Sequence Models

Trần Đức Mạnh

Ngày 22 tháng 9 năm 2022

Mục lục

Chương 1

Recurrent Neural Networks

1.1 Recurrent Neural Networks

1.1.1 Notation

- x: chuỗi input
- y: chuỗi output
- $x^{(i)<t>}, y^{(i)<t>}$: từ thứ t của chuỗi input/output trong example thứ i. Là một **one-hot vector** (Tức là vector mà chỉ có 1 phần tử nào đó bằng 1, còn lại bằng 0)

$$x^{<t>} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{1.1}$$

- $T_x^{(i)}, T_y^{(i)}$: độ dài của chuỗi input/output trong example thứ i
- Vocabulary: từ điển từ, một String Array

$$\text{vocabulary} = \begin{bmatrix} \text{a} \\ \text{aaron} \\ \vdots \\ \text{zulu} \end{bmatrix} \tag{1.2}$$

  - **Lưu ý**: $x^{<t>}$ có chiều bằng vocabulary, và có phần tử tại vị trí tương ứng với vocabulary bằng 1
  - Ví du: Trong Vocabulary, từ "and" ở vị trí 367, thì $x^{<t>}$ tương ứng với "and" sẽ là một one-hot vector, với phần tử thứ 367 bằng 1

1.1.2 Recurrent Neural Network Model

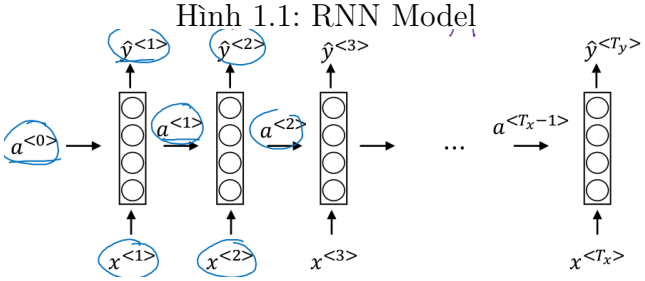
- Tại sao không sử dụng Standard network?

– Vì độ dài chuỗi input và chuỗi output có thể khác nhác nhau với mỗi examples.

– Vì chúng ta có quan tâm tới thứ tự của từ trong chuỗi, và Standard network không xử lý được thông tin ý

- Recurrent Neural Network

Hình 1.1: RNN Model

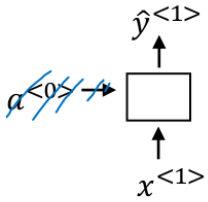


– Forward:

* $a^{<t>} = g(\omega_{aa}a^{<t-1>} + \omega_{ax}x^{<t>} + b_a) = g(\omega_a[a^{<t-1>}, x^{<t>}] + b_a)$
* $\hat{y}^{<t>} = g(\omega_{ya}a^{<t>} + b_y) = g(\omega_y a^{<t>} + b_y)$
  · ω_{ax}: weight tính cho a và nhân với x (tương tự: ω_{ya} là weight tính cho y và nhân với a)
  · b_a: bias tính cho a (tương tự: b_y: bias tính cho y)
  · $\omega_{aa}a^{<t-1>} + \omega_{ax}x^{<t>} = \omega_a[a^{<t-1>}, x^{<t>}]$
  · $\omega_a = [\omega_{aa} \vdots \omega_{ax}]$ (tức là ghép $\omega_{aa}(a, b)$ với $\omega_{ax}(a, c)$ thành $\omega_a(a, b + c)$)
  · $[a^{<t-1>}, x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix}$
  (tức là ghép $a^{<t-1>}(n, 1)$ với $x^{<t>}(m, 1)$ thành $\begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix}(n + m, 1)$)

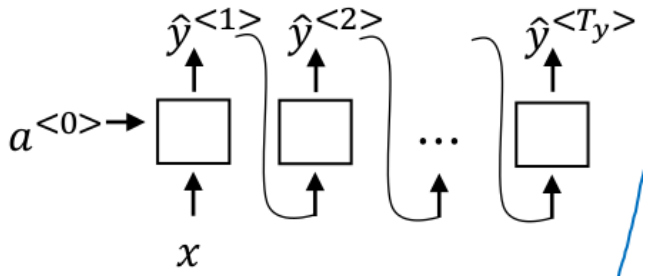
1.1.3 Different types of RNNs

- One to one:



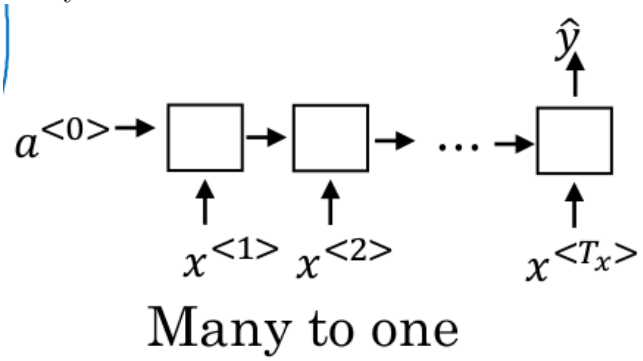
One to one

- One to many:

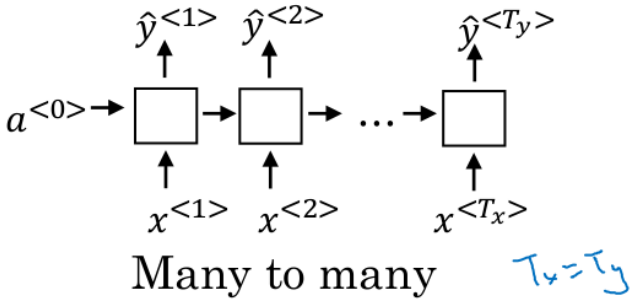


One to many

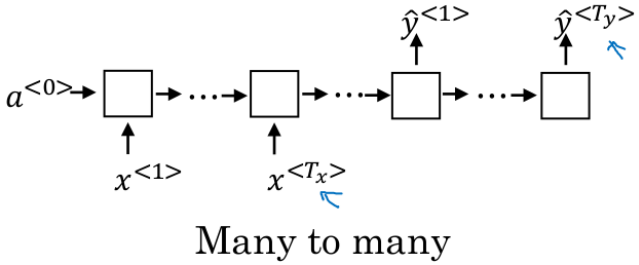
- Many to one:



- Many to many ($T_x = T_y$):



- Many to many (Sequence to Sequence):

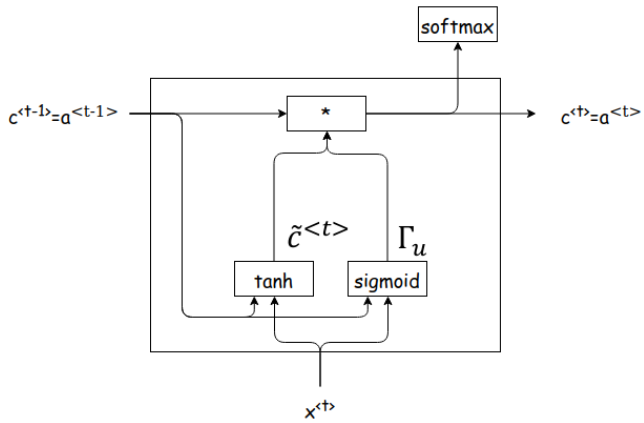


1.1.4 Language model and sequence generation

1.1.5 Vanishing gradients with RNNs

1.1.6 GRU

- Hình ảnh minh họa:

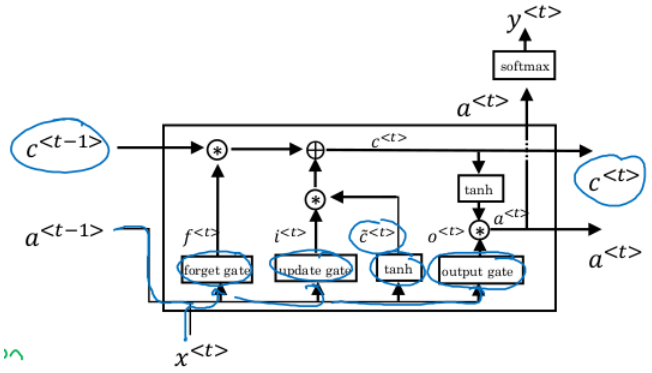


- Chi tiết:

- $\tilde{c}^{<t>} = tanh(\omega_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$
  * $\Gamma_r = \sigma(\omega_r[c^{<t-1>}, x^{<t>}] + b_r)$
- $\Gamma_u = \sigma(\omega_u[c^{<t-1>}, x^{<t>}] + b_u)$
- $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$ (tính toán ở hộp *)

1.1.7 LSTM

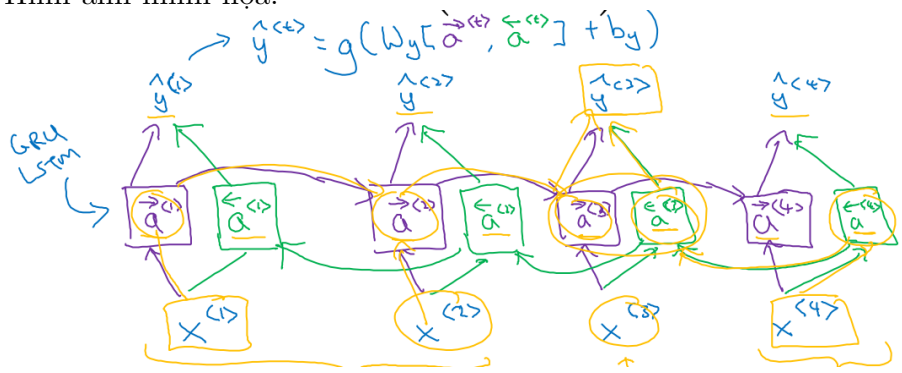
- Hình ảnh minh họa:



- Chi tiết:
  - $\tilde{c}^{<t>} = tanh(\omega_c[a^{<t-1>}, x^{<t>}] + b_c)$
  - $\Gamma_u = \sigma(\omega_u[a^{<t-1>}, x^{<t>}] + b_u)$
  - $\Gamma_f = \sigma(\omega_f[a^{<t-1>}, x^{<t>}] + b_f)$
  - $\Gamma_o = \sigma(\omega_o[a^{<t-1>}, x^{<t>}] + b_o)$
  - $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$
  - $a^{<t>} = \Gamma_o * tanh(c^{<t>})$

1.1.8 Bidiractional RNN

- RNN 2 chiều đưa ra để giải quyết hạn chế của RNN 1 chiều trong việc lấy thông tin của chuỗi. RNN 1 chiều chỉ có thể trích thông tin từ "quá khứ", trong khi thông tin từ "tương lai" cũng quan trọng không kém.
- Hình ảnh mình họa:



1.1.9 Deep RNNs

Chương 2

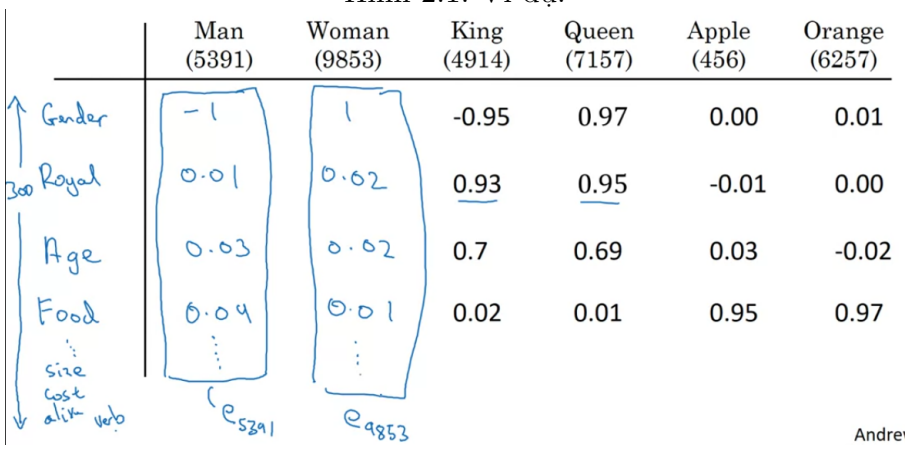
Natural Language Processing & Word Embeddings

2.1 Introduction to Word Embeddings

2.1.1 Word Representation

- Tại sao one-hot vector chưa đủ tốt?
  - Đặt vấn đề: Bạn có 1 chuỗi "I want orange . . . ", và Model của bạn học được rằng "juice" là từ thích hợp để điền vào . . . , tuy nhiên điều đó không có nghĩa rằng Model của bạn cũng biết điền "juice" vào nếu chuỗi là "I want apple . . . ". Điều này xảy ra vì one-hot vector không thể hiện sự tương đồng giữa "orange" và "juice".
  - Đề khắc phục điều đó, chúng ta sử dụng **Featurized Representation** thay cho One-Hot.
- Featurized Representation: Word embedding

Hình 2.1: Ví dụ:



  - Nhận xét ví dụ:
    - Vì one-hot vector chỉ có thể chỉ vị trí của từ trong Vocabulary *(như Man(5391) là để chỉ từ "Man" nằm ở vị trí 5391 trong Vocabulary)*, nên ta sử dụng Featurized Representation để có thể thể hiện cả những tính chất của từ.
    - Bằng cách này, ta thấy rõ Man và Woman khá tương đồng nhưng chỉ khác nhau rõ rệt về Gender, còn Apple và Orange lúc này có các chỉ số gần như giống nhau,

hay nói cách khác là lúc này ta đã biểu diễn được 2 từ này một cách tương đồng nhau hơn

  * e_{5391} là để chỉ vector tương ứng với từ 5391 trong Vocabulary *(trường hợp này là "Man")*, còn O_{5391} là để chỉ one-hot vector của từ đó

- Visualizing Word Embeddings

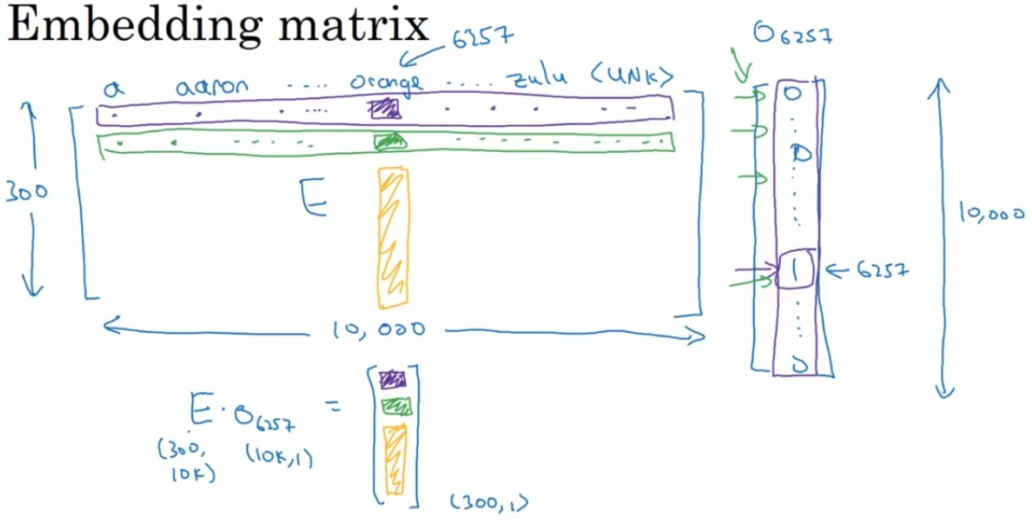
2.1.2 Using Word Embeddings

2.1.3 Properties of Word Embeddings

2.1.4 Embedding Matrix

- Gọi E là Embedding Matrix, ta có: $E \times O_t = e_t$ $((m, n) \times (n, 1) = (m, 1))$
  - m: số lượng features
  - n: số lượng từ trong Vocabulary
  - t: Vị trí của từ trong Vocabulary

Hình 2.2: Hình ảnh minh họa:

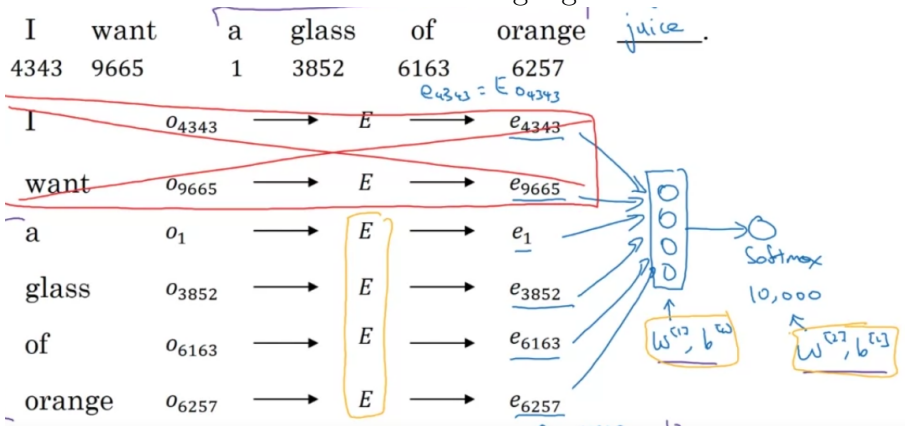


2.2 Learning Word Embeddings: Word2Vec & Glove

2.2.1 Learning Word Embeddings

- Neural Language Model
  - Ở hình 2.3, với context "I want a glass of orange ..." và ta muốn Model này điền "juice" vào chỗ ...
  - Để làm được điều đó, xây dựng Model theo hình 2.3 và cho nó học 5 parameters sau: $E, \omega^{[1]}, b^{[1]}, \omega^{[2]}, b^{[2]}$
  - Thay vì sử dụng tất 7 kí tự của chuỗi được cho, ta có thể chỉ sử dụng 5 kí tự gần ...để làm context

Hình 2.3: Neural Language Model



- Other context/target pair
  - Cho ví dụ: I want a glass of orange juice to go along with my cereal.
  - Context/target pair:
    - Last 4 words: "a glass of orange ? "
    - 4 words on left & right: "a glass of orange ? to go along with"
    - Last 1 word: "orange ? "
    - Nearby 1 word **(skip gram)**: "glass . . . ? "

2.2.2 Word2Vec

- Skip-grams model
  - Ý tưởng: Model này chọn 1 từ làm Context, rồi chọn 1 từ làm Target *(có nhiều cách chọn, nhưng ngày gần, cách 1 từ, cách 10 từ, . . .)*, rồi cho Model này học cách dự đoán Target từ Context. Nhưng mục đích của Model này không phải là để dự đoán chính xác Target mà là để **học Embedding Matrix** E.
  - Model:
    - Với Vocab Size = 10000, Context c và Target c, ta có:
    - $O_c \rightarrow E \rightarrow e_c \rightarrow (\text{Softmax}) \rightarrow \hat{y}$
    - Softmax: $P(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10000} e^{\theta_j^T e_c}}$ với θ_t = parameter của output t
    - Cost: $L(\hat{y}, y) = -\sum_{i=1}^{10000} y_i log(\hat{y}_i)$
- Problem with softmax classification
  - $P(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10000} e^{\theta_j^T e_c}}$ có Computation Cost cao vì tổng của 10000 khá lớn, và sẽ càng tốn nếu vocab size của bạn càng lớn.
  - Cách giải quyết:
    - Hierarchy Softmax

2.2.3 Negative Sampling

- Ý tưởng: là 1 Model khác được đưa ra nhằm giải quyết vấn đề computation cost của Skip-grams.
  Model này học cách nhận 1 cặp từ và đoán xem nó có đúng là 1 cặp context/target hay không.

- Dataset:

  - Dataset này được tạo ra bằng cách chọn 1 từ làm Context, rồi chọn 1 từ làm Target, label cặp từ này là 1. Rồi chọn K từ khác từ Vocabulary để làm Target rồi label K cặp này là 0.
  - Ví dụ: (word ở đây là Target, còn target? là label)

context	word	target?
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0

    * Small Dataset: $K = [5; 20]$
    * Large Dataset: $K = [5; 20]$
  - Cách chọn K từ để label 0 (Negative Sample): **(Chưa hiểu lắm)**

- Model:

  - $O_c \to E \to e_c \to \begin{matrix} O \\ O \\ \vdots \\ O \end{matrix}$
  - Layer cuối của model là 10000 binary classification, mỗi node trả lời câu hỏi tương ứng (juice? king? ...)
  - Mỗi node lúc này tính toán: $P(y = 1|c, t) = \sigma(\theta_t^T e_c)$ với với θ_t = parameter của output t
  - Với mỗi iteration:
    * Ta train $K + 1$ node (kèm với E)
    * Chọn K negative sample khác rồi train model
  - Vì thay vì train 10000 node, giờ ta chỉ train $K + 1$ nên computation cost rẻ hơn rất nhiều

2.2.4 GloVe Word Vectors

- **I don't thing this is important to fully understand**

2.3 Applications Using Word Embeddings

2.3.1 Sentiment Classification

2.3.2 Debiasing Word Embeddings

Chương 3

Seuqence Models & Attention Mechanism

3.0.1 Basic Model

3.0.2 Picking the Most Likely Sentence

- Trong Machine Translation *(hay những bài toán tương tự)*, chúng ta có thể có được nhiều hơn 1 chuỗi Output từ 1 chuỗi input, thì vấn đề ở đây là chúng ta không thể chọn bừa 1 trong những chuỗi Output đó *(vì trong số đó chắc chắn có chuỗi Output phù hợp hơn các chuỗi còn lại, ví dụ có nhiều cách dịch từ ngôn ngữ đích sang ngôn ngữ nguồn, nhưng chắc chắn sẽ có cách dịch tốt hơn là chỉ dịch word-by-word)*. Chính vì thế mà ta phải tìm cách **"Picking the Most Likely Sentence"**.
- Bài toán được hiểu như sau: chọn output $y = y^{<1>}, y^{<2>}, \ldots, y^{<T_y>}$ sao cho $P(y^{<1>}, y^{<2>}, \ldots, y^{<T_y>}|x)$ là lớn nhất

$$\arg \max_{y^{<1>},\ldots,y^{<T_y>}} P(y^{<1>}, \ldots, y^{<T_y>}|x)$$

- Một trong những cách không nên làm là **Greedy Search**:
  - Thuật toán: chọn từng từ, sao cho nó là tốt nhất. Chọn $y^{<1>}$ sao cho nó là tốt nhất rồi chọn $y^{<2>}$ sao cho nó là tốt nhất rồi cứ thế đến $y^{<T_y>}$
  - Lý do nó không tốt:
    - Cho 2 ví dụ sau:
      Ví dụ 1: I am visiting Frence
      Ví dụ 2: I am going to visiting Frence
      Với $P(y|x)$ của ví dụ 1 tốt hơn
    - Nhưng vì $P(\text{I am visiting}|x) < P(\text{I am going}|x)$ do "going" phổ biến hơn "visiting", nên Ví dụ 2 sẽ được chọn thay vì Ví dụ 1

3.0.3 Beam Search

- Thuật toán: Sau khi chạy Encoder, ở Decoder, sau khi tìm ra $\hat{y}^{<1>}$, tức tính được $P(\hat{y}^{<1>}|x)$ *(là vector có chiều bằng Vocab Size, với mỗi phần tử là xác suất từ đó xuất hiện ở đầu câu)*. từ $P(\hat{y}^{<1>}|x)$ ta chọn ra B **(Beam Width)** từ với xác suất cao nhất, lưu vào bộ nhớ.
  Rồi với mỗi từ được lưu trong bộ nhớ, ta tính $\hat{y}^{<2>}$ (ở đây tạo ra B mạng để tính), ta được $P(\hat{y}^{<2>}|x, \hat{y}^{<1>})$, rồi ta tính $P(\hat{y}^{<1>}, \hat{y}^{<2>}|x) = P(\hat{y}^{<1>}|x)P(\hat{y}^{<2>}|x, \hat{y}^{<1>})$ để từ đó chọn tiếp B từ với xác suất $P(\hat{y}^{<1>}, \hat{y}^{<2>}|x)$ cao nhất để tính $\hat{y}^{<3>}$

3.0.4 Refinements to Beam Search

- Ta có:

$$\arg\max_y P(y^{<1>}, \ldots, y^{<T_y>}|x) = \arg\max_y \prod_{t=1}^{T_y} P(y^{<t>}|x, y^{<1>}, \ldots, y^{<t-1>})$$

- Ta có thể thấy tích $P(y^{<t>}|x, y^{<1>}, \ldots, y^{<t-1>})$ có thể trở nên rất nhỏ đến mức máy tính không thể lưu chính xác.
  Nên chúng ta tính toán bằng biểu thức tương tự sau:

$$\arg\max_y \sum_{t=1}^{T_y} logP(y^{<t>}|x, y^{<1>}, \ldots, y^{<t-1>})$$

  Bằng cách tính *log* của xác suất, ta sẽ không phải lo về việc tràn số, đồng thời vẫn dữ nguyên được bản chất của phép toán.
  Tuy nhiên vẫn có thể xảy ra trường hợp tích xác suất quá nhỏ, dẫn đến giá trị *log* bị âm rất lớn, dẫn đến tràn số, thì ta thêm $\frac{1}{T_y^\alpha}$ để ngăn điều đó.

$$\frac{1}{T_y^\alpha} \sum_{t=1}^{T_y} logP(y^{<t>}|x, y^{<1>}, \ldots, y^{<t-1>})$$

  Trong đó α là hyperparameter, mục đích của α chỉ là để làm "mềm" phép tính (với $\alpha = 0.7$). Trong trường hợp $\alpha = 1$ là Length Normalize, còn $\alpha = 0$ là No Normalize.

3.0.5 Error Analysis in Beam Search

3.0.6 Bleu Score

3.0.7 Attention Model Intuition

3.0.8 Attention Model

Chương 4

Transformer Network