파이토치로 배우는 자연어 처리
6장. 자연어 처리를 위한 시퀀스 모델링 - 초급

박채원

목차

들어가며

시퀀스 : 순서가 있는 항목의 모음
전통적인 머신러닝은 데이터 포인트가 독립동일분포라고 가정하지만, 언어, 음성, 시계열 처럼 한 데이터 항목이 앞뒤 항목에 의존하기도 함.
언어를 이해하려면 시퀀스 이해는 필수
이전 장에서 다룬 다층 퍼셉트론, 합성곱 신경망 같은 피드 포워드 신경망은 시퀀스를 적절하게 모델링하지 못한다.

딥러닝에서 시퀀스 모델링은 숨겨진 '상태 정보' 또는 은닉 상태를 유지하는 것과 관련있다.
은닉 상태(시퀀스 표현)는 지금까지 시퀀스에서 본 모든 정보를 담는다.

이 장에선 시퀀스 데이터 분류를 배운다.
1. 기본적인 신경망 시퀀스 모델인 순환 신경망을 소개
2. RNN을 이용해 엔드-투-엔드 분류 예제(성씨 분류 예제)를 다룬다. (성씨 분류 예제에선 시퀀스 모델이 언어의 철자 패턴을 감지할 수 있는지 살펴본다.)

순환 신경망 소개

RNN의 목적은 시퀀스 텐서를 모델링 하는 것이다. RNN에는 여러 모델이 있는데 이 장에선 가장 기본적인 형태인 **엘만** RNN 만을 다룬다.
시퀀스 표현을 학습 -> 이를 위해 시퀀스의 현재 상태를 감지하는 은닉 상태 벡터를 관리한다.
현재 입력 벡터와 이전 은닉 상태 벡터로 은닉 상태 벡터를 계산한다.

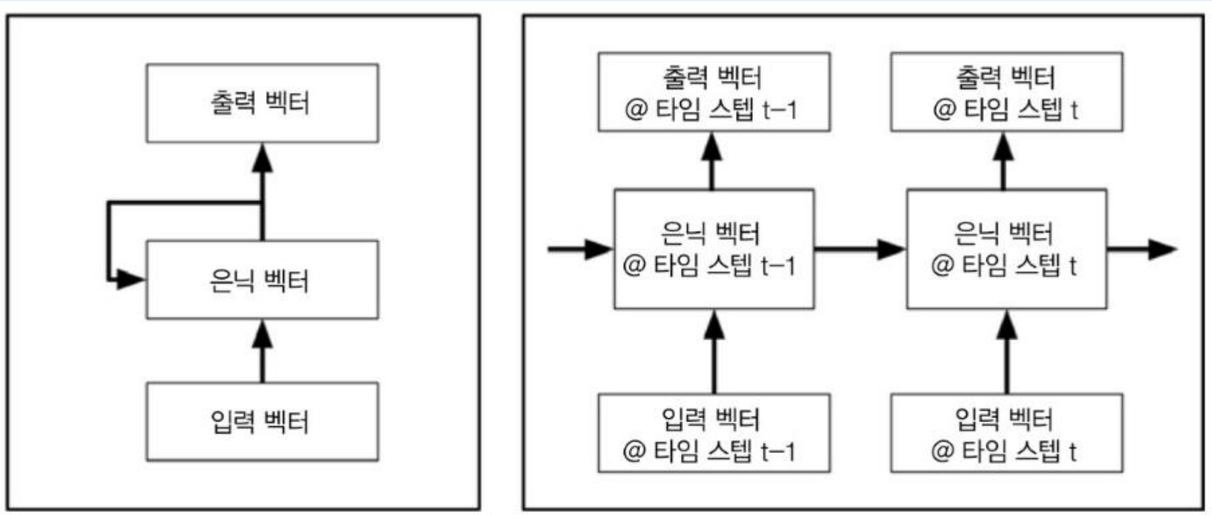


그림 6-1 엘만 RNN의 기능적 표현(왼쪽)은 피드백 루프를 통한 은닉 벡터의 순환 관계를 보여줍니다. 펼친 표현(오른쪽)은 계산 관계를 명확하게 보여줍니다. 각 타임 스텝의 은닉 벡터는 현재 타임 스텝의 입력과 이전 타임 스텝의 은닉 벡터에 의존합니다.

왼쪽 : 기능적 표현　　오른쪽 : 펼친 표현

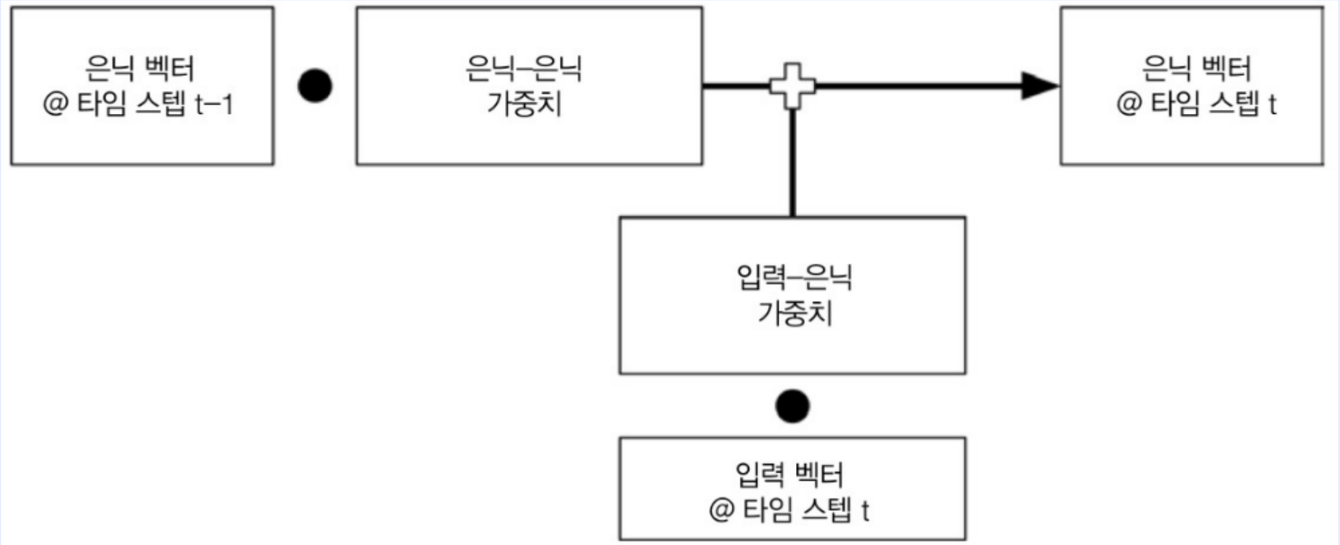
두 그림의 출력은 은닉 상태로 동일하다.

현재 타임 스텝의 입력 벡터와 이전 타임 스텝의 은닉 상태 벡터는 현재 타임 스텝의 은닉 상태 벡터에 매핑된다.

순환 신경망 소개

은닉-은닉 가중치 행렬을 사용해 이전 은닉 상태 벡터를 매핑하고 입력 -은닉 가중치 행렬을 사용해 입력 벡터를 매핑하여 새로운 은닉 벡터를 계산한다.

단어와 문장은 길이가 다양하므로 RNN 같은 시퀀스 모델은 가변 길이 시퀀스를 다룰 수 있어야한다.
시퀀스의 길이를 맞추기 위해서 이 책에선 **마스킹**이란 방법을 사용한다.
(간단히 말해, 마스킹을 사용하면 어떤 입력이 그레이디언트나 최종 출력에 포함되어서는 안될 때 신호를 보낼 수 있다.)



엘만 RNN 구현하기

```python
class ElmanRNN(nn.Module):
    """ RNNCell을 사용하여 만든 엘만 RNN """
    def __init__(self, input_size, hidden_size, batch_first=False):
        """
        매개변수:
            input_size (int): 입력 벡터 크기
            hidden_size (int): 은닉 상태 벡터 크기
            batch_first (bool): 0번째 차원이 배치인지 여부
        """
        super(ElmanRNN, self).__init__()
        self.rnn_cell = nn.RNNCell(input_size, hidden_size)

        self.batch_first = batch_first
        self.hidden_size = hidden_size
```

```python
def _initialize_hidden(self, batch_size):
    return torch.zeros((batch_size, self.hidden_size))

def forward(self, x_in, initial_hidden=None):
    """ ElmanRNN의 정방향 계산

    매개변수:
        x_in (torch.Tensor): 입력 데이터 텐서
            If self.batch_first: x_in.shape = (batch_size, seq_size, feat_size)
            Else: x_in.shape = (seq_size, batch_size, feat_size)
        initial_hidden (torch.Tensor): RNN의 초기 은닉 상태
    반환값:
        hiddens (torch.Tensor): 각 타임 스텝에서 RNN 출력
            If self.batch_first:
                hiddens.shape = (batch_size, seq_size, hidden_size)
            Else: hiddens.shape = (seq_size, batch_size, hidden_size)
    """
    if self.batch_first:
        batch_size, seq_size, feat_size = x_in.size()
        x_in = x_in.permute(1, 0, 2)
    else:
        seq_size, batch_size, feat_size = x_in.size()

    hiddens = []

    if initial_hidden is None:
        initial_hidden = self._initialize_hidden(batch_size)
        initial_hidden = initial_hidden.to(x_in.device)

    hidden_t = initial_hidden

    for t in range(seq_size):
        hidden_t = self.rnn_cell(x_in[t], hidden_t)
        hiddens.append(hidden_t)

    hiddens = torch.stack(hiddens)

    if self.batch_first:
        hiddens = hiddens.permute(1, 0, 2)

    return hiddens
```

forward 함수는 입력 텐서를 순회하면서 타임 스텝마다 은닉 상태 벡터를 계산한다.

이 클래스의 출력은 3차원 텐서이다.
배치에 있는 각 데이터 포인트와 타임 스텝에 대한 은닉 상태 벡터이다.

Surname Dataset 클래스

4장과 마찬가지로 문자 시퀀스 (성씨) 를 국적에 따라 분류
성씨 데이터셋의 각 데이터 포인트는 성씨와 해당하는 국적으로 표현된다.

```
class SurnameDataset(Dataset):
    @classmethod
    def load_dataset_and_make_vectorizer(cls, surname_csv):
        """데이터셋을 로드하고 새로운 Vectorizer 객체를 만듭니다

        매개변수:
            surname_csv (str): 데이터셋의 위치
        반환값:
            SurnameDataset의 객체
        """
        surname_df = pd.read_csv(surname_csv)
        train_surname_df = surname_df[surname_df.split=='train']
        return cls(surname_df, SurnameVectorizer.from_dataframe(train_surname_df))

    def __getitem__(self, index):
        """파이토치 데이터셋의 주요 진입 메서드

        매개변수:
            index (int): 데이터 포인트 인덱스
        반환값:
            다음 값을 담고 있는 딕셔너리:
                특성 (x_data)
                레이블 (y_target)
                특성 길이 (x_length)
        """
        row = self._target_df.iloc[index]

        surname_vector, vec_length = \
            self._vectorizer.vectorize(row.surname, self._max_seq_length)

        nationality_index = \
```

SurnameDataset 클래스는 벡터로 변환된 성씨와 국적을 나타내는 정수를 반환한다.
추가로 시퀀스의 길이를 반환.

```
            self._vectorizer.nationality_vocab.lookup_token(row.nationality)

        return {'x_data': surname_vector,
                'y_target': nationality_index,
                'x_length': vec_length}
```

데이터 구조

벡터 변환 파이프라인의 첫 번째 단계는 성씨에 있는 각 문자 토큰을 고유한 정수에 매핑하는 작업
5장에서 소개한 SequenceVocabulary를 사용
정수에 매핑할 뿐만 아니라 특별한 목적의 토큰도 활용.

UNK 토큰 - 입력에 어휘 사전에 없는 토큰이 있을 때
MASK 토큰 - 가변 길이 입력을 처리하는 데 사용
BEGIN-OF-SEQUENCE 토큰 - 시퀀스 앞에 추가
END-OF-SEQUENCE 토큰 - 시퀀스 뒤에 추가

```
class SurnameVectorizer(object):
    """ 어휘 사전을 생성하고 관리합니다 """
    def vectorize(self, surname, vector_length=-1):
        """
        매개변수:
            title (str): 문자열
            vector_length (int): 인덱스 벡터의 길이를 맞추기 위한 매개변수
        """
        indices = [self.char_vocab.begin_seq_index]
        indices.extend(self.char_vocab.lookup_token(token)
                       for token in surname)
        indices.append(self.char_vocab.end_seq_index)

        if vector_length < 0:
            vector_length = len(indices)

        out_vector = np.zeros(vector_length, dtype=np.int64)
        out_vector[:len(indices)] = indices
        out_vector[len(indices):] = self.char_vocab.mask_index

        return out_vector, len(indices)

    @classmethod
    def from_dataframe(cls, surname_df):
        """데이터셋 데이터프레임으로 SurnameVectorizer 객체를 초기화합니다.

        매개변수:
            surname_df (pandas.DataFrame): 성씨 데이터셋
        반환값:
            SurnameVectorizer 객체
        """
        char_vocab = SequenceVocabulary()
        nationality_vocab = Vocabulary()

        for index, row in surname_df.iterrows():
            for char in row.surname:
                char_vocab.add_token(char)
            nationality_vocab.add_token(row.nationality)

        return cls(char_vocab, nationality_vocab)
```

SurnameClassifier 모델

SurnameClassifier 모델은 임베딩층과 ElmanRNN층, Linear 층으로 구성된다.
입력은 SequenceVocabulary에서 정수로 매핑한 토큰이라고 가정

임베딩 층을 사용해 정수를 임베딩 -> RNN으로 시퀀스의 벡터 표현을 계산 , 이 벡터는 성씨에 있는 각 문자에 대한 은닉 상태를 나타냄. 성씨의 마지막 문자에 해당하는 벡터 추출. 이 최종 벡터가 전체 시퀀스 입력을 거쳐 전달된 결과물이라고 할 수 있다. -> 이 요약 벡터를 Linear 층으로 전달해 예측 벡터 계산.

```python
class SurnameClassifier(nn.Module):
    """ RNN으로 특성을 추출하고 MLP로 분류하는 분류 모델 """
    def __init__(self, embedding_size, num_embeddings, num_classes,
                 rnn_hidden_size, batch_first=True, padding_idx=0):
        """
        매개변수:
            embedding_size (int): 문자 임베딩의 크기
            num_embeddings (int): 임베딩할 문자 개수
            num_classes (int): 예측 벡터의 크기
                노트: 국적 개수
            rnn_hidden_size (int): RNN의 은닉 상태 크기
            batch_first (bool): 입력 텐서의 0번째 차원이 \
                                배치인지 시퀀스인지 나타내는 플래그
            padding_idx (int): 텐서 패딩을 위한 인덱스
                torch.nn.Embedding을 참고하세요
        """
        super(SurnameClassifier, self).__init__()

        self.emb = nn.Embedding(num_embeddings=num_embeddings,
                                embedding_dim=embedding_size,
                                padding_idx=padding_idx)
        self.rnn = ElmanRNN(input_size=embedding_size,
                            hidden_size=rnn_hidden_size,
                            batch_first=batch_first)
        self.fc1 = nn.Linear(in_features=rnn_hidden_size,
                             out_features=rnn_hidden_size)
        self.fc2 = nn.Linear(in_features=rnn_hidden_size,
                             out_features=num_classes)

    def forward(self, x_in, x_lengths=None, apply_softmax=False):
        """ 분류기의 정방향 계산

        매개변수:
            x_in (torch.Tensor): 입력 데이터 텐서
                x_in.shape는 (batch, input_dim)입니다
            x_lengths (torch.Tensor): 배치에 있는 각 시퀀스의 길이
                시퀀스의 마지막 벡터를 찾는 데 사용합니다
            apply_softmax (bool): 소프트맥스 활성화 함수를 위한 플래그
                크로스 엔트로피 손실을 사용하려면 False로 지정합니다
        반환값:
            결과 텐서. tensor.shape는 (batch, output_dim)입니다
        """
        x_embedded = self.emb(x_in)
        y_out = self.rnn(x_embedded)

        if x_lengths is not None:
            y_out = column_gather(y_out, x_lengths)
        else:
            y_out = y_out[:, -1, :]

        y_out = F.dropout(y_out, 0.5)
        y_out = F.relu(self.fc1(y_out))
        y_out = F.dropout(y_out, 0.5)
        y_out = self.fc2(y_out)

        if apply_softmax:
            y_out = F.softmax(y_out, dim=1)

        return y_out
```

```python
def column_gather(y_out, x_lengths):
    """ y_out에 있는 각 데이터 포인트에서 마지막 벡터를 추출합니다

    매개변수:
        y_out (torch.FloatTensor, torch.cuda.FloatTensor)
            shape: (batch, sequence, feature)
        x_lengths (torch.LongTensor, torch.cuda.LongTensor)
            shape: (batch,)

    반환값:
        y_out (torch.FloatTensor, torch.cuda.FloatTensor)
            shape: (batch, feature)
    """
    x_lengths = x_lengths.long().detach().cpu().numpy() - 1

    out = []
    for batch_index, column_index in enumerate(x_lengths):
        out.append(y_out[batch_index, column_index])

    return torch.stack(out)
```

모델 훈련과 결과

CrossEntropyLoss() 함수와 정답을 사용해 손실을 계산하고, 손실 값과 옵티마이저로 그레이디언트를 계산하고 이 그레이디언트로 모델의 가중치를 업데이트한다.

이 과정을 훈련 데이터에 있는 모든 배치에 반복한다.
검증 데이터에선 모델을 평가 모드로 설정하여 역전파를 끈다.
전체 과정을 특정 횟수의 에포크 동안 반복한다.

추론

```
def predict_nationality(surname, classifier, vectorizer):
    vectorized_surname, vec_length = vectorizer.vectorize(surname)
    vectorized_surname = torch.tensor(vectorized_surname).unsqueeze(dim=0)
    vec_length = torch.tensor([vec_length], dtype=torch.int64)

    result = classifier(vectorized_surname, vec_length, apply_softmax=True)
    probability_values, indices = result.max(dim=1)

    index = indices.item()
    prob_value = probability_values.item()

    predicted_nationality = vectorizer.nationality_vocab.lookup_index(index)

    return {'nationality': predicted_nationality, 'probability': prob_value, 'surname': surname}
```

```
# surname = input("Enter a surname: ")
classifier = classifier.to("cpu")
for surname in ['McMahan', 'Nakamoto', 'Wan', 'Cho']:
    print(predict_nationality(surname, classifier, vectorizer))
```

```
{'nationality': 'Irish', 'probability': 0.3047773540019989, 'surname': 'McMahan'}
{'nationality': 'Japanese', 'probability': 0.7197886109352112, 'surname': 'Nakamoto'}
{'nationality': 'Vietnamese', 'probability': 0.425054132938385, 'surname': 'Wan'}
{'nationality': 'Chinese', 'probability': 0.3542301058769226, 'surname': 'Cho'}
```

```
# 가장 좋은 모델을 사용해 테스트 세트의 손실과 정확도를 계산합니다
classifier.load_state_dict(torch.load(train_state['model_filename']))

classifier = classifier.to(args.device)
dataset.class_weights = dataset.class_weights.to(args.device)
loss_func = nn.CrossEntropyLoss(dataset.class_weights)

dataset.set_split('test')
batch_generator = generate_batches(dataset,
                                   batch_size=args.batch_size,
                                   device=args.device)
running_loss = 0.
running_acc = 0.
classifier.eval()

for batch_index, batch_dict in enumerate(batch_generator):
    # 출력을 계산합니다
    y_pred =  classifier(batch_dict['x_data'],
                         x_lengths=batch_dict['x_length'])

    # 손실을 계산합니다
    loss = loss_func(y_pred, batch_dict['y_target'])
    loss_t = loss.item()
    running_loss += (loss_t - running_loss) / (batch_index + 1)

    # 정확도를 계산합니다
    acc_t = compute_accuracy(y_pred, batch_dict['y_target'])
    running_acc += (acc_t - running_acc) / (batch_index + 1)

train_state['test_loss'] = running_loss
train_state['test_acc'] = running_acc
```

```
print("테스트 손실: {};".format(train_state['test_loss']))
print("테스트 정확도: {}".format(train_state['test_acc']))
```

```
테스트 손실: 1.8370853233337403;
테스트 정확도: 42.50000000000001
```

요약

- 시퀀스 데이터를 모델링하기 위한 순환 신경망을 소개했다.
- 가장 간단한 RNN인 엘만 RNN을 살펴보았다.
- 시퀀스 모델링의 목표는 시퀀스에 대한 표현(즉, 벡터)을 학습하는 것이다.
- 이 학습된 표현은 문제에 따라 다양한 방법으로 사용될 수 있습니다.
- 성씨 분류 예제를 보며 RNN이 부분 단어 수준에서 정보를 감지할 수 있다는 점을 배웠다.

6장. 자연어 처리를 위한 시퀀스 모델링 - 초급

Thank you