자연어처리 HW5

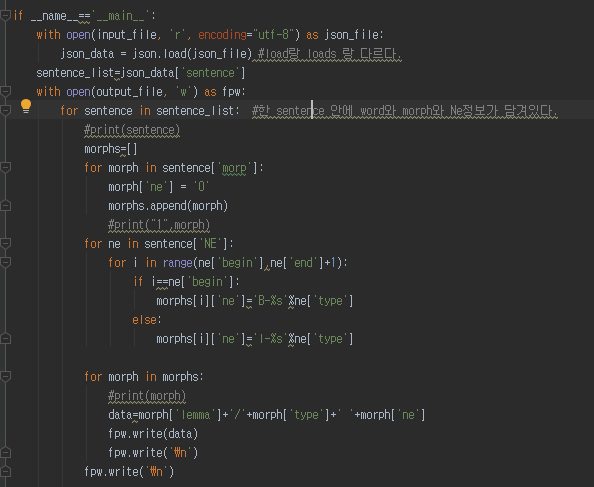
**Bi-directional LSTM for Named Entity Recognition**

120190211 한장훈

**1. 주요 코드 설명**

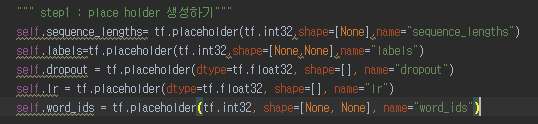
**숙제 1 Json2CoNLL.py**



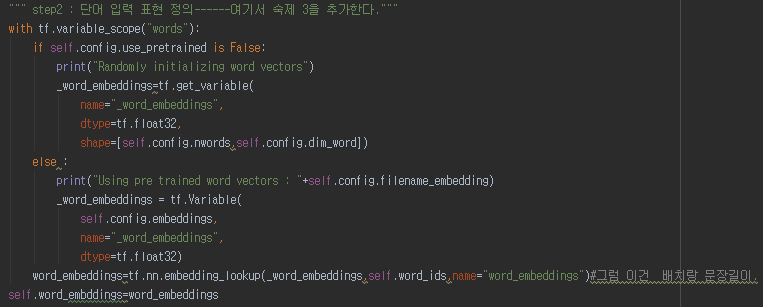
먼저 변환하려는 파일의 경로와 생성파일 경로를 지정했습니다. 저는 이 코드를 3번 돌려서 각각 train,dev,test를 만들었습니다. 

코드는 위와 같이 구현하였습니다. 일단 처음에 morp에 접근하여 ne를 0으로 초기화 한 후 morphs 라는 리스트를 만들었습니다. 이후 NE 정보를 탐색하면서 만들어놓은 morphs 리스트의 인덱스에 따라 morph[‘ne’] 정보를 업데이트 하였습니다. 그리고 형식에 맞게 파일에 써주었습니다.

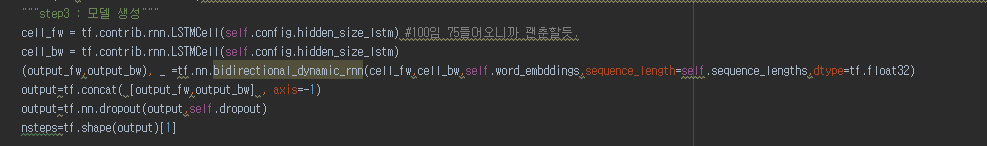
**숙제 2 model.py class NERmodel defbuild()**



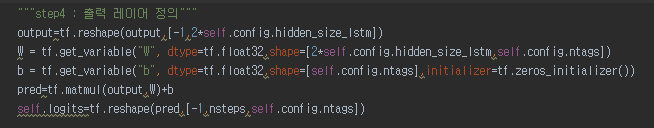
Place holder를 만드는 부분입니다. 여기서 sequence\_lengths는 batch size 이고Label은 batch size \*max length of sentence in batch 이므로 shape를 위와 같이 설정합니다. Dropout,lr은 단일 값이고 wordids는 batch size \* max length of sentence in batch 이므로 위 와 같이 설정합니다.



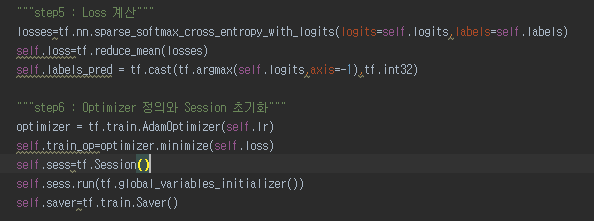
다음은 단어입력 표현 정의입니다. 먼저 words 라는 variable scope를 설정해 줍니다. config에는 미리 트레이닝 된 word embedding을 사용할지 말지 결정하는 flag가 있습니다. 이 flag에 따라 기존 학습한 임베딩을 가져다 쓸 수도 있고 아니면 랜덤하게 놓고 같이 학습할 수도 있습니다.



다음은 모델 생성입니다. Bi-directional LSTM이기 때문에 forward 방향 cell과 backward 방향 cell 하나씩, 총 2개를 생성합니다. Sequence length는 배치 사이즈를 넣어 줍니다. input값으론 word embedding을 넣어줍니다. Shape는 (?,?,50) 의 모양입니다. 리턴되는 output\_fw와 output\_bw는 각각 forward, backward output tensor입니다. shape은 각각 [batch\_size, max\_time, cell\_fw.output\_size] [batch\_size, max\_time, cell\_bw.output\_size] 입니다. 여기서 뒤에 (output\_state\_fw, output\_state\_bw) 는 필요 없기 때문에 받지 않습니다.

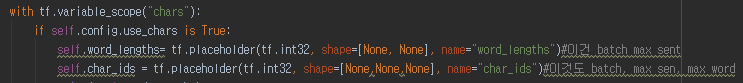


일단 output은 lstm이 앞뒤 2개이므로 2\*hidden\_size 입니다. 또한 label이 ntags만큼 있으므로 위와 같이 설정합니다. B는 tag 개수만큼 설정해 줍니다.

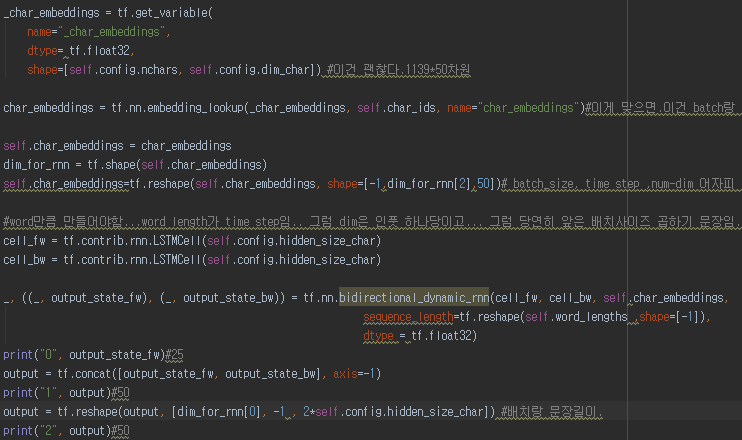


Loss는 cross entropy를 통하여 구합니다. Adam opitmizer를 사용하여 loss를 최소화 하고 sess.run을 통해 세션을 실행시킵니다. Tf.train.Saver()는 모델을 저장하는 코드입니다.

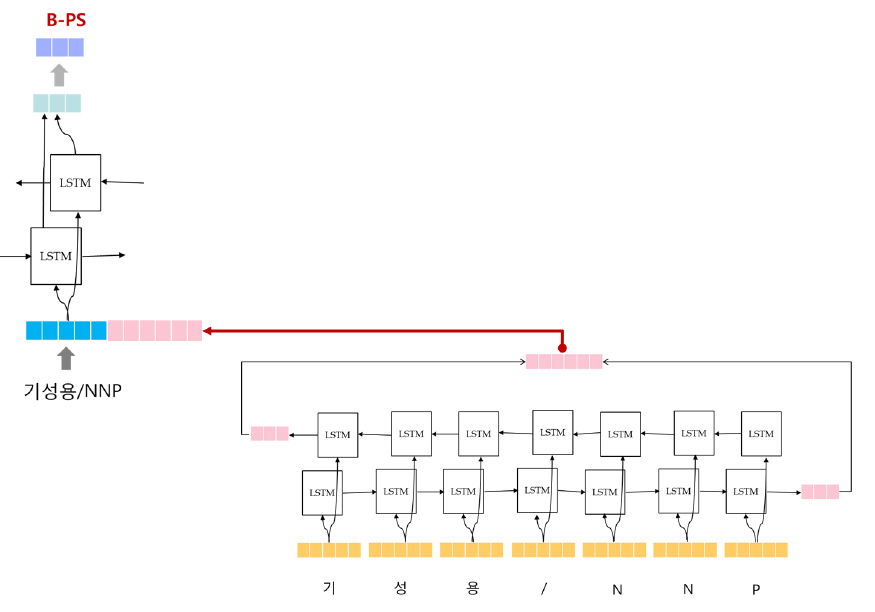
숙제 3 음절 자질 추가



숙제 2와 마찬가지로 placeholder 를 생성합니다. Word length는 batch size \*max length of sentence in batch 이고 char\_ids 는 batch size \* max length of sentence in batch \* max length of word이므로 위와 같이 초기화 하였습니다.



앞에서 했던 것처럼 랜덤하게 임베딩을 생성합니다. 이번에 음절자질을 추가할것이기 때문에 config의 음절에 관한 정보로 생성합니다. 여기서 bidirectional dynamic rnn에 넣어주기 위해선 3차원이어야 하는데. 현재 4차원입니다. 그래서 reshape를 위 그림과 같이 해줍니다. 한편 밑에 그림과 같이 time step 마다 한 음절씩 들어가야 하므로 time step에는 word length 만큼 들어가야합니다. 그래서 위와 같이 reshape를 해줍니다.



앞에서 와 같이 tf.nn.bidirectional\_dynamic\_rnn 만들어 주는데 sequence\_length로 word\_length가 들어가야합니다. 왜냐하면 timestep마다 단어의 음절을 보는데 그게 배치사이즈 \*문장 길이 만큼 있기 때문입니다. 또한 output에서 위로 가는게 아닌 마지막 state를 받아와야 하므로 숙제 2에서 와는 다르게 앞을 날리고 뒤에 2개를 받습니다. 이때 lstm은 2가지 state가 있어서 필요한 state만 받기위해 (\_, output\_state\_fw) 과 같이 받습니다.



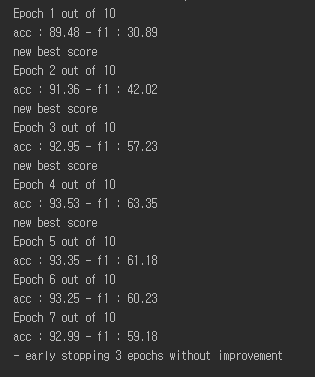
그림대로 두개의 state정보를 concat해주고 기존의 wordembedding에 합쳐주기위해 reshape해줍니다.



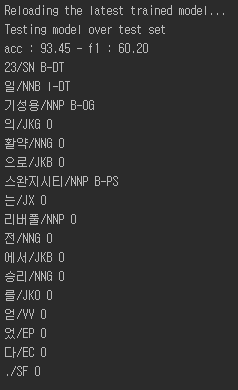
만약 음절 자질 정보를 사용한다고 하면 기존 word embedding과 concat해서 학습을 합니다.

**2. 실험결과.**

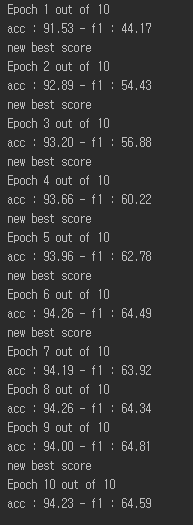
**1. 숙제2 random initialization train**

****

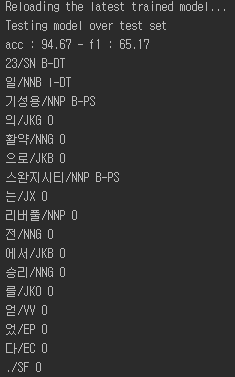
**2. 숙제2 random initialization evaluate**

****

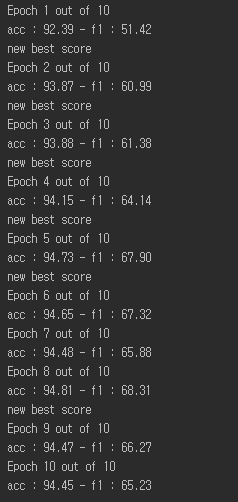
**3. 숙제2 pretrained train**

****

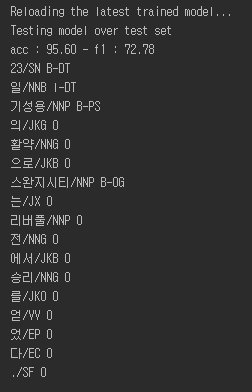
**4. 숙제2 pretrained evaluate**

****

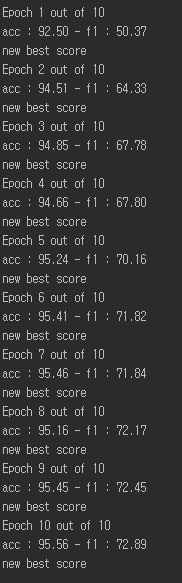
**5. 숙제3 random initialization train**

****

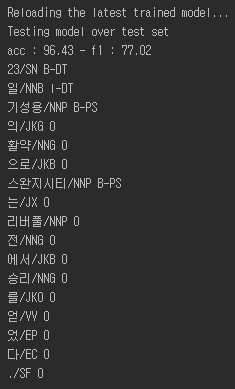
**6. 숙제3 random initialization evaluate**

****

**7. 숙제3 pretrained train**

****

**8. 숙제3 pretrained evaluate**

****

**3. 결과분석**

random initialization보다 pretrained 된 word embedding을 사용했을 때 성능이 더 높으면 음절 자질을 추가하여 학습했을 때 성능이 train과 evaluate 둘다 높았습니다.

**4. 기타.**

사용 언어 PYTHON 3.6

개발환경 JetBrains PyCharm 2018.3.5 x64

Code에 python code가 있습니다.

data에는 임베딩과 1/3을 제외한 숙제1에서 만든 데이터 파일이 있습니다.

학습 시 1/3이 아닌 전체 train.txt로 학습하였습니다.