Lecture 4: Dependency parsing

Presenter: Chu Đình Đức

Lecture plan

1. Dependency Parsing
2. Greedy Deterministic Transition-Based Parsing
3. Neural Dependency Parsing

1. Dependency Parsing

- Dependency parsing is the task of analyzing the syntactic dependency structure of a given input sentence S
- Formally, the dependency parsing problem asks to create a mapping from the input sentence with words $S = w_0 w_1 \ldots w_n$ (where w_0 is the ROOT) to its dependency tree graph G
- Model:
    1. Learning: Given a training set D of sentences annotated with dependency graphs, induce a parsing model M that can be used to parse new sentences
    2. Parsing: Given a parsing model M and a sentence S, derive the optimal dependency graph D for S according to M

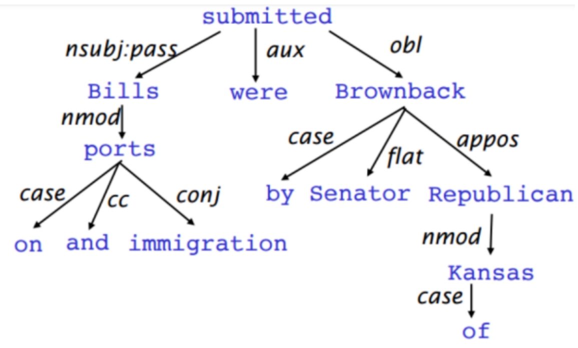


Figure 1: Dependency tree for the sentence "Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas"

2. Greedy Deterministic Transition-Based Parsing

This transition system is a state machine, which consists of *states* and *transitions* between those states

State:

For any sentence $S = w_0 w_1 \ldots w_n$, a state can be described with a triple $c = (\sigma, \beta, A)$

- a stack σ of words w_i from S
- a buffer β of words w_i from S
- a set of dependency arcs A of the form (w_i, r, w_j), where w_i, w_j are from S, and r describes a dependency relation

It follows that for any sentence S = w0w1…wn

- initial state: $([w_0]_\sigma, [w_1, \ldots, w_n]_\beta, \varnothing)$
- terminal state: $(\sigma, []_\beta, A)$

2. Greedy Deterministic Transition-Based Parsing

The model induces a sequence of transitions from *initial state* to *terminal state*

Transitions: 3 types of transitions between states

- SHIFT: remove the first word in the buffer and push it on top of the stack (pre-condition: buffer has to be non-empty)
- LEFT-ARC$_r$: add a dependency arc (w_j, r, w_i) to the arc set A, where w_j is the word at the top of the stack, w_i is the word second to the top of the stack. Remove w_i from the stack (pre-condition: the stack needs to contain at least two items and w_i can't be the ROOT)
- RIGHT-ARC$_r$: add a dependency arc (w_i, r, w_j) to the arc set A, where w_j is the word at the top of the stack, w_i is the word second to the top of the stack. Remove w_j from the stack (pre-condition: the stack needs to contain at least two items)

1. Shift $\sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A$
2. Left-Arc$_r$ $\sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_j, \beta, A \cup \{r(w_j,w_i)\}$
3. Right-Arc$_r$ $\sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_i, \beta, A \cup \{r(w_i,w_j)\}$

Figure 2: Transitions for Dependency Parsing.

3. Neural Dependency Parsing

- The primary distinction from previous models is the reliance on dense rather than sparse feature representations
- The model employs the arc-standard system for transitions, as presented in section 2
- Ultimately, the aim of the model is to predict a transition sequence from initial configuration c to a terminal configuration
- As the model is greedy, it attempts to correctly predict one transition $T \in \{SHIFT, LEFT\text{-}ARC_r, RIGHT\text{-}ARC_r\}$ at a time, based on features extracted from the configuration $c = (\sigma, \beta, A)$

3. Neural Dependency Parsing

The model we will describe employs the arc-standard system for transitions, as presented in section 2. Ultimately, the aim of the model is to predict a transition sequence from initial configuration *c* to a terminal configuration

Feature Selection: the features for a given sentence S generally include some subset of:

- S_{word}: vector representations for some of the words in S (and their dependents) at the top of the stack σ and buffer β
- S_{tag}: Part-of-Speech (*POS*) tags for some of the words in S. *POS* tags comprise a small, discrete set $P = \{NN, NNP, NNS, DT, JJ, \ldots\}$
- S_{label}: the arc-labels for some of the words in S. The arc-labels comprise a small, discrete set, describing the dependency relation: $L = \{amod, tmod, nsubj, csubj, dobj, \ldots\}$

3. Neural Dependency Parsing

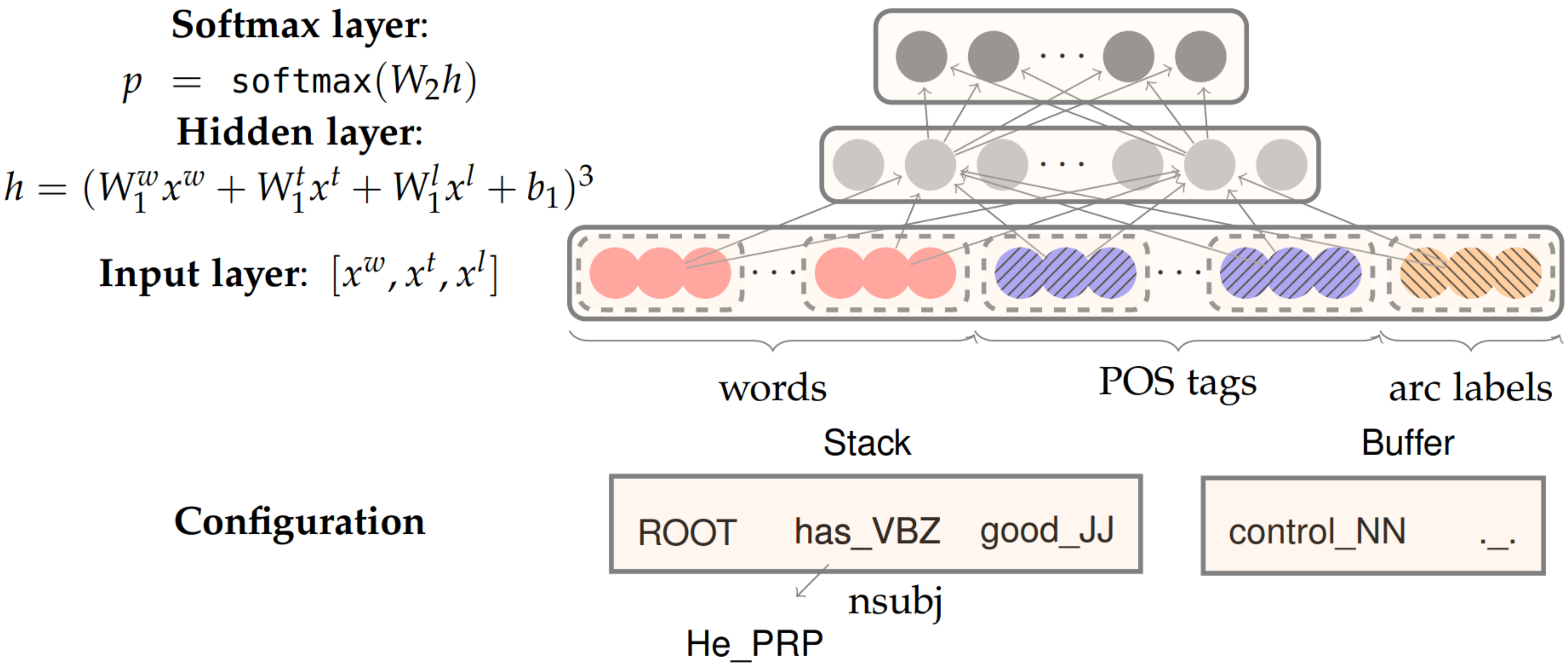
Feature Selection Example:

- S_{word}: the top 3 words on the stack and buffer: $s_1, s_2, s_3, b_1, b_2, b_3$. The first and second leftmost/rightmost children of the top two words on the stack: $lc_1(s_i), rc_1(s_i), lc_2(s_i), rc_2(s_i), i = 1, 2$. The leftmost of leftmost/rightmost of rightmost children of the top two words on the stack: $lc_1(lc_1(s_i)), rc_1(rc_1(s_i)), i = 1, 2$. The total S_{word} contains $n_w = 18$ elements
- S_{tag}: the corresponding *POS* tags for S_{tag} ($n_t = 18$)
- S_{label}: the corresponding arc labels of words ($n_l = 12$)

3. Neural Dependency Parsing

- We use a special NULL token for non-existent elements
- For each feature type, we will have a corresponding embedding matrix (E^w, E^t, E^l), mapping from the feature's one hot encoding to a d-dimensional dense vector representation, then concatenate these vectors into our inputs $[x^w, x^t, x^l]$

3. Neural Dependency Parsing



Note

- Relation set: here
- Example:
  - *nsubj:pass:* passive nominal subject
  - *nmod:* nominal modifier
  - *case:* case marking
  - *cc:* coordination
  - *conj:* conjunct
  - *aux:* auxiliary
  - *obl:* oblique nominal
  - *flat:* flat multiword expression
  - *appos:* appositional modifier

THANK YOU

Any question?