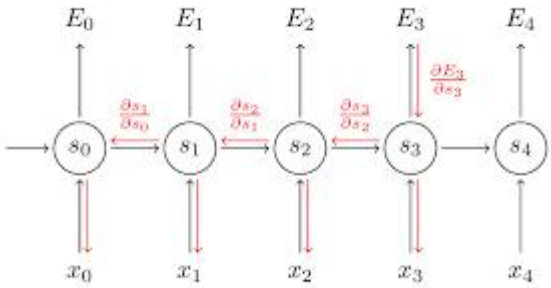
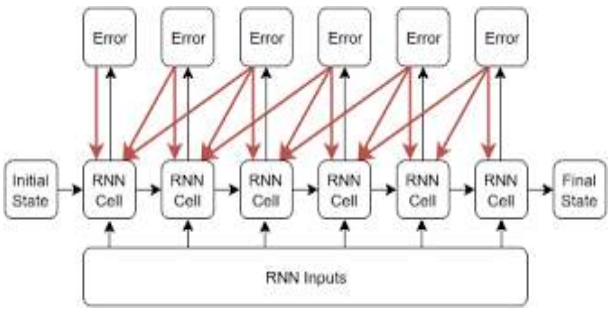
연구윤리 및 연구지도

(4주차 - 구현)

경북대학교 전자공학부
2016113566 김남영

1) BPTT와 Truncated BPTT를 정리하고 둘 사이의 차이를 간단한 예제를 통해 설명.





BPTT(Backpropagation Through Time): 위 왼쪽 그림처럼 시간 방향으로 펼친 신경망을 학습 하기위해 오차역전파법을 수행하는 것을 말한다. 문제는 시계열 데이터를 처리할 때는 중간 데이터를 계속 메모리에 유지해두어야 하므로, 시계열 데이터의 크기가 매우 클 때 사용하는 메모리 사용량이 커진다. 또한 시간 크기에 비례해서 기울기가 불안정해지는 문제도 있다.

Truncated BPTT(Truncated Backpropagation Through Time): BPTT의 문제점을 해결하기 위해 위 오른쪽 그림과 같이 여러 RNN 계층을 하나의 cell(또는 block) 단위로 묶어서 사용하는 방법을 사용한다. 즉, 순전파의 연결은 그대로 유지하나, 역전파의 연결을 cell 단위로 잘라내어 잘라낸 신경망 단위로 학습을 수행한다.

```python
class RNN: # RNN 계층 하나
    def __init__(self, Wx, Wh, b): # 초기화
        self.params = [Wx, Wh, b]
        self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]
        self.cache = None

    def forward(self, x, h_prev): # 이전에 저장된 h값을 이용해서 순전파
        Wx, Wh, b = self.params
        t = np.dot(h_prev, Wh) + np.dot(x, Wx) + b
        h_next = np.tanh(t) # 순전파 공식에 의해

        self.cache = (x, h_prev, h_next) # 역전파 때 사용하려고 저장.
        return h_next

    def backward(self, dh_next): # 상류에서 받은 dh_next를 이용하여 역전파
        Wx, Wh, b = self.params
        x, h_prev, h_next = self.cache

        dt = dh_next * (1 - h_next ** 2)
        db = np.sum(dt, axis=0)
        dWh = np.dot(h_prev.T, dt)
        dh_prev = np.dot(dt, Wh.T)
        dWx = np.dot(x.T, dt)
        dx = np.dot(dt, Wx.T) # 순전파 공식을 역전파 공식으로 바꾼 것.

        self.grads[0][...] = dWx # 기울기 계산하여 저장
        self.grads[1][...] = dWh
        self.grads[2][...] = db

        return dx, dh_prev # 하류로 보내는 정보
```

위 코드는 BPTT 구조를 따르는 RNN 계층 하나에 대한 코드이다. 역전파를 위해 cache 값에 이전 계층으로부터 받았던 정보와 다음 계층으로 줄 정보가 저장된다. 시계열 데이터의 길이가 길어질수록 기억해야할 정보가 많아지므로 메모리와 컴퓨팅 자원의 소모량이 늘어난다.

```python
class TimeRNN: # 위 RNN 계층을 여러개 이은 것, 즉 Truncated BPTT 구조에서 하나의 cell
    def __init__(self, Wx, Wh, b, stateful=False): # Stateful을 이용해서 은닉을 유지할지 결정
        self.params = [Wx, Wh, b]
        self.grads = [np.zeros_like(Wx), np.zeros_like(Wh), np.zeros_like(b)]
        self.layers = None

        self.h, self.dh = None, None
        self.stateful = stateful

    def forward(self, xs): # 여러개의 입력을 묶은 것
        Wx, Wh, b = self.params
        N, T, D = xs.shape
        D, H = Wx.shape

        self.layers = []
        hs = np.empty((N, T, H), dtype='f')

        if not self.stateful or self.h is None: #self.h가 None 인 것은 처음 호출이라는 뜻
            self.h = np.zeros((N, H), dtype='f') #첫 호출이거나 stateful이 false이면 영행렬로 초기화

        for t in range(T): # T개의 RNN 계층을 만들어 하나의 블록을 만듦
            layer = RNN(*self.params)
            self.h = layer.forward(xs[:, t, :], self.h) # T만큼이 한 묶음이니 오프셋을 t만큼 준 것
            hs[:, t, :] = self.h
            self.layers.append(layer)

        return hs

    def backward(self, dhs):
        Wx, Wh, b = self.params
        N, T, H = dhs.shape
        D, H = Wx.shape

        dxs = np.empty((N, T, D), dtype='f')
        dh = 0
        grads = [0, 0, 0]
        for t in reversed(range(T)):
            layer = self.layers[t]
            dx, dh = layer.backward(dhs[:, t, :] + dh)
            # 역전파시 RNN의 입력으로 dhs와 dhnext가 합쳐서 들어오므로
            dxs[:, t, :] = dx #dxs가 하류로 흘리는 기울기

            for i, grad in enumerate(layer.grads): #enumerate는 순서가 있는 객체를 순서대로 리턴
                grads[i] += grad

        for i, grad in enumerate(grads):
            self.grads[i][...] = grad
        self.dh = dh

        return dxs
```

위 코드는 T 길이 만큼의 RNN 계층을 묶어 하나의 cell로 만든 것이다. 따라서 순전파시, 여러 RNN계층을 거친 후 최종 h값이 리턴되고, 역전파시, 여러 RNN계층을 거슬러 올라간 후 최종 dxs값(기울기값)이 리턴된다. RNN 계층 하나와 구별되는 점은 cell의 상태를 나타내는 stateful 변수가 존재한다는 것이다. stateful이 True일 경우 은닉상태를 이어가겠다는 뜻으로, 다음 timeRNN 계층에 마지막 계산된 h값을 전달한다. 즉, stateful로 인해 Truncated BPTT 구조로 역전파가 가능한 것이다.

2) 시계열 데이터 처리 계층 구현.

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

        # 가중치 초기화 => 랜덤으로 설정
        embed_W = (rn(V, D) / 100).astype('f') # 단어 갯수(V)만큼 embed 층이 생성되므로 V*D 형상
        rnn_Wx = (rn(D, H) / np.sqrt(D)).astype('f') # D * H 형상
        rnn_Wh = (rn(H, H) / np.sqrt(H)).astype('f') # H * H 형상
        rnn_b = np.zeros(H).astype('f')
        affine_W = (rn(H, V) / np.sqrt(H)).astype('f') # 단어 갯수 만큼 값이 나와야하므로 H*V 형상
        affine_b = np.zeros(V).astype('f')

        # 여러 층을 묶어 다 time층으로 만듦
        self.layers = [
            TimeEmbedding(embed_W), # embedding 계층은 입력단어를 벡터로 바꾸는 층
            TimeRNN(rnn_Wx, rnn_Wh, rnn_b, stateful=True), #True이므로 계승가능
            TimeAffine(affine_W, affine_b)
        ]
        self.loss_layer = TimeSoftmaxWithLoss()
        self.rnn_layer = self.layers[1] # timeRNN 계층 정보를 따로 저장

        # 모든 가중치와 기울기를 리스트에 모은다.
        self.params, self.grads = [], []
        for layer in self.layers:
            self.params += layer.params
            self.grads += layer.grads

    def forward(self, xs, ts):
        for layer in self.layers:
            xs = layer.forward(xs)
        loss = self.loss_layer.forward(xs, ts)
        return loss

    def backward(self, dout=1):
        dout = self.loss_layer.backward(dout)
        for layer in reversed(self.layers):
            dout = layer.backward(dout)
        return dout
```

(책 예제)

책 예제에 있는 언어 데이터를 처리할 수 있는 모델이다. 이러한 모델을 실제로 학습할 때는, time size를 설정해 Truncated BPTT가 한번에 펼치는 시간의 크기(한 cell에서 RNN 개수)를 정하고, 그에 따라 데이터에 오프셋을 주어 순서대로 입력한다. 결론적으로는 에폭마다 손실을 구해 평균을 내어 퍼플렉시티를 구한다.

고찰

1. 어떤 모델이든 잘 만들어놓은 계층을 레고처럼 조립하는 것이 핵심이라는게 점점 코드에서 느껴진다. 하지만 내가 아무것도 참고하지 않고 저런 모델을 만들 수 있을지는 모르겠다.