A Appendix

A.1 Algorithm

Here we list two simple algorithms for making balanced binary trees on the target sentence. For our experiments of TrDec on binary trees, we use both algorithms to produce two versions of binary tree for each training sentence, and concatenate them as a form of data augmentation strategy.

Algorithm 1: The first method of making balanced binary tree

Input : w: the list of words in a sentence, l: start index, r: end index
Output : a balanced binary tree for words from l to r in w

Function `make_tree_v1(w, l, r)`**:**
    if $l = r$ **then**
        return TerminalNode($w[l]$)
    end

    $m = \text{floor}((l + r)/2)$ ▷ *index of split point*
    left_tree = make_tree_v1(w, l, m)
    right_tree = make_tree_v1($w, m + 1, r$)

    return NonTerminalNode(*left_tree*, *right_tree*)

Algorithm 2: The second method of making balanced binary tree

Input : w: the list of words in a sentence, l: start index, r: end index (inclusive)
Output : a balanced binary tree for words from l to r in w

Function `make_tree_v2(w, l, r)`**:**
    nodes = EmptyList()
    $i = 0$
    while $i <$ *len*$(w) - 1$ **do**
        lc = TerminalNode($w[i]$)
        rc = TerminalNode($w[i + 1]$)
        n = NonTerminalNode(*lc*, *rc*)
        nodes.append(n)
        $i = i + 2$
    end

    if $i \neq$ *len*(w) **then**
        n = TerminalNode($w[i]$)
        nodes.append(n)
    end

    return make_tree_v1(*nodes*, 0, *len*(*nodes*) $- 1$)