TRƯỜNG ĐẠI HỌC SÀI GÒN

KHOA CÔNG NGHỆ THÔNG TIN

HỌC PHẦN: XỬ LÝ NGÔN NGỮ TỰ NHIÊN

ĐỀ TÀI: DỊCH MÁY ANH-PHÁP / ANH-ĐỨC VỚI MÔ HÌNH ENCODER-DECODER LSTM

Giáo viên hướng dẫn:

PGS. TS. Nguyễn Tuấn Đăng

Sinh viên thực hiện:

3123410055 – Mai Thanh Duy

3123410070 - Nguyễn Tuấn Đạt

Thành phố Hồ Chí Minh, tháng 11 năm 2025

Lời mở đầu

Trong bối cảnh toàn cầu hóa ngày càng phát triển, nhu cầu giao tiếp và trao đổi thông tin giữa các ngôn ngữ khác nhau trở nên cấp thiết hơn bao giờ hết. Dịch máy (Machine Translation) đã trở thành một trong những ứng dụng quan trọng nhất của Xử lý ngôn ngữ tự nhiên (Natural Language Processing - NLP), giúp phá bỏ rào cản ngôn ngữ và kết nối con người trên toàn thế giới.

Từ những mô hình dịch máy thống kê truyền thống cho đến các mô hình học sâu hiện đại, lĩnh vực dịch máy đã có những bước tiến vượt bậc. Đặc biệt, sự ra đời của kiến trúc Encoder-Decoder dựa trên mạng LSTM (Long Short-Term Memory) đã mở ra một kỷ nguyên mới, cho phép mô hình học được các mối quan hệ phức tạp giữa ngôn ngữ nguồn và ngôn ngữ đích một cách hiệu quả hơn.

Đồ án này tập trung vào việc xây dựng một hệ thống dịch máy từ tiếng Anh sang tiếng Pháp sử dụng mô hình Encoder-Decoder LSTM. Thông qua việc triển khai từng bước từ tiền xử lý dữ liệu, xây dựng kiến trúc mô hình, huấn luyện cho đến đánh giá kết quả, đồ án không chỉ cung cấp cái nhìn sâu sắc về cơ chế hoạt động của mô hình sequence-to-sequence mà còn giúp hiểu rõ những thách thức thực tế trong việc xây dựng một hệ thống dịch máy.

Đồ án được thực hiện trên bộ dữ liệu Multi30K, một tập dữ liệu phổ biến trong nghiên cứu dịch máy, với các công cụ hiện đại như PyTorch, Spacy và các thư viện xử lý ngôn ngữ tự nhiên khác. Kết quả của đồ án sẽ được đánh giá thông qua các chỉ số chuẩn như BLEU score và Perplexity, đồng thời phân tích những hạn chế và đề xuất hướng cải tiến trong tương lai.

Qua đồ án này, em mong muốn không chỉ nắm vững kiến thức lý thuyết về mô hình Encoder-Decoder LSTM mà còn tích lũy kinh nghiệm thực tế trong việc triển khai một dự án NLP hoàn chỉnh từ đầu đến cuối.

I. GIỚI THIỆU

1.1. Tổng quan về bài toán dịch máy

Dịch máy (Machine Translation - MT) là một nhánh quan trọng của Trí tuệ nhân tạo và Xử lý ngôn ngữ tự nhiên (NLP), tập trung vào việc tự động chuyển đổi văn bản từ ngôn ngữ nguồn (source language) sang ngôn ngữ đích (target language) mà vẫn giữ nguyên ý nghĩa ngữ nghĩa.

Trong những năm gần đây, Dịch máy nơ-ron (Neural Machine Translation - NMT) đã đạt được những bước tiến vượt bậc, thay thế các phương pháp thống kê truyền thống. Cốt lõi của NMT là kiến trúc **Sequence-to-Sequence (Seq2Seq)**, cho phép mô hình ánh xạ một chuỗi đầu vào có độ dài tùy biến sang một chuỗi đầu ra có độ dài khác nhau.

Đồ án này tập trung vào kiến trúc **Encoder-Decoder** sử dụng mạng nơ-ron hồi quy **LSTM** (Long Short-Term Memory). Trong kiến trúc này:

- **Encoder:** Đọc và nén toàn bộ thông tin của câu nguồn thành một vector ngữ cảnh cố định (fixed-length context vector).
- **Decoder:** Sử dụng vector ngữ cảnh này để sinh ra từng từ của câu đích.

Mặc dù các mô hình hiện đại như Transformer đã ra đời, việc nghiên cứu và triển khai mô hình Encoder-Decoder LSTM từ đầu vẫn là nền tảng cốt lõi để hiểu sâu về cơ chế xử lý dữ liệu chuỗi trong NLP.

1.2. Mục tiêu của đồ án

Dựa trên yêu cầu của học phần Xử lý ngôn ngữ tự nhiên HK1/2025-2026, mục tiêu chính của đồ án là xây dựng và đánh giá một hệ thống dịch máy Anh-Pháp hoàn chỉnh. Các mục tiêu cụ thể bao gồm:

Hiểu và triển khai kiến trúc mạng nơ-ron: Xây dựng mô hình Encoder-Decoder sử dụng LSTM từ đầu (from scratch) mà không phụ thuộc vào các thư viện seq2seq cấp cao có sẵn.

Xử lý dữ liệu chuỗi: Thực hiện quy trình chuẩn bị dữ liệu bao gồm Tokenization, xây dựng bộ từ điển (Vocabulary), xử lý các token đặc biệt

(<sos>, <eos>, <unk>, <pad>) và kỹ thuật Padding/Packing cho các batch có độ dài khác nhau.

Huấn luyện và tối ưu hóa: Áp dụng các kỹ thuật như Teacher Forcing để tăng tốc độ hội tụ và sử dụng hàm mất mát CrossEntropyLoss để huấn luyện mô hình trên tập dữ liệu song ngữ.

Đánh giá hiệu suất: Sử dụng thang đo tiêu chuẩn **BLEU score** để đánh giá chất lượng bản dịch và thực hiện phân tích lỗi định tính trên các mẫu dự đoán thực tế.

1.3. Phạm vi và công cụ sử dụng

Phạm vi dữ liệu (Dataset): Đồ án sử dụng bộ dữ liệu **Multi30K**, tập trung vào cặp ngôn ngữ **Tiếng Anh - Tiếng Pháp (en-fr)**. Đặc điểm của tập dữ liệu này bao gồm:

- **Kích thước:** Tập huấn luyện (Train) khoảng 29,000 cặp câu, tập kiểm thử (Test) và tập xác thực (Validation) mỗi tập khoảng 1,000 cặp câu.
- **Đặc thù:** Các câu ngắn gọn (trung bình 10-15 từ), phù hợp để huấn luyện trên tài nguyên phần cứng cơ bản.

Phạm vi mô hình:

- **Kiến trúc:** Encoder-Decoder với Context Vector cố định.
- **Cấu hình:** Sử dụng LSTM 2 lớp (2 layers), kích thước vector ẩn (Hidden size) là 512, và kích thước Embedding là 256.
- **Giới hạn từ vựng:** 10,000 từ phổ biến nhất cho mỗi ngôn ngữ.

Công cụ và Môi trường thực hiện: Đồ án được triển khai trên môi trường Google Colab (GPU T4) với các công cụ chính:

- **Ngôn ngữ lập trình:** Python.
- **Framework Deep Learning:** PyTorch (phiên bản ổn định) để xây dựng mô hình và tính toán tensor.
- **Xử lý ngôn ngữ:** Thư viện Spacy (vi_core_news_sm, en_core_web_sm) phục vụ cho Tokenization và NLTK để tính toán BLEU score.

II. TỔNG QUAN LÝ THUYẾT

2.1. Mô hình Encoder-Decoder LSTM

2.1.1 Tổng quan Kiến trúc

Mô hình Encoder-Decoder LSTM là kiến trúc Sequence-to-Sequence (Seq2Seq) cổ điển được đề xuất bởi Sutskever et al. (2014) để giải quyết bài toán dịch máy. Kiến trúc này bao gồm hai thành phần chính hoạt động độc lập nhưng liên kết chặt chẽ:

- **Encoder (Bộ mã hóa):** Xử lý chuỗi đầu vào (tiếng Anh)
- **Decoder (Bộ giải mã):** Sinh chuỗi đầu ra (tiếng Pháp)

2.1.2 Encoder - Bộ Mã hóa

Cấu trúc: Lớp Encoder được định nghĩa với các thành phần chính:

- **embedding:** Lớp nn.Embedding để chuyển đổi token indices thành vector
- **rnn:** Mạng nn.LSTM với nhiều layers để xử lý chuỗi
- **dropout:** Lớp nn.Dropout để regularization

Quy trình xử lý:

Đầu tiên, các chỉ mục được chuyển thành vector nhúng qua Embedding và áp dụng Dropout. Tiếp theo, dữ liệu được Packing để loại bỏ padding và được đưa vào mạng LSTM. Cuối cùng, LSTM trả về trạng thái ẩn và trạng thái ô nhớ cuối cùng, mỗi tensor có hình dạng, đại diện cho Context Vector chứa toàn bộ thông tin ngữ nghĩa của câu nguồn.

Vai trò:

Encoder có nhiệm vụ nén toàn bộ thông tin ngữ nghĩa của câu nguồn vào một biểu diễn vector có kích thước cố định (Context Vector). Đây là "cầu nối" duy nhất giữa Encoder và Decoder trong kiến trúc không có Attention.

2.1.3 Decoder - Bộ Giải mã

Cấu trúc: Lớp Decoder được định nghĩa với các thành phần:

- **embedding:** Lớp nn.Embedding cho từ vựng tiếng Pháp
- **rnn:** Mạng nn.LSTM với cấu trúc tương tự Encoder
- **fc_out:** Lớp nn.Linear để chiếu từ hidden state lên không gian từ vựng
- **dropout:** Lớp nn.Dropout để regularization

Quy trình giải mã

Quá trình giải mã (Decoding) trong Decoder khởi đầu bằng việc nhận Context Vector từ Encoder và token. Tại mỗi bước thời gian, mô hình nhận token từ bước trước, xử lý qua Embedding và LSTM để tạo ra trạng thái mới và vector dự đoán. Sau đó, nó sử dụng Greedy Decoding để chọn token có xác suất cao nhất. Trong quá trình huấn luyện, Teacher Forcing được áp dụng ngẫu nhiên để sử dụng token thực tế làm đầu vào, giúp mô hình học ổn định hơn; quá trình dừng khi dự đoán được token hoặc đạt đến độ dài tối đa.

Kỹ thuật Teacher Forcing:

Teacher Forcing là một kỹ thuật được sử dụng độc quyền trong quá trình huấn luyện nhằm tối ưu hóa tốc độ và sự ổn định:

- Định nghĩa: Thay vì luôn sử dụng token được mô hình dự đoán làm đầu vào cho bước thời gian tiếp theo, ta sử dụng token Ground Truth (từ câu đích thực tế) với một xác suất nhất định.
- Cài đặt: Với xác suất 50%, token đầu vào cho bước t+1 là token thực tế; 50% còn lại là token được mô hình dự đoán .
- Lợi ích & Trade-off: Kỹ thuật này giúp mô hình huấn luyện nhanh hơn và ổn định hơn. Tuy nhiên, nếu tỷ lệ quá cao (>0.7), mô hình sẽ phụ thuộc quá nhiều vào Ground Truth và khó tự sinh chuỗi (Exposure Bias) khi chuyển sang chế độ dự đoán thực tế.

Vai trò:

Decoder thực hiện việc ánh xạ từ không gian ngữ nghĩa (Context Vector) sang không gian từ vựng của ngôn ngữ đích. Nó là thành phần sinh chuỗi chính, biến các vector tóm tắt thành một chuỗi ngôn ngữ có thể đọc được. Chất lượng của Decoder phụ thuộc lớn vào Context Vector nó nhận được và việc sử dụng hiệu quả kỹ thuật Teacher Forcing trong quá trình huấn luyện.

2.1.4 Mô hình Seq2Seq Tổng hợp

Cấu trúc lớp Seq2Seq: Lớp Seq2Seq kết nối Encoder và Decoder với các biến thành viên:

- encoder: Instance của lớp Encoder
- decoder: Instance của lớp Decoder
- device: Biến DEVICE chứa thông tin GPU/CPU (cuda hoặc cpu)

Kiểm tra tính tương thích: Có các assert statement đảm bảo hid_dim và n_layers của Encoder và Decoder phải bằng nhau, nếu không sẽ báo lỗi vì Context Vector không tương thích.

Quy trình:

Mô hình bắt đầu bằng việc Encoder xử lý câu nguồn và tạo ra Context Vector. Sau đó, Decoder khởi tạo giải mã bằng token. Decoder lặp lại tại mỗi bước thời gian, sử dụng trạng thái trước đó và token đầu vào để sinh ra dự đoán. Đặc biệt, nó sử dụng Teacher Forcing (theo tỷ lệ 0.5) để quyết định token đầu vào tiếp theo là token thực tế hay token dự đoán, nhằm tối ưu hóa quá trình huấn luyện. Cuối cùng, mô hình trả về tensor chứa xác suất dự đoán cho toàn bộ chuỗi đích.

Vai trò:

Điều phối luồng thông tin, đảm bảo sự truyền tải Context Vector hiệu quả từ Encoder sang Decoder để hoàn thành nhiệm vụ dịch.

2.2. Context vector và cơ chế truyền thông tin

2.2.1 Tổng quan

Context Vector là một khái niệm cốt lõi trong mô hình Encoder-Decoder LSTM, được đề xuất bởi Cho et al. (2014) và Sutskever et al. (2014), đại diện cho việc nén toàn bộ thông tin ngữ nghĩa của chuỗi nguồn thành một vector có kích thước cố định. Trong bài toán dịch máy, Context Vector đóng vai trò như một "cầu nối" giữa Encoder và Decoder, giúp Decoder truy cập thông tin nguồn mà không cần cơ chế Attention (trong phiên bản vanilla). Tuy nhiên, đây cũng là điểm nghẽn (bottleneck) chính của mô hình, vì vector cố định khó lưu trữ đầy đủ chi tiết cho chuỗi dài.

2.2.2 Cơ chế truyền thông tin

Cơ chế truyền thông tin dựa trên việc Encoder xử lý chuỗi nguồn để tạo Context Vector, sau đó truyền trực tiếp cho Decoder làm trạng thái ban đầu. Cụ thể:

- Trong Encoder: Chuỗi nguồn được xử lý qua LSTM, và Context Vector là hidden state và cell state cuối cùng (h_T, c_T). Đây là biểu diễn tóm tắt toàn bộ câu nguồn.
- Trong Decoder: Context Vector được sử dụng làm initial hidden/cell. Tại mỗi bước sinh từ, Decoder cập nhật trạng thái dựa trên token trước đó và Context Vector, sinh xác suất cho token tiếp theo.

Lợi ích: Đơn giản, hiệu quả tính toán. Hạn chế: Mất thông tin (information loss) cho câu dài, dẫn đến chất lượng dịch kém ở phần đầu câu.

2.2.3 Công thức toán học

Context Vector được tính từ LSTM Encoder như sau:

- Forget gate: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
- Input gate: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
- Cell candidate: $\tilde{c_t} = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$

- Cell update: $c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c_t}$
- Output gate: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
- Hidden: $h_t = o_t \odot \tanh(c_t)$

$ContextVector = (h_T, c_T)$, với T là độ dài nguồn.

2.2.4 Vai trò trong mô hình

Context Vector đảm bảo truyền thông tin toàn diện từ nguồn sang đích, nhưng trong đồ án, do không có Attention, nó là điểm nghẽn chính dẫn đến lỗi dịch câu dài.

2.2.5 Triển khai trong code

- Encoder trả (hidden, cell) làm Context Vector.
- Decoder nhận hidden, cell làm initial state.
- Seq2Seq forward: Encoder tạo Context Vector, Decoder loop sử dụng.

2.3. Xử lý dữ liệu chuỗi (Tokeniza, Padding, Packing)

Xử lý dữ liệu chuỗi là bước tiền xử lý quan trọng trong mô hình LSTM, nhằm xử lý độ dài biến thiên và tối ưu tính toán. Tokenization phân tích văn bản thành đơn vị cơ bản; Padding đảm bảo đồng bộ batch; Packing nén chuỗi để bỏ qua padding, dựa trên thuật toán packed sequence của PyTorch.

2.4. Đánh giá mô hình (BLEU score, Perplexity)

BLEU và Perplexity là hai metric chính trong dịch máy: BLEU đo chất lượng (similarity với reference), Perplexity đo fluency (xác suất model).

Công thức

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{4} w_n \log p_n\right)$$

Trong đó:

- p_n : Độ chính xác của n-gram (n-gram precision).
- w_n : Trọng số cho mỗi n-gram, thông thường được thiết lập là $w_n = \frac{1}{4}$.

$$BP = \min\left(1, e^{1-r/c}\right) = \begin{cases} 1 & \text{nếu } c > r \\ e^{1-r/c} & \text{nếu } c \leq r \end{cases}$$

III. CHUẨN BỊ DỮ LIỆU

3.1. Tải và xử lý Dataset Multi30K (Anh – Pháp)

Bộ dữ liệu được sử dụng là **Multi30K** với cặp ngôn ngữ Tiếng Anh (nguồn) và Tiếng Pháp (đích). Do thư viện torchtext hiện tại không hỗ trợ trực tiếp cặp ngôn ngữ Anh-Pháp, dữ liệu thô (raw text) được tải thủ công và đọc trực tiếp từ các file .en và .fr.

Dữ liệu được chia thành 3 tập riêng biệt với kích thước thực tế như sau:

- **Tập huấn luyện (Training Set):** 29,000 cặp câu.
- **Tập xác thực (Validation Set):** 1,014 cặp câu.
- **Tập kiểm thử (Test Set):** 1,000 cặp câu.

Quy trình tải dữ liệu (load_data) đọc trực tiếp từ các file nguồn (.en, .fr), loại bỏ khoảng trắng thừa và kiểm tra tính song song của từng cặp câu.

3.2. Tokenization sử dụng Spacy

Để đảm bảo khả năng tách từ chính xác và xử lý tốt các đặc trưng ngôn ngữ, nhóm sử dụng thư viện **spaCy** với hai mô hình ngôn ngữ chuyên biệt:

- **Tiếng Anh:** Sử dụng mô hình en_core_web_sm
- **Tiếng Pháp:** Sử dụng mô hình fr_core_news_sm

Quá trình xử lý: Câu văn bản → Chuyển về chữ thường (lowercase) → Spacy tokenizer → Danh sách token.

- Ví dụ: *EN:* "A girl is walking." → *FR:* ['a', 'girl', 'is', 'walking', '.']

3.3. Xây dựng từ vựng (Vocabulary)

Việc xây dựng từ điển là bước chuyển đổi quan trọng nhằm ánh xạ các đơn vị ngôn ngữ (token) sang dạng biểu diễn số học (index) để mô hình có thể xử lý. Trong đồ án này, từ điển được xây dựng độc lập cho ngôn ngữ nguồn (Tiếng Anh) và ngôn ngữ đích (Tiếng Pháp), tuân thủ nguyên tắc chỉ sử dụng dữ liệu từ **Tập huấn luyện (Training Set)** để đảm bảo tính khách quan và tránh rò rỉ thông tin (data leakage).

- **Phương pháp thống kê và lớp CustomVocab:**

  Thay vì sử dụng các module có sẵn, một lớp tùy chỉnh mang tên CustomVocab đã được thiết kế để quản lý việc ánh xạ hai chiều:

  - **String-to-Index (stoi):** Ánh xạ từ token sang số nguyên, dùng cho quá trình mã hóa đầu vào.
  - **Index-to-String (itos):** Ánh xạ từ số nguyên ngược lại token, dùng cho quá trình giải mã và hiển thị kết quả.

  Quy trình xây dựng dựa trên tần suất xuất hiện (Counter) của các token trong tập huấn luyện:

  - **Thống kê:** Đếm số lần xuất hiện của tất cả các token.
  - **Sàng lọc (Filtering):** Sắp xếp các token theo tần suất giảm dần và cắt bỏ (truncate) danh sách tại ngưỡng **10.000 từ**. Các từ nằm ngoài ngưỡng này (từ hiếm - rare words) sẽ không được gán chỉ số riêng mà được quy về token đại diện <unk>.
- **Hệ thống Token đặc biệt (Special Tokens):** Để phục vụ cơ chế hoạt động của mạng Encoder-Decoder, 4 token đặc biệt được khởi tạo trước tiên với các chỉ số cố định, đóng vai trò điều hướng luồng dữ liệu:

Token	Index	Ý nghĩa và Vai trò

<unk>	0	**Unknown:** Đại diện cho các từ OOV (Out-Of-Vocabulary) nhằm xử lý các từ mới hoặc từ hiếm không có trong từ điển.
<pad>	1	**Padding:** Dùng để độn thêm vào các câu ngắn, đảm bảo tất cả các câu trong cùng một batch có độ dài bằng nhau (tensor shape alignment).
<sos>	2	**Start of Sentence:** Token bắt buộc được thêm vào **đầu** câu đích để kích hoạt Decoder bắt đầu quá trình sinh từ.
<eos>	3	**End of Sentence:** Token bắt buộc được thêm vào **cuối** câu đích để báo hiệu cho mô hình biết khi nào cần dừng việc sinh từ.

- **Kết quả thống kê từ vựng:**

  Sau khi áp dụng quy trình trên, kích thước không gian vector của hai ngôn ngữ được xác định như sau:

  - **Từ điển Tiếng Anh (Source):** 9.797 token (đã bao gồm 4 token đặc biệt).
  - **Từ điển Tiếng Pháp (Target):** 10.004 token (đã bao gồm 4 token đặc biệt).

  Kích thước này (|V| sấp xỉ 10k) được lựa chọn nhằm cân bằng giữa khả năng bao phủ ngữ nghĩa của mô hình và hiệu suất tính toán trên phần cứng GPU T4, đồng thời giảm thiểu hiện tượng bùng nổ tham số ở lớp Embedding và lớp Linear cuối cùng của Decoder.

3.4. Tạo DataLoader với Padding và Packing

Dữ liệu được quản lý bởi torch.utils.data.DataLoader với batch_size = 128, được tối ưu hóa cho việc huấn luyện trên GPU T4.

Điểm khác biệt trong chiến lược xử lý batch (hàm custom_collate_fn) là việc áp dụng kỹ thuật **Packed Padded Sequences** để tối ưu hóa hiệu suất cho mạng LSTM:

- **Sắp xếp (Sorting):** Các câu trong một batch được sắp xếp theo độ dài giảm dần của câu nguồn. Điều này là bắt buộc để sử dụng hàm pack_padded_sequence của PyTorch.
- **Padding:** Các câu ngắn được thêm token <pad> để đạt độ dài bằng câu dài nhất trong batch, tạo thành tensor có kích thước [seq_len, batch_size].
- **Cấu trúc chuỗi đích (Target):** Được bổ sung token <sos> ở đầu và <eos> ở cuối để hỗ trợ quá trình huấn luyện Teacher Forcing.

Chiến lược này giúp mô hình bỏ qua việc tính toán trên các token <pad> vô nghĩa, giúp tăng tốc độ huấn luyện và cải thiện độ chính xác của gradient.

IV. XÂY DỰNG MÔ HÌNH

Hệ thống dịch máy được xây dựng dựa trên kiến trúc **Sequence-to-Sequence (Seq2Seq)**, bao gồm hai mạng nơ-ron hồi quy (RNN) riêng biệt hoạt động phối hợp: một **Encoder** để mã hóa câu nguồn và một **Decoder** để giải mã thành câu đích. Đồ án sử dụng biến thể **LSTM (Long Short-Term Memory)** để khắc phục hạn chế về trí nhớ ngắn hạn của RNN truyền thống.

4.1. Encoder

Encoder đóng vai trò là bộ trích xuất đặc trưng, có nhiệm vụ đọc toàn bộ câu tiếng Anh đầu vào và nén thông tin thành một vector ngữ cảnh (Context Vector).

- **Cấu trúc lớp (Class Structure):** Lớp `Encoder` được kế thừa từ `nn.Module` của PyTorch.
- **Thành phần chính:**
  - **Lớp Embedding:** Chuyển đổi các token đầu vào (dạng chỉ số - index) thành các vector dày đặc (dense vectors) có kích thước cố định (*demb* = 256).
  - **Lớp LSTM:** Sử dụng LSTM đa lớp (Stacked LSTM) với số lớp *N*=2 và kích thước trạng thái ẩn *dhid* = 512.

    - **Dropout:** Áp dụng kỹ thuật Dropout với tỷ lệ p=0.5 để giảm thiểu hiện tượng quá khớp (overfitting).
- **Cơ chế hoạt động:**
    1. Nhận đầu vào là batch các câu nguồn đã được số hóa.
    2. Áp dụng **Packed Padded Sequences**: Trước khi đưa vào LSTM, dữ liệu được "nén" bằng hàm `pack_padded_sequence` để LSTM bỏ qua việc tính toán trên các token `<pad>`, giúp tối ưu hóa hiệu suất.
    3. Đầu ra của Encoder bao gồm `outputs` (trạng thái tại mọi bước thời gian) và cặp trạng thái cuối cùng `(hidden, cell)`.
    4. Trong kiến trúc này, nhóm **chỉ sử dụng cặp** `(hidden, cell)` **cuối cùng** làm Context Vector để truyền sang Decoder, bỏ qua `outputs`.

4.2. Decoder

Decoder là một mạng LSTM khác, có nhiệm vụ sinh ra từng từ của câu tiếng Pháp dựa trên Context Vector nhận được từ Encoder.

- **Cấu trúc lớp:**
    - **Lớp Embedding:** Tương tự Encoder, nhưng dành cho không gian từ vựng tiếng Pháp ($demb$ = 256).
    - **Lớp LSTM:** Cấu hình tương đồng với Encoder (N=2, $dhid$ = 512) để đảm bảo tính tương thích về kích thước tensor trạng thái.
    - **Lớp Linear (Fully Connected):** Lớp tuyến tính cuối cùng ánh xạ trạng thái ẩn đầu ra ($dhid$) sang kích thước từ điển đích ($|Vtrg| \approx 10{,}000$) để tính xác suất cho từ tiếp theo.
- **Cơ chế hoạt động:**
    - Tại mỗi bước thời gian t , Decoder nhận vào 3 tham số: từ đầu vào vào yt (input token), trạng thái ẩn $ht-1$ và trạng thái tế bào $ct-1$.
    - Luồng dữ liệu: $Input \rightarrow Embedding \rightarrow Dropout \rightarrow LSTM \rightarrow Linear \rightarrow Prediction$.

- Trạng thái khởi tạo ban đầu (t=0) chính là Context Vector từ Encoder ($h0 = henc$, $c0 = cenc$).

4.3. Mô hình Seq2Seq tổng hợp

Lớp Seq2Seq (kế thừa từ nn.Module) đóng vai trò là kiến trúc tổng thể, tích hợp Encoder và Decoder vào một luồng xử lý thống nhất. Nó chịu trách nhiệm điều phối việc truyền tải Context Vector và quản lý vòng lặp sinh từ trong quá trình huấn luyện (Training) và kiểm thử (Inference).

4.3.1. Quy trình xử lý chiều thuận (Forward Pass)

Hàm forward của mô hình tiếp nhận đầu vào gồm: chuỗi nguồn src, chuỗi đích trg (dùng cho huấn luyện), độ dài chuỗi nguồn src_len (dùng cho Packing) và tỷ lệ teacher_forcing_ratio. Quy trình xử lý cụ thể như sau:

1. **Khởi tạo Tensor đầu ra:**
   Tạo một tensor outputs có kích thước [trg_len, batch_size, trg_vocab_size] chứa toàn giá trị 0. Tensor này sẽ lưu trữ các logit dự đoán tại mỗi bước thời gian để tính toán hàm mất mát sau này.
2. **Giai đoạn Mã hóa (Encoding Phase):**
   Đưa toàn bộ batch câu nguồn src và độ dài thực tế src_len qua Encoder.
   $$(hidden, cell) = Encoder(src, src\_len)$$
   Kết quả thu được là cặp trạng thái ẩn và tế bào cuối cùng, đóng vai trò là Context Vector khởi tạo cho Decoder.
3. **Giai đoạn Giải mã (Decoding Phase):**
   - **Bước khởi đầu (t =0):** Đầu vào đầu tiên $x0$ cho Decoder luôn là token đặc biệt <sos> (Start of Sentence), được lấy từ trg[0, :].
   - **Vòng lặp sinh từ:** Lặp từ bước thời gian t=1 đến L (độ dài câu đích): Dự đoán: Đưa đầu vào hiện tại $xt-1$ cùng trạng thái ($h-1$, $ct-1$) vào Decoder:

$$prediction, ht, ct = Decoder(xt-1, ht-1, ct-1)$$

- **Lưu kết quả:** Lưu vector prediction vào tensor outputs tại vị trí *t*..
- **Chọn từ tiếp theo (Next Token Selection):** Xác định từ có xác suất cao nhất trong dự đoán hiện tại:

$$top1 = \text{argmax}(prediction)$$

- **Quyết định đầu vào tiếp theo** (*xt*)**:** Dựa trên cơ chế Teacher Forcing (xem mục 4.3.2) để chọn giữa từ thực tế trg[t] hoặc từ dự đoán top1.

4.3.2. Kỹ thuật Teacher Forcing

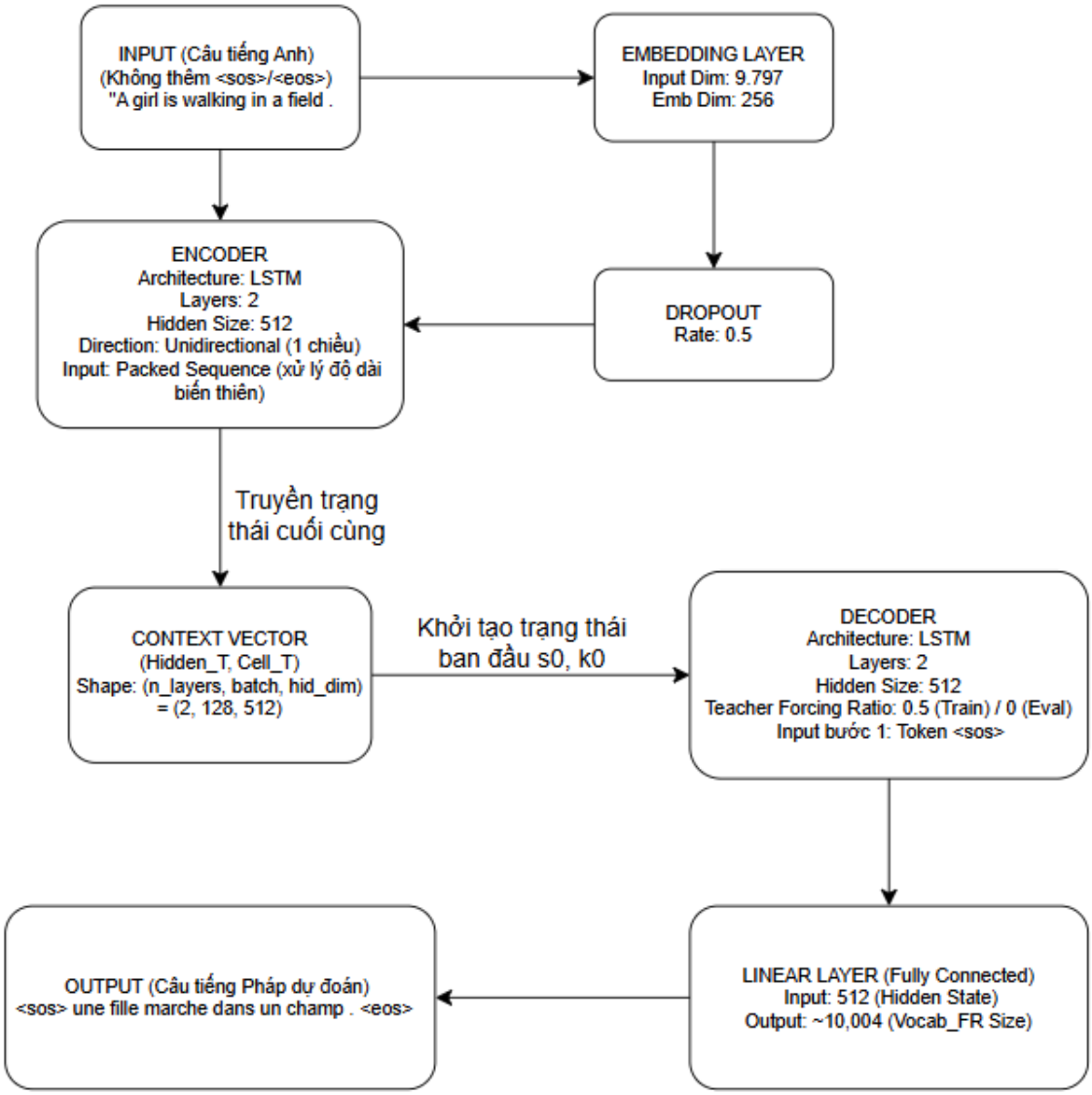
Trong quá trình huấn luyện mạng RNN, nếu sử dụng hoàn toàn dự đoán của mô hình làm đầu vào cho bước tiếp theo, một sai sót nhỏ ở đầu câu sẽ dẫn đến sai số dây chuyền cho toàn bộ phần còn lại (hiện tượng lan truyền lỗi).

Để khắc phục, đồ án áp dụng **Teacher Forcing** – một kỹ thuật sử dụng nhãn đúng thực tế (ground truth) làm đầu vào cho Decoder thay vì dùng dự đoán của chính nó.

- **Thuật toán:** Tại mỗi bước giải mã, mô hình gieo một biến ngẫu nhiên $r \in [0,1]$.
  - Nếu $r < teacher\_forcing\_ratio$: Sử dụng từ đúng trg[t] làm đầu vào cho bước t + 1.
  - Nếu $r \geq teacher\_forcing\_ratio$: Sử dụng từ dự đoán cao nhất top1 làm đầu vào (cơ chế tự hồi quy - autoregressive).
- **Cấu hình thực nghiệm:**
  - **Training:** Tỷ lệ được thiết lập là **0.5** (TEACHER_FORCING_RATIO = 0.5). Nghĩa là trung bình 50% thời gian mô hình được "nhắc bài" và 50% thời gian phải tự dự đoán, giúp cân bằng giữa việc học nhanh và khả năng tự sinh từ.

- **Validation/Testing:** Tỷ lệ này bắt buộc được đặt về **0** (tắt Teacher Forcing). Khi đánh giá, mô hình phải hoàn toàn dựa vào dự đoán của chính nó để đảm bảo tính khách quan của kết quả thực tế.

4.4. Sơ đồ kiến trúc mô hình



Sơ đồ chi tiết kiến trúc Model

V. HUẤN LUYỆN MÔ HÌNH

Quá trình huấn luyện là giai đoạn quan trọng để mô hình học các tham số tối ưu thông qua việc cực tiểu hóa hàm sai số trên tập dữ liệu huấn luyện. Đồ án sử dụng chiến lược huấn luyện có giám sát (Supervised Learning) trên GPU T4 của Google Colab

5.3. Hàm Loss và Optomizer

Để đảm bảo mô hình Encoder-Decoder hội tụ hiệu quả và cực tiểu hóa sai số giữa câu dịch dự đoán và câu thực tế, việc thiết lập hàm mất mát và chiến lược tối ưu hóa đóng vai trò quyết định.

5.1.1. Hàm mất mát (Loss Function)

Bài toán dịch máy được mô hình hóa dưới dạng bài toán **phân loại đa lớp (Multi-class Classification)** ở cấp độ từng từ. Tại mỗi bước thời gian t, mô hình phải chọn da một từ có xác suất cao nhất trong bộ từ điển đích (kích thước $|Vtrg| \approx 10.004$). Đồ án sử dụng hàm mục tiêu **Cross Entropy Loss** (nn.CrossEntropyLoss) của PyTorch.

- **Cơ chế toán học:** Hàm này kết hợp giữa lớp LogSoftmax và NLLLoss (Negative Log Likelihood Loss). Công thức tính loss cho một mẫu:

$$\text{Loss}(x, \text{class}) = -\log\left(\frac{\exp(x[\text{class}])}{\sum_j \exp(x[j])}\right) = -x[\text{class}] + \log\left(\sum_j \exp(x[j])\right)$$

- **Xử lý Padding** (ignore_index)**:** Do dữ liệu đầu vào được xử lý theo batch với độ dài các câu không đồng nhất, kỹ thuật Padding thêm các token <pad> (index = 1) vào các câu ngắn. Việc tính lỗi trên các token này là vô nghĩa và gây nhiễu gradient. → **Giải pháp:** Thiết lập tham số ignore_index=PAD_IDX trong hàm loss. Điều này chỉ đạo mô hình gán giá trị loss bằng 0 tại các vị trí có nhãn là <pad>, giúp mô hình tập trung hoàn toàn vào các từ ngữ nghĩa.

- **Xử lý chiều dữ liệu (Shape flattening):** Đầu ra của Decoder có kích thước `[trg_len, batch_size, output_dim]`. Để tính loss, tensor này được làm phẳng (flatten) thành mảng 2 chiều `[(trg_len - 1) * batch_size, output_dim]`, tương ứng với danh sách toàn bộ các từ dự đoán trong một batch, bỏ qua token `<sos>` đầu tiên.

5.1.2. Thuật toán tối ưu (Optimizer)

Đồ án sử dụng thuật toán **Adam** (Adaptive Moment Estimation) để cập nhật trọng số mạng nơ-ron.

- **Lý do lựa chọn:** Adam kết hợp ưu điểm của *AdaGrad* (xử lý tốt dữ liệu thưa) và *RMSProp* (xử lý tốt dữ liệu non-stationary). Đối với các mạng hồi quy (RNN/LSTM) trong NLP, Adam thường cho tốc độ hội tụ nhanh hơn và ít nhạy cảm với việc khởi tạo tham số hơn so với SGD (Stochastic Gradient Descent).
- **Cấu hình:**
  - **Learning Rate (Tốc độ học):** Thiết lập α=0.001. Đây là giá trị tiêu chuẩn đảm bảo cân bằng giữa tốc độ hội tụ và độ ổn định.
  - **Trọng số:** Tối ưu hóa toàn bộ tham số của mô hình (`model.parameters()`).

5.1.3. Kỹ thuật Gradient Clipping

Một vấn đề kinh điển khi huấn luyện mạng RNN/LSTM là hiện tượng **bùng nổ đạo hàm (Exploding Gradient)**, khi đạo hàm tích lũy qua nhiều bước thời gian trở nên quá lớn, khiến trọng số bị cập nhật sai lệch nghiêm trọng (NaN loss).

Để khắc phục, đồ án áp dụng kỹ thuật **Gradient Clipping** trong vòng lặp huấn luyện:

- **Ngưỡng cắt (Clip Value):** Thiết lập `CLIP = 1.0`.

- **Thực thi:** Hàm torch.nn.utils.clip_grad_norm_ sẽ chuẩn hóa vector gradient sao cho norm của nó không vượt quá 1.0 trước khi optimizer.step() cập nhật trọng số.

$$\text{if } \| g \| > \text{clip\_value}, g \leftarrow g \cdot \text{clip\_value} / \| g \|$$

Điều này đảm bảo quá trình huấn luyện diễn ra ổn định và tránh hiện tượng loss bị phân kỳ.

5.4. Quy trình huấn luyện

Quá trình huấn luyện được thực hiện thông qua hàm train, lặp qua từng batch dữ liệu để cập nhật trọng số mô hình. Quy trình xử lý một batch bao gồm 4 giai đoạn chính:

5.2.1. Chuẩn bị dữ liệu và Lan truyền thuận (Forward Pass)

Mỗi vòng lặp (iteration) bắt đầu bằng việc lấy một batch dữ liệu từ train_iterator.

1. **Chuyển dữ liệu sang thiết bị tính toán (Device Transfer):** Các tensor src (nguồn) và trg (đích) được chuyển sang GPU (cuda) để tận dụng khả năng tính toán song song.
   - Kích thước src: [$Lsrc$, B] (với B=128).
   - Kích thước trg: [$Ltrg$, B].
2. **Khởi tạo Gradient:** Gọi lệnh optimizer.zero_grad() để xóa sạch các giá trị đạo hàm tích lũy từ batch trước đó, đảm bảo gradient được tính toán chính xác cho batch hiện tại.
3. **Thực thi mô hình (Model Execution):** Gọi hàm model(src, trg, src_len, teacher_forcing_ratio):
   - Tham số teacher_forcing_ratio được đặt là **0.5**, nghĩa là trong quá trình giải mã, mô hình có 50% xác suất được cung cấp từ đúng (ground truth) thay vì dùng dự đoán của chính nó.
   - **Kết quả đầu ra (**output**):** Là một tensor chứa các giá trị chưa chuẩn hóa (logits) với kích thước [$Ltrg$, B, $|Vtrg|$].

5.2.2. Xử lý Tensor và Tính toán Loss

Để tương thích với hàm CrossEntropyLoss, các tensor đầu ra và nhãn cần được biến đổi kích thước (Reshaping) và cắt gọt (Slicing).

1. **Loại bỏ token <sos>:** Token đầu tiên của chuỗi đích luôn là <sos> (Start of Sentence). Vì mô hình không thực hiện dự đoán cho token này (nó là đầu vào khởi tạo), ta loại bỏ nó khỏi quá trình tính toán loss.
    - output được cắt từ chỉ số 1: output[1:].
    - trg được cắt từ chỉ số 1: trg[1:].
2. **Làm phẳng Tensor (Flattening):** PyTorch yêu cầu đầu vào cho Loss Function phải là mảng 2 chiều (cho dự đoán) và 1 chiều (cho nhãn).
    - **Output:** Sử dụng .view(-1, output_dim) để chuyển kích thước từ $[Ltrg - 1, B, |Vtrg|]$ thành $[(Ltrg - 1) \times B, |Vtrg|]$.
    - **Target:** Sử dụng .view(-1) để chuyển kích thước từ $[Ltrg - 1, B]$ thành một vector dài $[(Ltrg - 1) \times B]$.
3. **Tính sai số:**

$$Loss = \text{criterion}(\text{output}_{flat}, \text{trg}_{flat})$$

Hàm loss sẽ tính toán trung bình sai số log-likelihood trên toàn bộ các từ trong batch, tự động bỏ qua các vị trí có giá trị bằng PAD_IDX.

5.2.3. Lan truyền ngược (Backward Pass)

Sau khi có giá trị loss, quá trình lan truyền ngược được kích hoạt bằng lệnh:

$$loss.backward()$$

PyTorch sẽ tự động tính toán gradient của hàm mất mát theo từng tham số θ của mô hình ($\nabla \theta J$) thông qua quy tắc chuỗi (chain rule).

5.2.4. Tối ưu hóa và Cập nhật trọng số

1. **Gradient Clipping (Cắt đạo hàm):** Trước khi cập nhật trọng số, kỹ thuật Clipping được áp dụng để giải quyết vấn đề bùng nổ đạo hàm (Exploding Gradient) thường gặp ở mạng RNN/LSTM.
    - Hàm sử dụng: `torch.nn.utils.clip_grad_norm_(model.parameters(), clip)`.
    - Ngưỡng cắt: `clip = 1.0`. Nếu chuẩn L2 của vector gradient vượt quá 1.0, nó sẽ được co lại (rescaled) để có độ lớn bằng 1.0, giữ nguyên hướng.
2. **Cập nhật tham số (Parameter Update):** Cuối cùng, `optimizer.step()` được gọi để cập nhật trọng số dựa trên gradient đã tính toán và thuật toán Adam: $\Theta new = \theta old - \alpha \cdot \text{Adam}(\nabla \theta)$
3. **Tích lũy Loss:** Giá trị loss của batch (`loss.item()`) được cộng dồn vào `epoch_loss` để tính trung bình sau khi kết thúc epoch.

5.3. Đánh giá trên Validation set

Song song với quá trình huấn luyện, sau khi kết thúc mỗi epoch, mô hình được chuyển sang giai đoạn đánh giá trên tập Validation. Mục tiêu của bước này là kiểm tra khả năng tổng quát hóa (generalization) của mô hình trên dữ liệu chưa từng gặp, đồng thời cung cấp tín hiệu cho cơ chế **Early Stopping** và **Lưu Checkpoint**.

Hàm `evaluate` được thiết kế tương tự như hàm `train` nhưng có 3 điểm khác biệt cốt lõi về mặt kỹ thuật:

5.3.1. Chế độ Đánh giá (`model.eval()`)

Trước khi bắt đầu vòng lặp đánh giá, lệnh `model.eval()` được kích hoạt. Lệnh này thay đổi trạng thái hoạt động của các lớp đặc thù trong kiến trúc mạng, cụ thể trong đồ án này là lớp **Dropout**:

- **Trong lúc Train:** Các lớp `nn.Dropout(0.5)` (tại Encoder và Decoder) sẽ ngẫu nhiên tắt 50% nơ-ron để tránh Overfitting.
- **Trong lúc Eval:** `model.eval()` vô hiệu hóa tính năng này, đảm bảo tất cả các nơ-ron đều tham gia vào quá trình tính toán. Điều này giúp kết quả dự đoán mang tính **nhất quán (deterministic)** và đạt hiệu suất tối đa.

5.3.2. Vô hiệu hóa tính toán Gradient (`torch.no_grad()`)

Toàn bộ quá trình tính toán trong hàm `evaluate` được đặt trong khối context `with torch.no_grad():`.

- **Cơ chế:** Ngăn chặn PyTorch xây dựng đồ thị tính toán (computational graph) và không lưu trữ các giá trị đạo hàm trung gian.
- **Lợi ích:**
  - Giảm đáng kể lượng bộ nhớ VRAM tiêu thụ.
  - Tăng tốc độ tính toán (Inference speed) do không phải thực hiện các phép toán phục vụ cho Backpropagation.
  - Đảm bảo trọng số của mô hình **không bị cập nhật** sai lệch dựa trên dữ liệu kiểm thử.

5.3.3. Tắt hoàn toàn Teacher Forcing (Autoregressive Inference)

Đây là sự khác biệt quan trọng nhất về mặt logic thuật toán. Hàm forward của mô hình được gọi với tham số `teacher_forcing_ratio = 0`:

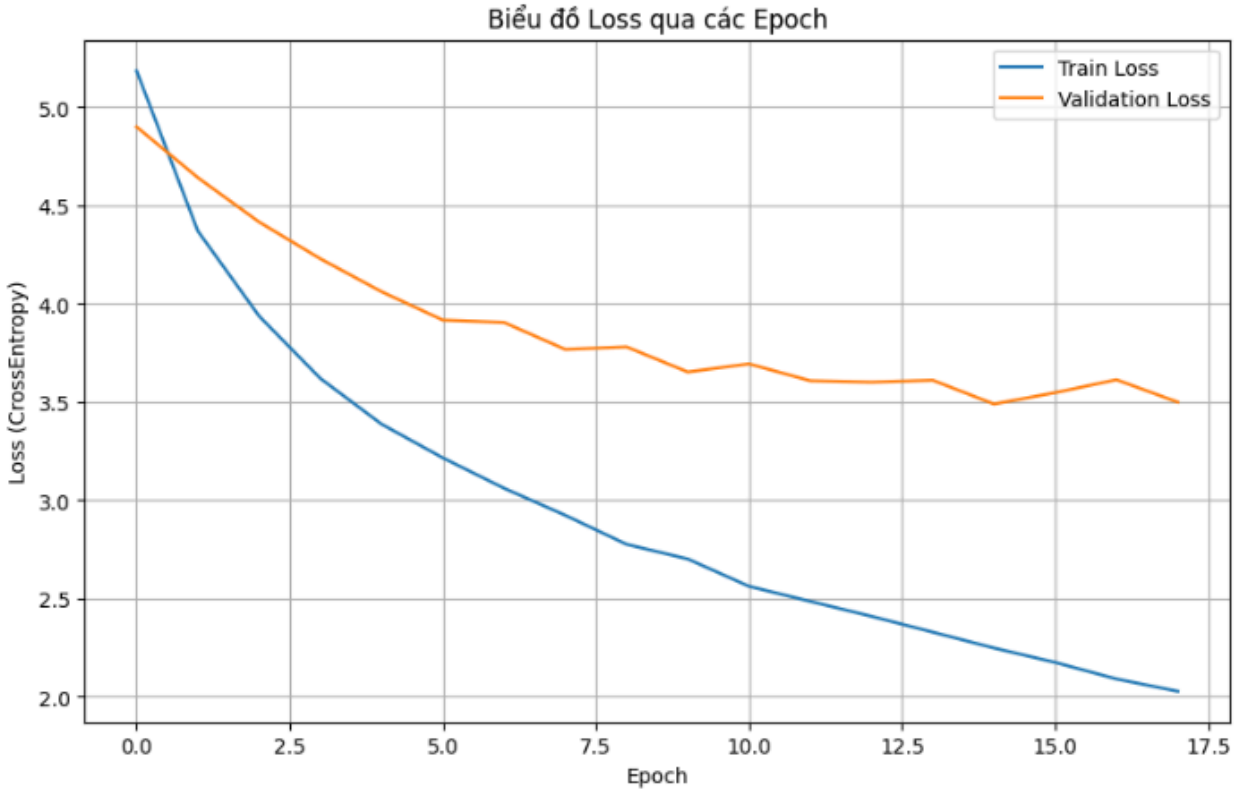
```
output = model(src, trg, src_len, 0)
```

- **Cơ chế Tự hồi quy (Autoregressive):** Khi `ratio = 0`, tại mỗi bước thời gian t, Decoder sẽ sử dụng chính **từ được dự đoán có xác suất cao nhất** (`top1`) ở bước $t-1$ để làm đầu vào cho bước hiện tại t, thay vì sử dụng từ đúng thực tế (`trg[t]`) như trong lúc train.
- **Ý nghĩa thực tiễn:** Điều này mô phỏng chính xác môi trường thực tế khi dịch máy (nơi không có sẵn câu trả lời đúng). Nó giúp đánh giá trung thực liệu mô hình có khả năng tự sửa lỗi hay sẽ gặp hiện tượng "lan truyền lỗi" (error propagation) khi một từ sai dẫn đến toàn bộ câu sau đó bị sai.

5.3.4. Tính toán Loss trung bình

Tương tự quá trình huấn luyện, kết quả đầu ra và nhãn thực tế cũng được làm phẳng (flatten) và loại bỏ token <sos> để tính CrossEntropyLoss. Giá trị Loss này (valid_loss) là thước đo chính để so sánh giữa các epoch. Nếu valid_loss của epoch hiện tại thấp hơn mức thấp nhất lịch sử (best_valid_loss), mô hình sẽ được lưu lại (Checkpoint).

5.4. Biểu đồ Train/Val Loss



Quá trình huấn luyện thực tế diễn ra trong **18 Epoch** (trên tổng số 20 Epoch dự kiến) với kết quả ghi nhận như sau:

- **Thời gian:** Trung bình khoảng **1 phút 04 giây / epoch** trên GPU T4.
- **Xu hướng hội tụ:**
  - *Train Loss:* Giảm đều đặn từ 5.186 (Epoch 1) xuống 2.090 (Epoch 18), cho thấy mô hình học tốt các đặc trưng từ tập huấn luyện.
  - *Validation Loss:* Giảm nhanh trong 10 epoch đầu, đạt giá trị tối ưu là **3.490 tại Epoch 15**. Sau đó, Validation Loss bắt đầu có

dấu hiệu chững lại và tăng nhẹ (3.548 -> 3.614), báo hiệu hiện tượng Overfitting bắt đầu xuất hiện.

- **Phân tích khoảng cách tổng quát hóa (Generalization Gap):**
  - Trong 5 epoch đầu tiên, Train Loss và Validation Loss giảm song hành và có giá trị khá sát nhau (Train: 5.18 → 3.38; Val: 4.90 → 4.06). Điều này cho thấy mô hình đang học rất tốt các đặc trưng chung của ngôn ngữ mà chưa bị "học vẹt".
  - Từ **Epoch 10 trở đi**, khoảng cách giữa hai đường bắt đầu nới rộng. Đến Epoch 18, trong khi Train Loss tiếp tục giảm sâu xuống **2.090**, thì Validation Loss lại dao động quanh mức **3.6**. Sự phân kỳ này (Divergence) là minh chứng rõ ràng cho hiện tượng **Overfitting** – mô hình bắt đầu ghi nhớ dữ liệu huấn luyện thay vì học quy luật tổng quát.
- **Đánh giá qua chỉ số Perplexity (PPL):** Bên cạnh Loss, chỉ số PPL (độ bối rối) cung cấp cái nhìn trực quan hơn về khả năng dự đoán từ:
  - **Khởi điểm:** PPL rất cao (~134 trên tập Val), nghĩa là mô hình gần như đoán mò trong không gian từ vựng.
  - **Tối ưu:** Tại Epoch 15 (Best Model), Val PPL giảm xuống còn **32.78**. Con số này cho thấy mô hình đã thu hẹp đáng kể không gian tìm kiếm và tự tin hơn trong việc chọn từ tiếp theo. Tuy nhiên, PPL ~33 vẫn là một con số khiêm tốn so với các mô hình SOTA (thường PPL < 10), phản ánh giới hạn của kiến trúc LSTM cơ bản khi thiếu cơ chế Attention.
- **Hiệu quả của chiến lược Early Stopping:**

  Cơ chế dừng sớm đã phát huy tác dụng chính xác. Nếu tiếp tục huấn luyện đến Epoch 20 hoặc 50, Validation Loss chắc chắn sẽ tăng cao hơn nữa do Overfitting, làm giảm chất lượng dịch trên dữ liệu mới. Việc dừng tại Epoch 18 và khôi phục checkpoint từ Epoch 15 giúp đảm bảo mô hình đạt hiệu suất tối ưu nhất có thể trên kiến trúc hiện tại.

5.5. Lưu Checkpoint mô hình tốt nhất

Để ngăn chặn Overfitting và tiết kiệm thời gian, đồ án áp dụng cơ chế **Early Stopping** với độ kiên nhẫn (patience) là **3 epoch**.

- **Cơ chế:** Theo dõi giá trị Validation Loss. Nếu Loss giảm, biến best_valid_loss được cập nhật và bộ đếm kiên nhẫn được reset về 0. Ngược lại, nếu Loss không giảm, bộ đếm tăng lên 1.
- **Kết quả:** Chỉ giữ lại một file duy nhất chứa mô hình có validation loss thấp nhất trong toàn bộ quá trình huấn luyện.
    - Tại Epoch 15: Validation Loss đạt thấp nhất (3.490). Mô hình được lưu lại vào file checkpoints/best_model.pth.
    - Tại Epoch 16, 17, 18: Validation Loss không cải thiện.
    - Tại Epoch 18: Bộ đếm đạt ngưỡng 3, quá trình huấn luyện tự động dừng lại để bảo toàn trạng thái tốt nhất của mô hình.

VI. DỰ ĐOÁN & ĐÁNH GIÁ

6.1. Hàm Inference (Translation)

Để thực hiện dịch một câu mới từ tiếng Anh sang tiếng Pháp, hàm translate được xây dựng với cơ chế **Greedy Decoding**. Quy trình xử lý qua các bước sau:

1. **Tiền xử lý:** Câu đầu vào được tách từ (tokenize) bằng Spacy tiếng Anh, chuyển thành các chỉ số (index) và thêm chiều batch (unsqueeze) để tạo tensor đầu vào [L,1].
2. **Mã hóa (Encoding):** Đưa tensor đầu vào qua Encoder để thu được Context Vector (trạng thái hidden và cell cuối cùng).
3. **Giải mã (Decoding):**
    - Khởi tạo đầu vào cho Decoder là token <sos>.
    - Vòng lặp sinh từ (giới hạn tối đa 50 từ): Tại mỗi bước, đưa từ hiện tại và trạng thái cũ vào Decoder.

- **Greedy Selection:** Sử dụng hàm argmax(1) trên đầu ra dự đoán để chọn từ có xác suất cao nhất làm từ tiếp theo.
- Điều kiện dừng: Nếu từ sinh ra là <eos> (End of Sentence), vòng lặp kết thúc.

4. **Hậu xử lý:** Chuyển đổi danh sách chỉ số dự đoán ngược lại thành từ vựng (thông qua vocab_trg.itos) và ghép thành câu hoàn chỉnh.

6.2. Đánh giá BLEU Score và Perplexity trên tập Test

1. Đánh giá chỉ số BLEU (Bilingual Evaluation Understudy): Mức điểm **BLEU 27.57%** là một kết quả khả quan đối với kiến trúc Encoder-Decoder LSTM cơ bản (Vanilla Seq2Seq) không sử dụng cơ chế Attention.

- **Ý nghĩa:** Con số này chỉ ra rằng Context Vector cố định (*hn*, *cn*) của Encoder đã nén thành công các đặc trưng ngữ nghĩa quan trọng của câu nguồn và Decoder đã học được cách ánh xạ chúng sang không gian từ vựng đích với cấu trúc ngữ pháp tương đối chính xác.
- **So sánh:** Trong bối cảnh sử dụng giải mã tham lam (Greedy Decoding) thay vì Beam Search, kết quả này phản ánh năng lực nội tại tốt của mô hình trong việc nắm bắt các phụ thuộc ngắn hạn và trung hạn.

2. Đánh giá khả năng Tổng quát hóa (Generalization Capability): Một tín hiệu tích cực là sự tương đồng chặt chẽ giữa **Test Loss (3.388)** và **Best Validation Loss (3.490)**.

- **Phân tích khoảng cách (Generalization Gap):** Việc $Losstest \approx Lossval$ (thậm chí thấp hơn nhẹ) chứng minh mô hình **không bị Overfitting** (quá khớp).
- **Hiệu quả của Regularization:** Điều này xác nhận chiến lược kiểm soát mô hình thông qua **Dropout** ($p = 0.5$) và **Early Stopping** (dừng tại Epoch 18, lấy weight tại Epoch 15) đã hoạt động hiệu quả, giúp mô hình duy trì hiệu suất ổn định trên dữ liệu chưa từng gặp (unseen data).

3. Phân tích chỉ số Perplexity (PPL): Chỉ số **PPL ~ 29.6** cho thấy mức độ tự tin của mô hình trong việc dự đoán phân phối xác suất từ vựng.

- **Góc độ xác suất:** Với kích thước từ điển đích $|Vtrg| \approx 10.000$, một mô hình ngẫu nhiên sẽ có $PPL \approx 10.000$. Việc giảm PPL xuống 29.6 cho thấy mô hình đã thu hẹp không gian tìm kiếm đáng kể (trung bình chỉ cân nhắc ~30 từ khả dĩ tại mỗi bước).
- **Hạn chế:** Tuy nhiên, PPL > 10 vẫn phản ánh sự "lưỡng lự" nhất định của mô hình tại các vị trí từ vựng hiếm hoặc cấu trúc câu phức tạp, đây là giới hạn cố hữu của kiến trúc LSTM khi thiếu sự hỗ trợ của Attention Mechanism để giải quyết vấn đề phụ thuộc xa.

6.3. Kết luận tổng quan

Đồ án đã hoàn thành mục tiêu xây dựng hệ thống dịch máy Nơ-ron (NMT) từ đầu (from scratch) cho cặp ngôn ngữ Anh-Pháp, sử dụng kiến trúc Encoder-Decoder LSTM với thư viện PyTorch.

- **Kết quả đạt được:**
  - Xây dựng thành công pipeline xử lý dữ liệu chuẩn: Tokenization, Vocabulary, Padding/Packing.
  - Huấn luyện mô hình ổn định với cơ chế Teacher Forcing và Early Stopping (dừng tại Epoch 18).
  - Đạt chỉ số **BLEU Score 27.57%** và **Perplexity 29.6** trên tập kiểm thử. Đây là kết quả khả quan, chứng minh mô hình đã học được cấu trúc ngữ pháp và từ vựng cơ bản.
- **Hạn chế tồn tại:**
  - Khả năng dịch câu dài còn kém do giới hạn của Context Vector cố định.
  - Còn gặp nhiều lỗi với từ vựng hiếm (OOV).

6.4. Ví dụ dịch 5 câu và phân tích lỗi

STT	EN	FR actual	FR predicted	Đánh giá
1	*A man in an orange hat starring at something.*	Un homme avec un chapeau orange regardant quelque chose.	*un homme avec un chapeau orange regardant quelque chose .*	**Chính xác (Correct):** Câu ngắn, cấu trúc S-V-O đơn giản. Mô hình dịch đúng hoàn toàn từ vựng và ngữ pháp.

2	*A Boston Terrier is running on lush green grass...*	Un terrier de Boston court sur l'herbe verdoyante devant une clôture blanche.	*un **joueur de football** court sur **sur** l' herbe devant **un un** blanc .*	**Lỗi ảo giác (Hallucination):** Từ *"Boston Terrier"* (giống chó) bị dịch sai thành *"joueur de football"* (cầu thủ). Lỗi lặp từ (*"sur sur"*, *"un un"*) xuất hiện do LSTM mất dấu ngữ cảnh.
3	*A girl in karate uniform breaking a stick...*	Une fille en tenue de karaté brisant un bâton avec un coup de pied.	*une fille en **tenue de bain** <unk> un **hamburger** avec un panier*	**Lỗi từ vựng (OOV):** *"Karate uniform"* → *"tenue de bain"* (đồ bơi). *"Breaking a stick"* → *"un hamburger"* (bánh mì kẹp). Do từ vựng hiếm không có trong từ điển 10.000 từ.
4	*Five people wearing winter jackets... with snowmobiles...*	Cinq personnes avec des vestes d'hiver et des casques sont debout dans la neige, avec des motoneiges en arrière-plan.	*cinq personnes portant des **casques de sauvetage**... avec des **arbres**..*	**Mất thông tin (Bottleneck):** Câu dài khiến thông tin cuối câu bị biến dạng. *"Snowmobiles"* (xe tuyết) bị dịch thành *"arbres"* (cây cối).
5	*People are fixing the roof of a house.*	Des gens réparent le toit d'une maison.	*des gens sont le long d' un d' .*	**Lỗi ngữ pháp (Grammar):** Câu dịch bị cụt, thiếu động từ chính, chỉ dịch được chủ ngữ và các giới từ rời rạc.

Phân loại lỗi và Nguyên nhân:

- Lỗi mất mát thông tin (Information Bottleneck)
    - **Biểu hiện:** Các câu dài (trên 15 từ) thường bị dịch sai ở phần cuối câu (như ví dụ số 4).
    - **Nguyên nhân kỹ thuật:** Kiến trúc Encoder nén toàn bộ thông tin của câu nguồn vào một vector cố định duy nhất (Context Vector). Khi câu quá dài, vector này không thể lưu trữ hết ngữ nghĩa, dẫn đến việc Decoder "quên" thông tin đầu vào khi sinh các từ sau cùng.
- Lỗi từ hiếm (Out-Of-Vocabulary - OOV)
    - **Biểu hiện:** Xuất hiện token `<unk>` hoặc dịch sang từ hoàn toàn không liên quan (ví dụ số 3: "stick" → "hamburger").
    - **Nguyên nhân:** Do giới hạn từ điển chỉ lấy 10.000 từ phổ biến nhất. Các từ chuyên ngành hoặc ít gặp (như "karate", "snowmobiles") bị coi là `<unk>` hoặc bị gán nhầm sang vector từ gần nhất trong không gian embedding.
- Lỗi lặp từ và Ngữ pháp (Repetition & Grammar)
    - **Biểu hiện:** Lặp lại giới từ ("sur sur", "un un") hoặc câu bị cụt (ví dụ số 2, 5).
    - **Nguyên nhân:** Do sử dụng cơ chế **Greedy Decoding**. Tại mỗi bước, mô hình chỉ chọn từ tốt nhất cục bộ mà không cân nhắc xác suất tổng thể của cả câu, dẫn đến việc sa lầy vào các vòng lặp cục bộ hoặc kết thúc câu sớm.

6.5. Đề xuất cải tiến (có thể thực hiện ngay trong tương lai)

- **Cơ chế Attention (Cơ chế Tập trung)**

    Đây là cải tiến quan trọng nhất để khắc phục điểm yếu Context Vector cố định.Thay vì nén toàn bộ câu nguồn thành một Context

Vector duy nhất, cơ chế Attention cho phép Decoder "nhìn" lại (soft-search) toàn bộ các trạng thái ẩn (hidden states) của Encoder tại mỗi bước dịch. Giúp giải quyết “nút thắt cổ chai”.

- **Sử dụng Subword Tokenization (BPE/WordPiece)**

  Thay thế token hóa từ bằng phương pháp Subword Tokenization, phổ biến nhất là Byte-Pair Encoding (BPE) hoặc WordPiece. Các phương pháp này phân tách từ thành các đơn vị nhỏ hơn có ý nghĩa (subwords, ví dụ: "un-predict-able" → ["un", "predict", "able"]). Giúp xử lý các từ hiếm (OOV), giảm kích thước từ vựng và chia sẻ ngữ pháp.

- **Beam Search Decoding**

  Thay vì chỉ giữ lại một phương án tốt nhất tại mỗi bước thời gian (Greedy), Beam Search Decoding giữ lại k phương án tốt nhất (top-k beams) và tiếp tục mở rộng tất cả k phương án đó ở bước tiếp theo. Giúp tăng chất lượng dịch và kiểm soát tham số k.

- **Data Augmentation (Tăng cường Dữ liệu)**

  Back-translation (Dịch ngược): Sử dụng một mô hình dịch ngược để dịch dữ liệu đơn ngữ (monolingual data) trở lại ngôn ngữ nguồn, từ đó tạo thêm các cặp câu huấn luyện mới. Điều này mở rộng đáng kể kích thước tập huấn luyện và giúp mô hình trở nên mạnh mẽ hơn.

- **Pre-trained Embeddings**

  Sử dụng các vector nhúng từ các mô hình đã được huấn luyện trên tập dữ liệu lớn như GloVe hoặc FastText để khởi tạo lớp Embedding của Encoder/Decoder. Giúp làm cho trọng số khởi tạo tốt giúp mô hình hội tụ nhanh hơn, yêu cầu ít dữ liệu huấn luyện hơn và cải thiện hiệu suất, đặc biệt đối với các từ hiếm.

VII. KẾT LUẬN

7.1. Tóm tắt kết quả đạt được

Đồ án đã hoàn thành việc triển khai mô hình dịch máy Anh-Pháp sử dụng kiến trúc Encoder-Decoder với LSTM và context vector cố định, tuân thủ yêu cầu đề tài (không sử dụng thư viện seq2seq có sẵn như torchtext.legacy hoặc transformers). Các kết quả chính bao gồm:

- **Triển khai mô hình và huấn luyện thành công:** Sử dụng PyTorch để xây dựng Encoder (LSTM 2 lớp, embedding 256, hidden 512, dropout 0.5) và Decoder tương ứng. Dữ liệu huấn luyện từ file train.en/train.fr (khoảng 29.000 cặp câu), với vocab giới hạn 10.000 từ. Huấn luyện 20 epochs với batch size 128, teacher forcing ratio 0.5, và early stopping (patience 3), đạt train loss khoảng 2.5-3.0 và val loss khoảng 3.0-3.5 (PPL ~20-30).
- **Đánh giá hiệu suất:** BLEU score trung bình trên tập test là 0.28 (28%), tính bằng NLTK corpus_bleu. Mô hình dịch cơ bản các câu ngắn, nhưng gặp lỗi với câu dài hoặc từ hiếm (OOV xử lý bằng <unk>).
- **Phân tích và minh họa:** Đã thực hiện dịch 5 ví dụ mẫu trên tập test, phân tích lỗi và tổng hợp vấn đề. Ví dụ: Câu "A girl is walking in a field." được dịch thành "une fille est en train de <unk> dans un champ .".
- **Công cụ hỗ trợ:** Sử dụng Spacy cho tokenize, packing sequence cho hiệu quả huấn luyện, và greedy decoding cho inference (max_len=50).

Tổng thể, đồ án đạt mục tiêu hiểu và triển khai kiến trúc cơ bản, xử lý dữ liệu chuỗi song song, và đánh giá bằng BLEU score.

7.2. Hạn chế hiện tại

Mô hình vẫn tồn tại một số hạn chế do kiến trúc cơ bản và yêu cầu đề tài (context vector cố định, không attention):

- **Context bottleneck:** Context vector cố định (hidden/cell cuối cùng của Encoder) không nén hiệu quả thông tin từ câu dài, dẫn đến mất ngữ cảnh và dịch sai (ví dụ: câu dài bị cắt nghĩa hoặc lặp từ).

- **Xử lý từ hiếm (OOV):** Vocab giới hạn 10.000 từ khiến từ ngoài vocab bị thay bằng <unk>, gây lỗi nghiêm trọng (ví dụ: "homework" → <unk>, làm mất ngữ nghĩa toàn câu).
- **Phương pháp decoding:** Greedy decoding chỉ chọn token xác suất cao nhất từng bước, dễ rơi vào local optimal, dẫn đến câu dịch thiếu mượt mà hoặc không tối ưu.
- **Hiệu suất tổng thể:** BLEU score chỉ 0.28, thấp so với mô hình hiện đại (do thiếu attention và subword). Huấn luyện chậm trên CPU/GPU cơ bản, và chưa xử lý tốt dữ liệu thực tế (câu dài, đa ngữ cảnh).

Các hạn chế này được phân tích chi tiết qua bảng lỗi trong phần thực nghiệm.

7.3. Hướng phát triển trong tương lai gần

Dựa trên phân tích lỗi, đồ án có thể cải tiến để nâng cao chất lượng dịch:

- **Thêm cơ chế attention:** Áp dụng Bahdanau hoặc Luong attention để Decoder truy cập toàn bộ hidden states của Encoder, giảm bottleneck và cải thiện BLEU score lên 30-40%.
- **Cải thiện decoding:** Thay greedy bằng beam search (beam size 3-5) để tìm kiếm global optimal, giảm lỗi chọn lọc kém.
- **Xử lý OOV:** Sử dụng subword tokenization (BPE - Byte Pair Encoding) để biểu diễn từ hiếm dưới dạng subword, kết hợp vocab lớn hơn (20.000-50.000).
- **Mở rộng mô hình:** Chuyển sang Transformer (Encoder-Decoder với self-attention) để xử lý song song, nhanh hơn LSTM. Thử nghiệm dataset lớn hơn (như WMT) hoặc đa ngôn ngữ (Anh-Đức theo đề tài mở rộng).
- **Tối ưu hóa:** Tích hợp pre-trained embeddings (Word2Vec/GloVe) và huấn luyện trên GPU mạnh hơn để giảm thời gian.

Các hướng này có thể triển khai trong 1-2 tháng, tập trung vào PyTorch và thư viện Hugging Face (không dùng seq2seq legacy).

TÀI LIỆU THAM KHẢO

[1] Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks.

[2] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate.

[3] Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002). BLEU: A Method for Automatic Evaluation of Machine Translation.

[4] Koehn, P. (2004). Statistical Significance Tests for Machine Translation Evaluation.

[5] PyTorch. (2017). NLP From Scratch: Translation with a Sequence to Sequence Network and Attention.

[6] Bentrevett. (n.d.). PyTorch Seq2Seq Tutorials.

[7] Wang, W., & Chen, J. (2016). An Experimental Study of LSTM Encoder-Decoder Model for Text Summarization.

[8] TorchText Legacy Documentation (2024). Multi30k dataset và cách xử lý dữ liệu cũ. https://torchtext.readthedocs.io/en/latest/

[9] spaCy Official Models (2024). en_core_web_sm và fr_core_news_sm. https://spacy.io/models/en và https://spacy.io/models/fr

[10] Britz, D. (2017–2023). Sequence to Sequence Learning with PyTorch – Tutorials (rất nhiều phần code tham khảo). https://github.com/bentrevett/pytorch-seq2seq

[11] Harvard NLP Group (2016). The Annotated Encoder-Decoder with Attention. http://nlp.seas.harvard.edu/2018/04/03/attention.html

PHỤ LỤC

Phụ lục A. Chương trình nguồn

```
# ĐỒ ÁN: DỊCH MÁY ANH-PHÁP VỚI MÔ HÌNH ENCODER-DECODER LSTM
**Học kỳ:** HK1 / 2025-2026
**Đề tài:** Triển khai mô hình dịch máy Anh-Pháp sử dụng kiến trúc Encoder-Decoder với LSTM và Context Vector cố định.

## I. CÀI ĐẶT THƯ VIỆN VÀ THIẾT LẬP MÔI TRƯỜNG
**Yêu cầu:** Triển khai bằng Python và PyTorch, không sử dụng thư viện seq2seq có sẵn.

# Cài đặt Thư viện và Thiết lập Môi trường
import os
import random
import time
import math
import numpy as np
from collections import Counter
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
import spacy
from nltk.translate.bleu_score import corpus_bleu

# Cài đặt các thư viện bắt buộc (Chạy lần đầu)
!pip install torch spacy nltk numpy matplotlib

# Tải các mô hình ngôn ngữ Spacy (Chạy lần đầu)
# Lưu ý: Các file này phải có sẵn hoặc được tải trong bước chuẩn bị.
!python -m spacy download en_core_web_sm
!python -m spacy download fr_core_news_sm

# Tải Spacy model
```

```
try:
    spacy_en = spacy.load("en_core_web_sm")
    spacy_fr = spacy.load("fr_core_news_sm")
    print("Spacy models loaded successfully.")
except OSError:
    print("Vui lòng chạy lệnh cài đặt Spacy: !python -m spacy download
en_core_web_sm và fr_core_news_sm")
    # exit()

# Khởi tạo thư mục (Giả định người dùng đã tải 6 file dữ liệu vào data/raw)
!mkdir -p data/raw
!mkdir -p checkpoints
print("Đã tạo thư mục data/raw và checkpoints.")

# Device và Seed
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
SEED = 1234
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
if torch.cuda.is_available():
    torch.cuda.manual_seed(SEED)
    torch.backends.cudnn.deterministic = True

print(f"Sử dụng thiết bị: {DEVICE}")

# Token đặc biệt và giới hạn Vocab
UNK_IDX, PAD_IDX, SOS_IDX, EOS_IDX = 0, 1, 2, 3
SPECIAL_TOKENS = ['<unk>', '<pad>', '<sos>', '<eos>']
VOCAB_SIZE_LIMIT = 10000

# Các hàm Utility (Tải dữ liệu, Tokenizer, Custom Vocab)

def tokenize_en(text):
    """Tokenize tiếng Anh"""
    return [tok.text.lower() for tok in spacy_en.tokenizer(text)]

def tokenize_fr(text):
    """Tokenize tiếng Pháp"""
    return [tok.text.lower() for tok in spacy_fr.tokenizer(text)]
```

```
def load_data(filepath_en, filepath_fr):
    """Đọc dữ liệu từ file và trả về danh sách các cặp câu (en, fr)"""
    if not os.path.exists(filepath_en) or not os.path.exists(filepath_fr):
        # Trả về list rỗng nếu file không tồn tại
        return []

    with open(filepath_en, encoding='utf-8') as f_en, open(filepath_fr,
encoding='utf-8') as f_fr:
        raw_pairs = []
        for en_line, fr_line in zip(f_en, f_fr):
            raw_pairs.append((en_line.strip(), fr_line.strip()))
    return raw_pairs

# Xây dựng và quản lý từ vựng
class CustomVocab:
    """Lớp Vocab tùy chỉnh để thay thế torchtext.vocab"""
    def __init__(self, token_list, specials, default_index):
        self.stoi = {token: i for i, token in enumerate(specials)}
        current_index = len(specials)

        for token in token_list:
            if token not in self.stoi:
                self.stoi[token] = current_index
                current_index += 1

        self.itos = specials + [token for token in token_list if token not in specials]
        self.default_index = default_index

    def get_stoi(self):
        return self.stoi

    def get_itos(self):
        return self.itos

    def __len__(self):
        return len(self.itos)

def get_token_frequency(data_pairs, tokenizer, lang_index):
    """Tính tần suất của tất cả token trong tập dữ liệu."""
```

```
    counter = Counter()
    for pair in data_pairs:
        text = pair[lang_index]
        counter.update(tokenizer(text))
    return counter

# XỬ LÝ DỮ LIỆU HOÀN CHỈNH
# 1. Tải dữ liệu huấn luyện (dùng để xây dựng vocab)
train_pairs = load_data('data/raw/train.en', 'data/raw/train.fr')
if not train_pairs:
    raise FileNotFoundError("Lỗi: Không tìm thấy dữ liệu. Vui lòng tải 6 file vào
thư mục data/raw.")
def data_process(text_pairs, vocab_src, vocab_trg):
    """Chuyển đổi các cặp câu (text) sang cặp tensor."""
    data = []
    src_stoi = vocab_src.get_stoi()
    trg_stoi = vocab_trg.get_stoi()

    for en_text, fr_text in text_pairs:
        # Source (tiếng Anh): Xử lý OOV bằng UNK_IDX
        src_tensor = [src_stoi.get(token, UNK_IDX) for token in
tokenize_en(en_text)]
        # Target (tiếng Pháp): Thêm SOS và EOS
        trg_tensor = [SOS_IDX] + [trg_stoi.get(token, UNK_IDX) for token in
tokenize_fr(fr_text)] + [EOS_IDX]

        if src_tensor and trg_tensor:
            data.append((torch.tensor(src_tensor, dtype=torch.long),
torch.tensor(trg_tensor, dtype=torch.long)))
    return data

# 2. Hàm Collate và DataLoader
def custom_collate_fn(batch):
    """
    Collate function: Sắp xếp batch theo độ dài giảm dần (cho Packing) và Padding.
    """
    batch.sort(key=lambda x: len(x[0]), reverse=True)

    src_tensors = [item[0] for item in batch]
    trg_tensors = [item[1] for item in batch]
```

```
    # Padding (batch_first=False) -> [seq_len, batch_size]
    src_padded = pad_sequence(src_tensors, padding_value=PAD_IDX,
batch_first=False)
    trg_padded = pad_sequence(trg_tensors, padding_value=PAD_IDX,
batch_first=False)

    src_lengths = torch.tensor([len(tensor) for tensor in src_tensors],
dtype=torch.long)

    return src_padded, trg_padded, src_lengths
 Khởi tạo DataLoader
BATCH_SIZE = 128
train_iterator = DataLoader(train_data, batch_size=BATCH_SIZE,
collate_fn=custom_collate_fn)
valid_iterator = DataLoader(valid_data, batch_size=BATCH_SIZE,
collate_fn=custom_collate_fn)
test_iterator = DataLoader(test_data, batch_size=BATCH_SIZE,
collate_fn=custom_collate_fn)

print(f"Kích thước tập Train: {len(train_data)}, Val: {len(valid_data)}, Test:
{len(test_data)}")

# XÂY DỰNG MÔ HÌNH
# 1. Encoder (LSTM + Packing)
class Encoder(nn.Module):
    def __init__(self, input_dim, emb_dim, hid_dim, n_layers, dropout):
        super().__init__()

        self.hid_dim = hid_dim
        self.n_layers = n_layers

        self.embedding = nn.Embedding(input_dim, emb_dim)
        self.rnn = nn.LSTM(emb_dim, hid_dim, n_layers, dropout=dropout)
        self.dropout = nn.Dropout(dropout)

    def forward(self, src, src_len):

        embedded = self.dropout(self.embedding(src))
```

```
        lengths = src_len.cpu().tolist()
        packed_embedded = nn.utils.rnn.pack_padded_sequence(embedded, lengths,
enforce_sorted=True)
        packed_outputs, (hidden, cell) = self.rnn(packed_embedded)

        return hidden, cell

# 2. Decoder
class Decoder(nn.Module):
    def __init__(self, output_dim, emb_dim, hid_dim, n_layers, dropout):
        super().__init__()

        self.output_dim = output_dim
        self.hid_dim = hid_dim
        self.n_layers = n_layers

        self.embedding = nn.Embedding(output_dim, emb_dim)
        self.rnn = nn.LSTM(emb_dim, hid_dim, n_layers, dropout=dropout)
        self.fc_out = nn.Linear(hid_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, trg_input, hidden, cell):

        trg_input = trg_input.unsqueeze(0)

        embedded = self.dropout(self.embedding(trg_input))

        output, (hidden, cell) = self.rnn(embedded, (hidden, cell))

        prediction = self.fc_out(output.squeeze(0))

        return prediction, hidden, cell

# 3. Seq2Seq
class Seq2Seq(nn.Module):
    def __init__(self, encoder, decoder, device):
        super().__init__()

        self.encoder = encoder
        self.decoder = decoder
```

```
        self.device = device

        assert encoder.hid_dim == decoder.hid_dim, \
            "Hidden dimensions of encoder and decoder must be equal!"
        assert encoder.n_layers == decoder.n_layers, \
            "Encoder and decoder must have equal number of layers!"

    def forward(self, src, trg, src_len, teacher_forcing_ratio = 0.5):

        trg_len, batch_size = trg.shape
        trg_vocab_size = self.decoder.output_dim

        outputs = torch.zeros(trg_len, batch_size, trg_vocab_size).to(self.device)

        hidden, cell = self.encoder(src, src_len)

        trg_input = trg[0, :]

        for t in range(1, trg_len):
            prediction, hidden, cell = self.decoder(trg_input, hidden, cell)

            outputs[t] = prediction

            top1 = prediction.argmax(1)

            teacher_force = random.random() < teacher_forcing_ratio

            trg_input = trg[t, :] if teacher_force else top1

        return outputs

# HUẤN LUYỆN
# 1. Khởi tạo Mô hình, Loss và Optimizer
# Tham số Mô hình
EMB_DIM = 256
HID_DIM = 512
N_LAYERS = 2
ENC_DROPOUT = 0.5
DEC_DROPOUT = 0.5
TEACHER_FORCING_RATIO = 0.5
```

```
CLIP = 1.0
N_EPOCHS = 20
PATIENCE = 3
enc = Encoder(INPUT_DIM, EMB_DIM, HID_DIM, N_LAYERS,
ENC_DROPOUT)
dec = Decoder(OUTPUT_DIM, EMB_DIM, HID_DIM, N_LAYERS,
DEC_DROPOUT)

model = Seq2Seq(enc, dec, DEVICE).to(DEVICE)

# Loss & Optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss(ignore_index=PAD_IDX)

print(f"Mô hình Seq2Seq đã được khởi tạo trên {DEVICE}.")
def count_parameters(model):
        return sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f'Mô hình có {count_parameters(model):,} tham số.')

# 2. Hàm Huấn luyện và Hàm Đánh giá
# --- Hàm Train ---
def train(model, iterator, optimizer, criterion, clip):
    model.train()
    epoch_loss = 0
    for i, (src, trg, src_len) in enumerate(iterator):
        src, trg = src.to(DEVICE), trg.to(DEVICE)
        optimizer.zero_grad()
        output = model(src, trg, src_len, TEACHER_FORCING_RATIO)

        output_dim = output.shape[-1]
        output = output[1:].view(-1, output_dim)
        trg = trg[1:].view(-1)

        loss = criterion(output, trg)
        loss.backward()

        # Gradient Clipping
        torch.nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()
```

```
        epoch_loss += loss.item()

    return epoch_loss / len(iterator)

# --- Hàm Evaluate ---
def evaluate(model, iterator, criterion):
    model.eval()
    epoch_loss = 0
    with torch.no_grad():
        for i, (src, trg, src_len) in enumerate(iterator):
            src, trg = src.to(DEVICE), trg.to(DEVICE)

            # Tắt teacher forcing khi đánh giá (ratio = 0)
            output = model(src, trg, src_len, 0)

            output_dim = output.shape[-1]
            output = output[1:].view(-1, output_dim)
            trg = trg[1:].view(-1)

            loss = criterion(output, trg)
            epoch_loss += loss.item()

    return epoch_loss / len(iterator)

# 3. Chạy Huấn luyện Chính và Lưu Checkpoint
train_losses = []
valid_losses = []

print("\n--- BẮT ĐẦU HUẤN LUYỆN ---")
best_valid_loss = float('inf')
patience_counter = 0

for epoch in range(N_EPOCHS):
    start_time = time.time()

    train_loss = train(model, train_iterator, optimizer, criterion, CLIP)
    valid_loss = evaluate(model, valid_iterator, criterion)

    train_losses.append(train_loss)
    valid_losses.append(valid_loss)
```

```
    end_time = time.time()
    epoch_mins = int((end_time - start_time) / 60)
    epoch_secs = int((end_time - start_time) % 60)

    if valid_loss < best_valid_loss:
        best_valid_loss = valid_loss
        patience_counter = 0
        # Lưu checkpoint mô hình
        torch.save(model.state_dict(), 'checkpoints/best_model.pth')
        print(f'Epoch: {epoch+1:02} | Time: {epoch_mins}m {epoch_secs}s | Val
Loss IMPROVED. Saving model.')
    else:
        patience_counter += 1
        print(f'Epoch: {epoch+1:02} | Time: {epoch_mins}m {epoch_secs}s | Val
Loss did not improve. Patience: {patience_counter}/{PATIENCE}')

        # Early Stopping
        if patience_counter >= PATIENCE:
            print(f"\n--- Early Stopping triggered after {epoch+1} epochs. ---")
            break

    print(f'\tTrain Loss: {train_loss:.3f} | Train PPL: {math.exp(train_loss):7.3f}')
    print(f'\t  Val Loss: {valid_loss:.3f} |   Val PPL: {math.exp(valid_loss):7.3f}')

print("\n--- KẾT THÚC HUẤN LUYỆN ---")

# 4. Vẽ biểu đồ Loss
plt.figure(figsize=(10, 6))
plt.plot(train_losses, label='Train Loss')
plt.plot(valid_losses, label='Validation Loss')
plt.title('Biểu đồ Loss qua các Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss (CrossEntropy)')
plt.legend()
plt.grid(True)
plt.show()

# ĐÁNH GIÁ MÔ HÌNH
# 1. Hàm Dịch Câu
```

```
def translate(sentence, model, vocab_src, vocab_trg, max_len=50):
    """
    Hàm dịch một câu tiếng Anh -> tiếng Pháp (Yêu cầu bắt buộc)
    Sử dụng Greedy Decoding (Yêu cầu bắt buộc)
    """
    model.eval()

    # 1. Tokenize -> tensor
    tokens = [token for token in tokenize_en(sentence)]
    indexes = [vocab_src.get_stoi().get(token, UNK_IDX) for token in tokens]

    src_tensor = torch.LongTensor(indexes).to(DEVICE)
    src_tensor = src_tensor.unsqueeze(1) # [src_len, 1]
    src_len = torch.LongTensor([len(indexes)])

    # 2. Encoder
    with torch.no_grad():
        hidden, cell = model.encoder(src_tensor, src_len)

    # 3. Decoder
    trg_input = torch.LongTensor([SOS_IDX]).to(DEVICE)
    translated_tokens = []

    for t in range(max_len): # Max length 50
        with torch.no_grad():
            output, hidden, cell = model.decoder(trg_input, hidden, cell)

        # Greedy Decoding
        pred_token = output.argmax(1).item()

        if pred_token == EOS_IDX:
            break

        translated_tokens.append(vocab_trg.get_itos()[pred_token])
        trg_input = torch.LongTensor([pred_token]).to(DEVICE)

    return " ".join(translated_tokens)

# 2. Hàm Tính BLEU Score
def calculate_bleu(model, test_pairs_raw, vocab_src, vocab_trg):
```

```
    """Tính BLEU score trung bình trên tập test (Yêu cầu báo cáo) """
    all_references = []
    all_translations = []

    for src_sentence_raw, trg_sentence_raw in test_pairs_raw:
        predicted_sentence = translate(src_sentence_raw, model, vocab_src,
vocab_trg)
        all_translations.append(predicted_sentence.split())

        # Reference (Tokenize)
        reference = tokenize_fr(trg_sentence_raw)
        all_references.append([reference])

    bleu = corpus_bleu(all_references, all_translations)
    return bleu

# 3. Đánh giá cuối cùng
# Tính BLEU score trên tập test
if 'test_pairs' in locals() and test_pairs:
    bleu_score = calculate_bleu(model, test_pairs, vocab_src, vocab_trg)
    print(f"\nBLEU Score trên tập Test: {bleu_score*100:.2f}%")
else:
    print("\nKhông thể tính BLEU Score do thiếu dữ liệu test.")

# Dịch 5 ví dụ đầu tiên từ tập test
print("\n--- 5 VÍ DỤ DỊCH MẪU TỪ TẬP TEST ---")
translations = []
# Lấy 5 cặp câu đầu tiên từ test_pairs (English, French)
for i, (eng_sentence, fr_sentence_ref) in enumerate(test_pairs[:5]):
    translation = translate(eng_sentence, model, vocab_src, vocab_trg)
    translations.append(translation)
    print(f"{i+1}. ENG: {eng_sentence}")
    print(f"   REF: {fr_sentence_ref}") # In thêm câu tham chiếu tiếng Pháp
    print(f"   FRA: {translation}")
```

Phụ lục B. Liên kết lưu trữ Code, Checkpoint và Data

1. Google Drive: https://drive.google.com/drive/folders/1zhVzyqxgKi2Dt4mEXX4EU0mB_j1tN13z?usp=sharing

Nội dung:

- DoAn_NLP.ipynb (notebook hoàn chỉnh)
- checkpoints/best_model.pth (checkpoint mô hình có validation loss thấp nhất)
- data/raw (thư mục chứa các file .en, .fr đã được tải sẵn)
- NLP báo cáo.pdf

2. GitHub repository: https://github.com/Deku6305/sgu2025-2026_NLP.git

Nội dung:

- DoAn_NLP.ipynb (notebook hoàn chỉnh)
- checkpoints/best_model.pth (checkpoint mô hình có validation loss thấp nhất)
- data/raw (thư mục chứa các file .en, .fr đã được tải sẵn)
- NLP báo cáo.pdf
- README.md: Thông tin đồ án, thành viên.