딥러닝 응용 I / 중간고사 7. 3. 2020

1. 다음은 전처리 코드이다. 출력될 5개를 순서대로 쓰시오.

```
text = 'You say goodbye and I say hello.'
text = text.lower()
print(text)

text = text.replace('.', ' .')
print(text)

words = text.split(' ')
print(words)

word_to_id = {}
id_to_word = {}

for word in words:
    if word not in word_to_id:
        new_id = len(word_to_id)
        word_to_id[word] = new_id
        id_to_word[new_id] = word

corpus = np.array([word_to_id[w] for w in words])
print(corpus)
print(word_to_id)
```

2. (1번 문제 계속)
   (i) 어휘들의 희소 표현(sparse representation)을 통해 말뭉치(corpus)를 행렬로 표현하시오.
   (ii) 윈도우 크기를 2로 잡았을 때 말뭉치(corpus)의 동시발생 행렬(co-occurrence matrix)을 구하시오.
   (iii) 위 희소표현과 동시발생 행렬 각각에 대해 You와 I의 코사인 유사도(cosine similarity) 값을 구하시오.

3. (1번 문제 코드 계속) 출력될 행렬과 벡터를 순서대로 쓰시오.

```
window_size=2
target = corpus[window_size:-window_size]
contexts = []

for idx in range(window_size, len(corpus)-window_size):
    cs = []
    for t in range(-window_size, window_size + 1):
        if t == 0:
            continue
        cs.append(corpus[idx + t])
    contexts.append(cs)

print(np.array(contexts))
print(np.array(target))
```

§ 4,5,8번 문제의 말뭉치는

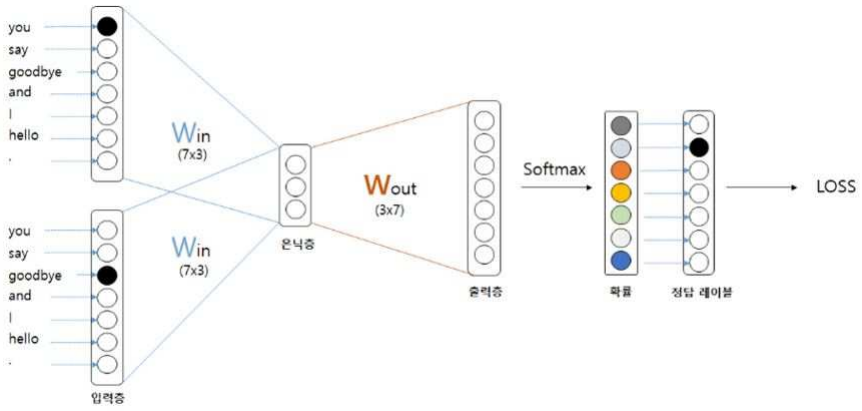
You say goodbye and I say hello.

이고 1번 문제의 코드에 의해 전처리 되어 있다. 윈도우 크기는 1이다. 가중치 행렬은

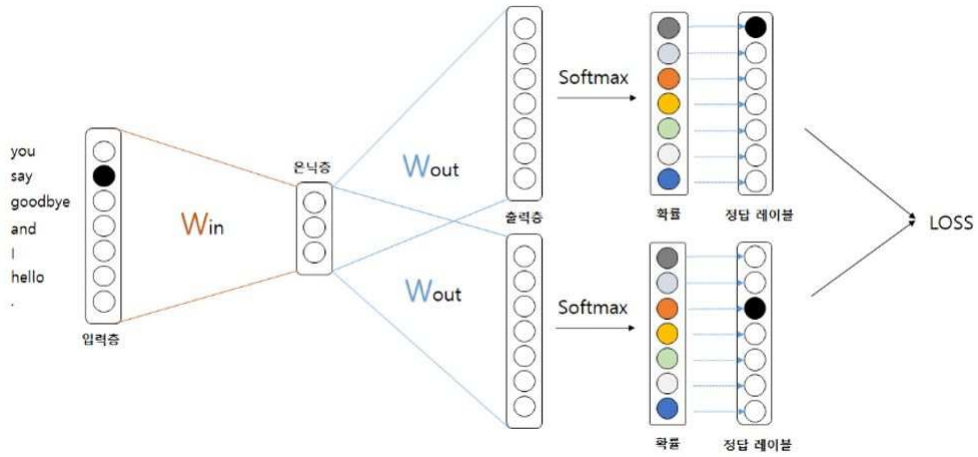
$$W_{\text{in}} = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \\ 4 & 5 & 6 \\ 5 & 6 & 7 \\ 6 & 7 & 8 \\ 7 & 8 & 9 \end{pmatrix}, \qquad W_{\text{out}} = \begin{pmatrix} 0 & -\frac{1}{2}\log 2 & -\log 2 & 2\log 2 & 2\log 2 & 2\log 2 & 2\log 2 \\ -\log 2 & -\frac{1}{3}\log 2 & \log 2 & 0 & 0 & 0 & 0 \\ \log 2 & \log 2 & 0 & -\log 2 & -\log 2 & -\log 2 & -\log 2 \end{pmatrix} \tag{1}$$

이다.

4. 가중치 행렬 W_{in}과 W_{out}이 (1)인 다음과 같은 simple CBOW 모델을 생각하자. 첫번째 맥락(context)인 (you, goodbye)를 입력했을때 출력되는 손실함수 값을 구하시오.



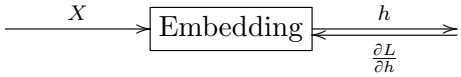
5. 가중치 행렬 W_{in}과 W_{out}이 (1)인 다음과 같은 simple skip-gram 모델을 생각하자. 첫번째 타겟(target)인 say를 입력했을때 출력되는 손실함수 값을 구하시오.



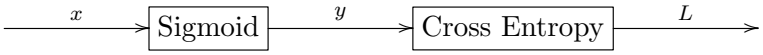
6. Embedding층의 가중치 행렬 W, 입력되는 단어 id의 배치 묶음 X, 위에서 흘러들어오는 미분 $\frac{\partial L}{\partial h}$이 각각

$$W = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{pmatrix}, \qquad X = [0, 3, 0, 3], \qquad \frac{\partial L}{\partial h} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

일 때, 출력값 h와 가중치 행렬에 대한 미분 $\frac{\partial L}{\partial W}$을 각각 구하시오.



7. Sigmoid-with-Loss 계층

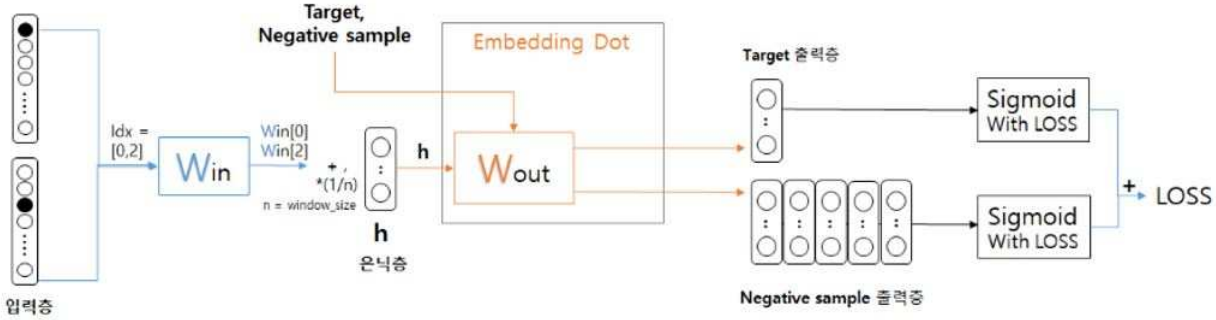


의 역전파는

$$\frac{\partial L}{\partial x} = y - t$$

로 주어짐을 미분법을 이용하여 보이시오. (x, y, t는 모두 스칼라이고 t는 No일때 0, Yes일때 1인 라벨)

8. 가중치 행렬 W_{in}과 W_{out}이 (1)인 다음과 같은 완전판 CBOW 모델을 생각하자. 첫번째 context인 (you, goodbye)를 입력하고 negative sampling한 세 단어가 you와 I와 hello였을 때 출력되는 손실함수 값을 구하시오.



9. 다음은 negative sampling을 구현하는 클래스이다. 5개의 빈칸을 순서대로 채우시오.

```
class UnigramSampler:
    def __init__(self, corpus, power, sample_size):
        self.sample_size = sample_size
        self.vocab_size = None
        self.word_p = None
        counts = collections.Counter()
        for word_id in corpus:
            counts[word_id] += 1
        vocab_size = len(counts)
        self.vocab_size = vocab_size
        self.word_p = np.zeros(vocab_size)
```

```
        for i in range(vocab_size):
            self.word_p[i] = counts[i]
        self.word_p = np.[ ](self.word_p, power)
        self.word_p [ ] np.sum(self.word_p)

    def get_negative_sample(self, target):
        batch_size = target.shape[0]
        negative_sample = np.zeros((batch_size, self.sample_size), dtype=np.int32)
        for i in range(batch_size):
            p = self.word_p.copy()
            target_idx = target[i]
            p[target_idx] = [ ]
            p [ ] p.sum()
        negative_sample[i, :] = np.random.[ ](self.vocab_size, size=self.sample_size,
                                              replace=False, p=p)
        return negative_sample
```

10. 다음은 완전판 CBOW 모델을 구현하는 코드이다. 5개의 빈칸을 순서대로 채우시오.

```
class CBOW:
    def __init__(self, vocab_size, hidden_size, window_size, corpus):
        V, H = vocab_size, hidden_size
        W_in = 0.01 * np.random.randn(V, H).astype('f')
        W_out = 0.01 * np.random.randn(V, H).astype('f')
        self.in_layers = []
        for i in range([ ]):
            layer = Embedding(W_in)
            self.in_layers.append(layer)
        self.ns_loss = NegativeSamplingLoss(W_out, corpus, power=0.75, sample_size=5)
        layers = self.in_layers + [self.ns_loss]
        self.params, self.grads = [], []
        for layer in layers:
            self.params += layer.params
            self.grads += layer.grads
        self.word_vecs = [ ]

    def forward(self, contexts, target):
        h = 0
        for i, layer in [ ](self.in_layers):
            h += layer.forward(contexts[:, i])
        h *= [ ]
        loss = self.ns_loss.forward(h, target)
        return loss

    def backward(self, dout=1):
        dout = self.ns_loss.backward(dout)
        dout *= [ ]
        for layer in self.in_layers:
            layer.backward(dout)
        return None
```

딥러닝 응용 I / 기말고사 7. 3. 2020

1. 다음 쌍곡함수에 관한 공식을 유도하시오.

$$\cosh^2 x - \sinh^2 x = 1, \qquad \mathrm{sech}^2 x = 1 - \tanh^2 x$$
$$\frac{d}{dx}\sinh x = \cosh x, \qquad \frac{d}{dx}\cosh x = \sinh x, \qquad \frac{d}{dx}\tanh x = \mathrm{sech}^2 x$$

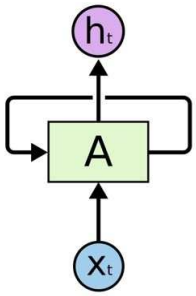
2. 데이터에 대한 가중치 행렬 W_x, hidden state에 대한 가중치 행렬 W_h, 편향 벡터 b가 각각

$$W_x = \begin{pmatrix} \log 2 & \log 2 \\ \log 4 & \log 6 \end{pmatrix}, \qquad W_h = \begin{pmatrix} -5\log 2 & -5\log 3 \\ 5\log 2 & 5\log 3 \end{pmatrix}, \qquad b = [0, 0]$$

인 다음과 같은 RNN층을 생각하자. 최초 시각 $t = 0$과 그 다음 시각 $t = 1$에서 입력된 데이터 x_0와 x_1이

$$x_0 = [-1, 1], \qquad x_1 = [2, -1]$$

이라 하자. 시각 $t = 0$과 $t = 1$일 때 만들어지는 hidden state h_0와 h_1을 구하시오.



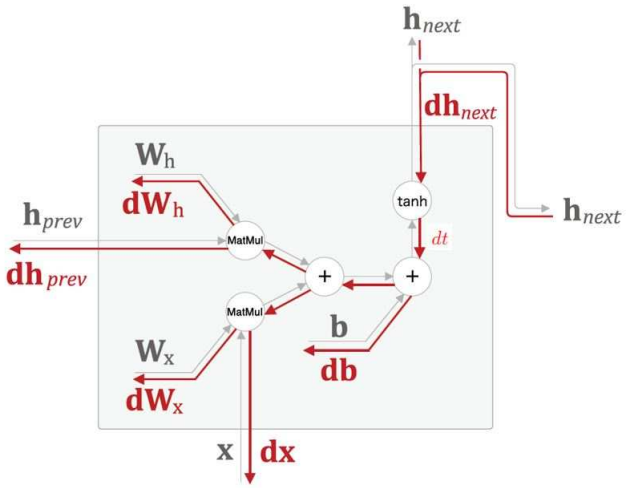
3. (2번 문제 계속) 다음은 RNN층의 계산 그래프이다. 시각 $t = 1$에서 흘러들어온 hidden state에 관한 두 미분의 합이

$$\frac{\partial L}{\partial h_1} = [\frac{125}{16}, 0]$$

일 때, 계산 그래프를 이용하여 미분

$$\frac{\partial L}{\partial W_h}, \qquad \frac{\partial L}{\partial h_0}, \qquad \frac{\partial L}{\partial W_x}, \qquad \frac{\partial L}{\partial x_1} \qquad \frac{\partial L}{\partial b}$$

의 값을 구하시오.



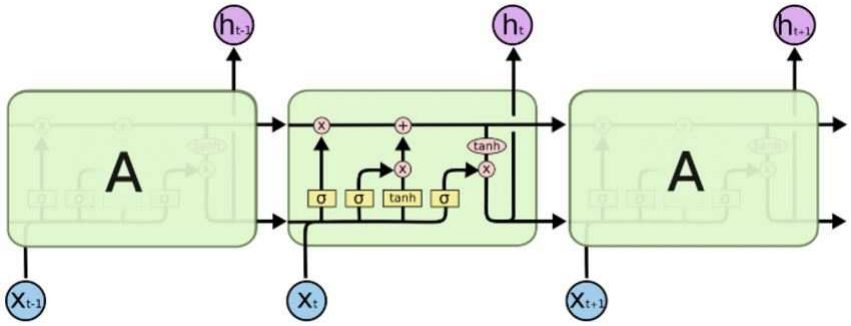
4. 데이터에 대한 가중치 행렬 W_x, hidden state에 대한 가중치 행렬 W_h, 편향 벡터 b가 각각

$$W_x^{(f)} = W_x^{(i)} = W_x^{(g)} = W_x^{(o)} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$
$$W_h^{(f)} = W_h^{(i)} = W_h^{(g)} = W_h^{(o)} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},$$
$$b^{(f)} = b^{(i)} = b^{(g)} = b^{(o)} = [0, 0]$$

인 다음과 같은 LSTM층을 생각하자. 여기서 윗첨자 $(f), (i), (g), (o)$는 각각 forget gate, input gate, RNN 변환, output gate의 parameter임을 의미한다. 입력된 데이터 x_t와 전 시각의 LSTM층에서 들어온 hidden state h_{t-1}, cell state c_{t-1}이

$$x_t = [\log 2, 0], \qquad h_{t-1} = [0, \log 3], \qquad c_{t-1} = [\frac{9}{10}, \frac{4}{5}]$$

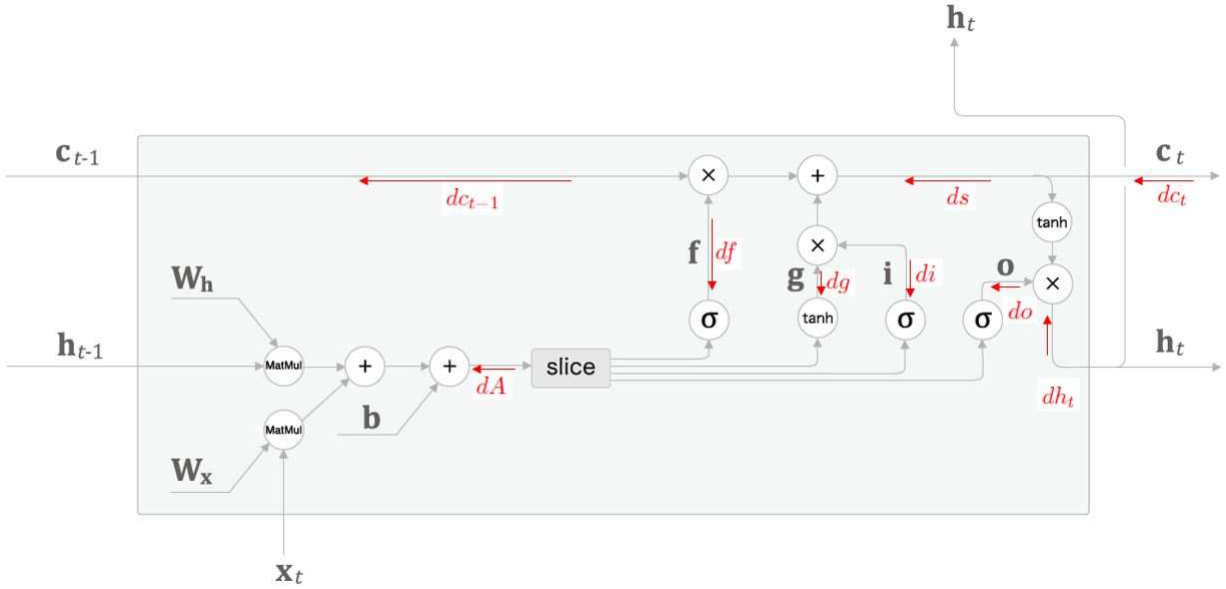
와 같을 때 다음 시각의 LSTM층에 전달할 hidden state h_t와 cell state c_t를 구하시오.



5. (4번 문제 계속) 다음은 LSTM층의 계산 그래프이다. 흘러들어온 hidden state에 관한 두 미분의 합 dh_t와 cell state에 관한 미분 dc_t가

$$dh_t = [0, 4], \qquad dc_t = [30, 19]$$

와 같을 때, 계산그래프를 이용하여 dc_{t-1}과 slice노드가 밑으로 흘려보내는 미분 dA 를 구하시오.



6. 다음은 RNN을 구현하는 클래스이다. 5개의 빈 칸을 순서대로 채우시오.

```
class RNN:
    def __init__(self, Wx, Wh, b):
        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):
        Wx, Wh, b = self.params
        t = np.dot(h_prev, Wh) + np.dot(x, Wx) + b
        h_next = np.[     ]
        self.cache = (x, h_prev, h_next)
        return h_next

    def backward(self, dh_next):
        Wx, Wh, b = self.params
        x, h_prev, h_next = self.cache
        dt = dh_next * [     ]
        db = np.sum(dt, [     ])
        dWh = np.dot([     ], dt)
        dh_prev = np.dot(dt, [     ])
        생략 ...
```

7. 다음은 truncated BTPP를 고려한 RNN을 구현하는 클래스이다. 4개의 빈칸에 공통으로 들어갈 코드를 쓰시오.

```
class TimeRNN:
    def __init__(self, Wx, Wh, b, stateful=False):
        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):
        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.h = np.zeros((N, H), dtype='f')
        for t in range(T):
            layer = RNN(*self.params)
            self.h = layer.forward(xs[     ], self.h)
            hs[     ] = 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[     ] + dh)
            dxs[     ] = dx
        for i, grad in enumerate(layer.grads):
            grads[i] += grad
        for i, grad in enumerate(grads):
            self.grads[i][...] = grad
        self.dh = dh
        return dxs
```

8. 다음은 LSTM을 구현하는 클래스이다. 5개의 빈칸을 순서대로 채우시오.

```
class LSTM:
    def __init__(self, Wx, Wh, b):
        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, c_prev):
        Wx, Wh, b = self.params
        N, H = h_prev.shape
        A = np.dot(x, Wx) + np.dot(h_prev, Wh) + b
        f = A[:, :H]
        g = A[:, H:2*H]
        i = A[:, 2*H:3*H]
        o = A[:, 3*H:]
        f = sigmoid(f)
        g = np.tanh(g)
        i = sigmoid(i)
        o = sigmoid(o)
        c_next = [     ]
        h_next = [     ]
        self.cache = (x, h_prev, c_prev, i, f, g, o, c_next)
        return h_next, c_next

    def backward(self, dh_next, dc_next):
        Wx, Wh, b = self.params
        x, h_prev, c_prev, i, f, g, o, c_next = self.cache
        tanh_c_next = np.tanh(c_next)
        ds = dc_next + (dh_next * o) * (1 - tanh_c_next ** 2)
        dc_prev = ds * f
        di = ds * g
        df = ds * c_prev
```

```
do = dh_next * tanh_c_next
dg = ds * i
di *= i * (1 - i)
df *= f * (1 - f)
do *= ____
dg *= ____
dA = ____((df, dg, di, do))
생략 ...
```

9. 다음은 2층 LSTM, dropout, 가중치 공유(weight tying)로 RNN 언어모델을 개선한 클래스의 초기화 코드이다. 가중치 공유(weight tying)를 염두에 두고 1개의 빈칸을 채우시오.

```
class BetterRnnlm(BaseModel):
    def __init__(self, vocab_size=10000, wordvec_size=650,
                 hidden_size=650, dropout_ratio=0.5):
        V, D, H = vocab_size, wordvec_size, hidden_size
        rn = np.random.randn
        embed_W = (rn(V, D) / 100).astype('f')
        lstm_Wx1 = (rn(D, 4 * H) / np.sqrt(D)).astype('f')
        lstm_Wh1 = (rn(H, 4 * H) / np.sqrt(H)).astype('f')
        lstm_b1 = np.zeros(4 * H).astype('f')
        lstm_Wx2 = (rn(H, 4 * H) / np.sqrt(H)).astype('f')
        lstm_Wh2 = (rn(H, 4 * H) / np.sqrt(H)).astype('f')
        lstm_b2 = np.zeros(4 * H).astype('f')
        affine_b = np.zeros(V).astype('f')
        self.layers = [
            TimeEmbedding(embed_W),
            TimeDropout(dropout_ratio),
            TimeLSTM(lstm_Wx1, lstm_Wh1, lstm_b1, stateful=True),
            TimeDropout(dropout_ratio),
            TimeLSTM(lstm_Wx2, lstm_Wh2, lstm_b2, stateful=True),
            TimeDropout(dropout_ratio),
            TimeAffine(____, affine_b)
        ]
        self.loss_layer = TimeSoftmaxWithLoss()
        self.lstm_layers = [self.layers[2], self.layers[4]]
        self.drop_layers = [self.layers[1], self.layers[3], self.layers[5]]
        self.params, self.grads = [], []
        for layer in self.layers:
            self.params += layer.params
            self.grads += layer.grads
```

10. 다음은 seq2seq를 구현하는 클래스이다. 2개의 빈칸을 순서대로 채우시오.

```
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[:, [redacted]], ts[:, [redacted]]
        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
```