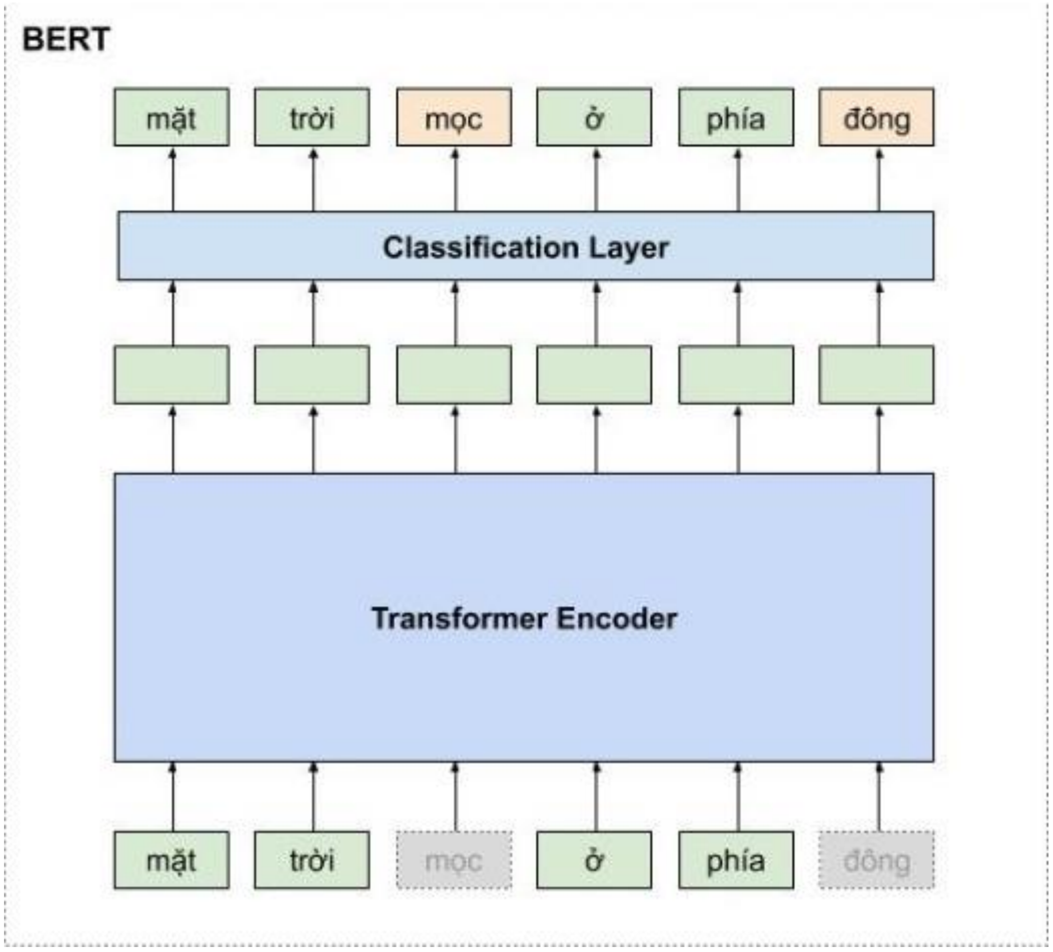


Figure 1.1. An example model of BERT

In summary, the overall architecture of the Transformer model consists of two main parts: the encoder and the decoder. The encoder is used to learn the representation vector of the sentence, with the expectation that this vector carries the perfect information of that sentence. The decoder converts the representation vector into the target language.

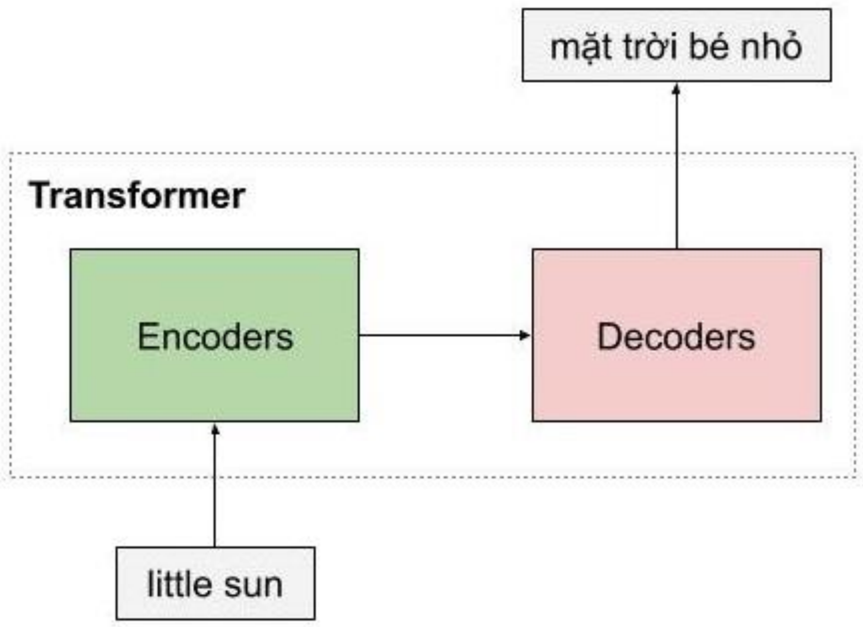


Figure 1.2. An example of Machine Translation using encoders and decoders

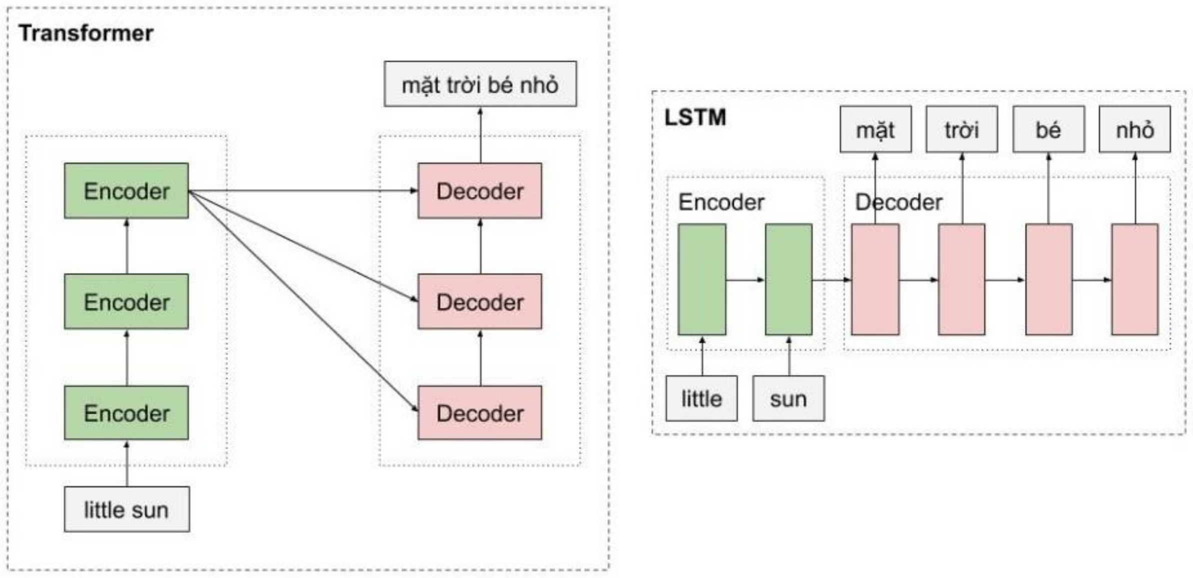


Figure 1.3. The comparison between Transformer model and LSTM model

As you can see in the figure 1.3, the encoders of the Transformer model are a type of feedforward neural nets, consisting of multiple encoder layers, each of which processes words simultaneously. In contrast, with the LSTM model, words must be processed sequentially.

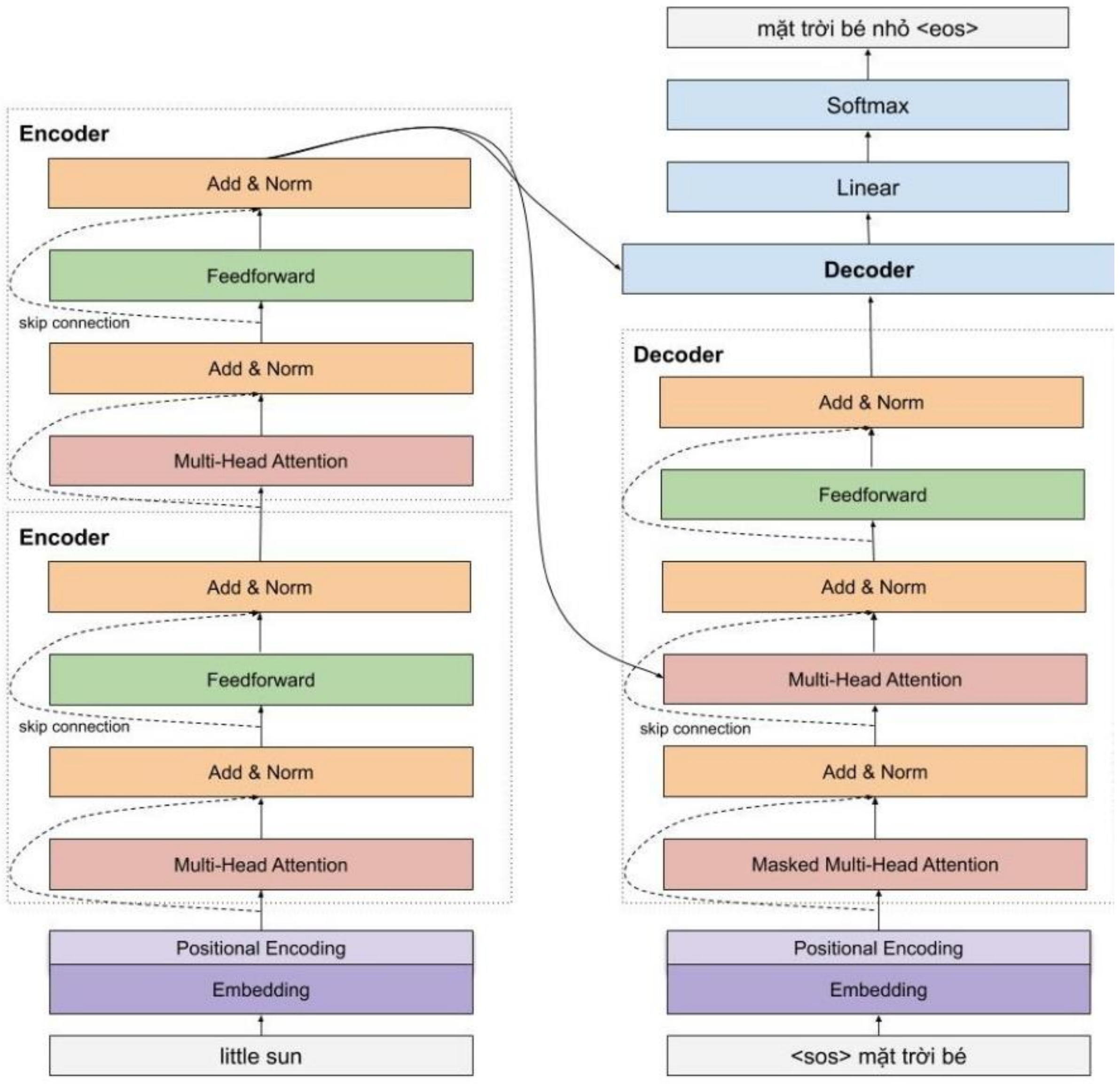


Figure 1.4. Translation translates a natural speech from encoder to decoder from English to Vietnamese

+ Position Encoding is used to incorporate information about the positions of words into the Transformer model.

+ The Sinusoidal Position Encoding method is used as follows:

- The position of each word is encoded using a vector of the same size as the word embedding, and it is directly added to the word embedding as shown in the following figure:

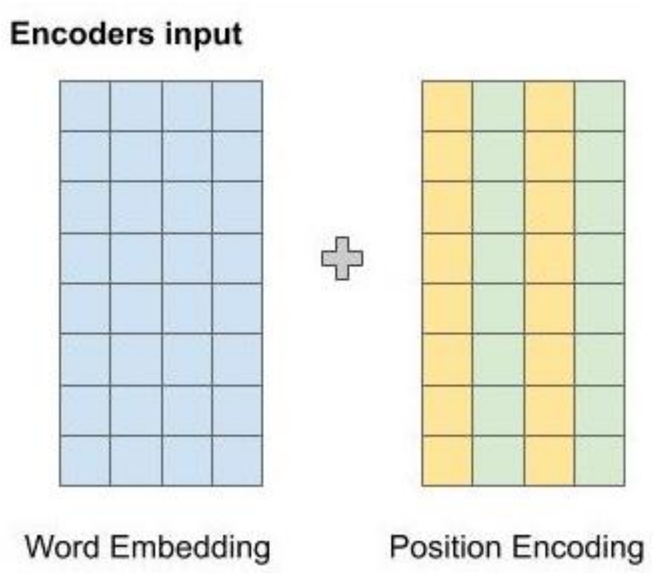


Figure 1.5. An example for encoders input

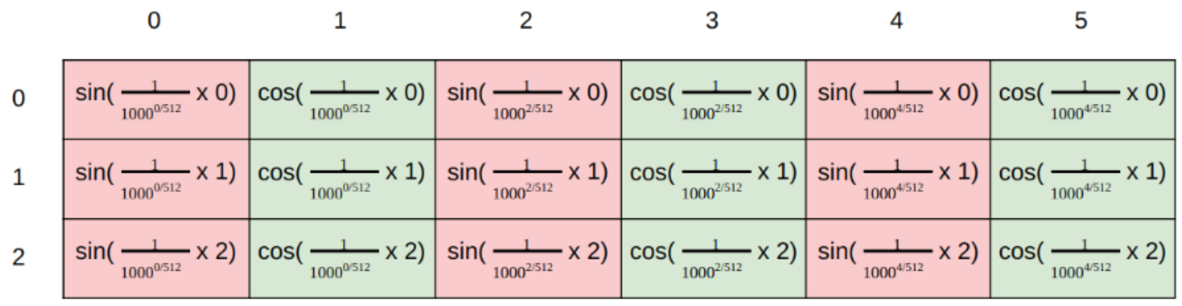
- For even positions in the above figure, the author uses the sine function, and for odd positions, the author uses the cosine function to calculate the value at that dimension.
- The general formula is:

$$p_t^i = f(t)^i = \begin{cases} sin(w_k * t) & \text{if } i = 2k \\ cos(w_k * t) & \text{if } i = 2k+1 \end{cases}$$

Where:

$$w_k = \frac{1}{10000^{2k/d}}$$

+ The figure below illustrates how the author calculates the position encoding. Suppose we have word embeddings of 6 dimensions, so the position encoding also has 6 dimensions. Each row corresponds to a word. The values of the vectors at each position are calculated using the formula in the figure below:



	0	1	2	3	4	5
0	$\sin(\frac{1}{1000^{0/512}} \times 0)$	$\cos(\frac{1}{1000^{0/512}} \times 0)$	$\sin(\frac{1}{1000^{2/512}} \times 0)$	$\cos(\frac{1}{1000^{2/512}} \times 0)$	$\sin(\frac{1}{1000^{4/512}} \times 0)$	$\cos(\frac{1}{1000^{4/512}} \times 0)$
1	$\sin(\frac{1}{1000^{0/512}} \times 1)$	$\cos(\frac{1}{1000^{0/512}} \times 1)$	$\sin(\frac{1}{1000^{2/512}} \times 1)$	$\cos(\frac{1}{1000^{2/512}} \times 1)$	$\sin(\frac{1}{1000^{4/512}} \times 1)$	$\cos(\frac{1}{1000^{4/512}} \times 1)$
2	$\sin(\frac{1}{1000^{0/512}} \times 2)$	$\cos(\frac{1}{1000^{0/512}} \times 2)$	$\sin(\frac{1}{1000^{2/512}} \times 2)$	$\cos(\frac{1}{1000^{2/512}} \times 2)$	$\sin(\frac{1}{1000^{4/512}} \times 2)$	$\cos(\frac{1}{1000^{4/512}} \times 2)$

Figure 1.6. Formula for calculating the value of vectors

+ In this context, many people may wonder why the proposed position representation can encode the positional information of words.

+ Let me provide an example to illustrate this: Imagine you have numbers from 0 to 15. You will notice that the rightmost bit changes fastest for each number, followed by the second rightmost bit, and so on for the other bits.

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Figure 1.7. The illustration shows changing the bit in the example above

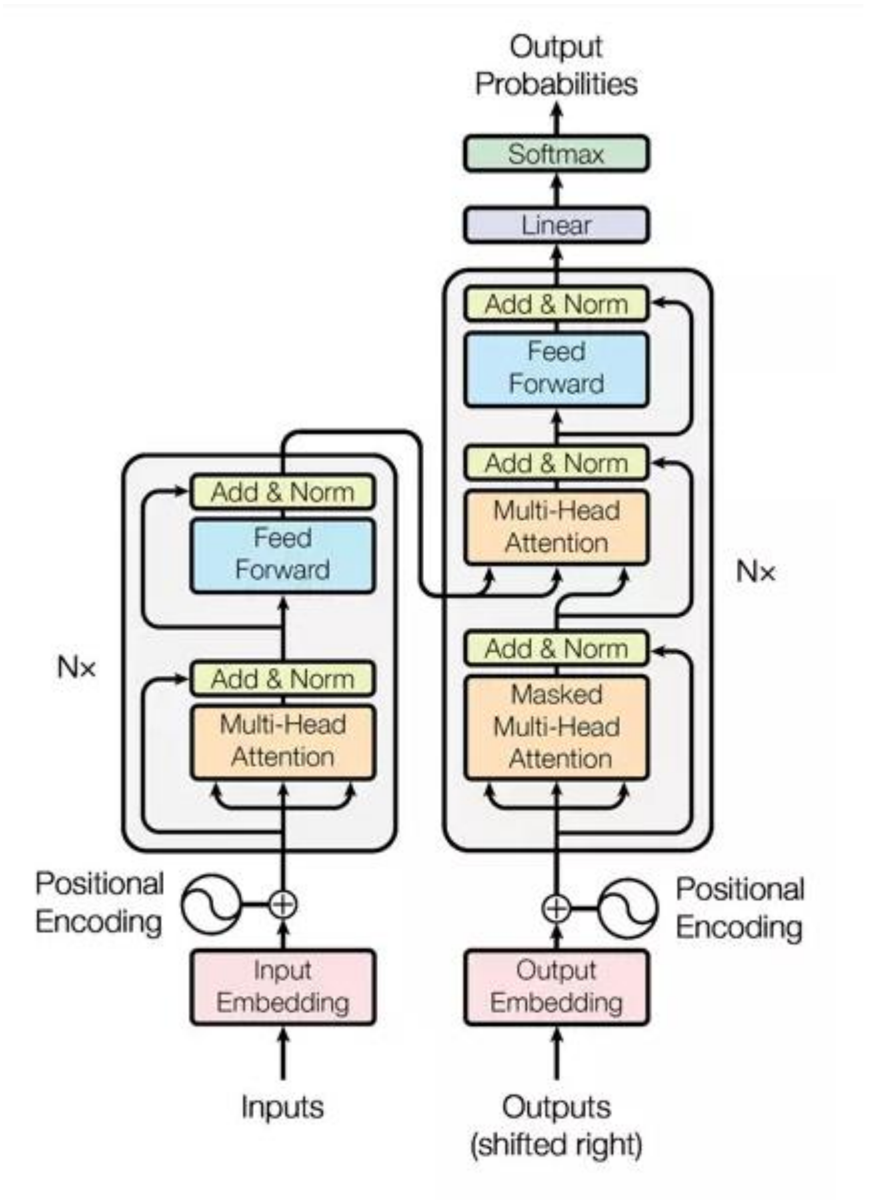


Figure 1.8. Architecture of the Transformer

+ Explanation for Figure 1.8:

- Positional encoding: Since the Transformer does not have recurrent or convolutional layers, it does not inherently know the order of input tokens. Therefore, there needs to be a way for the model to know this information, which is the task of positional encoding. After the embedding layers, which produce token embeddings, we add positional encoding vectors representing the position of each word in the sentence.

- Normalization Layer: In the diagram's architecture, the "Add & Norm" layer refers to the normalization layer. This layer simply normalizes the output of the multi-head attention, improving convergence efficiency.
- Residual Connection: The residual connection is a simple concept of adding the input of a block to its output. This connection allows stacking multiple layers in the network. In the diagram, the residual connection is used after the FFN (Feed-Forward Network) and attention blocks. In the "Add" part of "Add & Norm," it represents the residual connection.
- Feed-Forward Block: This is a basic block where, after performing computations in the attention block at each layer, the next block is the FFN. You can understand that the attention mechanism helps gather information from the input tokens, and the FFN processes that information.

2. Explanation of the Encoder:

+ The Transformer encoder can consist of multiple identical encoder layers. Each encoder layer of the Transformer comprises two main components: multi-head attention and a feedforward network. It also includes skip connections and normalization layers.

+ Thanks to the multi-head attention component, there is a difference between the LSTM model and the Transformer model.

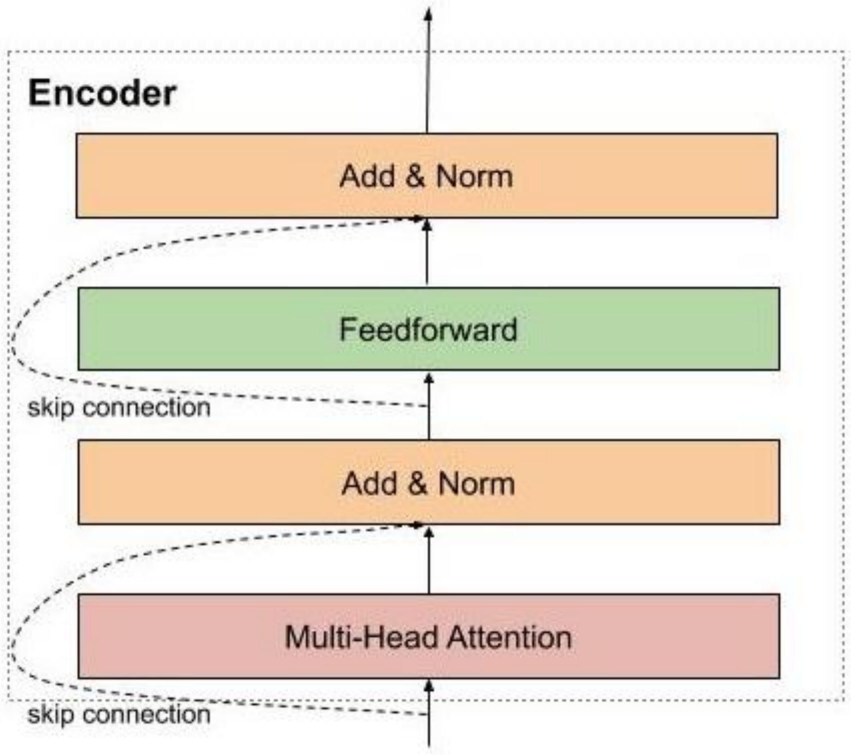


Figure 2.1. An example of encoder

+ The first encoder receives the matrix representation of the words, which has been augmented with positional information through positional encoding. Then, this matrix is processed by the multi-head attention. Multi-head attention is essentially self-attention, but to allow the model to pay attention to different patterns, multiple self-attention mechanisms are used.

3. Explanation of Self-Attention Layer:

+ Self-attention allows the model to use information from related words when encoding a word.

+ For example:

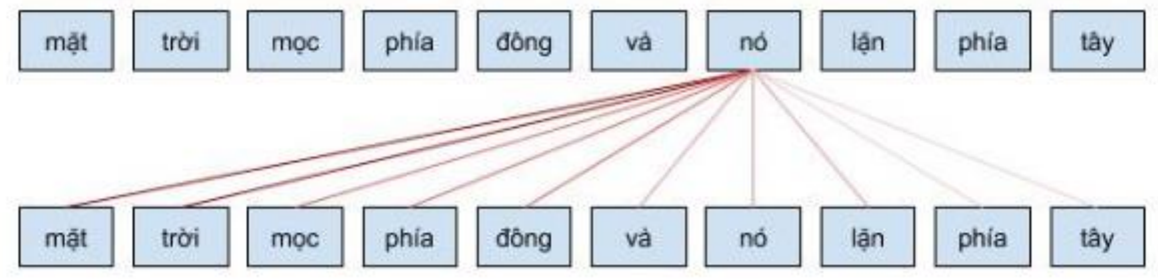


Figure 3.1. An example of using Self-Attention Layer

+ In this example, when the word "nó" is encoded, it pays attention to related words like "mặt trời".

+ We can imagine the self-attention mechanism as similar to Google's search mechanism.

+ First, for each word, we need to create three vectors: a query vector, a key vector, and a value vector, by multiplying the input matrix representing the words with the corresponding learned weight matrices.

- Query vector: a vector that contains the information of the word to be searched or compared. It is similar to the query in a Google search.
- Key vector: a vector that represents the information of the words to be compared with the queried word. For example, it can be seen as the web pages that Google compares to the search keyword.
- Value vector: a vector that represents the content or meaning of the words. You can think of it as the content displayed to the user after searching, like the content of a web page.

+ The process of calculating the attention vector can be summarized in the following three steps:

- Step 1: Compute the query, key, and value matrices by initializing weight matrices for query, key, and value. Then multiply the input with these weight matrices to obtain the corresponding matrices.
- Step 2: Calculate the attention weights. Multiply the key and query matrices obtained in the previous step to compare the query with the key and learn their correlation. Then normalize the weights to the range [0, 1] using the softmax function. A weight of 1 means the query is similar to the key, and a weight of 0 means they are dissimilar.
- Step 3: Calculate the output. Multiply the attention weights with the value matrix. This means that we represent a word by the weighted average (attention weights) of the value matrix.

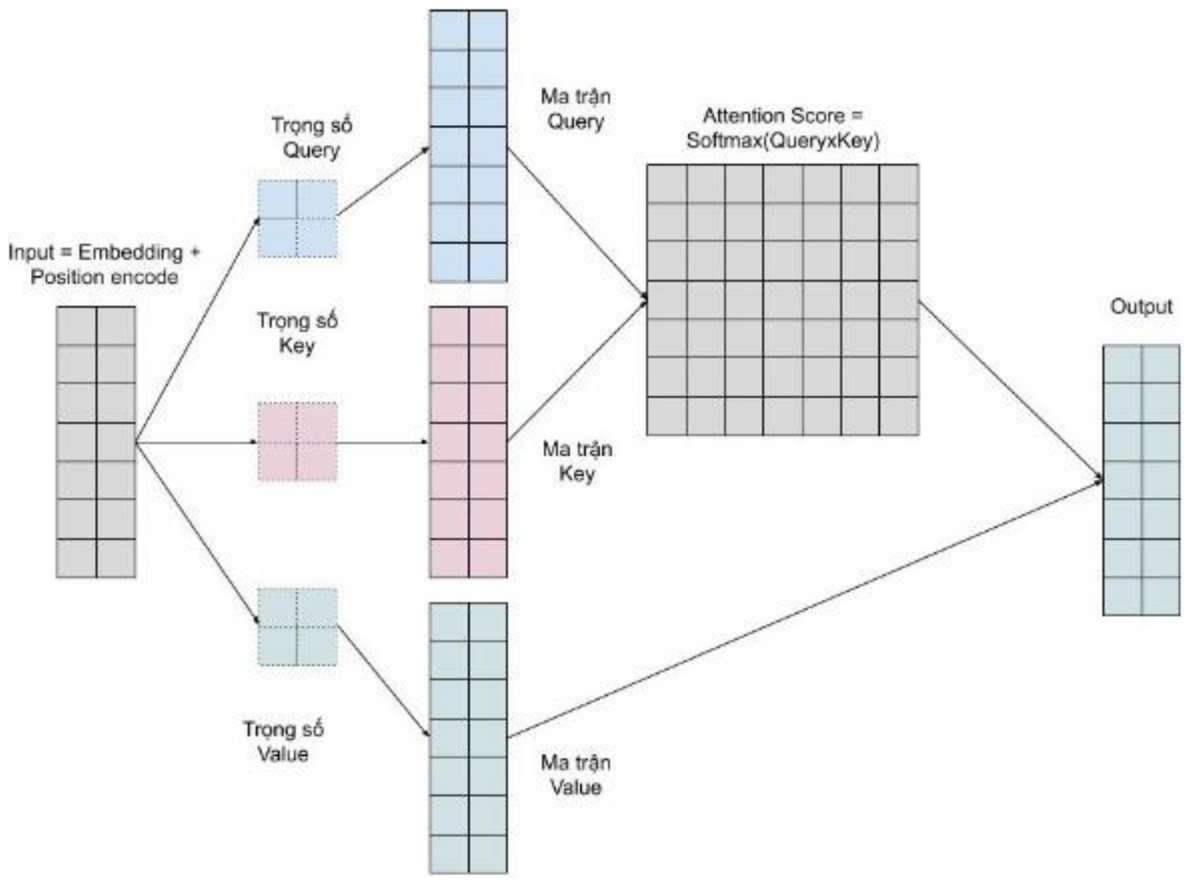


Figure 3.2. Attention vector calculation process

4. Explanation of Multi-Head Attention:

+ We want the model to learn different types of relationships between words. With each self-attention mechanism, we learn one pattern. To expand this capability, we simply add multiple self-attention mechanisms. This means that we need multiple query, key, and value matrices. Now the key, query, and value weight matrices have an additional depth dimension.

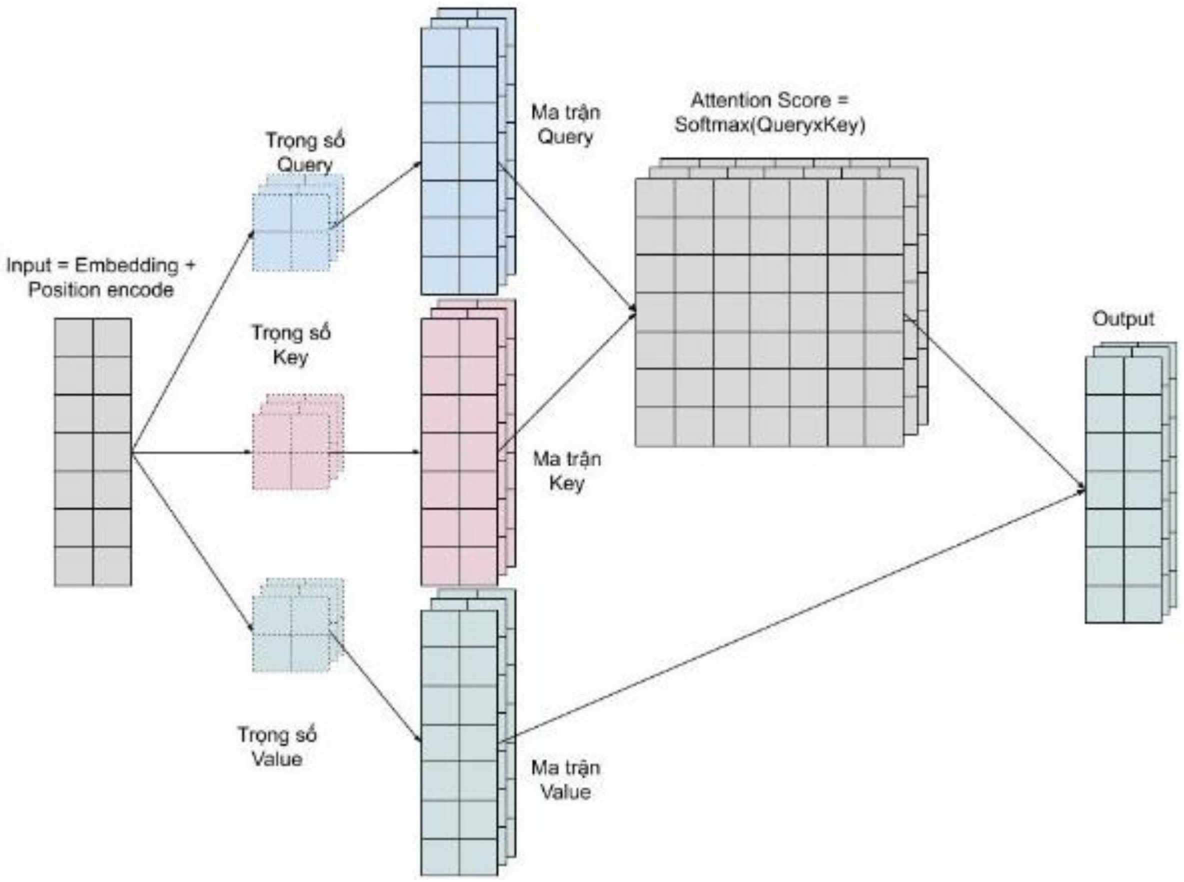


Figure 4.1. An example of Multi Head Attention

+ Multi-head attention allows the model to simultaneously attend to different patterns, such as:

- Attending to the previous word of a given word.
- Attending to the next word of a given word.
- Attending to related words of a given word.

5. Explanation of Feed-Forward Neural Networks:

+ After computing self-attention and obtaining the input representations, the model utilizes a feed-forward neural network (FFNN) layer to further process these representations. This layer applies nonlinear transformations to generate the final output representations of the encoder.

6. Residual Connections and Normalization Layers:

+ In the Transformer architecture, residual connections and normalization layers are used extensively, as they are beneficial for the model's training convergence and help prevent information loss during training. For example, they help preserve the positional information encoded in the word positions.

7. Continue pretrain and fine-tuning:

+ Common approaches to pretraining and fine-tuning Transformer-based Encoder models:

1/ Pretraining:

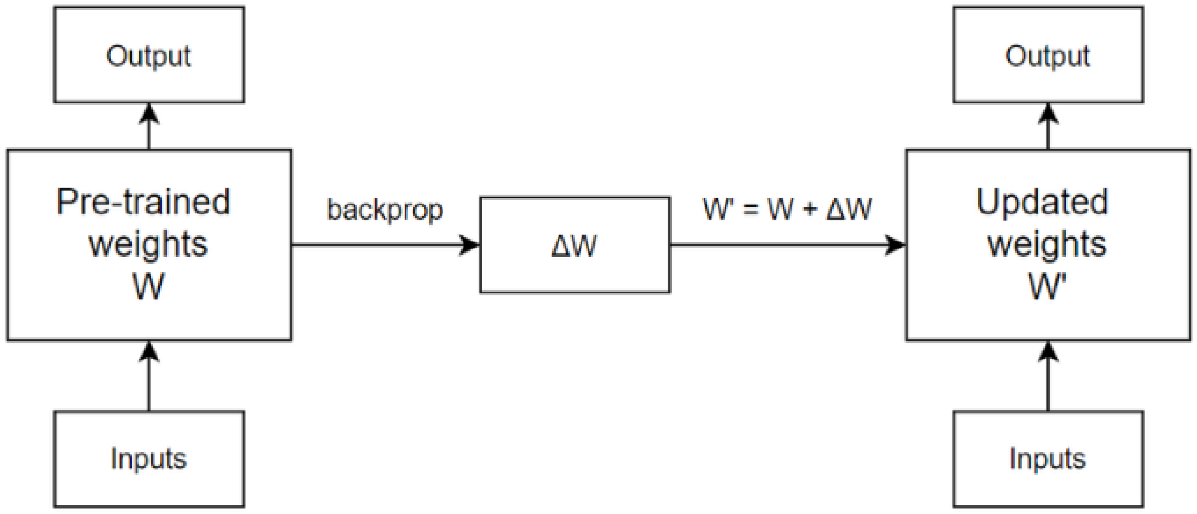
+ Unsupervised Pretraining: The Transformer-based Encoder model is pretrained on a large amount of unlabeled natural language data. This process often involves using an auxiliary task such as predicting the next word in a sentence or predicting masked words to guide the learning process.

2/ Find-tuning:

+ Full Model Fine-tuning: After pretraining, the Transformer-based Encoder model can be fine-tuned on a specific task. In this process, a small amount of labeled data is used to adjust the model's weights. The model's output is compared to the ground truth labels, and a loss function such as Cross-Entropy is used to compute the error and update the network weights using backpropagation.

+ Fine-tuning the Last Layer: Instead of fine-tuning the entire model, another approach is to only fine-tune the last layer of the Transformer-based Encoder model. Typically, a custom output layer is added to the model to predict the task-specific outcomes, and only the weights of this layer are updated during fine-tuning. The weights of other layers in the model are kept frozen.

+ Transfer Learning: Another fine-tuning approach is to use a pretrained Transformer-based Encoder model for one task and reuse it for another related task. The weights of the model are adjusted only for the corresponding output part of the new task, while other parts of the model are kept unchanged.



Hình 3. Quá trình cập nhật một layer của model trong fine-tuning

Figure 7.1. The process of updating one layer of the model in fine-tuning

Note for the figure 7.1: The pretrained weights W of the model will be transformed into updated weights W' based on the weight changes ΔW obtained from the backpropagation process. In the next iteration, W' is updated again with a different ΔW.

PART 2: BERT (Bidirectional Encoder Representations from Transformers):

1. Introduction:

+ BERT is a transformer-based encoder model that was introduced by Google in 2018. The BERT model was proposed in BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding by Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova.

+ BERT is a method of **pre-training language representations**, meaning that we train a general-purpose "language understanding" model on a large text corpus (like Wikipedia), and then use that model for downstream NLP tasks that we care about (like text classification, named entity recognition, and question-answering).
+ BERT outperforms previous methods because it is the first **unsupervised**, deeply bidirectional system for pre-training NLP.
+ **Unsupervised** means that BERT was trained using only a plain text corpus, which is important because an enormous amount of plain text data is publicly available on the web in many languages.
+ BERT is a model with absolute position embeddings so it's usually advised to pad the inputs on the right rather than the left.
+ In order to learn relationships between sentences, we also train on a simple task which can be generated from any monolingual corpus: Given two sentences A and B, is B the actual next sentence that comes after A, or just a random sentence from the corpus?
+ The model must predict the original sentence, but has a second objective: inputs are two sentences A and B (with a separation token in between). With probability 50%, the sentences are consecutive in the corpus, in the remaining 50% they are not related. The model has to predict if the sentences are consecutive or not.

2. Using BERT Base model uncased:

+ Uncased means that the text has been lowercased before WordPiece tokenization. Uncased also strips out any accent markers.

e.g., John Smith becomes john smith.

+ Cased means that the true case and accent markers are preserved.
Typically, the Uncased model is better unless you know that case information is important for your task
+ Architecture: 12-layer, 768-hidden, 12-heads, 110M parameters
1/ Transformer Encoder: The core of BERT is a stack of transformer encoder layers. The base model typically consists of 12 transformer layers, although there are variations with different numbers of layers.
+ Each encoder layer has two sub-layers: multi-head self-attention and feed-forward neural networks.

a. Multi-Head Self-Attention: This layer allows each token to attend to other tokens in the input sequence to learn contextual representations. It employs multiple attention heads to capture different types of dependencies.
b. Feed-Forward Neural Networks: After self-attention, a simple feed-forward neural network is applied to each token representation independently.

2/ Input Embeddings: Each token is then represented by three types of embeddings:

a. Token Embeddings: Each token is transformed into a fixed-size vector representation.

b. Segment Embeddings: BERT can process multiple sentences in a single sequence, so it assigns a unique segment ID to each token to distinguish between different sentences.
c. Position Embeddings: Since BERT is a transformer model that doesn't inherently consider the order of tokens, position embeddings are added to capture the positional information of each token.

3/ Hidden Layers and Hidden Size: The hidden layers and hidden size in BERT base uncased refer to the dimensions of the hidden representations within the model.

- In the base model, the hidden size is typically 768, meaning each token representation has a vector of size 768.
- The hidden layers correspond to the stacked transformer encoder layers.

4/ Attention Heads and Attention Size: BERT uses multi-head self-attention, which means each attention mechanism is split into multiple attention heads.

- In the base model, the number of attention heads is also set to 12, aligning with the number of transformer layers.
- The attention size, in this case, would be the hidden size divided by the number of attention heads (768/12 = 64).

5/ Pre-training and Fine-tuning: The BERT base model uncased is pre-trained on a large corpus of unlabeled text using tasks such as Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). After pre-training, the model can be fine-tuned on specific downstream tasks by adding an additional task-specific layer and training it on labeled data.

PART 3: GPT MODEL

1. A brief introduction of GPT and GPT models:

GPT stands for:

- Generative (Generative AI): a technology capable of producing content, such as text and imagery.
- Pre-trained: Pre-trained models are saved networks that have already been taught to resolve a problem or accomplish a specific task using a large dataset.
- Transformer: a deep learning architecture that transforms an input into another type of output.

⇨ GPT is a generative AI technology that has been previously trained to transform its input into a different type of output.

+ GPT was developed by OpenAI in 2018. Since then, OpenAI has officially released some iterations of the GPT model:

- GPT-1 (2018): contained 117 million parameters and was pre-trained on a large text data using an unsupervised learning technique to predict the next word in a sentence.
- GPT-2 (2019): with 1.5 billion parameters, this model can generate longer and more coherent text.
- GPT-3 (2020): contained 175 billion parameters and was trained on an enormous corpus of text data (books, articles, web pages). This model can generate text that is virtually indistinguishable from human-written content.
- GPT-3.5 Turbo (2022): ChatGPT - based on GPT-3.5 Turbo Model, which is a chatbot app (or conversation AI model) that can engage in human-like conversations with users.
- GPT- 4 (2023): this model was pre trained to predict the next token using both public data and "data licensed from third-party providers, and was then fine-tuned with reinforcement learning from human (RLFH) and AI feedback. Unlike GPT-3.5 Turbo, GPT-4 can take images as well as text as input.

2. GPT's model Architecture

+ The GPT model's architecture is based on the transformer model. The Transformer model uses self-attention mechanisms to process input sequences of variable length, making it well-suited for NLP tasks. GPT simplifies the architecture by substituting encoder-decoder blocks with decoder blocks.

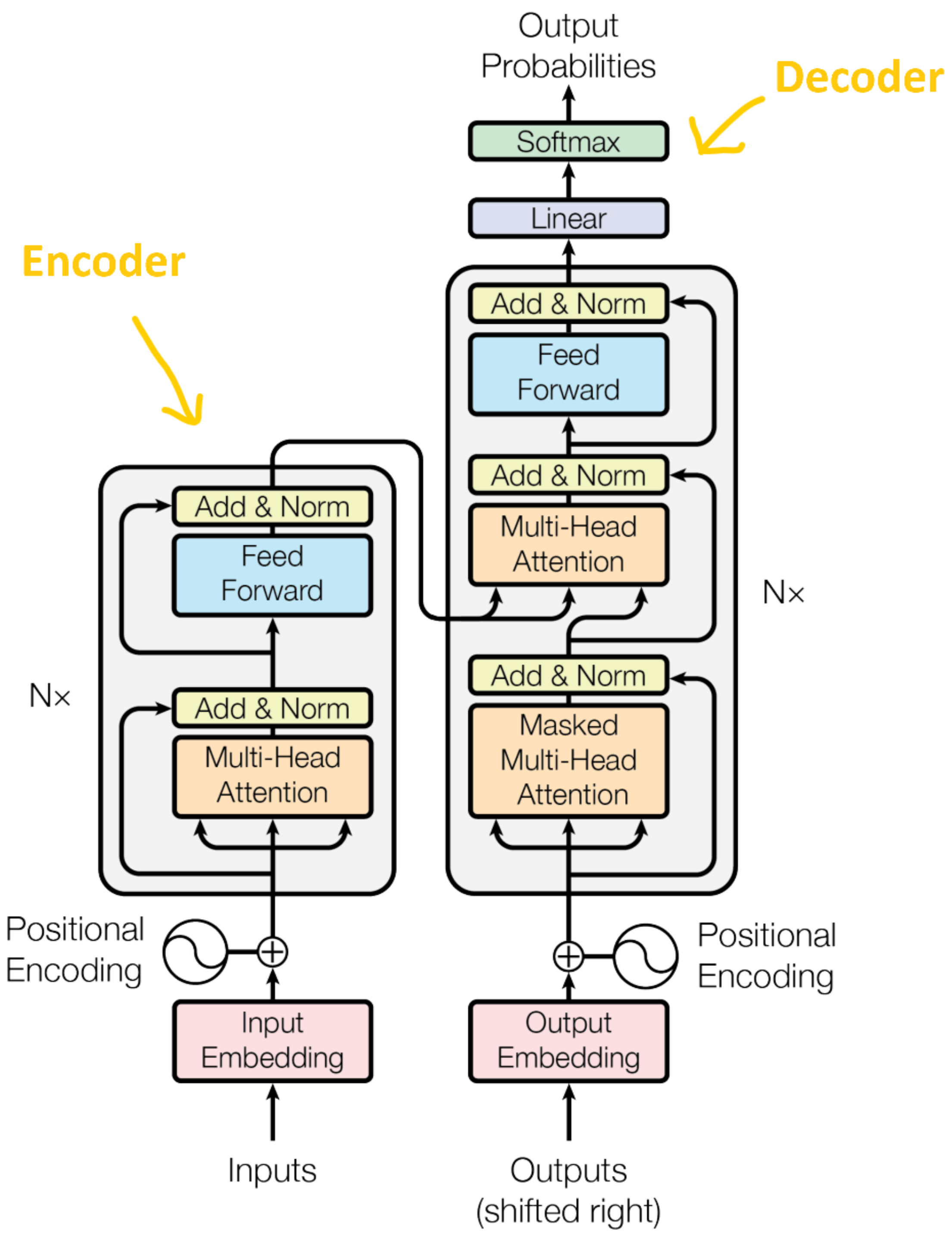


Figure 3.2.1. Transformer Model

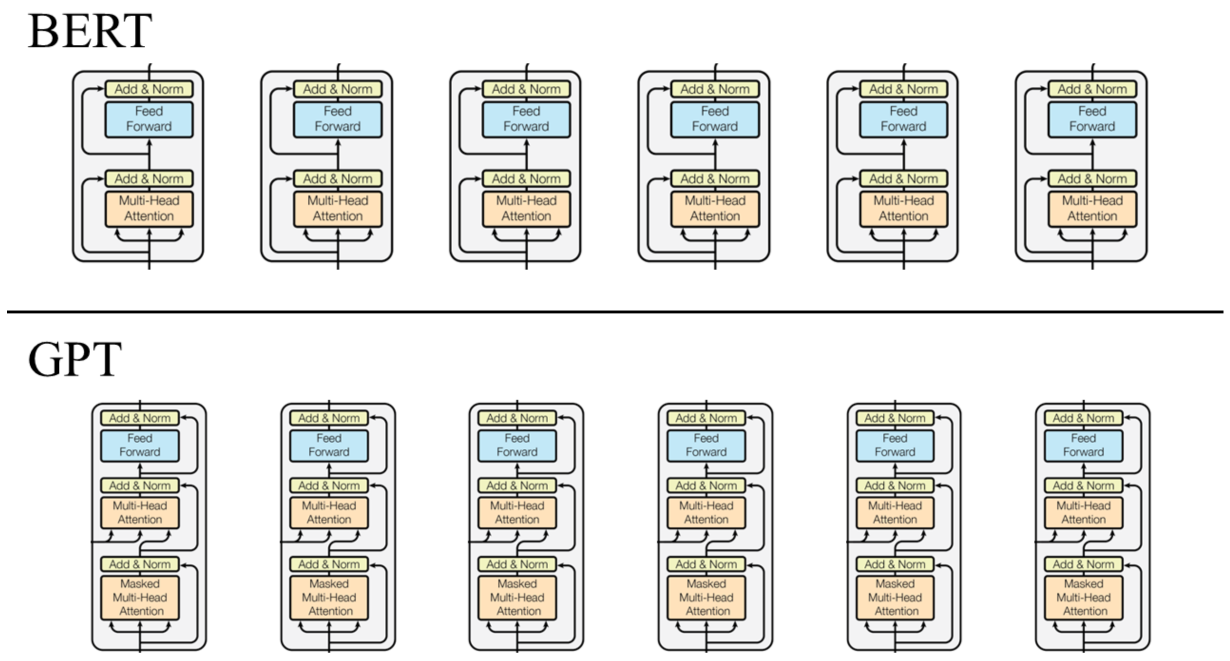


Figure 3.2.2. BERT vs GPT

+ The components of GPT architecture:

- Input Embedding layer: maps the input tokens to continuous vector representation, which can be processed by the transformer blocks.
- Positional encoding: the transformer blocks do not have any notion of order or position, positional encoding is added to the input embeddings to provide information about the relative position of tokens.
    - Masking: In some cases, masking may be necessary to mask certain input tokens (e.g., in language modeling tasks, the model should only use tokens that come before the target token).
    - Transformer blocks
- Linear and Softmax functions: the softmax function is applied to the output of the final layer of the model and commonly used for classification tasks. It generates a probability distribution over a set of output classes. The output of

the final layer is specifically converted into a set of logits before being normalized with the softmax function.

  - The normalized values obtained from the model can be interpreted as the likelihood or probability that a particular input belongs to each of the output classes.
  - The query, key, and value vectors for each token in the input sequence are frequently calculated using linear functions in the attention mechanism.
  - The output of the multi-head attention layer is transformed using them in the feedforward layers.
  - The output layer also employs linear functions to forecast the following token in the sequence.

- Pretraining: key component of GPT architecture. The model is trained on a large amount of data in an unsupervised manner.
- Fine-tuning: the process of adapting a pre-trained neural network model to a new task or dataset by further training the model on that task or dataset. (Fine-tuning in GPT involves adjusting the parameters of the pre-trained model to optimize performance on a specific downstream task)
- Language modeling: key task of GPT architecture, language modeling task is performed during the pre-training phase of the model. It is the task of predicting the next word in sequence based on the previous words. It allows the model to learn relationships between the words and their meaning in the training data.
- Unsupervised learning: GPT models use unsupervised learning in the pre-training phase to understand the relationships between the words and their context in the training data.

REFERENCES

1. Jessica Schulze.(2024) What is GPT? GPT-3, GPT-4 and More Explained. What Is GPT? GPT-3, GPT-4, and More Explained
2. **Adam VanBuskirk. (2023). A Brief History of The Generative Pre-trained Transformer (GPT) Language Models. A Brief History of The Generative Pre-trained Transformer (GPT) Language Models**
3. **2305.10435 (arxiv.org)**
4. **Fine-tuning một cách hiệu quả và thân thiện với phần cứng: Adapters và LoRA (viblo.asia)**
5. **Từng mô hình GPT của OpenAI được giải thích và so sánh - TechTimes.vn**
6. **[Từ Transformer Đến Language Model] Bài 2: Kiến trúc và phương pháp Generative-Pretraining của GPT model (viblo.asia)**
7. **Khoa học dữ liệu (phamdinhkhanh.github.io)**
8. **Tìm hiểu về kiến trúc Transformer (viblo.asia)**
9. **Transformer Neural Network - Mô hình học máy biến đổi thế giới NLP - VinBigdata - Blog**
10. **Fine tuning pre-trained model trong pytorch và áp dụng vào Visual Saliency Prediction (viblo.asia)**