HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION RESEARCH

TOPIC: Text segmentation

LE XUAN HIEU (GR1)

hieu.lx215201@sis.hust.edu.vn

NGUYEN HOANG VIET (GR1)

viet.nh515255@sis.hust.edu.vn

NGUYEN MINH HIEU (GR2)

hieu.nm194760@sis.hust.edu.vn

VU TUAN HUNG (GR2)

hung.vt194773@sis.hust.edu.vn

Lecturer: Ph.D. Tran Viet Trung

Department: Computer Science

School: School of Information and Communications Technology

HANOI, 01/2024

TABLE OF CONTENTS

Chapter 1. Introduction

Text segmentation is a fundamental problem in natural language processing that plays a crucial role in text analysis and comprehension. This process divides written text into meaningful units (such as words, sentences, or topics) for further analysis of information and various language understanding tasks.

The goal of this project is to develop a model that can automatically identify and break online articles down into paragraphs. The project aims to achieve high accuracy in segmentation, ensure that the paragraph results are rational in Vietnamese semantics.

Traditional solutions to this problem heavily rely on defined features. The recently proposed neural models do not need manual feature engineering, but they either suffer from requiring large amounts of training data to be effective, large computational resources and long training time.

This project uses LSTM network for its capability to capture sequential information and long-term dependencies within the text. Additionally, CRF is integrated to model optimizing sequence labeling and improving prediction quality of the segmented paragraphs. This hybrid approach aims to combine the strengths of both models for improved paragraph segmentation accuracy.

This project utilizes a comprehensive dataset derived from online business articles, encompassing a diverse range of writing styles, and domains. The dataset includes articles with various paragraph structures and lengths, providing an abundant source for training and evaluating the proposed segmentation models.

Chapter 2. Methodology

In our project's methodology, we adopt a composite approach that integrates Sentence BERT, Bi-LSTM, and CRF to form a robust architecture for sentence-level tagging. This methodology is designed to comprehend and categorize sentences within larger text bodies effectively, addressing the complexities of natural language. Sentence BERT is utilized for its superior sentence embeddings, Bi-LSTM for capturing sequential dependencies, and CRF for its sequence modeling and tagging coherence. Together, these components synergize to create a powerful model capable of understanding and structuring text with high accuracy and contextual sensitivity. This strategy represents a holistic approach to sentence-level tagging, combining the strengths of individual techniques to tackle the challenges inherent in understanding natural language.

I. Model input – Sentence BERT

Sentence BERT, also known as SBERT, is a modification of pre-trained BERT network that uses siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine similarity. This innovative adaptation is particularly crucial for various NLP tasks, which require understanding the context and content of entire sentences.

In our IOB sequence tagging project, Sentence BERT serves as the foundational layer, providing robust embeddings for input sentences. Unlike traditional BERT that generates token-level embeddings, Sentence BERT processes the input text to produce embeddings at the sentence level, capturing the contextual meaning and nuances of each entire sentence. This approach is significantly advantageous for our sequence tagging model as it ensures that the semantic representation of each sentence is rich and comprehensive, leading to more accurate and meaningful tagging.

II. Bidirectional Long Short-term Memory Model

Bi-LSTM networks are a sophisticated extension of traditional recurrent neural networks (RNNs) and are specifically designed to improve the model's understanding of the context in sequence prediction problems. While commonly used for token-level tasks, Bi-LSTM's capabilities extend gracefully to sentence-level tagging, wherein the objective is to understand and categorize sentences as whole entities within larger texts or paragraphs. In our project focusing on sentence-level tagging, Bi-LSTM is employed to capture the nuances and dependencies of sentences in their context, an essential task for accurately categorizing each sentence into segments.

In sentence-level tagging, sentences are often interpreted in the context of their surrounding sentences, requiring the model to look both backwards and forwards in the sequence of sentences. The sequence of sentence embeddings is fed into the Bi-LSTM network, which learns and retains relevant features capturing a rich understanding of each sentence's context and role within the larger text. These features are then used in conjunction with a CRF layer to predict the tag for each sentence, effectively categorizing it as the beginning of a new segment or a continuation of the previous one.

III. CRF layer

In our sentence-level tagging project, the CRF layer plays a crucial role in refining and making the final decisions on how sentences are tagged, especially in distinguishing between the beginning of a new segment ('B') and continuation of an existing segment ('I').

Conditional Random Fields (CRFs) are a class of statistical modeling methods often used in pattern recognition and machine learning and have become particularly prevalent in various natural language processing tasks, including sentence-level tagging. CRFs consider the context and sequence of sentences, making globally optimal predictions that consider the constraints and dependencies of labeling decisions across sentences in a segment or document.

CRFs work on the principle of structured prediction: they model the conditional probability of a sequence of labels given a sequence of input features. These input features are typically derived from the sentence embeddings and contextual information captured by the preceding Sentence BERT and Bi-LSTM layers in our model. The CRF then uses these features to understand and learn the patterns and constraints inherent in the sequence of sentences and their tags.

Chapter 3. Implementation

I. Data preparation

1. Data collection and preprocessing

- Data is crawled from websites focusing on business content, specifically:
  - https://diendandoanhnghiep.vn/
  - https://vtc.vn/
  - https://vnexpress.net/

- Using the BeautifulSoup library, a series of GET requests are sent to these websites, extracting HTML tags containing URLs of detailed articles from each site. These URLs are then appended into a single list.

```
url_set = set()
url_content = []
for i in range(1, last_page + 1):
  url = url_1 + str(i) + url_2

  page = urllib.request.urlopen(url)
  soup = BeautifulSoup(page, 'html.parser')
  links = soup.find_all('div', class_='frame pull-left mr-15 mb-15')

  for link in links:
    sub_url= link.find('a').get('href')
    url_set.add(sub_url)
```

Figure 3.1. Collect URLs

- Subsequently, the BeautifulSoup library uses xpath() to send GET requests to the specific articles in the list. HTML tags are filtered within each source to extract paragraph tags, which are then added to a list to form complete articles.

```
file_location = "/content/drive/MyDrive/data/"
dem = 0
for link in url_set:

    page = urllib.request.urlopen(link)
    soup = BeautifulSoup(page, 'html.parser')
    dom = etree.HTML (str(soup))
    paragraphs = dom.xpath('//div[@class="post-content"]/p//text()')
    if len(paragraphs) == 0 : paragraphs = dom.xpath('//div[@class="post-content"]/p//text()')
    if len(paragraphs) == 0: continue
    if len(paragraphs[0]) == 1 : continue
    paragraph = ""
    for p in paragraphs:
      paragraph = paragraph + p
    if paragraph == " ": continue

    obj = Obj(link, paragraph)
    url_content.append(obj.toJSON())
    with open(file_location + str(dem) + ".txt", 'w', encoding='utf-8') as f:
      json.dump(obj.toJSON(), f ,ensure_ascii=False, indent = 1)
```

Figure 3.2. Extract content of articles

- Following this, the extracted paragraphs are exported to a JSON file, comprising two main sections: the content of the articles and the associated URL links. The JSON file follows a schema with two parts: "content" and "url." This structured format facilitates clear organization and subsequent analysis of the crawled data.

```
class Obj:
  def __init__(self, url, content):
    self.url = url
    self.content = content
  def toJSON(self):
        return json.dumps(self, default=lambda o: o.__dict__,
            sort_keys=True, ensure_ascii=False)
  def get_content(self):
        return self._content

  def get_url(self):
        return self._url
```

Figure 3.3. Output class

"{\"content\": \"Ông Lê Viết Hải, Chủ tịch HĐQT Tập đoàn xây dựng Hoà Bình vừa vượt qua các vòng hiệp thương, chính thức trở thành 1 trong 50 người ứng cử ĐBQH khoá XV tại TP.HCM.Trả lời VTC News, ông Lê Viết Hải cho biết, ông rất vinh dự khi được Hiệp hội Doanh nghiệp TP.HCM (HUBA) tín nhiệm và đề cử. Với sự tín nhiệm của HUBA và sự đồng thuận của UBND TP.HCM, đặc biệt là tinh thần rất cởi mở của Đảng khi tìm kiếm những người có tâm huyết đóng góp cho đất nước nên ông thấy bản thân dù không là đảng viên nhưng phải có trách nhiệm tham gia.\\\"Từ trước tới nay, tôi chưa từng nghĩ sẽ ứng cử ĐBQH hay làm chính trị mà chỉ tập trung cho việc kinh doanh. Tuy nhiên lần này, tôi nhận thấy trách nhiệm của mình trong vai trò lãnh đạo một hiệp hội và một tập đoàn dẫn đầu của ngành kinh tế khá quan trọng là xây dựng\\\", ông Hải nói.Ông Hải cho rằng, trong vai trò đại biểu Quốc hội, ông có thể chia sẻ suy nghĩ, trăn trở với những người xây dựng hành lang pháp lý, thể chế để có những thay đổi tốt hơn nữa.\\\"Việt Nam vẫn chưa phát triển tương xứng với tiềm năng quốc gia, do đó cần phải tập trung nhiều hơn nữa trí tuệ của những trí thức tâm huyết với đất nước để có nhiều sáng kiến, giải pháp bứt phá trong giai đoạn tới.Việt Nam đang bước vào thập kỷ vàng 2020 - 2030 hết sức quan trọng khi đang có cơ cấu dân số vàng với lực lượng lao động dồi dào. Chúng ta phải nỗ lực nhiều hơn vào đúng thời điểm vàng của lịch sử để vượt qua được bẫy thu nhập trung bình. Thời gian vốn là vàng nhưng trong thập kỷ vàng, thời gian còn quý hơn gấp bội\\\", ông Hải nhận định.Khi được hỏi về điều đầu tiên sẽ làm nếu trúng cử, ông Hải cho biết sẽ tập trung nghiên cứu chiến lược đưa ngành công nghiệp xây dựng Việt Nam trở thành ngành kinh tế mũi nhọn để bứt phá nền kinh tế. \\\"Không phải do tôi làm xây dựng mà tôi đưa ra đề nghị này, nhưng tôi thấy có cơ hội rất lớn để ngành xây dựng có thể đóng góp một cách hiệu quả cho sự bứt phá của nền kinh tế nếu ta xuất khẩu dịch vụ xây dựng tổng hợp (tổng thầu).Thị trường xây dựng nước ngoài có quy mô lên đến 12.000 tỷ USD, gấp khoảng 750 lần quy mô thị trường trong nước (16 tỷ USD). Nếu mình có một chiến lược tầm vóc quốc gia thì có thể hiện thực hóa mục tiêu đó\\\", Chủ tịch Tập đoàn xây dựng Hòa Bình phân tích.Là đại diện cho người dân, khối doanh nghiệp, ông Hải cho rằng cần phải thu thập nhiều ý kiến đóng góp, nguyện vọng của họ. Trong đó, điều cốt yếu là cần phải huy động trí tuệ của rất nhiều người, chỉ có vậy mới có thể hoàn thành nhiệm vụ của một đại biểu nhân dân.Hiện nay ở nước ta, tình trạng luật ra rồi mà chưa thi hành được diễn ra phổ biến. Vì vậy, phải nghiên cứu để luật đi kèm được với hướng dẫn thi hành, không phải chờ Chính phủ ra nghị định, thông tư hướng dẫn mới được áp dụng.Ngoài ra, hệ thống luật pháp cần sự sáng tạo, không rập khuôn của một quốc gia nào nhưng phải có sự kế thừa, học hỏi từ các nước phát triển. Đòi hỏi phải có sự kết hợp tinh hoa của thế giới và sự sáng tạo, huy động trí tuệ, tâm huyết của người Việt để có hệ thống luật pháp hoàn thiện và phù hợp nhất. \\\"Sắp tới nếu có ý kiến đóng góp, tôi nghĩ tại sao Quốc hội không kết hợp với Chính phủ để ra luật và ra luôn những quy định về thi hành.Tôi cho rằng chất vấn Chính phủ là một trong những nhiệm vụ quan trọng của ĐBQH, song vấn đề phải có nhiều đóng góp về giải pháp để đưa ra những quy định pháp lý logic, khoa học, chặt chẽ và minh bạch. Điều đó có được từ những kinh nghiệm thực tiễn trong doanh nghiệp và chắc chắn tôi sẽ kiến nghị để có thể đóng góp xây dựng những chính sách bám sát đời sống của khối doanh nghiệp\\\", ông Hải hứa hẹn.\", \"url\": \"https://vtc.vn//chu-tich-tap-doan-xay-dung-hoa-binh-va-viec-dau-tien-neu-trung-cu-dbqh-khoa-xv-ar602341.html\"}"

Figure 3.4. Output sample

2. Text segmentation using OpenAI API

- The **callapi** module is designed to facilitate the interaction with the OpenAI API, providing essential functions for the semantic segmentation of Vietnamese news articles. It features a primary function, **segment**, which sends a formatted prompt to the API instructing it to organize the content into well-defined sections. The output is received in a JSON format that includes both topics and sentences, maintaining the original text's integrity. A secondary function, **get_completion**, manages the API responses, ensuring consistency in output by controlling the model's randomness. This module is crucial for the preprocessing phase of our dataset, setting the stage for subsequent training and evaluation in our deep learning pipeline. This is our prompt:

> *Your task is to semantically divide a news article in Vietnamese into many sections.*
>
> *The news article is delimited by triple backticks.*
> *A cluster of sentences that support a common idea or subtopic form a section.*
>
> *The number of sections should be not too many and not too few.*
> *Make sure orginal text unchanged.*
>
> *Print list of sections in the json format. Each section item has 2 keys: topic and sentences. Sentences is a list of sentence in the section.*

```
[
    {
        "topic": "Ông Shigetaka Komori từ chức chủ tịch Fujifilm",
        "sentences": [
            "Theo Nikkei, ông Shigetaka Komori - người đã 20 năm chèo lái Fujifilm Holdings (Nhật Bản) vượt qua sự suy thoái của ngành phim ảnh và chuyển đổi
            "Sau đó, ông sẽ giữ vai trò cố vấn trưởng của công ty."
        ]
    },
    {
        "topic": "Thay thế ông Shigetaka Komori",
        "sentences": [
            "Kenji Sukeno, CEO Fujifilm sẽ thay Shigetaka Komori làm Chủ tịch.",
            "Còn ông Teiichi Goto được bổ nhiệm giữ vai trò CEO."
        ]
    },
    {
        "topic": "Lịch sử và phát triển của Fujifilm",
        "sentences": [
            "Ông Komori năm nay 81 tuổi, được bổ nhiệm làm Chủ tịch Fujifilm vào năm 2000.",
            "Đó cũng là thời điểm doanh nghiệp Nhật Bản này bị ảnh hưởng nặng nề bởi nhu cầu máy ảnh phim đi xuống khi máy ảnh kỹ thuật số ngày càng phổ biến
            "Năm 2008, Fujifilm chuyển sang thị trường dược phẩm sau khi mua lại Toyama Chemical.",
            "Đến năm 2017, công ty tiếp tục củng cố hoạt động kinh doanh y tế thông qua thương vụ mua Wako Pure Chemical Industries.",
            "Fujifilm được dự báo đạt doanh thu hợp nhất kỷ lục trong năm tài chính kết thúc vào tháng 3/2021 nhờ tăng trưởng mạnh trong lĩnh vực kinh doanh
        ]
    },
    {
        "topic": "Teiichi Goto và vai trò mới của ông",
        "sentences": [
            "Teiichi Goto, CEO sắp tới của Fujifilm, đang phụ trách mảng kinh doanh thiết bị y tế và thâu tóm mảng thiết bị chuẩn đoán hình ảnh của Hitachi."
            "Với vai trò mới, ông sẽ đẩy nhanh việc mở rộng hoạt động ở nước ngoài của Fujifilm."
        ]
    }
]
```


Figure 3.5. Json output format of OpenAI api

- The **format** module is responsible for converting the Json format returned by GPT model to the correct format suitable for our task, which is a list of sentences and their corresponding tags. The first sentence of each section is labeled as tag ‘b’, and remaining sentences are labeled as tag ‘i’. Because of our target is to divide an article into semantic sections/paragraphs, we do not need the topic of each section.


```
{
    "sentences": [
        "Theo Nikkei, ông Shigetaka Komori - người đã 20 năm chèo lái Fujifilm Holdings (Nhật Bản) vượt qua sự suy thoái của ngành phim ảnh và chuyển đổi thà
        "Sau đó, ông sẽ giữ vai trò cố vấn trưởng của công ty.",
        "Kenji Sukeno, CEO Fujifilm sẽ thay Shigetaka Komori làm Chủ tịch.",
        "Còn ông Teiichi Goto được bổ nhiệm giữ vai trò CEO.",
        "Ông Komori năm nay 81 tuổi, được bổ nhiệm làm Chủ tịch Fujifilm vào năm 2000.",
        "Đó cũng là thời điểm doanh nghiệp Nhật Bản này bị ảnh hưởng nặng nề bởi nhu cầu máy ảnh phim đi xuống khi máy ảnh kỹ thuật số ngày càng phổ biến.",
        "Năm 2008, Fujifilm chuyển sang thị trường dược phẩm sau khi mua lại Toyama Chemical.",
        "Đến năm 2017, công ty tiếp tục củng cố hoạt động kinh doanh y tế thông qua thương vụ mua Wako Pure Chemical Industries.",
        "Fujifilm được dự báo đạt doanh thu hợp nhất kỷ lục trong năm tài chính kết thúc vào tháng 3/2021 nhờ tăng trưởng mạnh trong lĩnh vực kinh doanh dược
        "Teiichi Goto, CEO sắp tới của Fujifilm, đang phụ trách mảng kinh doanh thiết bị y tế và thâu tóm mảng thiết bị chuẩn đoán hình ảnh của Hitachi.",
        "Với vai trò mới, ông sẽ đẩy nhanh việc mở rộng hoạt động ở nước ngoài của Fujifilm."
    ],
    "labels": [
        "b",
        "i",
        "b",
        "i",
        "b",
        "i",
        "i",
        "i",
        "i",
        "b",
        "i"
    ]
}
```


Figure 3.6. Correct data format for our task

- Depending on the functions of **callapi** and **format**, the **process** module is pivotal in streamlining the data processing pipeline, ensuring that the raw input is systematically converted into the desired structured format for use in our deep learning models.

3. Dataset splitting

- The **build_dataset** module in folder **dataset** is an essential component of the data preparation process in our text segmentation project. Its primary function is to partition the dataset into three subsets: training, validation, and testing. This separation is critical for the model's training and evaluation phases. In our project, we first shuffle the dataset randomly and then split it into train/val/test datasets with the ratios 0.8, 0.1, 0.1 respectively. Finally, this module writes the articles from each subset into separate files in a JSON format.

```
folder_path = '../crawl/final_data/small'
train_size = 0.8
val_size = 0.1

file_names = os.listdir(folder_path)
total_size = len(file_names)
indices = list(range(total_size))
random.shuffle(indices)

train_set = [file_names[i] for i in indices[:int(train_size * total_size)]]
val_set = [file_names[i] for i in indices[int(train_size * total_size): int((train_size + val_size) * total_size)]]
test_set = [file_names[i] for i in indices[int((train_size + val_size) * total_size):]]

assert total_size == len(train_set) + len(val_set) + len(test_set)
```

Figure 3.7. build_dataset module

II. Deep network model architecture

1. Data loader

- The **data_loader** module is an important component in our training pipeline, designed to load and prepare batches of the data model.
- At the heart of this module is the **ArticleDataset** class, which inherits from PyTorch's Dataset. It overrides the necessary methods to load data from a file and to return the length of the dataset and individual items by index. The constructor of this class invokes the load_data method, which converts sentences into embeddings and pairs them with their corresponding labels (tag ‘b’ is marked as 1, tag ‘i’ is marked as 0). In this project, instead of building own model for generating sentence embeddings, we utilizes the pretrained models from HuggingFace ecosystems to save time and resources.

```python
class ArticleDataset(Dataset):
    Le Xuan Hieu
    def __init__(self, file_path):
        self.data = self.load_data(file_path)

    Le Xuan Hieu
    def __len__(self):
        return len(self.data)

    Le Xuan Hieu
    def __getitem__(self, idx):
        return self.data[idx]

    1 usage  Le Xuan Hieu
    def load_data(self, file_path):
        data = []
        model = SentenceTransformer('bkai-foundation-models/vietnamese-bi-encoder')

        with open(file_path, 'r', encoding='utf-8') as f:
            articles = json.loads(f.read())
            for article in articles:
                labels = [1 if label == 'b' else 0 for label in article['labels']]
                embeddings = model.encode(article['sentences'])
                assert len(embeddings) == len(labels)
                data.append((embeddings, labels))
        return data
```

Figure 3.8. ArticleDataset class

- The **collate_fn** function is another crucial component. It is used by the DataLoader to combine individual data items into a batch, handling necessary padding for both embeddings and labels to ensure uniformity. It also creates a mask to identify the actual length of each sequence in the batch.

```python
def collate_fn(batch):
    embeddings, labels = zip(*batch)
    # Pad the embeddings
    padded_embeddings = pad_sequence([torch.tensor(e) for e in embeddings], batch_first=True, padding_value=0.0)
    # Pad the labels
    padded_labels = pad_sequence([torch.tensor(l) for l in labels], batch_first=True, padding_value=0)
    # Create the masks
    masks = torch.zeros_like(padded_labels, dtype=torch.bool)
    for i, length in enumerate([len(l) for l in labels]):
        masks[i, :length] = 1  # Mark the actual label positions as True
    return padded_embeddings, padded_labels, masks
```

Figure 3.9. collate_fn function

- The **get_data_loader** function is the module's interface, providing a DataLoader for the specified dataset mode—training, validation, or testing. It checks for the existence of pre-saved embeddings to avoid redundant computation. If available, it loads the dataset directly; if not, it processes and saves the dataset for future use.

```python
def get_data_loader(mode, batch_size, shuffle):
    switcher = {
        "train": ("train_set.pth", "train.txt"),
        "val": ("val_set.pth", "val.txt"),
        "test": ("test_set.pth", "test.txt")
    }
    if os.path.exists(switcher[mode][0]):
        article_dataset = torch.load(switcher[mode][0])
    else:
        article_dataset = ArticleDataset(os.path.join('dataset', switcher[mode][1]))
        torch.save(article_dataset, switcher[mode][0])
    return DataLoader(article_dataset, batch_size=batch_size, collate_fn=collate_fn, shuffle=shuffle)
```

Figure 3.10. get_data_loader function

2. Network architecture

- The **net** module introduces the **LSTM_CRF_Model**, a neural network model that combines a Long Short-Term Memory (LSTM) network with a Conditional Random Field (CRF) layer for sequence tagging tasks. This architecture is particularly suited for labeling sequences, such as text segmentation, where contextual information is crucial for accurate predictions.

  The model is defined with the following components:

  + LSTM Layer: A bidirectional LSTM that processes input embeddings. Its architecture is designed to capture dependencies in both forward and reverse directions of the input sequence.

  + Linear Layer: A fully connected layer (hidden2tag) that maps the output of the LSTM to the space of the tag set.

  + CRF Layer: The CRF layer uses the output from the LSTM to predict a sequence of tags that is globally optimized across the sequence as opposed to independent tag predictions. Instead of building the CRF layer from scratch, we utilize the pytorch-crf library.

- The model's forward method computes LSTM features that serve as inputs to the CRF layer for tag sequence prediction. The loss method computes the negative log-likelihood loss which is suitable for training with CRF, while the predict method decodes the highest scoring sequence of tags for a given input, which is used during inference.

```python
class LSTM_CRF_Model(nn.Module):
    Le Xuan Hieu
    def __init__(self, embedding_dim=768, hidden_dim=200, num_tags=2, num_layers=4, dropout=0.5):
        super(LSTM_CRF_Model, self).__init__()
        self.lstm = nn.LSTM(embedding_dim, hidden_dim // 2,
                            num_layers=num_layers, bidirectional=True, dropout=dropout)
        self.hidden2tag = nn.Linear(hidden_dim, num_tags)
        self.crf = CRF(num_tags, batch_first=True)

    Le Xuan Hieu
    def forward(self, embeds):
        # embeds: (batch_size, seq_len, embedding_dim)
        lstm_out, _ = self.lstm(embeds)
        # lstm_out: (batch_size, seq_len, hidden_dim)
        lstm_feats = self.hidden2tag(lstm_out)
        # lstm_feats: (batch_size, seq_len, num_tags)
        return lstm_feats

    2 usages (2 dynamic)  Le Xuan Hieu
    def loss(self, feats, tags, mask=None):
        # tags: (batch_size, seq_len)
        # mask: (batch_size, seq_len)
        return -self.crf(feats, tags, mask)

    1 usage (1 dynamic)  Le Xuan Hieu
    def predict(self, feats, mask=None):
        return self.crf.decode(feats, mask)
```

Figure 3.11. Network architecture

III. Training and optimization

1. Utility module

- **load_checkpoint** function is responsible for loading a saved model checkpoint from disk. It verifies the existence of the checkpoint file and then proceeds to load various components such as the model state, training epoch, and loss values. It prints the train and validation losses from the last checkpoint and returns the epoch number, allowing training to continue from that point.

```python
def load_checkpoint(filename, dir_path, model):
    file_path = os.path.join(dir_path, filename)
    if not os.path.exists(file_path):
        raise f"{file_path} does not exist"

    print("loading %s" % filename)
    checkpoint = torch.load(file_path)
    model.load_state_dict(checkpoint["state_dict"])
    epoch = checkpoint["epoch"]
    train_loss = checkpoint["train_loss"]
    val_loss = checkpoint["val_loss"]
    print(f"epoch {epoch}, train_loss={train_loss}, val_loss={val_loss}")
    return epoch
```

Figure 3.12. load_checkpoint function

- **save_checkpoint** function is used to to save the current state of the model, including its parameters, the current epoch, and the training and validation loss values. It creates the directory for saving if it does not exist and saves the checkpoint with a filename indicating the epoch number. This not only allows for the resumption of training but also facilitates the evaluation of the model's performance over time.

```python
def save_checkpoint(filename, dir_path, model, epoch, train_loss, val_loss):
    if not os.path.exists(dir_path):
        os.mkdir(dir_path)

    checkpoint = {"state_dict": model.state_dict(), "epoch": epoch, "train_loss": train_loss, "val_loss": val_loss}
    file_path = os.path.join(dir_path, filename + ".epoch%d" % epoch)
    torch.save(checkpoint, file_path)
    print("saved %s" % filename)
```

Figure 3.13. save_checkpoint function

2. Training process

- The **train_and_validate** function encapsulates the core logic of our training pipeline, integrating model training with validation, checkpoint management, and performance reporting.

  + **Checkpoint loading**: The function begins by optionally loading a model checkpoint if provided, setting the starting epoch based on the loaded state.

  + **Training loop**: For each epoch, the model is set to training mode. A loop over the training data loader computes the loss for each batch and updates the model parameters using backpropagation and the optimizer.

  + **Validation**: After each epoch, the model is evaluated using a validation set to calculate loss and performance metrics (precision, recall, and F1 score).

+ **Logging**: The function logs key statistics at each epoch, including the training loss and validation metrics, providing insight into the model's performance over time.
+ **Checkpoint saving**: At regular intervals (SAVE_EVERY epochs), the function saves a checkpoint of the model's current state, including the epoch and loss values.

```
def train_and_validate(model, train_loader, val_loader, optimizer, epochs, saved_filename, saved_dir,
                       loaded_filename=None, loaded_dir=None):
    if loaded_filename is not None and loaded_dir is not None:
        loaded_epoch = load_checkpoint(loaded_filename, loaded_dir, model)
    else:
        loaded_epoch = 0

    for epoch in trange(loaded_epoch + 1, loaded_epoch + 1 + epochs):
        model.train()
        total_loss = 0

        for embeds, tags, mask in train_loader:
            model.zero_grad()
            feats = model(embeds)
            loss = model.loss(feats, tags, mask.byte())
            total_loss += loss.item()
            loss.backward()
            optimizer.step()

        val_loss, val_precision, val_recall, val_f1 = validate_model(model, val_loader)
        train_loss = total_loss / len(train_loader)
        print(
            f"Epoch {epoch}/{loaded_epoch + epochs}, "
            f"Training Loss: {train_loss}, "
            f"Val loss: {val_loss}, "
            f"Val precision: {val_precision}, "
            f"Val recall: {val_recall}, "
            f"Val F1: {val_f1}")
        if epoch % SAVE_EVERY == 0:
            save_checkpoint(saved_filename, saved_dir, model, epoch, train_loss, val_loss)
```

Figure 3.14. Train function

IV. Testing and evaluation

- The **validate_model** function is a crucial component of our training pipeline, providing an assessment of the model's performance on a validation dataset. It is designed to evaluate the model without making any adjustments to its parameters.
- The function uses the model's predict method to generate predictions for the validation data. It collects these predictions along with the true labels, considering only the relevant parts of the sequences as indicated by the mask.

- It computes precision, recall, and F1 score using the collected predictions and true labels. These metrics provide a more nuanced understanding of the model's performance beyond simple loss calculation, considering factors like false positives and false negatives.

```python
def validate_model(model, val_loader):
    model.eval()
    total_loss = 0

    all_predictions = []
    all_true_labels = []

    with torch.no_grad():
        for embeds, tags, mask in val_loader:
            feats = model(embeds)
            loss = model.loss(feats, tags, mask.byte())  # Calculate loss
            total_loss += loss.item()
            best_paths = model.predict(feats, mask.byte())

            for i in range(len(tags)):
                for j in range(len(tags[i])):
                    if mask[i][j].item():
                        all_predictions.append(best_paths[i][j])
                        all_true_labels.append(tags[i][j])

    avg_loss = total_loss / len(val_loader)
    precision = precision_score(all_true_labels, all_predictions, average='weighted')
    recall = recall_score(all_true_labels, all_predictions, average='weighted')
    f1 = f1_score(all_true_labels, all_predictions, average='weighted')
    return avg_loss, precision, recall, f1
```

Figure 3.15. Validate function

- The **inference** function is designed to apply the trained model to new, unseen sentences, predicting their respective tags. This function is essential for deploying the model in real-world applications, where it needs to make predictions on data it hasn't encountered during training.

```python
from sentence_transformers import SentenceTransformer

def inference(sentences, model):
    sentenceEncoder = SentenceTransformer('bkai-foundation-models/vietnamese-bi-encoder')
    embeds = torch.tensor(sentenceEncoder.encode(sentences)).unsqueeze(0)
    return model.predict(model(embeds))
```

Figure 3.16. Inference function

Chapter 4. Result

The performance of our text segmentation model, even after many trials of different hyperparameters, has consistently led to overfitting. The graph provided exemplifies the outcome from one of these trials over 20 epochs, utilizing a configuration of 256 hidden dimensions, three layers, and a dropout rate of 0.3. This trend of overfitting was a common theme throughout our experiments, regardless of the variations in model hyperparameters. The training loss decreases as expected, showing the model's improved fit to the training data. In contrast, the validation loss presents a troubling pattern: it decreases only to a point before it begins to oscillate and ultimately ascend, indicating that the model's predictive performance on unseen data is deteriorating rather than improving with further training.

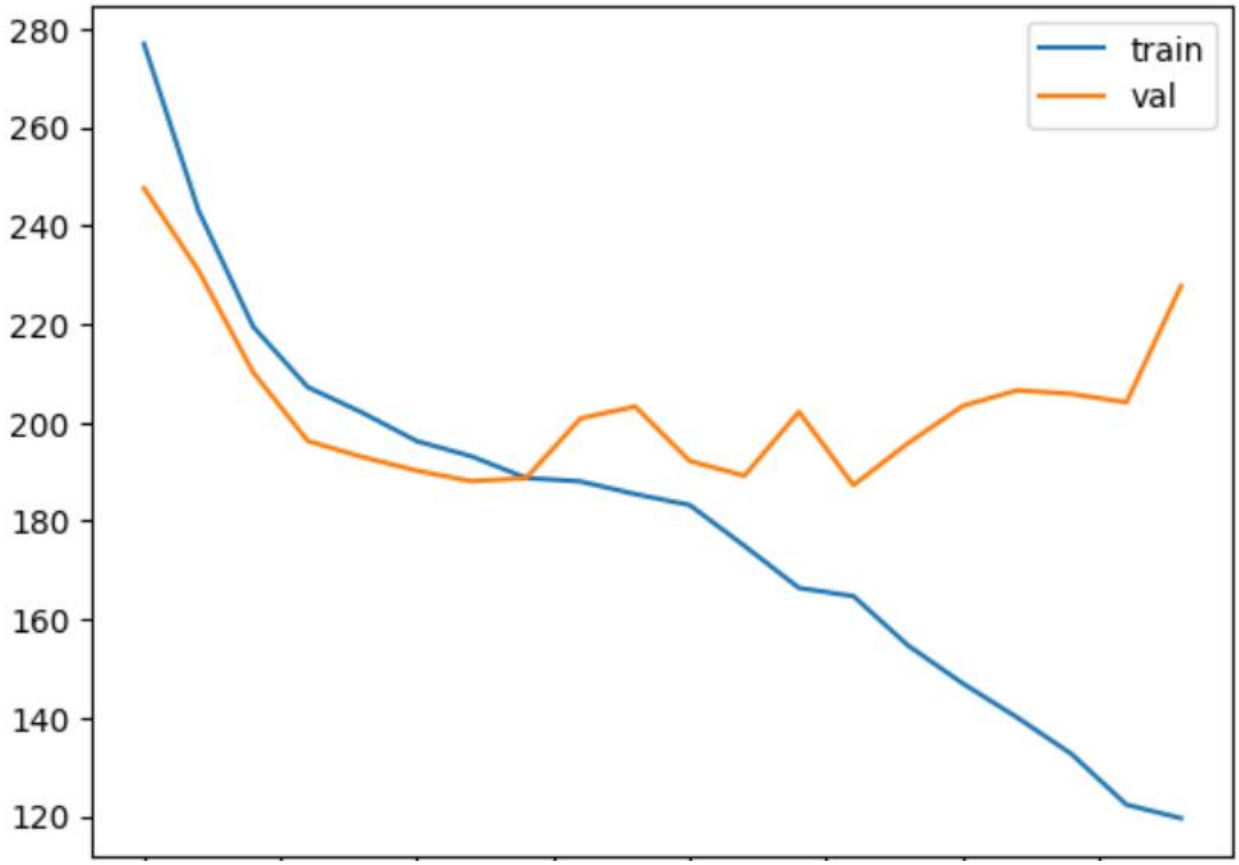

Figure 4.1. Overfitting problem

The persistent overfitting suggests that the model is becoming too specialized to the training data, unable to capture the broader patterns necessary for successful text segmentation on unseen articles. The dataset's size and quality are likely contributing factors. While 1,600 articles may seem adequate, it appears insufficient for the complexity of the task at hand, particularly if the data quality is compromised. Processing through the

OpenAI API, rather than by humans, may have introduced inconsistencies or noise that the model has inadvertently learned, further exacerbating the overfitting issue.

Our findings clearly indicate the need for a more robust dataset, both in terms of quantity and quality. Future efforts will be directed toward data augmentation, cleaning, and possibly sourcing higher-quality data. Additionally, the consistent overfitting despite various parameter adjustments points to a potential need for reevaluating our model architecture. Exploring more sophisticated architectures or regularization techniques could also be a pathway to mitigating the overfitting problem and achieving a model that performs well not just on our training data but is capable of successfully segmenting new articles.

Chapter 5. Conclusion

In conclusion, our project presents an approach to text segmentation, specifically targeting Vietnamese business news articles. We successfully built a model using SentenceBert, BiLSTM, and CRF, demonstrating our ability to leverage current research and methodologies to address text segmentation. Importantly, this project served as a valuable learning experience in navigating academic literature to identify the most appropriate approaches for our specific task.

Despite these successes, the project was not without its challenges, particularly in the realm of resources. The need to construct our own dataset due to the absence of suitable pre-existing ones led to limitations in both the size and quality of our data, which consequently impacted our model's effectiveness and its ability to generalize across different texts. This challenge was further compounded by our limited experience in selecting optimal parameters and mitigating overfitting, which are critical skills in machine learning and model fine-tuning. These hurdles underscored the necessary balance between model complexity and data sufficiency, emphasizing the need for robust datasets in training effective models.

For future work, we are committed to addressing these challenges by expanding our dataset, which will help improve the model's accuracy and robustness. We are also keen on exploring alternative approaches, particularly delving into the potential of transformers, which represent the cutting-edge of NLP technology. Applying the segmented paragraphs to summarization tasks will further demonstrate the utility of our model, opening up new avenues for research and application. As we continue our work, we aim not only to refine our model but also to contribute valuable insights and tools to the field of text segmentation, ultimately enhancing the way automated systems understand and interact with human language.

Chapter 6. Member assignments

Assignment	**Members**
Data crawling	Vu Tuan Hung (50%) Nguyen Minh Hieu (50%)
API calling	Le Xuan Hieu (50%) Nguyen Hoang Viet (50%)
Data cleaning	Nguyen Hoang Viet (100%)
Model building	Le Xuan Hieu (50%) Vu Tuan Hung (25%) Nguyen Minh Hieu (25%)
Project coding and management	Le Xuan Hieu (100%)
Report writing	Le Xuan Hieu (50%) Nguyen Minh Hieu (20%) Vu Tuan Hung (20%) Nguyen Hoang Viet (10%)