자연어 처리를 위한 시퀀스 모델링 - 고급

2021.12.03

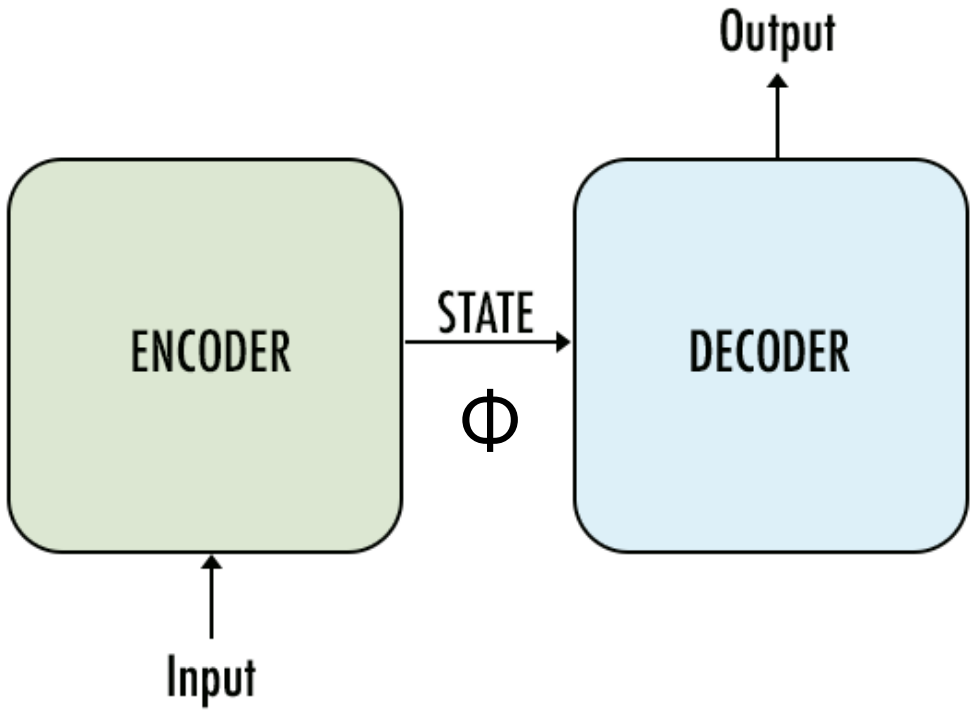
최혜원

이 장에서 다룰 개념

- 시퀀스-투-시퀀스 모델링
- 양방향 모델
- 어텐션 메커니즘
- 신경망 기계 번역 구현 과정

8.1 시퀀스-투-시퀀스 모델, 인코더-디코더 모델, 조건부 생성

- S2S모델 : 인코더-디코더 모델이라는 신경망 모델의 일종.



8.1 시퀀스-투-시퀀스 모델, 인코더-디코더 모델, 조건부 생성

- 조건부 생성 모델 : 입력표현 x 대신 일반적인 조건문맥 C를 사용해 디코더가 출력을 생성

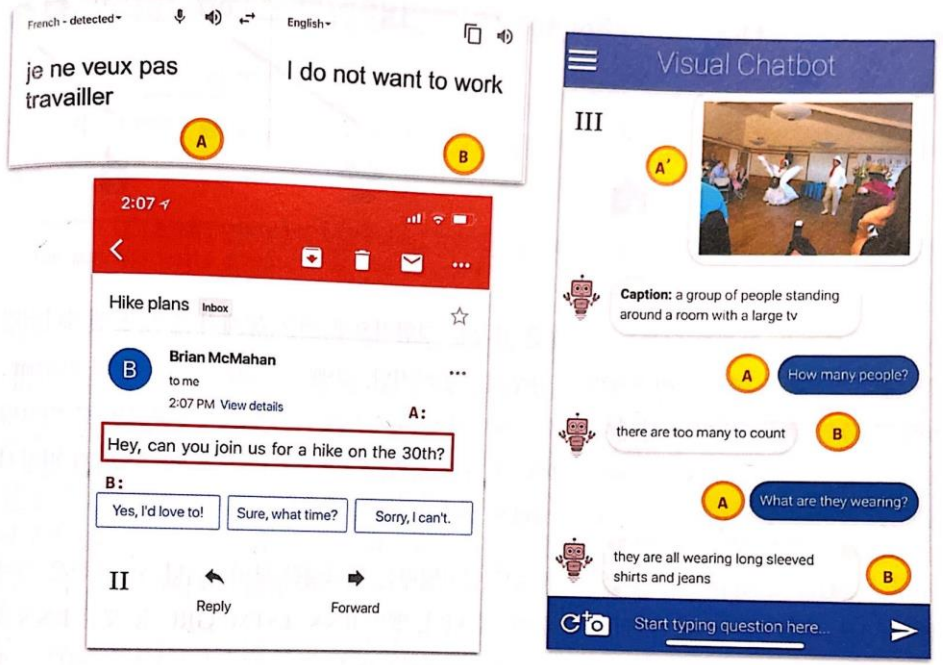


그림 8-2 인코더–디코더 모델을 사용해 문제를 해결하는 작업들. 기계 번역(왼쪽 위: 입력 A는 프랑스어 문장이고 출력 B는 영어 문장입니다), 이메일 응답 추천(왼쪽 아래: 입력 A는 이메일 텍스트이고 출력 B는 가능한 답변 중 하나입니다)이 있습니다. 오른쪽의 예는 더 복잡합니다. 챗봇의 입력 이미지(A')에 관한 A의 질문에 답변을 합니다. A와 A'의 인코딩이 응답(B)의 생성 조건입니다. 이런 작업이 모두 조건부 생성 작업입니다.

8.1 시퀀스-투-시퀀스 모델, 인코더-디코더 모델, 조건부 생성

- 정렬(alignment) : 입력과 출력 사이의 매핑

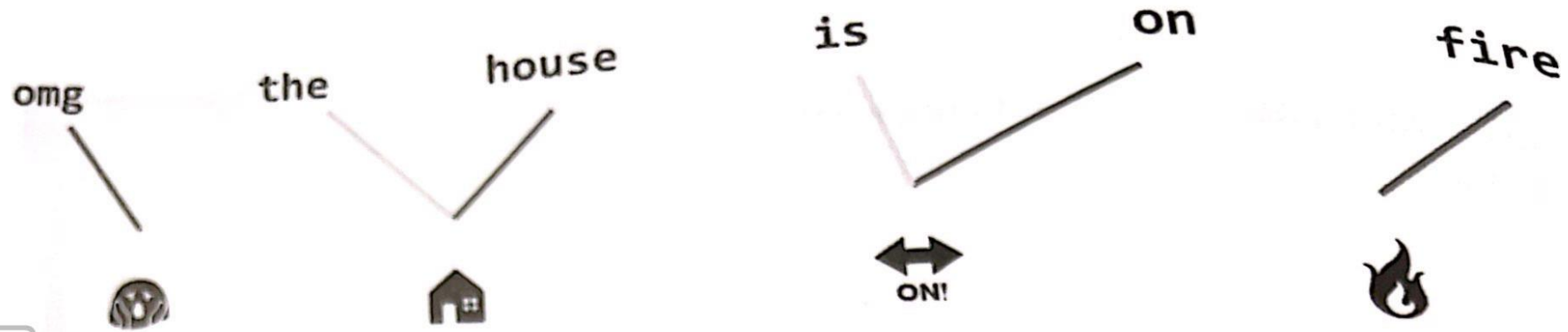


그림 8-3 325 예측 단계인 이모지 번역. 두 시퀀스에 있는 토큰 사이의 정렬은 번역 대상을 나타냅니다.

8.1 시퀀스-투-시퀀스 모델, 인코더-디코더 모델, 조건부 생성

- 인코더 – 디코더 모형

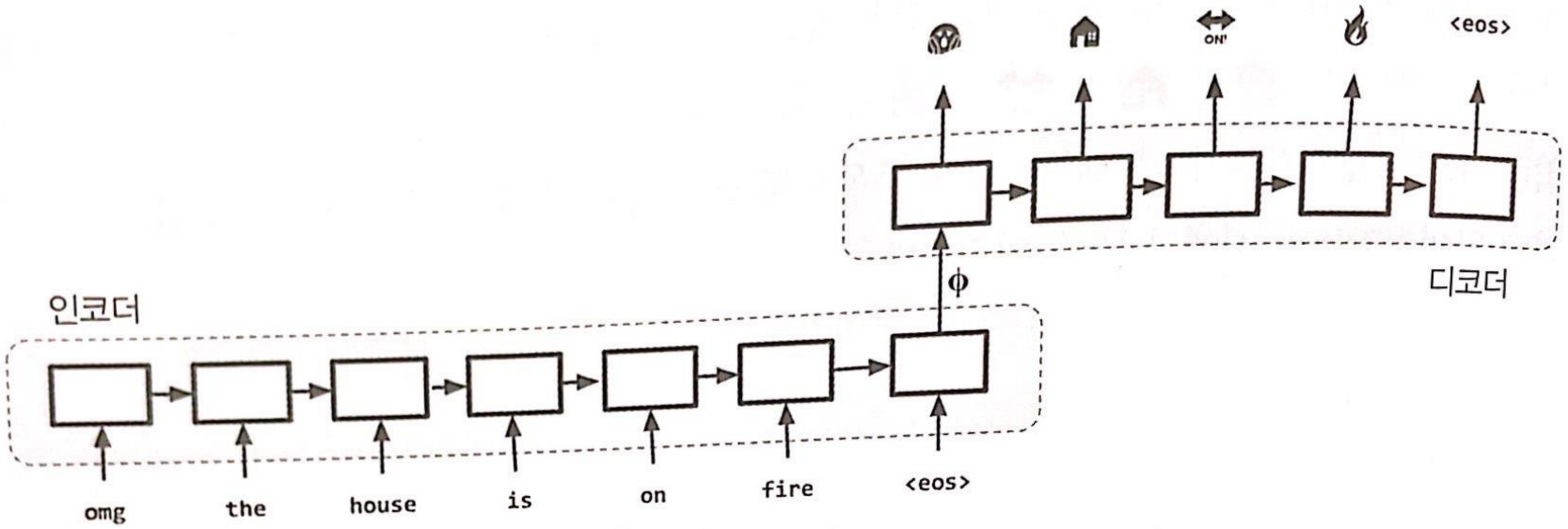


그림 8-4 영어를 이모지로 번역하는 S2S 모델

8.2 강력한 시퀀스 모델링 : 양방향 순환 모델

- 정방향 표현과 역방향 표현을 연결해서 단어의 최종 표현을 만든다.

"The man who hunts ducks out on the weekends"

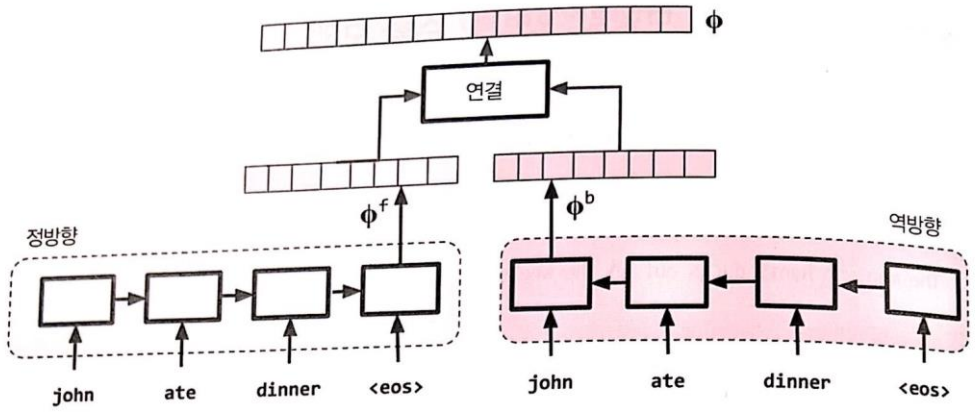


그림 8-5 시퀀스 분류를 위한 양방향 RNN 모델

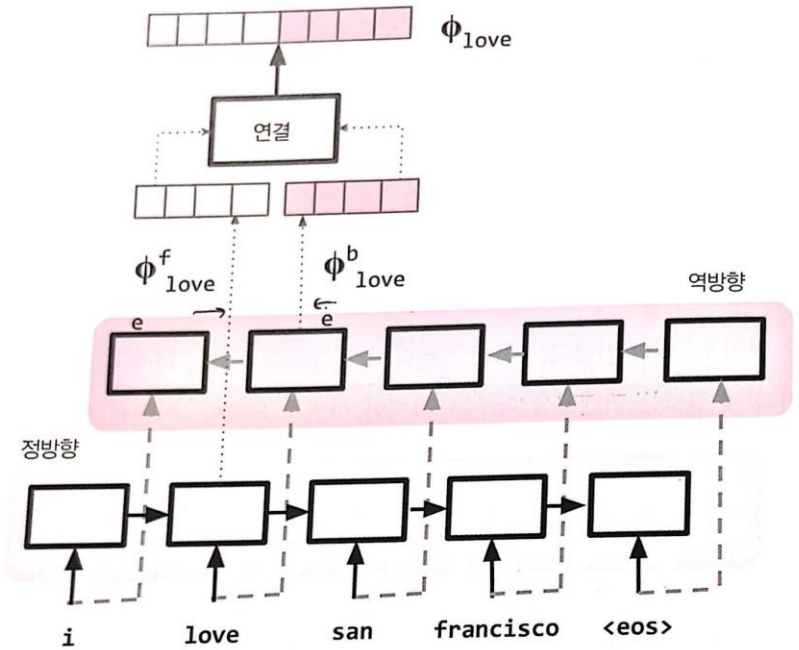


그림 8-6 시퀀스 레이블링을 위한 양방향 RNN 모델

8.3 강력한 시퀀스 모델링 : 어텐션

- 8.1절의 S2S모델의 문제점 : 전체 입력 문장을 하나의 벡터 Φ에 밀어 넣는다.
  -> 역전파시에 그래디언트 소실 : 훈련이 어렵
  -> 긴 문장에서는 입력 정보를 감지하지 못함

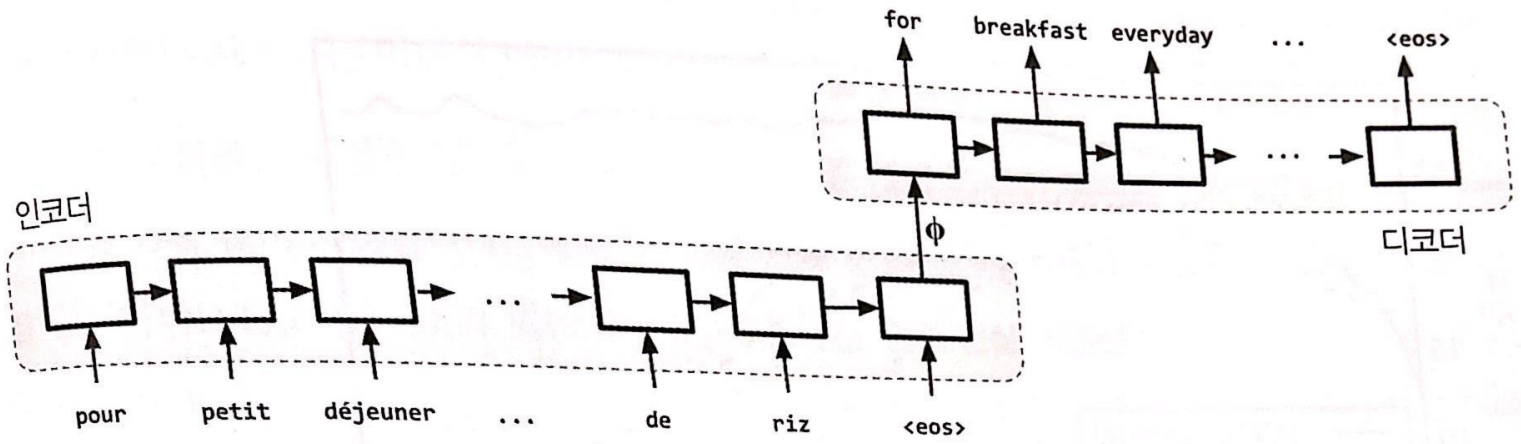


그림 3-7 인코더-디코더 모델을 사용해 긴 프랑스어 문장을 영어로 번역하기. 최종 표현 ϕ는 입력에서 넓은 범위의 의 [illegible]을 감지하지 못하고 훈련을 어렵게 합니다.

8.3 강력한 시퀀스 모델링 : 어텐션

- 해결책 : 어텐션(Attention) – 출력을 생성할 때 관련된 입력 부분에 초점을 맞춤.

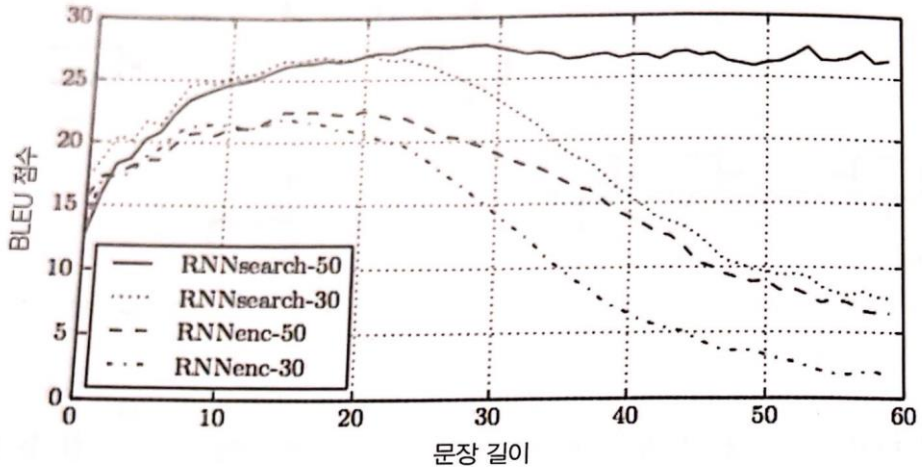


그림 8-8 어텐션이 필요한 이유. 이 그래프는 어텐션을 사용할 때(RNNsearch–30, RNNsearch–50)와 그렇지 않을 때(RNNenc–30, RNNenc–50) 기계 번역 시스템의 BLEU 점수 변화를 보여줍니다. RNN*–30와 RNN*–50 시스템은 각각 최대 단어 길이가 30과 50인 문장에서 훈련했습니다. 어텐션이 없는 기계 번역 시스템은 문장 길이가 길수록 성능이 감소합니다. 어텐션을 사용하면 긴 문장의 번역 성능이 향상되지만, 기계 번역 성능의 안정성은 훈련된 문장의 길이에 비례합니다. (출처: 바다나우 등의 2015년 논문)

8.3.1 심층 신경망의 어텐션(관련 용어 정리)

- Value, Key : 인코더의 최종 혹은 중간 타임스텝의 은닉상태
- Query : 디코더의 이전 은닉 상태
- 어텐션 벡터, 어텐션 가중치, 정렬 : 주의를 기울이려는 값을 개수와 차원이 같은 벡터
- 문맥벡터(글림스) : 어텐션 가중치와 인코더 상태 값이 연결된 벡터
- 호환성 함수 : 어텐션 벡터가 이것을 이용해 업데이트 된다.
- 콘텐츠 인식 어텐션
- 위치 인식 어텐션 : 쿼리벡터와 키만 사용
- 소프트 어텐션 : 어텐션 가중치를 0과 1 사이의 실수로 둠
- 하드 어텐션 : 0 혹은 1인 이진 벡터를 학습
- 전역 어텐션 : 입력의 모든 타임 스텝에 대해 인코더의 상태를 사용
- 지역 어텐션 : 현재 타임 스텝 주위에 있는 입력에만 의존
- 지도 어텐션 : 동시에 훈련되는 별도의 신경망을 사용해 어텐션 함수를 학습
- 멀티헤드 어텐션 : 여러 어텐션 벡터를 사용해서 입력의 다양한 영역을 추적
- 셀프 어텐션 : 입력의 어떤 영역이 다른 영역에 영향을 미치는지 학습하는 메커니즘
- 멀티모달 어텐션 : 입력의 형태가 이미지와 음성처럼 다양할 때

8.4 시퀀스 생성 모델 평가

- 참조 출력 : 여러 모델을 비교할 때 얼마나 참조출력에 가까운지를 점수로 매김. (정답이 여럿인 모델이기 때문)
- 1) 사람평가 : 사람들간의 평가가 다를 수 있기 때문에 평가자 간의 일치율(HTER- 추가, 삭제, 이동한 횟수를 헤아려 계산한 가중치가 적용된 편집 거리)을 사용.
- 2) 자동 평가 : n-그램 중복 기반 지표, 혼란

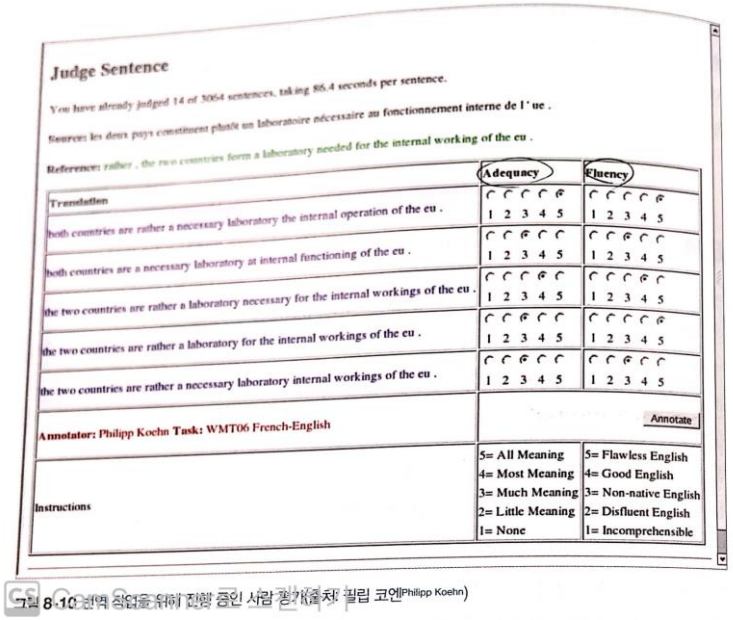


그림 8-10 번역 작업을 위해 진행 중인 사람 평가(출처: 필립 코엔Philipp Koehn)

8.4 시퀀스 생성 모델 평가

- 2) 자동 평가 :

n-그램 중복 기반 지표 - BLEU, ROUGE, METEOR

혼란도 – 시퀀스 x의 확률이 P(x)일 때

$$\text{Perplexity}(x) = 2^{-P(x)\log P(x)}$$

But, 혼란도는 세 가지 문제가 있다.

1. 공식에 지수함수가 있어서 과장된 지표이다.
2. 모델의 오차율에 직접 반영되지 않는다.
3. 혼란도가 향상되더라도 사람이 판단하기에는 향상되었다고 느끼지 못할 수 있다.

8.5 예제: 신경망 기계 번역(데이터셋)

- 데이터셋 : 타토에바 프로젝트(영어 – 프랑스어)
- 전처리 :
  - 모든 문장을 소문자로
  - NLTK의 영어, 프랑스어 토큰화를 각 문장 쌍에 적용.
  - NLTK의 언어에 특화된 단어 토큰화를 적용해 토큰 리스트 생성
  - 데이터의 범위를 좁힘(' I am’, ‘he is’, ‘she is’, 'they are’, ‘you are’, ‘we are’로 시작하는 영어문장) -> 135842에서 13062로 10배나 줄임.
  - train(70%), valid(15%), test(15%)로 나눔.

8.5 예제: 신경망 기계 번역

```python
class NMTVectorizer(object):
    """ 어휘 사전을 생성하고 관리합니다 """
    def __init__(self, source_vocab, target_vocab, max_source_length, max_target_length):
        """
        매개변수:
            source_vocab (SequenceVocabulary): 소스 단어를 정수에 매핑합니다
            target_vocab (SequenceVocabulary): 타깃 단어를 정수에 매핑합니다
            max_source_length (int): 소스 데이터셋에서 가장 긴 시퀀스 길이
            max_target_length (int): 타깃 데이터셋에서 가장 긴 시퀀스 길이
        """
        self.source_vocab = source_vocab
        self.target_vocab = target_vocab

        self.max_source_length = max_source_length
        self.max_target_length = max_target_length


    def _vectorize(self, indices, vector_length=-1, mask_index=0):
        """인덱스를 벡터로 변환합니다

        매개변수:
            indices (list): 시퀀스를 나타내는 정수 리스트
            vector_length (int): 인덱스 벡터의 길이
            mask_index (int): 사용할 마스크 인덱스; 거의 항상 0
        """
        if vector_length < 0:
            vector_length = len(indices)

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

        return vector
```

```python
def from_dataframe(cls, bitext_df):
    """ 데이터셋 데이터프레임으로 NMTVectorizer를 초기화합니다

    매개변수:
        bitext_df (pandas.DataFrame): 텍스트 데이터셋
    반환값
    :
        NMTVectorizer 객체
    """
    source_vocab = SequenceVocabulary()
    target_vocab = SequenceVocabulary()

    max_source_length = 0
    max_target_length = 0

    for _, row in bitext_df.iterrows():
        source_tokens = row["source_language"].split(" ")
        if len(source_tokens) > max_source_length:
            max_source_length = len(source_tokens)
        for token in source_tokens:
            source_vocab.add_token(token)

        target_tokens = row["target_language"].split(" ")
        if len(target_tokens) > max_target_length:
            max_target_length = len(target_tokens)
        for token in target_tokens:
            target_vocab.add_token(token)

    return cls(source_vocab, target_vocab, max_source_length, max_target_length)
```

8.5 예제: 신경망 기계 번역

```python
def vectorize(self, source_text, target_text, use_dataset_max_lengths=True):
    """ 벡터화된 소스 텍스트와 타깃 텍스트를 반환합니다

    벡터화된 소스 텍스트는 하나의 벡터입니다.
    벡터화된 타깃 텍스트는 7장의 성씨 모델링과 비슷한 스타일로 두 개의 벡터로 나뉩니다.
    각 타임 스텝에서 첫 번째 벡터가 샘플이고 두 번째 벡터가 타깃이 됩니다.

    매개변수:
        source_text (str): 소스 언어의 텍스트
        target_text (str): 타깃 언어의 텍스트
        use_dataset_max_lengths (bool): 최대 벡터 길이를 사용할지 여부
    반환값:
        다음과 같은 키에 벡터화된 데이터를 담은 딕셔너리:
            source_vector, target_x_vector, target_y_vector, source_length
    """
    source_vector_length = -1
    target_vector_length = -1

    if use_dataset_max_lengths:
        source_vector_length = self.max_source_length + 2
        target_vector_length = self.max_target_length + 1

    source_indices = self._get_source_indices(source_text)
    source_vector = self._vectorize(source_indices,
                                    vector_length=source_vector_length,
                                    mask_index=self.source_vocab.mask_index)

    target_x_indices, target_y_indices = self._get_target_indices(target_text)
    target_x_vector = self._vectorize(target_x_indices,
                                      vector_length=target_vector_length,
                                      mask_index=self.target_vocab.mask_index)
    target_y_vector = self._vectorize(target_y_indices,
                                      vector_length=target_vector_length,
                                      mask_index=self.target_vocab.mask_index)
    return {"source_vector": source_vector,
            "target_x_vector": target_x_vector,
            "target_y_vector": target_y_vector,
            "source_length": len(source_indices)}
```

```python
def _get_source_indices(self, text):
    """ 벡터로 변환된 소스 텍스트를 반환합니다

    매개변수:
        text (str): 소스 텍스트; 토큰은 공백으로 구분되어야 합니다
    반환값:
        indices (list): 텍스트를 표현하는 정수 리스트
    """
    indices = [self.source_vocab.begin_seq_index]
    indices.extend(self.source_vocab.lookup_token(token) for token in text.split(" "))
    indices.append(self.source_vocab.end_seq_index)
    return indices

def _get_target_indices(self, text):
    """ 벡터로 변환된 타깃 텍스트를 반환합니다

    매개변수:
        text (str): 타깃 텍스트; 토큰은 공백으로 구분되어야 합니다
    반환값:
        튜플: (x_indices, y_indices)
            x_indices (list): 디코더에서 샘플을 나타내는 정수 리스트
            y_indices (list): 디코더에서 예측을 나타내는 정수 리스트
    """
    indices = [self.target_vocab.lookup_token(token) for token in text.split(" ")]
    x_indices = [self.target_vocab.begin_seq_index] + indices
    y_indices = indices + [self.target_vocab.end_seq_index]
    return x_indices, y_indices
```

8.5 예제: 신경망 기계 번역

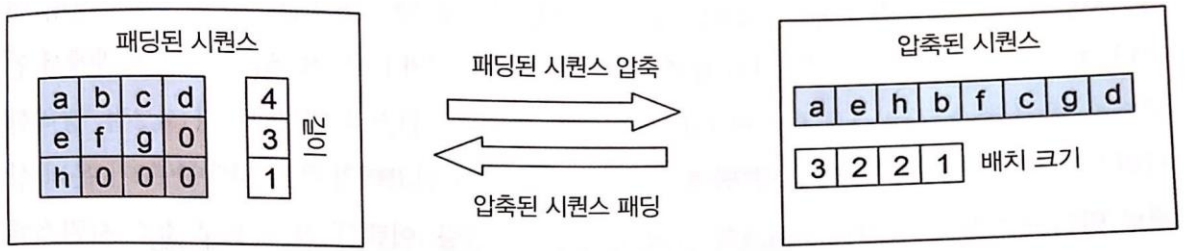


그림 8-11 패딩된 시퀀스의 행렬과 길이가 왼쪽에 나타나 있습니다. 패딩된 행렬이 가변 길이 시퀀스를 표현하는 기본적인 방법입니다. 오른쪽에 0으로 패딩하고 행 벡터를 쌓습니다. 파이토치에서 패딩된 시퀀스 행렬을 하나의 텐서 표현인 PackedSequence로 압축할 수 있으며 배치 크기와 함께 오른쪽에 나타나 있습니다. 이런 표현을 사용하면 GPU가 타임 스텝마다 얼마나 많은 시퀀스(배치 크기)가 있는지 파악하면서 시퀀스를 처리할 수 있습니다.


```
def generate_nmt_batches(dataset, batch_size, shuffle=True,
                         drop_last=True, device="cpu"):
    """ 파이토치 DataLoader를 감싸고 있는 제너레이터 함수: NMT 버전 """
    dataloader = DataLoader(dataset=dataset, batch_size=batch_size,
                            shuffle=shuffle, drop_last=drop_last)

    for data_dict in dataloader:
        lengths = data_dict['x_source_length'].numpy()
        sorted_length_indices = lengths.argsort()[::-1].tolist()

        out_data_dict = {}
        for name, tensor in data_dict.items():
            out_data_dict[name] = data_dict[name][sorted_length_indices].to(device)
        yield out_data_dict
```

8.5 예제: 신경망 기계 번역

```python
class NMTModel(nn.Module):
    """ 신경망 기계 번역 모델 """
    def __init__(self, source_vocab_size, source_embedding_size,
                 target_vocab_size, target_embedding_size, encoding_size,
                 target_bos_index):
        """
        매개변수:
            source_vocab_size (int): 소스 언어에 있는 고유한 단어 개수
            source_embedding_size (int): 소스 임베딩 벡터의 크기
            target_vocab_size (int): 타깃 언어에 있는 고유한 단어 개수
            target_embedding_size (int): 타깃 임베딩 벡터의 크기
            encoding_size (int): 인코더 RNN의 크기
            target_bos_index (int): BEGIN-OF-SEQUENCE 토큰 인덱스
        """
        super(NMTModel, self).__init__()
        self.encoder = NMTEncoder(num_embeddings=source_vocab_size,
                                  embedding_size=source_embedding_size,
                                  rnn_hidden_size=encoding_size)
        decoding_size = encoding_size * 2
        self.decoder = NMTDecoder(num_embeddings=target_vocab_size,
                                  embedding_size=target_embedding_size,
                                  rnn_hidden_size=decoding_size,
                                  bos_index=target_bos_index)

    def forward(self, x_source, x_source_lengths, target_sequence, sample_probability=0.0):
        """ 모델의 정방향 계산

        매개변수:
            x_source (torch.Tensor): 소스 텍스트 데이터 텐서
                x_source.shape는 (batch, vectorizer.max_source_length)입니다.
            x_source_lengths torch.Tensor): x_source에 있는 시퀀스 길이
            target_sequence (torch.Tensor): 타깃 텍스트 데이터 텐서
            sample_probability (float): 스케줄링된 샘플링 파라미터
                디코더 타임 스텝마다 모델 예측에 사용할 확률
        반환값:
            decoded_states (torch.Tensor): 각 출력 타임 스텝의 예측 벡터
        """
        encoder_state, final_hidden_states = self.encoder(x_source, x_source_lengths)
        decoded_states = self.decoder(encoder_state=encoder_state,
                                      initial_hidden_state=final_hidden_states,
                                      target_sequence=target_sequence,
                                      sample_probability=sample_probability)
        return decoded_states
```

```python
class NMTEncoder(nn.Module):
    def __init__(self, num_embeddings, embedding_size, rnn_hidden_size):
        """
        매개변수:
            num_embeddings (int): 임베딩 개수는 소스 어휘 사전의 크기입니다
            embedding_size (int): 임베딩 벡터의 크기
            rnn_hidden_size (int): RNN 은닉 상태 벡터의 크기
        """
        super(NMTEncoder, self).__init__()

        self.source_embedding = nn.Embedding(num_embeddings, embedding_size, padding_idx=0)
        self.birnn = nn.GRU(embedding_size, rnn_hidden_size, bidirectional=True, batch_first=True)

    def forward(self, x_source, x_lengths):
        """ 모델의 정방향 계산

        매개변수:
            x_source (torch.Tensor): 입력 데이터 텐서
                x_source.shape는 (batch, seq_size)이다.
            x_lengths (torch.Tensor): 배치에 있는 아이템의 길이 벡터
        반환값:
            튜플: x_unpacked (torch.Tensor), x_birnn_h (torch.Tensor)
                x_unpacked.shape = (batch, seq_size, rnn_hidden_size * 2)
                x_birnn_h.shape = (batch, rnn_hidden_size * 2)
        """
        x_embedded = self.source_embedding(x_source)
        # PackedSequence 생성; x_packed.data.shape=(number_items, embeddign_size)
        x_packed = pack_padded_sequence(x_embedded, x_lengths.detach().cpu().numpy(),
                                        batch_first=True)

        # x_birnn_h.shape = (num_rnn, batch_size, feature_size)
        x_birnn_out, x_birnn_h  = self.birnn(x_packed)
        # (batch_size, num_rnn, feature_size)로 변환
        x_birnn_h = x_birnn_h.permute(1, 0, 2)

        # 특성 펼침; (batch_size, num_rnn * feature_size)로 바꾸기
        # (참고: -1은 남은 차원에 해당합니다,
        #       두 개의 RNN 은닉 벡터를 1로 펼칩니다)
        x_birnn_h = x_birnn_h.contiguous().view(x_birnn_h.size(0), -1)

        x_unpacked, _ = pad_packed_sequence(x_birnn_out, batch_first=True)

        return x_unpacked, x_birnn_h
```

8.5 예제: 신경망 기계 번역

```python
class NMTDecoder(nn.Module):
    def __init__(self, num_embeddings, embedding_size, rnn_hidden_size, bos_index):
        """
        매개변수:
            num_embeddings (int): 임베딩 개수는 타깃 어휘 사전에 있는 고유한 단어의 개수이다
            embedding_size (int): 임베딩 벡터 크기
            rnn_hidden_size (int): RNN 은닉 상태 크기
            bos_index(int): begin-of-sequence 인덱스
        """
        super(NMTDecoder, self).__init__()
        self._rnn_hidden_size = rnn_hidden_size
        self.target_embedding = nn.Embedding(num_embeddings=num_embeddings,
                                             embedding_dim=embedding_size,
                                             padding_idx=0)
        self.gru_cell = nn.GRUCell(embedding_size + rnn_hidden_size,
                                   rnn_hidden_size)
        self.hidden_map = nn.Linear(rnn_hidden_size, rnn_hidden_size)
        self.classifier = nn.Linear(rnn_hidden_size * 2, num_embeddings)
        self.bos_index = bos_index
        self._sampling_temperature = 3

    def _init_indices(self, batch_size):
        """ BEGIN-OF-SEQUENCE 인덱스 벡터를 반환합니다 """
        return torch.ones(batch_size, dtype=torch.int64) * self.bos_index

    def _init_context_vectors(self, batch_size):
        """ 문맥 벡터를 초기화하기 위한 0 벡터를 반환합니다 """
        return torch.zeros(batch_size, self._rnn_hidden_size)
```

```python
def forward(self, encoder_state, initial_hidden_state, target_sequence, sample_probability=0.0):
    """ 모델의 정방향 계산

    매개변수:
        encoder_state (torch.Tensor): NMTEncoder의 출력
        initial_hidden_state (torch.Tensor): NMTEncoder의 마지막 은닉 상태
        target_sequence (torch.Tensor): 타깃 텍스트 데이터 텐서
        sample_probability (float): 스케줄링된 샘플링 파라미터
            디코더 타임 스텝마다 모델 예측에 사용할 확률
    반환값:
        output_vectors (torch.Tensor): 각 타임 스텝의 예측 벡터
    """
    if target_sequence is None:
        sample_probability = 1.0
    else:
        # 가정: 첫 번째 차원은 배치 차원입니다
        # 즉 입력은 (Batch, Seq)
        # 시퀀스에 대해 반복해야 하므로 (Seq, Batch)로 차원을 바꿉니다
        target_sequence = target_sequence.permute(1, 0)
        output_sequence_size = target_sequence.size(0)

    # 주어진 인코더의 은닉 상태를 초기 은닉 상태로 사용합니다
    h_t = self.hidden_map(initial_hidden_state)

    batch_size = encoder_state.size(0)
    # 문맥 벡터를 0으로 초기화합니다
    context_vectors = self._init_context_vectors(batch_size)
    # 첫 단어 y_t를 BOS로 초기화합니다
    y_t_index = self._init_indices(batch_size)

    h_t = h_t.to(encoder_state.device)
    y_t_index = y_t_index.to(encoder_state.device)
    context_vectors = context_vectors.to(encoder_state.device)

    output_vectors = []
    self._cached_p_attn = []
    self._cached_ht = []
    self._cached_decoder_state = encoder_state.cpu().detach().numpy()
```

```python
    for i in range(output_sequence_size):
        # 스케줄링된 샘플링 사용 여부
        use_sample = np.random.random() < sample_probability
        if not use_sample:
            y_t_index = target_sequence[i]

        # 단계 1: 단어를 임베딩하고 이전 문맥과 연결합니다
        y_input_vector = self.target_embedding(y_t_index)
        rnn_input = torch.cat([y_input_vector, context_vectors], dim=1)

        # 단계 2: GRU를 적용하고 새로운 은닉 벡터를 얻습니다
        h_t = self.gru_cell(rnn_input, h_t)
        self._cached_ht.append(h_t.cpu().detach().numpy())

        # 단계 3: 현재 은닉 상태를 사용해 인코더의 상태를 주목합니다
        context_vectors, p_attn, _ = verbose_attention(encoder_state_vectors=encoder_state,
                                                       query_vector=h_t)

        # 부가 작업: 시각화를 위해 어텐션 확률을 저장합니다
        self._cached_p_attn.append(p_attn.cpu().detach().numpy())

        # 단계 4: 현재 은닉 상태와 문맥 벡터를 사용해 다음 단어를 예측합니다
        prediction_vector = torch.cat((context_vectors, h_t), dim=1)
        score_for_y_t_index = self.classifier(F.dropout(prediction_vector, 0.3))

        if use_sample:
            p_y_t_index = F.softmax(score_for_y_t_index * self._sampling_temperature, dim=1)
            # _, y_t_index = torch.max(p_y_t_index, 1)
            y_t_index = torch.multinomial(p_y_t_index, 1).squeeze()

        # 부가 작업: 예측 성능 점수를 기록합니다
        output_vectors.append(score_for_y_t_index)

    output_vectors = torch.stack(output_vectors).permute(1, 0, 2)

    return output_vectors
```

8.5 예제: 신경망 기계 번역_어텐션

```
45  def verbose_attention(encoder_state_vectors, query_vector):
46      """ 원소별 연산을 사용하는 어텐션 메커니즘 버전
47
48      매개변수:
49          encoder_state_vectors (torch.Tensor): 인코더의 양방향 GRU에서 출력된 3차원 텐서
50          query_vector (torch.Tensor): 디코더 GRU의 은닉 상태
51      """
52      batch_size, num_vectors, vector_size = encoder_state_vectors.size()
53      vector_scores = torch.sum(encoder_state_vectors * query_vector.view(batch_size, 1, vector_size),
54                                dim=2)
55      vector_probabilities = F.softmax(vector_scores, dim=1)
56      weighted_vectors = encoder_state_vectors * vector_probabilities.view(batch_size, num_vectors, 1)
57      context_vectors = torch.sum(weighted_vectors, dim=1)
58      return context_vectors, vector_probabilities, vector_scores
59
60  def terse_attention(encoder_state_vectors, query_vector):
61      """ 점곱을 사용하는 어텐션 메커니즘 버전
62
63      매개변수:
64          encoder_state_vectors (torch.Tensor): 인코더의 양방향 GRU에서 출력된 3차원 텐서
65          query_vector (torch.Tensor): 디코더 GRU의 은닉 상태
66      """
67      vector_scores = torch.matmul(encoder_state_vectors, query_vector.unsqueeze(dim=2)).squeeze()
68      vector_probabilities = F.softmax(vector_scores, dim=-1)
69      context_vectors = torch.matmul(encoder_state_vectors.transpose(-2, -1),
70                                     vector_probabilities.unsqueeze(dim=2)).squeeze()
71      return context_vectors, vector_probabilities
72
```

8.6 요약

- 조건부 생성 모델이라는 문맥이 조건으로 주어졌을 때 출력 시퀀스를 만드는 데 초점.
- 조건 문맥이 다른 시퀀스에서 유도될 때 시퀀스 투 시퀀스(S2S)
- 양방향 모델
- 어텐션 메커니즘
- 엔드투엔드 기계번역 예제