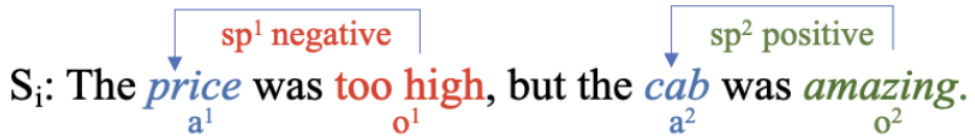
AI VIET NAM – AIO COURSE 2023

Project: Aspect-Based Sentiment Analysis

Quoc-Thai Nguyen, Duong-Thuan Nguyen, và Quang-Vinh Dinh
PR-Team: Minh-Châu Phạm, Hoàng-Nguyên Vũ và Đăng-Nhã Nguyễn

Ngày 5 tháng 2 năm 2024

Phần I. Giới thiệu



Subtask	Input	Output
Aspect Term Extraction (ATE)	S_i	a^1, a^2
Aspect Term Sentiment Classification (ATSC)	$S_i + a^1$, $S_i + a^2$	sp^1, sp^2
Aspect Sentiment Pair Extraction (ASPE)	S_i	(a^1, sp^1), (a^2, sp^2)
Aspect Oriented Opinion Extraction (AOOE)	$S_i + a^1$, $S_i + a^2$	o^1, o^2
Aspect Opinion Pair Extraction (AOPE)	S_i	(a^1, o^1), (a^2, o^2)
Aspect Opinion Sentiment Triplet Extraction (AOSTE)	S_i	(a^1, o^1, sp^1), (a^2, o^2, sp^2)

Hình 1: Phân tích cảm xúc mức khía cạnh (Aspect-Based Sentiment Analysis)

.

Phân tích cảm xúc mức khía cạnh (Aspect-Based Sentiment Analysis) là bài toán có nhiều ứng dụng hiện nay trong việc phân tích các khía cạnh khác nhau của các bình luận, đánh giá về các sản phẩm, dịch vụ,... Ví dụ mình hoạ được minh hoạ như hình 1, với câu đầu vào: "The price was too high, but the cab was amazing.". Trong câu này, có hai khía cạnh được đánh giá là: "price" và "amazing". Với khía cạnh "price" được đánh giá là "too high" nên sẽ là "negative" và với khía cạnh "positive" được đánh giá là "amazing" nên sẽ là "positive".

Dựa vào việc xác định các giá trị đầu ra của mô hình, chúng ta có thể có một số bài toán nhỏ hơn như sau:

1. Aspect Term Extraction (ATE) hoặc Aspect-Based Term Extraction: trích xuất các khía cạnh được đánh giá trong bình luận
2. Aspect Term Sentiment Classification (ATSC): dựa vào câu đầu vào và các khía cạnh được đánh giá trong bình luận để dự đoán cảm xúc của bình luận
3. Aspect Sentiment Pair Extraction (ASPE): dựa vào câu đầu vào, trính xuất ra khía cạnh và dự đoán cảm xúc của các khía cạnh
4. Aspect Oriented Opinion Extraction (AOOE): dựa vào câu đầu vào và khía cạnh, dự đoán đoạn văn bản thể hiện cảm xúc của khía cạnh
5. Aspect Opinion Pair Extraction (AOPE): dựa vào câu đầu vào trích xuất thông tin về khía cạnh và đoạn văn bản thể hiện cảm xúc của khía cạnh
6. Aspect Opinion Sentiment Triplet Extraction (AOSTE): dựa vào câu đầu vào, trích xuất các thông tin về khía cạnh, đoạn văn bản thể hiện cảm xúc của khía cạnh và cảm xúc của khía cạnh trong bình luận

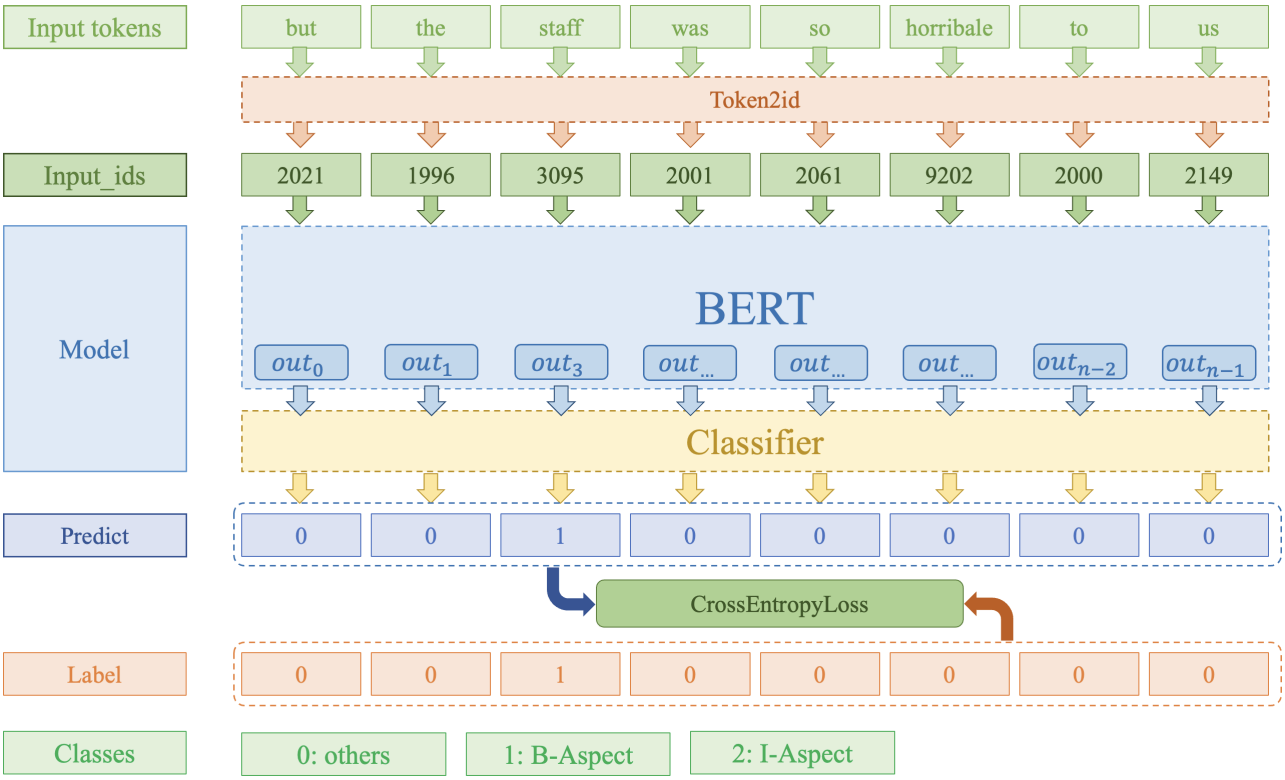
Trong phần này, chúng ta sẽ xây dựng mô hình giải quyết vấn đề cho bài toán ASPE. Dựa vào 2 bước chính:

1. Bước 1: Dự đoán từ trong văn bản thể hiện khía cạnh, hay chính là bài toán ATE
2. Bước 2: Dựa vào văn bản đầu vào và vị trí từ thể hiện khía cạnh dự đoán cảm xúc cho khía cạnh, hay là bài toán ATSC

Các thực nghiệm được xây dựng trên bộ dữ liệu SemEval-2014 Task 4: Aspect Based Sentiment Analysis và đã qua tiền xử lý cơ bản gồm: xoá bỏ dấu câu, chuẩn hoá và tách dựa vào khoảng trắng.

Phần II. Aspect Term Extraction

Ở trong phần này, chúng ta xây dựng mô hình ATE dựa vào các phương pháp giải quyết bài toán sequence classification được sử dụng trong POS Tagging, NER,...



Hình 2: Aspect term extraction pipeline.

1. Build Dataset

```
# download dataset
!gdown 1d7JABk4jViI-USjLsWmhGkvzi8uQIL5C
!unzip ./data.zip

# load dataset
import pandas as pd

train_df = pd.read_csv('./data/restaurants_train.csv')
test_df = pd.read_csv('./data/restaurants_test.csv')

# tokenization
from transformers import BertTokenizer

model_name = "bert-base-uncased"
tokenizer = BertTokenizer.from_pretrained(model_name)

# build dataset
import torch
from torch.utils.data import Dataset

class ABSADataset(Dataset):
```

```
    def __init__(self, df, tokenizer):
        self.df = df
        self.tokenizer = tokenizer

    def __getitem__(self, idx):
        tokens, tags, pols = self.df.iloc[idx, :3].values

        tokens = tokens.replace("'", "").strip("][").split(', ')
        tags = tags.strip(']['').split(', ')
        pols = pols.strip(']['').split(', ')

        bert_tokens = []
        bert_tags = []
        bert_pols = []
        for i in range(len(tokens)):
            t = self.tokenizer.tokenize(tokens[i])
            bert_tokens += t
            bert_tags += [int(tags[i])]*len(t)
            bert_pols += [int(pols[i])]*len(t)

        bert_ids = self.tokenizer.convert_tokens_to_ids(bert_tokens)

        ids_tensor = torch.tensor(bert_ids)
        tags_tensor = torch.tensor(bert_tags)
        pols_tensor = torch.tensor(bert_pols)
        return bert_tokens, ids_tensor, tags_tensor, pols_tensor

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

train_ds = ABSADataset(train_df, tokenizer)
test_ds = ABSADataset(test_df, tokenizer)

# dataloader
from torch.nn.utils.rnn import pad_sequence

def padding(samples):
    ids_tensors = [s[1] for s in samples]
    ids_tensors = pad_sequence(ids_tensors, batch_first=True)

    tags_tensors = [s[2] for s in samples]
    tags_tensors = pad_sequence(tags_tensors, batch_first=True)

    pols_tensors = [s[3] for s in samples]
    pols_tensors = pad_sequence(pols_tensors, batch_first=True)

    masks_tensors = torch.zeros(ids_tensors.shape, dtype=torch.long)
    masks_tensors = masks_tensors.masked_fill(ids_tensors != 0, 1)

    return ids_tensors, tags_tensors, pols_tensors, masks_tensors

from torch.utils.data import DataLoader

batch_size = 32
train_loader = DataLoader(
    train_ds, batch_size=batch_size, shuffle=True, collate_fn=padding
)
test_loader = DataLoader(
    test_ds, batch_size=batch_size, shuffle=True, collate_fn=padding
)
```

2. Modeling

```python
from transformers import BertModel

class ABTEBert(torch.nn.Module):
    def __init__(self, model_name):
        super(ABTEBert, self).__init__()
        self.bert = BertModel.from_pretrained(model_name)
        self.linear = torch.nn.Linear(self.bert.config.hidden_size, 3)
        self.loss_fn = torch.nn.CrossEntropyLoss()

    def forward(self, ids_tensors, masks_tensors, tags_tensors):
        bert_outputs= self.bert(
            input_ids=ids_tensors, attention_mask=masks_tensors, return_dict=False
            )
        bert_outputs = bert_outputs[0]

        linear_outputs = self.linear(bert_outputs)
        if tags_tensors is not None:
            tags_tensors = tags_tensors.view(-1)
            linear_outputs_ = linear_outputs.view(-1,3)
            loss = self.loss_fn(linear_outputs_, tags_tensors)
            return loss, linear_outputs
        else:
            return linear_outputs
```

3. Trainer

```python
import time
import numpy as np
from sklearn.metrics import classification_report

def train_epoch(model, optimizer, train_loader, device):
    losses = []
    for batch in (train_loader):
        ids_tensors, tags_tensors, _, masks_tensors = batch
        ids_tensors = ids_tensors.to(device)
        tags_tensors = tags_tensors.to(device)
        masks_tensors = masks_tensors.to(device)

        loss, _ = model(
            ids_tensors=ids_tensors,
            masks_tensors=masks_tensors,
            tags_tensors=tags_tensors
        )
        losses.append(loss.item())
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
    return sum(losses)/len(losses)

def evaluate_epoch(model, valid_loader, device):
    losses = []

    preds, labels = [], []
    with torch.no_grad():
        for batch in (valid_loader):
            ids_tensors, tags_tensors, _, masks_tensors = batch
            ids_tensors = ids_tensors.to(device)
```

```
            tags_tensors = tags_tensors.to(device)
            masks_tensors = masks_tensors.to(device)

            loss, outputs = model(
                ids_tensors=ids_tensors,
                masks_tensors=masks_tensors,
                tags_tensors=tags_tensors
            )
            losses.append(loss.item())

            _, p = torch.max(outputs, dim=2)
            preds += list([int(j) for i in p for j in i ])
            labels += list([int(j) for i in tags_tensors for j in i ])

    acc = np.mean(np.array(preds) == np.array(labels))
    return sum(losses)/len(losses), acc

def train(model, model_name, save_model, optimizer, train_loader, valid_loader,
    num_epochs, device):
    train_losses = []
    eval_accs, eval_losses = [], []
    best_loss_eval = 100
    times = []
    for epoch in range(1, num_epochs+1):
        epoch_start_time = time.time()
        # Training
        train_loss = train_epoch(model, optimizer, train_loader, device)
        train_losses.append(train_loss)

        # Evaluation
        eval_loss, eval_acc = evaluate_epoch(model, valid_loader, device)
        eval_accs.append(eval_acc)
        eval_losses.append(eval_loss)

        # Save best model
        if eval_loss < best_loss_eval:
            torch.save(model.state_dict(), save_model + f'/{model_name}.pt')

        times.append(time.time() - epoch_start_time)
        # Print loss, acc end epoch
        print("-" * 59)
        print(
            "| End of epoch {:3d} | Time: {:5.2f}s | Train Loss {:8.3f} "
            "| Valid Accuracy {:8.3f} | Valid Loss {:8.3f} ".format(
                epoch, time.time() - epoch_start_time, train_loss, eval_acc, eval_loss
            )
        )
        print("-" * 59)

    # Load best model
    model.load_state_dict(torch.load(save_model + f'/{model_name}.pt'))
    model.eval()
    metrics = {
        'train_loss': train_losses,
        'valid_accuracy': eval_accs,
        'valid_loss': eval_losses,
        'time': times
    }
    return model, metrics
```

4. Training

```python
save_model = "./model"
model = ABTEBert(model_name)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)
num_epochs = 5
best_model, metrics = train(
    model, model_name, save_model, optimizer, train_loader, test_loader, num_epochs,
    device
)
```

Kết quả training và accuracy trên tập test là 92.1

```
-------------------------------------------------------------
| End of epoch   1 | Time: 41.50s | Train Loss    0.284 | Valid Accuracy    0.908 | Valid Loss    0.233
-------------------------------------------------------------
-------------------------------------------------------------
| End of epoch   2 | Time: 36.46s | Train Loss    0.161 | Valid Accuracy    0.916 | Valid Loss    0.212
-------------------------------------------------------------
-------------------------------------------------------------
| End of epoch   3 | Time: 37.13s | Train Loss    0.100 | Valid Accuracy    0.919 | Valid Loss    0.249
-------------------------------------------------------------
-------------------------------------------------------------
| End of epoch   4 | Time: 37.27s | Train Loss    0.049 | Valid Accuracy    0.914 | Valid Loss    0.305
-------------------------------------------------------------
-------------------------------------------------------------
| End of epoch   5 | Time: 37.50s | Train Loss    0.020 | Valid Accuracy    0.921 | Valid Loss    0.338
-------------------------------------------------------------
```

Hình 3: Quá trình huấn luyện mô hình ATE.

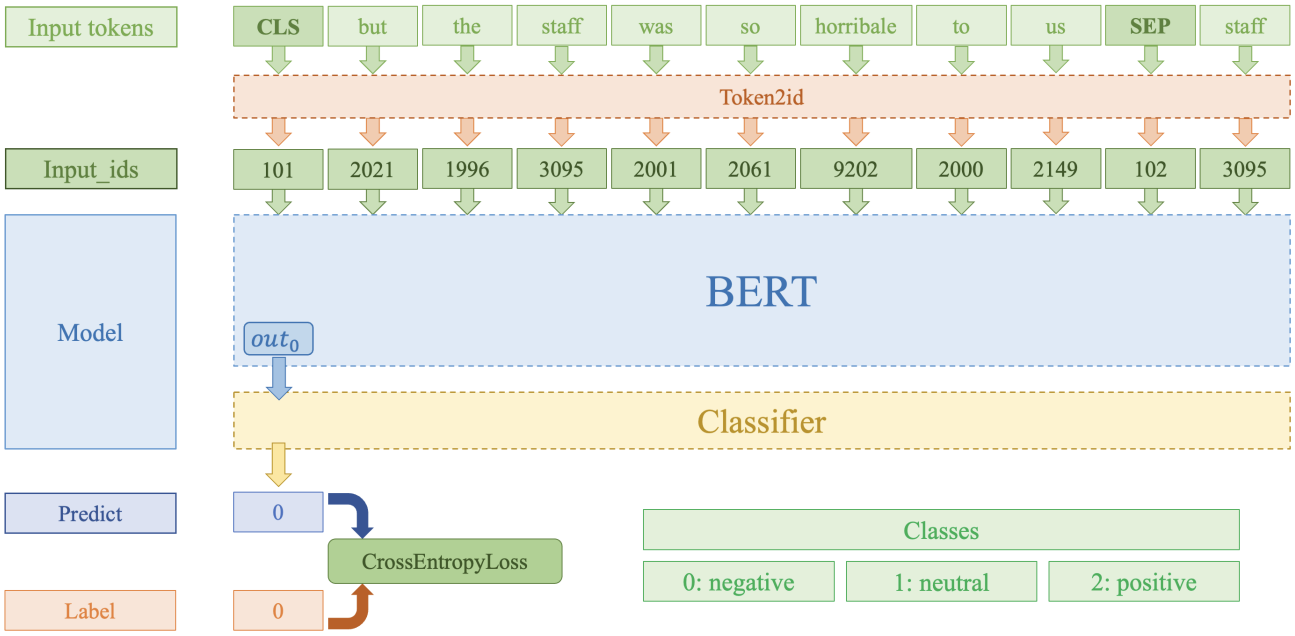
5. Prediction

Sau quá trình huấn luyện và mô hình có độ đo đánh giá tốt nhất được sử dụng để gán nhãn cho các câu đầu vào

```
def predict(best_model, sentence, device):
    word_pieces = list(tokenizer.tokenize(sentence))
    input_ids = tokenizer.convert_tokens_to_ids(word_pieces)
    input_tensor = torch.tensor([input_ids]).to(device)

    with torch.no_grad():
        outputs = model(input_tensor, None, None)
        _, predictions = torch.max(outputs, dim=2)

    predictions = predictions[0].tolist()
    return word_pieces, predictions, outputs

sentence = "the bread is top notch as well"
_, pred, _ = predict(best_model, sentence, device) # => [0, 1, 0, 0, 0, 0, 0] => bread
```

Phần III. Aspect Term Sentiment Classification

Trong phần 1, chúng ta đã tình được từ là khía cạnh trong văn bản đầu vào, ở phần này, chúng ta xây dựng mô hình dựa vào câu đầu vào và khía cạnh được dự đoán xây dựng mô hình dự đoán cảm xúc.



Hình 4: Aspect term sentiment classification pipeline.

1. Build Dataset

```
# load dataset
import pandas as pd

train_df = pd.read_csv('./data/restaurants_train.csv')
test_df = pd.read_csv('./data/restaurants_test.csv')

# tokenizer
from transformers import BertTokenizer

model_name = "bert-base-uncased"
tokenizer = BertTokenizer.from_pretrained(model_name)

# build dataset
import torch
from torch.utils.data import Dataset

class ABSADataset(Dataset):
    def __init__(self, df, tokenizer):
        self.df = df
        self.tokenizer = tokenizer

    def __getitem__(self, idx):
```

```
        tokens, tags, pols = self.df.iloc[idx, :3].values

        tokens = tokens.replace("'", "").strip("][").split(', ')
        tags = tags.strip(']['').split(', ')
        pols = pols.strip(']['').split(', ')

        bert_tokens = []
        bert_att = []
        pols_label = 0
        for i in range(len(tokens)):
            t = self.tokenizer.tokenize(tokens[i])
            bert_tokens += t
            if int(pols[i]) != -1:
                bert_att += t
                pols_label = int(pols[i])

        segment_tensor = [0] + [0]*len(bert_tokens) + [0] + [1]*len(bert_att)
        bert_tokens = ['[CLS]'] + bert_tokens + ['[SEP]'] + bert_att


        bert_ids = self.tokenizer.convert_tokens_to_ids(bert_tokens)

        ids_tensor = torch.tensor(bert_ids)
        pols_tensor = torch.tensor(pols_label)
        segment_tensor = torch.tensor(segment_tensor)

        return bert_tokens, ids_tensor, segment_tensor, pols_tensor

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

train_ds = ABSADataset(train_df, tokenizer)
test_ds = ABSADataset(test_df, tokenizer)

# dataloder
from torch.nn.utils.rnn import pad_sequence

def padding(samples):
    ids_tensors = [s[1] for s in samples]
    ids_tensors = pad_sequence(ids_tensors, batch_first=True)

    segments_tensors = [s[2] for s in samples]
    segments_tensors = pad_sequence(segments_tensors, batch_first=True)

    label_ids = torch.stack([s[3] for s in samples])

    masks_tensors = torch.zeros(ids_tensors.shape, dtype=torch.long)
    masks_tensors = masks_tensors.masked_fill(ids_tensors != 0, 1)

    return ids_tensors, segments_tensors, masks_tensors, label_ids

from torch.utils.data import DataLoader

batch_size = 32
train_loader = DataLoader(
    train_ds, batch_size=batch_size, shuffle=True, collate_fn=padding
)
test_loader = DataLoader(
    test_ds, batch_size=batch_size, shuffle=True, collate_fn=padding
)
```

2. Modeling

```python
from transformers import BertModel

class ABSABert(torch.nn.Module):
    def __init__(self, model_name):
        super(ABSABert, self).__init__()
        self.bert = BertModel.from_pretrained(model_name)
        self.linear = torch.nn.Linear(self.bert.config.hidden_size, 3)
        self.loss_fn = torch.nn.CrossEntropyLoss()

    def forward(self, ids_tensors, masks_tensors, segments_tensors, lable_tensors):
        out_dict = self.bert(
            input_ids=ids_tensors,
            attention_mask=masks_tensors,
            token_type_ids=segments_tensors
        )
        linear_outputs = self.linear(out_dict['pooler_output'])

        if lable_tensors is not None:
            loss = self.loss_fn(linear_outputs, lable_tensors)
            return loss, linear_outputs
        else:
            return linear_outputs
```

3. Trainer

```python
import time
import numpy as np

def train_epoch(model, optimizer, train_loader, device):
    losses = []
    for batch in (train_loader):
        ids_tensors, segments_tensors, masks_tensors, label_ids = batch
        ids_tensors = ids_tensors.to(device)
        segments_tensors = segments_tensors.to(device)
        label_ids = label_ids.to(device)
        masks_tensors = masks_tensors.to(device)

        loss, _ = model(
            ids_tensors=ids_tensors,
            masks_tensors=masks_tensors,
            segments_tensors=segments_tensors,
            lable_tensors=label_ids
        )
        losses.append(loss.item())
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
    return sum(losses)/len(losses)

def evaluate_epoch(model, valid_loader, device):
    losses = []

    preds, labels = [], []
    with torch.no_grad():
        for batch in (valid_loader):
            ids_tensors, segments_tensors, masks_tensors, label_ids = batch
            ids_tensors = ids_tensors.to(device)
            segments_tensors = segments_tensors.to(device)
            masks_tensors = masks_tensors.to(device)
            label_ids = label_ids.to(device)

            loss, outputs = model(
                ids_tensors=ids_tensors,
                masks_tensors=masks_tensors,
                segments_tensors=segments_tensors,
                lable_tensors=label_ids
            )
            losses.append(loss.item())

            _, p = torch.max(outputs, dim=1)
            preds += list([int(i) for i in p])
            labels += list([int(i) for i in label_ids])

    acc = np.mean(np.array(preds) == np.array(labels))
    return sum(losses)/len(losses), acc

def train(model, model_name, save_model, optimizer, train_loader, valid_loader,
    num_epochs, device):
    train_losses = []
    eval_accs, eval_losses = [], []
    best_loss_eval = 100
    times = []
    for epoch in range(1, num_epochs+1):
```

```
        epoch_start_time = time.time()
        # Training
        train_loss = train_epoch(model, optimizer, train_loader, device)
        train_losses.append(train_loss)

        # Evaluation
        eval_loss, eval_acc = evaluate_epoch(model, valid_loader, device)
        eval_accs.append(eval_acc)
        eval_losses.append(eval_loss)

        # Save best model
        if eval_loss < best_loss_eval:
            torch.save(model.state_dict(), save_model + f'/{model_name}.pt')

        times.append(time.time() - epoch_start_time)
        # Print loss, acc end epoch
        print("-" * 59)
        print(
            "| End of epoch {:3d} | Time: {:5.2f}s | Train Loss {:8.3f} "
            "| Valid Accuracy {:8.3f} | Valid Loss {:8.3f} ".format(
                epoch, time.time() - epoch_start_time, train_loss, eval_acc, eval_loss
            )
        )
        print("-" * 59)

    # Load best model
    model.load_state_dict(torch.load(save_model + f'/{model_name}.pt'))
    model.eval()
    metrics = {
        'train_loss': train_losses,
        'valid_accuracy': eval_accs,
        'valid_loss': eval_losses,
        'time': times
    }
    return model, metrics
```

4. Training

```
save_model = "./model"
model = ABSABert(model_name)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)
num_epochs = 5
best_model, metrics = train(
    model, model_name, save_model, optimizer, train_loader, test_loader, num_epochs,
    device
)
```

Kết quả training và accuracy trên tập test là 81.4

```
-----------------------------------------------------------
| End of epoch   1 | Time: 46.77s | Train Loss    0.829 | Valid Accuracy    0.748 | Valid Loss    0.644
-----------------------------------------------------------
-----------------------------------------------------------
| End of epoch   2 | Time: 39.83s | Train Loss    0.539 | Valid Accuracy    0.801 | Valid Loss    0.507
-----------------------------------------------------------
-----------------------------------------------------------
| End of epoch   3 | Time: 39.13s | Train Loss    0.353 | Valid Accuracy    0.814 | Valid Loss    0.469
-----------------------------------------------------------
-----------------------------------------------------------
| End of epoch   4 | Time: 40.16s | Train Loss    0.218 | Valid Accuracy    0.777 | Valid Loss    0.623
-----------------------------------------------------------
-----------------------------------------------------------
| End of epoch   5 | Time: 39.52s | Train Loss    0.155 | Valid Accuracy    0.805 | Valid Loss    0.587
-----------------------------------------------------------
```

Hình 5: Quá trình huấn luyện ATSC.

5. Prediction

```python
def predict(model, tokenizer, sentence, aspect, device):
    t1 = tokenizer.tokenize(sentence)
    t2 = tokenizer.tokenize(aspect)

    word_pieces = ['[CLS]'] + t1 + ['[SEP]'] + t2

    segment_tensor = [0] + [0]*len(t1) + [0] + [1]*len(t2)

    input_ids = tokenizer.convert_tokens_to_ids(word_pieces)
    input_tensor = torch.tensor([input_ids]).to(device)
    segment_tensor = torch.tensor([segment_tensor]).to(device)

    with torch.no_grad():
        outputs = model(input_tensor, None, segment_tensor, None)
        _, predictions = torch.max(outputs, dim=1)

    return word_pieces, int(predictions), outputs

sentence = "The bread is top notch as well"
aspect = "bread"
predict(best_model, tokenizer, sentence, aspect, device) # => 2
```

Phần 4. Câu hỏi trắc nghiệm

Cho câu bình luận như sau: "The food is very fresh."

Câu hỏi 1 Dựa vào câu bình luận trên xác định giá trị dự đoán mô hình giải quyết bài toán ATE?
a) food
b) food, positive
c) food, very, fresh
d) food, very fresh, positive

Câu hỏi 2 Dựa vào câu bình luận trên xác định giá trị dự đoán mô hình giải quyết bài toán ASPE?
a) food
b) food, positive
c) food, very, fresh
d) food, very fresh, positive

Câu hỏi 3 Dựa vào câu bình luận trên xác định giá trị dự đoán mô hình giải quyết bài toán AOPE?
a) food
b) food, positive
c) food, very, fresh
d) food, very fresh, positive

Câu hỏi 4 Dựa vào câu bình luận trên xác định giá trị dự đoán mô hình giải quyết bài toán AOSTE?
a) food
b) food, positive
c) food, very, fresh
d) food, very fresh, positive

Câu hỏi 5 Mô hình giải quyết bài toán ATE ở phần 2 trên sử dụng bộ dữ liệu nào?
a) IMDB-Review
b) Penn Tree Bank
c) SemEval-2014 Task 4
d) SQuAD

Câu hỏi 6 Số lượng samples được sử dụng cho tập train trong phần thực nghiệm là
a) 3600
b) 3601
c) 3602
d) 3603

Câu hỏi 7 Mô hình giải quyết bài toán ATE ở phần 2 trên dựa vào bài toán tổng quát nào sau đây?
a) Sequence Labeling
b) Machine Translation

c) Topic Modeling
d) Sumarization

Câu hỏi 8 Mô hình tiền huấn luyện sử dụng cho bài toán ATE ở phần 2 trên là?
a) BERT
b) RoBERTA
c) DEBERTA
d) DistilBERT

Câu hỏi 9 Mô hình tiền huấn luyện sử dụng cho bài toán ASTC ở phần 3 trên là?
a) BERT
b) RoBERTA
c) DEBERTA
d) DistilBERT

Câu hỏi 10 Hàm mục tiêu để giải quyết bài toán ASTC ở phần 3 là
a) Masked Language Model
b) Next Sentence Prediction
c) Cả 2 đáp án trên đều đúng
d) Cả 2 đáp án trên đều sai

- Hết -