lab6_intro_transformers

Harito ID

2025-11-18

Lab 6: Giới thiệu về Transformers

Mục tiêu:

- Ôn lại kiến thức cơ bản về kiến trúc Transformer.
- Sử dụng các mô hình Transformer tiền huấn luyện (pretrained models) để thực hiện các tác vụ NLP cơ bản.
- Làm quen với thư viện `transformers` của Hugging Face.

---

1. Kiến thức cơ bản: Ôn tập về Transformers

Trước khi đi vào thực hành, chúng ta hãy cùng ôn lại một số khái niệm cốt lõi của kiến trúc Transformer.

1.1. Kiến trúc Transformer Kiến trúc Transformer ban đầu được giới thiệu trong bài báo "Attention Is All You Need" cho nhiệm vụ dịch máy. Nó bao gồm hai phần chính: **Encoder** và **Decoder**.

- **Encoder**: Đọc và hiểu văn bản đầu vào (input text) để tạo ra một chuỗi các biểu diễn (representations) giàu ngữ cảnh.
- **Decoder**: Dựa vào biểu diễn của Encoder và các token đã được sinh ra trước đó để tạo ra văn bản đầu ra (output text).
- **Self-Attention**: Đây là cơ chế cốt lõi cho phép Transformer cân nhắc tầm quan trọng của các từ khác nhau trong câu khi xử lý một từ cụ thể. Nó giúp mô hình nắm bắt được các mối quan hệ ngữ nghĩa và ngữ pháp phức tạp.

1.2. Các loại mô hình Transformer Dựa trên kiến trúc gốc, có ba loại mô hình Transformer chính:

1. **Encoder-only (chỉ Encoder)**:
    - **Ví dụ**: BERT, RoBERTa, ALBERT.
    - **Đặc điểm**: Được huấn luyện để hiểu sâu sắc ngữ cảnh của một câu. Chúng có khả năng nhìn "hai chiều" (bidirectional), tức là xem xét cả các từ đứng trước và sau một từ để hiểu nó.
    - **Tác vụ phù hợp**: Phân loại văn bản, nhận dạng thực thể tên (NER), trả lời câu hỏi, và **Masked Language Modeling (MLM)**.
2. **Decoder-only (chỉ Decoder)**:

- **Ví dụ**: GPT (GPT-2, GPT-3), BLOOM.
- **Đặc điểm**: Được huấn luyện để dự đoán từ tiếp theo trong một chuỗi. Chúng chỉ có khả năng nhìn "một chiều" (unidirectional), tức là chỉ xem xét các từ đã xuất hiện trước đó.
- **Tác vụ phù hợp**: Sinh văn bản (text generation), **Next Token Prediction**.

3. **Encoder-Decoder**:
   - **Ví dụ**: T5, BART, MarianMT.
   - **Đặc điểm**: Kết hợp cả hai thành phần, phù hợp cho các tác vụ chuyển đổi từ chuỗi này sang chuỗi khác (sequence-to-sequence).
   - **Tác vụ phù hợp**: Dịch máy, tóm tắt văn bản.

---

2. Cài đặt

Để thực hiện các bài tập, bạn cần cài đặt thư viện `transformers` và `torch`.

```
pip install transformers torch
```

---

3. Bài tập thực hành

Chúng ta sẽ sử dụng `pipeline` của Hugging Face, một công cụ trừu tượng hóa cao giúp dễ dàng sử dụng các mô hình phức tạp cho các tác vụ cụ thể.

Bài 1: Khôi phục Masked Token (Masked Language Modeling) Trong tác vụ này, chúng ta sẽ che một vài từ trong câu bằng một token đặc biệt (thường là `[MASK]`) và yêu cầu mô hình dự đoán xem từ gốc là gì. Chúng ta sẽ sử dụng một mô hình thuộc họ BERT.

Yêu cầu: Sử dụng pipeline `fill-mask` để dự đoán từ bị thiếu trong câu sau: `Hanoi is the [MASK] of Vietnam.`

Code mẫu:

```python
from transformers import pipeline

# 1. Tải pipeline "fill-mask"
# Pipeline này sẽ tự động tải một mô hình mặc định phù hợp (thường là một biến thể của
mask_filler = pipeline("fill-mask")

# 2. Câu đầu vào với token [MASK]
input_sentence = "Hanoi is the [MASK] of Vietnam."

# 3. Thực hiện dự đoán
# top_k=5 yêu cầu mô hình trả về 5 dự đoán hàng đầu
predictions = mask_filler(input_sentence, top_k=5)

# 4. In kết quả
print(f"Câu gốc: {input_sentence}")
for pred in predictions:
    print(f"Dự đoán: '{pred['token_str']}' với độ tin cậy: {pred['score']:.4f}")
    print(f"  -> Câu hoàn chỉnh: {pred['sequence']}")
```

Câu hỏi:

1. Mô hình đã dự đoán đúng từ `capital` không?
2. Tại sao các mô hình Encoder-only như BERT lại phù hợp cho tác vụ này?

Bài 2: Dự đoán từ tiếp theo (Next Token Prediction) Tác vụ này yêu cầu mô hình sinh ra phần tiếp theo của một đoạn văn bản cho trước. Chúng ta sẽ sử dụng một mô hình thuộc họ GPT.

Yêu cầu: Sử dụng pipeline `text-generation` để sinh ra phần tiếp theo cho câu: `The best thing about learning NLP is`

Code mẫu:

```
from transformers import pipeline

# 1. Tải pipeline "text-generation"
# Pipeline này sẽ tự động tải một mô hình phù hợp (thường là GPT-2)
generator = pipeline("text-generation")

# 2. Đoạn văn bản mồi
prompt = "The best thing about learning NLP is"

# 3. Sinh văn bản
# max_length: tổng độ dài của câu mồi và phần được sinh ra
# num_return_sequences: số lượng chuỗi kết quả muốn nhận
generated_texts = generator(prompt, max_length=50, num_return_sequences=1)

# 4. In kết quả
print(f"Câu mồi: '{prompt}'")
for text in generated_texts:
    print("Văn bản được sinh ra:")
    print(text['generated_text'])
```

Câu hỏi:

1. Kết quả sinh ra có hợp lý không?
2. Tại sao các mô hình Decoder-only như GPT lại phù hợp cho tác vụ này?

Bài 3: Tính toán Vector biểu diễn của câu (Sentence Representation) Một trong những ứng dụng mạnh mẽ nhất của BERT là khả năng chuyển đổi một câu thành một vector số có chiều dài cố định, nắm bắt được ngữ nghĩa của câu đó. Vector này có thể được dùng cho các tác vụ khác như phân loại, tìm kiếm tương đồng, v.v.

Có hai cách phổ biến để lấy vector biểu diễn cho cả câu:

1. Lấy vector đầu ra của token `[CLS]` (token đặc biệt được thêm vào đầu mỗi câu).
2. Lấy trung bình cộng của các vector đầu ra của tất cả các token trong câu (Mean Pooling). Cách này thường cho kết quả tốt hơn.

Yêu cầu: Viết code để tính toán vector biểu diễn cho câu `This is a sample sentence.` bằng phương pháp Mean Pooling.

Code mẫu:

```
import torch
from transformers import AutoTokenizer, AutoModel
```

```python
# 1. Chọn một mô hình BERT
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModel.from_pretrained(model_name)

# 2. Câu đầu vào
sentences = ["This is a sample sentence."]

# 3. Tokenize câu
# padding=True: đệm các câu ngắn hơn để có cùng độ dài
# truncation=True: cắt các câu dài hơn
# return_tensors='pt': trả về kết quả dưới dạng PyTorch tensors
inputs = tokenizer(sentences, padding=True, truncation=True, return_tensors='pt')

# 4. Đưa qua mô hình để lấy hidden states
# torch.no_grad() để không tính toán gradient, tiết kiệm bộ nhớ
with torch.no_grad():
    outputs = model(**inputs)

# outputs.last_hidden_state chứa vector đầu ra của tất cả các token
last_hidden_state = outputs.last_hidden_state
# shape: (batch_size, sequence_length, hidden_size)

# 5. Thực hiện Mean Pooling
# Để tính trung bình chính xác, chúng ta cần bỏ qua các token đệm (padding tokens)
attention_mask = inputs['attention_mask']
mask_expanded = attention_mask.unsqueeze(-1).expand(last_hidden_state.size()).float()
sum_embeddings = torch.sum(last_hidden_state * mask_expanded, 1)
sum_mask = torch.clamp(mask_expanded.sum(1), min=1e-9)
sentence_embedding = sum_embeddings / sum_mask

# 6. In kết quả
print("Vector biểu diễn của câu:")
print(sentence_embedding)
print("\nKích thước của vector:", sentence_embedding.shape)
```

Câu hỏi:

1. Kích thước (chiều) của vector biểu diễn là bao nhiêu? Con số này tương ứng với tham số nào của mô hình BERT?
2. Tại sao chúng ta cần sử dụng `attention_mask` khi thực hiện Mean Pooling?

---

4. Nộp bài

Hoàn thành các đoạn code và trả lời các câu hỏi trong file notebook/markdown và nộp lại theo hướng dẫn.