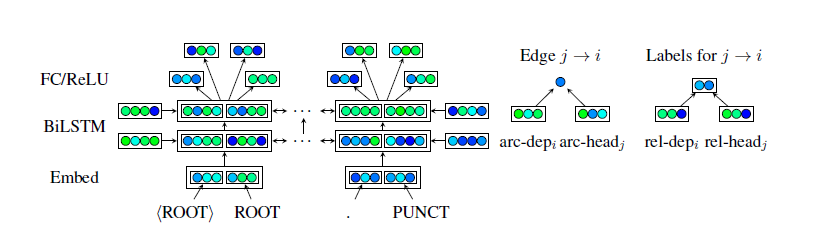
1. Introduction

이 시스템은 딥 바이아핀(2개) 뉴럴 디펜던시 파서를 기반으로 한다. 이 디펜던시 파서에서는 각 단어들의 벡터표현을 생성하기 위해서 LSTM네트워크(우리 모델은 GRU)를 쓰고, 그 후에는 생성된 벡터들을 바이아핀 분류기에서 사용한다. 이를 통해 각각의 의존 노드들의 지배소 토큰을 예측해내고 결과의 클라스를 예측해낸다. 이것을 매우 다양한 트리뱅크(주석 말뭉치)에 적용하기 위해서, 이 시스템에 크게 2가지를 확장할 것이다.

(1) LSTM을 사용하여 나온 자소 기반 시퀀스로부터 나온 단어표현을 합친다. 이것은 우리 모델에서 빈도수가 적은 희귀한 단어들에 대한 모델의 대처능력을 상승시켜줄것이다.

(2) 파싱에 이용된 구조와 거의 비슷한 구조를 이용하여 트리뱅크에 이용할 태거를 훈련시킨다. 이는 품사 태그의 퀄리티의 잠재적인 향상을 이용하기 위한것이다. 이런 접근법이, attachment accuracy metrics와 품사 태깅에 의하여, 공유된 task 데이터셋의 매크로 평균에 대해서 매우 높은 성능의 결과를 얻어낸다.

주목해야할 점중 하나는 바로 관계의 단순함이란 것이다. 우리의 접근법은 언어마다 하나의 태거와 파서를 사용하며, 이 2개는 오직 단어와 태그로만 훈련이 된 태거/파서이다. 그러므로, 우리는 앙상블, 레마(사전에서 나오는 기본형태의 단어), 또는 형태학적 특징이 필요가 없다(이것들은 정확도를 잠재적으로 높일수 있지만, 여기서는 사용하지 않는다.).



2. Architecture

2.1 Deep biaffine parser

이 접근법의 기본적인 구조는 ~~~~~(다른 논문들 내용)~~~~~그래프 기반의 뉴럴 파서와 비슷합니다. Dozat and Manning's 2017 파서에서는, 모델의 input이 토큰들의 시퀀스 및 그들의 품사 태그였습니다. 이 input은 이후에 다층 bidirectional LSTM 네트워크에 투입됩니다. (cell state를 제외한)마지막 LSTM층의 output state는 4개의 분리된 ReLu층으로 투입됩니다. 이는, 4개의 특수화된 벡터를 생성합니다.

(1) 자신이 dependent한 단어라 할때, head(label)를 찾는 벡터

(2) 자신이 head인 단어라 할때, 자신의 dependent단어들(label)을 찾는 벡터

(3) 자신이 dependent한 단어라 할때, 자기 자신의 label을 결정하는 벡터

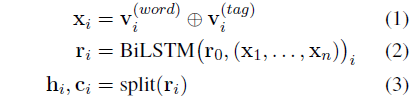
(4) 자신이 head인 단어라 할때, 자신의 dependent단어들의 label을 결정하는 벡터

이 벡터들은 후에 2개의 biaffine 분류기에서 사용됩니다.

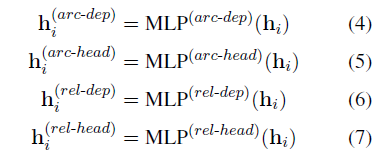
(1) 각각의 토큰 쌍에 대한 점수를 계산, 제일 높은 점수는 곧 토큰의 가장 그럴듯한 head를 지칭합니다.

(2) 주어진 토큰/헤드 쌍에 대한 각각의 label에 대한 점수를 계산, 가장 높은 점수는 (head에서 dependent로 향하는) arc에 대한 가장 그럴듯한 레이블을 의미합니다. 그림1 참조

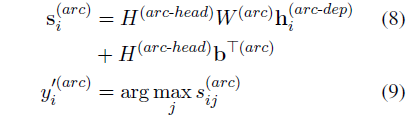
공식적으로 보면, n개의 단어 임베딩의 시퀀스 word(v1, v2, ..., vn)과 n개의 태그 임베딩 tag(v1, v2, ..., vn) 주어졌을때(Section 2.2에 자세히 묘사됨), 우리는 각 쌍을 결합하고 그 결합한 쌍을 BiLSTM에 초기 state r0와 함께 넣어줍니다.



각각의 hidden state hi로부터 ReLu 퍼셉트론 층을 이용하여 4개의 분리된 벡터를 생성해냅니다.

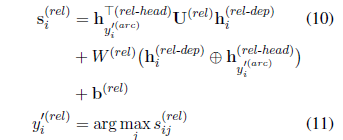


토큰 i에 대한 예측값 yi’(arc)를 만들어 내기 위해서, (arc) 히든벡터를 포함하여 biaffine 분류기를 사용합니다.



전형적인 Affine 분류기(Wh + b의 형태)와 (8)식 사이의 유사성, 그리고 각각의 W와 b가 H(arc-head)에 의해서 변형된다는 것을 명심하자. 또한 biaffine layer의 두개의 줄에서의 식은 직관적인 해석을 가집니다 : 첫번째줄은 단어 j가 단어 i의 head가 될 확률과 관련되어 있습니다. 이는 2개의 h(arc) 벡터(head, dep)에서 정보가 주어질 경우 성립합니다. 예를 들어서, 단어 i가 ‘the’이고 단어 j가 ‘cat’이라 할 때 단어 i가 단어 j에 의존할 확률이 예시가 됩니다. (the가 cat에 의존) 두번째줄은 단어 j가 단어 i의 head가 될 확률과 관련되어 있습니다. 이는 head의 벡터에서만 정보가 주어질 경우 성립합니다.(dep 벡터가 식에 들어있지 않음) 예를 들어서, 단어 j가 ‘the’라고 할때 단어 i가 단어 j에 의존할 확률이 단어 i가 뭐든지 간에 매우 작을 확률임이 분명합니다.

단어 i에 대한 head yi’을 결정한 이후, 우리는 또다른 biaffine 변환을 사용합니다. 이번에는 (rel) 벡터를 포함하며, 이는 예측 label을 생성하는 용도입니다.



역시나 마찬가지로, (10)에서의 식도 직관적으로 해석할 수 있습니다. 첫번째 줄은 2개의 h(rel)벡터에 대한 정보가 주어졌을 때, 레이블을 조사하는 확률과 관련되어 있습니다. 예를 들어서, 주어진 단어 i가 ‘the’이고 head가 ‘cat’이라 할 때, label ‘det’의 확률을 들 수 있다. 두번째 줄은 h(rel)벡터에 대한 정보 2개 중 하나가 주어졌을 때, 레이블을 조사하는 확률과 관련되어 있습니다. 예를 들어서, 주어진 단어 i가 ‘the’ 또는 주어진 단어 j가 ‘cat’일 때, label ‘det’의 확률을 들 수 있다.

우리는 이 2개의 biaffine 분류기(arc, rel)를 결합하여 훈련시킵니다. 바로 그들의 소프트맥스 크로스 엔트로피 loss의 합을 최적화 시킴으로써 훈련시킬 겁니다. Test를 할때에는, 우리는 주어진 root에 대한 tree들을 반복적으로 순환 cycle을 찾아내어 제거함으로써 tree가 잘 생성되게 한 이후에, 제일 높은 점수를 가지는 tree 하나를 골라냅니다. 이렇게 하는 것이 간단하면서도 효율적이라 할 수 있습니다.

3. Training details

우리의 모델은 Dozat and Manning(2017)과 같은 하이퍼파라미터 세팅을 사용할 것입니다. 몇가지를 제외하고 말이죠. Parser는 3개의 BiLSTM층을 사용하며(우리 연구과제에서는 단층 BiLSTM을 사용할 예정) 100차원의 단어 임베딩, 100차원의 태그 임베딩, 그리고 200차원의 recurrent states(양 방향 전부)를 사용할 것입니다. Arc 분류기는 400차원의 head/dependent 벡터 states를 사용하며, label 분류기는 100차원의 head/dependent 벡터 states를 사용할 것입니다. 우리는 단어 임베딩과 태그 임베딩을 독립적으로 33% 확률로 드랍할 것입니다. 우리는 같은 확률의 드롭아웃을 LSTM(우리 모델에서는 GRU), ReLU층, 분류기들에서 input과 recurrecnt connection들을 33%의 확률도 드롭아웃 시킵니다. 그리고 Adam으로 최적화 시킵니다. 학습률은 2e-3으로 세팅하며 beta1과 beta2는 0.9로 설정합니다.(??) 우리는 모델을 30000번 정도의 training step으로 돌리며(1 step당 1개의 미니배치가 들어가며, 이 미니배치는 대략 5000개의 토큰이 있다) 1000번 미만의 training step인 상태라면 100번마다 세이브를 하며, 그 이후에는 validation 정확도가 높아질 때 마다 세이브 시킵니다. 혹은 validation data가 없을 경우에는 training 정확도가 상승할 때 마다 저장한다. 5000번의 training step이 정확도 상승 없이 지나갈 경우, 훈련을 중단합니다.

Character 모델의 경우, 100차원의 uncased character 임베딩을 사용하며, 400차원의 recurrent states를 사용합니다.(우리 모델에서는 Character Based CNN을 사용할 예정) character들은 드롭하지 않지만, LSTM과 attention에서는 33% 드롭아웃 합니다.(우리 모델에서는 상관X)