Natural Language Processing: Project 2

Seung-Hoon Na

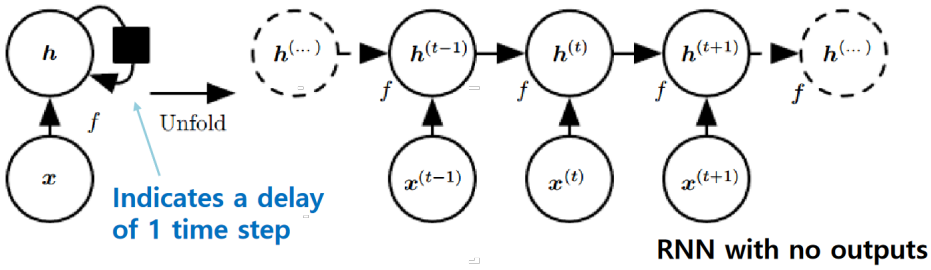
May 12, 2024

1 Recurrent Neural Networks

본 주제에서는 Recurrent Neural Networks (RNN)을 위한 이론적인 내용을 정리하고 `numpy`로 구현해보는 것을 목표로 한다.

RNN은 시퀀스 데이터를 처리하는데 효과적인 뉴럴 구조이며, 자연언어는 단어열로 구성된 대표적인 시퀀스 데이터로 딥러닝 초장기에는 대부분 RNN을 이용하여 자연어처리 태스크를 수행했었다. RNN은 입력이 Forward과정을 통해 다중 레이어 (multi-layer)를 거쳐 최종 아웃풋이 출력되는 Feed-forward network (FFN)과 다르게, 특정 레이어상에서의 activation 값 (표상)이 이전 레이어로 회귀하는 recurrent connection이 있는 neural network계열을 말한다.

다음 그림은 output이 없는 RNN구조로, t번째 입력과 이전 은닉 상태로부터 현재 t번째 은닉 상태가 $\mathbf{h}^{(t)} = g\left(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}\right)$의 식을 통해 업데이트가 되는 것을 보여주고 있다. 왼쪽이 recurrent connection이 있는 계산 구조를, 오른쪽은 recurrent connection을 unfolding하여 실제 시퀀스의 각 입력이 시간별로 주어질 때 forward계산 과정을 보다 상세히 보여주고 있다.

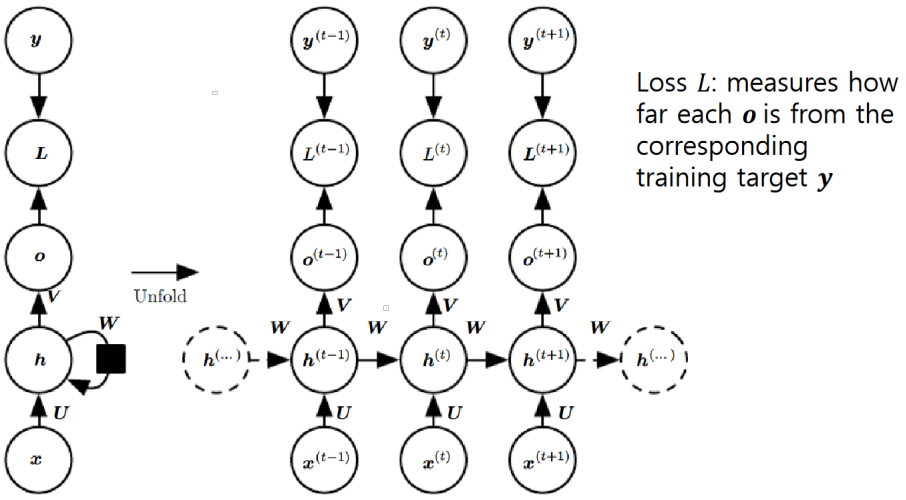


RNN은 시퀀스를 받아들이므로, 매 순간마다 새로운 입력이 들어오게 되며, 위 그림의 recurrent connection의 경우 "1 time step의 지연"라는 의미는 은닉층의 표상이 이전 레이어로 회귀할때 새로운 입력과 합류하는 시점이 바로 그 다음 시점이라는 뜻이다. 오른쪽 그림에서 보여지듯이, t시점의 입력과 $t-1$의 은닉상태 표상이 결합된, $(\mathbf{h}^{(t-1)}; \mathbf{x}^{(t)})$이 t번째 시점에 hidden layer가 받아들이는 총 입력 표상이 된다.

RNN은 위의 순환 구조를 통해 주어진 문장이나 단락의 입력을 모두 받아들인 후에 최종 시점의 은닉 상태 표상(hidden state representation)를 전체 입력에 대한 요약적 벡터가 간주하고, sentiment classification과 같은 "**문장 분류**"를 수행할 수 있으며, 또는 입력 시퀀스의 각각의 t번째 시점별로, 출력을 생성하도록 함으로써 각 단어별 태그를 예측하는 "**순차 태깅**"태스크도 수행할 수 있다.

1.1 Classical RNN

순차 태깅을 수행할 수 있는 classical RNN구조는 다음과 같다.



이때, 단순성을 위해, 순차 태깅은 POS tagging과 같이 태깅 결과가 분류범주에서 속하는 이산적인 (discrete)한 케이스로 한정되었다고 가정하자[1].

입력문장이 $sent = w_1, \cdots, w_T$로 주어졌다고 하자. 먼저 **word embedding** 레이어에서는, 문장 내 각 단어에 대한 Word embedding 벡터를 $\mathsf{Emd}(w_t) \in \mathbb{R}^d$로 가져온 후, T개의 단어 임베딩 벡터로 구성된 입력시퀀스 $\mathbf{X} \in \mathbb{R}^{d \times T}$를 아래와 같이 만들어낸다.

$$\mathbf{X} = [\mathsf{Emd}(w_1), \cdots, \mathsf{Emd}(w_T)] \tag{1}$$

이때, $\mathsf{Emd}(w)$는 단어 w의 word embedding vector를 리턴하는 함수이며, $\mathcal{V}$를 단어들의 전체 집합, 그리고 **word embedding을 위한 parameter matrix**를 $\mathbf{E} \in \mathbb{R}^{d \times |\mathcal{V}|}$라고 표기 할때, $\mathsf{Emd}(w)$는 다음과 같이 표현할 수 있다.

$$\mathsf{Emd}(w) = \mathbf{E} \cdot \mathsf{onehot}(w) \tag{2}$$

여기서, $\mathsf{onehot}(w) \in \mathbb{R}^{|\mathcal{V}|}$는 단어 w의 인덱스만 1이고 나머지는 0으로 구성된 one-hot vector이다.

위의 그림의 classical RNN의 시퀀스 입력이 단어임베딩 열인 $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_T]$이라고 하면, classical RNN의 update되는 식은 다음과 같다.

$$\begin{aligned} \mathbf{z}^{(t)} &= \mathbf{U}\mathbf{x}^{(t)} + \mathbf{W}\mathbf{h}^{(t-1)} \\ \mathbf{h}^{(t)} &= f\left(\mathbf{z}^{(t)}\right) \\ \mathbf{o}^{(t)} &= \mathsf{softmax}\left(\mathbf{V}\mathbf{h}^{(t)}\right) \end{aligned} \tag{3}$$

[1]가령, 주식예측의 경우 output이 연속적인 값을 갖는 경우라고 할 수 있으며, 해당 경우는 순차 회귀 등으로 불릴 수 있을 것이다. 여기서 순차 태깅은 각 시점별로 '분류'를 행하는 시퀀스 태스크를 말한다.

여기서, $\mathbf{U} \in \mathbb{R}^{d_h \times d}$, $\mathbf{W} \in \mathbb{R}^{d_h \times d_h}$, $\mathbf{V} \in \mathbb{R}^{|\mathcal{Y}| \times d_h}$는 RNN의 **parameter weight matrices**들이며[2], $\mathcal{Y}$는 순차 태깅에서 출력으로 가능한 전체 태그들의 집합 (예: 품사 태깅에서는 모든 가능한 품사들의 집합)을 의미하고, f는 activation function 이다 (relu나 tanh 등 함수).

RNN기반 순차태깅에서는 매 시점마다 Loss가 발생하여, 입력 문장에 대한 Loss는 전체 시점에서 발생된 Loss들의 합이다. 이를 구하기 위해, 우선 학습 단계에서, 입력문장 $sent = w_1, \cdots, w_T$외에 추가로 정답(ground-truth) 시퀀스가 $label = l_1, \cdots, l_T$로 주어진다고 가정하자. 각 출력 레이블 l_t에 대한 one-hot 벡터를 $\mathbf{y}^{(t)} = \mathsf{onehot}(l_t) \in \{0,1\}^{|\mathcal{Y}|}$라고 하면, 결국 출력 시퀀스 벡터가 $\mathbf{Y} = \left[\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(T)}\right]$로 표현될 수 있다. 즉, 정리하면,

$$\begin{aligned} \mathbf{Y} &= [\mathsf{onehot}(l_1), \cdots, \mathsf{onehot}(l_T)] \\ &= \left[\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(T)}\right] \end{aligned} \tag{4}$$

입력 문장과 정답 $(sent, label)$에 대한 L는 다음과 같이 정리될 수 있다.

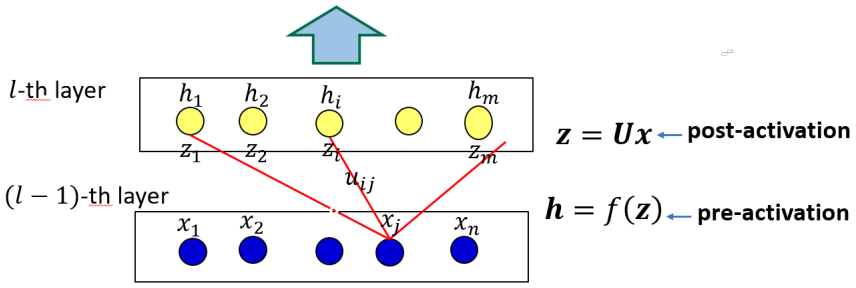
$$\begin{aligned} L_t &= -\mathbf{y}^{(t)} \cdot \mathsf{log}(\mathbf{o}^{(t)}) \\ L &= \sum_{t=1}^{T} L_t \end{aligned} \tag{5}$$

이때 L_t는 출력의 확률분포 $\mathbf{o}^{(t)}$의 정답 확률 분포 $\mathbf{y}^{(t)}$에 대한 **cross entropy** 이다.

본 과제에서는 classical RNN을 학습하기 위해서 backpropagation 알고리즘인 **BPTT** (backpropagation through time)을 먼저 유도할 것이다. 관련하여 다음 문항에 대해서 답하시오.

1. RNN의 BPTT를 유도하기 전에, FNN의 역전파 핵심 과정을 먼저 살펴보기 위해서, 다음 그림은 FNN의 layer의 표상이 이후 layer에 전달되는 forward의 한 step을 보여준다. 그림에서 보다시피, 이전 $l-1$-th layer의 activation 표상을 $\mathbf{x}$라고 할때, l-th layer의 preactivation과 post-activation표상은 각각 다음과 같이 얻어진다.

$$\begin{aligned} \mathbf{z} &= \mathbf{U}\mathbf{x} \\ \mathbf{h} &= f(\mathbf{z}) \end{aligned} \tag{6}$$



[2]Bias 항들은 생략해서 표현하였고, 입력벡터의 차원을 1개 확장하여 항상 1이 되도록 하여 weight matrix에 포함시켰다고 간주하자.

각 표상별로 Loss L에 대한 gradient를 δ_h, δ_z, δ_x라고 표기하고, 정의는 다음과 같다.

$$\begin{aligned}\delta_h &= \frac{\partial L}{\partial \mathbf{h}} \\ \delta_z &= \frac{\partial L}{\partial \mathbf{z}} \\ \delta_x &= \frac{\partial L}{\partial \mathbf{x}}\end{aligned} \tag{7}$$

δ_h가 주어졌다고 가정하고, δ_z와 δ_x을 각각 **chain-rule**을 통해 구하는 간략식을 기술하고, 유도과정을 상세히 보이시오.

2. 위의 문항에서 L의 $\mathbf{U}$에 대한 gradient인 $\frac{\partial L}{\partial \mathbf{U}}$을 **chain-rule**를 통해 간단히 표현하고, 그 유도과정을 상세히 보이시오.

$$\frac{\partial L}{\partial \mathbf{U}} = \tag{8}$$

3. 역전파 알고리즘에서는 output layer에 대한 Loss의 gradient가 먼저 계산이 되어야 한다. 특히, 분류 문제에서는 $\mathsf{softmax}$ 레이어를 거친 출력 확률분포와 정답 확률 분포간의 **cross entropy**를 loss로 취한다.

$$\begin{aligned}\mathbf{o} &= \mathsf{softmax}\left(\tilde{\mathbf{o}}\right) \\ L &= -\mathbf{y} \cdot \log\left(\mathbf{o}\right)\end{aligned} \tag{9}$$

여기서, $\mathbf{y} \in \{0,1\}^{|\mathcal{Y}|}$로 one-hot vector (합이 1이므로 확률 벡터라 볼 수 있다)이고, $\mathbf{o}$는 출력 확률 벡터라고 할때, 다음 L의 $\tilde{\mathbf{o}}$에 대한 gradient식의 유도과정을 명확하게 보이시오.

$$\frac{\partial L}{\partial \tilde{\mathbf{o}}} = \tilde{\mathbf{o}} - \mathbf{y} \tag{10}$$

4. classical RNN으로 돌아가서, 주어진 입력과 정답 $(sent, label)$이 주어지고, 그에 대한 loss가 Eq. 5라고 할때, 앞 세 문항의 역전파 일반 공식을 이용하여, BPTT를 위해 아래 각 parameter의 gradient를 구하는 공식들을 유도하시오.

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{E}} &= \\ \frac{\partial L}{\partial \mathbf{U}} &= \\ \frac{\partial L}{\partial \mathbf{W}} &= \\ \frac{\partial L}{\partial \mathbf{V}} &=\end{aligned}$$

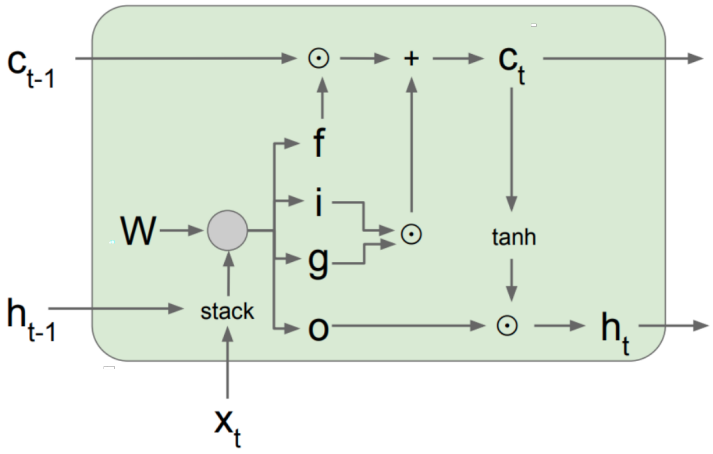
5. Eq. 11를 참조하여, 학습데이터셋 $\mathcal{D} = (sent_i, label_i)_{i=1}^{N}$ 으로 총 N개의 문장과 정답 레이블열로 구성되었다고 할때, stochastic gradient descent방법을 이용한 BPTT알고리즘의 pseudo code를 기술하시오.

6. Classical RNN의 Eq. (11)에 따라, 학습 알고리즘이 vanishing gradients또는 exploding gradients문제에 빠지는 이유를 기술하시오.

1.2 Long Short Term Memory (LSTM)

RNN은 시퀀스의 time축으로 unfolding된 구조로, 시퀀스의 길이만큼의 "deep" layers까지 forward가 진행되는 신경망이라 볼 수 있다. 그러나, Classical RNN의 BPTT알고리즘은 Vanishing gradient 등의 문제로 인해, 입력 시퀀스상에서 long term dependency를 모델링하는데 한계를 갖는다.

LSTM은 시퀀스상에서 받은 입력 정보들을 저장하고 삭제할 수 있는 별도의 **memory cell**을 둠으로써, 한참 이전의 입력 정보를 memory cell상에서 보관할 수 있도록 함으로써 구조적으로 long term dependency를 해결한 방법이다. 특히, 각각의 입력을 projection한 이후에 이들을 memory cell에 추가할 지, 그리고 기존의 memory cell정보 중 어떤 부분을 유지할 지 등을 모두 **gating mechanism**을 통해 정교하게 자동으로 제어하게 된다. 다음 그림은 LSTM에서 memory cell과 gating mechanism을 통해 은닉 상태 표상이 얻어지는 one-step 과정을 보여준다.



RNN의 순환형 forwarding과정을 일반적화여 g 함수를 사용하여 $\mathbf{h}^{(t)} = g\left(\mathbf{h}^{(t-1)}; \mathbf{x}^{(t)}\right)$ 라고 표현한다면, classical RNN은 g함수가 다음과 같다.

$$\begin{aligned} \mathbf{h}^{(t)} &= g_{RNN}\left(\mathbf{h}^{(t-1)}; \mathbf{x}^{(t)}\right) \\ &= f\left(\begin{bmatrix} \mathbf{W} & \mathbf{U} \end{bmatrix} \begin{bmatrix} \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(\mathbf{t})} \end{bmatrix}\right) \end{aligned} \tag{11}$$

LSTM의 g함수는 다음과 같이 기술할 수 있다.

$$
\begin{aligned}
\mathbf{h}^{(t)} &= g_{LSTM}\left(\mathbf{h}^{(t-1)}; \mathbf{x}^{(t)}\right) \\
&= \mathbf{r}^{(t)} \odot \mathbf{c}^{(t)} \\
\mathbf{i}^{(t)} &= \sigma\left(\begin{bmatrix} \mathbf{W}^{(i)} & \mathbf{U}^{(i)} \end{bmatrix} \begin{bmatrix} \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(\mathbf{t})} \end{bmatrix}\right) \\
\mathbf{f}^{(t)} &= \sigma\left(\begin{bmatrix} \mathbf{W}^{(f)} & \mathbf{U}^{(f)} \end{bmatrix} \begin{bmatrix} \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(\mathbf{t})} \end{bmatrix}\right) \\
\mathbf{r}^{(t)} &= \sigma\left(\begin{bmatrix} \mathbf{W}^{(r)} & \mathbf{U}^{(r)} \end{bmatrix} \begin{bmatrix} \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(\mathbf{t})} \end{bmatrix}\right) \\
\tilde{\mathbf{c}}^{(t)} &= f\left(\begin{bmatrix} \mathbf{W}^{(c)} & \mathbf{U}^{(c)} \end{bmatrix} \begin{bmatrix} \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(\mathbf{t})} \end{bmatrix}\right) \\
\mathbf{c}^{(t)} &= \mathbf{f}^{(t)} \odot \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \odot \tilde{\mathbf{c}}^{(t-1)}
\end{aligned} \tag{12}
$$

여기서, σ는 sigmoid함수이고, $\mathbf{i}^{(t)}$, $\mathbf{f}^{(t)}$, $\mathbf{r}^{(t)}$를 각각 **input gates**, **forget gates**, **output gates**라고 하고 [3], $\tilde{\mathbf{c}}^{(t)}$를 **new memory cell** 표상을 가리키며, $\mathbf{W}^{(i)}$, $\mathbf{U}^{(i)}$, $\mathbf{W}^{(f)}$, $\mathbf{U}^{(f)}$, $\mathbf{W}^{(r)}$, $\mathbf{U}^{(r)}$, $\mathbf{W}^{(c)}$, $\mathbf{U}^{(c)}$는 파라미터이다.

세개의 gates를 구하는 식을 다음과 같이 간결히 기술할 수도 있다.

$$
\begin{aligned}
\begin{bmatrix} \mathbf{i}^{(t)} \\ \mathbf{f}^{(t)} \\ \mathbf{r}^{(t)} \end{bmatrix} &= \sigma\left(\begin{bmatrix} \mathbf{W}^{(i)} & \mathbf{U}^{(i)} \\ \mathbf{W}^{(f)} & \mathbf{U}^{(f)} \\ \mathbf{W}^{(r)} & \mathbf{U}^{(r)} \end{bmatrix} \begin{bmatrix} \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(\mathbf{t})} \end{bmatrix}\right) \\
&= \sigma\left(\begin{bmatrix} \mathbf{W}^{(ifr)} & \mathbf{U}^{(ifr)} \end{bmatrix} \begin{bmatrix} \mathbf{h}^{(t-1)} \\ \mathbf{x}^{(\mathbf{t})} \end{bmatrix}\right)
\end{aligned} \tag{13}
$$

여기서, $\mathbf{W}^{(ifr)} \in \mathbb{R}^{3d_h \times d_h}$, $\mathbf{U}^{(ifr)} \in \mathbb{R}^{3d_h \times d}$는 세개의 gates에 대한 parameter matrices를 결합한 것이다.

1. 순차 태깅의 학습예제로 입력과 정답 $(sent, label)$쌍이 주어지고, 그에 대한 loss가 Eq. 5라고 할때, classical RNN에서 활용되었던 역전파 일반 공식을 참조하여, LSTM학습을 위한 BPTT알고리즘에서 각 parameter들의 gradient 구하는 공식들을 유도하시오.

[3] output gates의 경우 보통 첨자를 o를 이용하는데, 앞서 $\mathbf{o}^{(t)}$ 표현을 사용했기 때문에, **read gates**로 간주하고 r 첨자를 사용하였다

$$\frac{\partial L}{\partial \mathbf{E}} =$$
$$\frac{\partial L}{\partial \mathbf{U}^i} =$$
$$\frac{\partial L}{\partial \mathbf{W}^i} =$$
$$\frac{\partial L}{\partial \mathbf{U}^f} =$$
$$\frac{\partial L}{\partial \mathbf{W}^f} =$$
$$\frac{\partial L}{\partial \mathbf{U}^r} =$$
$$\frac{\partial L}{\partial \mathbf{W}^r} =$$
$$\frac{\partial L}{\partial \mathbf{U}^c} =$$
$$\frac{\partial L}{\partial \mathbf{W}^c} =$$
$$\frac{\partial L}{\partial \mathbf{V}} =$$

2. Eq. 14 및 유도과정을 참조하여, 학습데이터셋 $\mathcal{D} = (sent_i, label_i)_{i=1}^N$ 으로 총 N개의 문장과 정답 레이블열로 구성되었다고 할때, stochastic gradient descent방법을 이용한 LSTM의 파라미터 학습을 위한 BPTT알고리즘의 pseudo code를 기술하시오.

1.3 Classical RNN 및 LSTM의 구현 (600 points)

본 절에서는 앞에서 정리한 Classical RNN과 LSTM을 `numpy`코드 (gpu확장을 위해 `cupy`버전도 추가 포함)로 구현해보고 영문 품사 태깅 (POS tagging)에 적용해보는 것을 목표로 한다.

1.3.1 Classical RNN 및 LSTM의 Supervised Learning 구현 및 POS tagging에 적용

앞서 HMM 과제와 마찬가지로, Classical RNN과 LSTM학습을 위해 영문 POS Tagging 데이터셋을 사용하고, 데이터셋은 다음과 같이 Training/dev/test셋을 위한 세개의 파일과 Training의 raw corpus, 또한 추가로 word embedding 및 LSTM을 사전학습하기 위한 보다 규모가 큰 raw corpus 로 구성되어 있다.

- Training dataset (`tagged_train.txt`): 단어별 품사태그가 부착된 POS Tagging 태스크를 위한 학습데이터셋
- Development dataset (`tagged_dev.txt`): 단어별 품사태그가 부착된 POS Tagging 태스크를 위한 개발용 데이터셋
- Test dataset (`tagged_test.txt`): 단어별 품사태그가 부착된 POS Tagging 태스크를 위한 평가용 데이터셋

- Raw Training dataset (`raw_train.txt`): 코퍼스상 tokenizer가 적용된 학습 데이터셋상에서 원문 문장셋
- Extra Raw dataset (`raw_extra.txt`): 코퍼스상 tokenizer가 적용된 보다 ㄱ모가 큰 Raw corpus (Word embedding 및 LSTM을 사전학습하는데 활용)

다음은 `raw_train.txt` 파일의 일부를 보여준다.

```
1130b36a9ba93d8fcbf07a2ac0048d4c281cd57c::29 Geordie Nicholson .
1130b36a9ba93d8fcbf07a2ac0048d4c281cd57c::30 Web site: burgessyachts.com .
1130b36a9ba93d8fcbf07a2ac0048d4c281cd57c::31 'Big Aron' Why we like it This
    capacious 2004 yacht was refitted in 2006.
```

다음은 `tagged_train.txt` 파일의 일부를 보여준다.

```
42c027e4ff9730fbb3de84c1af0d2c506e41c3e4::0 LONDON/NNP ,/, England/NNP (/(
    Reuters/NNP )/) --/: Harry/NNP Potter/NNP star/NN Daniel/NNP
    Radcliffe/NNP gains/NNS access/NN to/TO a/DT reported/VBN
     - 128) °20/CD million/CD (/( $/$ 41.1/CD million/CD )/) fortune/NN
    as/IN he/PRP turns/VBZ 18/CD on/IN Monday/NNP ,/, but/CC he/PRP
    insists/VBZ the/DT money/NN wo/MD n't/RB cast/VB a/DT spell/NN on/IN
    him/PRP ./.
42c027e4ff9730fbb3de84c1af0d2c506e41c3e4::1 Daniel/NNP Radcliffe/NNP as/IN
    Harry/NNP Potter/NNP in/IN ``/`` Harry/NNP Potter/NNP and/CC the/DT
    Order/NN of/IN the/DT Phoenix/NNP ''/'' To/TO the/DT disappointment/NN
    of/IN gossip/NN columnists/NNS around/IN the/DT world/NN ,/, the/DT
    young/JJ actor/NN says/VBZ he/PRP has/VBZ no/DT plans/NNS to/TO
    fritter/VB his/PRP$ cash/NN away/RB on/IN fast/JJ cars/NNS ,/, drink/NN
    and/CC celebrity/NN parties/NNS ./.
42c027e4ff9730fbb3de84c1af0d2c506e41c3e4::2 ``/`` I/PRP do/VBP n't/RB
    plan/VB to/TO be/VB one/CD of/IN those/DT people/NNS who/WP ,/, as/RB
    soon/RB as/IN they/PRP turn/VBP 18/CD ,/, suddenly/RB buy/VBP
    themselves/PRP a/DT massive/JJ sports/NNS car/NN collection/NN or/CC
    something/NN similar/JJ ,/, ''/'' he/PRP told/VBD an/DT Australian/JJ
    interviewer/NN earlier/RBR this/DT month/NN ./.
```

위의 파일에서 각 라인의 첫번째 token은 문장 고유 id를 의미한다.

참고로, 본 데이터셋은 공개 영문 Raw corpus상에서 Stanford의 POS tagger를 적용하여, 자동으로 얻은 데이터셋이다. 따라서, Tagging된 데이터셋상에서 오류가 있을 수 있다.

다음 상세 요구사항을 참조하여 `numpy`코드를 작성하시오.

1. **Classical RNN과 LSTM의 Supervised learning을 위한 BPTT 구현**: 위의 Training셋에 있는 모든 문장을 읽어들여, $\mathcal{D} = (sent_i, label_i)_{i=1}^{N}$ 를 구성하여, Stochastic gradient descent에 기반하여 Classical RNN과 LSTM각각 BPTT를 수행하는 학습 코드를 작성하시오. Classical RNN과 LSTM 학습 각각 별도의 코드로 구성하고, 구분이 되도록 별개의 파일(또는 디렉토리)로 저장한다 (예: RNN의 경우 `RNN_train.py`, LSTM의 경우 `LSTM_train.py`).

   학습된 모델 파라미터는 별도의 파일로 저장이 되어, 추론 단계에서 저장된 모델파일을 읽어들여 테스트 및 평가할 수 있도록 한다.

2. **Classical RNN과 LSTM기반 순차 태깅 Inference 구현**: 앞서 얻은 Classical RNN과 LSTM 모델 파일을 읽어들여, 주어진 입력 문장에 대해서 forward를 수행하여 가장 높은 태그열을 출력하는 numpy코드를 작성하시오.

태스트를 위해서, 인자로 model파일을 입력으로 받도록 하고, 콘솔환경에서 문장은 공백라인으로 끝나도록 하고, 엔터로 입력이 끝나면 태그열을 생성하도록 하라. 배치 처리를 위해 python argparse등에 기반하여 모델파일외에 입력파일과 출력파일을 추가 인자로 제시하도록 하여, 입력파일내에 있는 모든 문장에 대한 viterbi tagging결과를 출력파일에 각각 출력하도록 하라.

3. **학습된 Classical RNN과 LSTM 평가**: 위의 학습된 Classical RNN과 LSTM모델을 바탕으로 제공된 Test셋상에서 적용하여, POS tagging 평가를 수행하시오. 평가 결과로 1) 총 단어수, 2) HMM에 기반하여 POS Tag를 정확히 맞춘 총 단어수, 3) Accuracy 의 수치가 포함되도록 출력하시오. 이때, Accuracy는 다음과 같이 정의된다.

$$Accuracy = \frac{\textit{예측된 POS Tag가 정답과 일치하는 총 단어수}}{\textit{테스트셋상 총 단어수}} \tag{14}$$

4. **Word embedding 사전학습 구현 및 테스트**: Word2vec의 Skipgram방법을 참조하여 numpy로 추가로 구현하여, 별도로 제공되는 대규모 코퍼스상에서 word embedding matrix **E**을 사전학습하는 numpy코드를 작성하시오. 학습 결과 얻어진 word embedding parameter등은 모두 별도의 model파일로 저장될 수 있도록 되어야 한다.

   학습된 word embedding을 바탕으로 `Paris - France + Italy = Rome`등과 같은 vector difference가 성립하는지 확인해보고 테스트 결과를 간단히 정리하시오.

   - Word2vec 논문:

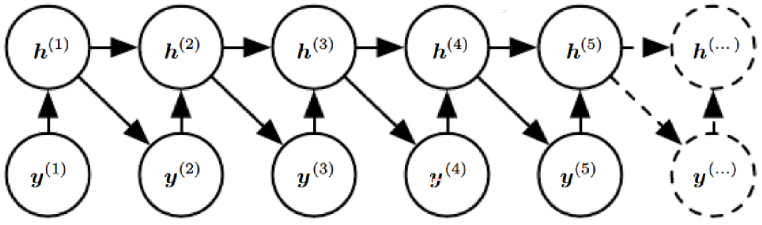
   - `https://arxiv.org/pdf/1301.3781`
   - `https://arxiv.org/pdf/1310.4546`

5. **사전학습 Word embedding을 입력받도록 LSTM학습기 확장 및 평가**: 앞서 구현된 LSTM의 학습코드를 확장하여, argparse등으로 옵션을 추가하여 word embedding model file을 입력 받도록 하여, 사전학습된 word embedding으로 초기화하여 finetuning하도록 하시오.

   또한, 별도의 평가를 수행하여 사전학습 Word embedding을 추가한 이후에 품사 태깅상 성능 개선이 어느정도 이루어지는지도 확인해보시오.

6. **RNN LM에 기반한 LSTM 전체 사전학습 코드 작성**: word embedding만이 아니라, 이번에는 동일한 추가 코퍼스 상에서 LSTM을 **RNN language model 목적 함수**를 통해 전체를 사전학습하는 numpy코드를 작성하시오. 코드를 전체 새롭게 구현할 필요는 없이, 현재 시점에서 정답이 그 다음단어가 되도록, 순차 태깅용 학습기를 조금 수정하거나 확장적용하면 된다. 학습된 결과는 마찬가지로 별도의 model 파일로 저장이 되어야 한다.

   다음 그림은 RNNLM의 계산 그래프를 보여준다.



RNNLM의 목적함수는 $sent = (\langle s \rangle, w_1, \cdots, w_T right)$가 주어졌다고 했을 때, $\mathcal{Y} = \mathcal{V}$이고, $label = (w_1, w_2, \cdots, w_T, \langle s \rangle)$이 정답 (ground-truth)으로 자동으로 만들어진 후, 이들 $(sent, label)$쌍이 self-supervised learning형태로 학습 데이터로 제시된다고 보면 된다.

7. **RNN LM에 기반한 LSTM 사전학습된 모델을 입력받도록 LSTM학습기 확장 및 평가**: 앞서 구현된 LSTM의 학습코드를 확장하여, argparse등으로 옵션을 추가하여 pretrained LSTM model file을 입력 받도록 하여, 사전학습된 LSTM model로 전체 parameters를 초기화하여 finetuning하도록 하시오.

   또한, 별도의 평가를 수행하여 LSTM 사전학습 과정을 추가한 이후에 품사 태깅상 성능 개선이 어느정도 이루어지는지도 확인해보시오.

8. **GPU를 이용하기 위해 Cupy버전 추가**: 사전학습까지 수행하기 위해서는, GPU를 이용하여 학습속도를 개선시켜야 한다. 이를 위해 적어도 LSTM 사전학습하는 코드는 cupy버전으로 확장하여, GPU적용까지 수행하도록 하시오. 실제 수행후에 CPU기반 버전과 GPU기반 바전의 학습 속도 차이를 보고서에 정리하시오.

2 제출 내용 및 평가 방식

Section 1.1과 Section 1.2에 대해서는 **문제 답안 상세 결과물**을 제출해야 한다.

Section 1.3의 구현 문제의 경우 **python**코드외에 결과보고서도 함께 제출해야 하며, 정리하면 다음과 같다.

- **코드 전체**
- **테스트 결과**: 각 내용별 테스트 코드 및 해당 로그 또는 출력 결과.
- **결과보고서**: 구현 방법을 요약한 보고서.

Section 1.3 구현 문항의 평가항목 및 배점은 다음과 같다.

- **각 세부내용의 구현 정확성 및 완결성 (80%)**
- **코드의 Readability 및 체계성 (10%)**
- **결과 보고서의 구체성 및 완결성 (10%)**