Phân tích Code Phân loại Cảm xúc với PhoBERT

Tổng quan về Luồng xử lý

Đây là một hệ thống phân loại cảm xúc (sentiment analysis) sử dụng mô hình PhoBERT - một biến thể của BERT được tối ưu hóa cho tiếng Việt. Hệ thống có thể phân loại văn bản thành 3 loại cảm xúc: **Tích cực (POS)**, **Tiêu cực (NEG)**, và **Trung tính (NEU)**.

Chi tiết các thành phần

1. Import và Dependencies

python

```python
import torch
import torch.nn as nn
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import pandas as pd
import os
from utils import load_config, load_model
```

Công nghệ sử dụng:

- **PyTorch**: Framework deep learning chính
- **Transformers (Hugging Face)**: Thư viện để sử dụng các mô hình pre-trained
- **PhoBERT**: Mô hình BERT được train trên corpus tiếng Việt

2. Kiến trúc Mô hình - PhoBERTClassifier

python

```python
class PhoBERTClassifier(nn.Module):
    def __init__(self, phobert_model):
        super().__init__()
        self.phobert = phobert_model
        self.dropout = nn.Dropout(0.1)

    def forward(self, input_ids, attention_mask):
        outputs = self.phobert(input_ids, attention_mask=attention_mask)
        return outputs.logits
```

Luồng xử lý trong mô hình:

1. **Input**: Nhận `input_ids` (token đã encode) và `attention_mask`
2. **PhoBERT Processing**: Xử lý qua các transformer layers
3. **Output**: Trả về logits (điểm số chưa normalize) cho 3 classes

Đặc điểm kỹ thuật:

- Sử dụng dropout 0.1 để tránh overfitting
- Tận dụng classifier có sẵn của PhoBERT thay vì tự xây dựng

3. Hàm Dự đoán (Inference)

python

```python
def predict(model, tokenizer, text, device, max_len=128):
    model.eval()
    inputs = tokenizer.encode_plus(
        text, max_length=max_len, padding="max_length",
        truncation=True, return_tensors="pt"
    )
    input_ids = inputs["input_ids"].to(device)
    attention_mask = inputs["attention_mask"].to(device)

    with torch.no_grad():
        outputs = model(input_ids, attention_mask)
        pred = torch.argmax(outputs, dim=1).cpu().item()

    label_map = {0: "POS", 1: "NEG", 2: "NEU"}
    return label_map.get(pred, "unknown")
```

Luồng xử lý dự đoán:

1. **Preprocessing**:
   - Tokenize văn bản input
   - Padding/truncation về max_length=128
   - Chuyển đổi thành tensor
2. **Model Inference**:
   - Disable gradient calculation (`no_grad()`)
   - Forward pass qua mô hình
   - Áp dụng argmax để lấy class có điểm cao nhất
3. **Postprocessing**:
   - Map từ class index về label string
   - Trả về kết quả dự đoán

4. Luồng chính (Main Execution)

4.1 Khởi tạo và Load Model

python

```python
config = load_config()
project_root = config["project_root"]
model_path = os.path.join(project_root, "models", "phobert_best.pt")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

tokenizer = AutoTokenizer.from_pretrained("vinai/phobert-base")
model = PhoBERTClassifier(
    AutoModelForSequenceClassification.from_pretrained(
        "vinai/phobert-base", num_labels=3
    )
)
model.load_state_dict(load_model(model_path))
model = model.to(device)
```

Các bước quan trọng:

- Load cấu hình từ file config
- Detect và sử dụng GPU nếu có
- Load tokenizer và model từ Hugging Face Hub
- Load weights đã train từ file `.pt`

4.2 Test với văn bản mẫu

python

```python
test_text = "Câu này có tự nhiên không?"
result = predict(model, tokenizer, test_text, device)
print(f"📝 Văn bản: {test_text}")
print(f"🔍 Dự đoán: {result}")
```

4.3 Batch Processing từ CSV

python

```python
if os.path.exists(input_path):
    data = pd.read_csv(input_path, usecols=["comment"])
    predictions = [predict(model, tokenizer, text, device) for text in data["comment"]]
    data["prediction"] = predictions
    output_path = os.path.join(project_root, "data", "processed", "predictions_with_tda.csv")
    data.to_csv(output_path, index=False)
```

Luồng batch processing:

1. Đọc file CSV chứa comments
2. Dự đoán từng comment một cách tuần tự
3. Thêm cột prediction vào DataFrame
4. Xuất kết quả ra file CSV mới

Kiến trúc Tổng thể

```
Input Text → Tokenizer → PhoBERT Encoder → Classification Head → Logits → ArgMax → Label
     ↓           ↓              ↓                ↓              ↓        ↓       ↓
"Tôi yêu AI"  [101,1234,...]  Hidden States   [0.1,0.8,0.1]   [0,1,2]    1     "NEG"
```

Đặc điểm Kỹ thuật

Ưu điểm:

- **Hiệu quả cho tiếng Việt**: PhoBERT được train specifically cho tiếng Việt
- **Pre-trained**: Tận dụng knowledge từ large corpus
- **3-class classification**: Phân loại chi tiết hơn binary classification
- **Batch processing**: Hỗ trợ xử lý hàng loạt từ file CSV

Hạn chế:

- **Sequential processing**: Không tận dụng được batch processing cho inference
- **Fixed max_length**: Giới hạn 128 tokens có thể không đủ cho văn bản dài
- **Memory usage**: Model size khá lớn (110M parameters)

Cải tiến có thể áp dụng

1. **Batch Inference**: Xử lý multiple samples cùng lúc
2. **Dynamic batching**: Nhóm các samples có độ dài tương tự
3. **Model optimization**: Quantization, pruning để giảm model size
4. **Async processing**: Xử lý bất đồng bộ cho throughput cao hơn

Module Utils - Các Hàm Tiện ích

1. load_config()

python

```python
def load_config():
    config_path = os.path.join(os.path.dirname(__file__), "..", "config.json")
    default_config = {"project_root": os.path.abspath(os.path.join(os.path.dirname(__file__), '
    try:
        with open(config_path, "r", encoding="utf-8") as f:
            return {**default_config, **json.load(f)}
    except FileNotFoundError:
        print(f"⚠️ Không tìm thấy config.json, sử dụng giá trị mặc định: {default_config}")
        return default_config
```

Chức năng:

- Load cấu hình từ file `config.json` ở thư mục parent
- Fallback về cấu hình mặc định nếu không tìm thấy file
- Merge config từ file với default config
- **Pattern**: Configuration management với graceful fallback

2. save_model() & load_model()

python

```python
def save_model(model, path):
    torch.save(model.state_dict(), path)

def load_model(path):
    return torch.load(path)
```

Chức năng:

- **save_model**: Lưu state_dict của PyTorch model (chỉ weights, không lưu architecture)
- **load_model**: Load state_dict từ file
- **Best practice**: Chỉ lưu weights thay vì toàn bộ model object

3. ensure_dir()

python

```python
def ensure_dir(directory):
    os.makedirs(directory, exist_ok=True)
```

Chức năng:

- Tạo thư mục nếu chưa tồn tại
- `exist_ok=True`: Không raise error nếu thư mục đã tồn tại
- **Use case**: Đảm bảo thư mục output exists trước khi save file

4. save_log()

python

```python
def save_log(message, log_path=None):
    if log_path is None:
        config = load_config()
        log_path = os.path.join(config["project_root"], "logs", "training_log.txt")
    ensure_dir(os.path.dirname(log_path))
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    with open(log_path, "a", encoding="utf-8") as f:
        f.write(f"[{timestamp}] {message}\n")
```

Chức năng:

- Logging system với timestamp
- Auto-create log directory
- Append mode để không overwrite logs cũ
- **Format**: `[YYYY-MM-DD HH:MM:SS] message`

Cấu trúc Project

```
project_root/
├── config.json              # Cấu hình dự án
├── utils.py                 # Utility functions
├── predict.py               # Script inference chính
├── models/
│   └── phobert_best.pt      # Saved model weights
├── data/
│   └── processed/
│       ├── augmented_test_2k_with_tda.csv
│       └── predictions_with_tda.csv
└── logs/
    └── training_log.txt     # Training logs
```

Luồng xử lý tổng thể

1. **Initialization Phase**:

```
load_config() → Load project settings
↓
Setup paths, device detection
↓
Load tokenizer + model architecture
↓
load_model() → Load trained weights
```

2. **Inference Phase**:

```
Input text → Tokenization → Model forward → Prediction
↓
save_log() → Log prediction results
```

3. **Batch Processing Phase**:

```
Load CSV → Iterate predictions → Save results
↓
ensure_dir() → Create output directory
↓
Export predictions to CSV
```

Design Patterns được áp dụng

1. Configuration Pattern

- Centralized config management
- Environment-specific settings
- Fallback mechanisms

2. Utility Pattern

- Common functions được tách riêng
- Reusable across different modules
- Single responsibility principle

3. Logging Pattern

- Structured logging với timestamp
- Configurable log paths
- Append-only để preserve history

4. Error Handling

- Graceful degradation (fallback config)
- File existence checks
- Directory auto-creation

Best Practices được implement

1. **Separation of Concerns**: Utils tách biệt khỏi business logic
2. **Configuration Management**: External config file
3. **Path Management**: Relative paths từ project root
4. **Error Resilience**: Fallback mechanisms
5. **Resource Management**: Proper file handling
6. **Logging**: Trackable operations

Kết luận

Đây là một implementation solid cho sentiment analysis tiếng Việt, sử dụng state-of-the-art architecture (BERT) được fine-tune cho domain cụ thể. Code structure rõ ràng với:

- **Modular design**: Tách biệt utilities và main logic
- **Configuration-driven**: Flexible setup thông qua config file
- **Production-ready**: Proper logging, error handling, directory management
- **Maintainable**: Clean code structure, easy to extend và modify

Hệ thống này sẵn sàng cho production deployment và có thể scale up dễ dàng.