**Báo cáo tiến trình học tập tuần 6**

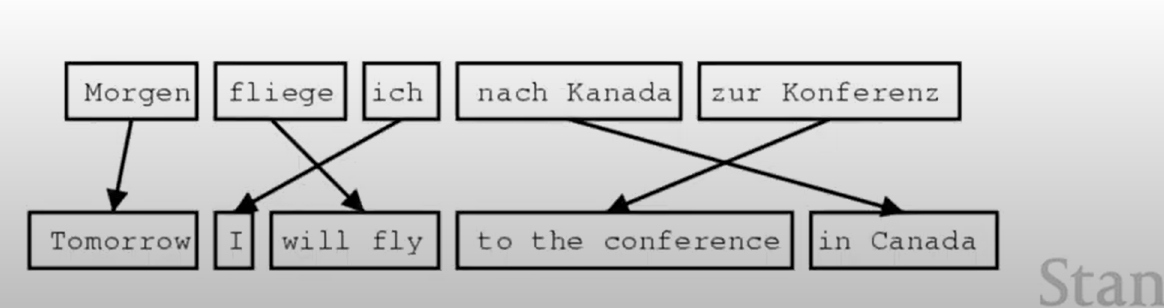
**Machine Translation**

* **MT** là việc dịch một đoạn văn bản x từ 1 ngôn ngữ (ngôn ngữ nguồn) thành một đoạn văn bản y ở một ngôn ngữ khác (ngôn ngữ đích)

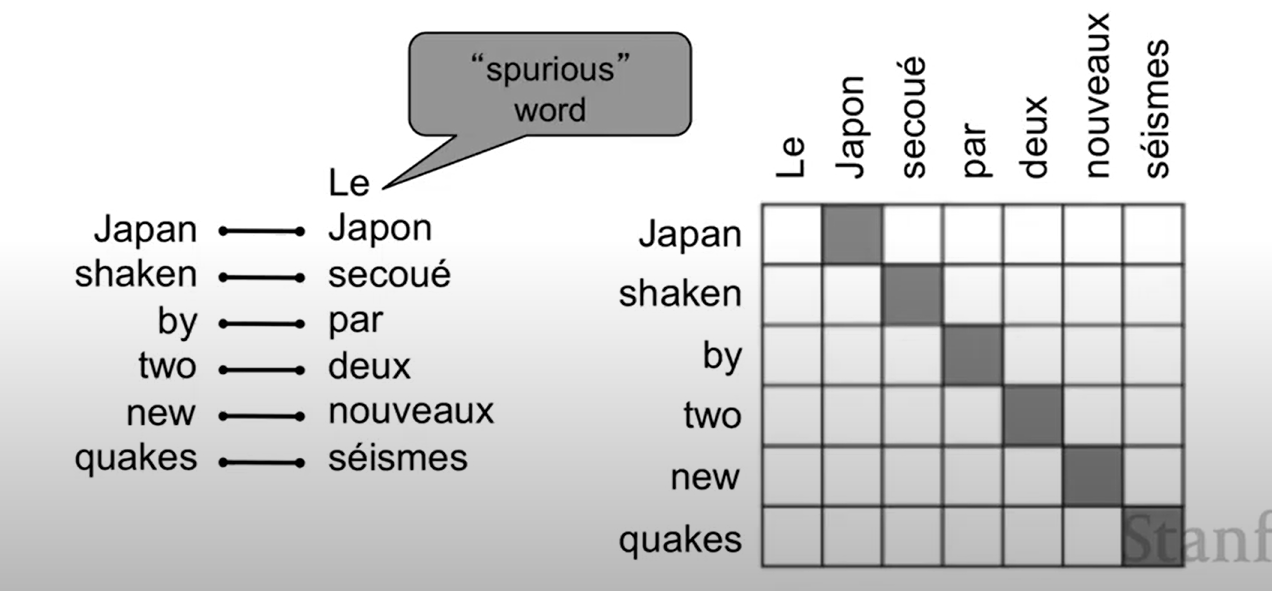
1. **Statistical Machine Translation (dịch máy thống kê\_1990s-2010)**
2. **Giới thiệu**
   * Học một mô hình xác suất từ dữ liệu
   * Bài toán: cho một câu tiếng Pháp x, tìm ra câu tiếng Anh tốt nhất y.
   * Ta muốn chọn câu tiếng Anh y có xác suất cao nhất khi biết câu tiếng Pháp x.

* Áp dụng định lý Bayes, ta có:
* Vì P(x) không phụ thuộc vào y, nên ta bỏ qua:
* Hai thành phần chính:
  + Translation Model **:**
    - Mô hình dịch: học cách các từ/ngữ trong tiếng Anh y được dịch sang tiếng Pháp x
    - Đảm bảo tính chính xác (fidelity).
    - Được học từ **song ngữ (parallel data)**: cặp câu Pháp–Anh song song.
  + Language Mode **:**
    - Mô hình ngôn ngữ: học cách viết câu tiếng Anh sao cho tự nhiên, trôi chảy.
    - Đảm bảo **tính lưu loát (fluency)**.
    - Được học từ **dữ liệu đơn ngữ (monolingual data)**, ví dụ kho văn bản tiếng Anh.

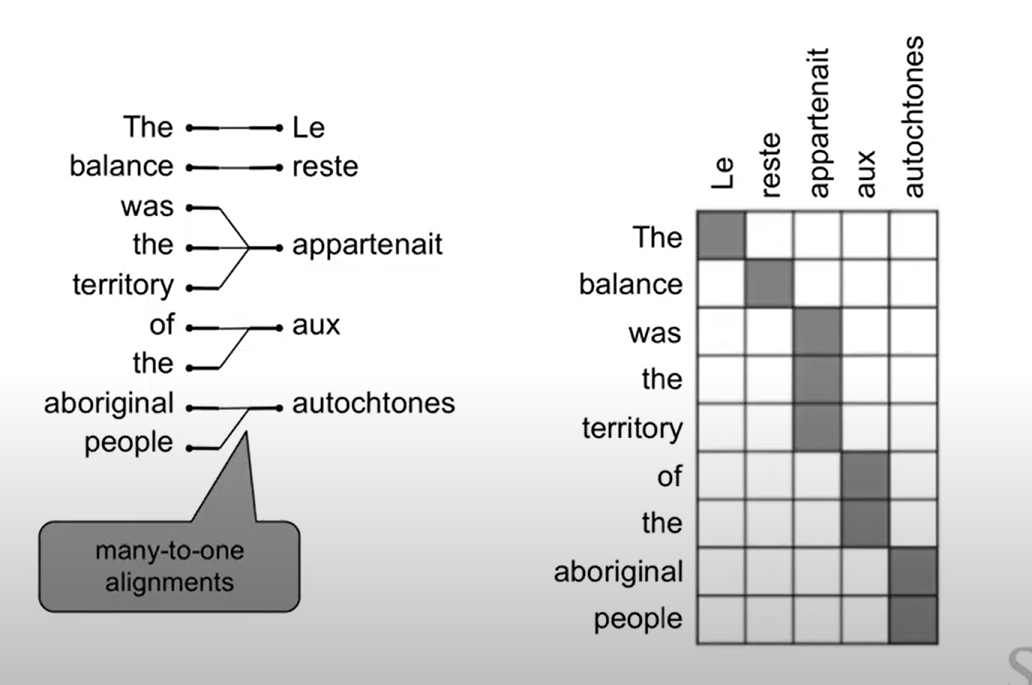
1. **How to learn translation model :**
   * Chúng ta cần số lượng lớn dữ liệu song ngữ (parallel data).
   * Với các cặp câu song ngữ, chúng ta không biết được từ nào khớp với từ nào, nên cần thêm biến ẩn .
   * chính là **alignment** (căn chỉnh), nghĩa là ánh xạ giữa từ/ngữ trong câu nguồn x và câu đích y.

****

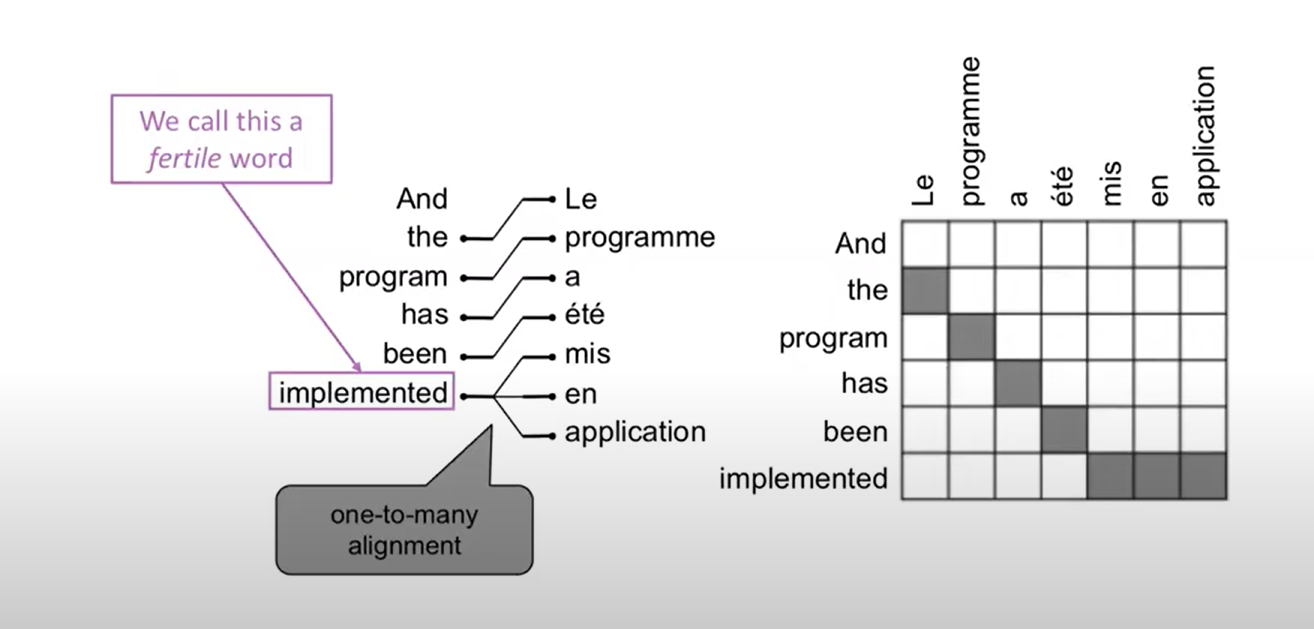
* + Sự khác biệt về loại hình ngôn ngữ khiên việc căn chỉnh không đơn giản (Ví dụ như trật tự thay đỏi khi dịch từ tiếng anh sang pháp)
  + Hay một số từ không có từ “đối ứng” ở ngôn ngữ kia, gọi là “**suprious word**” (từ thừa).

****

* + Alignment có thể là quan hệ many-to-one.

****

* + Alignment có thể là quan hệ one-to-many.

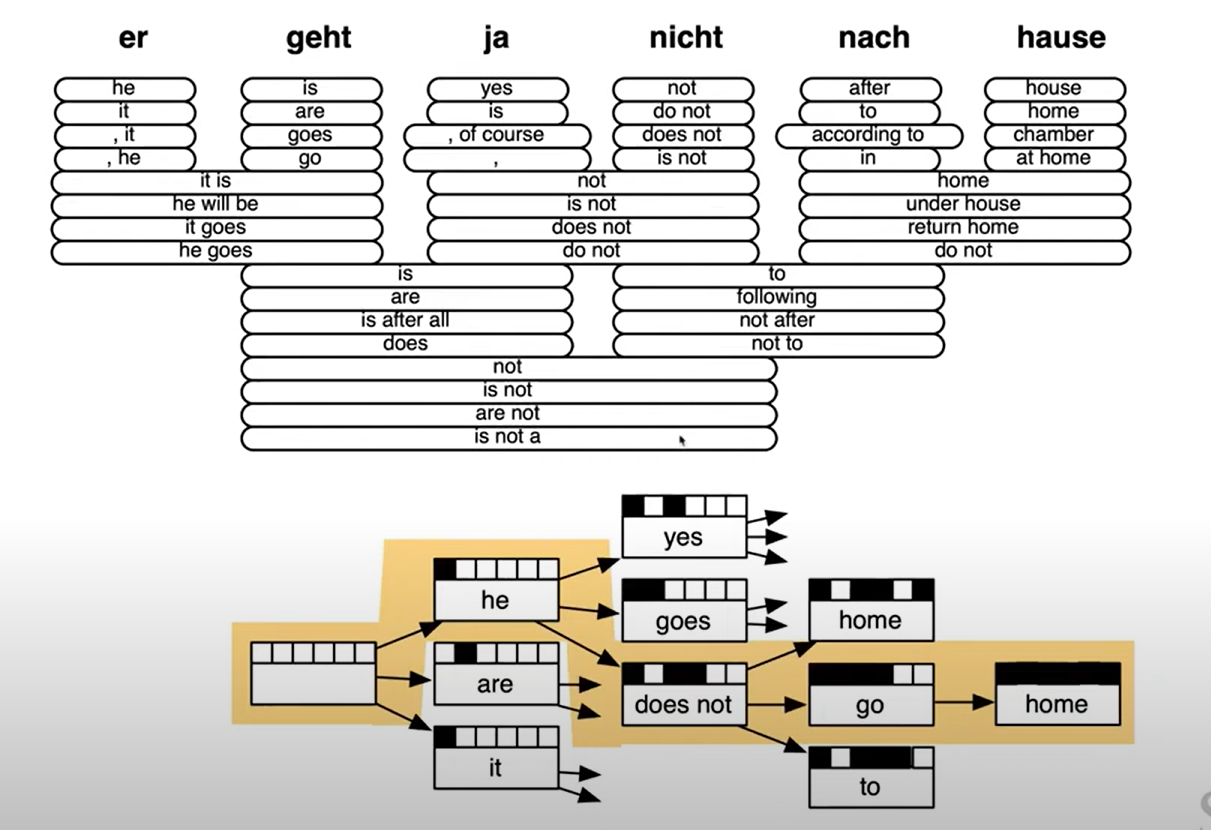
****

* + Alignment có thể là quan hệ many-to-many.



* + Vì alignment không có nhãn, ta phải dùng các thuật toán học với biến ẩn, tiêu biểu là **EM algorithm**.
  + EM giúp ước lượng tham số của phân phối khi có biến ẩn.

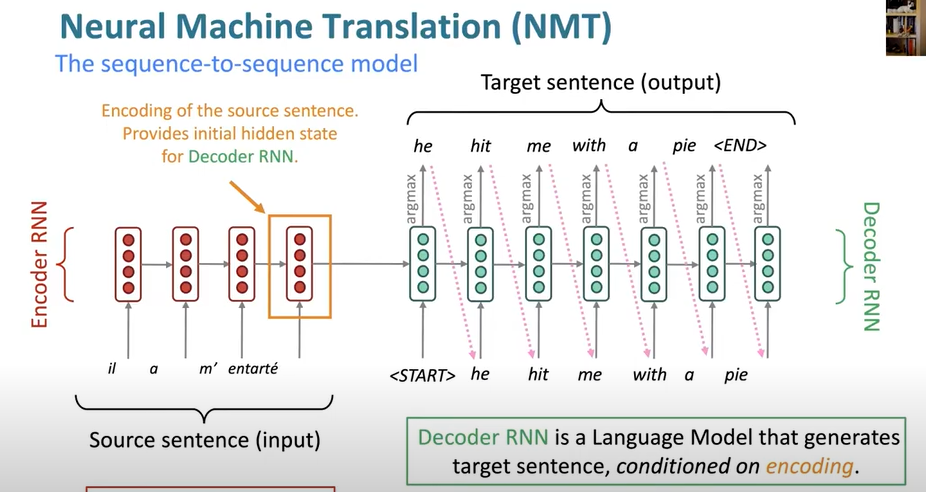
1. **Decoding for SMT**
   * **Translation Model (TM)**: cho biết câu nguồn → các từ/cụm từ đích có thể là gì, với xác suất bao nhiêu.
   * **Language Model (LM)**: cho biết câu đích nào mượt và tự nhiên hơn.
   * **Decoder**: tìm kiếm trong không gian các bản dịch khả dĩ, chọn câu tối ưu.
   * **Thay vì liệt kê hết trường hợp (quá đắt), đặt giả định từ dịch độc lập với nhau (giúp đơn giản hóa mô hình), sử dụng dynamic programming (ví dụ:** Viterbi algorithm) **để tìm nghiệm tối ưu toàn cục.**



* + - * SMT là một hệ thống rất phức tạp, có hàng trăm chi tiết nhỏ (chọn mô hình, smoothing, alignment, decoding,…)
      * Phải biên soạn **phrase tables**, dictionaries, song song corpora, reordering table, mỗi ngôn ngữ mới lại phải làm lại, với mỗi cặp ngôn ngữ lại phải build và tinh chỉnh hệ thống riêng.

1. **Neural Machine Translation (2014)**
2. **Giới thiệu**

* Đây là phương pháp dịch máy sử dụng một mạng neural duy nhất, nghĩa là thay vì các bước rời rạc như SMT, NMT gom tất cả vào một mạng neural duy nhất.
* Kiến trúc sequence-to-sequence (Seq2Seq) là kiến trúc mạng neural phổ biển trong NMT, nó sử dụng 2 RNNs.
  + **Encoder RNN:** đọc chuỗi đầu vào (câu nguồn) và nén nó thành một vector ngữ cảnh (context vector).
  + **Decoder RNN:** dùng vector này để sinh chuỗi đầu ra (câu đích) từng bước một.

****

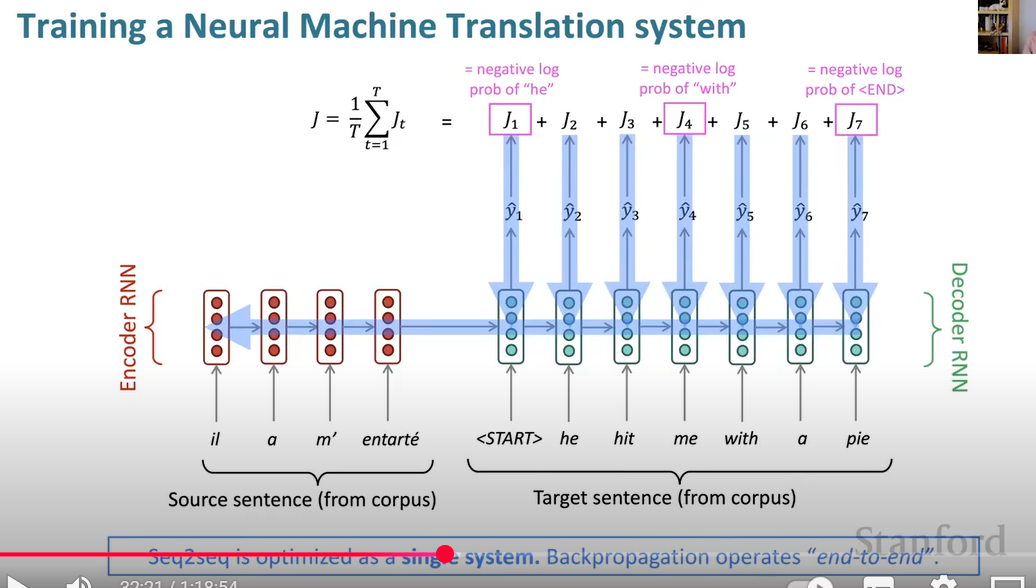
1. **Tính linh hoạt của Seq2Seq**

Rất nhiều task có thể được xử lý bằng Seq2Seq:

* **Tóm tắt văn bản (Summarization)**
* **Hội thoại (Dialogue):** chuỗi đầu vào là các câu nói trước đó trong cuộc hội thoại (previous utterances), và chuỗi đầu ra là câu trả lời tiếp theo (next utterance).
* **Phân tích cú pháp (parsing):** Bạn nhập vào một chuỗi (câu văn), và mô hình sẽ tạo ra một chuỗi đầu ra biểu diễn cấu trúc ngữ pháp của câu đó (output parse as sequence).
* **Code generation**

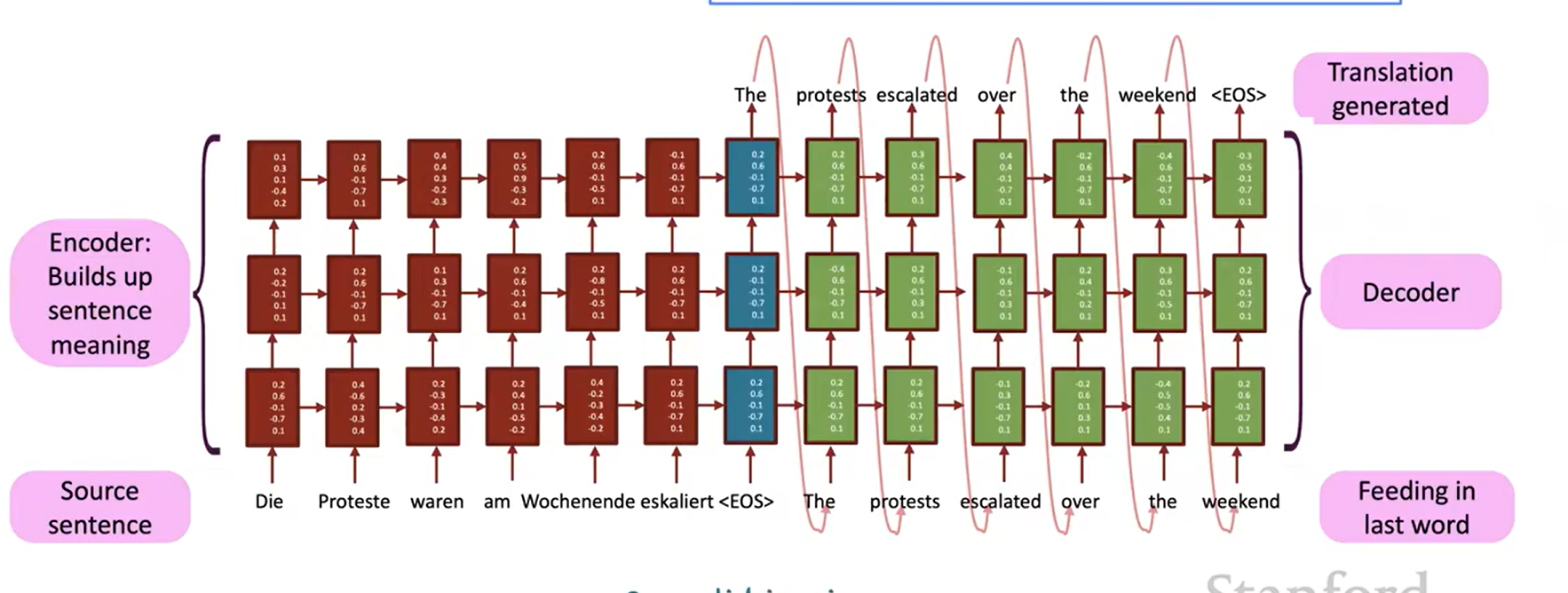
1. **Seq2Seq là Conditional Language Model và cách train NMT**

* **Language Model:** vì decoder của seq2seq có nhiệm vụ dự đoán từ tiếp theo trong câu đích y.
* **Conditional:** khác với language model thông thường (chỉ dự đoán từ tiếp theo dựa vào các từ trước đó trong cùng ngôn ngữ), seq2seq còn dựa vào câu nguồn x để dự đoán.
* Công thức tính xác suất của toàn bộ câu đích y, khi biết câu nguồn x:
* Quá trình tối ưu trong NMT cũng chạy ngược (backpropagation) như mọi mạng neural khác.
* Xác suất từ đó xuất hiện cao, thì J càng thất, mô hình càng tối ưu.

****

1. **Multi-layer RNNs**

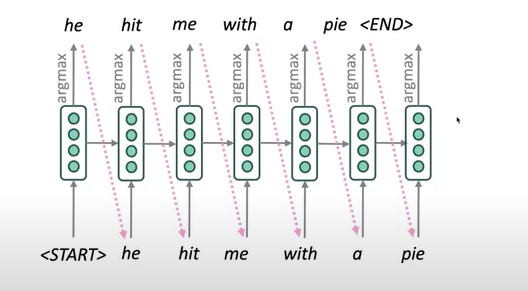
* Ngoài chiều thời gian, ta có thể xếp nhiều lớp RNN chồng lên nhau, gọi là multi-layer RNN hay stacked RNN.
* Điều này cho phép mạng học được biểu diễn phức tạp hơn:
  + **Các lớp thấp:** trích xuất đắc trưng cấp thấp. Ví dụ: quan hệ ngữ pháp cục bộ, mấu ngắn hạn.
  + **Các lớp cao:** trích xuất đặc trưng cấp cao hơn. Ví dụ: ý nghĩa, quan hệ dài hạn.

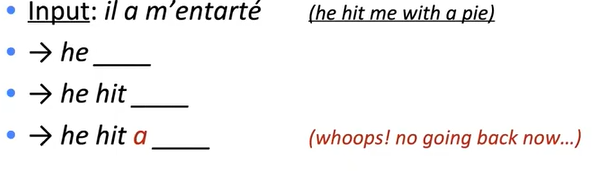
****

* Theo kết quả nghiên cứu thì với NMT, Encoder RNN tốt nhất từ 2 đến 4 layers, Decoder RNN tốt nhất là 4 layers.

1. **Mô hình dự đoán**
2. **Greedy decoding**

* Tại mỗi bước, decoder tính ra phân phối xác suất trên toàn bộ từ vựng
* Greedy decoding = chọn từ có xác suất cao nhất (argmax) ngay lập tức
* Điều này sinh ra một số vấn đề như:
  + Quá tham lam (greedy): chỉ nhìn từng bước, không xét đến toàn bộ câu.
  + Có thể chọn từ tốt nhất tại thời điểm đó nhưng dẫn đến câu tổng thể kém tự nhiên hoặc sai nghĩa.

****

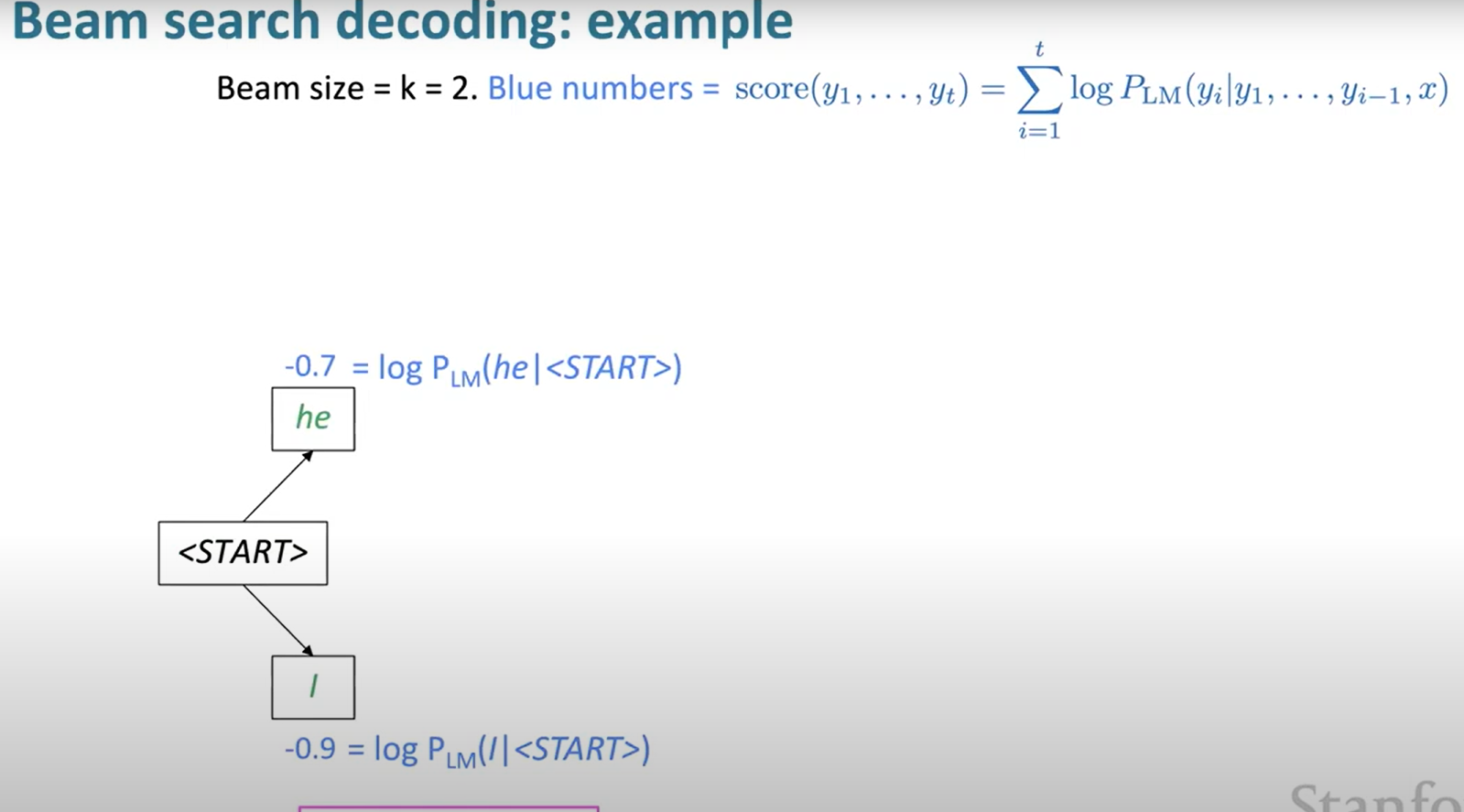
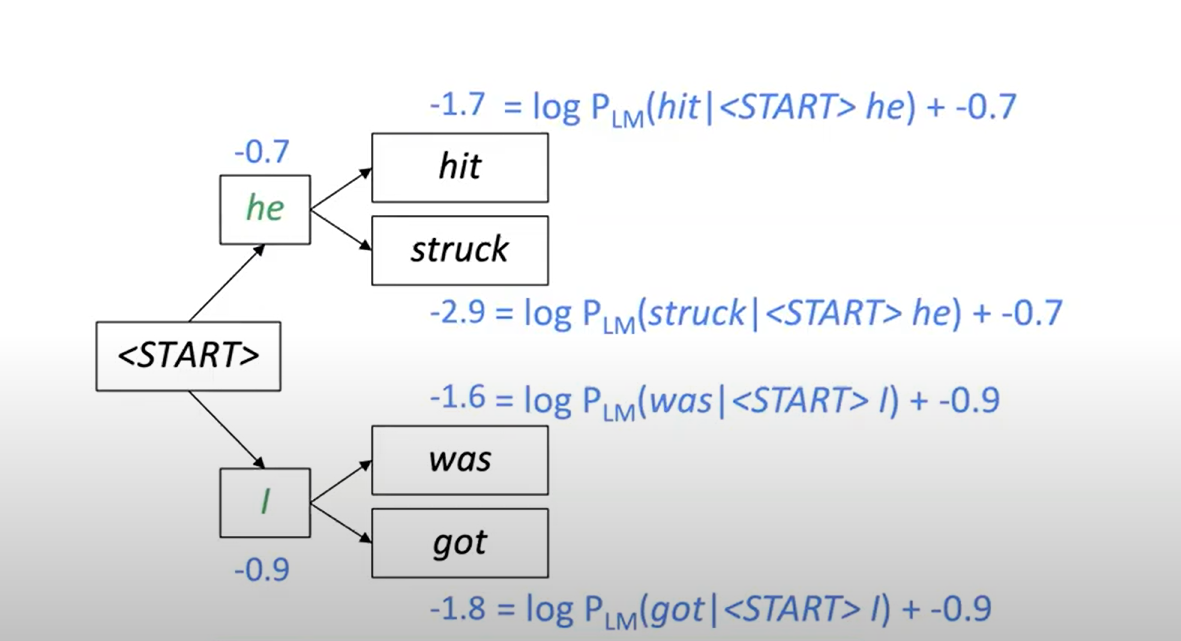
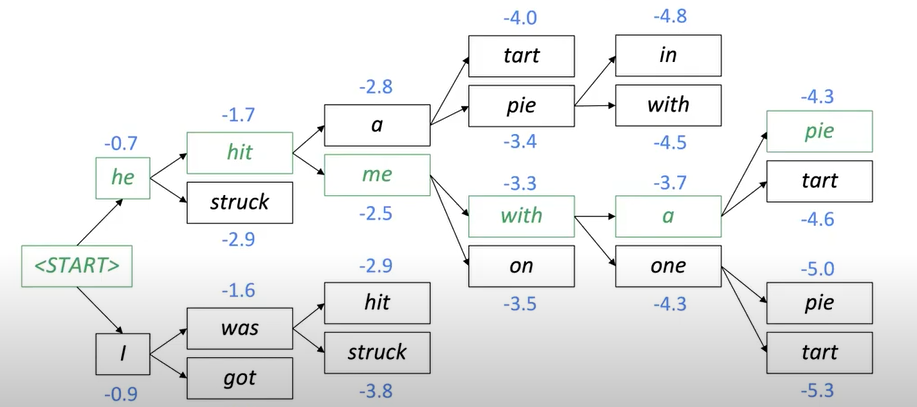
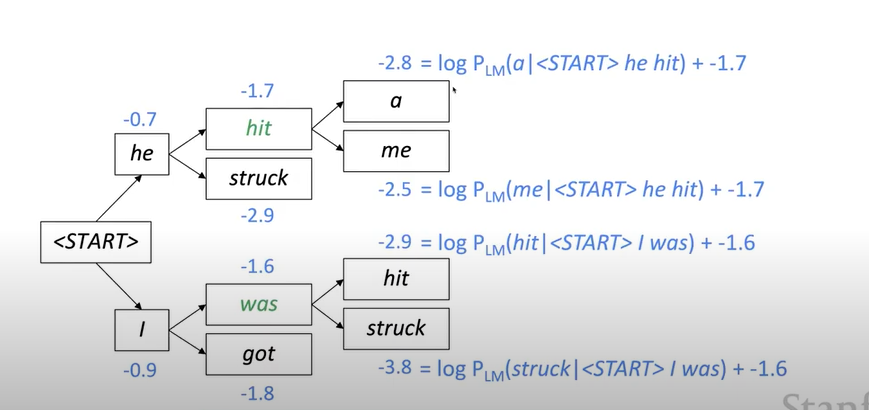
****

1. **Exhaustive search (tìm kiếm toàn bộ)**

* Chúng ta muốn dịch một câu có độ dài T, ý tưởng là thử tất cả mọi câu y có thể, tính xác suất, rồi chọn câu có lớn nhất
* Với vocab size V, độ dài câu T thì số khả năng = .
* Exhaustive search đúng về mặt lý thuyết nhưng không khả thi (too expensive).

1. **Beam Search Decoding**

* Ý tưởng chính: Ở mỗi bước của decoder, thay vì chỉ chọn **1 từ xác suất cao nhất** (greedy) hoặc duyệt hết tất cả (exhaustive), ta giữ lại **k câu khả dĩ nhất** (gọi là hypotheses).
* Mỗi hypothesis có điểm só là log xác suất:
* Ưu & nhược điểm:
  + Hiệu quả hơn exhaustive search rất nhiều
  + Cho kết quả tốt hơn greedy (tránh mắc kẹt ở lựa chọn sớm sai)
  + Không đàm bảo tìm ra nghiệm tối ưu toàn cục
  + Càng tăng beam size thì kết quả càng tốt, nhưng tốc độ chậm lại.

********

* Điều kiện dừng:
  + Trong greedy decoding, nó sẽ dừng ngay khi sinh ra <END>.
  + Trong Beam Search Decoding thì khác, nó có nhiều giả thuyết chạy song song, mỗi giả thuyết có thể sinh <END> ở các thời điểm khác nhau, có 2 cách phổ biến để ép Beam Search dừng hẳn:
    - Khi đạt đến timestep T (giả sử mình đặt giới hạn câu dịch là 50 từ, thì khi đến 50 từ nó sẽ dừng)
    - Khi đã có ít nhất n giả thuyết hoàn chỉnh.
* Trong cách tính điểm, vì xác suất luôn nên với các câu dài thì tổng điểm luôn thấp hơn các câu ngắn, điều này dẫn đến mô hình bị thiên vị:

1. **Ưu nhược điểm của NMT**
2. **Ưu điểm**

* Hiệu suất tốt hơn:
  + Trôi chảy hơn
  + Sử dụng ngữ cảnh tốt hơn
  + Sử dụng tốt hơn các cụm từ tương đồng
* Một mạng lưới duy nhất được tối ưu từ đầu đến cuối, không có các thành phần phụ phải tối ưu hóa riêng lẻ
* Yêu cầu ít công sức kỹ thuật từ con người hơn
  + Không cần thiết kế các đặc trưng
  + Cùng một phương pháp cho tất cẩ các ngôn ngữ, chỉ cần cung cấp đủ dữ liệu huấn luyện.

1. **Nhược điểm**

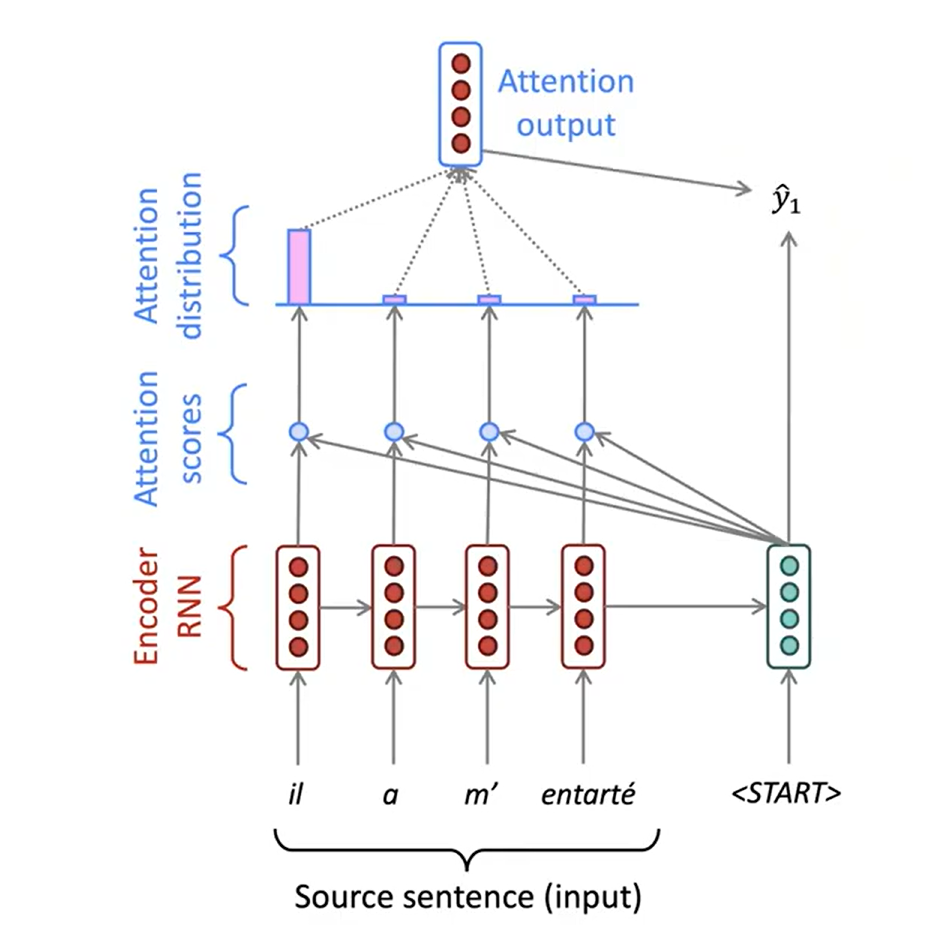
* NMT khó diễn giải điều này khiến việc debug trở nên khó khăn
* NMT khó kiểm soát dẫn đến mối lo ngại về an toàn.

1. **Attention**
2. **Vấn đề Bottlenck**

* Trong các mô hình Seq2Seq cổ điển, tất cả thông tin của câu nguồn bị ép vào một vector duy nhất.
* Nếu câu quá dài 🡪 vector này không thể giữ đủ thông tin 🡪 dịch bị sai hoặc mất ý.

1. **Attention**

* Ý tưởng chính: tại mỗi bước của bộ **giải mã (decoder)**, thay vì chỉ dựa vào một vector ngữ cảnh cố định, nó sẽ tạo ra một **kết nối trực tiếp** với bộ **mã hóa (encoder)** để **tập trung vào một phần cụ thể** của chuỗi nguồn (chuỗi đầu vào).

****

* Chúng ta có các encoder hidden states
* Ở timestep t, decoder có hidden state
* Tính Attention scores bằng dot:
* Dùng softmax để chuẩn hóa thành một attention distribution:
* Attention output được tính bằng công thức, hidden state đóng góp nhiều hay ít sẽ được đánh dấu.:
* Ghép với decoder hidden sate :

1. **Ưu điểm vượt trội của Attention**

* Cải thiện hiệu quả dịch tự động
* Cơ chế attention giống cách con người dịch: khi dịch một câu, ta thường nhìn lại câu gốc thay vì cố nhớ hết.
* Giải quyết the bottleneck problem (vấn đề nghẽn cổ chai - vì toàn bộ thông tin phải nến vào một vector duy nhất), attention cho phép decoder truy cập trực tiếp encoder states, bỏ quả được nút thắt này.
* Giúp tránh được vanishing gradient problem, attention cung cấp **đường tắt (shortcut)** để truyền tín hiệu trực tiếp từ decoder đến các encoder states xa → giảm mất mát gradient.
* Attention cung cấp cho ta một alignment (căn chỉnh) giữa source và target gần như miễn phí mà không cần huấn luyện thủ công như trước.

1. **Một số biến thể của attention**

Cụ thể là các cách tính khác để tính attention score:

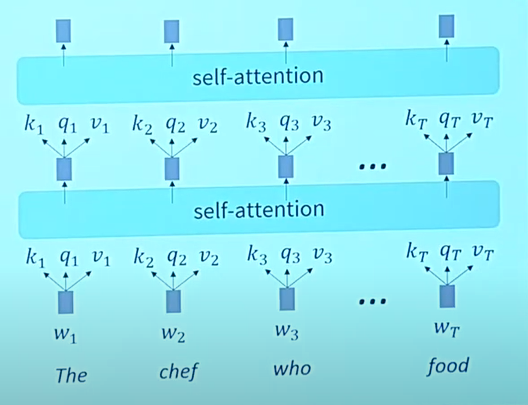
* Multiplicative attention:
  + Thêm trọng số
  + Giúp điều chỉnh s và khác chiều hoặc cần học cách so sánh phức tạp hơn.
* Reduced rank multiplicative attention:
  + giảm chiều để tính dot product rẻ hơn, với .
* Additive attention:

1. **Self-Attention**
2. **Giới thiệu**

* **Attention cổ điển:** Query đến từ decoder, còn Key và Value đến từ encoder.
* **Self-Attention:** Query, Key, Value đều xuất phát từ cùng một chuỗi (ví dụ: câu nguồn trong encoder).
  + Mỗi từ trong câu vừa “hỏi” (Query), vừa “đưa chìa khóa” (Key), vừa “cung cấp thông tin” (Value).
  + Điều này cho phép mô hình học được quan hệ phụ thuộc giữa các từ trong cùng một câu.

1. **Thuật toán Self-Attention**

* Attention hoạt động dựa trên các query, key, value:
  + Chúng ta có một số query . Mỗi query
  + Chúng ta có một số key . Mỗi key
  + Chúng ta có một số value . Mỗi value
* Các query, key và value được lấy từ một nguồn cung.
* Tính sự tương đồng giữa query (từ đang xét) và key (tất cả các từ trong chuỗi, bao gồm cả từ đang xét) :
* Sau đó ta tính được phân bố xác suất bằng hàm softmax:
* Tính vector đầu ra cho từ thứ i:

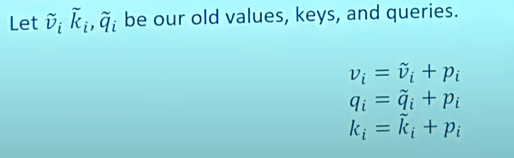
****

1. **Hạn chế của self-attention**
   * 1. **Không có khái niệm thứ tự**

* Self-attention không thể thay thế hoàn toàn RNN vì nó không có khái niệm thứ tự, không biết từ nào đứng trước, từ nào đứng sau**.**

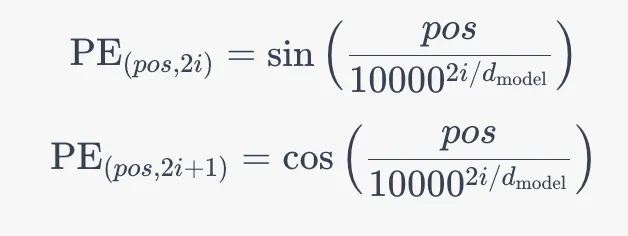
**Giải pháp**

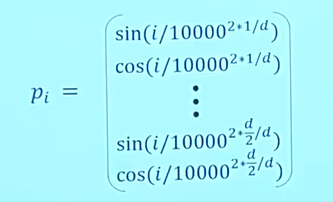
* Gán cho mỗi từ i một vector vị trí

****

* Positional encoding thường được cộng ngay lớp đầu tiên, có thể cộng thực tiếp hoặc nối (concatnate).

1. **Biểu diễn vị trí bằng hàm sin/cos**

****

****

**Ưu điểm:**

* Không cần học, giảm số tham số.
* Tính chu kỳ giúp mô hình không chỉ học vị trí tuyệt đối mà còn học quan hệ tương đối.

**Nhược điểm:**

* Vector này **không học được** (fixed) (vị trí cứng nhắc, không thể thay đổi), nên mô hình không thể tối ưu theo dữ liệu.
  + - 1. **Mô hình tự học vector vị trí**
* Cho mô hình tự học vector vị trí trong quá trình training:
  + Ta học một ma trận
  + Trong đó cột chính là vector vị trí ứng với vị trí i, và ta có T từ.

**Ưu điểm:**

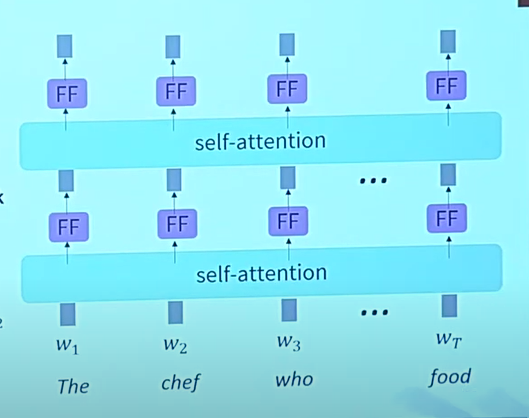
* Mô hình có thể học biểu diễn vị trí tối ưu theo dữ liệu thực tế (thay vì bị ràng buộc bởi sin/cos).
* Đơn giản để triển khai.

**Nhược điểm:**

* Không thể extrapolate (ngoại suy) ra ngoài phạm vi đã học.
  + Ví dụ: nếu trong training chỉ thấy câu dài ≤ 512, thì khi test với câu dài 1000 từ, mô hình sẽ không có vector vị trí cho các từ > 512.
    1. **Không có phi tuyến tính**
* Self-attention bản chất chỉ là tổ hợp tuyến tính, và việc chồng nhiều lớp chỉ là việc trộn tuyến tính, nó thiếu phi tuyến tính để biểu diễn các quan hệ phức tạp.

**Giải pháp:**

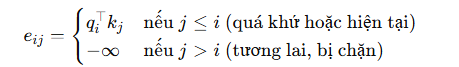
* Thêm một Feed-Forward Netword (FFN) để xử lý từng vector output.
* Công thức:
* FFN được định nghĩa bằng 2 lớp Linear, với W1 là trọng số lớp 1 và W2 là trọng số lớp 2.

****

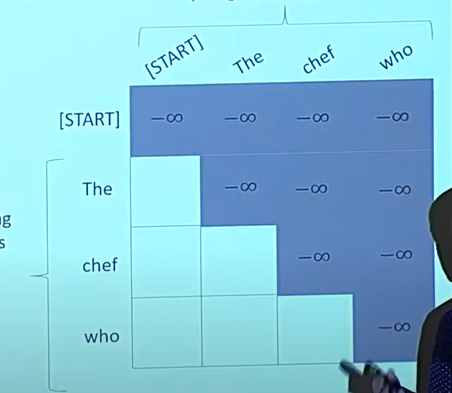
* + 1. **Cần đảm bảo decoder không nhìn thấy các từ tương lai**
* Trong decoder self-attention: ta sinh từ theo thứ tự, nên khi dự đoán từ tiếp theo thì không được nhìn thấy các từ tương lai (chưa sinh ra).

**Giải pháp: Masking**

Công thức:

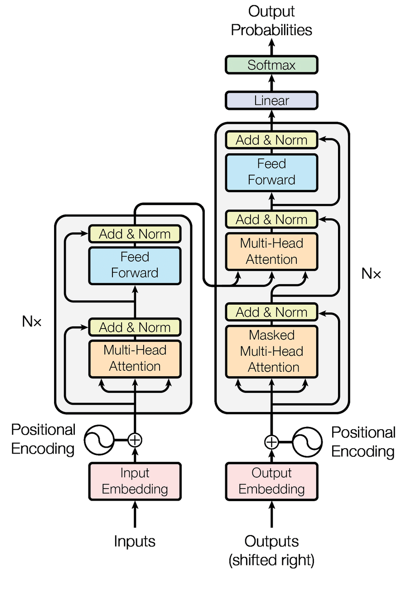


* Khi đưa vào softmax, các giá trị sẽ thành 0 🡪 mô hình không chú ý được tới từ tương lai.

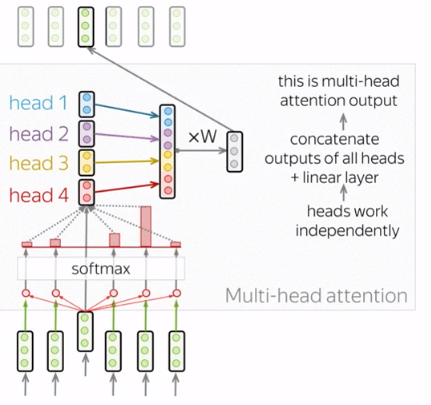


1. **Transformer**

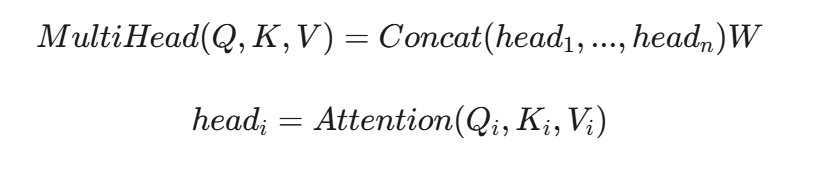
* Trong bài báo “attention is all you need”, cấu trúc transformer chỉ sử dụng attention, với encoder có Nx=6 layers chồng lên nhau, decoder cũng có Nx=6 layers chồng lên nhau.



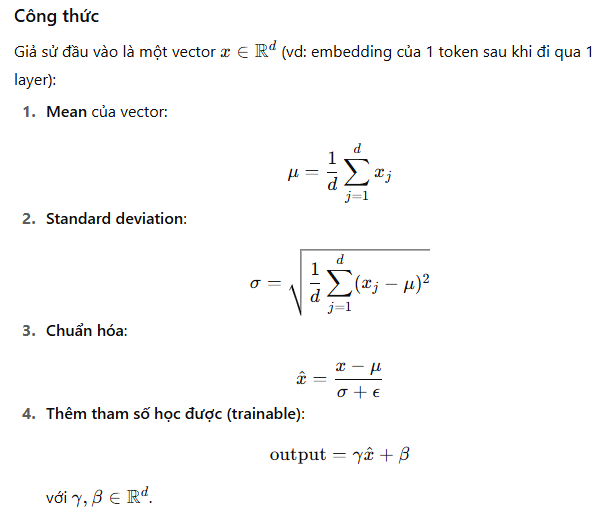
* + - * 1. **Multi-Head Attention**
* Với một head self-attention chỉ học được một hiểu quan hệ (ví dụ: quan hệ cú pháp, quan hệ từ gần, …)
* Thay vì chỉ có 1 head, ta chạy nhiều head song song, mỗi head sẽ học một cách “chú ý” riêng



**Công thức:**



* + - * 1. **Add & Norm**
* **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**: sau khi thực hiện tính toán từ lớp trước, khi đến residual nó sẽ cộng lại input ban đầu để đảm bảo mô hình vẫn giữ nguyên được thông tin gốc input ban đầu 🡪 tránh vanishing gradient khi chồng nhiều lớp.
  + - * 1. **Scaled Dot Product**
* Khi chiều không gian d lớn 🡪 giá trị dot product thường rất lớn
* Softmax khi nhận input lớn sẽ bị cực trị hóa, khiến gradient nhỏ đi 🡪 vanishing gradient khi backpropagation)
* Người ta chia cho với h là số head khi dùng multi-head.

****

* + - * 1. **Nhược điểm**
* Độ phức tạp tính toán, tổng số phép toán của Self-attention tỉ lệ với :

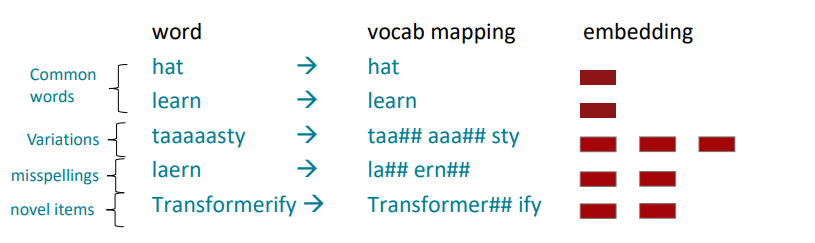
Trong đó:

* T = độ dài chuỗi
* d = số chiều của embedding
* Như thế trong thực tế thì chi phí sẽ rất lớn.

1. **Pretraining**
2. **Subword Modeling (Mô hình hóa từ con)**
3. **Hạn chế của fixed vocabulary (từ vựng cố định)**

* Từ vựng cố định (fixed vocabulary) trong NLP truyền thống giới hạn ở hàng chục nghìn từ, dẫn đến vấn đề với từ mới hoặc ngôn ngữ có hình thái phức tạp (ví dụ: Swahili với hàng trăm biến thể động từ). Từ lạ, lỗi chính tả được thay bằng "UNK" (unknown), làm giảm hiệu suất.

1. **Byte-Pair Encoding algorithm (BPE)**
2. Bắt đầu với một từ vựng chỉ chứa các ký tự và một ký hiệu "kết thúc từ" (end-of-word)
3. Sử dụng một kho văn bản (corpus of text), tìm cặp ký tự liền kề "a,b" phổ biến nhất, thêm "ab" làm một từ con (subword).
4. Thay thế các trường hợp của cặp ký tự đó bằng từ con mới; lặp lại cho đến khi đạt được kích thước từ vựng mong muốn.



1. **Hạn chế của word2vec và động lực cho pretraing**
2. **Hạn chế của word2vec**

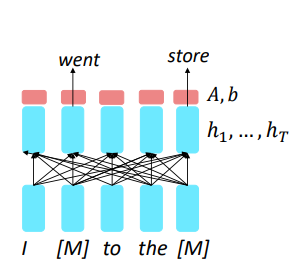
* Word2Vec (hoặc GloVe) gán một vector cố định cho mỗi từ, không phụ thuộc vào ngữ cảnh. Ví dụ:
  + Từ "record" có cùng embedding dù là danh từ ("a record") hay động từ ("to record").
  + Điều này gây khó khăn trong việc hiểu ý nghĩa đa dạng của từ trong các ngữ cảnh khác nhau.

1. **Pretraining**

* Thay vì chỉ học embedding cho từng từ, pretraining học **toàn bộ tham số của mô hình** (bao gồm cả embedding và các layer Transformer) trên dữ liệu lớn.
* Pretraining sử dụng các mục tiêu tự giám sát (self-supervised objectives) như:
  + **Masked Language Modeling (MLM)**: Che một phần input và dự đoán (dùng trong BERT).
  + **Autoregressive Language Modeling**: Dự đoán từ tiếp theo (dùng trong GPT).
* Pretraining giải quyết rất tốt **vấn đề dữ liệu nhỏ**, bằng cách sử dụng dữ liệu gắn nhãn khổng lồ (như Wikipedia, web crawl, ..) để học, sau đó mô hình được finetuning trên dữ liệu nhỏ hiệu quả hơn.

1. **Pretraining cho 3 kiểu kiến trúc**
2. **Pretraining Encoders**

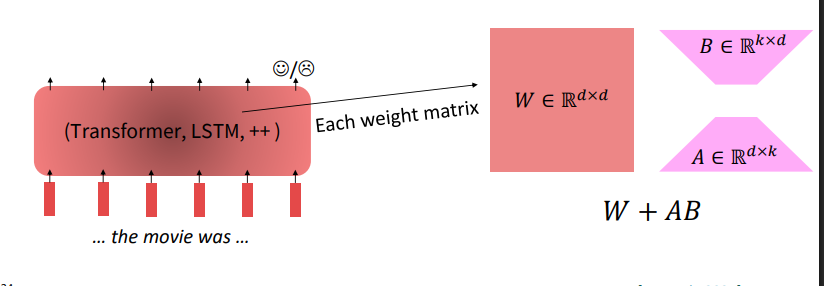
* Encoders là bidirectional (một token nhìn được cả trái lẫn phải), vì thế mô hình không thể học theo kiểu truyền thống là next word prediction.
* Ý tưởng: che (mask) một số từ trong câu, rồi yêu cầu mô hình dự đoán lại.



* Tiêu biểu là mô hình **BERT**, dùng **Masked LM** làm pretraining objective, chi tiết cách mask trong BERT như sau:
* Với ~15% số token:
  + 80% → thay bằng [MASK] (vd: *I [MASK] to store*).
  + 10% → thay bằng một token ngẫu nhiên khác (vd: *I pizza to store*).
  + 10% → giữ nguyên token (vd: *I went to store*), nhưng vẫn bắt mô hình dự đoán lại vị trí đó.
* Nếu **luôn dùng [MASK]**, mô hình sẽ quen với việc thấy [MASK] → không giống dữ liệu thực tế lúc finetuning (khi inference không hề có [MASK]).
* Cách trộn ngẫu nhiên này giúp mô hình học representation tốt hơn và không bị phụ thuộc quá nhiều vào token [MASK].
* Một số ứng dụng của BERT:
  + **QQP**: phát hiện câu hỏi có cùng nghĩa.
  + **QNLI**: kiểm tra suy luận (liệu câu trả lời có phù hợp với câu hỏi).
  + **SST-2**: phân tích cảm xúc (sentiment analysis).
  + **CoLA**: kiểm tra tính ngữ pháp.
  + **STS-B**: so sánh mức độ giống nhau về ngữ nghĩa.
  + **MRPC**: phát hiện paraphrase (câu có cùng nghĩa).
  + **RTE**: suy luận ngôn ngữ quy mô nhỏ.
* **Một số cách fine-tuning:**
  + **Prefix-tuning: ta freeze toàn bộ mô hình gốc, sau đó thêm một chuỗi vector đặc biệt gọi là prefix (có thể coi như prompt learnable). Ví dụ, vector ở prefix dưới đây sẽ là encode thông tin kiểu, dịch Anh-Pháp:**

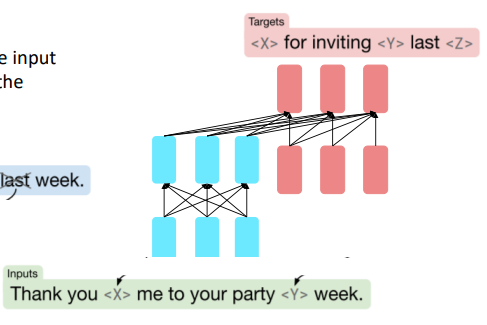
[Prefix] I love apples 🡪 J’aime les pommes

* + **Low-Rank Adaptation (LoRa):**
    - Thay vì fine-tune full ma trận lớn, ta chỉ học một nhiễu low-rank (hai ma trận nhỏ), giúp tiết kiệm nhưng vẫn giữ được hiệu quả.

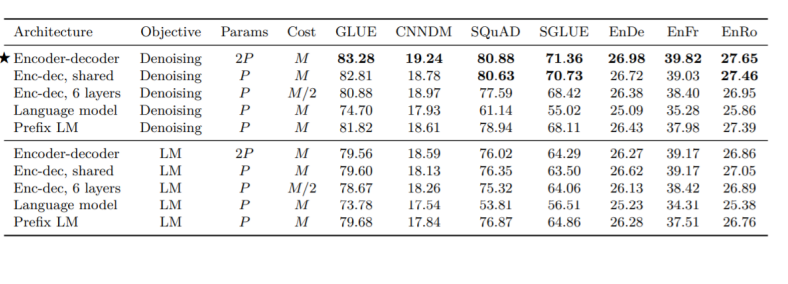


1. **Pretraining encoder-decoders**

* Encoder học representation mạnh, còn decoder học sinh chuỗi như LM, nhận thấy cách tốt nhất không phải LM thuần mà là **span corruption:**
  + Xóa bỏ (mask) một đoạn liên tiếp (span) trong câu.
  + Thay thế đoạn đó bằng một token đặc biệt như <x>.
  + Mục tiêu của decoder: sinh lại chính xác đoạn bị che đi.



* Với các task khác nhau, Denoising (span corruption) cho kết quả tốt hơn LM truyền thống.

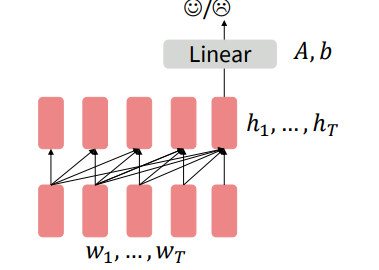


1. **Pretraining Decoders**

* Decoder được huấn luyện như một language model (LM), tức là nó học phân phối xác suất:

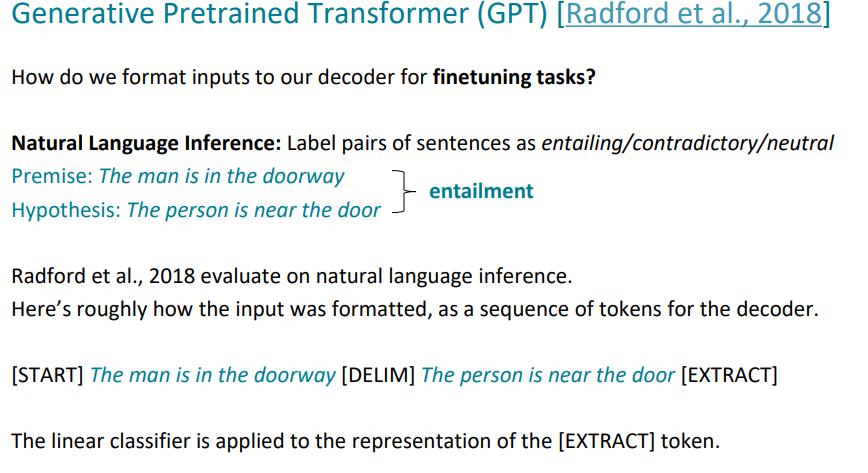
🡪Dự đoán từ tiếp theo dựa vào các từ trước đó.

* Khi fine-tune cho downsteam task cụ thể, giả sử là bài toán phân loại văn bản, thì ta dùng hidden state cuối cùng và gắn thêm một linear classifier:

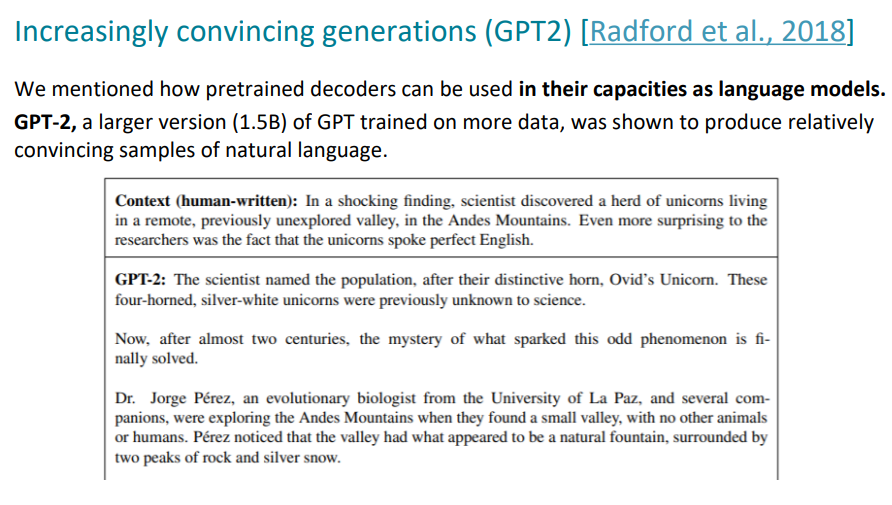


* Với task sinh chuỗi, các từ đi qua decoder, tạo ra dãy hidden states, hidden state ở bước t-1 được đưa qua linear layer với trọng số A, b sau đó được đưa qua hàm softmax và dự đoán từ tiếp theo .

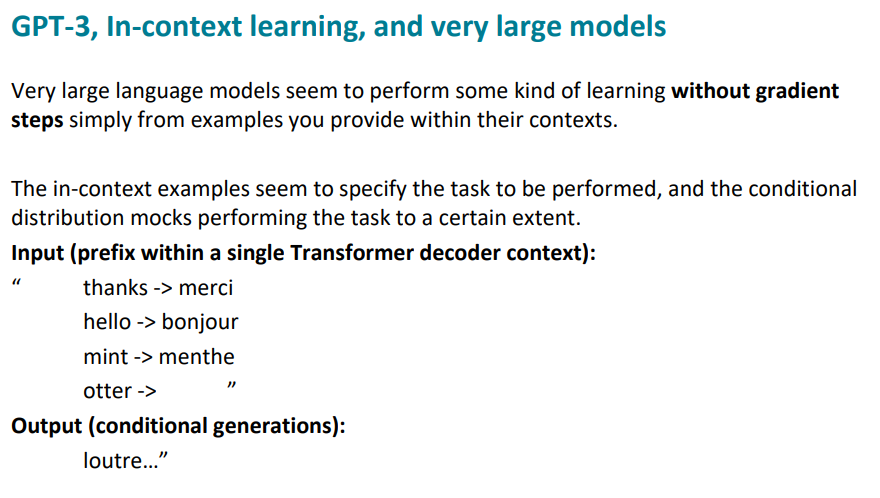
GPT-1:

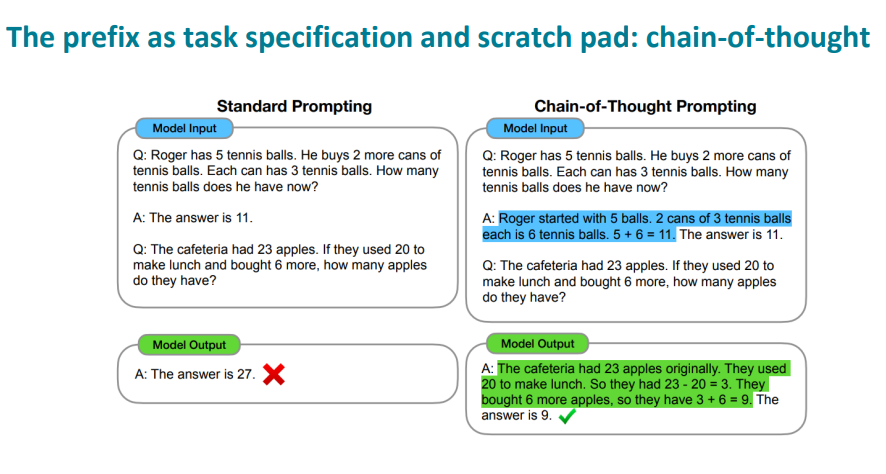
****

**GPT-2:**

****

**GPT-3:**

****

****

**Cơ chế của transfomer:**

* Encoder học hiểu ngữ cảnh, ngữ pháp của toàn bộ đầu vào.
* Decoder sẽ có attention Q hỏi nên sinh gì tiếp theo, sau đó nó sẽ lấy K và V của encoder để sinh từ tiếp theo.
* Hiểu ví dụ này:

Input:

“The cat sat on the mat.”

Output (đang sinh):

“Con mèo …”

* Lúc này Q của decoder sẽ hỏi là sinh gì tiếp theo đây, còn K và V của decoder chưa toàn bộ nội dung đã sinh. Lúc này Cross entropy sẽ cung cấp K và V của encoder là từ phù hợp nhất tiếp theo, và nó sẽ sinh ra từ đó. Q của encoder là nên chú ý đến ai trong input này, nó giống cơ chế chính của attention là chú ý.