**GNN**

A graph with text and words

Description automatically generated with medium confidence

Reinforcement learning

3 bài toán là

Node , Cạnh , Đồ thị

Bài toàn về encoder chỉ giải mã , chúng sẽ self-attention lên hết các từ

Ví dụ: tên nguyễn huy hoàng ( thì khi attention ở từ huy , thì encoder sẽ quan tâm cả nguyễn và hoàng , nhưng ở decoder nó sẽ chỉ quan tâm từ tên và từ nguyễn , khi dự đoán từ sau từ huy nó sẽ tinh chỉnh và so sánh kết quả với từ hoàng , từ đó sẽ có trọng số và đường đi của vecto khi mà kết thúc từ hoàng )

Đúng rồi! Khi nói về \*\*các từ tương lai\*\* trong ngữ cảnh của \*\*decoder\*\*, chúng chính là các từ \*\*chưa xuất hiện\*\* trong câu hoặc \*\*các từ nằm sau\*\* từ đang được xét.

### Cụ thể:

- \*\*Từ tương lai\*\*: Là những từ trong chuỗi đầu ra mà decoder chưa sinh ra hoặc chưa xử lý đến. Ví dụ, trong câu "nay ăn phở rất":

- Nếu decoder đang ở vị trí xử lý từ "phở", thì từ "rất" sẽ được coi là \*\*tương lai\*\* vì nó chưa được sinh ra trong quá trình.

### Cách hoạt động:

- \*\*Decoder chỉ xem xét\*\* các từ đã được sinh ra (các từ ở vị trí trước đó) để quyết định từ tiếp theo mà nó sẽ sinh ra.

- \*\*Các từ sau từ hiện tại\*\* (tức là từ chưa sinh ra) sẽ \*\*không được chú ý\*\* trong quá trình này.

### Tóm tắt:

- \*\*Các từ tương lai\*\* là những từ chưa xuất hiện trong chuỗi đầu ra mà decoder đang xử lý. Khi xử lý một từ, decoder chỉ chú ý đến các từ đã sinh ra trước đó để tạo ra ngữ cảnh chính xác cho từ tiếp theo.

Nếu bạn có câu hỏi nào khác hoặc muốn tìm hiểu thêm về chủ đề này, hãy cho tôi biết nhé!

Giải thích về lớp attention

\*\*Self-attention\*\* là một cơ chế quan trọng trong mô hình \*\*Transformer\*\*, giúp mô hình "chú ý" đến các từ khác trong chuỗi đầu vào (hoặc đầu ra) để xây dựng các biểu diễn ngữ nghĩa mạnh mẽ hơn. Self-attention giúp mô hình hiểu được mối quan hệ giữa các từ trong chuỗi mà không cần phải xử lý tuần tự như các mô hình trước (RNN, LSTM).

### Các bước trong \*\*Self-attention\*\*:

Self-attention hoạt động qua ba bước chính: \*\*Query\*\*, \*\*Key\*\*, và \*\*Value\*\*. Đây là các thành phần giúp mô hình xác định "tầm quan trọng" của các từ khác đối với một từ hiện tại.

#### 1. \*\*Đầu vào\*\*:

- Mỗi từ trong chuỗi đầu vào (hoặc đầu ra) sẽ được biểu diễn dưới dạng \*\*embedding\*\*. Giả sử có một chuỗi từ: "I love programming".

#### 2. \*\*Query, Key, Value\*\*:

- \*\*Query\*\* (Q), \*\*Key\*\* (K), và \*\*Value\*\* (V) được tính từ các embeddings của từ. Mỗi từ sẽ được chuyển thành ba vector khác nhau:

- \*\*Q\*\* (Query): Mô tả những gì từ hiện tại muốn tìm kiếm.

- \*\*K\*\* (Key): Mô tả các "mối quan tâm" của các từ khác.

- \*\*V\*\* (Value): Chứa thông tin giá trị mà từ hiện tại có thể lấy từ các từ khác.

#### 3. \*\*Tính toán Attention Scores\*\*:

- Mỗi từ hiện tại sẽ "chú ý" đến các từ khác trong chuỗi để quyết định mức độ quan trọng của các từ này đối với nó. Để làm điều này, ta tính \*\*attention score\*\* giữa \*\*Query\*\* của từ hiện tại và \*\*Key\*\* của các từ khác. Công thức tính attention score là:

\[

\text{Attention score}(Q, K) = Q \cdot K^T

\]

- Mỗi từ hiện tại sẽ tính toán sự liên quan (sự tương đồng) giữa \*\*Query\*\* của mình và \*\*Key\*\* của tất cả các từ khác. Đây chính là mức độ "chú ý" mà từ hiện tại dành cho các từ khác.

#### 4. \*\*Softmax\*\*:

- Sau khi tính toán \*\*attention scores\*\*, các giá trị này sẽ được đưa qua \*\*softmax\*\* để chuyển chúng thành xác suất. Điều này đảm bảo rằng tổng của các weights (mức độ chú ý) bằng 1, và các từ có mức độ quan trọng cao sẽ có weight lớn hơn.

\[

\text{Attention weight} = \text{softmax}(\text{Attention score})

\]

#### 5. \*\*Kết hợp Value\*\*:

- Cuối cùng, mỗi từ sẽ kết hợp thông tin từ các từ khác (dựa trên \*\*Value\*\* của các từ đó) theo \*\*mức độ chú ý\*\* mà nó đã tính toán ở bước trước.

\[

\text{Output} = \sum (\text{Attention weights} \times V)

\]

- Kết quả là một biểu diễn tổng hợp của từ hiện tại, được xây dựng từ thông tin của các từ khác trong chuỗi.

---

### \*\*Các chế độ Self-attention\*\*:

1. \*\*Scaled Dot-Product Attention\*\*:

- Đây là phiên bản cơ bản của self-attention. Mỗi \*\*Query\*\* được nhân với \*\*Key\*\* để tính attention score, sau đó chia cho căn bậc hai của kích thước của Key để tránh giá trị quá lớn.

- Công thức tính attention:

\[

\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d\_k}}\right)V

\]

- \(d\_k\) là kích thước của vector Key.

2. \*\*Multi-Head Attention\*\*:

- Thay vì chỉ sử dụng một set các Q, K, V, mô hình \*\*multi-head attention\*\* chia chúng thành nhiều "đầu" (heads). Mỗi head sẽ tính toán attention riêng biệt, và sau đó kết hợp kết quả từ tất cả các head lại.

- Điều này cho phép mô hình học được các mối quan hệ khác nhau từ nhiều góc độ khác nhau, giúp mô hình mạnh mẽ hơn trong việc nắm bắt các thông tin ngữ nghĩa phức tạp.

3. \*\*Masked Self-attention\*\* (dành cho Decoder):

- Khi mô hình sinh văn bản (như GPT hoặc các mô hình autoregressive), cần sử dụng \*\*masked self-attention\*\*. Trong trường hợp này, mỗi từ chỉ có thể "chú ý" đến các từ đã sinh ra trước đó và không được phép nhìn vào các từ tương lai.

- Điều này đảm bảo tính tuần tự trong quá trình sinh ngôn ngữ.

---

### \*\*Ví dụ về Self-attention\*\*:

Giả sử ta có chuỗi "I love programming" và muốn tính toán self-attention cho từ "love":

1. \*\*Tạo Q, K, V cho "love"\*\*:

- Ta tạo ra các vector \*\*Q\*\*, \*\*K\*\*, và \*\*V\*\* cho từ "love" từ embedding của nó.

2. \*\*Tính attention score\*\*:

- Ta tính toán \*\*Q\*\* của "love" với \*\*K\*\* của các từ khác ("I", "love", "programming").

- Kết quả là một số điểm chú ý cho từng từ, ví dụ:

- Attention score("love", "I") = 0.5

- Attention score("love", "love") = 1.2

- Attention score("love", "programming") = 0.3

3. \*\*Softmax\*\*:

- Ta đưa các attention score qua hàm softmax để có xác suất (weights):

- Attention weight("love", "I") = 0.2

- Attention weight("love", "love") = 0.5

- Attention weight("love", "programming") = 0.3

4. \*\*Kết hợp Value\*\*:

- Cuối cùng, ta kết hợp các \*\*Value\*\* của các từ theo các weights này:

- Output = 0.2 \* Value("I") + 0.5 \* Value("love") + 0.3 \* Value("programming")

Kết quả là một biểu diễn ngữ nghĩa mới cho từ "love", kết hợp thông tin từ các từ khác trong câu.

---

### Tóm lại:

\*\*Self-attention\*\* giúp mỗi từ trong chuỗi "chú ý" đến các từ khác để xây dựng biểu diễn ngữ nghĩa. Nó rất linh hoạt vì không yêu cầu xử lý tuần tự như các mô hình trước đây (RNN, LSTM), giúp mô hình Transformer có thể xử lý dữ liệu song song và học được các mối quan hệ xa giữa các từ.

+ cộng với từ nhúng ban đầu

Đúng vậy! Sau khi tính toán \*\*self-attention\*\*, phần kết quả từ \*\*attention\*\* sẽ được \*\*cộng vào vector từ\*\* để tạo ra \*\*biểu diễn ngữ nghĩa mới\*\* cho từ đó. Đây là bước quan trọng để làm phong phú thêm thông tin ngữ nghĩa của từ hiện tại bằng cách kết hợp nó với thông tin từ các từ khác trong chuỗi.

### Quá trình chi tiết của \*\*Self-attention\*\*:

1. \*\*Tính toán Attention Weights\*\*:

- Như đã giải thích, mỗi từ hiện tại (ví dụ, từ "love" trong câu "I love programming") sẽ tính toán điểm chú ý (attention score) giữa mình và các từ khác (ví dụ, "I", "love", "programming").

- Những điểm này sẽ được đưa qua hàm \*\*softmax\*\* để tạo ra các \*\*attention weights\*\* (mức độ quan trọng của từng từ).

2. \*\*Lấy thông tin từ các từ khác\*\*:

- Mỗi từ sẽ sử dụng \*\*attention weights\*\* để kết hợp thông tin từ các \*\*Value\*\* của các từ khác. Các từ có mức độ chú ý cao sẽ đóng góp nhiều hơn vào biểu diễn cuối cùng.

- Giả sử các \*\*Value\*\* của từ "I", "love", và "programming" lần lượt là \( V\_{\text{I}} \), \( V\_{\text{love}} \), và \( V\_{\text{programming}} \).

3. \*\*Kết hợp với Value\*\*:

- Mỗi từ trong chuỗi đầu vào (hoặc đầu ra) sẽ nhận được một "biểu diễn mới", là sự kết hợp của \*\*Value\*\* của tất cả các từ khác, trọng số theo \*\*attention weights\*\*.

- Biểu diễn mới của từ "love" (sau self-attention) sẽ là:

\[

\text{New representation of "love"} = \text{Attention Weight}\_{\text{I}} \times V\_{\text{I}} + \text{Attention Weight}\_{\text{love}} \times V\_{\text{love}} + \text{Attention Weight}\_{\text{programming}} \times V\_{\text{programming}}

\]

4. \*\*Cộng với Embedding gốc\*\* (Optional trong một số mô hình):

- Sau khi tính toán biểu diễn mới từ self-attention, thường sẽ có một bước \*\*cộng residual\*\* (tức là cộng kết quả với vector embedding gốc của từ).

- Việc này giúp duy trì thông tin ban đầu của từ trong suốt quá trình tính toán, và tránh mất thông tin khi thực hiện các phép toán phức tạp.

- Cộng residual có thể được thực hiện như sau:

\[

\text{Final output} = \text{Original embedding} + \text{New representation from attention}

\]

- Tuy nhiên, trong một số mô hình như \*\*GPT\*\* hoặc các mô hình Transformer khác, bước cộng residual này có thể được thực hiện sau khi \*\*layer normalization\*\*.

---

### Tóm lại:

- Sau khi tính toán \*\*self-attention\*\*, kết quả \*\*attention\*\* sẽ được \*\*cộng vào vector từ gốc\*\*, tạo thành \*\*biểu diễn mới\*\* của từ đó.

- Việc này giúp mỗi từ "học" được thông tin từ các từ khác trong chuỗi, từ đó có một \*\*biểu diễn ngữ nghĩa phong phú hơn\*\* cho từng từ.

- Trong một số trường hợp, phần kết quả từ self-attention có thể được \*\*cộng thêm với embedding gốc\*\* của từ để giữ lại thông tin gốc, giúp mô hình học tốt hơn.