밑바닥부터 시작하는 딥러닝2

CHAPTER 7, 8. RNN을 사용한 문장 생성(어텐션)

20180376 안제준

7-1 언어 모델을 사용한 문장 생성

1. RNN을 사용한 문장 생성의 순서

그림 7-2 언어 모델은 다음에 출현할 단어의 확률분포를 출력한다.

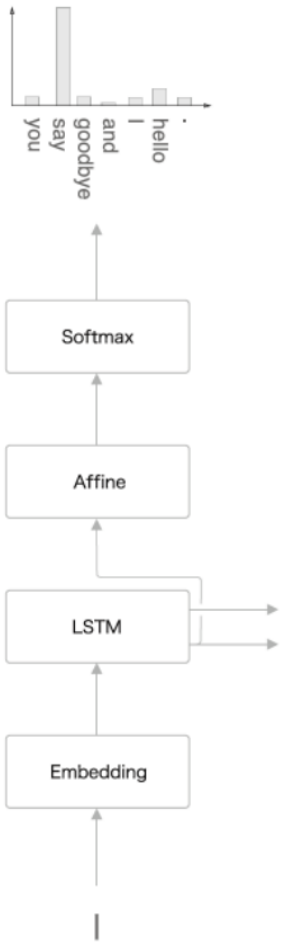
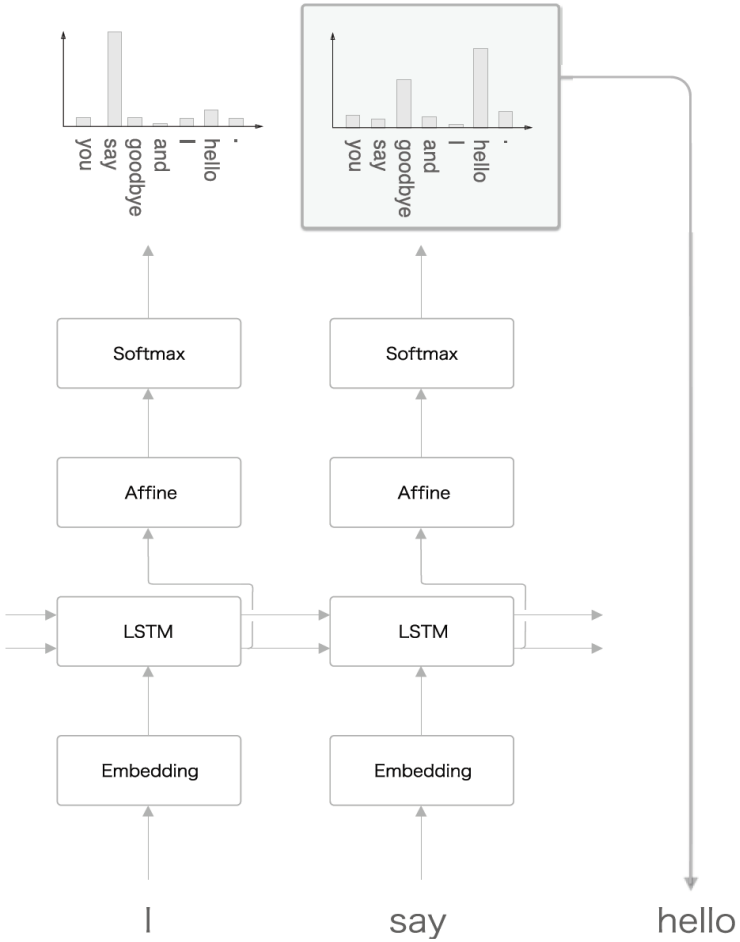


그림 7-4 확률분포 출력과 샘플링을 반복한다.



7-1 언어 모델을 사용한 문장 생성

2. 문장 생성 구현

```
import sys
sys.path.append('..')
import numpy as np
from common.functions import softmax
from rnnlm import Rnnlm
from better_rnnlm import BetterRnnlm


class RnnlmGen(Rnnlm):
    def generate(self, start_id, skip_ids=None, sample_size=100):
        word_ids = [start_id]

        x = start_id
        while len(word_ids) < sample_size:
            x = np.array(x).reshape(1, 1)
            score = self.predict(x)
            p = softmax(score.flatten())

            sampled = np.random.choice(len(p), size=1, p=p)
            if (skip_ids is None) or (sampled not in skip_ids):
                x = sampled
                word_ids.append(int(x))

        return word_ids
```

generate( ) : 문장 생성 함수

start_id : 최초로 주는 단어의 ID

skip_id : 단어 ID의 리스트(ex. [12, 20]).

sample_size : 샘플링하는 단어의 수

7-1 언어 모델을 사용한 문장 생성

3. 더 좋은 문장으로

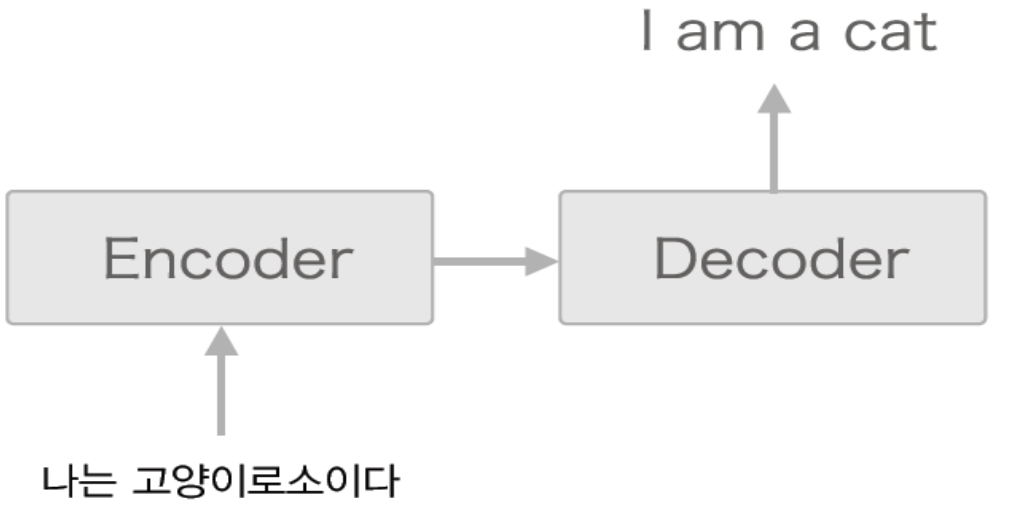
더 좋은 언어 모델을 사용하면 더 좋은 문장을 생성할 수 있다.

->더 큰 말뭉치를 사용하면 더 자연스러운 문장을 생성할 수 있다.

7-2 seq2seq

1. seq2seq의 원리

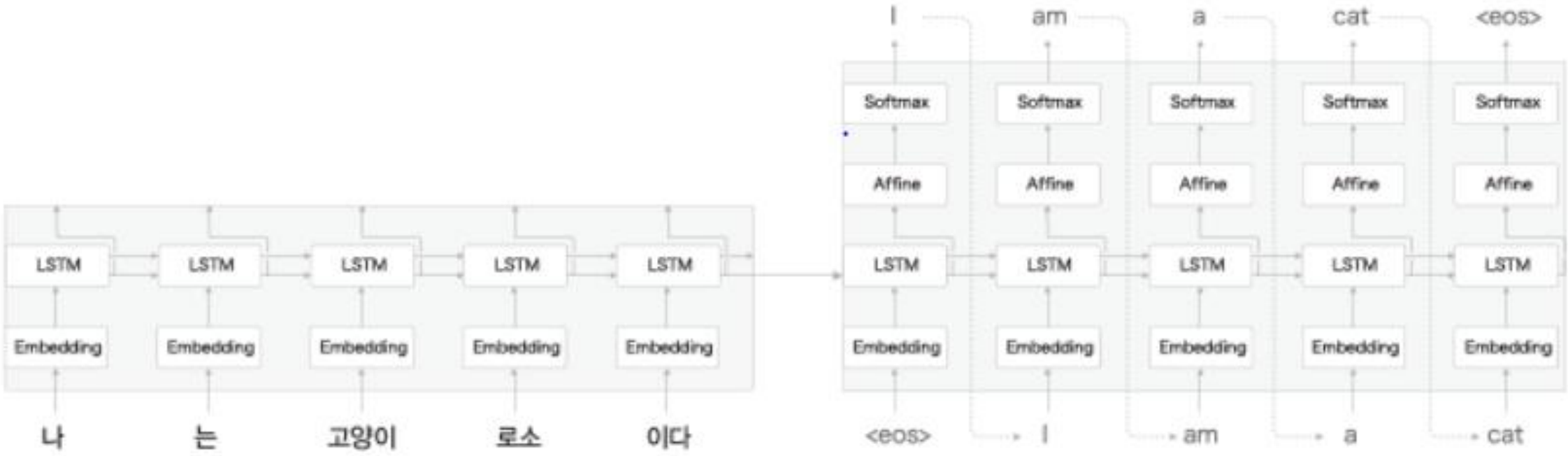
그림 7-5 Encoder와 Decoder가 번역을 수행하는 예



7-2 seq2seq

1. seq2seq의 원리

그림 7-9 seq2seq의 전체 계층 구성



7-2 seq2seq

2. 시계열 데이터 변환용 장난감 문제

그림 7-10 seq2seq에 덧셈 예제들을 학습시킨다.

7-2 seq2seq

3. 가변 길이 시계열 데이터

그림 7-11 미니배치 학습을 위해 '공백 문자'로 패딩을 수행하여 입력 · 출력 데이터의 크기를 통일한다.

입력

5	7	+	5			
6	2	8	+	5	2	1
2	2	0	+	8		

출력

_	6	2		
_	1	1	4	9
_	2	2	8	

7-2 seq2seq
4. 덧셈 데이터셋

```
import sys
sys.path.append('..')
from dataset import sequence


(x_train, t_train), (x_test, t_test) = ##
    sequence.load_data('addition.txt', seed=1984)
char_to_id, id_to_char = sequence.get_vocab()

print(x_train.shape, t_train.shape)
print(x_test.shape, t_test.shape)
# (45000, 7) (45000, 5)
# (5000, 7) (5000, 5)

print(x_train[0])
print(t_train[0])
# [ 3  0  2  0  0 11  5]
# [ 6  0 11  7  5]

print(''.join([id_to_char[c] for c in x_train[0]]))
print(''.join([id_to_char[c] for c in t_train[0]]))
# 71+118
# _189
```

x_train[0] id [3 0 2 0 0 11 5]

↓

char[7 1 + 1 1 8 <eos>]

t_train[0] id [6 0 11 7 5]

↓

char[_ 1 8 9 <eos>]

7-3 seq2seq 구현

1. Encoder 클래스

그림 7-13 Encoder의 입출력

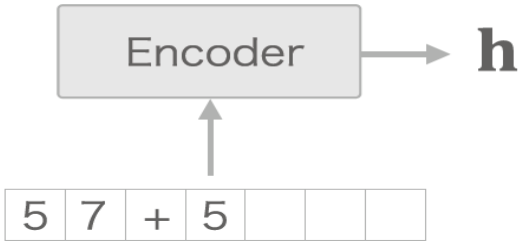


그림 7-15 Encoder를 Time 계층으로 구현한다.

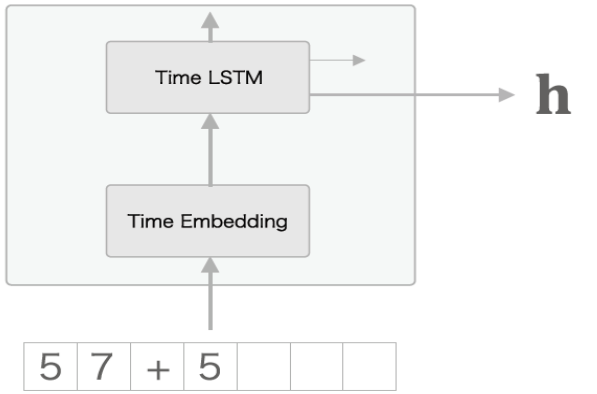
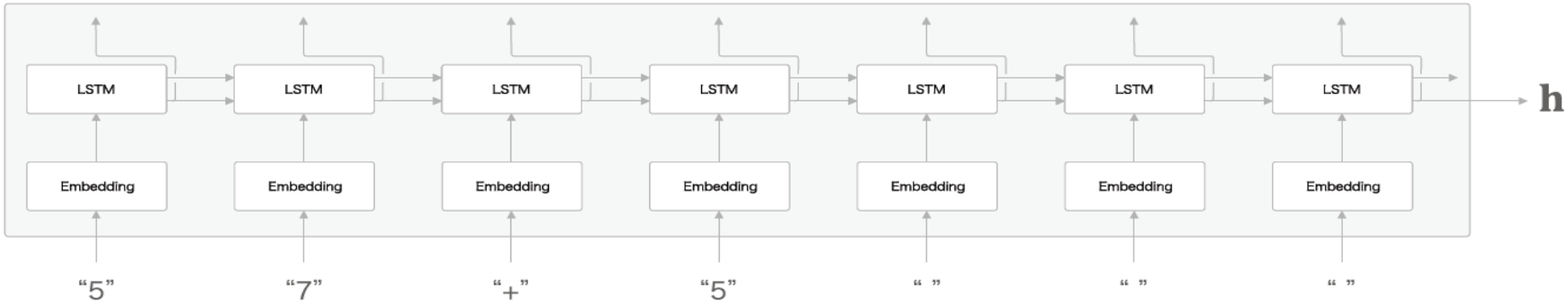


그림 7-14 Encoder의 계층 구성



7-3 seq2seq 구현

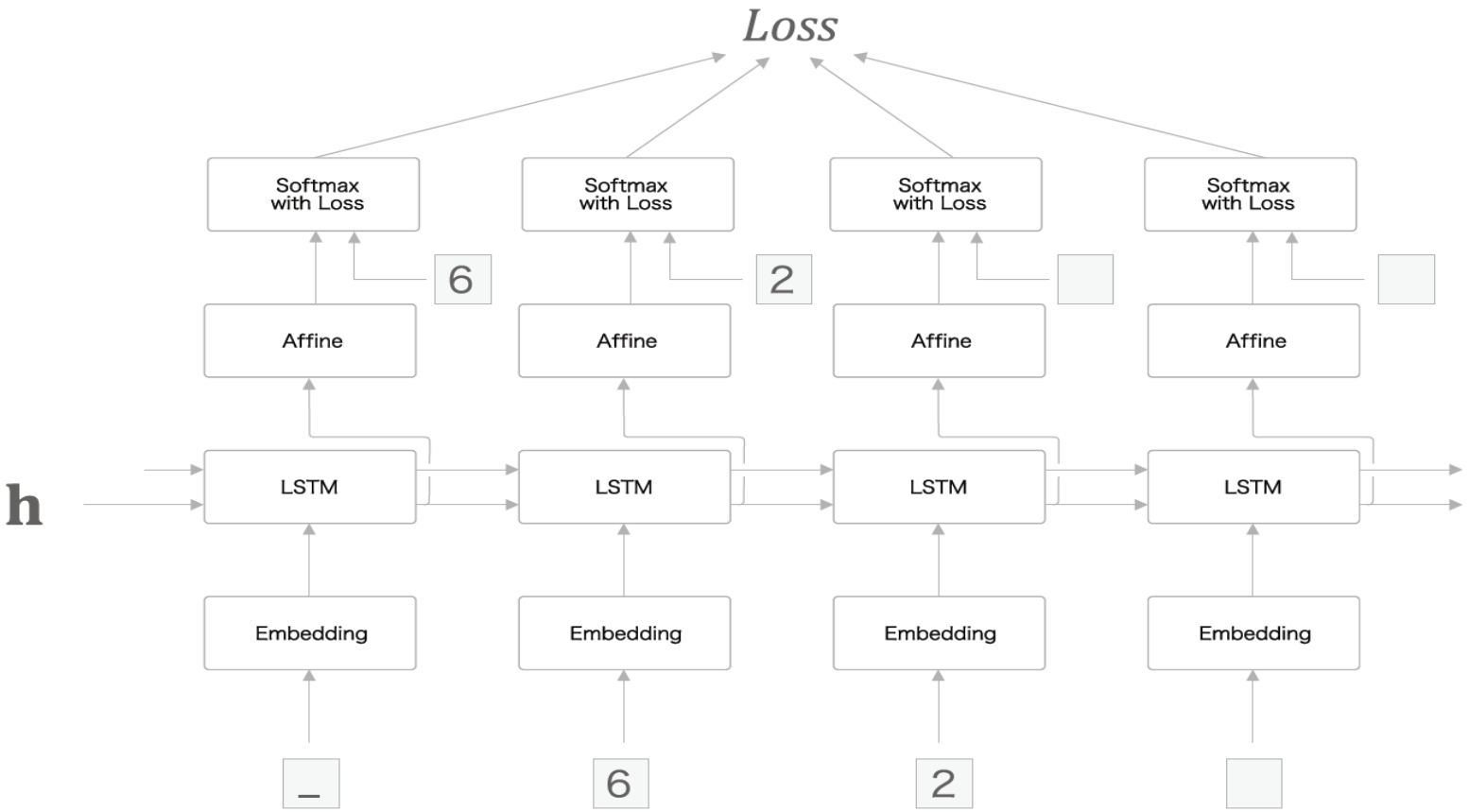
1. Encoder 클래스 (구현)

```python
class Encoder:
    def __init__(self, vocab_size, wordvec_size, hidden_size):
        # vocab_size: 어휘 수(문자의 종류)
        #  -> 0~9, '+' , '_', ' ' 로 13가지
        # wordvec_size: 문자 벡터의 차원 수
        # hidden_size: LSTM계층의 은닉 상태 벡터의 차원 수
        V, D, H = vocab_size, wordvec_size, hidden_size
        rn = np.random.randn

        embed_W = (rn(V, D) / 100).astype('f')
        lstm_Wx = (rn(D, 4 * H) / np.sqrt(D)).astype('f')
        lstm_Wh = (rn(H, 4 * H) / np.sqrt(H)).astype('f')
        lstm_b = np.zeros(4 * H).astype('f')

        # 계층 생성
        self.embed = TimeEmbedding(embed_W)
        self.lstm = TimeLSTM(lstm_Wx, lstm_Wh, lstm_b, stateful=False)

        self.params = self.embed.params + self.lstm.params  # 가중치 매개변수
        self.grads = self.embed.grads + self.lstm.grads # 기울기
        self.hs = None

    def forward(self, xs):
        xs = self.embed.forward(xs)
        hs = self.lstm.forward(xs)
        self.hs = hs
        return hs[:, -1, :] # 마지막 시각의 은닉 상태

    def backward(self, dh):
        dhs = np.zeros_like(self.hs)
         # 마지막 은닉 상태에 대한 기울기 dh를 dhs의 해당 위치에 할당
        dhs[:, -1, :] = dh

        dout = self.lstm.backward(dhs)
        dout = self.embed.backward(dout)
        return dout
```

7-3 seq2seq 구현

2. Decoder 클래스

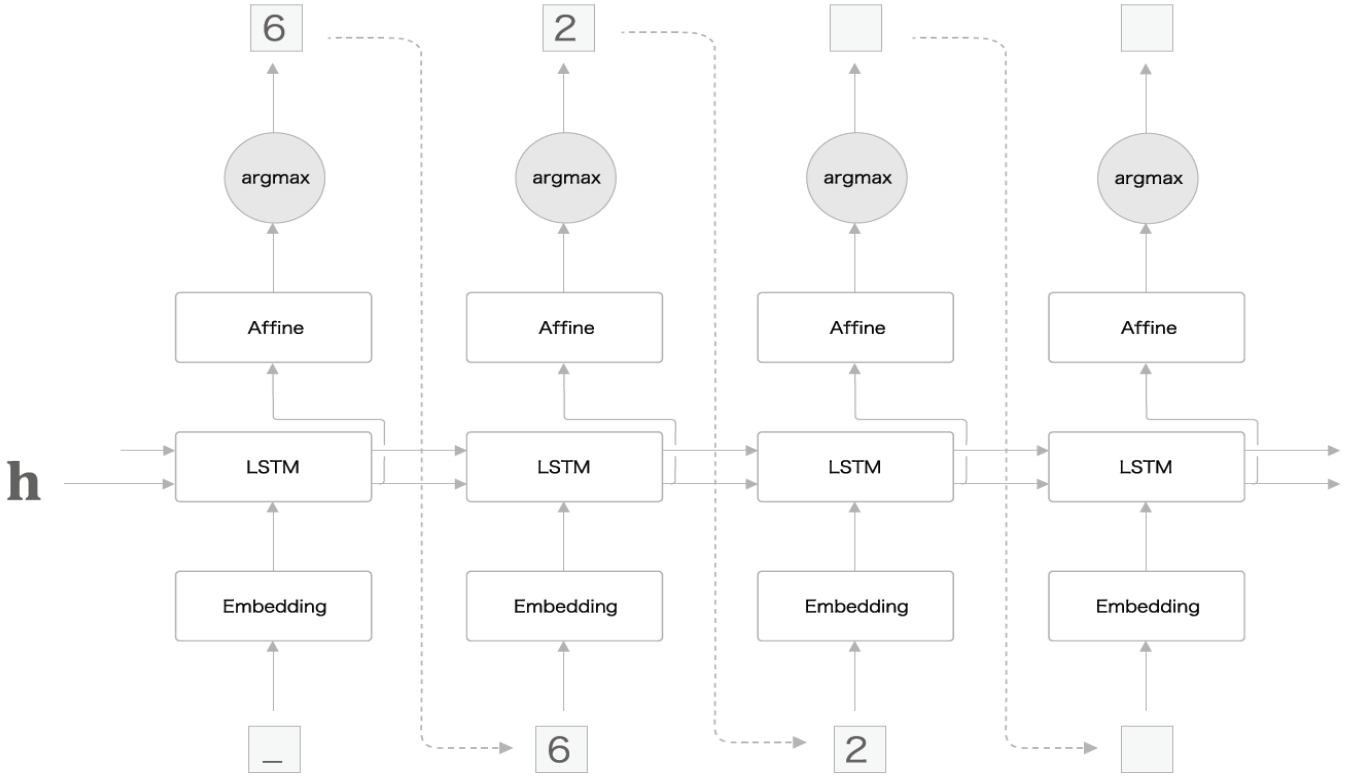
그림 7-17 Decoder의 계층 구성(학습 시)



7-3 seq2seq 구현

2. Decoder 클래스

그림 7-18 Decoder의 문자열 생성 순서: argmax 노드는 Affine 계층의 출력 중 값이 가장 큰 원소의 인덱스(문자 ID)를 반환한다.



7-3 seq2seq 구현

2. Decoder 클래스 (구현)

```python
class Decoder:
    def __init__(self, vocab_size, wordvec_size, hidden_size):
        V, D, H = vocab_size, wordvec_size, hidden_size
        rn = np.random.randn

        embed_W = (rn(V, D) / 100).astype('f')
        lstm_Wx = (rn(D, 4 * H) / np.sqrt(D)).astype('f')
        lstm_Wh = (rn(H, 4 * H) / np.sqrt(H)).astype('f')
        lstm_b = np.zeros(4 * H).astype('f')
        affine_W = (rn(H, V) / np.sqrt(H)).astype('f')
        affine_b = np.zeros(V).astype('f')

        # 계층 생성
        self.embed = TimeEmbedding(embed_W)
        self.lstm = TimeLSTM(lstm_Wx, lstm_Wh, lstm_b, stateful=True)
        self.affine = TimeAffine(affine_W, affine_b)

        self.params, self.grads = [], []
        for layer in (self.embed, self.lstm, self.affine):
            self.params += layer.params
            self.grads += layer.grads

    def forward(self, xs, h):
        self.lstm.set_state(h)

        out = self.embed.forward(xs)
        out = self.lstm.forward(out)
        score = self.affine.forward(out)
        return score

    def backward(self, dscore):
        # dscore : softmax 계층으로부터의 기울기
        dout = self.affine.backward(dscore)
        dout = self.lstm.backward(dout)
        dout = self.embed.backward(dout)
        dh = self.lstm.dh
        return dh
```

7-3 seq2seq 구현

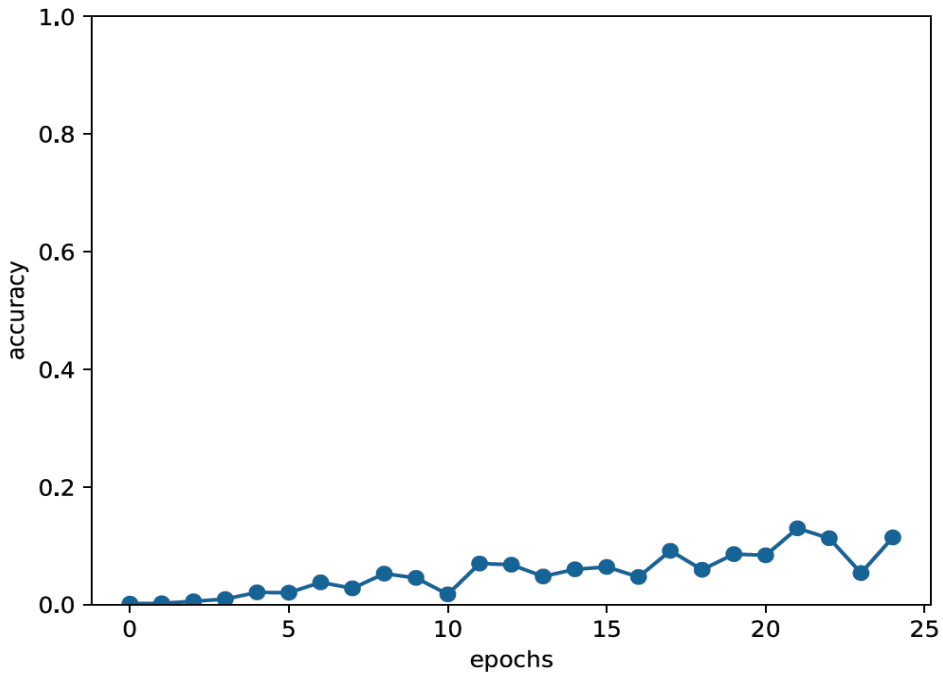
3. Seq2seq 클래스 (구현)

```
class Seq2seq(BaseModel):
    def __init__(self, vocab_size, wordvec_size, hidden_size):
        V, D, H = vocab_size, wordvec_size, hidden_size
        self.encoder = Encoder(V, D, H)
        self.decoder = Decoder(V, D, H)
        self.softmax = TimeSoftmaxWithLoss()

        self.params = self.encoder.params + self.decoder.params
        self.grads = self.encoder.grads + self.decoder.grads

    def forward(self, xs, ts):
        decoder_xs, decoder_ts = ts[:, :-1], ts[:, 1:]

        h = self.encoder.forward(xs)
        score = self.decoder.forward(decoder_xs, h)
        loss = self.softmax.forward(score, decoder_ts)
        return loss

    def backward(self, dout=1):
        dout = self.softmax.backward(dout)
        dh = self.decoder.backward(dout)
        dout = self.encoder.backward(dh)
        return dout

    def generate(self, xs, start_id, sample_size):
        h = self.encoder.forward(xs)
        sampled = self.decoder.generate(h, start_id, sample_size)
        return sampled
```

7-3 seq2seq 구현

4. seq2seq 평가

그림 7-22 정답률 추이




```python
import sys
sys.path.append('..')
import numpy as np
import matplotlib.pyplot as plt
from dataset import sequence
from common.optimizer import Adam
from common.trainer import Trainer
from common.util import eval_seq2seq
from seq2seq import Seq2seq
from peeky_seq2seq import PeekySeq2seq


# 데이터셋 읽기
(x_train, t_train), (x_test, t_test) = sequence.load_data('addition.txt')
char_to_id, id_to_char = sequence.get_vocab()

# 하이퍼파라미터 설정
vocab_size = len(char_to_id)
wordvec_size = 16
hideen_size = 128
batch_size = 128
max_epoch = 25
max_grad = 5.0

# 모델/ 옵티마이저/ 트레이너 생성
model = Seq2seq(vocab_size, wordvec_size, hideen_size)
optimizer = Adam()
trainer = Trainer(model, optimizer)

acc_list = []
for epoch in range(max_epoch):
    trainer.fit(x_train, t_train, max_epoch=1,
                batch_size=batch_size, max_grad=max_grad)

    correct_num = 0
    for i in range(len(x_test)):
        question, correct = x_test[[i]], t_test[[i]]
        verbose = i < 10
        correct_num += eval_seq2seq(model, question, correct,
                                    id_to_char, verbose, is_reverse)

    acc = float(correct_num) / len(x_test)
    acc_list.append(acc)
    print('검증 정확도 %.3f%%' % (acc * 100))
```

7-4 seq2seq 개선

1. 입력 데이터 반전(Reverse)

그림 7-23 입력 데이터를 반전시키는 예

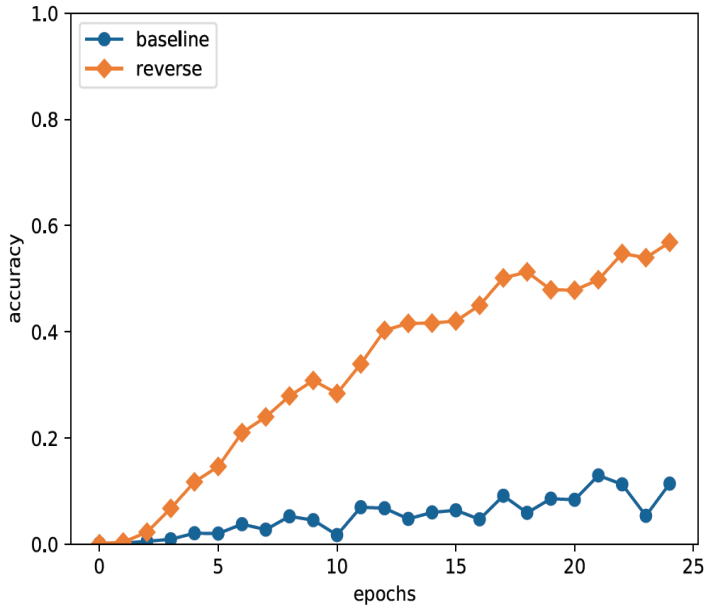
5	7	+	5			
6	2	8	+	5	2	1
2	2	0	+	8		
⋮						

반전

			5	+	7	5
1	2	5	+	8	2	6
		8	+	0	2	2
			⋮			

```
x_train, x_test = x_train[:, ::-1], x_test[:, ::-1]
```

그림 7-24 seq2seq의 정답률 추이: baseline은 앞 절의 결과, reverse는 입력 데이터를 반전시킨 결과



7-4 seq2seq 개선

2. 엿보기(Peeky)

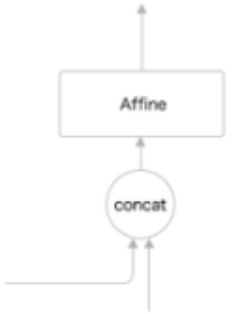
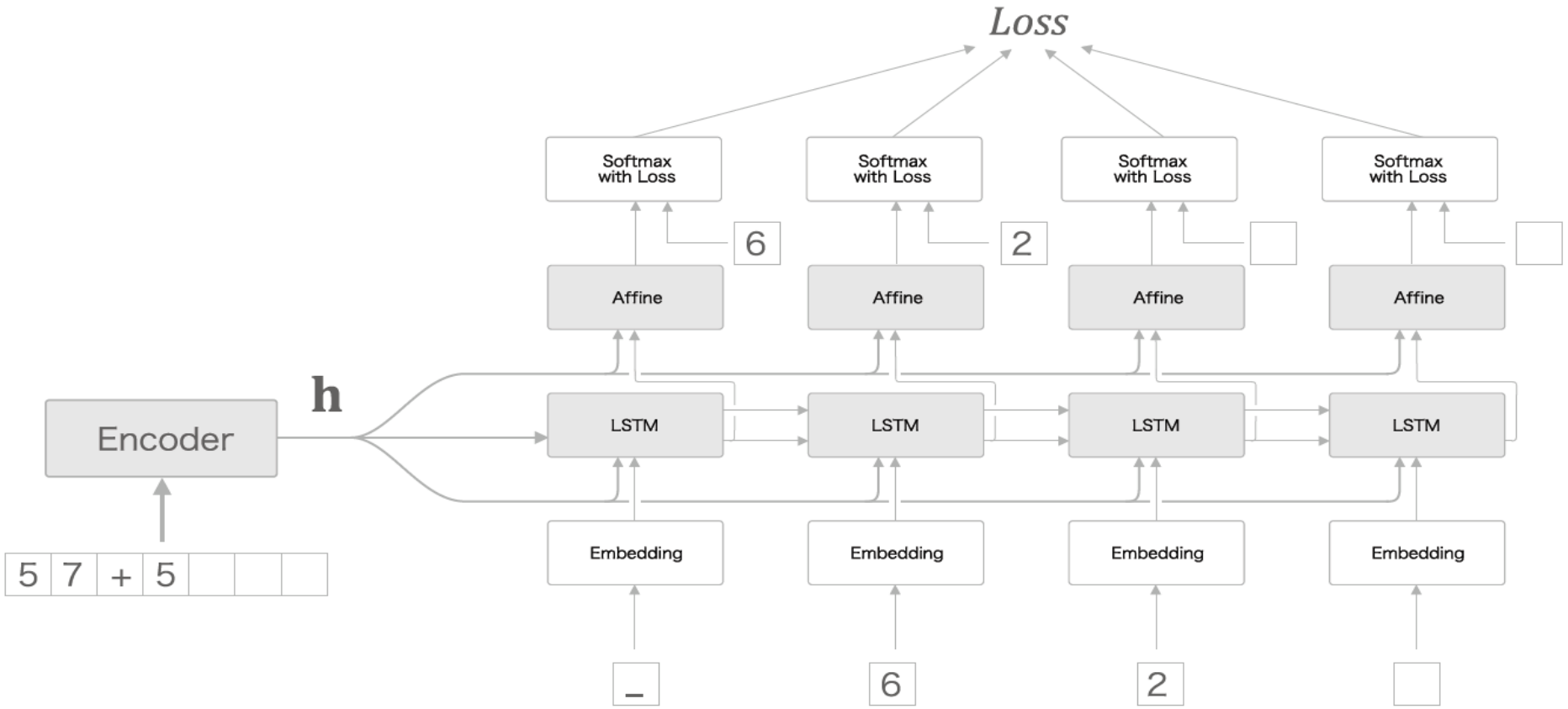


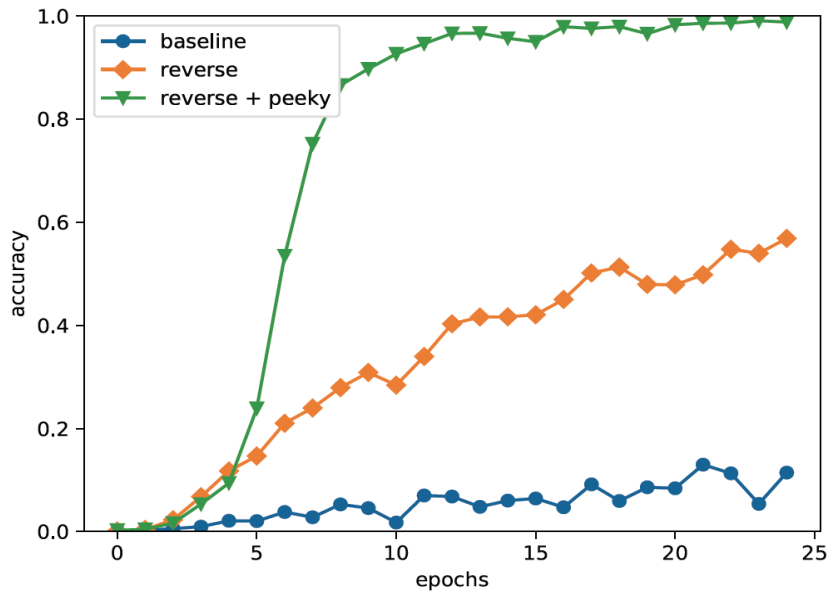
그림 7-26 개선 후: Encoder의 출력 **h**를 모든 시각의 LSTM 계층과 Affine 계층에 전해준다.



7-4 seq2seq 개선

2. 엿보기(Peeky)

그림 7-28 'reverse + peeky' 조합: 두 가지 개선을 모두 적용한 결과




```python
class PeekyDecoder:
    def __init__(self, vocab_size, wordvec_size, hidden_size):
        V, D, H = vocab_size, wordvec_size, hidden_size
        rn = np.random.randn

        embed_W = (rn(V, D) / 100).astype('f')
        lstm_Wx = (rn(H + D, 4 * H) / np.sqrt(H + D)).astype('f')
        lstm_Wh = (rn(H, 4 * H) / np.sqrt(H)).astype('f')
        lstm_b = np.zeros(4 * H).astype('f')
        affine_W = (rn(H + H, V) / np.sqrt(H + H)).astype('f')
        affine_b = np.zeros(V).astype('f')

        # 계층 생성
        self.embed = TimeEmbedding(embed_W)
        self.lstm = TimeLSTM(lstm_Wx, lstm_Wh, lstm_b, stateful=True)
        self.affine = TimeAffine(affine_W, affine_b)

        self.params, self.grads = [], []
        for layer in (self.embed, self.lstm, self.affine):
            self.params += layer.params
            self.grads += layer.grads
        self.cache = None

    def forward(self, xs, h):
        N, T = xs.shape
        N, H = h.shape

        self.lstm.set_state(h)

        out = self.embed.forward(xs)
        # h를 시계열만큼 복제해서 hs에 저장
        hs = np.repeat(h, T, axis=0).reshape(N, T, H)
        # hs와 out을 연결
        out = np.concatenate((hs, out), axis=2)
        # lstm 계층에 입력
        out = self.lstm.forward(out)
        # hs와 out을 연결
        out = np.concatenate((hs, out), axis=2)

        score = self.affine.forward(out)
        self.cache = H
        return score
```

7-5 seq2seq를 이용한 애플리케이션

1. 챗봇

7-5 seq2seq를 이용한 애플리케이션

2. 이미지 캡셔닝

그림 7-31 이미지 캡셔닝을 수행하는 seq2seq의 신경망 구성 예

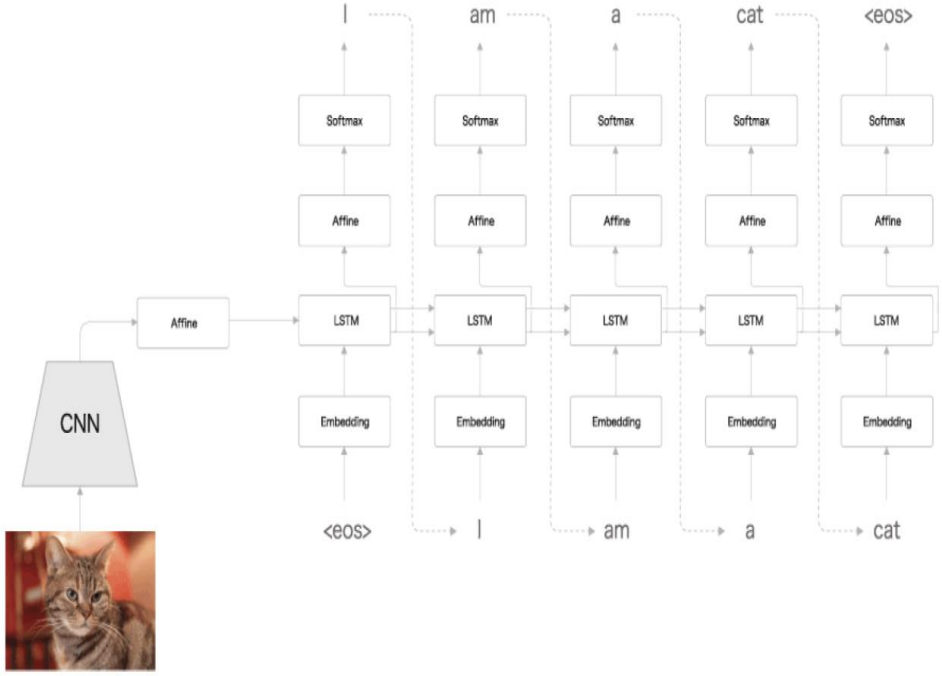
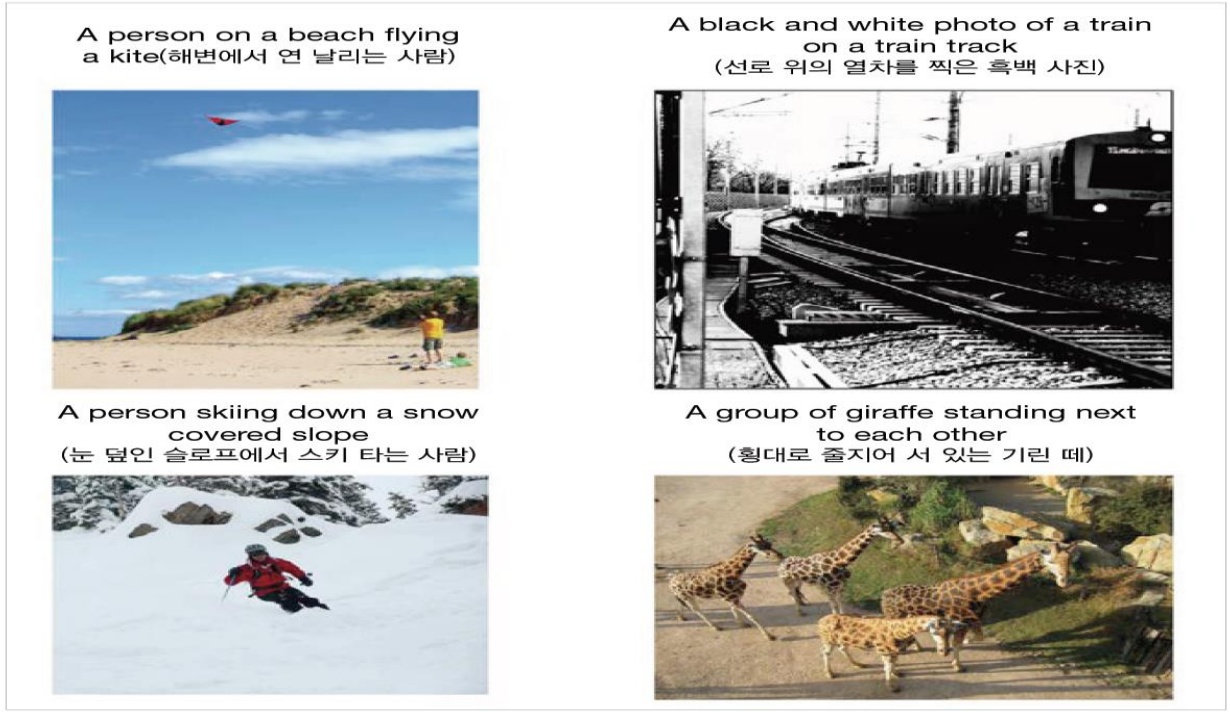


그림 7-32 이미지 캡셔닝의 예: 이미지를 텍스트로 변환(문헌 [47]에서 발췌)



7-6 정리

1.RNN을 이용한 언어 모델은 새로운 문장을 생성할 수 있다.

2.문장을 생성할 때는 하나의 단어(혹은 문자)를 주고 모델의 출력(확률분포)에서 샘플링하는 과정을 반복한다.

3.RNN을 2개 조합함으로써 시계열 데이터를 다른 시계열 데이터로 변환할 수 있다.

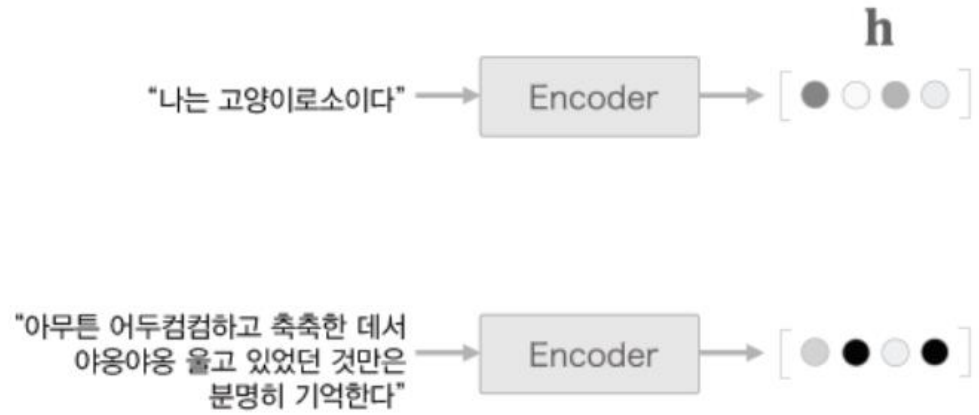
4.Seq2seq는 Encoder가 출발어 입력문을 인코딩하고, 인코딩된 정보를 Decoder가 받아 디코딩하여 도착어 출력문을 얻는다.

5.입력문을 반전시키는 기법(Reverse), 또는 인코딩된 정보를 Decoder의 여러 계층에 전달하는 기법(Peeky)은 seq2seq의 정확도 향상에 효과적이다.

6.기계 번역, 챗봇, 이미지 캡셔닝 등 seq2seq는 다양한 애플리케이션에 이용할 수 있다.

8-1 어텐션의 구조

1. seq2seq의 문제점



1. 하나의 고정된 벡터에 모든 정보를 압축하다보니 정보 손실이 발생한다.

2. RNN의 고질적인 문제인 기울기 손실 문제가 존재한다.

8-1 어텐션의 구조

2. Encoder 개선

그림 8-2 Encoder의 시각별(단어별) LSTM 계층의 은닉 상태를 모두 이용(**hs**로 표기)

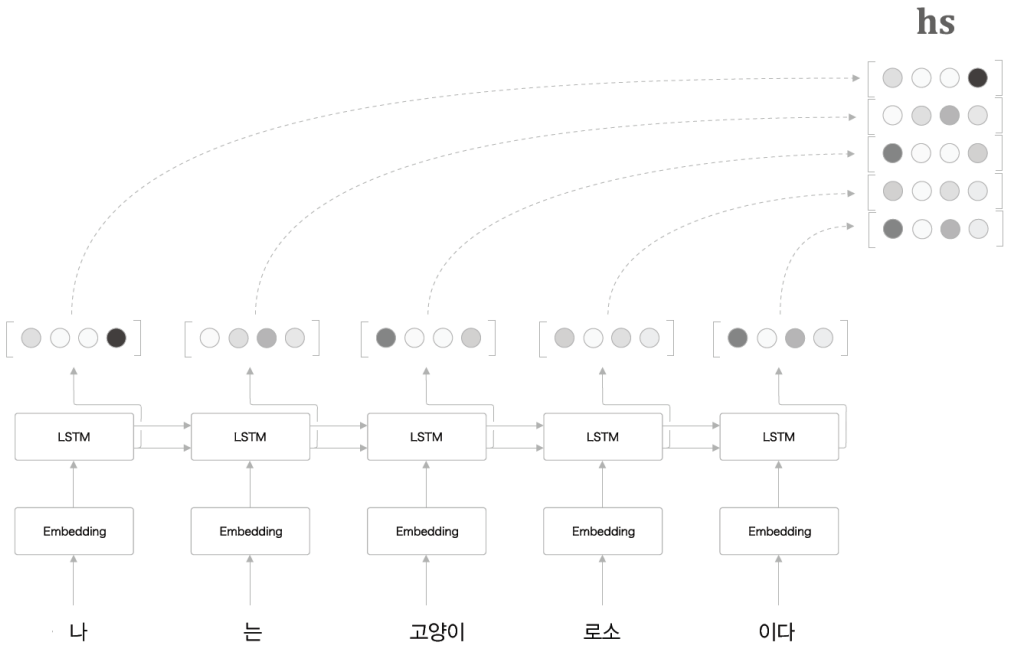
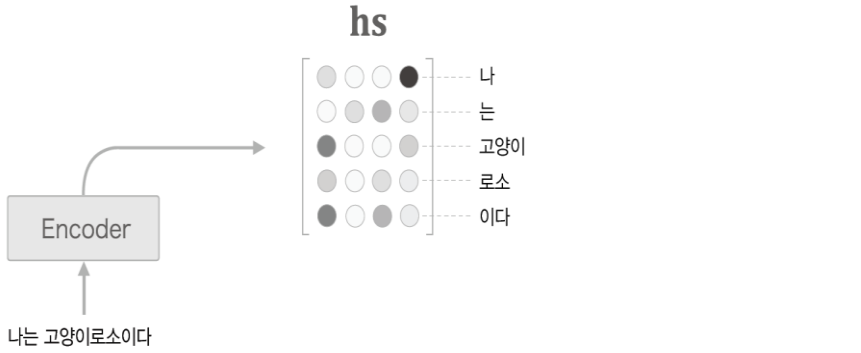


그림 8-3 Encoder의 출력 **hs**는 단어 수만큼의 벡터를 포함하며, 각각의 벡터는 해당 단어에 대한 정보를 많이 포함한다.



8-1 어텐션의 구조

3. Decoder 개선 - 1

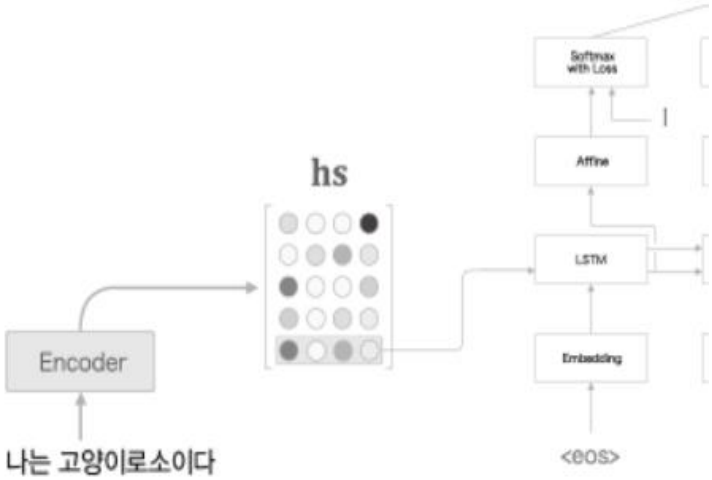
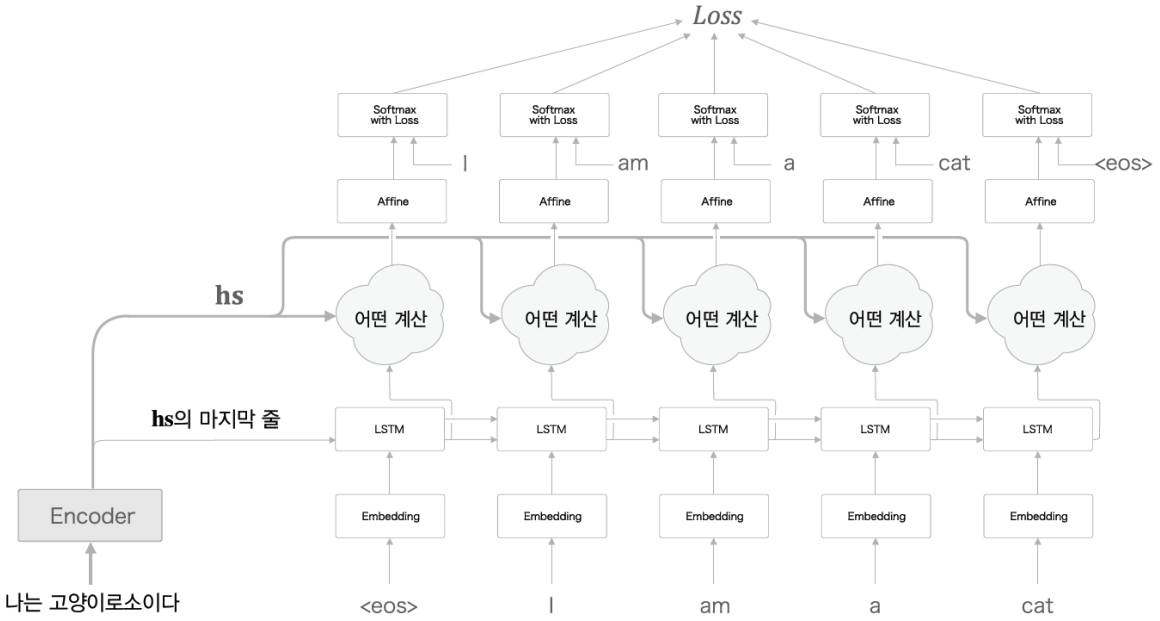


그림 8-6 개선 후의 Decoder의 계층 구성



8-1 어텐션의 구조

3. Decoder 개선 - 1

그림 8-7 각 단어에 대해서 그것이 얼마나 중요한지를 나타내는 '가중치'를 구한다(구하는 방법은 뒤에서 설명).

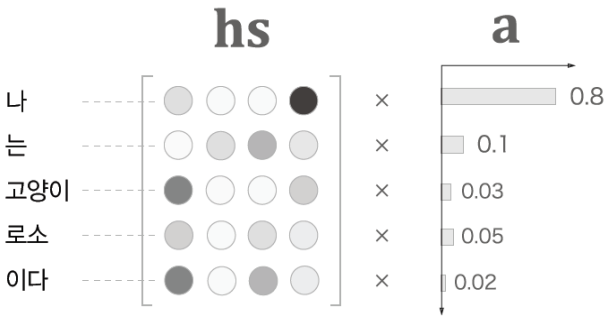
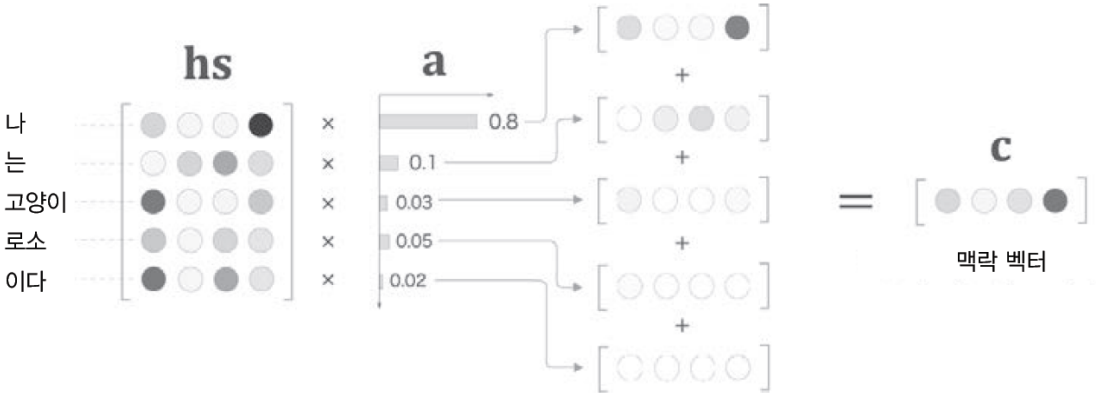


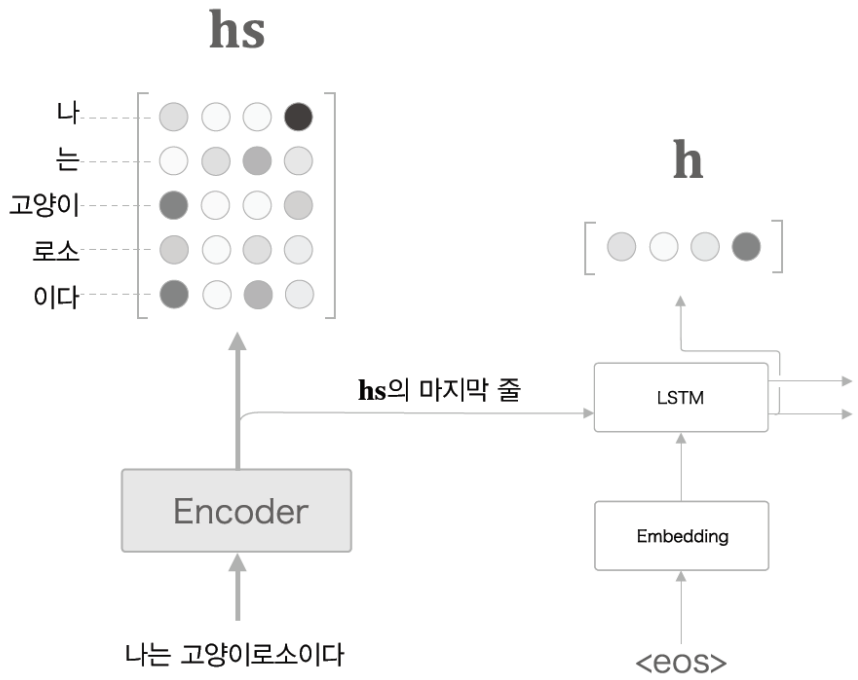
그림 8-8 가중합을 계산하여 '맥락 벡터'를 구한다.

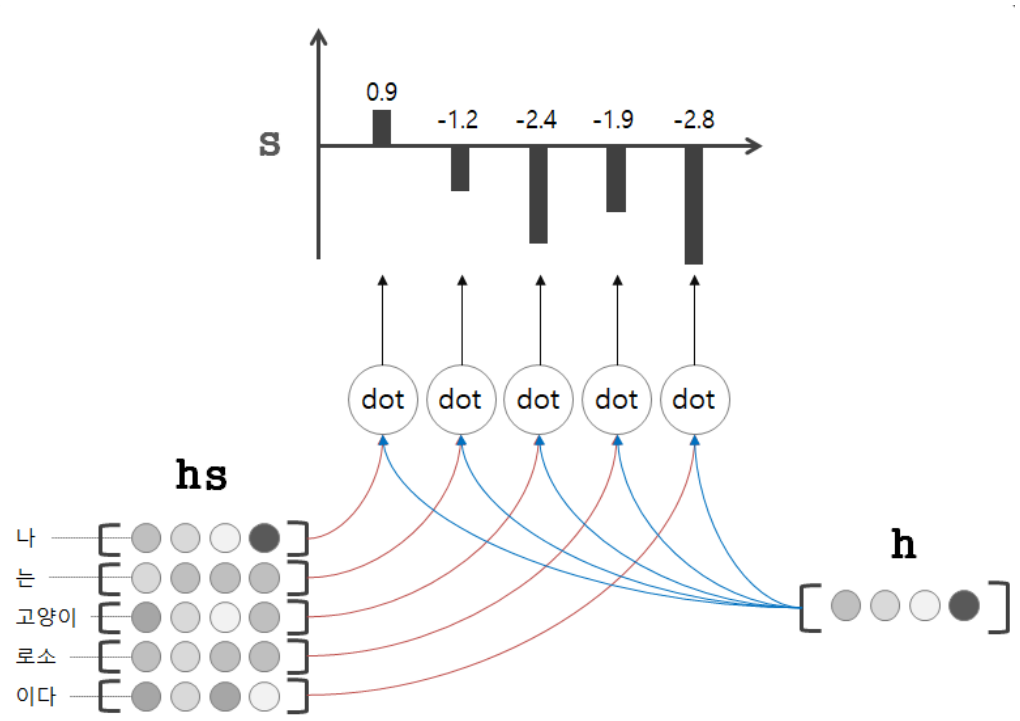


8-1 어텐션의 구조

4. Decoder 개선 - 2

그림 8-12 Decoder의 첫 번째 LSTM 계층의 은닉 상태 벡터

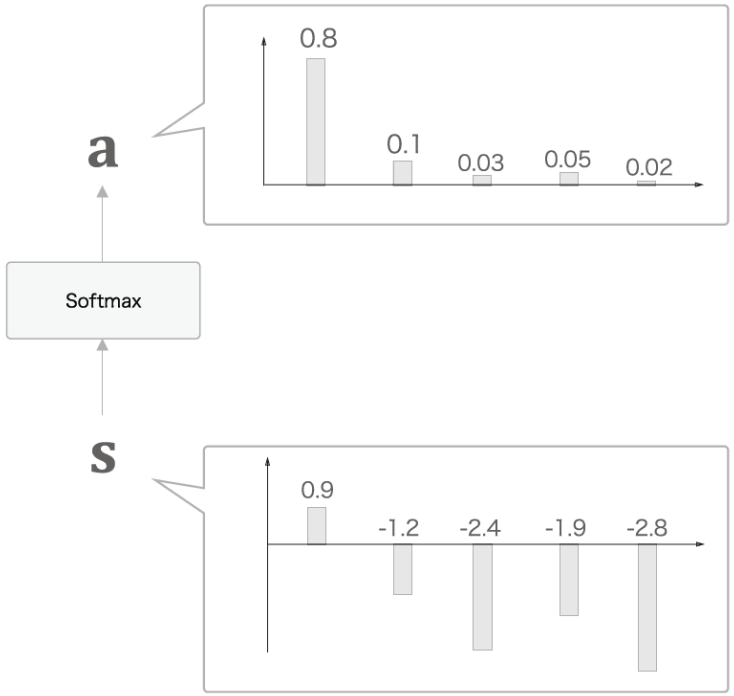




8-1 어텐션의 구조

4. Decoder 개선 - 2

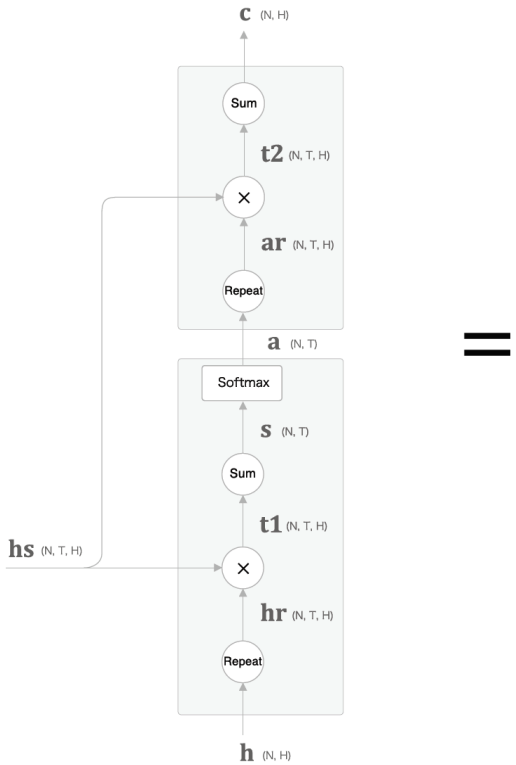
그림 8-14 Softmax를 통한 정규화



8-1 어텐션의 구조

5. Decoder 개선 - 3

그림 8-16 맥락 벡터를 계산하는 계산 그래프



=

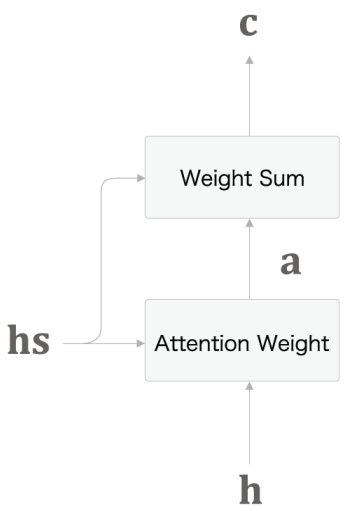
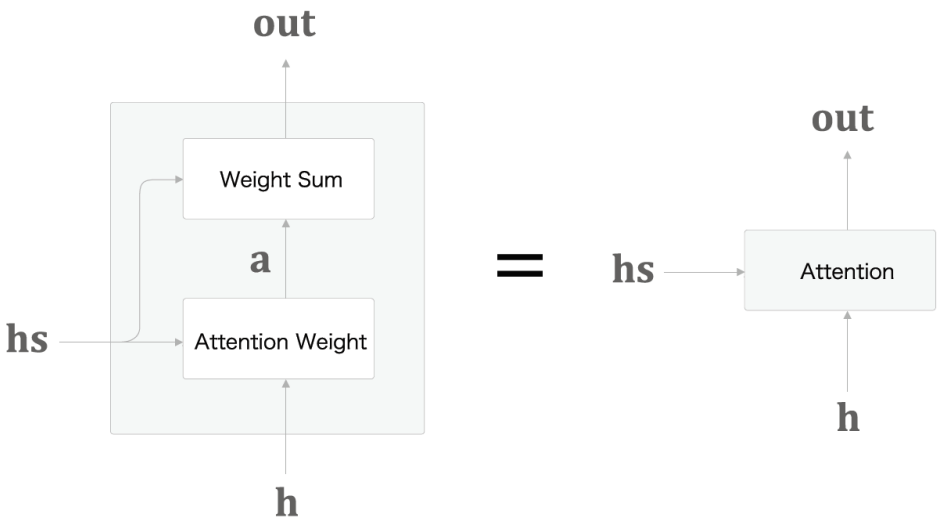


그림 8-17 왼쪽 계산 그래프를 Attention 계층으로 정리



8-1 어텐션의 구조

그림 8-18 Attention 계층을 갖춘 Decoder의 계층 구성

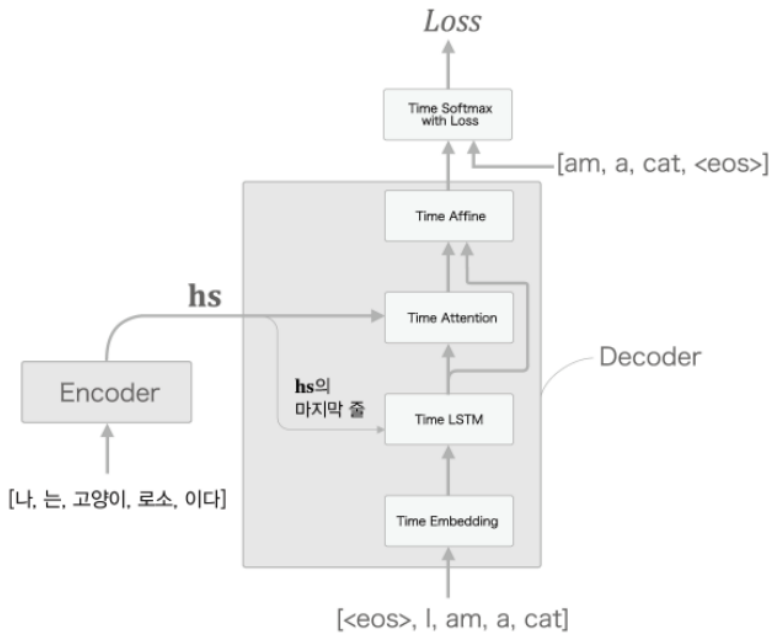
8-2 어텐션을 갖춘 seq2seq 구현

1. Encoder 구현

```
import sys
sys.path.append('..')
from common.time_layers import *
from seq2seq import Encoder, Seq2seq
from attention_layer import TimeAttention


class AttentionEncoder(Encoder):
    def forward(self, xs):
        xs = self.embed.forward(xs)
        hs = self.lstm.forward(xs)
        return hs

    def backward(self, dhs):
        dout = self.lstm.backward(dhs)
        dout = self.embed.backward(dout)
        return dout
```

8-2 어텐션을 갖춘 seq2seq 구현

2. Decoder 구현




```python
class AttentionDecoder:
    def __init__(self, vocab_size, wordvec_size, hidden_size):
        V, D, H = vocab_size, wordvec_size, hidden_size
        rn = np.random.randn

        embed_W = (rn(V, D) / 100).astype('f')
        lstm_Wx = (rn(D, 4 * H) / np.sqrt(D)).astype('f')
        lstm_Wh = (rn(H, 4 * H) / np.sqrt(H)).astype('f')
        lstm_b = np.zeros(4 * H).astype('f')
        affine_W = (rn(2*H, V) / np.sqrt(2*H)).astype('f')
        affine_b = np.zeros(V).astype('f')

        self.embed = TimeEmbedding(embed_W)
        self.lstm = TimeLSTM(lstm_Wx, lstm_Wh, lstm_b, stateful=True)
        self.attention = TimeAttention()  # Attention 레이어
        self.affine = TimeAffine(affine_W, affine_b)
        layers = [self.embed, self.lstm, self.attention, self.affine]

        self.params, self.grads = [], []
        for layer in layers:
            self.params += layer.params
            self.grads += layer.grads

    def forward(self, xs, enc_hs):
        h = enc_hs[:,-1]
        self.lstm.set_state(h)

        out = self.embed.forward(xs)
        dec_hs = self.lstm.forward(out)
        c = self.attention.forward(enc_hs, dec_hs)  # context vector
        out = np.concatenate((c, dec_hs), axis=2)  # context_vector & lstm h_t
        score = self.affine.forward(out)

        return score

    def backward(self, dscore):
        dout = self.affine.backward(dscore)
        N, T, H2 = dout.shape
        H = H2 // 2

        dc, ddec_hs0 = dout[:,:,:H], dout[:,:,H:]
        denc_hs, ddec_hs1 = self.attention.backward(dc)
        ddec_hs = ddec_hs0 + ddec_hs1
        dout = self.lstm.backward(ddec_hs)
        dh = self.lstm.dh
        denc_hs[:, -1] += dh
        self.embed.backward(dout)

        return denc_hs
```

8-2 어텐션을 갖춘 seq2seq 구현

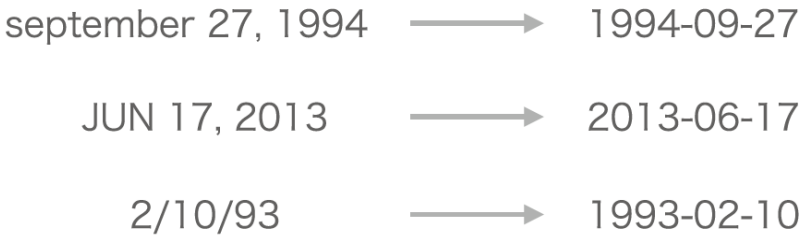
3. seq2seq 구현

```
class AttentionSeq2seq(Seq2seq):
    def __init__(self, vocab_size, wordvec_size, hidden_size):
        args = vocab_size, wordvec_size, hidden_size
        self.encoder = AttentionEncoder(*args)
        self.decoder = AttentionDecoder(*args)
        self.softmax = TimeSoftmaxWithLoss()

        self.params = self.encoder.params + self.decoder.params
        self.grads = self.encoder.grads + self.decoder.grads
```

8-3 어텐션 평가

1. 날짜 형식 변환 문제

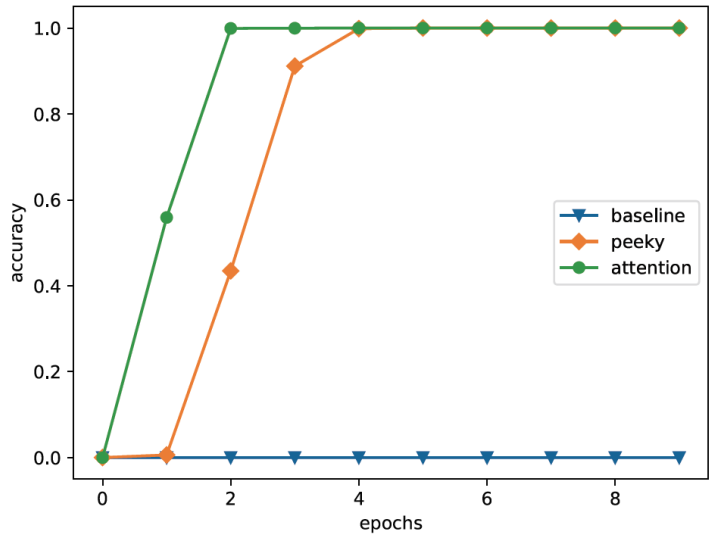
그림 8-22 날짜 형식 변환의 예



8-3 어텐션 평가

2. 어텐션을 갖춘 seq2seq의 학습

그림 8-26 다른 모델과의 비교: 'baseline'은 앞 장의 단순한 seq2seq, 'peeky'는 엿보기를 적용한 seq2seq(입력 문장 반전은 모든 모델에서 사용)

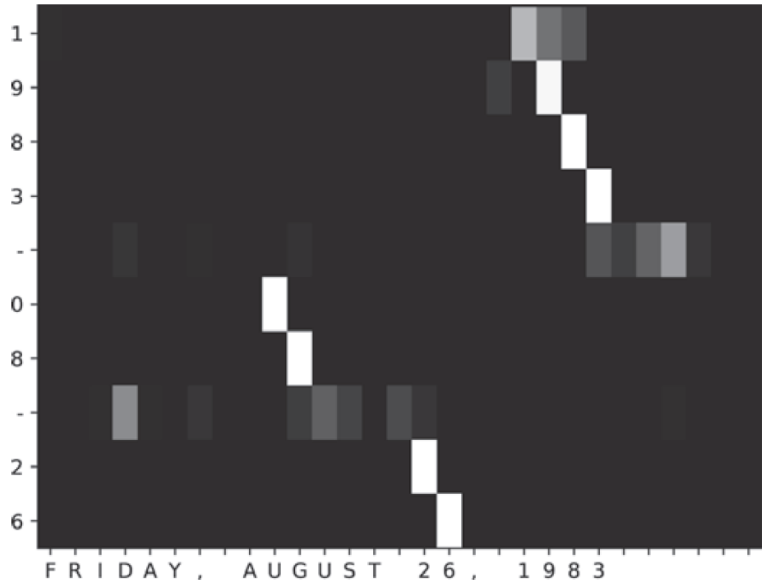



```python
# 데이터 읽기
(x_train, t_train), (x_test, t_test) = sequence.load_data('date.txt')
char_to_id, id_to_char = sequence.get_vocab()

# 입력 문장 반전
x_train, x_test = x_train[:, ::-1], x_test[:, ::-1]

# 하이퍼파라미터 설정
vocab_size = len(char_to_id)
wordvec_size = 16
hidden_size = 256
batch_size = 128
max_epoch = 10
max_grad = 5.0

model = AttentionSeq2seq(vocab_size, wordvec_size, hidden_size)
# model = Seq2seq(vocab_size, wordvec_size, hidden_size)
# model = PeekySeq2seq(vocab_size, wordvec_size, hidden_size)

optimizer = Adam()
trainer = Trainer(model, optimizer)

acc_list = []
for epoch in range(max_epoch):
    trainer.fit(x_train, t_train, max_epoch=1,
                batch_size=batch_size, max_grad=max_grad, eval_interval=150)

    correct_num = 0
    for i in range(len(x_test)):
        question, correct = x_test[[i]], t_test[[i]]
        verbose = i < 10
        correct_num += eval_seq2seq(model, question, correct,
                                    id_to_char, verbose, is_reverse=True)

    acc = float(correct_num) / len(x_test)
    acc_list.append(acc)
    print('정확도 %.3f%%' % (acc * 100))
```

8-3 어텐션 평가

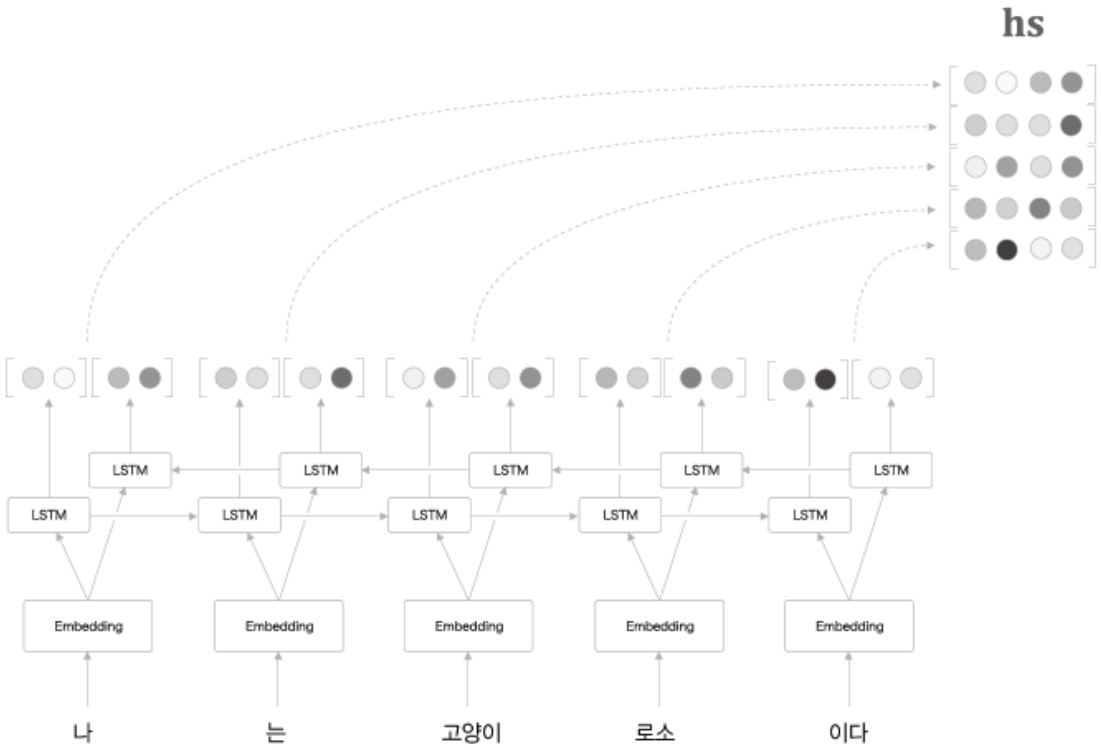
3. 어텐션 시각화

그림 8-27 학습된 모델을 사용하여 시계열 변환을 수행했을 때의 어텐션 가중치 시각화: 가로축은 입력 문장, 세로축은 출력 문장, 맵의 각 원소는 밝을수록 값이 크다(1.0에 가깝다).



8-4 어텐션에 관한 남은 이야기

1. 양방향 RNN

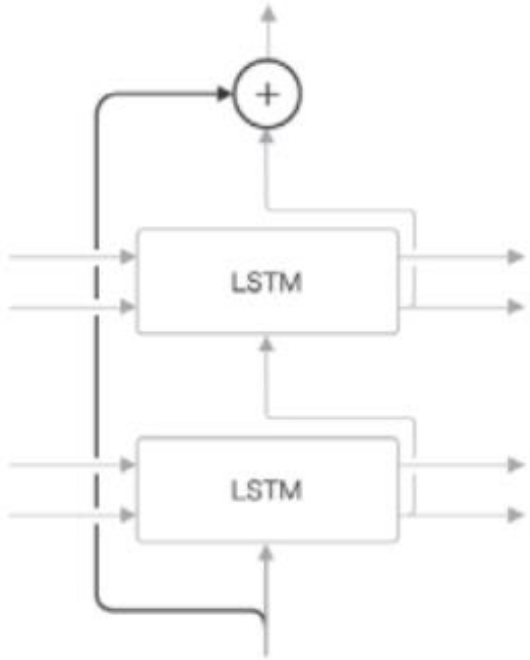


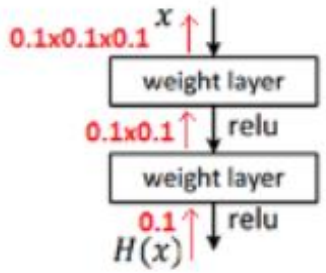

```python
class TimeBiLSTM:
    def __init__(self, Wx1, Wh1, b1,
                 Wx2, Wh2, b2, stateful=False):
        self.forward_lstm = TimeLSTM(Wx1, Wh1, b1, stateful)
        self.backward_lstm = TimeLSTM(Wx2, Wh2, b2, stateful)
        self.params = self.forward_lstm.params + self.backward_lstm.params
        self.grads = self.forward_lstm.grads + self.backward_lstm.grads

    def forward(self, xs):
        o1 = self.forward_lstm.forward(xs)
        o2 = self.backward_lstm.forward(xs[:, ::-1])  # backward를 위해 입력데이터 반전
        o2 = o2[:, ::-1]

        out = np.concatenate((o1, o2), axis=2)  # forward, backward concat
        return out

    def backward(self, dhs):
        H = dhs.shape[2] // 2
        do1 = dhs[:, :, :H]
        do2 = dhs[:, :, H:]

        dxs1 = self.forward_lstm.backward(do1)
        do2 = do2[:, ::-1]
        dxs2 = self.backward_lstm.backward(do2)
        dxs2 = dxs2[:, ::-1]
        dxs = dxs1 + dxs2
        return dxs
```

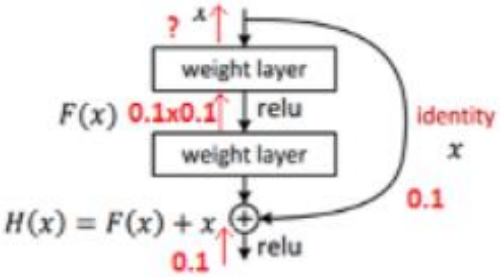
8-4 어텐션에 관한 남은 이야기

3. seq2seq 심층화와 skip연결

그림 8-34 LSTM 계층의 skip 연결 예



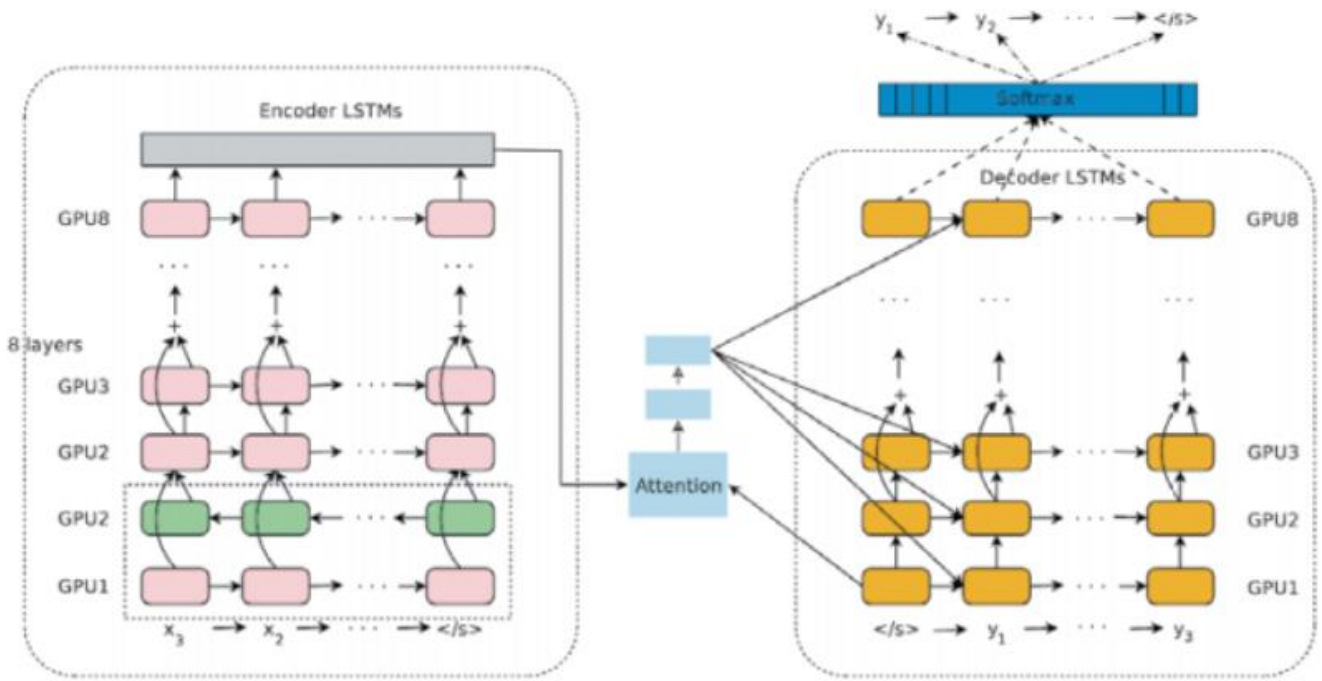




8-5 어텐션 응용

1. 구글 신경망 기계 번역(GNMT)

그림 8-35 GNMT의 계층 구성(문헌 [50]에서 발췌)



8-5 어텐션 응용

2. 트랜스포머

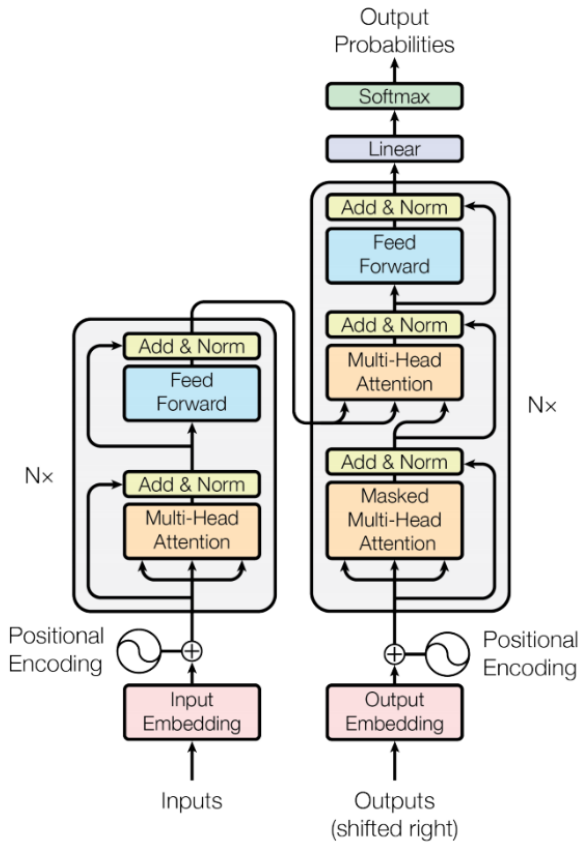


Figure 1: The Transformer - model architecture.

8-6 정리

- 번역이나 음성 인식 등, 한 시계열 데이터를 다른 시계열 데이터로 변환하는 작업에서는 시계열 데이터 사이의 대응 관계가 존재하는 경우가 많다
- 어텐션은 두 시계열 데이터 사이의 대응 관계를 데이터로부터 학습한다.
- 어텐션에서는 벡터의 내적을 사용해 벡터 사이의 유사도를 구하고, 그 유사도를 이용한 가중합 벡터가 어텐션의 출력이 된다.
- 어텐션에서 사용하는 연산은 미분 가능하기 때문에 오차역전파법으로 학습할 수 있다.
- 어텐션이 산출하는 가중치(확률)을 시각화하면 입출력의 대응 관계를 볼 수 있다.
- 외부 메모리를 활용한 신경망 확장 연구 예에서는 메모리를 읽고 쓰는 데 어텐션을 사용했다.