**CHAPTER 1: Mô hình Transformer based Encoder**

1. Giới thiệu tổng quát về Transformer model:

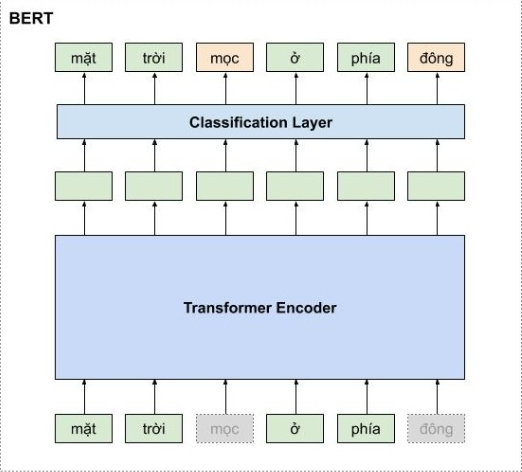
+ Transformer là nền tảng cho các mô hình nổi tiếng khác như Bidirectional Encoder Representations from Transformer (hay còn gọi là BERT), được sử dụng để giải quyết nhiều bài toán trong xử lý ngôn ngữ và tiếng nói, ví dụ như bài toán dịch tự động, bài toán sinh ngôn ngữ, phân loại, nhận dạng thực thể, nhận dạng tiếng nói, chuyển văn bản thành tiếng nói. Tuy nhiên, khác với mô hình RNNs, Transformer không xử lý các phần tử đầu vào trong một chuỗi một cách tuần tự. Transformer sẽ sử dụng tính năng tính toán song song nhiều bài toán cùng lúc (tức là đa nhiệm) của GPU để xử lý các phần tử một cách không tuần tự trong một câu ngôn ngữ tự nhiên. Điều này giúp giảm đáng kể thời gian cho việc train model cũng như giải quyết vấn đề Gradient Vanishing do RNNs gây ra.

+ Không sử dụng kiến trúc Recurrent (hồi quy) như RNNs mà Transformer sử dụng self-attention. Trong kiến trúc self-attention, Transformer chứa 6 encoder và 6 decoder. Mỗi encoder chứa 2 lớp: Self-attention và mạng truyền thẳng (FNN)

+ Self-Attention là cơ chế giúp encoder xem xét các từ khác trong lúc mã hóa một từ cụ thể, vì vậy, Transformers có thể hiểu được sự liên quan giữa các từ trong một câu, kể cả khi chúng có khoảng cách xa không giống như trường hợp Gradient Vanishing của RNNs. Các decoder cũng có kiến trúc giống như vậy nhưng giữa chúng có một lớp attention để nó có thể tập trung các phần liên quan của đầu vào, chẳng hạn như có dấu phẩy ở chính giữa câu thì decoder sẽ xem xét nghĩa giữa 2 vế trong câu đó có liên quan với nhau hay không rồi trả về đầu ra một cách thích đáng cho người dùng trong việc mảng Machine Translation (dịch máy).

+ Self-Attention có 4 bước:

1. Tạo ra bộ 3 vector từ các vector input của encoder. Tại encoder đầu tiên, vector input là word embedding của từ. Như vậy với mỗi từ, ta sẽ có 3 vector Query, Key và Value. Các vector này được tạo nên bởi phép nhân ma trận giữa vector đầu vào và 3 ma trận trọng số tương ứng với Query, Key và Value mà chúng ta sử dụng trong quá trình huấn luyện. 3 vector này đóng vai trò khác nhau và đều quan trọng đối với attention.
2. Tính điểm. Với mỗi từ, ta cần tính điểm của các từ khác trong câu đối với từ này. Giá trị này giúp quyết định từ nào cần được chú ý và chú ý bao nhiêu khi mã hóa một từ. Điểm được tính bằng tích vô hướng giữa véctơ Query của từ đang xét với lần lượt các vectơ Key của các từ trong câu. Ví dụ, khi ta tính self-attention trên từ có vị trí 1, điểm của nó với chính nó là q1.k1, điểm của nó với từ thứ hai là q1.k2, v..v..
3. Chuẩn hóa điểm. Trong bài báo gốc, điểm được chia cho 8 (căn bậc 2 của 64 – số chiều của vectơ Key). Điều này giúp cho độ dốc trở nên ổn định hơn. Tiếp theo, giá trị này được truyền qua hàm softmax để đảm bảo các giá trị điểm đều dương và có tổng không vượt quá 1.
4. Nhân vectơ Value với mỗi giá trị điểm đã tính phía trên rồi cộng lại với nhau. Ý đồ của việc này là bảo toàn giá trị vectơ của các từ cần được chú ý và loại bỏ vectơ của các từ không liên quan (bằng cách nhân nó với một số rất nhỏ, ví dụ như 0.001).

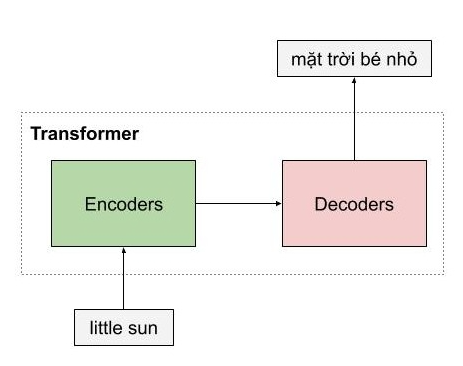


*Figure 1.1. An example model of BERT*

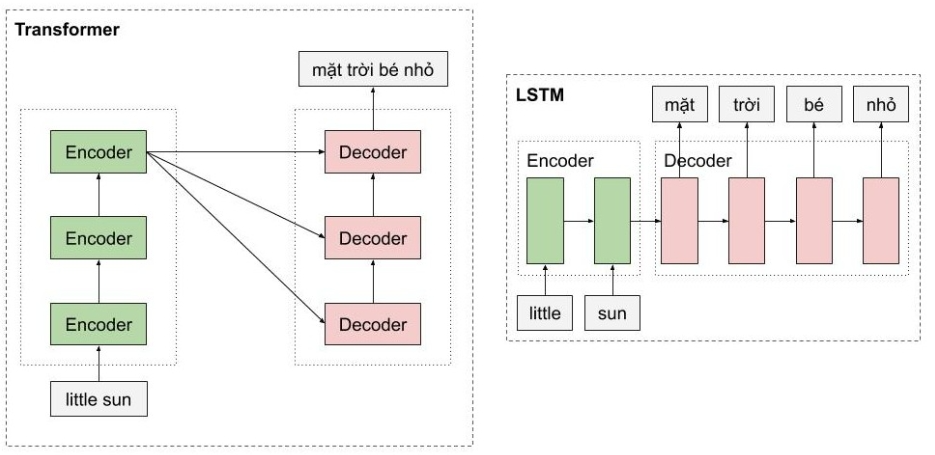
Cấu trúc: Mô hình có các lớp “Transformer Encoder” để mã hóa từ và “Classification Layer” để phân loại hoặc giải thích dữ liệu đầu vào.

Tóm gọn lại là: kiến trúc tổng quan của mô hình transformer bao gồm 2 phần lớn là encoder và decoder. Encoder dùng để học vector biểu diễn của câu với mong muốn rằng vector này mang thông tin hoàn hảo của câu đó. Decoder thực hiện chức năng chuyển vector biểu diễn kia thành ngôn ngữ đích.

Ví dụ:

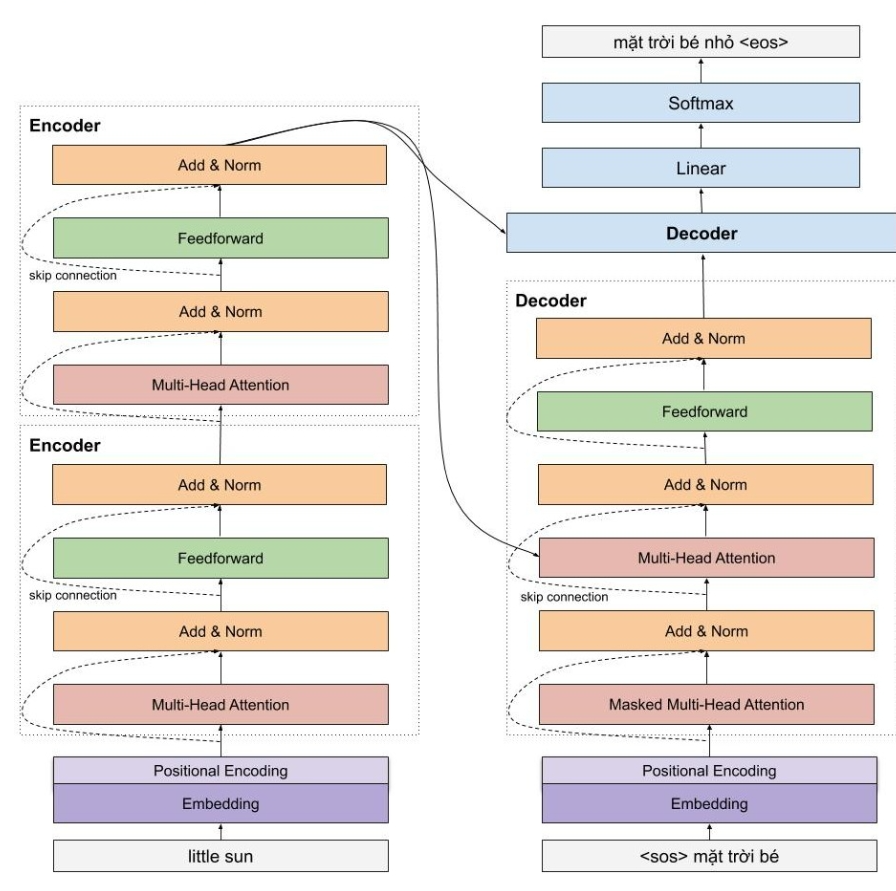


*Figure 1.2. An example of Machine Translation using encoders and decoders*



*Figure 1.3. The comparison between Transformer model and LSTM model*

Như các bạn thấy trong hình trên thì Encoders của mô hình transformer là một dạng feedforward neural nets, bao gồm nhiều encoder layer khác, mỗi encoder layer này xử lý đồng thời các từ. Trong khi đó, với mô hình LSTM, thì các từ phải được xử lý tuần tự.

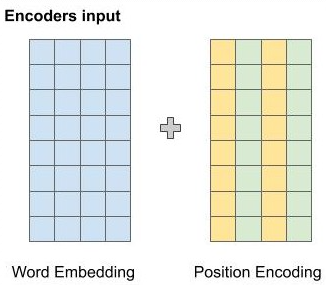


*Figure 1.4. Translation translates a natural speech from encoder to decoder from English to Vietnamese*

+ Position Encoding dùng để đưa thông tin về vị trí của các từ vào mô hình transformer.

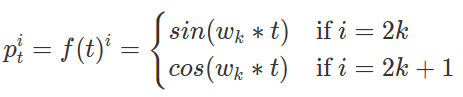
+ Sử dụng phương pháp Sinusoidal Position Encoding như sau:

* Vị trí của các từ được mã hóa bằng một vector có kích thước bằng word embedding và được cộng trực tiếp vào word embedding trong hình sau:

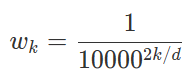


*Figure 1.5. An example for encoders input*

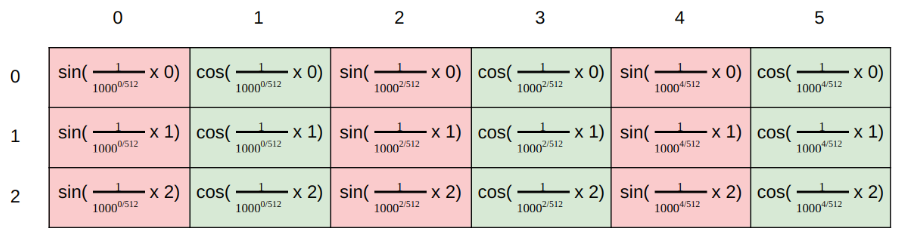
* Tại vị trí chẵn trong hình trên thì tác giả sử dụng hàm sin và với vị trí lẻ tác giả sử dụng hàm cos để tính giá trị tại chiều đó.
* Công thức tổng quát:



Trong đó:



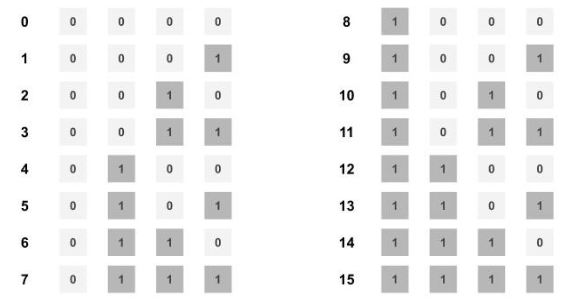
Trong hình dưới đây sẽ minh họa cho cách tính position encoding của tác giả. Giả sử chúng ta có word embedding có 6 chiều thì position encoding cũng có tương ứng là 6 chiều. Mỗi dòng tương ứng với một từ. Giá trị của các vector tại mỗi vị trí được tính toán theo công thức ở hình dưới:



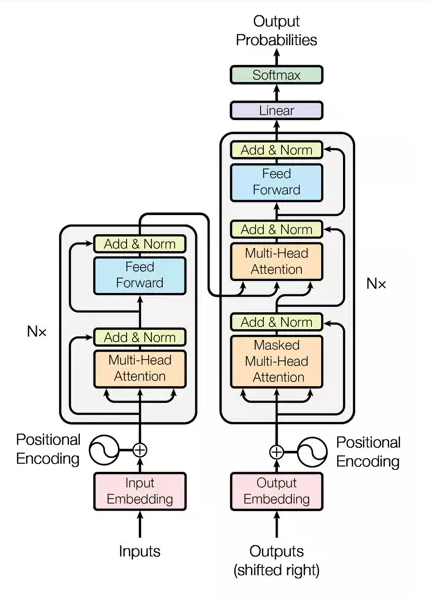
*Figure 1.6. Formula for calculating the value of vectors*

Thì ở chỗ này sẽ có nhiều người thắc mắc là tại sao với cách biểu diễn vị trí như tác giả đề xuất lại có thể mã hóa thông tin vị trí của từ?

Thì mình sẽ đưa ra ví dụ như sau: Hãy tưởng tượng bạn có các số từ 0-15 thì bạn sẽ thấy rằng bit ngoài cùng bên phải thay đổi nhanh nhất mỗi 1 số, và sau đó là bit bên phải thứ 2, thay đổi mỗi 2 số, tương tự cho các bit khác.



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

**

*Figure 1.8. Architecture of the Transformer*

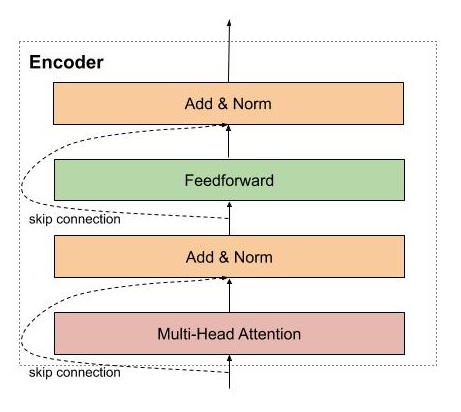
Chú thích cho figure 1.8:

* **Positional encoding**: Bởi vì transformer không có các mạng hồi tiếp hay mạng tích chập nên nó sẽ không biết được thứ tự của các token đầu vào. Vì vậy, cần phải có cách nào đó để cho mô hình biết được thông tin này. Đó chính là nhiệm vụ của positional encoding. Như vậy, sau bước nhúng từ (embedding layers) để thu được các tokens thì ta sẽ cộng nó với các vector thể hiện vị trí của từ trong câu.
* **Lớp Normalization**: Trong hình ảnh cấu trúc, có lớp "Add & Norm" thì từ Norm thể hiện cho lớp Normalization. Lớp này đơn giản là sẽ chuẩn hóa lại đầu ra của multi-head attention, mang lại hiệu quả cho việc nâng cao khả năng hội tụ.
* **Kết nối Residual**: Kết nối residual bản chất rất đơn giản: thêm đầu vào của một khối vào đầu ra của nó. Với kết nối này giúp mạng có thể chồng được nhiều layers. Như trên hình, kết nối residual sẽ được sử dụng sau các khối FFN và khối attention. Như trên hình từ "Add" trong "Add & Norm" sẽ thể hiện cho kết nối residual.
* **Khối Feed-Forward**: Đây là khối cơ bản, sau khi thực hiện tính toán ở khối attention ở mỗi lớp thì khối tiếp theo là FFN. Có thể hiểu là cơ chế attention giúp thu thập thông tin từ những tokens đầu vào thì FFN là khối xử lí những thông tin đó.

1. Giải thích về encoder:

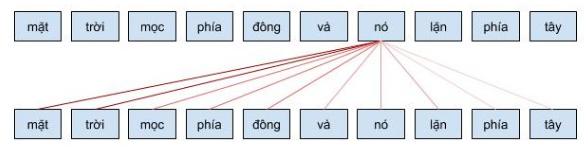
+ Encoder của mô hình Transformer có thể bao gồm nhiều encoder layer tương tự nhau. Mỗi encoder layer của Transformer lại bao gồm 2 thành phần chính là multi head attention và feedforward network, ngoài ra còn có cả skip connection và normalization layer.

+ Nhờ vào thành phần multi-head attention nên đã có sự khác biệt giữa mô hình LSTM và mô hình Transformer



Encoder đầu tiên sẽ nhận ma trận biểu diễn của các từ đã được cộng với thông tin vị trí thông qua positional encoding. Sau đó, ma trận này sẽ được xử lý bởi Multi Head Attention. Multi Head Attention thật chất là self-attention như mình đã đề cập ở trên, nhưng mà để mô hình có thể có chú ý nhiều pattern khác nhau, tác giả đơn giản là sử dụng nhiều self-attention.

1. Giải thích về Self Attention Layer:

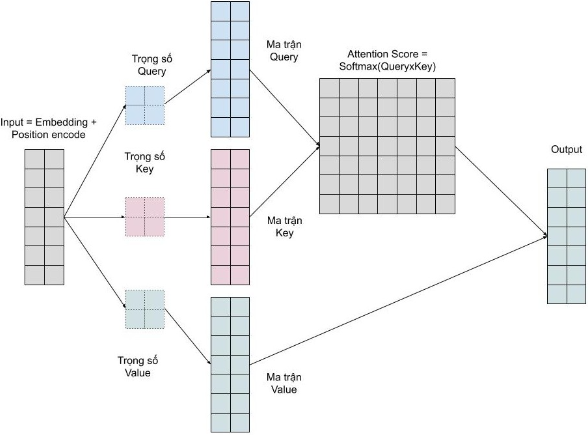
* Self-Attention cho phép mô hình khi mã hóa một từ có thể sử dụng thông tin của những từ liên quan tới nó.
* Ví dụ: 

Trong ví dụ này, khi từ “nó” được mã hóa, nó sẽ chú ý vào các từ liên quan như là “mặt trời”

* Chúng ta có thể tưởng tượng cơ chế self attention giống như cơ chế tìm kiếm (search) của Google.
* Đầu tiên với mỗi từ chúng ta cần tạo ra 3 vector: query vector, key vector và value vector bằng cách nhân ma trận biểu diễn các từ đầu vào với ma trận học tương ứng.
* query vector: vector dùng để chứa thông tin của từ được tìm kiếm, so sánh. Giống như là câu query của google search.
* key vector: vector dùng để biểu diễn thông tin các từ được so sánh với từ cần tìm kiếm ở trên. Ví dụ, như các trang web mà google sẽ so sánh với từ khóa mà bạn tìm kiếm.
* value vector: vector biểu diễn nội dung, ý nghĩa của các từ. Các bạn có thể tượng tượng, nó như là nội dung trang web được hiển thị cho người dùng sau khi tìm kiếm.

+ Quá trình tính toán attention vector có thể được tóm tắt làm 3 bước sau:

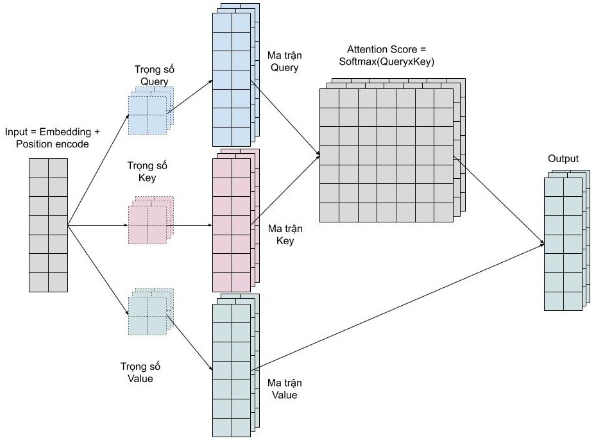
* Bước 1: Tính ma trận query, key, value bằng cách khởi tạo 3 ma trận trọng số query, key, value. Sau đó nhân input với các ma trận trọng số này để tạo thành 3 ma trận tương ứng.
* Bước 2: Tính attention weights. Nhân 2 ma trận key, query vừa được tính ở trên với nhau để với ý nghĩa là so sánh giữa câu query và key để học mối tương quan. Sau đó thì chuẩn hóa về đoạn [0, 1] bằng hàm softmax. 1 có nghĩa là câu query giống với key, 0 có nghĩa là không giống./
* Bước 3: Tính output. Nhân attention weights với ma trận value. Điều này có nghĩa là chúng ta biểu diễn một từ bằng trung bình có trọng số (attention weights) của ma trận value.



*Figure 3.1. Attention vector calculation process*

1. Giải thích về Multi Head Attention

Chúng ta muốn mô hình có thể học nhiều kiểu mối quan hệ giữa các từ với nhau. Với mỗi self-attention, chúng ta học được một kiểu pattern, do đó để có thể mở rộng khả năng này, chúng ta đơn giản là thêm nhiều self-attention. Tức là chúng ta cần nhiều ma trận query, key, value mà thôi. Giờ đây ma trận trọng số key, query, value sẽ có thêm 1 chiều depth nữa.



*Figure 4.1. An example of Multi Head Attention*

Multi head attention cho phép mô hình chú ý đến đồng thời những pattern dễ quan sát được như sau:

* Chú ý đến từ kế trước của một từ
* Chú ý đến từ kế sau của một từ
* Chú ý đến những từ liên quan của một từ

1. Giải thích về Feed-Forward Neural Networks

Sau khi tính toán Self-Attention và thu được các biểu diễn từ đầu vào, mô hình sử dụng một lớp mạng nơ-ron truyền thẳng (feed-forward neural network) để tiếp tục xử lý các biểu diễn này. Lớp này áp dụng các phép biến đổi phi tuyến tính để tạo ra các biểu diễn đầu ra cuối cùng của Encoder.

1. Residuals Connection và Normalization Layer

Trong kiến trúc của mô hình transformer, residuals connection và normalization layer được sử dụng mọi nơi, giống như tinh thần của nó. 2 kỹ thuật giúp cho mô hình huấn luyện nhanh hội tụ hơn và tránh mất mát thông tin trong quá trình huấn luyện mô hình, ví dụ như là thông tin của vị trí các từ được mã hóa.

**CHAPTER 2: Mô hình GPT**

Mô hình GPT (Generative Pre-trained Transformer) là một dạng mô hình Transformer đã được tiền huấn luyện trên một lượng lớn dữ liệu không giám sát. GPT được sử dụng chủ yếu trong các tác vụ sinh văn bản tự động như tạo tiêu đề bài báo, tạo văn bản mô tả hình ảnh, và hoàn thiện đoạn văn.

+ Kiến trúc của mô hình GPT dựa trên các thành phần chính:

* Transformer Encoder Stack: Mô hình GPT sử dụng một số lượng lớp Encoder stack lên nhau để biểu diễn đầu vào. Mỗi lớp Encoder trong stack bao gồm một lớp Multi-Head Self-Attention và một lớp Feed-Forward Neural Networks. Các lớp này giúp mô hình học cách lấy thông tin từ các từ trước đó và xây dựng biểu diễn ngữ cảnh cho các từ hiện tại.
* Positional Encoding: Vì mô hình GPT không sử dụng thông tin vị trí tương đối giữa các từ trong câu, một lớp Positional Encoding được sử dụng để thêm thông tin về vị trí vào biểu diễn từ. Positional Encoding giúp mô hình hiểu được sự thứ tự và ngữ cảnh trong câu.
* Masked Language Model (MLM): Quá trình tiền huấn luyện của GPT thường sử dụng một phương pháp gọi là Masked Language Model (MLM). Trong MLM, một số từ trong câu đầu vào được ngẫu nhiên chọn và che đi (mask). Mô hình phải dự đoán các từ bị che đi dựa trên ngữ cảnh và thông tin từ các từ xung quanh.
* Sinh văn bản: Sau quá trình tiền huấn luyện, mô hình GPT có khả năng sinh ra văn bản tự động bằng cách dự đoán từ tiếp theo dựa trên ngữ cảnh trước đó. Mô hình có thể được sử dụng để tạo ra các đoạn văn bản mô tả hình ảnh, viết tiêu đề, hoàn thiện câu, và nhiều tác vụ khác liên quan đến sinh văn bản tự động.

+ Dưới đây là một số thông tin về các phiên bản GPT khác nhau:

1/ GPT-1:

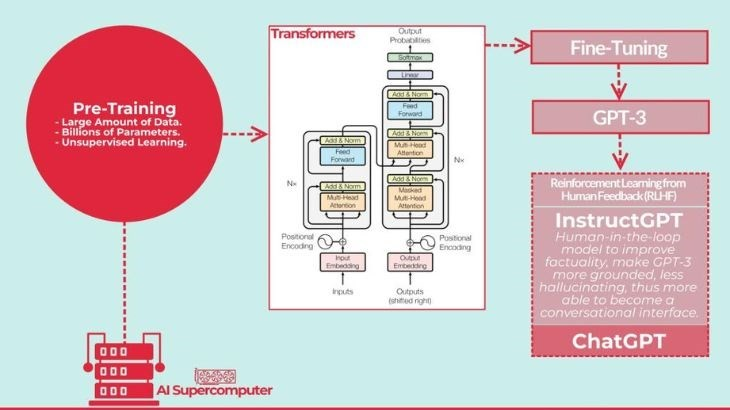
* GPT-1 là phiên bản đầu tiên của mô hình GPT, được phát hành vào năm 2018.
* Nó sử dụng kiến trúc Transformer và có 117 triệu tham số.
* Một trong những điểm mạnh của GPT-1 là khả năng tạo ngôn ngữ trôi chảy và mạch lạc khi được đưa ra lời nhắc hoặc ngữ cảnh.
* Tuy nhiên, GPT-1 có một số hạn chế, bao gồm xu hướng tạo văn bản lặp lại và khả năng suy luận hạn chế.

2/ GPT-2:

* GPT-2 được phát hành vào năm 2019 và là phiên bản kế thừa của GPT-1.
* Nó chứa 1,5 tỷ tham số, lớn hơn đáng kể so với GPT-1.
* GPT-2 có khả năng tạo ra các chuỗi văn bản mạch lạc và chân thực hơn.

3/ GPT-3:

* GPT-3 là phiên bản tiếp theo, được phát hành vào năm 2020.
* Với 175 tỷ tham số, GPT-3 là một trong những mô hình lớn nhất và mạnh nhất hiện nay.
* Nó có khả năng thực hiện nhiều tác vụ xử lý ngôn ngữ tự nhiên, từ trả lời câu hỏi đến tạo văn bản và dịch ngôn ngữ.



*Figure 2.1.1. OpenAI's ChatGPT operating structure is based on the GPT model*

**CHAPTER 3: FINE-TUNING CHO MÔ HÌNH Transformer based Encoder**

* Fine-tuning là một kỹ thuật huấn luyện mô hình học máy đã được tiền huấn luyện (pretrained) trên tập dữ liệu lớn bằng cách tiếp tục huấn luyện mô hình trên tập dữ liệu nhỏ hơn, cụ thể là tập dữ liệu liên quan đến nhiệm vụ cụ thể mà bạn muốn mô hình thực hiện.
* Trong Machine Learning, Fine-tuning là một phương pháp của transfer learning, sử dụng weight của một pre-trained model để train với một bộ data mới, phù hợp với mục đích của người dùng và số lượng dataset thường nhỏ hơn khi pre-train. Việc làm này giúp tăng độ chính xác của model so với việc train trực tiếp với bộ dataset nhỏ của chúng ta.

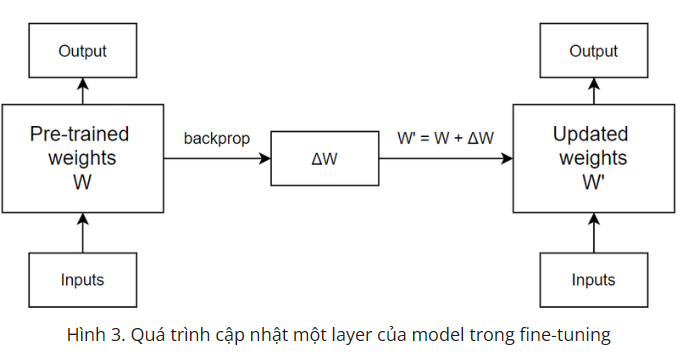
+ Các cách thông thường để tiến hành tiền huấn luyện và tinh chỉnh mô hình Transformer based Encoder:

1/ Pretraining (Tiền huấn luyện):

* Tiền huấn luyện không giám sát: Mô hình Transformer based Encoder được tiền huấn luyện trên một lượng lớn dữ liệu ngôn ngữ tự nhiên không được gán nhãn. Quá trình này thường sử dụng một tác vụ phụ (auxiliary task) như dự đoán từ tiếp theo trong câu (next word prediction) hoặc dự đoán từ bị che đi (masked word prediction) để hướng dẫn quá trình học.

2/ Fine-tuning (Tinh chỉnh):

* Tinh chỉnh toàn bộ mô hình: Sau khi tiền huấn luyện, mô hình Transformer based Encoder có thể được tinh chỉnh trên tác vụ cụ thể. Trong quá trình này, một lượng nhỏ dữ liệu gán nhãn được sử dụng để điều chỉnh các trọng số của mô hình. Đầu ra của mô hình được so sánh với nhãn thực tế và sử dụng một hàm mất mát (loss function) như Cross-Entropy để tính toán độ lỗi và cập nhật trọng số mạng nơ-ron thông qua thuật toán lan truyền ngược (backpropagation).
* Tinh chỉnh phần cuối của mô hình: Thay vì tinh chỉnh toàn bộ mô hình, một cách tiếp cận khác là chỉ tinh chỉnh phần cuối của mô hình Transformer based Encoder. Thông thường, một lớp đầu ra tùy chỉnh được thêm vào mô hình để dự đoán kết quả cho tác vụ cụ thể, và chỉ các trọng số của lớp này được cập nhật trong quá trình tinh chỉnh. Các trọng số của các lớp khác trong mô hình được giữ nguyên.
* Transfer Learning (Học chuyển giao): Một cách tinh chỉnh khác là sử dụng mô hình Transformer based Encoder đã được tiền huấn luyện cho một tác vụ và sử dụng lại nó cho một tác vụ khác có liên quan. Trọng số của mô hình chỉ được điều chỉnh cho phần đầu ra tương ứng với tác vụ mới, trong khi các phần khác của mô hình được giữ nguyên.



Chú thích cho hình: Pre-trained weights 𝑊 của model sẽ biến thành updated weights 𝑊′ dựa trên giá trị weight cần thay đổi Δ𝑊 thu được từ quá trình backprop. Và ở iteration tiếp theo, 𝑊′ lại được update với một Δ𝑊 khác.