자연언어처리 기말 프로젝트

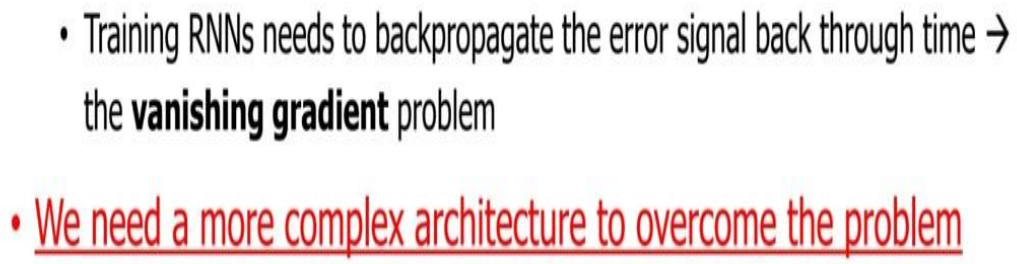
AI융합학부 20213084 이성일

 # 목차

모델 선택

먼저 과제 환경 구축 (torch와 cuda환경을 구축하고 실행을 개인 노트북으로 진행하였음) 및 문제 해결을 위한 시간 등의 여건을 관리 하지 못한 저의 불찰로 적절한 모델 선택 기준으로 다양한 실험을 하지 못하였습니다.

그러나 제시된 모델들 중 MLP 등의 단순 신경망 보단 자연어처리에 적합한 RNN 관련 모델이 절절한 학습을 진행하고 좋아 보였고 강의 자료 등을 참고하여 RNN이 가지고 있는 vanishing gradient 모델을 overcome하는 gates mechanism이 추가된 LSTM 모델을 고안하였습니다.



(강의 자료 중 RNN의 문제에 관한 slice)

모델 구조 설명

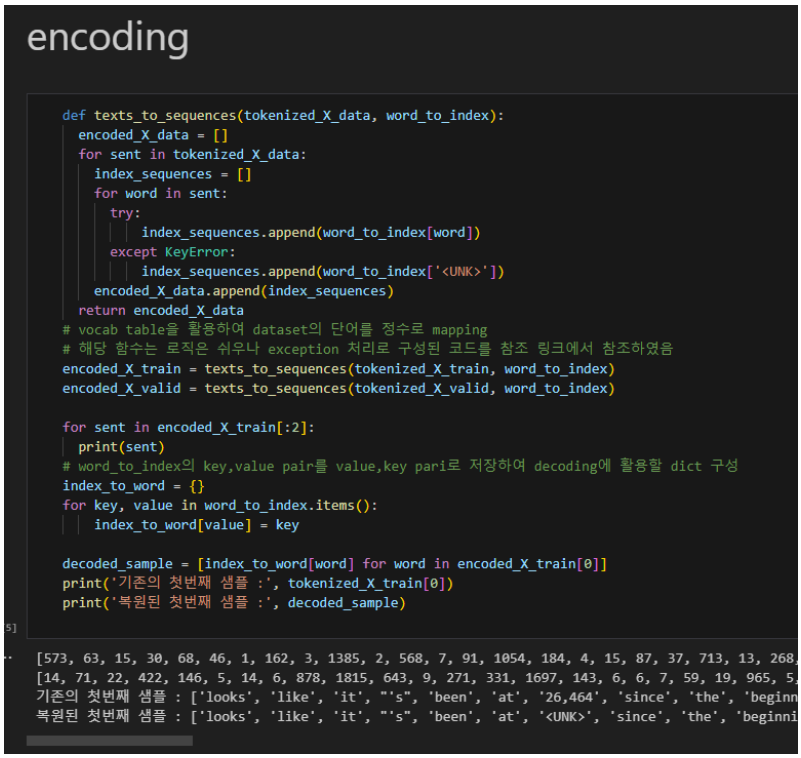
먼저 뒤 참조 section에서 첨부한 link에서 A – Z 까지 언어모델설계와 review data 등을 변환하고, embedding을 구성하는 과정을 예시코드 등으로 친절하게 설명해주었고, 과제 명세상 해당 자료를 참조하여도 큰 문제가 없다 판단하여 해당 link의 흐름을 한 줄 한 줄 이해하며 열심히 따라가서 구현하였습니다.

제출 코드에 자세히 제가 이해하고 수정하며 추가한 주석을 추가하여 놓았고 해당 보고서에는 코드의 핵심 logic을 간단한 캡처본과 함께 정리 제시 하겠습니다.

1. Data preprocessing 및 feature 구성

먼저 comment는 nltk white tokenizer로 tokenize하고 중간 과제와 마찬가지 방식으로 vocab을 구성하였습니다. (min_occur 및 counter 등을 동일하게 사용)

이후 빈도로 idx mapping이 된 vocab dict의 값을 활용하여 comments의 각 vocab들을 정수로 encoding 하여 추후 pre-trained embedding 활용없이 word embedding을 구성할 수 있다는 아이디어를 참조 링크를 통해 얻을 수 있었고, 길이가 다른 comments들에 padding을 해주는 과정으로 train data를 구성하는 방식을 참조 하였습니다.



```python
# np array로 데이터 feature 생성 및 인자값에 맞게 padding
def pad_sequences(sentences, max_len):
    features = np.zeros((len(sentences), max_len), dtype=int)
    for index, sentence in enumerate(sentences):
        if len(sentence) != 0:
            features[index, :len(sentence)] = np.array(sentence)[:max_len]
            # sentence 길이에 해당하는 값을 feature array에 넣어준다, array 복사 연산 또한 차원을 맞춰주었음
    return features

padded_X_train = pad_sequences(encoded_X_train, max_len=max_len)
padded_X_valid = pad_sequences(encoded_X_valid, max_len=max_len)

print('훈련 데이터의 크기 :', padded_X_train.shape)
print('검증 데이터의 크기 :', padded_X_valid.shape)
```

2. Model 구성

```python
# 최종 단어 집합크기, em_dim, hid_dim, out_dim (label 2개에 맞추어 2로 설정) , dropout_p -> 50% 제공
# Embedding 층을 통해 단어집합을 vector로구성
# lstm 모델을 사용, 0차원을 batch로구성
# 최종 output 출력을 위한 linear layer -> 2
class TextModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, output_dim, dropout_p = 0.5):
        super(TextModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim)
        self.dropout_p = dropout_p

    def forward(self, x):
        embedded = self.embedding(x)  # (batch_size, seqence 길이) -> (batch_size, seqence 길이, embedding_dim)

        # LSTM은 (hidden state, cell state)를 반환한다
        lstm_out, (hidden, cell) = self.lstm(embedded)  # lstm_out: (batch_size, seqence 길이, hidden_dim), hidden: (1, batch_size, hidden_dim)

        last_hidden = hidden.squeeze(0)  # (batch_size, hidden_dim)
        logits = self.fc(last_hidden)  # (batch_size, output_dim) # 최종 2개의 label값에 대한 추론
        return logits
```

인자 전달 및, 입력 차원 구성이 mismatch 나지 않게 입-출력 차원을 주석처리하며 모델을 구성하였고 기본적 torch 신경망 구성 로직을 갖추는 것을 목표로 하였습니다.

사전에 설명하였듯 LSTM을 사용하여 model을 구성하였고 출력차원은 label의 범주인 2로 설정하여 주었습니다. 이는 추후에 softmax등의 함수를 사용하여 test 결과에 맞는 확률로 출력하였습니다. 이외의 embedding_dim, hidden_dim 과 같은 hyper params는 100과 128을 주었고, 데이터 augmentation 등이 없어 overfitting위험이 있다 판단하여 dropout_p 인자를 추가로 사용하였습니다.

3. Train and Test

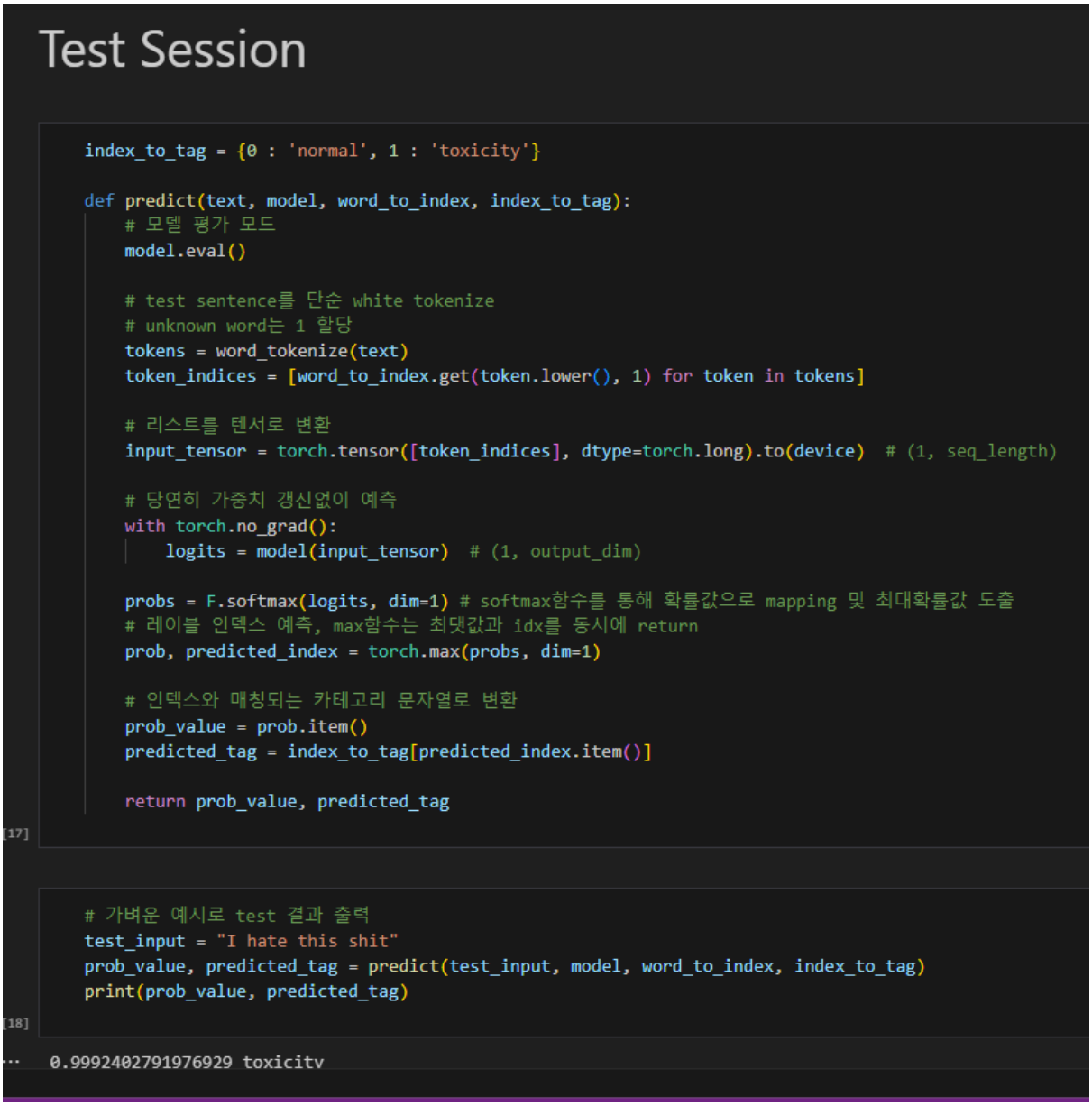
기본적 신경망 규칙에 맞게 코드를 구성하고 학습을 진행하였습니다. 분리한 train set으로 학습을 진행하였고 valid set으로 모델을 평가하였습니다. Adam optimizer, CrossEntropy loss function 등을 활용해 오류역전파를 통한 가중치 갱신을 하였습니다.

Batch size는 32에서 안정적인 성능 (accuracy)를 이미 출력한다고 판단하여 고정하였습니다.

Num_epochs 시간, 정확도 측면 등을 고려하여 10으로 설정하였습니다.

```
num_epochs = 10

# 검증 loss init
best_val_loss = float('inf')

# Training loop
# tqdm으로 진행상태 표시
for epoch in tqdm(range(num_epochs)):
    # Training
    train_loss = 0
    train_correct = 0
    train_total = 0
    model.train()
    for batch_X, batch_y in train_dataloader:
        # Forward pass (32 size batch)
        batch_X, batch_y = batch_X.to(device), batch_y.to(device)
        # batch_X.shape == (batch_size, max_len)
        logits = model(batch_X)

        # loss 계산
        loss = criterion(logits, batch_y)
        # 가중치 초기화
        optimizer.zero_grad()
        # 오류 역전파
        loss.backward()
        optimizer.step()

        # Calculate training accuracy and loss
        train_loss += loss.item()
        train_correct += calculate_accuracy(logits, batch_y) * batch_y.size(0)
        train_total += batch_y.size(0)

    train_accuracy = train_correct / train_total
    train_loss /= len(train_dataloader)

    # Validation
    val_loss, val_accuracy = evaluate(model, valid_dataloader, criterion, device)

    print(f'Epoch {epoch+1}/{num_epochs}:')
    print(f'Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.4f}')
    print(f'Validation Loss: {val_loss:.4f}, Validation Accuracy: {val_accuracy:.4f}')
```

Test 단계에서는 가중치 갱신 없이 sentence 한 개씩 받아 tensor로 변환하여 예측값을 softmax활성화 함수를 활용하여 확률로 출력해주었고, 이를 torch.max함수를 통해pred값



과 함께 출력형식을 맞추고, submission.csv파일로 만들어 저장하였습니다.

결과 및 참조

submission.csv

```
,probability,pred
0,0.9955850839614868,normal
1,0.9984419941902161,normal
2,0.9998955726623535,normal
3,0.9999489784240723,normal
4,0.9153735041618347,normal
5,0.9975278973579407,normal
6,0.9997950196266174,normal
7,0.9978054165840149,normal
8,0.9998101592063904,normal
9,0.9999586343765259,normal
10,0.999164342880249,normal
11,0.9997186064720154,normal
12,0.9287434816360474,toxicity
13,0.6031337976455688,normal
14,0.9918416738510132,normal
15,0.6604557037353516,toxicity
16,0.9999082088470459,normal
17,0.9997256398200989,normal
18,0.9523900747299194,normal
19,0.9995636343955994,normal
20,0.9868558049201965,normal
21,0.9997411370277405,normal
```

```
 50%|                                                  | 5/10 [05:53<05:56, 71.21s/it]
Epoch 5/10:
Train Loss: 0.3130, Train Accuracy: 0.9052
Validation Loss: 0.3180, Validation Accuracy: 0.9043

 60%|                                                  | 6/10 [07:06<04:46, 71.74s/it]
Epoch 6/10:
Train Loss: 0.3133, Train Accuracy: 0.9052
Validation Loss: 0.3173, Validation Accuracy: 0.9041

 70%|                                                  | 7/10 [08:17<03:34, 71.60s/it]
Epoch 7/10:
Train Loss: 0.2295, Train Accuracy: 0.9223
Validation Loss: 0.1610, Validation Accuracy: 0.9452
Validation loss improved from 0.3149 to 0.1610. 체크포인트를 저장합니다.

 80%|                                                  | 8/10 [09:28<02:23, 71.52s/it]
Epoch 8/10:
Train Loss: 0.1458, Train Accuracy: 0.9437
Validation Loss: 0.1419, Validation Accuracy: 0.9495
Validation loss improved from 0.1610 to 0.1419. 체크포인트를 저장합니다.

 90%|                                                  | 9/10 [10:39<01:11, 71.40s/it]
Epoch 9/10:
Train Loss: 0.1025, Train Accuracy: 0.9633
Validation Loss: 0.1300, Validation Accuracy: 0.9552
Validation loss improved from 0.1419 to 0.1300. 체크포인트를 저장합니다.
100%|                                                 | 10/10 [11:51<00:00, 71.1
Epoch 10/10:
Train Loss: 0.0789, Train Accuracy: 0.9719
Validation Loss: 0.1344, Validation Accuracy: 0.9573
```

Do you wa
extension

추가적으로 현재 본인은 노트북에 torch 및 cuda dependency 파일을 설치하여 cuda로 코드를 실행하였습니다. 파일 첫번째 셀의 라이브러리만 잘 import 된다면 문제없이 실행되는 바 입니다. 이하는 참조 문헌입니다.

해당 문헌은 이번 과제와 관련되어 잘 짜여진 솔루션을 자세하게 제공하고 있지만, LLM이 생성한 코드 등을 활용하거나 open소스를 활용하기 보다 학습적인 측면 및 자연어처리에 관한 이해적인 측면에서 도움되는 문헌이라고 판단하여, 열심히 분석하고, 모르는 부분은 검색해가며 이해하였습니다. 해당 문헌의 전체적인 logic을 따라가며 작성했으나 또한 코드 흐름을 제 방식대로 다시 구성하거나, 제출 코드의 거의 매 라인 또는 단락마다 흐름 및 이해에 관한 주석을 달았습니다.

(https://wikidocs.net/217083) GRU로 IMDB 리뷰분류 관련

(https://wikidocs.net/64779) 임베딩 벡터 생성 관련