Giải thích Code PhoBERT Sentiment Analysis

🎯 Tổng quan

Code này thực hiện **phân loại cảm xúc** (sentiment analysis) cho văn bản tiếng Việt sử dụng mô hình PhoBERT. Mô hình sẽ phân loại comment thành 3 nhãn: **POS** (tích cực), **NEG** (tiêu cực), **NEU** (trung tính).

📚 Import Libraries

python

```python
import os
import torch
import torch.nn as nn
import pandas as pd
from torch.utils.data import Dataset, DataLoader
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from sklearn.metrics import accuracy_score
```

Giải thích:

- `torch`: Framework deep learning chính
- `pandas`: Xử lý dữ liệu dạng bảng (CSV)
- `transformers`: Thư viện Hugging Face để sử dụng BERT
- `sklearn.metrics`: Tính toán độ chính xác

⚡ **Có thể tái sử dụng:** Các import này chuẩn cho hầu hết bài toán NLP với PyTorch

---

⚙️ Cấu hình tham số

python

```python
DATA_PATH = "data/processed/augmented_test_2k.csv"
MODEL_SAVE_PATH = "models/phobert_simple.pt"
MAX_LEN = 128
BATCH_SIZE = 16
EPOCHS = 3
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Giải thích chi tiết:

- `DATA_PATH`: Đường dẫn file CSV chứa dữ liệu train
- `MODEL_SAVE_PATH`: Nơi lưu mô hình sau khi train
- `MAX_LEN = 128`: Độ dài tối đa của câu (token). Câu dài hơn sẽ bị cắt, ngắn hơn sẽ được padding
- `BATCH_SIZE = 16`: Số mẫu xử lý cùng lúc (nhỏ để tiết kiệm RAM)
- `EPOCHS = 3`: Số lần duyệt qua toàn bộ dataset
- `DEVICE`: Tự động chọn GPU nếu có, không thì dùng CPU

💡 **Ví dụ:**

- Câu "Tôi thích sản phẩm này" → sau tokenize có thể thành [101, 1234, 5678, 9012, 102] (5 tokens)
- Nếu MAX_LEN=128 → padding thêm 123 token [PAD] để đủ 128 tokens

⚡ **Có thể tái sử dụng:** Template cấu hình này cho mọi bài toán classification

---

🗂️ Custom Dataset Class

python

```python
class SimpleDataset(Dataset):
    def __init__(self, dataframe, tokenizer, max_len):
        self.sentences = dataframe["comment"].values
        self.labels = dataframe["label"].map({"POS": 0, "NEG": 1, "NEU": 2}).fillna(0).values
        self.tokenizer = tokenizer
        self.max_len = max_len
```

Giải thích:

- Kế thừa từ `torch.utils.data.Dataset`
- `self.sentences`: Lấy cột "comment" từ DataFrame
- `self.labels`: Chuyển đổi nhãn text thành số:
  - "POS" → 0 (tích cực)
  - "NEG" → 1 (tiêu cực)
  - "NEU" → 2 (trung tính)
- `fillna(0)`: Nếu có nhãn trống, gán là 0 (POS)

python

```python
def __getitem__(self, idx):
    sentence = self.sentences[idx]
    label = torch.tensor(int(self.labels[idx]), dtype=torch.long)

    encoding = self.tokenizer.encode_plus(
        sentence,
        max_length=self.max_len,
        padding="max_length",
        truncation=True,
        return_tensors="pt"
    )

    return {
        "input_ids": encoding["input_ids"].squeeze(),
        "attention_mask": encoding["attention_mask"].squeeze(),
        "label": label
    }
```

Giải thích từng bước:

1. **Lấy dữ liệu:** `sentence = self.sentences[idx]`
   - Ví dụ: idx=0 → sentence = "Sản phẩm này rất tốt!"
2. **Chuyển label thành tensor:** `label = torch.tensor(int(self.labels[idx]), dtype=torch.long)`
   - Ví dụ: "POS" → tensor(0)
3. **Tokenize câu:** `tokenizer.encode_plus()`
   - `sentence`: "Sản phẩm này rất tốt!"
   - `max_length=128`: Giới hạn 128 tokens
   - `padding="max_length"`: Thêm [PAD] để đủ 128 tokens
   - `truncation=True`: Cắt bớt nếu > 128 tokens
   - `return_tensors="pt"`: Trả về PyTorch tensors
4. **Kết quả trả về:**

   python

   ```python
   {
       "input_ids": tensor([101, 1234, 5678, ..., 0, 0, 0]),      # Token IDs
       "attention_mask": tensor([1, 1, 1, ..., 0, 0, 0]),        # 1=real token, 0=padding
       "label": tensor(0)                                          # POS = 0
   }
   ```

⚡ **Có thể tái sử dụng:** Class này cho mọi bài toán text classification, chỉ cần đổi cột dữ liệu và mapping labels

---

📊 Load và kiểm tra dữ liệu

python

```
df = pd.read_csv(DATA_PATH, on_bad_lines='skip')
print(f"📊 Dữ liệu có {df.shape[0]} dòng, Nhãn: {df['label'].unique()}")
```

Giải thích:

- `on_bad_lines='skip'`: Bỏ qua các dòng lỗi format
- `df.shape[0]`: Số dòng dữ liệu
- `df['label'].unique()`: Các nhãn duy nhất

💡 **Ví dụ output:**

```
📊 Dữ liệu có 2000 dòng, Nhãn: ['POS' 'NEG' 'NEU']
```

🤖 Load PhoBERT Model

python

```
tokenizer = AutoTokenizer.from_pretrained("vinai/phobert-base")
model = AutoModelForSequenceClassification.from_pretrained("vinai/phobert-base", num_labels=3)
model = model.to(DEVICE)
```

Giải thích:

- `AutoTokenizer`: Tự động load tokenizer phù hợp với PhoBERT
- `AutoModelForSequenceClassification`: Load mô hình BERT cho classification
- `num_labels=3`: 3 nhãn output (POS, NEG, NEU)
- `model.to(DEVICE)`: Chuyển model lên GPU nếu có

💡 **Quá trình:**

1. Download pre-trained PhoBERT (lần đầu)
2. Thêm classification head với 3 neurons output
3. Chuyển toàn bộ lên GPU

🔄 DataLoader

python

```
dataset = SimpleDataset(df, tokenizer, max_len=MAX_LEN)
dataloader = DataLoader(dataset, batch_size=BATCH_SIZE, shuffle=True)
```

Giải thích:

- Tạo dataset object từ DataFrame
- DataLoader chia data thành các batch nhỏ
- `shuffle=True`: Trộn ngẫu nhiên data mỗi epoch

💡 **Ví dụ:**

- 2000 samples, batch_size=16 → 125 batches
- Mỗi batch có 16 samples (trừ batch cuối có thể ít hơn)

🏋️ Training Loop

python

```
optimizer = torch.optim.Adam(model.parameters(), lr=2e-5)
loss_fn = nn.CrossEntropyLoss()
```

Chuẩn bị:

- `Adam optimizer`: Thuật toán tối ưu với learning rate 2e-5 (rất nhỏ cho BERT)
- `CrossEntropyLoss`: Loss function cho multi-class classification

python

```python
model.train()
for epoch in range(EPOCHS):
    total_loss = 0
    for batch in dataloader:
        input_ids = batch["input_ids"].to(DEVICE)
        attention_mask = batch["attention_mask"].to(DEVICE)
        labels = batch["label"].to(DEVICE)

        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask)
        loss = loss_fn(outputs.logits, labels)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
```

Giải thích từng bước:

1. **Set training mode:** `model.train()`
   - Bật dropout, batch normalization
2. **Mỗi batch:**
   - Chuyển data lên GPU: `.to(DEVICE)`
   - Reset gradients: `optimizer.zero_grad()`
   - Forward pass: `outputs = model(...)`
   - Tính loss: `loss = loss_fn(outputs.logits, labels)`
   - Backward pass: `loss.backward()`
   - Update weights: `optimizer.step()`

💡 **Ví dụ một batch:**

```
Input: 16 câu comment
→ Forward pass → Predictions: [[0.1, 0.7, 0.2], [0.8, 0.1, 0.1], ...]
→ Loss calculation với true labels: [1, 0, ...]
→ Backpropagation → Update weights
```

⚡ **Có thể tái sử dụng:** Template training loop này cho mọi bài toán classification

---

📏 Evaluation

python

```python
model.eval()
all_preds, all_labels = [], []

with torch.no_grad():
    for batch in dataloader:
        # ... forward pass ...
        preds = torch.argmax(outputs.logits, dim=1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

accuracy = accuracy_score(all_labels, all_preds)
```

Giải thích:

- `model.eval()`: Tắt dropout, batch norm cho evaluation
- `torch.no_grad()`: Không tính gradients (tiết kiệm memory)
- `torch.argmax()`: Lấy class có xác suất cao nhất
- `accuracy_score()`: Tính % dự đoán đúng

💡 **Ví dụ:**

```
Predictions: [[0.1, 0.8, 0.1], [0.9, 0.05, 0.05]]
→ argmax → [1, 0] (NEG, POS)
True labels: [1, 0]
→ Accuracy = 2/2 = 100%
```

💾 Lưu model

python

```python
torch.save(model.state_dict(), MODEL_SAVE_PATH)
```

Giải thích:

- Chỉ lưu weights của model (không lưu architecture)
- Load lại: `model.load_state_dict(torch.load(MODEL_SAVE_PATH))`

🔧 Những điểm có thể tái sử dụng

✅ Template hoàn toàn tái sử dụng:

1. **Cấu hình tham số** - Chỉ cần đổi paths và hyperparameters
2. **Dataset class** - Đổi tên cột và label mapping
3. **Training loop** - Standard cho mọi bài toán classification
4. **Evaluation code** - Chuẩn cho multi-class classification

🔄 Cần customize theo bài toán:

1. **Model selection** - Đổi từ PhoBERT sang BERT khác
2. **Label mapping** - Theo số lượng classes cụ thể
3. **Preprocessing** - Tùy theo format dữ liệu
4. **Metrics** - Có thể thêm precision, recall, F1-score

💡 Cải tiến có thể áp dụng:

1. **Train/Validation split** - Tách data để đánh giá tốt hơn
2. **Early stopping** - Dừng sớm khi overfitting
3. **Learning rate scheduling** - Giảm LR theo epoch
4. **Data augmentation** - Tăng cường dữ liệu
5. **Ensemble methods** - Kết hợp nhiều models

🚀 Cách chạy code:

1. **Chuẩn bị dữ liệu:** File CSV với cột `comment` và `label`
2. **Cài đặt thư viện:** `pip install torch transformers pandas scikit-learn`
3. **Chạy:** `python train_phobert.py`
4. **Kết quả:** Model được lưu tại `models/phobert_simple.pt`