Neural Dependency Parser

0. Refeence

- A Fast and Accurate Dependency Parser using Neural Networks, Chen et al.

1. Introduction

최근의 parsing 흐름은 feature-based discrimintive dependency parsers을 이용하여 훌륭한 성공을 거두었다. 하지만 이러한 parsers는 1) 사용하는 feature가 매우 spare하고 2) feature templates을 만들기 위해 많은 비용이 들어가며 3) 계산량이 매우 heavy하다. 이러한 문제점을 해결하기 위해 본 논문엣는 neural network 기반 dependency parser을 제안한다. 본 논문의 neural network classifier는 transition-based dependency parser 틀에서 parsing decision을 만든다. 따라서 기존의 transition-based dependency부터 살펴본다.

2. Transition-based Dependncy Parsing

greedy parsing 중 하나로, arc-standard system을 이용한 parser(Nivre, 2004)을 살펴본다. arc-standard system에는 configuration $c = (s, b, A)$가 있고 s는 stack, b는 buffer, A는 dependency arcs 이다. 초기 configuration은 $s = [Root],\ b = [w_1, \cdots, w_n],\ A = \emptyset$이다. arc-standard system parser는 모든 단어와 그들의 POS tags, head of a word 와 label, stack/buffer에서 단어의 위치를 이용한다. 전통적인 접근법은 stack나 buffer에서 한 개~세 개의 요소를 결합한 indictor feature을 사용한다. 바로 이 features가 아래의 문제를 가진다.

- Sparsity
  features는 매우 sparse하고 특히 dependency parsing에서 문제가 된다. dependency parsing에서는 단어-단어의 interactions에 의존하기 때문이다. 본 논문에서는 이러한 features들의 중요성을 밝혀냈다. 즉, 사용해야할 중요한 features지만 너무 sparse해서 문제가 된다는 것이다.

- Incompleteness
  거의 모든 feature templates에 존재하는 피할 수 없는 문제이다. 아무리 수동으로 잘 만진다 해도, 모든 유용한 단어의 combination을 얻을 수 없다.

- Expensive feature computation
  indicator features을 만드는 것은 매우 expensive하다. 실제 다른 논문에서도, feature computation에서 시간이 매우 많이 소모되었다고 보고되었다.

3. Neural Network Based Parser

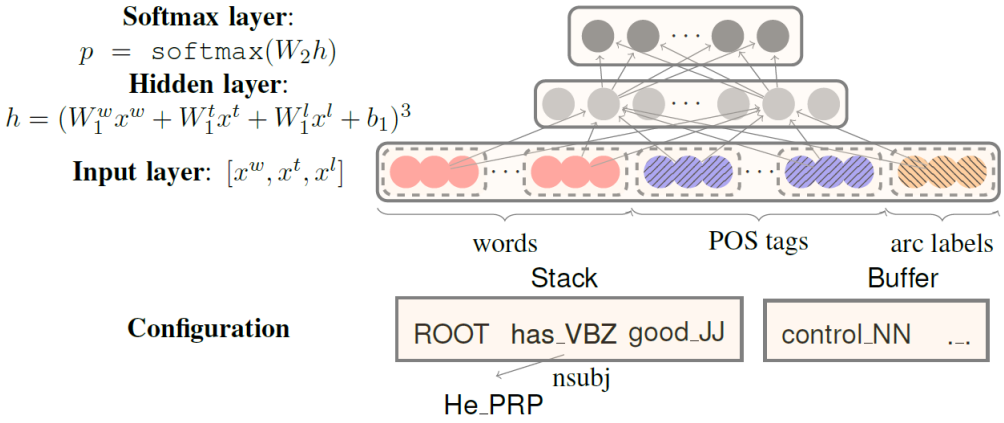

그림 1

그림 1은 본 논문에서 제시하는 model architecture이다. 자세히 보면 단어, pos tags, arc labels가 하나의 벡터로 concatenate 되어 있다. 모델을 자세히 살펴보자.

- 각 단어, pos tags, arc labels를 d차원 임베딩 벡터로 표현한다. $w_i^w, e_i^t, e_j^l \in \mathbb{R}^d$는 i번째 단어, pos tag, j번째 arc label을 의미한다.
- 예측에 도움이 될만한 set of elements를 고른다. 이를 S^w, S^t, S^l이라고 한다. n_w, n_t, n_l을 각 set 원소의 갯수라고 하자. 즉, 예를 들어 $S^w = \{w_1, \cdots, w_{n_w}\}$이다.
- 이제 하나의 hidden layer가 있는 neural network을 만드는데, S^w, S^t, S^l로부터 원소를 뽑아 이에 대응하는 임베딩 벡터를 input layer로 넣는다.
- 즉, 위 그림을 보면 input layer가 $[x^w, x^t, x^l]$인데, $x^w = [e_{w_1}^w; e_{w_2}^w; \cdots; e_{w_{n_w}}^w]$이다. x^t, x^l도 비슷한 방식으로 만든다.
- input layer가 hidden layer, cube activation function을 통과한다.

$$h = \left(W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1\right)^3$$

- classification을 위해 softmax layer을 마지막에 통과한다.