lab5_rnn_for_ner

Harito ID

2025-11-18

Lab 5: Xây dựng mô hình RNN cho bài toán Nhận dạng Thực thể Tên (NER)

1. Mục tiêu

Trong bài thực hành này, chúng ta sẽ tiếp tục áp dụng Mạng Nơ-ron Hồi quy (RNN) để xây dựng một mô hình hoàn chỉnh cho bài toán Nhận dạng Thực thể Tên (Named Entity Recognition - NER).

Sau khi hoàn thành bài lab, bạn có thể:

- Tải và tiền xử lý dữ liệu NER từ thư viện `datasets` của Hugging Face.
- Xây dựng từ điển (vocabulary) cho từ và nhãn NER.
- Tạo một lớp `Dataset` tùy chỉnh trong PyTorch cho bài toán token classification.
- Xây dựng một mô hình RNN đơn giản sử dụng `nn.Embedding`, `nn.RNN`, và `nn.Linear`.
- Huấn luyện và đánh giá hiệu năng của mô hình trên bộ dữ liệu CoNLL 2003.

2. Bộ dữ liệu: CoNLL 2003

Chúng ta sẽ sử dụng bộ dữ liệu CoNLL 2003, một trong những bộ dữ liệu benchmark tiêu chuẩn cho bài toán NER. Thay vì đọc file thủ công, chúng ta sẽ sử dụng thư viện `datasets` của Hugging Face để tải và quản lý dữ liệu một cách tiện lợi.

Dữ liệu NER được gán nhãn theo định dạng IOB (Inside, Outside, Beginning). Ví dụ:

- `B-PER`: Bắt đầu một thực thể Tên người (Person).
- `I-PER`: Bên trong một thực thể Tên người.
- `B-LOC`: Bắt đầu một thực thể Địa điểm (Location).
- `I-LOC`: Bên trong một thực thể Địa điểm.
- `O`: Không phải là một thực thể (Outside).

Ví dụ một câu đã được gán nhãn:

U.N.	official	Ekeus	heads	for	Baghdad	.
B-ORG	O	B-PER	O	O	B-LOC	O

3. Các bước thực hiện

Task 1: Tải và Tiền xử lý Dữ liệu

1. **Tải dữ liệu từ Hugging Face:**

- Sử dụng hàm `datasets.load_dataset("conll2003", trust_remote_code=True)` để tải bộ dữ liệu. Thao tác này có thể mất vài phút.
- Dữ liệu trả về là một `DatasetDict` chứa các split: `train`, `validation`, và `test`.

2. **Trích xuất câu và nhãn:**
   - Từ đối tượng dataset đã tải, hãy trích xuất các câu (danh sách token) và các nhãn tương ứng.
   - `train_sentences = dataset["train"]["tokens"]`
   - `train_tags = dataset["train"]["ner_tags"]`
   - **Lưu ý:** Nhãn `ner_tags` đang ở dạng số nguyên. Bạn cần lấy ánh xạ từ số sang tên nhãn (string) bằng cách truy cập `dataset["train"].features["ner_tags"].featur` Hãy chuyển đổi tất cả các nhãn số về dạng string (ví dụ: `B-PER`, `I-PER`, `O`).
3. **Xây dựng Từ điển (Vocabulary):**
   - Từ dữ liệu huấn luyện (train), tạo ra hai từ điển:
     - `word_to_ix`: Ánh xạ mỗi từ duy nhất sang một chỉ số (index) nguyên. Thêm một token đặc biệt là `<UNK>` cho các từ không có trong từ điển và `<PAD>` cho việc đệm (padding).
     - `tag_to_ix`: Ánh xạ mỗi nhãn NER (dạng string) duy nhất sang một chỉ số nguyên.
   - In ra kích thước của hai từ điển này.

Task 2: Tạo PyTorch Dataset và DataLoader

1. **Tạo lớp `NERDataset`:**
   - Tạo một lớp kế thừa từ `torch.utils.data.Dataset`.
   - `__init__`: Nhận vào danh sách các câu (dạng token), danh sách các chuỗi nhãn, và hai từ điển `word_to_ix`, `tag_to_ix`.
   - `__len__`: Trả về tổng số câu trong bộ dữ liệu.
   - `__getitem__`: Nhận vào một `index` và trả về một cặp tensor: `(sentence_indices, tag_indices)`. Tensor này chứa các chỉ số của từ/nhãn trong câu tương ứng. Sử dụng token `<UNK>` cho các từ không có trong `word_to_ix`.
2. **Tạo `DataLoader`:**
   - Khởi tạo `DataLoader` cho cả tập train và validation.
   - Viết một hàm `collate_fn` để đệm (pad) các câu và nhãn trong cùng một batch về cùng một độ dài (độ dài của câu dài nhất batch đó). Sử dụng `torch.nn.utils.rnn.pad_sequence` với `batch_first=True`. Giá trị padding cho câu nên là index của token `<PAD>`, và giá trị padding cho nhãn có thể là một số nguyên đặc biệt (ví dụ: -1 hoặc index của một nhãn `<PAD>` nếu bạn thêm nó vào `tag_to_ix`).

Task 3: Xây dựng Mô hình RNN

- Dựa trên lớp `SimpleRNNForTokenClassification` đã thấy trong bài lab về POS tagging, hãy định nghĩa lại mô hình.
- Mô hình sẽ bao gồm 3 lớp chính:
  1. `nn.Embedding`: Chuyển đổi chỉ số của từ thành vector.
  2. `nn.RNN` (hoặc `nn.LSTM`, `nn.GRU` để có kết quả tốt hơn): Xử lý chuỗi vector embedding.
  3. `nn.Linear`: Ánh xạ output của RNN sang không gian nhãn để dự đoán.

- Hãy khởi tạo mô hình với các tham số phù hợp: `vocab_size`, `embedding_dim`, `hidden_dim`, và `output_size` (số lượng nhãn NER).

Task 4: Huấn luyện Mô hình

1. **Khởi tạo:**
    - Khởi tạo mô hình, optimizer (ví dụ: `torch.optim.Adam`), và loss function.
    - Sử dụng `nn.CrossEntropyLoss`. Đây là lựa chọn phù hợp cho bài toán phân loại đa lớp trên từng token.
    - **Quan trọng:** Thiết lập tham số `ignore_index` của `CrossEntropyLoss` bằng với giá trị bạn đã dùng để đệm cho nhãn trong `collate_fn`. Điều này giúp loss function bỏ qua các vị trí padding khi tính toán.
2. **Viết vòng lặp huấn luyện:**
    - Lặp qua một số lượng `epoch` nhất định (ví dụ: 3-5 epochs).
    - Trong mỗi epoch, lặp qua từng `batch` từ `DataLoader` huấn luyện.
    - Thực hiện 5 bước kinh điển: (1) Xóa gradient cũ, (2) Forward pass, (3) Tính loss, (4) Backward pass, (5) Cập nhật trọng số.
    - In ra giá trị loss trung bình sau mỗi epoch.

Task 5: Đánh giá Mô hình

1. **Viết hàm `evaluate`:**
    - Đặt mô hình ở chế độ đánh giá: `model.eval()`.
    - Tắt việc tính toán gradient: `with torch.no_grad(): ...`
    - Lặp qua từng batch trong `DataLoader` của tập validation.
    - Với mỗi batch, lấy dự đoán của mô hình bằng cách áp dụng `torch.argmax` trên chiều cuối cùng của output.
    - So sánh dự đoán với nhãn thật để tính toán độ chính xác (accuracy). **Lưu ý:** chỉ tính accuracy trên các token không phải là padding.
    - (Nâng cao) Để đánh giá NER một cách chính xác hơn, người ta thường dùng các chỉ số như Precision, Recall, và F1-score trên từng loại thực thể. Thư viện `seqeval` có thể giúp bạn làm điều này.
2. **Báo cáo kết quả:**
    - In ra độ chính xác cuối cùng trên tập validation.
    - Viết một hàm `predict_sentence(sentence)` nhận vào một câu mới (dạng chuỗi), xử lý nó và in ra các cặp `(từ, nhãn_dự_đoán)`.

4. Nộp bài

- Nộp link report cho bài lab.
- Ghi lại kết quả độ chính xác cuối cùng trên tập validation vào cuối file này.

KẾT QUẢ THỰC HIỆN

(Dành cho sinh viên điền vào)

- **Độ chính xác trên tập validation:** ...
- **Ví dụ dự đoán câu mới:**

- **Câu:** “VNU University is located in Hanoi”
- **Dự đoán:** …