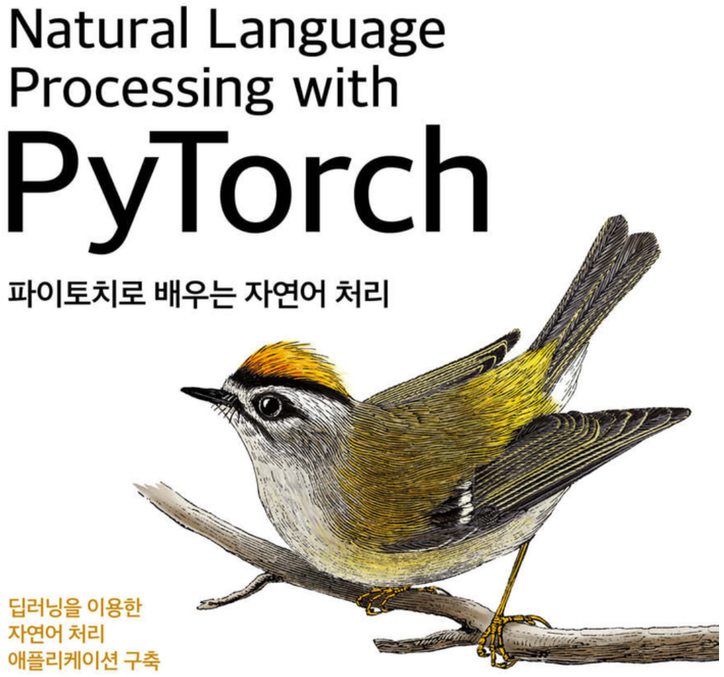


Natural Language Processing with PyTorch
파이토치로 배우는 자연어처리

6장 자연어 처리를 위한 시퀀스 모델링 - 초급

발표자: 이다현

6장 자연어 처리를 위한 시퀀스 모델링 - 초급

목차

6.1 **순환 신경망 소개**

Sequence

- 순서가 있는 항목의 모음집
- 즉, 각 항목의 위치가 중요하다는 뜻
- cat 과 act는 각 항목의 위치(순서)가 다르기 때문에 의미가 달라짐

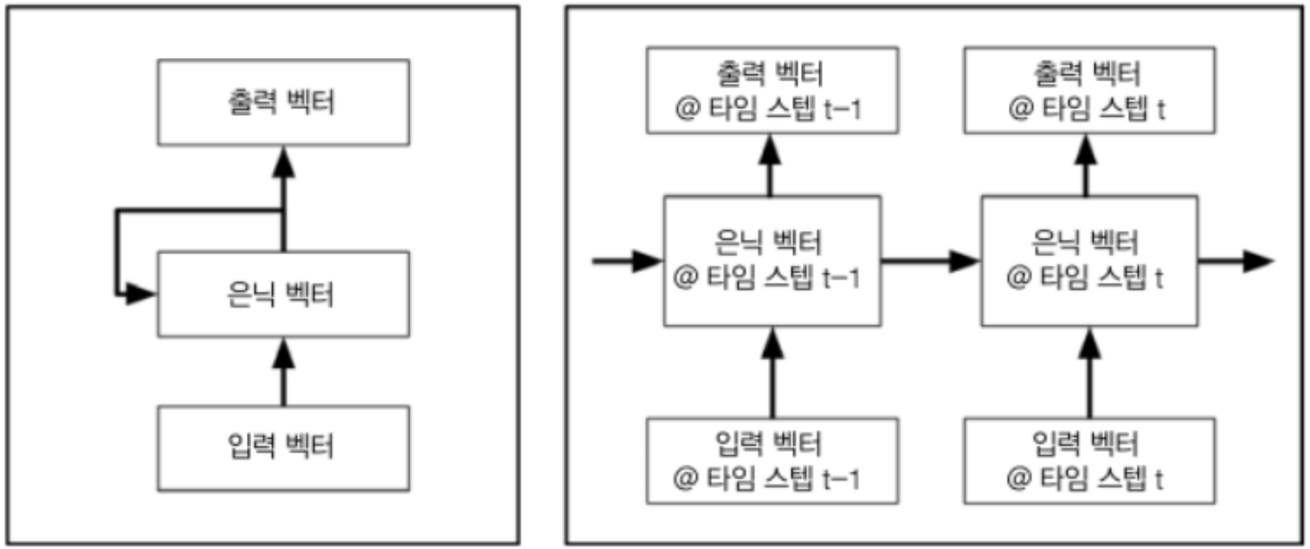
Sequence modeling

- hidden state(은닉 상태)를 유지하는 것과 관련
- 유지한다: 시퀀스의 각 항목을 처리할 때 이전 단계의 정보를 은닉 상태라는 형태로 저장하고 사용
- 은닉 상태: 이전 타임 스텝의 정보를 저장하는 메커니즘
- 기본적인 신경망 시퀀스 모델: Recurrent Nueral Network(RNN, 순환 신경망)

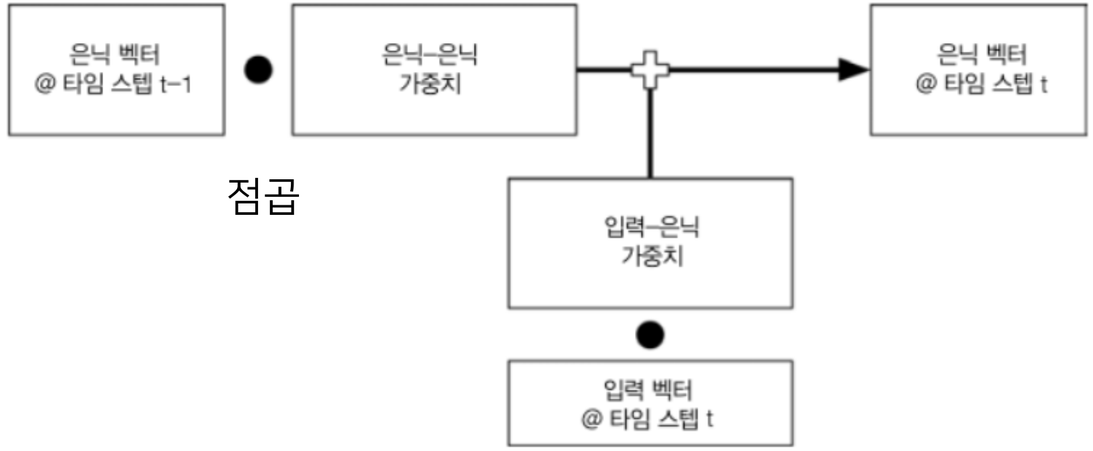
6.1 **순환 신경망 소개**

ElmanRNN

- RNN의 한 종류
- 현재 은닉 “벡터“ = 현재 입력 “벡터” + 이전 은닉 “벡터”
- 이번 챕터에서 엘만 RNN의 예측 대상은 은닉 벡터
- 가변 길이 시퀀스를 다루기 위해 마스킹 기법 사용
- 마스킹: 패딩된 부분이 실제 데이터 처리에 영향을 주지 않도록 하는 기법



파이토치로 배우는 자연어처리 그림 6-1



파이토치로 배우는 자연어처리 그림 6-2

파이토치로 ElmanRNN 구현

```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

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

```python
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._initial_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
```

x_in: (seq_length, batch_size, num_features)
seq_length: 문장에서 단어의 수
batch_size: 이 차원은 동시에 처리되는 데이터 샘플의 수
num_features: 자연어 처리에서 한 단어를 표현하는 임베딩 벡터의 크기

hiddens: 계산된 은닉 상태 텐서

파라미터 공유

CNN의 파라미터 공유

- 공간에 따라 파라미터(커널)를 공유
- 이동불변성을 학습
- 이동불변성: 감지하려는 특성이 입력 데이터의 어느 위치에 있든 잡아낼 수 있다

RNN의 파라미터 공유

- 타임 스텝(시간)에 따라 파라미터(은닉-은닉 및 입력-은닉 가중치)를 공유
- 시퀀스 불변성을 학습

결론

- RNN과 CNN 모두 파라미터 공유를 통해 모델의 학습 효율성과 일반화 능력을 향상시킴
- 모델이 학습 데이터에만 국한되지 않고 새로운 시퀀스에서도 유사한 패턴을 일관되게 인식

6.2 예제: 문자 RNN으로 성씨 국적 분류하기

SurnameDataset

```
    @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 = ₩
            self._vectorizer.nationality_vocab.lookup_token(row.nationality)

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

벡터로 변환된 성씨 + 국적을 나타내는 정수를 반환
추가로 시퀀스 길이를 반환

6.2 예제: 문자 RNN으로 성씨 국적 분류하기

SurnameVectorizer

```
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)
```

vectorize 함수: 성씨(surname) 문자열을 숫자 인덱스의 벡터로 변환하는 과정을 수행

- 변환된 인덱스 벡터(out_vector)와 실제 성씨의 길이(len(indices))가 반환

6.2 예제: 문자 RNN으로 성씨 국적 분류하기

SurnameClassifier

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

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

    return y_out
```

```
def column_gather(y_out, x_lengths):
    ''' y_out에 있는 각 데이터 포인트에서 마지막 벡터 추출합니다

    조금 더 구체적으로 말하면 배치 행 인덱스를 순회하면서
    x_lengths에 있는 값에 해당하는 인덱스 위치의 벡터를 반환합니다.

    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)
```

임베딩 층을 사용해 정수(SequenceVocabulary에서 토큰을 정수로 매핑) 임베딩
--> RNN으로 시퀀스의 벡터 표현 계산
--> 성씨에 마지막 문자에 해당하는 벡터 추출
--> 요약 벡터를 Linear 층(MLP)로 전달해 예측 벡터 계산

6.2 예제: 문자 RNN으로 성씨 국적 분류하기

모델 훈련 결과

In [13]:
```
# 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
```

In [14]:
```
print("테스트 손실: {};".format(train_state['test_loss']))
print("테스트 정확도: {}".format(train_state['test_acc']))
```

```
테스트 손실: 1.8372113943099972;
테스트 정확도: 42.50000000000001
```

CrossEntropyLoss()함수를 사용해 손실 계산
--> 손실 값과 옵티마이저로 그레디언트 계산
--> 그레디언트로 모델의 가중치 업데이트

*검증 데이터에서는 모델을 평가모드로 설정
-> 편향되지 않은 모델의 성능을 얻는 목적으로만 사용

In [15]:
```
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}
```

In [16]:
```
# 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': 0.3048343360424042, 'surname': 'McMahan'}
{'nationality': 'Japanese', 'probability': 0.7194132804870605, 'surname': 'Nakamoto'}
{'nationality': 'Vietnamese', 'probability': 0.4251473546028137, 'surname': 'Wan'}
{'nationality': 'Chinese', 'probability': 0.3545195758342743, 'surname': 'Cho'}
```

6.3 **요약**

- 시퀀스 데이터 타입과 시퀀스 모델링에 대해 소개
- 시퀀스 모델인 RNN와 가장 간단한 RNN인 엘만 RNN에 대해 살펴봄
- RNN과 CNN의 파라미터 공유 개념과 장점에 대해 소개함
- 시퀀스 모델링의 목표는 시퀀스에 대한 표현(즉, 벡터)를 학습하는 것
- 성씨 분류 예제를 통해 RNN이 부분 단어 수준에서 정보를 감지할 수 있다는 점을 확인함