연구윤리 및 연구지도
(6주차 - time LSTM)

경북대학교 전자공학부
2016113566 김남영

1) Time LSTM 계층 구현 (코드 한줄씩 분석)

```python
class TimeLSTM:
    def __init__(self, Wx, Wh, b, stateful=False): #Truncated BPTT 역전파를 이용하기 위해 stateful 이용
        self.params = [Wx, Wh, b] # 가중치 매개변수
        self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)] #기울기 초기화
        self.layers = None

        self.h, self.c = None, None
        self.dh = None
        self.stateful = stateful # Truncated BPTT를 위해 상태정보 저장

    def forward(self, xs): # 순전파, Xs = [X0, X1, ... , Xt-1]을 의미(한번에 입력)
        Wx, Wh, b = self.params
        N, T, D = xs.shape
        H = Wh.shape[0]

        self.layers = []
        hs = np.empty((N, T, H), dtype='f') # 입력과 같은 형상의 출력 hs

        if not self.stateful or self.h is None: # 상태정보가 false이거나 h가 비어있다면,
            self.h = np.zeros((N, H), dtype='f') # 처음 시작한다는 말이므로,
        if not self.stateful or self.c is None: # h와 c를 zero값으로 초기화
            self.c = np.zeros((N, H), dtype='f')

        for t in range(T): # 시계열 데이터의 크기만큼 반복
            layer = LSTM(*self.params) #LSTM 계층 생성 * T개 만큼
            self.h, self.c = layer.forward(xs[:, t, :], self.h, self.c)
            #이전에 저장된 h와 c값(처음이라면 zero), 그리고 입력 xs를 넣어 순전파
            hs[:, t, :] = self.h # 출력 변수에 저장

            self.layers.append(layer)

        return hs
```

```python
    def backward(self, dhs): # 역전파, 이전에서 넘어온 dhs가 인수
        Wx, Wh, b = self.params
        N, T, H = dhs.shape
        D = Wx.shape[0]

        dxs = np.empty((N, T, D), dtype='f') # 기울기를 구해서 다음 계층으로 넘기기 위한 그릇
        dh, dc = 0, 0

        grads = [0, 0, 0]
        for t in reversed(range(T)): #요소를 뒤집어서 거꾸로 만들고(reversed)
            layer = self.layers[t] # 뒤쪽 계층부터 시작
            dx, dh, dc = layer.backward(dhs[:, t, :] + dh, dc) # 순전파 때 분기했으므로 역전파 때 합산
            dxs[:, t, :] = dx # 다음 계층으로 넘길 기울기
            for i, grad in enumerate(layer.grads): # enumerate는 인덱스값 까지 리턴
                grads[i] += grad #

        for i, grad in enumerate(grads): # enumerate는 인덱스값까지 리턴함
            self.grads[i][...] = grad # 각 계층 기울기 합산해서 최종 기울기를 저장
        self.dh = dh
        return dxs # 하류로 내보내지는 기울기

    def set_state(self, h, c=None):
        self.h, self.c = h, c

    def reset_state(self):
        self.h, self.c = None, None
```

2) Time LSTM 계층을 활용한 신경망 구성 및 결과확인 (코드 한줄씩 분석)

```python
class Rnnlm:
    def __init__(self, vocab_size=10000, wordvec_size=100, hidden_size=100):
        V, D, H = vocab_size, wordvec_size, hidden_size
        rn = np.random.randn

        # 가중치를 랜덤으로 초기화
        embed_W = (rn(V, D) / 100).astype('f') # 단어를 벡터로 변환하는 embedded층, 단어 입력만큼 V*D 형상
        lstm_Wx = (rn(D, 4 * H) / np.sqrt(D)).astype('f') # 가중치 Wx와 Wh
        lstm_Wh = (rn(H, 4 * H) / np.sqrt(H)).astype('f')
        lstm_b = np.zeros(4 * H).astype('f') # 편향
        affine_W = (rn(H, V) / np.sqrt(H)).astype('f') # affine 계층 생성
        affine_b = np.zeros(V).astype('f')

        # 계층 생성
        self.layers = [ #순서대로 생성
            TimeEmbedding(embed_W),
            TimeLSTM(lstm_Wx, lstm_Wh, lstm_b, stateful=True), # Truncated BPTT를 위해
            TimeAffine(affine_W, affine_b)
        ]
        self.loss_layer = TimeSoftmaxWithLoss() # softmaxwithLoss층은 따로 생성
        self.lstm_layer = self.layers[1] # reset_state를 위해

        # 모든 가중치와 기울기를 리스트에 모음
        self.params, self.grads = [], [] # 가중치와 기울기를 담을 리스트 생성
        for layer in self.layers:
            self.params += layer.params # 가중치 다 더해서 인스턴트 변수에 저장
            self.grads += layer.grads # 기울기 다 더해서 인스턴트 변수에 저장

    def predict(self, xs):
        for layer in self.layers:
            xs = layer.forward(xs) #timesoftmaxwithLoss 층을 제외하고 순전파
        return xs

    def forward(self, xs, ts):
        score = self.predict(xs) #loss_layer 이전까지 다 구해서 score에 저장
        loss = self.loss_layer.forward(score, ts)
        return loss

    def backward(self, dout=1):
        dout = self.loss_layer.backward(dout) # loss_layer 우선 역전파 실시
        for layer in reversed(self.layers): #reversed해서 거꾸로 진행
            dout = layer.backward(dout) # 거꾸로 역전파 진행
        return dout

    def reset_state(self): # 신경망 상태를 초기화
        self.lstm_layer.reset_state()
```