ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN MÔN HỌC:

Vietnamese Correction

Môn học: Xử lý ngôn ngữ tự nhiên - CS221.O22

Giảng viên hướng dẫn: TS. Nguyễn Trọng Chỉnh

Sinh viên thực hiện:

Đoàn Nhật Tiến - 22521463
Trần Thượng Trường Sơn - 22521264
Nguyễn Minh Thiện - 22521391

MỤC LỤC

1. Giới thiệu đề tài

- Nhóm chọn đề tài Vietnamese correction (sửa lỗi chính tả Tiếng Việt) với mục tiêu phát triển mô hình học sâu để tự động phát hiện và sửa chữa lỗi chính tả trong các văn bản Tiếng Việt. Với đặc điểm phức tạp của ngôn ngữ Tiếng Việt, việc sai chính tả không chỉ gây khó hiểu trong các tin nhắn, mail hằng ngày mà còn ảnh hưởng đến chất lượng của các tài liệu chữ như sách, báo, truyện, bài báo cáo,...
- Xác định bài toán:
    + Input: một câu văn Tiếng Việt có thể có lỗi chính tả.
    + Output: một câu văn đúng chính tả.

2. Bộ ngữ liệu

- Dữ liệu được tổng hợp tại link github https://github.com/duyvuleo/VNTC.git với 27 chủ đề khác nhau. Sau đó dữ liệu này sẽ được tạo lỗi chính tả để tạo input cho mô hình.

Mẫu	Câu đúng chính tả	Mô tả lỗi
Xinđừn gọit lanha kinh doanh	Xin đừng gọi tôi là nhà kinh doanh	- Dấu cách sau chữ xin bị mất - "đừng" thiếu chữ 'g' - Dấu cách sau từ "là" bị thiếu - Từ "là" mất dấu huyền - Từ "nhà" mất dấu huyền
Đâycũng là cơ hiểể	Đây cũng là cơ hội để	- Dấu cách sau chữ đây bị thiếu - Từ "hội" mất chữ 'ộ'
Song khiếm khuyếtliên quan đến xản phẩm	Song khiếm khuyết liên quan đến sản phẩm	- Mất dấu cách sau từ "khuyết" - Chữ 's' trong từ "sản" bị đổi thành 'x'
Khach sạn Khải Hoàn cắt hợp đồ với tyền đạoDạiLâm.	Khách sạn Khải Hoàn cắt hợp đồng với tiền đạo Đại Lâm.	- Từ "tiền" bị đổi chữ 'i' thành 'y' - Mất dấu cách sau từ đạo và Đại - Từ "Đại" bị đổi 'Đ' thành 'D'
cho con đihọc thêm đến 9h tối	cho con đi học thêm đến 9h tối	- Mất dấu cách sau từ "đi"
Nghệ sỹ dc iêu thíchnhất	Nghệ sĩ được yêu thích nhất	- Chữ 'i' trong từ "sĩ" bị đổi thành 'y' - Mất dấu cách sau từ thích

Sơn màu dỏ lên cácô màu đtrên mẫu	Sơn màu đỏ lên các ô màu đỏ trên mẫu	- Chữ ‘đ’ trong từ “đỏ’ bị đổi thành ‘d’ - Mất dấu cách sau từ “các” và chữ “đ” (đáng lẻ phải là từ “đỏ” nhưng bị mất chữ ‘ỏ’)
Sẽcó phim hoạt hình Tây Du Ký mang tầm cỡ uốc ế .	Sẽ có phim hoạt hình Tây Du Ký mang tầm cỡ quốc tế.	- Mất dấu cách sau từ “Sẽ” - Thiếu 2 chữ ‘q’ và ‘t’ trong 2 từ “quốc” và “tế”
Hientại , Ialy đứn đầu bảng 5	Hiện tại, Italy đứng đầu bảng 5	- Chữ ‘ệ’ trong từ “Hiện” bị mất hết dấu thành chữ ‘e’ - Từ “đứng” bị mất chữ ‘d’
Nguoi dùng có thể vào đây để tải bản dùng thử	Người dùng có thể vào đây để tải bản dùng thử	- Từ “người” bị mất hết dấu ở 2 chữ ‘ư’,’ơ’ thành ‘u’,’o’
Thực ra chính bạn cũng cs thể cải thiện đôi chancủa mình để chúng nguột nà hon .	Thực ra chính bạn cũng có thể cải thiện đôi chân của mình để chúng nuột nà hơn.	- Lỗi viết tắt ‘cs’ - cũng - Mất dấu câu - Mất khoảng trắng
Nhưng mẹ ơi , mẹcó biết lúc ấy con nghĩthế ào không.	Nhưng mẹ ơi, mẹ có biết lúc ấy con nghĩ thế nào không.	- Mất khoảng cách giữa hai từ - Mất kí tự ‘ào’ - ‘nào’
Oracle và chính quyền Mỹ tanh cxi quanh ịnh nghĩa thị trường .	Oracle và chính quyền Mỹ tranh cãi quanh định nghĩa thị trường.	- Mất kí tự
Net nhẫn nhục, cam chịu của bà Vũ làm người ta có thể nhóy lòng	Nét nhẫn nhục, cam chịu của bà Vũ làm người ta có thể nhói lòng.	- Nhầm lẫn giữa ‘y’ và ‘i’ - nhosy - nhói
Đã hai tuần kể từ khi chúng tôi nsi chuyện lần cuối .	Đã hai tuần kể từ khi chúng tôi nói chuyện lần cuối.	- Gõ nhầm nsi - nói
T muốn mở lớp dạy pim truyền hình .	Tôi muốn mở lớp dạy phim truyền hình.	- Lỗi viết tắt ‘T’ với ‘Tôi’ - Mất kí tự pim - phim
Nghệ si Thanh Hoài lam ngh bằng cái tâm trong sang .	Nghệ sĩ Thanh Hoài làm nghề bằng cái tâm trong sáng.	-Sai dấu hỏi với dấu ngã - Bị mất từ ngh - nghề - Mất dấu câu lam - làm, sang - sáng
Cáctrường học tạiAnh quốc đang gấp rut tuyển sinh cho năm học mới .	Các trường học tại Anh quốc đang gấp rút tuyển sinh cho năm học mới.	- Lỗi mất dấu cách - Lỗi mất dấu rut - rút

Phí nha ở và ăn (2 bữa trính) tại 1 gia đình người ban xứ.	Phí nhà ở và ăn (2 bữa chính) tại 1 gia đình người bản xứ.	- Mất dấu câu ‘nha’ - ‘nhà’ - Nhầm lẫn giữa “tr’ - ‘ch’ - Quên không gõ dấu ‘người’ - ‘ngươi;
Con số của Charb Sorn không được nêu lên sau mỗi lần chọn lựa .	Con số của Charb Sorn không được nêu lên sau mỗi lần chọn lựa.	- Đúng chính tả
Cả nơiheo hút đã thấp thoáng nhwng chảo ăngten parabol .	Cả nơi heo hút đã thấp thoáng những chảo ăngten parabol.	- Không có khoảng trắng - Sai ‘nhwng’ - ‘nhưng’
Ông giữ chức vụ này suốt từ năm 1998 và từ chức vào năm 2003 .	Ông giữ chức vụ này suốt từ năm 1998 và từ chức vào năm 2003.	- Đúng chính tả
Nhưng thật kỳ lạ, cátr sắp xếp ấy lại ... khôngra nhạc .	Nhưng thật kỳ lạ, cách sắp xếp ấy lại... không ra nhạc.	- Gõ sai - Không có khoảng trắng
Cóbằng cử nhân giỏi/xuất xắccấcnganh kỹ thuật robot , cơ khí , vật lý	Có bằng cử nhân giỏi/xuất sắc các ngành kỹ thuật robot, cơ khí, vật lý.	- Không có khoảng trắng - Nhầm lẫn giữa ‘x’ -’s’

3. Phương pháp.

3.1. Model Seq2Seq.

3.1.1. Giới thiệu về GRU.

- GRU (Gated Recurrent Unit) là một dạng mạng nơ-ron hồi quy (RNN) nâng cao, được thiết kế để giải quyết các vấn đề phổ biến của RNN như hiện tượng mất mát gradient và khó khăn trong việc duy trì thông tin dài hạn trong các chuỗi dữ liệu. GRU được giới thiệu bởi Kyunghyun Cho và cộng sự vào năm 2014 và trở thành một trong những mô hình phổ biến trong lĩnh vực xử lý ngôn ngữ tự nhiên và các ứng dụng dự đoán chuỗi.

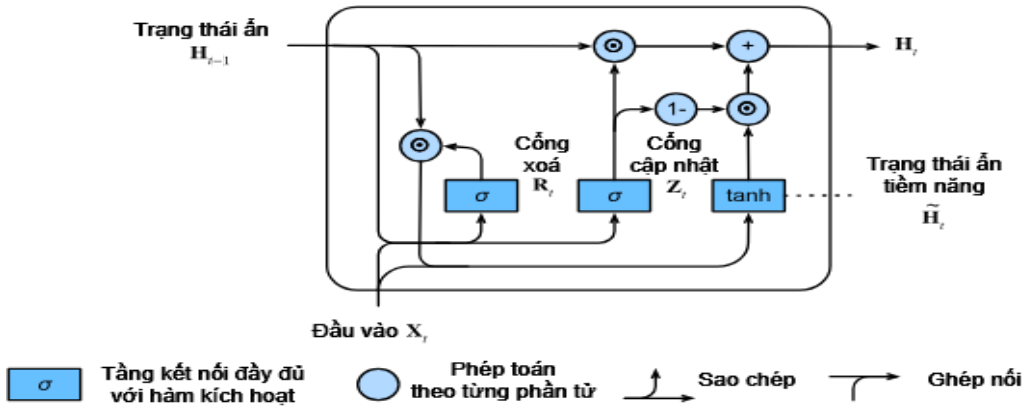
- Cấu trúc chính của GRU bao gồm:

  1. Trạng thái ẩn (Hidden State): Tương tự như RNN, GRU duy trì một trạng thái ẩn tại mỗi bước thời gian t. Trạng thái ẩn này chứa thông tin đã xem xét từ tất cả các bước thời gian trước đó và là nơi lưu trữ các biểu diễn ngữ nghĩa của chuỗi đang được xử lý.

  2. Cổng cập nhật (Update Gate):Cổng này quyết định mức độ nào thông tin mới sẽ được cập nhật vào trạng thái ẩn. Điều này giúp GRU quyết định liệu nên giữ lại

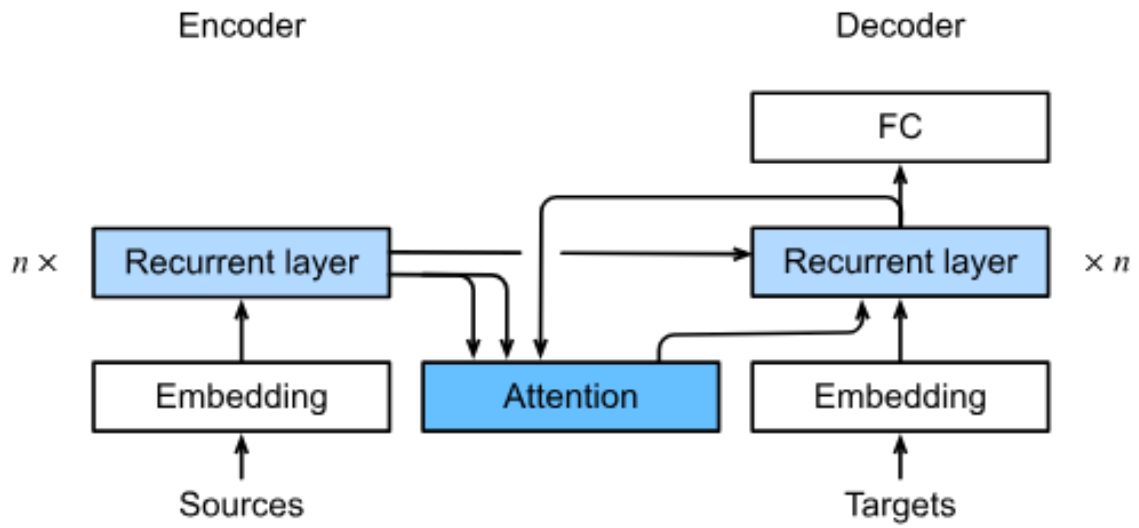
thông tin cũ hay cập nhật với thông tin mới. Cổng cập nhật có giá trị từ 0 đến 1, với 0 nghĩa là không cập nhật thông tin nào và 1 nghĩa là giữ lại toàn bộ thông tin mới.

3. Cổng reset (Reset Gate): Cổng này quyết định phần nào của trạng thái ẩn cũ nên được "đặt lại" (reset). Công dụng của cổng này là để quyết định xem liệu mô hình có nên quên đi một số thông tin cũ trong trạng thái ẩn hay không.



3.1.2. Cơ chế Attention.

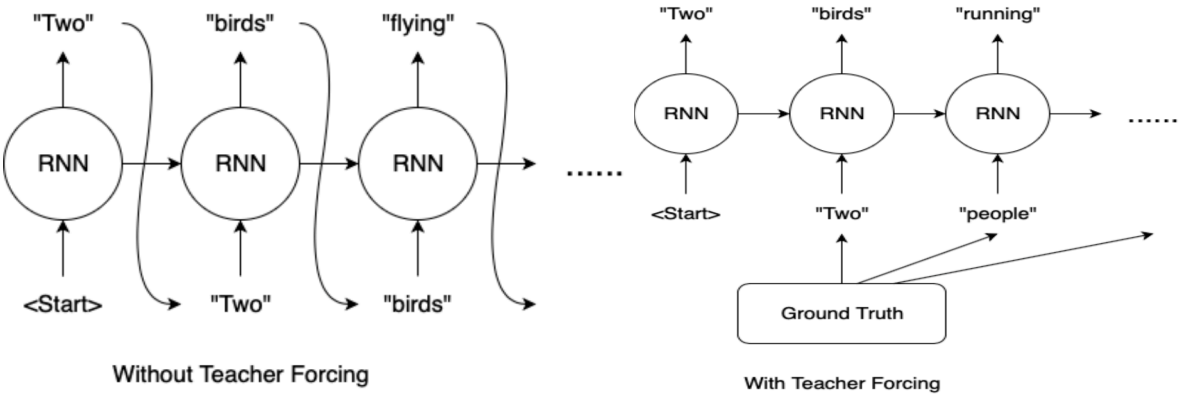
- Bahdanau Attention được giới thiệu bởi Dzmitry Bahdanau và cộng sự trong bài báo "Neural Machine Translation by Jointly Learning to Align and Translate" vào năm 2015. Đây là một phương pháp attention có khả năng học được các trọng số attention một cách tự động và cho phép mô hình tập trung vào các phần tử quan trọng của chuỗi đầu vào một cách linh hoạt. Mục tiêu của nó là cải thiện hiệu năng của mô hình Seq2Seq bằng cách thay đổi đầu vào của Decoder với các thông tin từ Input Sequence.
- Khi dự đoán một mã thông báo, nếu không phải tất cả các token đầu vào đều có liên quan, mô hình sẽ căn chỉnh (hoặc tham dự) chỉ với các phần của chuỗi đầu vào có liên quan đến dự đoán hiện tại. Điều này đạt được bằng cách coi biến ngữ cảnh như một đầu ra của sự chú ý pooling.




```python
class Attention(nn.Module):

    def __init__(self, enc_hid_dim, dec_hid_dim):
        # enc_hid_dim = 512, dec_hid_dim = 512
        super().__init__()

        self.attn = nn.Linear((enc_hid_dim * 2) + dec_hid_dim, dec_hid_dim)
        self.v = nn.Linear(dec_hid_dim, 1, bias=False)

    def forward(self, hidden, encoder_outputs):
        """
        hidden: batch_size x hid_dim (4 x 512)
        encoder_outputs: src_len x batch_size x 2*hid_dim, (46 x 4 x 1024)
        outputs: batch_size x src_len (4 x 46)
        """

        batch_size = encoder_outputs.shape[1] # 4
        src_len = encoder_outputs.shape[0] # 46

        hidden = hidden.unsqueeze(1).repeat(1, src_len, 1) # (4 x 46 x 512)

        encoder_outputs = encoder_outputs.permute(1, 0, 2) # (4 x 46 x 1024)

        energy = F.relu(self.attn(torch.cat((hidden, encoder_outputs), dim=2))) # ( 4 x 46 x 512)

        attention = self.v(energy).squeeze(2) # 4 x 46

        return F.softmax(attention, dim=1)
```

3.1.3. Kỹ thuật Teacher Forcing.

- Những bài toán có nhiều phần nhỏ mà kết quả của phần trước ảnh hưởng trực tiếp đến kết quả của phần sau. Nếu bạn làm sai phần đầu tiên thì nhiều khả năng bạn sẽ làm sai kết quả của toàn bài.Đặc biệt trong các bài toán xử lý thông tin dạng chuỗi, việc dự đoán sai một phần tử có thể làm cả chuỗi phía sau bị sai lệch theo. Teacher Forcing khắc phục sự phụ thuộc giữa các phần tử trong chuỗi bằng cách đánh giá mô hình trên từng phần của bài toán lớn. Đáp án của phần trước được cung cấp làm dữ kiện cho phần tiếp theo.



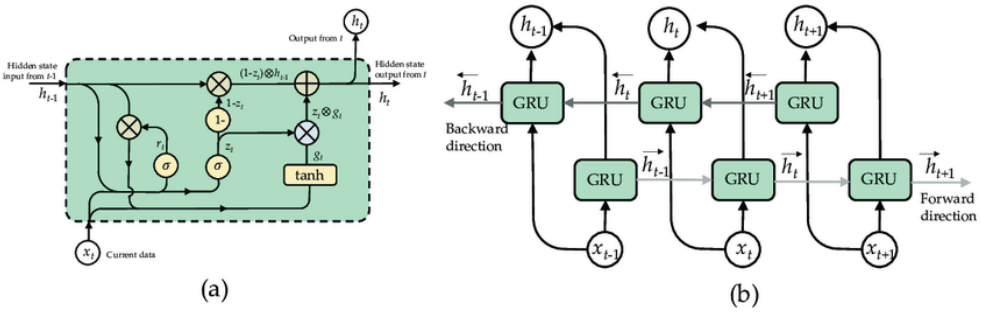
- Ưu điểm :
  Teacher Forcing giúp tăng tốc độ huấn luyện mô hình. Khi mới bắt đầu huấn luyện, mô hình có những dự đoán rất tồi. Nếu không sử dụng kỹ thuật này mô hình có thể sẽ bị lạc lối trong cả mớ các lựa chọn thử-sai cho đến khi ngẫu nhiên tìm được đáp án đúng. Với một chuỗi dài, thời gian để mô hình hội tụ sẽ tăng theo hàm mũ.
- Nhược điểm :
  Mặc dù vậy, Teacher Forcing có thể làm cho mô hình bị phụ thuộc vào các thông tin hỗ trợ. Kết quả là khi hoạt động trong thực tế (hoặc đánh giá trên tập test), kết quả của mô hình có thể kém hơn.

3.1.4. Giới thiệu về Seq2Seq.

- Seq2Seq (Sequence-to-Sequence) là một kiến trúc mô hình trong lĩnh vực học sâu, được thiết kế đặc biệt để xử lý các bài toán có đầu vào và đầu ra là các chuỗi dữ liệu có độ dài khác nhau. Kiến trúc Seq2Seq thường được sử dụng cho các bài toán như dịch máy, tóm tắt văn bản, chatbot và nhiều ứng dụng khác có liên quan đến xử lý ngôn ngữ tự nhiên.
- Seq2Seq bao gồm hai thành phần chính là:

  + Encoder (Bộ mã hóa):

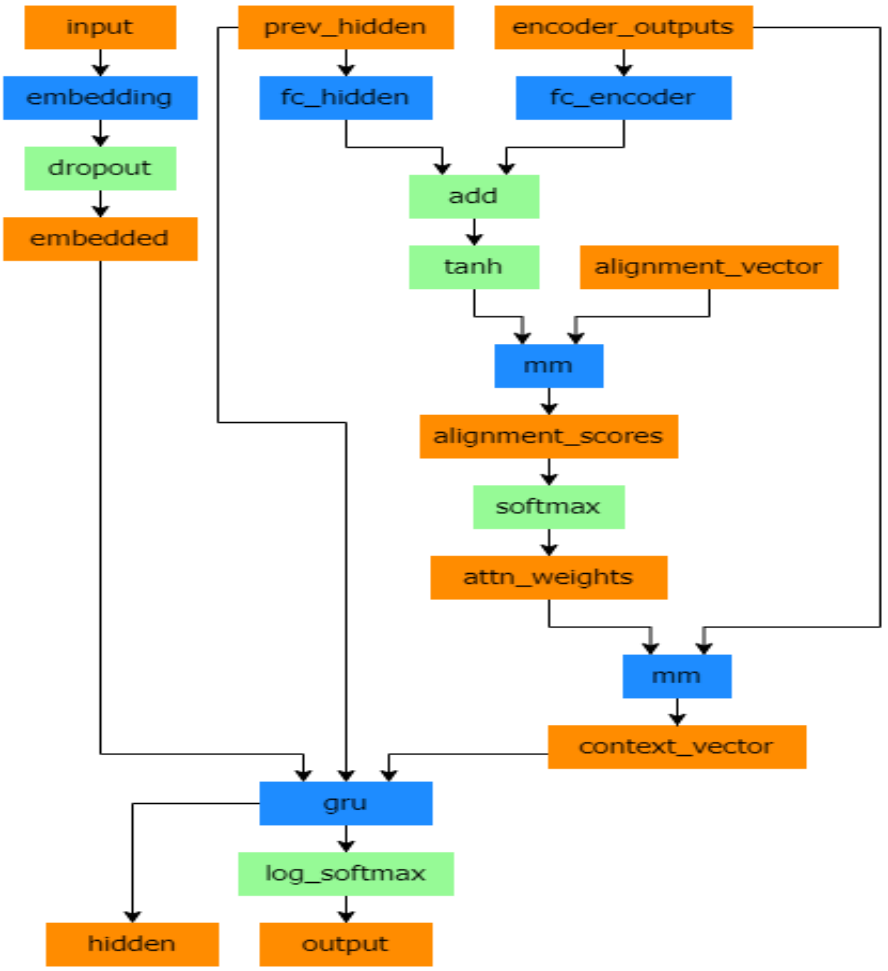
  - Bộ mã hóa là một mạng nơ-ron hồi quy (RNN) hoặc biến thể của nó (như GRU, LSTM) ở đây nhóm dùng BiGRU có nhiệm vụ chuyển đổi đầu vào (ví dụ: câu văn bản) thành một vectơ trạng thái ẩn. BiGRU là một phiên bản mở rộng của GRU, trong đó có hai lớp GRU chạy song song: một lớp đọc chuỗi dữ liệu từ đầu đến cuối (forward direction) và lớp còn lại đọc chuỗi dữ liệu từ cuối về đầu (backward direction). Kết quả từ hai lớp này được kết hợp lại để tạo ra một biểu diễn phong phú hơn của dữ liệu.
  - Ví dụ : forward là câu “Tôi iêu bạn” và câu backward là câu “nạb uêi iôT”



- Mỗi kí tự trong câu được biểu diễn bằng một vectơ embedding, sau đó được đưa vào bộ mã hóa để sinh ra trạng thái ẩn cuối cùng của bộ mã hóa.
- Trạng thái ẩn này chứa các thông tin quan trọng về câu đầu vào, giúp bộ giải mã hiểu và sinh ra đầu ra phù hợp.

```python
class Encoder(nn.Module):
    def __init__(self, input_dim, emb_dim, enc_hid_dim, dec_hid_dim, dropout):
        # input_dim = len(vocab) = 232 , emb_dim = 256, enc_hid_dim = 512, dec_hid_dim = 512, dropout = 0.5
        super().__init__()
        # embedding các kí tự trong scr thành vector có số chiều là (232,256)
        self.embedding = nn.Embedding(input_dim, emb_dim)
        self.rnn = nn.GRU(emb_dim, enc_hid_dim, bidirectional=True)
        self.fc = nn.Linear(enc_hid_dim * 2, dec_hid_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, src):
        """
        src: src_len x batch_size (46 x 4)
        outputs: src_len x batch_size x 2*hid_dim  (bidirectional) (46 x 4 x 2*512)
        hidden: batch_size x hid_dim (4 x 512)
        """
        # embedding các kí tự trong scr thành vector có số chiều là (232,256)
        embedded = self.dropout(self.embedding(src))
        outputs, hidden = self.rnn(embedded)

        hidden = F.tanh(self.fc(torch.cat((hidden[-2, :, :], hidden[-1, :, :]), dim=1)))

        return outputs, hidden # (46 x 4 x 1024) , (4 x 512)
```

+ Decoder-Attention (Bộ giải mã):



- Bộ giải mã là một mạng nơ-ron hồi quy khác (cũng có thể là RNN, GRU, LSTM) ở đây nhóm dùng GRU, đầu vào của bộ giải mã (decoder) tại mỗi bước bao gồm hai thành phần chính: câu target và hidden, output của encoder
- Bộ giải mã được huấn luyện để dự đoán các từ tiếp theo trong chuỗi đầu ra dựa trên trạng thái ẩn từ bộ mã hóa và từ trước đó đã sinh ra.
- Tính toán trọng số attention giữa trạng thái ẩn hiện tại của Decoder và tất cả các trạng thái ẩn của Encoder, tạo context vector. Sử dụng context vector và trạng thái ẩn hiện tại để tạo ra từ tiếp theo trong chuỗi đầu ra.
- Thay vì sử dụng từ dự đoán, mô hình sử dụng từ thật sự từ chuỗi đầu ra để làm đầu vào cho bước tiếp theo (teacher forcing)
- Quá trình dự đoán này diễn ra bước đầu tiên một kí tự một, bắt đầu bằng kí tự mã hóa đặc biệt (ví dụ như 'SOS' - Start of Sentence) và kết thúc khi gặp kí tự dừng (ví dụ như 'EOS' - End of Sentence) hoặc khi đạt đến một số giới hạn về độ dài (MAXLEN)

```python
class Decoder(nn.Module):
    def __init__(self, output_dim, emb_dim, enc_hid_dim, dec_hid_dim, dropout, attention):
        super().__init__()

        self.output_dim = output_dim # output = input_dim = len(Vocab) = 232
        self.attention = attention
        #  embedding các kí tự trong tgt thành vector có số chiều là (232,256)
        self.embedding = nn.Embedding(output_dim, emb_dim)
        self.rnn = nn.GRU((enc_hid_dim * 2) + emb_dim, dec_hid_dim)
        self.fc_out = nn.Linear((enc_hid_dim * 2) + dec_hid_dim + emb_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

    def forward(self, input, hidden, encoder_outputs):
        """
        inputs: batch_size (4,)
        hidden: batch_size x hid_dim (4 x 512)
        encoder_outputs: src_len x batch_size x 2*hid_dim (46 x 4 1024)
        """
        input = input.unsqueeze(0) # (1 x 4)
        embedded = self.dropout(self.embedding(input)) # (1 x 4 x 256)
        a = self.attention(hidden, encoder_outputs)  # (4 x 46)
        a = a.unsqueeze(1)  # (4 x 1 x 46)
        encoder_outputs = encoder_outputs.permute(1, 0, 2)  # (4 x 46 x 1024
        weighted = torch.bmm(a, encoder_outputs)  # batch matrix multiplication (4 x 1 x 1024)
        weighted = weighted.permute(1, 0, 2)  # (1 x 4 x 1024)

        rnn_input = torch.cat((embedded, weighted), dim=2)  # (1 x 4 x 1280)

        output, hidden = self.rnn(rnn_input, hidden.unsqueeze(0)) # (1 x 4 x 512) ,  (1 x 4 x 512 )


        assert (output == hidden).all()

        embedded = embedded.squeeze(0) # (4 x 256)
        output = output.squeeze(0) # (4 x 512)
        weighted = weighted.squeeze(0) # (4 x 1024)

        prediction = self.fc_out(torch.cat((output, weighted, embedded), dim=1)) # (4 x 232)

        return prediction, hidden.squeeze(0), a.squeeze(1)
```

3.1.5. Greedy Search - Decoder.

- Greedy decoding là một phương pháp đơn giản và phổ biến được sử dụng trong các mô hình ngôn ngữ, đặc biệt là trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP). Khi sử dụng greedy decoding, tại mỗi bước, mô hình sẽ chọn từ tiếp theo có xác suất cao nhất để tạo ra câu hoặc chuỗi đầu ra.

- Các bước :
    - Tại mỗi bước, bộ giải mã tạo ra xác suất cho tất cả các token có thể có.
    - Token có xác suất cao nhất được chọn và thêm vào chuỗi dịch hiện tại.
    - Điều kiện dừng có thể là độ dài tối đa của chuỗi hoặc token kết thúc chuỗi (eos_token) được tạo ra.

```python
def translate(src, model, max_seq_length=128, sos_token=1, eos_token=2):
    "data: Bxsrc_len"
    model.eval()
    device = src.device

    with torch.no_grad():
        src = src.transpose(1, 0)  # src_len x B
        memory = model.forward_encoder(src)

        translated_sentence = [[sos_token] * src.shape[1]]
        char_probs = [[1] * src.shape[1]]

        max_length = 0

        while max_length <= max_seq_length and not all(np.any(np.asarray(translated_sentence).T == eos_token, axis=1)):
            tgt_inp = torch.LongTensor(translated_sentence).to(device)
            output, memory = model.forward_decoder(tgt_inp, memory)
            output = F.softmax(output, dim=-1)
            output = output.to('cpu')

            values, indices = torch.topk(output, 5)

            indices = indices[:, -1, 0]
            indices = indices.tolist()

            values = values[:, -1, 0]
            values = values.tolist()
            char_probs.append(values)

            translated_sentence.append(indices)
            max_length += 1

            del output

        translated_sentence = np.asarray(translated_sentence).T

        char_probs = np.asarray(char_probs).T
        char_probs = np.multiply(char_probs, translated_sentence > 3)
        char_probs = np.sum(char_probs, axis=-1) / (char_probs > 0).sum(-1)

    return translated_sentence, char_probs
```

3.1.6. Hàm LabelSmoothingLoss.

- LabelSmoothingLoss là một phương pháp giúp cải thiện quá trình huấn luyện mô hình học sâu bằng cách giảm thiểu hiện tượng overfitting và cải thiện độ chính xác của mô hình. Đây là một kỹ thuật regularization phổ biến trong deep learning. Giá trị tham số smoothing thường được đặt ở một giá trị nhỏ hơn 1, để làm giảm sự tự tin của mô hình vào dự đoán của mình và khuyến khích nó học cách đưa ra dự đoán có tính tổng quát hơn.

```python
class LabelSmoothingLoss(nn.Module):
    def __init__(self, classes, padding_idx, smoothing=0.0, dim=-1):
        super(LabelSmoothingLoss, self).__init__()
        self.confidence = 1.0 - smoothing
        self.smoothing = smoothing
        self.cls = classes
        self.dim = dim
        self.padding_idx = padding_idx
```

```python
self.criterion = LabelSmoothingLoss(len(self.vocab), padding_idx=self.vocab.pad, smoothing=0.1)
```

- Nhóm dùng số lượng class là số kí tự (char) trong vocab và smoothing=0.1

3.1.7. Tối ưu mô hình bằng AdamW và OneCycleLR.

- AdamW là một biến thể của thuật toán Adam, trong đó W đại diện cho "Weight Decay" (sự suy giảm trọng số).

- OneCycleLR đề xuất sử dụng một chu kỳ duy nhất cho learning rate và momentum trong suốt quá trình huấn luyện. Chu kỳ này bao gồm hai giai đoạn chính: giai đoạn tăng tốc (acceleration phase) và giai đoạn giảm tốc (deceleration phase).

```
self.optimizer = AdamW(self.model.parameters(), betas=(0.9, 0.98), eps=1e-09)
# num_iters = 80000
# MAX_LR = 0.0003    # lr will inscrease from 2e-5 to MAX_LR in iter 0 -> iter NUM_ITERS * PCT_START, then decrease to 2e-5
# PCT_START = 0.1
self.scheduler = OneCycleLR(self.optimizer, total_steps=self.num_iters, pct_start=PCT_START, max_lr=MAX_LR)
```

3.2. Fine-tune BARTpho.

3.2.1. Giới thiệu.

- BARTpho được công bố bởi VinAi Research vào năm 2021 với hai phiên bản: BARTpho - syllable và BARTpho - word. BARTpho sử dụng lại kiến trúc và bộ trọng số pretrained của mô hình BART, đặc biệt phù hợp cho các tác vụ tạo sinh ngôn ngữ tự nhiên.

3.2.2. Kiến trúc.

- Cả hai mô hình BARTpho-syllable và BART-phoword đều sử dụng kiến trúc với 12 lớp mã hóa và giải mã cùng với phương pháp huấn luyện trước của BART. Cụ thể, quá trình huấn luyện trước của BART gồm hai giai đoạn chính: (i) làm lỗi văn bản đầu vào bằng một hàm noising tùy ý, và (ii) học cách tái tạo lại văn bản gốc, tức là tối ưu hóa entropy chéo giữa đầu ra của bộ giải mã và văn bản gốc.
- BART sử dụng kiến trúc chuẩn Transformer, nhưng sử dụng hàm kích hoạt GeLU [32] thay vì ReLU và khởi tạo tham số từ phân phối chuẩn N(0, 0.02). Theo BART, BARTpho áp dụng hai loại nhiễu trong hàm noising, bao gồm điền văn bản (text infilling) và hoán vị câu (sentence permutation).
- Đối với điền văn bản, tác giả lấy mẫu một số đoạn văn có độ dài được rút ra từ phân phối Poisson (λ = 3.5) và thay thế mỗi đoạn bằng một biểu tượng đặc biệt <mask>. Đối với hoán vị câu, các câu liên tiếp được nhóm lại để tạo ra các khối câu có 512 biểu tượng, và các câu trong mỗi khối sau đó được xáo trộn theo thứ tự ngẫu nhiên. Theo mô hình mBART, tác giả cũng thêm một lớp chuẩn hóa theo lớp lên đầu cả ở bộ mã hóa và giải mã.

4. Cài đặt.

4.1. Tiền xử lý dữ liệu.

- Dữ liệu sau khi đã được tổng hợp sẽ được tách thành các câu hoặc các đoạn văn ngắn, sau đó được "làm sạch", loại bỏ các ký tự lạ, chỉ lấy chữ số, chữ cái không dấu và có dấu, các dấu câu.

```
alphabet = '^[
_abcdefghijklmnopqrstuvwxyz0123456789áàảãạâấầẩẫậăắằẳẵặóòỏõọôố
ồổỗộơớờởỡợéèẻẽẹêếềểễệúùủũụưứừửữựíìỉĩịýỳỷỹỵđ!/\"\',\-\.:;?_\(
\)]+$'
data=[]

for sentence in data_processed:
    if re.match(alphabet, sentence.lower()):
        sentence = sentence.strip()
        data.append(sentence)
```

- Chia tập dữ liệu thành 2 tập train và test (tỉ lệ 1:9)
- Để các mô hình học hiệu quả hơn, dữ liệu trong tập huấn luyện sẽ được tách thành từng ngram với số lượng từ tối đa của mỗi mẫu là 5 từ ngoài ra còn tách theo dấu câu, mỗi sample sẽ dịch qua 1 từ.

```
def extract_phrases(text):
        return re.findall(r'\w[\w ]+', text)

def gen_ngrams(text, n=5):
    tokens = text.split()
    return [tokens] if len(tokens) < n else
nltk.ngrams(text.split(), n)

def processing(data):
    phrases =
itertools.chain.from_iterable(extract_phrases(text) for text
in data)
    phrases = [p.strip() for p in phrases if len(p.split()) >
1]
    list_ngrams = []
    for p in tqdm(phrases):
        list_ngrams.extend(" ".join(ngr) for ngr in
gen_ngrams(p, 5) if len(" ".join(ngr)) < 46)
    return list_ngrams
```

- Ví dụ:

```
result = processing([train_data[1]])
train_data[1],ab
```

```
100%|██████████| 10/10 [00:00<00:00, 41734.37it/s]
('Các hệ tủ có thể có hoặc không cánh, được phân chia thành các khu vực, treo quần, áo, áo dài và váy, để quần áo gấp, để đồ lót, để cặp và túi, để giầy dép, để đồ lặt vặt... Việc sắp xếp hợp lý sẽ giúp bạn dễ dàng cất và lấy đồ.',
 ['Các hệ tủ có thể',
  'hệ tủ có thể có',
  'tủ có thể có hoặc',
  'có thể có hoặc không',
  'thể có hoặc không cánh',
  'được phân chia thành các',
  'phân chia thành các khu',
  'chia thành các khu vực',
  'treo quần',
  'áo dài và váy',
  'để quần áo gấp',
  'để đồ lót',
  'để cặp và túi',
  'để giầy dép',
  'để đồ lặt vặt',
  'Việc sắp xếp hợp lý',
```

- Thống kê dữ liệu

	Train	Test
Trước khi ngram	323.994 câu/đoạn văn nhỏ	35.999 câu/đoạn văn nhỏ
Sau khi ngram	6,58 triệu ngrams	35.999 câu/đoạn văn nhỏ

*Mỗi câu/đoạn văn nhỏ trung bình 30 từ.

- Tiến hành “add noise”(tức tạo lỗi chính tả) trên cả hai tập để tạo cặp input-output theo một số rules:
    + Cặp từ hay sai: thay thế từ từ đúng thành từ sai mà người Việt Nam hay viết sai. Ví dụ: sương - xương, sĩ - sỹ, sẽ - sẻ, sữa - sửa,...
    + Cặp phụ âm hay sai: thay thế phụ âm đúng thành phụ âm sai mà người Việt Nam hay viết sai. Ví dụ: ch - tr, n - l, qu - v, r - g,…
    + Cặp từ viết kiểu teencode: về cơ bản vẫn là viết từ từ đúng thành từ sai tuy nhiên tập trung nhiều đối với các bạn trẻ vì mục đích viết nhanh hoặc thể hiện sự gần gũi. Ví dụ: mình - mik, vô - zo, biết - bit, rồi - r, tôi - t,…
    + Lỗi telex: hay xảy ra khi người viết quên bật telex hay bấm thừa kí tự thêm dấu. Ví dụ: â - aa, ở - owr, ú - us, Ô - OO, …
    + Lỗi bấm thiếu chữ: người viết đã bấm nút rồi nhưng bị trượt, bấm quá nhẹ. Ví dụ: nhưng - nhng, uống - ống, con - cn, ngủ - gủ,...
    + Lỗi bấm thiếu dấu cách: vẫn là bấm nút trượt nhưng đặc biệt trong trường hợp là nút cách. Ví dụ: Hôm nay trời thật đẹp - Hômnay trời thậtđẹp,...

Link dữ liệu sau khi đã được xử lý:

https://huggingface.co/datasets/tiendoan/ngrams_vnc_dataset

4.2. Seq2Seq.

- Class Vocab : ánh xạ giữa các từ và các chỉ số (index) tương ứng của chúng. Lớp Vocab giúp chuyển đổi các từ thành các chỉ số mà mô hình có thể xử lý, và ngược lại, giúp chuyển đổi các chỉ số đầu ra của mô hình thành các từ tương ứng.

```python
class Vocab():
    def __init__(self, chars):
        self.pad = 0
        self.sos = 1
        self.eos = 2

        self.chars = chars
        self.c2i = {c: i + 3 for i, c in enumerate(chars)}
        self.i2c = {i + 3: c for i, c in enumerate(chars)}

        self.i2c[0] = '<pad>' #  Used for padding short sentences
        self.i2c[1] = '<sos>' #  Start-of-sentence token
        self.i2c[2] = '<eos>' #  End-of-sentence token
    # char to index
    def encode(self, chars):
        return [self.sos] + [self.c2i[c] for c in chars] + [self.eos]
    # index to char
    def decode(self, ids):
        first = 1 if self.sos in ids else 0
        last = ids.index(self.eos) if self.eos in ids else None
        return ''.join([self.i2c[i] for i in ids[first:last]])

    def __len__(self):
        return len(self.c2i) + 3 # add 3 token defaul

    def batch_decode(self, arr):
        return [self.decode(ids) for ids in arr]

    def __str__(self):
        return self.chars
```

- Class Tranform_Dataset : Chuyển các kí tự trong 5-grams thành số (encoder của Vocab). Xử lý các 5-grams quá dài và trả về 5-grams ‘src’ và 5-grams ‘tgt’ thành tensor để phù hợp với pytorch

```python
class Tranform_Dataset(torch.utils.data.Dataset):
    def __init__(self, ngrams, noise_ngrams, vocab, maxlen):
        self.ngrams = ngrams
        self.noise_ngrams = noise_ngrams
        self.vocab = vocab
        self.maxlen = maxlen

    def __getitem__(self, idx):
        correct_sent = self.ngrams[idx]
        noise_sent = self.noise_ngrams[idx]

        correct_sent_idxs = self.vocab.encode(correct_sent)
        noise_sent_idxs = self.vocab.encode(noise_sent)

        src_len = len(noise_sent_idxs)
        if self.maxlen - src_len < 0:
            noise_sent_idxs = noise_sent_idxs[:self.maxlen]
            src_len = len(noise_sent_idxs)
            print("Over length in src")
        src = np.concatenate((
            noise_sent_idxs,
            np.zeros(self.maxlen - src_len, dtype=np.int32)))

        tgt_len = len(correct_sent_idxs)
        if self.maxlen - tgt_len < 0:
            correct_sent_idxs = correct_sent_idxs[:self.maxlen]
            tgt_len = len(correct_sent_idxs)
            print("Over length in target")
        tgt = np.concatenate((
            correct_sent_idxs,
            np.zeros(self.maxlen - tgt_len, dtype=np.int32)))

        return {
            'src': torch.LongTensor(src),
            'tgt': torch.LongTensor(tgt),
        }

    def __len__(self):
        return len(self.ngrams)
```

- Các parameter được định nghĩa trước :

```python
ENC_EMB_DIM = 256
DEC_EMB_DIM = 256
ENC_HID_DIM = 512
DEC_HID_DIM = 512
ENC_DROPOUT = 0.5
DEC_DROPOUT = 0.5

NUM_ITERS = 80000
BATCH_SIZE = 4
PRINT_PER_ITER = 1
VALID_PER_ITER = 2000
MAX_SAMPLE_VALID = 10000

MAX_LR = 0.0003
PCT_START = 0.1
```

- Class Seq2Seq :

```
class Seq2Seq(nn.Module):
    def __init__(self, input_dim, output_dim, encoder_embbeded, decoder_embedded, encoder_hidden, decoder_hidden,
                 encoder_dropout=0.1, decoder_dropout=0.1):
        super().__init__()
        attn = Attention(encoder_hidden, decoder_hidden)
        self.encoder = Encoder(input_dim, encoder_embbeded, encoder_hidden, decoder_hidden, encoder_dropout)
        self.decoder = Decoder(output_dim, decoder_embedded, encoder_hidden, decoder_hidden, decoder_dropout, attn)

    def forward_encoder(self, src):
        """
        src: timestep x batch_size x channel
        hidden: batch_size x hid_dim
        encoder_outputs: src_len x batch_size x hid_dim
        """
        encoder_outputs, hidden = self.encoder(src)
        return (hidden, encoder_outputs)
    def forward_decoder(self, tgt, memory):
        """
        tgt: timestep x batch_size
        hidden: batch_size x hid_dim
        encouder: src_len x batch_size x hid_dim
        output: batch_size x 1 x vocab_size
        """
        tgt = tgt[-1]
        hidden, encoder_outputs = memory
        output, hidden, _ = self.decoder(tgt, hidden, encoder_outputs)
        output = output.unsqueeze(1)
        return output, (hidden, encoder_outputs)
```

```
def forward(self, src, trg, teacher_forcing_ratio=0.5):
    """
    src:  46 x 4
    trg: 46 x 4
    outputs: batch_size x tgt_len x vocab_size (4 x 46 x 232)
    """
    batch_size = src.shape[1] # 4
    trg_len = trg.shape[0]  # trg_len = trg.shape[0] # 46
    trg_vocab_size = self.decoder.output_dim # 232
    device = src.device
    outputs = torch.zeros(trg_len, batch_size, trg_vocab_size).to(device)
    encoder_outputs, hidden = self.encoder(src)
    # first input to the decoder is the <sos> tokens
    input = trg[0, :]
    for t in range(1, trg_len):
        output, hidden, _ = self.decoder(input, hidden, encoder_outputs)
        outputs[t] = output
        top1 = output.argmax(1)
        # Teacher force
        rnd_teacher = torch.rand(1).item()
        teacher_force = rnd_teacher < teacher_forcing_ratio
        input = trg[t, :] if teacher_force else top1
    outputs = outputs.transpose(0, 1).contiguous()
    return outputs
```

```
self.model = Seq2Seq(input_dim=INPUT_DIM, output_dim=OUTPUT_DIM, encoder_embbeded=ENC_EMB_DIM,
                     decoder_embedded=DEC_EMB_DIM,
                     encoder_hidden=ENC_HID_DIM, decoder_hidden=DEC_HID_DIM, encoder_dropout=ENC_DROPOUT,
                     decoder_dropout=DEC_DROPOUT)
```

- Training Model :

```
def step(self, batch):
    self.model.train()

    batch = self.batch_to_device(batch)
    src, tgt = batch['src'], batch['tgt'] # 4x46,  4x46
    src, tgt = src.transpose(1, 0), tgt.transpose(1, 0)  # batch x src_len -> src_len x batch (4 x 46) -> (46 x 4)

    outputs = self.model(src, tgt)  # src : src_len x B, outpus : B x tgt_len x vocab

    #        loss = self.criterion(rearrange(outputs, 'b t v -> (b t) v'), rearrange(tgt_output, 'b o -> (b o)'))
    outputs = outputs.view(-1, outputs.size(2))  # flatten(0, 1)

    tgt_output = tgt.transpose(0, 1).reshape(-1)  # flatten()   # tgt: tgt_len xB , need convert to B x tgt_len

    loss = self.criterion(outputs, tgt_output)

    self.optimizer.zero_grad()

    loss.backward()

    torch.nn.utils.clip_grad_norm_(self.model.parameters(), 1)

    self.optimizer.step()
    self.scheduler.step()

    return loss.item()
```

```
def train(self):
    print("Begin training from iter: ", self.iter)
    total_loss = 0

    total_loader_time = 0
    total_gpu_time = 0
    best_cer = -1

    data_iter = iter(self.train_gen)
    for _ in range(self.num_iters):
        self.iter += 1

        start = time.time()

        try:
            batch = next(data_iter)
        except StopIteration:
            data_iter = iter(self.train_gen)
            batch = next(data_iter)

        total_loader_time += time.time() - start

        start = time.time()
        loss = self.step(batch)
        total_gpu_time += time.time() - start

        total_loss += loss
        self.train_losses.append((self.iter, loss))
```

4.2. Fine-Tune BARTpho

- Để phù hợp với bài toán sửa lỗi chính tả Tiếng Việt, nhóm em sẽ sử dụng phiên bản BARTpho-syllable
- Sử dụng model và tokenizer từ Hugging face "vinai/bartpho-syllable"
- Độ dài chuỗi tối đa là: 64

```
MODEL_NAME = "vinai/bartpho-syllable"
MAX_LENGTH = 64

model = AutoModelForSeq2SeqLM.from_pretrained(MODEL_NAME)
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
```

- Tham số:

```
args = Seq2SeqTrainingArguments(
    do_train=True,
    do_eval=True,
    output_dir=f"tiendoan/correctDoancs221",
    num_train_epochs=5,
    learning_rate=1e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    gradient_accumulation_steps=4,
    evaluation_strategy="steps",
    eval_steps=40_000,
    save_strategy="steps",
    logging_steps=40_000,
    save_total_limit=3,
    predict_with_generate=True,
    fp16=True,
    push_to_hub=True,
)
```

5. Mô phỏng.

- Bắt đầu là một batch có size là 4 chứa ‘src’ và ‘tgt’ đã được class Vocab encoder và class Tranform_Dataset chuyển thành tensor

```
batch = trainer.batch_to_device(batch)
src, tgt = batch['src'], batch['tgt']
```

```
src.shape, tgt.shape
```

```
(torch.Size([4, 46]), torch.Size([4, 46]))
```

```
src[0], tgt[0]
```

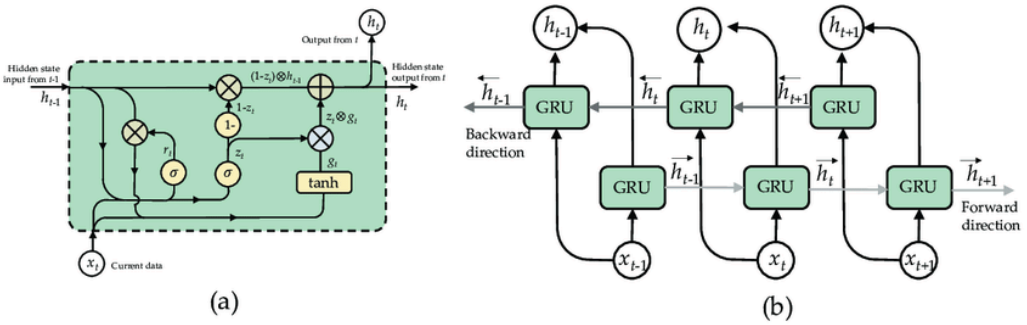
```
(tensor([  1, 143,  75,  47, 231, 143, 139,  37,  97, 231,  95,  99, 143, 231,
          41,  75,  77,  61, 145, 231, 169,   5,   2,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0], device='cuda:0'),
 tensor([  1, 143,  75,  67, 231, 143, 139,  37,  97, 231,  95, 121, 143, 231,
          41,  75,  77,  61, 145, 231, 169,   5,   2,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0], device='cuda:0'))
```

- Đảo chiều của seq_length lên trước, từ (4,46) thành (46,4).

```
src, tgt = batch['src'], batch['tgt']
src, tgt = src.transpose(1, 0), tgt.transpose(1, 0)
```

- Sau đó src được truyền vào Encoder của mô hình Seq2Seq :
  + Layer Embedding : có đầu vào là 46x4 và đầu ra là 256. Mỗi token tương ứng với một kí tự trong ‘src’ qua embedding sẽ được chuyển thành vector có kích thước 256. ‘src’ qua lớp embedding thành (46 x 4 x 256)
  + Sau đó sẽ đưa qua lớp dropout để gán ngẫu nhiên giá trị 0 cho một nửa các phần tử trong vector embedding của các kí tự để tránh overfit và giảm tài nguyên tính toán.
  + Đến 1 lớp BiGRU. Vì max_length= 46 nên sẽ có 46x2 khối GRU tương ứng cho từng kí tự và theo 2 chiều

    ```
    self.rnn = nn.GRU(input_size=256,hidden_size=512,
    bidirectional=True)
    ```



  + Mỗi x_t là 1 vector embedding của 1 kí tự và truyền theo 2 chiều thuận và nghịch. h không có mũ là output, mỗi h_t là 1 vector sau khi đã nối output đã tính toán xong của 2 khối GRU 512*2 = 1024(thuận và nghịch). h có dấu mũi tên ở trên đầu là giá trị hidden theo mỗi chiều được đưa qua các khối GRU tiếp theo để tiếp tục học.

+ Cuối cùng từ output của lớp embedding là (46x4x256), sau khi qua lớp GRU, ta sẽ được encoder_output có shape là (46x4x1024) và hidden cuối cùng (theo 2 chiều) của lớp GRU có shape là (2x4x512).
+ Ta cũng gộp hai hidden lại thành một hidden(4x1024) để lưu trữ thêm thông tin. Sau đó truyền vào lớp Linear để phi tuyến tính và để phù hợp với đầu vào Decoder-Attention
+ Vậy encoder_output (46 x 4 x 1024), hidden (4 x 512)

```
def forward(self, src):
    """
    src: src_len x batch_size (46 x 4)
    outputs: src_len x batch_size x 2*hid_dim  (bidirectional) (46 x 4 x 2*512)
    hidden: batch_size x hid_dim (4 x 512)
    """
    embedded = self.dropout(self.embedding(src))
    outputs, hidden = self.rnn(embedded)
    hidden = F.tanh(self.fc(torch.cat((hidden[-2, :, :], hidden[-1, :, :]), dim=1)))
    return outputs, hidden # (46 x 4 x 1024) , (4 x 512)
```

- Sau khi xong phần encoder, ta sẽ thực hiện phần "Attention".
  + Lấy hidden cuối cùng "repeat" thành shape (4x46x512) để gộp vào encoder_output sau khi đã đảo shape thành (4x46x1024), ta được kết quả có shape là (4x46x1536).
  + Sau đó đưa kết quả này qua lớp nn.Linear(1536,512) để giảm kích thước features thành shape mới là (4x46x512). Tiếp tục đưa qua lớp nn.Linear(512,1), ta được kết quả có shape là (4x46) (đã được squeeze). Cuối cùng đưa qua hàm kích hoạt softmax. Để tính toán mức độ quan trọng của từng kí tự trong câu.

```
def forward(self, hidden, encoder_outputs):
    """
    hidden: batch_size x hid_dim (4 x 512)
    encoder_outputs: src_len x batch_size x 2*hid_dim, (46 x 4 x 1024)
    outputs: batch_size x src_len (4 x 46)
    """
    batch_size = encoder_outputs.shape[1] # 4
    src_len = encoder_outputs.shape[0] # 46
    hidden = hidden.unsqueeze(1).repeat(1, src_len, 1) # (4 x 46 x 512)
    encoder_outputs = encoder_outputs.permute(1, 0, 2) # (4 x 46 x 1024)
    energy = F.relu(self.attn(torch.cat((hidden, encoder_outputs), dim=2))) # ( 4 x 46 x 512)
    attention = self.v(energy).squeeze(2) # 4 x 46
    return F.softmax(attention, dim=1)
```

- Tới phần chính của decoder :
  + Chạy vòng lặp 45 lần tương ứng với max_length (token đầu tiên là 1 (<sos>), bắt đầu dự đoán từ token thứ 2).Ở vòng lặp thứ nhất, ta đưa vào token 1. Đưa qua lớp embedding để tạo 1 vector embedding có kích thước 256 cho token này, ta được kết quả 'embedded' có shape là (1x4x256).
  + Ta sẽ lấy độ quan trọng của từng kí tự (4x46) được tính bằng Class Attention ở trên nhân ma trận với encoder_output ta sẽ được kết quả 'weighted' có shape là (1x4x1024) (đã được đảo shape)
  + Ta sẽ gộp 2 giá trị 'embedded' và 'weighted' để được input 'gru_input' có shape là (1x4x1280) đưa vào 1 lớp nn.GRU(1280,512) (mỗi lần lặp là 1 kí tự đưa vào 1 khối GRU).
    Ngoài ra lớp GRU này cũng nhận vào giá trị hidden đã tính ở lớp encoder.

Kết quả của lớp này ta nhận được 2 giá trị 'output', 'hidden' đều có shape là (1x4x512).

+ Gộp 3 giá trị 'embedded' ,'weighted', 'output' lại ta được kết quả có shape là (4x1792), sau đó đưa qua lớp nn.Linear(1792,232) để được prediction
+ Như vậy ta tính được prediction cho kí tự đầu tiên và hidden. Hidden sẽ được gán lại để tiếp tục làm input ở các lần lặp tiếp theo.
+ Lưu vào outputs tổng với vị trí index tương ứng để lúc sau tính loss.
+ Thay vì lấy luôn kí tự với giá trị lớn nhất trong prediction để đưa vào kí tự tiếp theo. Ta sẽ thực hiện teacher force với tỉ lệ 0.5, hoặc là lấy kí tự có xác suất lớn nhất hoặc là lấy luôn kí tự đúng trong target tại vị trí tương ứng để đưa tiếp tục vào lần lặp tiếp theo.
+ Như vậy ở mỗi lần lặp hidden sẽ được cập nhất lại, input token kí tự tương ứng có thể lấy kí tự có giá trị lớn nhất trong prediction của lần lặp trước đó hoặc lấy luôn kí tự đúng và encoder_output sẽ giữ nguyên.
+ Cuối cùng, sau khi chạy đủ 45 lần. Ta sẽ được outputs có shape là 4x46x232.

```
def forward(self, src, trg, teacher_forcing_ratio=0.5):
    """
    src:  46 x 4
    trg: 46 x 4
    outputs: batch_size x tgt_len x vocab_size (4 x 46 x 232)
    """
    batch_size = src.shape[1] # 4
    trg_len = trg.shape[0]  # 46
    trg_vocab_size = self.decoder.output_dim # 232
    device = src.device
    outputs = torch.zeros(trg_len, batch_size, trg_vocab_size).to(device)
    encoder_outputs, hidden = self.encoder(src)
    input = trg[0, :] # first input to the decoder is the <sos> tokens
    for t in range(1, trg_len):
        output, hidden, _ = self.decoder(input, hidden, encoder_outputs)
        outputs[t] = output
        # Teacher force
        rnd_teacher = torch.rand(1).item()
        teacher_force = rnd_teacher < teacher_forcing_ratio
        input = trg[t, :] if teacher_force else output.argmax(1)
    outputs = outputs.transpose(0, 1).contiguous()
    return outputs
```

- Từ outputs trên ta sẽ tính loss bằng hàm LabelSmoothingLoss. Hàm được sử dụng để tối ưu là AdamW.
- Khi dự đoán một câu mới, sẽ tách câu thành các phần dựa vào dấu phẩy (extract_phrases) và lưu vị trí của chúng để còn gộp lại. Các phrases ta sẽ gen_grams thành 5-grams.
- Các 5-grams đưa vào mô hình để dự đoán bằng greedy searching. Ta có được predict_grams
- Tái tạo lại một chuỗi văn bản từ danh sách các ngram dự đoán bằng cách đếm số lần xuất hiện của các từ tại mỗi vị trí và chọn từ xuất hiện nhiều nhất tại mỗi vị trí đó.

Kết quả cuối cùng là chuỗi văn bản được ghép lại từ các từ xuất hiện nhiều nhất tại mỗi vị trí.
- Sau đó decoder từ chỉ số sang chuỗi ta sẽ được câu đã xử lý chính tả

6. Metric đánh giá và kết quả so sánh.

6.1. Metric.

6.1.1. Char error rate (CER).

- Tính toán chỉ số CER dựa trên kỹ thuật khoảng cách Levenshtein bằng cách đếm số lượng tối thiểu các hoạt động cấp ký tự cần thiết để chuyển đổi văn bản tham chiếu đầu vào thành tệp đầu ra.
- CER được tính theo công thức sau:

$$CER = \frac{S + D + I}{N}$$

Trong đó:

S = Số lần thay thế

D = Số lần xóa

I = Số lần chèn

N = Tổng số ký tự trong văn bản tham chiếu

Mẫu số N có thể được tính theo công thức: N = S + D + C (trong đó C = Số ký tự đúng)

Kết quả của phương trình này đại diện cho tỷ lệ phần trăm ký tự trong đầu ra không chính xác so với văn bản tham chiếu đầu vào. Giá trị CER càng thấp (mô hình hoàn hảo khi CER=0), hiệu suất của mô hình càng tốt.

6.1.2. Word error rate (WER).

- Nếu CER thường được sử dụng trong việc phát hiện và trích xuất các tài liệu, chuỗi ký tự có trình tự cụ thể (ví dụ: biển số xe, số điện thoại…) thì WER thường được áp dụng khi liên quan đến việc phiên âm các đoạn văn và câu chứa các từ có nghĩa (ví dụ: các trang sách, báo).
- Công thức của WER giống với công thức của CER, nhưng thay vào đó, WER hoạt động ở cấp độ từ. Nó thể hiện số lượng từ thay thế, xóa hoặc chèn cần thiết để chuyển một câu thành câu khác.
- WER thường có mối liên quan quan mật thiết với CER (miễn là tỷ lệ lỗi không quá cao), mặc dù giá trị WER luôn luôn ghi nhận cao hơn CER.
- Công thức tính tỷ lệ lỗi từ (WER):

$$WER = \frac{S_w + D_w + I_w}{N_w}$$

Câu đúng	Câu sau khi được add noise	Câu sau khi sửa	Điểm CER của câu đã sửa và cách tính	Điểm WER của câu đã sửa và cách tính
Đây không phải là dấu hiệu tốt đẹp khi World Cup đã sắp đến ngày khai mạc.	Đây khong fai là dấu hiệu tốt đẹpkhi Worlgi Cup dã sp đến ngày khaimạc .	Đây không phải là dấu hiệu tốt đẹp khi World Cup sp đến ngày khai mạc.	S = 0, D = 0, I = 4 N = 74 CER = $\frac{S+D+I}{N}$ =5.41%	S = 1, D = 0, I = 1 N = 17 WER = $\frac{S+D+I}{N}$ =11.76%
Thông qua các hoạt động văn hóa, tôi cũng muốn mở mang kiến thức cần thiết cho nghề nghiệp.	Thôn qua á hoạt động văn óa , toi cũng muốn mở man kien thức cần thiết cho ngề nghiệp .	Thông qua các hoạt động văn óóa, tôi cũng muốn mở mang kiến thức cần thiết cho nghề nghiệp.	S = 1, D = 0, I = 0 N = 91 CER = $\frac{S+D+I}{N}$ =1.1%	S = 1, D = 0, I = 0 N = 19 WER = $\frac{S+D+I}{N}$ =5.26%
Những lúc như vậy, chị thường làm gì để vượt qua ?	Nhữn lsc nghư vậy , chịthường làm gì để vượt qua ?	Những lúc như vậy, chị thường làm gì để vượt?	S = 0, D = 0, I = 5 N = 50 CER = $\frac{S+D+I}{N}$ =10%	S = 1, D = 0, I = 2 N = 12 WER = $\frac{S+D+I}{N}$=25%
Những hệ thống đèn trần kết hợp đèn treo tường, đèn ngủ tạo hiệu quả chiếu sáng tối ưu cho căn phòng ngủ.	Nhung hthốnđèng trần kết hợp đèn treo tường , đèn ngủ tạo hieu quả chiếu xán tối ưucho cănphòng ngủ .	Những hhốn đèn kết hợp hợp treo tường, đèn ngủ tạo hiệu quả chiếu sáng tối ưu cho căn phòng ngủ.	S = 3, D = 0, I =9 N = 105 CER = $\frac{S+D+I}{N}$ =11.43%	S = 2, D = 0, I = 2 N = 23 WER = $\frac{S+D+I}{N}$ =17.39%
Sau khi được thả, hai người đã gặp Ackermann hàng	Sau khiđược thả ,hai nguoiđã gặp Ackermann hàng	Sau khi được thả,hai người đã gặp hàng năm để	S = 0, D = 0, I = 16 N = 89	S = 1, D = 0, I = 3 N = 19

năm để lên kế hoạch đột kích ngân hàng.	nam đểlên kế hoạch đột kích ngânhàng .	lên kế hoạch đột kích hàng.	CER = $\frac{S+D+I}{N}$ =17.98%	WER = $\frac{S+D+I}{N}$ =21.05%
Các ca sĩ nổi tiếng thế giới sẽ nối vòng tay lớn tại sân vận động Mỹ Đình, Hà Nội vào 26/6.	Các ca sĩnổi tiếng thếgiới xẽ nối vòng taylớn tại sân vận độngMỹ Đình , Hà Nội vào 26/6 .	Các ca sĩ nổi tiếng thế giới vòng tay lớn tại sân động Mỹ Đình, Hà Nội vào 26/6.	S = 0, D = 0, I = 11 N = 91 CER = $\frac{S+D+I}{N}$ =12.09%	S = 0, D = 0, I = 3 N = 22 WER = $\frac{S+D+I}{N}$ =13.64%

6.2. Kết quả so sánh.

- Kết quả đánh giá trên tập test của 2 phương pháp :B

Phương pháp	Char error rate(%)	Word error rate(%)
Seq2Seq - Attention	6.97	14.93
Fine-tune BARTpho	2.18	6.18