Các hàm chính trong thư viện Transformers

1. Model Classes - Các lớp mô hình

1.1 AutoModelForSequenceClassification

python

```python
from transformers import AutoModelForSequenceClassification

# Load model pre-trained cho classification
model = AutoModelForSequenceClassification.from_pretrained(
    "vinai/phobert-base",
    num_labels=3  # Số class cần phân loại
)

# Các tham số quan trọng:
# - num_labels: Số lượng class output
# - output_attentions: Trả về attention weights
# - output_hidden_states: Trả về hidden states của các layer
```

Ví dụ sử dụng:

python

```python
# Load PhoBERT cho sentiment analysis
model = AutoModelForSequenceClassification.from_pretrained(
    "vinai/phobert-base",
    num_labels=3,
    output_attentions=True,
    output_hidden_states=True
)
```

1.2 Các Model Classes khác

python

```python
# Cho câu hỏi - trả lời
from transformers import AutoModelForQuestionAnswering
qa_model = AutoModelForQuestionAnswering.from_pretrained("vinai/phobert-base")

# Cho token classification (NER)
from transformers import AutoModelForTokenClassification
ner_model = AutoModelForTokenClassification.from_pretrained("vinai/phobert-base")

# Cho language modeling
from transformers import AutoModelForMaskedLM
lm_model = AutoModelForMaskedLM.from_pretrained("vinai/phobert-base")

# Cho text generation
from transformers import AutoModelForCausalLM
gen_model = AutoModelForCausalLM.from_pretrained("gpt2")
```

2. Tokenizer Classes

2.1 AutoTokenizer

python

```python
from transformers import AutoTokenizer

# Load tokenizer tương ứng với model
tokenizer = AutoTokenizer.from_pretrained("vinai/phobert-base")

# Các method quan trọng:
# - encode(): Chuyển text thành IDs
# - decode(): Chuyển IDs thành text
# - encode_plus(): Encode với nhiều options hơn
# - batch_encode_plus(): Encode nhiều câu cùng lúc
```

Ví dụ chi tiết:

python

```python
tokenizer = AutoTokenizer.from_pretrained("vinai/phobert-base")

# 1. Encode đơn giản
text = "Tôi yêu Việt Nam"
tokens = tokenizer.encode(text)
print(tokens)  # [0, 2024, 1234, 5678, 9012, 2]

# 2. Encode với options
inputs = tokenizer.encode_plus(
    text,
    max_length=128,        # Độ dài tối đa
    padding="max_length",  # Pad đến max_length
    truncation=True,       # Cắt nếu quá dài
    return_tensors="pt",   # Trả về PyTorch tensors
    return_attention_mask=True,  # Trả về attention mask
    return_token_type_ids=True   # Trả về token type IDs
)

print(inputs.keys())
# dict_keys(['input_ids', 'attention_mask', 'token_type_ids'])

# 3. Decode ngược lại
decoded = tokenizer.decode(tokens, skip_special_tokens=True)
print(decoded)  # "Tôi yêu Việt Nam"

# 4. Batch encoding
texts = ["Câu thứ nhất", "Câu thứ hai", "Câu thứ ba"]
batch_inputs = tokenizer.batch_encode_plus(
    texts,
    max_length=64,
    padding=True,
    truncation=True,
    return_tensors="pt"
)
```

2.2 Tokenizer Methods chi tiết

python

```python
# Tokenize thành từng token
tokens = tokenizer.tokenize("Xin chào Việt Nam")
print(tokens)  # ['Xin', 'chào', 'Việt', 'Nam']

# Convert tokens thành IDs
token_ids = tokenizer.convert_tokens_to_ids(tokens)
print(token_ids)  # [1234, 5678, 9012, 3456]

# Convert IDs thành tokens
tokens_back = tokenizer.convert_ids_to_tokens(token_ids)
print(tokens_back)  # ['Xin', 'chào', 'Việt', 'Nam']

# Lấy vocab size
vocab_size = tokenizer.vocab_size
print(f"Vocab size: {vocab_size}")  # Vocab size: 64000

# Special tokens
print(tokenizer.cls_token)  # <s>
print(tokenizer.sep_token)  # </s>
print(tokenizer.pad_token)  # <pad>
print(tokenizer.unk_token)  # <unk>
```

3. Training Classes

3.1 Trainer

python

```python
from transformers import Trainer, TrainingArguments

# Cấu hình training
training_args = TrainingArguments(
    output_dir="./results",
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=10,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    greater_is_better=True
)

# Khởi tạo Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics_function
)

# Train model
trainer.train()

# Evaluate
eval_results = trainer.evaluate()
```

```
# Save model
trainer.save_model("./saved_model")
```

3.2 TrainingArguments chi tiết

python

```python
training_args = TrainingArguments(
    # Đường dẫn output
    output_dir="./results",

    # Training parameters
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    gradient_accumulation_steps=1,

    # Learning rate
    learning_rate=2e-5,
    lr_scheduler_type="linear",
    warmup_steps=500,

    # Regularization
    weight_decay=0.01,
    max_grad_norm=1.0,

    # Logging
    logging_dir="./logs",
    logging_steps=10,
    logging_strategy="steps",

    # Evaluation
    evaluation_strategy="epoch",
    eval_steps=500,

    # Saving
    save_strategy="epoch",
    save_steps=500,
    save_total_limit=3,
    load_best_model_at_end=True,
```

```
    # Metrics
    metric_for_best_model="accuracy",
    greater_is_better=True,

    # Others
    dataloader_num_workers=4,
    remove_unused_columns=False,
    push_to_hub=False
)
```

4. Pipeline - Sử dụng nhanh

4.1 Text Classification Pipeline

python

```python
from transformers import pipeline

# Tạo pipeline cho classification
classifier = pipeline(
    "text-classification",
    model="vinai/phobert-base",
    tokenizer="vinai/phobert-base"
)

# Sử dụng
results = classifier("Sản phẩm này rất tốt!")
print(results)
# [{'label': 'POSITIVE', 'score': 0.9998}]

# Batch prediction
texts = ["Tốt quá!", "Tệ vãi!", "Bình thường"]
results = classifier(texts)
for text, result in zip(texts, results):
    print(f"{text}: {result['label']} ({result['score']:.4f})")
```

4.2 Các Pipeline khác

python

```python
# Question Answering
qa_pipeline = pipeline("question-answering")
result = qa_pipeline(
    question="Ai là tác giả của Harry Potter?",
    context="Harry Potter được viết bởi J.K. Rowling."
)
print(result['answer'])  # "J.K. Rowling"

# Named Entity Recognition
ner_pipeline = pipeline("ner", aggregation_strategy="simple")
entities = ner_pipeline("Tôi là Nguyễn Văn An, sống ở Hà Nội")

# Text Generation
generator = pipeline("text-generation", model="gpt2")
generated = generator("Hôm nay thời tiết", max_length=50)

# Fill Mask
fill_mask = pipeline("fill-mask")
result = fill_mask("Tôi yêu <mask> Nam")

# Summarization
summarizer = pipeline("summarization")
summary = summarizer("Văn bản dài cần tóm tắt...")
```

5. Configuration Classes

5.1 AutoConfig

python

```python
from transformers import AutoConfig

# Load config của model
config = AutoConfig.from_pretrained("vinai/phobert-base")

print(config.hidden_size)      # 768
print(config.num_attention_heads)  # 12
print(config.num_hidden_layers)   # 12
print(config.vocab_size)       # 64000

# Tạo config tùy chỉnh
custom_config = AutoConfig.from_pretrained(
    "vinai/phobert-base",
    num_labels=5,  # Thay đổi số labels
    hidden_dropout_prob=0.2,  # Thay đổi dropout
    attention_probs_dropout_prob=0.2
)
```

6. Data Processing

6.1 Dataset Classes

python

```python
from transformers import Dataset
import torch

class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

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

# Sử dụng
texts = ["Text 1", "Text 2", "Text 3"]
labels = [0, 1, 2]

encodings = tokenizer(
    texts,
    truncation=True,
    padding=True,
    return_tensors="pt"
)

dataset = CustomDataset(encodings, labels)
```

6.2 DataCollator

python

```python
from transformers import DataCollatorWithPadding

# Tự động padding trong batch
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

# Sử dụng với DataLoader
from torch.utils.data import DataLoader

dataloader = DataLoader(
    dataset,
    batch_size=16,
    collate_fn=data_collator,
    shuffle=True
)
```

7. Utilities và Helper Functions

7.1 Model Save/Load

python

```python
# Save model và tokenizer
model.save_pretrained("./my_model")
tokenizer.save_pretrained("./my_model")

# Load lại
model = AutoModelForSequenceClassification.from_pretrained("./my_model")
tokenizer = AutoTokenizer.from_pretrained("./my_model")

# Save chỉ weights
torch.save(model.state_dict(), "model_weights.pth")

# Load weights
model.load_state_dict(torch.load("model_weights.pth"))
```

7.2 Device Handling

python

```python
import torch

# Auto detect device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Multi-GPU
if torch.cuda.device_count() > 1:
    model = torch.nn.DataParallel(model)
```

8. Ví dụ hoàn chỉnh sử dụng Transformers

python

```python
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    TrainingArguments,
    Trainer,
    pipeline
)
import torch
from torch.utils.data import Dataset
import pandas as pd

# 1. Load model và tokenizer
model_name = "vinai/phobert-base"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(
    model_name,
    num_labels=3
)

# 2. Prepare data
class SentimentDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

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

    def __getitem__(self, idx):
        text = str(self.texts[idx])
        label = self.labels[idx]
```

```
        encoding = self.tokenizer.encode_plus(
            text,
            max_length=self.max_len,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt'
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }

# 3. Training với Trainer
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=64,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer=tokenizer,
)

# Train
trainer.train()
```

```python
# 4. Sử dụng Pipeline cho inference
classifier = pipeline(
    "text-classification",
    model=model,
    tokenizer=tokenizer,
    device=0 if torch.cuda.is_available() else -1
)

# Test
result = classifier("Sản phẩm này thật tuyệt vời!")
print(result)
```

9. Tips và Best Practices

9.1 Memory Optimization

python

```python
# Gradient checkpointing để tiết kiệm memory
model.gradient_checkpointing_enable()

# Mixed precision training
from transformers import TrainingArguments
training_args = TrainingArguments(
    fp16=True,  # Sử dụng float16
    dataloader_pin_memory=False,
    gradient_accumulation_steps=4  # Tích lũy gradient
)
```

9.2 Monitoring Training

python

```python
# Logging với wandb
training_args = TrainingArguments(
    report_to="wandb",
    run_name="phobert-sentiment",
    logging_steps=50
)

# Custom metrics
def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)
    return {
        'accuracy': (predictions == labels).mean(),
        'f1': f1_score(labels, predictions, average='weighted')
    }
```

10. Troubleshooting

10.1 Common Issues

python

```python
# 1. Tokenizer không match với model
# ✅ Luôn sử dụng cùng model name
tokenizer = AutoTokenizer.from_pretrained("vinai/phobert-base")
model = AutoModelForSequenceClassification.from_pretrained("vinai/phobert-base")

# 2. Out of Memory
# ✅ Giảm batch size hoặc max_length
training_args = TrainingArguments(
    per_device_train_batch_size=8,  # Thay vì 16
    gradient_accumulation_steps=2   # Compensate với accumulation
)

# 3. Slow tokenization
# ✅ Sử dụng Fast tokenizer
tokenizer = AutoTokenizer.from_pretrained("vinai/phobert-base", use_fast=True)
```

Transformers là thư viện rất mạnh với đầy đủ tools từ training đến deployment. Phần nào bạn muốn tìm hiểu sâu hơn?