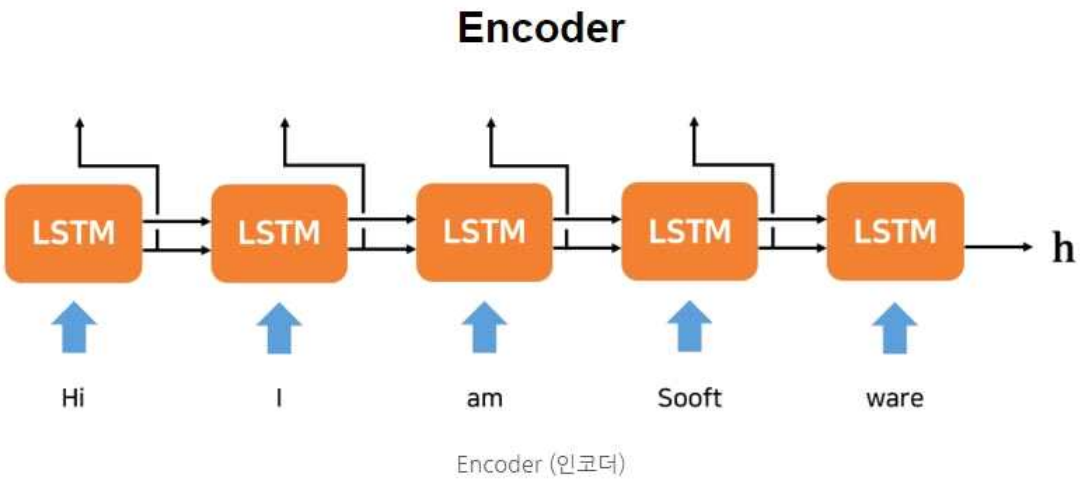
연구윤리 및 연구지도
(9주차 보고서)

경북대학교 전자공학부
2016113566 김남영

1) seq2seq 코드 구현 (코드 한줄씩 분석)

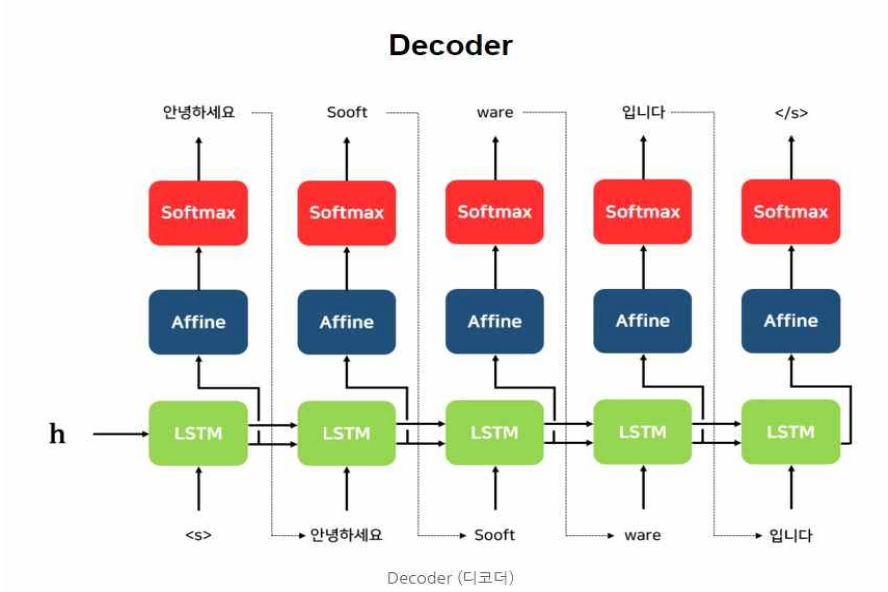


Encoder (인코더)

인코더의 구조

```python
class Encoder:
    def __init__(self, vocab_size, wordvec_size, hidden_size):
        V, D, H = vocab_size, wordvec_size, hidden_size
        # V는 어휘 수( 0~9, +, 공백 -> 13가지)
        # D는 문자 벡터의 차원 수
        # H는 은닉 상태 벡터의 차원 수
        rn = np.random.randn

        #위 Encoder 구조에 따라 파라미터를 초기화
        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')

        #위 Encoder 구조에 따르면 encoder는 TimeEmbedding, TimeLSTM 계층으로 이루어져 있음
        #위에서 초기화 시킨 파라미터를 이용하여 계층 생성
        self.embed = TimeEmbedding(embed_W)
        self.lstm = TimeLSTM(lstm_Wx, lstm_Wh, lstm_b, stateful=False)

        # 계층 각각의 파라미터를 더해서 params와 grads에 저장
        self.params = self.embed.params + self.lstm.params
        self.grads = self.embed.grads + self.lstm.grads
        self.hs = None

    def forward(self, xs):
        xs = self.embed.forward(xs)
        #Time Embedding과 TimeLSTM 에서 구현해둔 순전파를 그대로 호출하면 됨
        hs = self.lstm.forward(xs)
        self.hs = hs
        return hs[:, -1, :] # 마지막 시각의 은닉 상태만 필요하므로 마지막 hs만 반환

    def backward(self, dh): # dh는 마지막 은닉 상태 h에 대한 기울기
        dhs = np.zeros_like(self.hs) #hs와 같은 크기의 텐서 dhs를 생성
        dhs[:, -1, :] = dh #dh를 dhs에 할당
        #dh는 마지막 은닉상태에 대한 기울기이므로 [:,-1,:]가 해당 위치임

        dout = self.lstm.backward(dhs) # 구현해둔 역전파를 그대로 호출하면 됨
        dout = self.embed.backward(dout)
        return dout
```

인코더 구현



Decoder (디코더)

디코더의 구조

```python
class Decoder: # 인코더에서 받은 h를 이용해 다른 문자열을 출력

        #decoder는 학습시에는 softmax 계층을 이용하지만
        #데이터 생성시에는 argmax(결정적) 계층을 이용하므로
        #decoder class에서는 그 이전의 affine 계층까지만 구현함

    def __init__(self, vocab_size, wordvec_size, hidden_size):
        V, D, H = vocab_size, wordvec_size, hidden_size
        # V는 어휘 수( 0~9, +, 공백 -> 13가지)
        # D는 문자 벡터의 차원 수
        # H는 은닉 상태 벡터의 차원 수
        rn = np.random.randn

        #Decoder 구조에 따라 필요한 파라미터를 초기화
        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')

        #Decoder 구조에 따라, 위에서 초기화한 파라미터로 필요한 계층을 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 = [], [] #params와 grads 리스트를 생성
        for layer in (self.embed, self.lstm, self.affine):
            self.params += layer.params # embed, lstm, affine 계층의 params를 한번에 더하기
            self.grads += layer.grads # embed, lstm, affine 계층의 grads를 한번에 더하기
```

디코더의 구현

```python
def forward(self, xs, h): #학습시 이용되는 메소드
    self.lstm.set_state(h)
    #순서대로 forward 호출하면 됨
    out = self.embed.forward(xs)
    out = self.lstm.forward(out)
    score = self.affine.forward(out)
    return score

def backward(self, dscore): # 순서대로 역전파 호출하면 됨
    dout = self.affine.backward(dscore)
    dout = self.lstm.backward(dout)
    dout = self.embed.backward(dout)
    dh = self.lstm.dh #결국 역으로 전파하고자 하는것은 lstm의 dh이므로
    return dh # dh를 꺼내어 반환

def generate(self, h, start_id, sample_size): #생성시 이용되는 메소드
    #encoder로부터 h를 받고, 최초 문자의 id인 start id 와
    #생성하는 문자 수인 sample_size를 인수로 한다.
    sampled = [] # 생성된 단어를 담기위한 리스트 생성
    sample_id = start_id
    self.lstm.set_state(h)

    for _ in range(sample_size): #생성 문자 수 만큼만 반복
        x = np.array(sample_id).reshape((1, 1)) # wordvector를 전파시키기위해 재배열
        out = self.embed.forward(x)
        out = self.lstm.forward(out)
        score = self.affine.forward(out) # 순전파해서 score를 낸다.

        sample_id = np.argmax(score.flatten()) #score가 가장 높은 문자의 ID를 선택
        sampled.append(int(sample_id)) #sampled에 계속해서 선택된 id를 덧붙이는 작업

    return sampled
```

디코더는 학습을 위한 forward메소드와 단어 생성을 위한 generate 메소드가 분리되어 있음

```python
class Seq2seq(BaseModel): # encoder와 decoder를 연결하고 손실 계산만 하면됨
    def __init__(self, vocab_size, wordvec_size, hidden_size):
        V, D, H = vocab_size, wordvec_size, hidden_size
        self.encoder = Encoder(V, D, H) # encoder 생성
        self.decoder = Decoder(V, D, H) # decoder 생성
        self.softmax = TimeSoftmaxWithLoss() #손실계산을 위한 계층 생성

        #encoder, decoder 각각의 params와 grads를 더해서 seq2seq에 담음
        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:]
        #decoder_xs에 대해서는 ts를 전부(?),
        #decoder_ts에 대해서는 ts에서 시작 구분 문자(index가 0인것)을 제외

        h = self.encoder.forward(xs) #입력을 인코더에 통과시켜 h를 얻음
        score = self.decoder.forward(decoder_xs, h)
        #h에 의해 decoder의 lstm이 set되고 decoder_xs를 입력으로하여 순전파
        loss = self.softmax.forward(score, decoder_ts) #손실함수에 통과시켜 loss 구하기
        return loss

    def backward(self, dout=1): # 순서대로 역전파
        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) #encoder에 통과시켜 h를 구하고
        sampled = self.decoder.generate(h, start_id, sample_size) #문자를 생성시킨다.
        return sampled
```

encoder와 decoder를 연결하기만 하면 됨.

```
| 에폭 1 |  반복 301 / 351 | 시간 28[s] | 손실 1.74
| 에폭 1 |  반복 321 / 351 | 시간 30[s] | 손실 1.75
| 에폭 1 |  반복 341 / 351 | 시간 31[s] | 손실 1.74
Q 77+85
T 162
X 100
---
Q 975+164
T 1139
X 1000
---
Q 582+84
T 666
X 1000
---
Q 8+155
T 163
X 100
---
Q 367+55
T 422
X 1000
---
```

첫 에폭에서는 참값과 매우 다른 결과값

```
| 에폭 25 |  반복 301 / 351 | 시간 31[s] | 손실 0.76
| 에폭 25 |  반복 321 / 351 | 시간 33[s] | 손실 0.80
| 에폭 25 |  반복 341 / 351 | 시간 35[s] | 손실 0.79
Q 77+85
T 162
O 162
---
Q 975+164
T 1139
X 1136
---
Q 582+84
T 666
X 671
---
Q 8+155
T 163
X 166
---
```

25 에폭에서는 참값과 매우 근사한 결과값

2) Image Captioning 코드 구현 및 결과 확인 (코드 한줄씩 분석)

```python
from keras_applications.vgg16 import VGG16
from keras_preprocessing import image
from keras_applications.vgg16 import preprocess_input

class Encoder:
    def __init__(self, image):
        model = VGG16(weights='imagenet', include_top=False)

        img_path = image
        img = image.load_img(img_path, target_size=(224, 224))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)
        x = preprocess_input(x)

        features = model.predict(x)
        features = features.flatten()


        return features
```

Encoder를 CNN으로 변환하고 나오는 feature map을 평탄화 한 뒤 affine 계층을 통과시키고, 그 결과를 decoder의 입력으로 주면 Image Captioning이 구현가능하다. 책에 별도의 코드가 없어 keras에서 학습되어 있는 vgg16을 사용하여 encoder를 만들어보려 했지만, decoder의 인수로 무엇을 주어야하는지 등이 정리가 되지 않아 더 이상 구현을 할 수 없었다. vgg16의 return 값으로 어떤 모양의 행렬이 만들어지는지, 평탄화 했을 때의 모양, affine 계층을 통과시킬 때 어떻게 파라미터를 주어야 하는지 등을 곰곰이 다시 생각해보아야겠다.

고찰

1. 계층을 레고처럼 끼워맞추면 된다고 했지만, 생각처럼 행렬의 형상을 생각하거나 파라미터를 설정하는 일이 쉽지가 않았다.
2. affine 계층 같은 기본 계층도 그냥 import 하여 쓰다보니 역할과 사용법이 헷갈린다. 다시 공부해야겠다.