Dependency Parsing

Nguyễn Hữu Hoàng

Nội dung trình bày

1. Tổng quan về Dependency Parsing
2. Phương pháp Transition-based
3. Phương pháp Graph-based
4. Các cách tiếp cận hiện nay
5. Một số kết quả cài đặt

1. Tổng quan về Dependency Parsing

1.1. Dependency Parsing là gì?

1.2. Các nhãn phụ thuộc (Dependency Labels)

1.3. Các tính chất của cây cú pháp phụ thuộc.

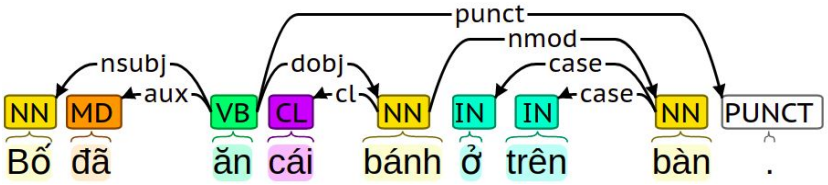
1.4. Các vấn đề cần giải quyết của bài toán phân tích cú pháp phụ thuộc

1.1. Dependency Parsing là gì

- Tiếng Việt: Phân tích cú pháp phụ thuộc
- Thuộc 1 kiểu bài toán phân tích cú pháp
- Không phân tích chủ ngữ, vị ngữ, các cụm danh từ, cụm động từ,… thay vì đó, phân tích quan hệ phụ thuộc giữa các từ trong câu với nhau.
- Thường liên quan chặt chẽ đến bài toán Gán nhãn từ loại (Part Of Speech Tagging)
- Được bắt đầu quan tâm nhiều từ thập kỷ trước do sự giàu thông tin mà kiểu phân tích này mang lại.

1.1. Dependency Parsing là gì

Ví dụ về cây cú pháp phụ thuộc:



1.1. Dependency Parsing là gì

- Một quan hệ phụ thuộc thể hiện bằng 1 mũi tên có hướng, trong đó:
  - Phần có mũi tên là dependent (modifier, subordinate, ...)
  - Phần còn lại là head (governor, regent, ...)
  - Nhãn phụ thuộc tương ứng giữa 2 từ
- Một cấu trúc phụ thuộc gồm có:
  - Các quan hệ phụ thuộc (directed arcs)
  - Nhãn phụ thuộc tương ứng của các quan hệ này
  - Thường kèm với nhãn từ loại tương ứng của 1 từ

Cây cú pháp thường sẽ có thêm 1 nút root nối với nút không có head trong câu, quan hệ đi kèm cũng có nhãn là root.

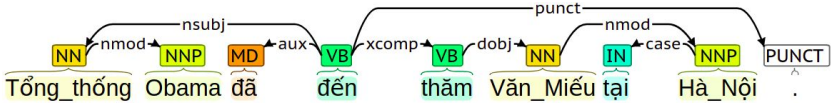
1.1. Dependency Parsing là gì

Các ứng dụng của phân tích cú pháp phụ thuộc:

- Nhận diện thực thể

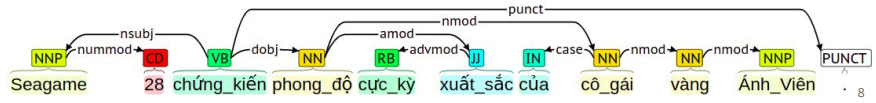
- Trích rút quan hệ.

- Dịch máy



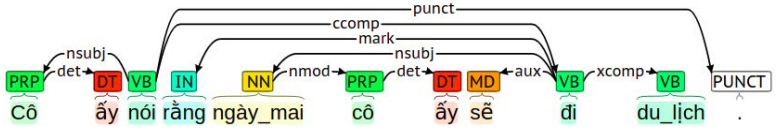
1.2. Các nhãn phụ thuộc

- Một số nhãn phụ thuộc:
  - nsubj (Nominal subject): chủ ngữ, chủ thể
  - nsubjpass: chủ ngữ bị động
  - dobj (Direct object): tân ngữ trực tiếp
  - iobj (indirect object): tân ngữ gián tiếp
  - nmod (Nominal modifier): danh từ bổ nghĩa
  - amod (Adjectival modifier): tính từ bổ nghĩa
  - nummod (Numeric modifier): số từ bổ nghĩa
  - advmod (Adverbial modifier): thành phần bổ nghĩa mang tính chất trạng từ.



1.2. Các nhãn phụ thuộc

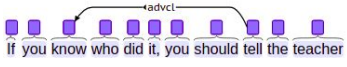
- Một số nhãn phụ thuộc:
  - ccomp (Clausal component): Mệnh đề thành phần
  - xcomp (Open clausal component): Mệnh đề thành phần mở rộng
  - aux (Auxiliary): phụ từ, trợ động từ
  - det (Determiner): định từ
  - mark: là từ đánh dấu ngăn cách giữa 2 mệnh đề
  - punct: dấu câu

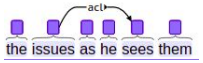


1.2. Các nhãn phụ thuộc

- Một số nhãn phụ thuộc:
  - advcl (Adverbial clause modifier): Mệnh đề trạng ngữ bổ nghĩa
  - acl (Adjectival clause): Mệnh đề phụ thuộc
  - ...

Xem thêm: http://universaldependencies.org/u/dep/



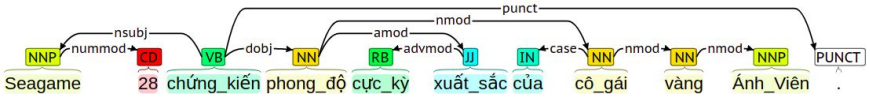


1.3. Các tính chất của cây cú pháp phụ thuộc

- Xét cây cú pháp là 1 đồ thị với các từ là các đỉnh (node), các quan hệ là các cạnh (arc)
- Đồ thị cú pháp phụ thuộc này có 4 tính chất:
  - Weakly Connected (Kết nối yếu)
  - Acyclic (Không có chu kỳ)
  - Single head (1 từ chỉ có duy nhất 1 head)
  - Projective

1.3. Các tính chất của cây cú pháp phụ thuộc

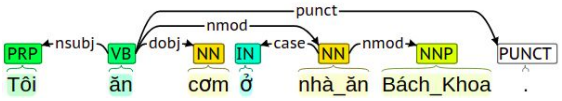
- Weakly Connected:
  - Với mọi node i, luôn tồn tại 1 node j sao cho có 1 cạnh nối i -> j hoặc j -> i
- Acyclic:
  - Nếu tồn tại cạnh i->j, thì không thể tồn tại 1 đường đi j->*i
- Single head:
  - Nếu có cạnh i -> j, thì sẽ không có cạnh k -> j, với k != i



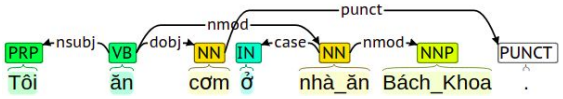
1.3. Các tính chất của cây cú pháp phụ thuộc

- Projective: (tính chất này không bắt buộc)
  - Nếu tồn tại cạnh i->j, thì với mọi k nằm giữa i và j, luôn có đường đi i ->* k
  - Một cách trực quan, không có cạnh chéo nhau khi vẽ cây cú pháp tuần tự theo câu

Projective



Non-Projective

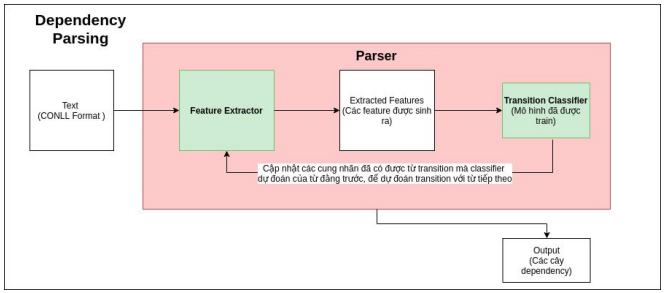


1.4. Các vấn đề cần giải quyết

- Với bài toán phân tích cú pháp phụ thuộc, có nhiều cách tiếp cận khác nhau.
- Tương tự như nhiều bài toán NLP, ta có 2 hướng phổ biến:
  - Rule-based, dựa trên luật mà quyết định giữa 2 từ có quan hệ phụ thuộc gì
  - Data-driven, dựa trên dữ liệu, áp dụng phương pháp học máy để học ra mô hình quyết định quan hệ giữa các từ.
- Trong phần trình bày này, chúng ta chỉ xem xét hướng data-driven với 2 phương pháp chính:
  - Transition-based
  - Graph-based

1.4. Các vấn đề cần giải quyết

Ví dụ hệ thống transition-based



1.4. Các vấn đề cần giải quyết

Có 3 vấn đề chính cần giải quyết trong bài toán phân tích cú pháp phụ thuộc hướng dữ liệu (data-driven):

- Lựa chọn đặc trưng để học. (Feature Extractor)

- Thuật toán học máy. (Learning Algorithm)

- Thuật toán phân tích. (Parsing Algorithm)

1.4. Các vấn đề cần giải quyết

Lựa chọn đặc trưng:

- ➢ Ở đây giai đoạn này, cần lựa chọn ra những đặc trưng tốt nhất để học ra mô hình quyết định các quan hệ phụ thuộc giữa các từ.
- ➢ Các đặc trưng này thường được lựa chọn bởi các chuyên gia trong lĩnh vực này
- ➢ Thường gồm các core feature (từ, nhãn từ loại,... của các từ đang xét và các từ xung quanh) và các feature template (các kết hợp giữa các core feature, ...)
- ➢ Cần lựa chọn cẩn thận, việc đưa các đặc trưng không có ích làm tăng độ phức tạp tính toán và tăng nguy cơ overfit mô hình

1.4. Các vấn đề cần giải quyết

Thuật toán học máy:

- Dùng học máy để huấn luyện ra mô hình cung cấp khả năng quyết định giữa 2 từ có quan hệ phụ thuộc gì và nhãn gì hay không.
- Sử dụng các đặc trưng đã được lựa chọn ở bước trước.
- Khác nhau giữa các phương pháp Transition-based và Graph-based
  - Transition-based: tại 1 thời điểm, quyết định transition tiếp theo là gì.
  - Graph-based: quyết định điểm (score) của từng cạnh nối 2 từ trong câu

1.4. Các vấn đề cần giải quyết

Thuật toán phân tích cú pháp:

- Thuật toán này giúp xây dựng được cây phụ thuộc tốt nhất với các quyết định của mô hình được huấn luyện.
- Thuật toán này cũng đóng vai trò kiểm soát điều khiển các thành phần trong quá trình phân tích, lấy kết quả từ mô hình dự đoán cho các từ phía trước cung cấp cho phần Extractor, lấy đặc trưng ra đẩy vào mô hình tiếp tục dự đoán cho các từ phía sau.

1.4. Các vấn đề cần giải quyết

- Các vấn đề này đều cần giải quyết với cả 2 phương pháp Transition-based và Graph-based.
- Do cách tiếp cận của 2 phương pháp này khác nhau, nên các thuật toán bên trong khá khác nhau. Tuy nhiên, phần lựa chọn đặc trưng chia sẻ khá nhiều đặc trưng chung giống nhau
- Các phần tiếp theo sẽ lần lượt trình bày về 2 phương pháp này

2. Phương pháp Transition-based

2.1. Giới thiệu.

2.2. Thuật toán phân tích cú pháp.

2.3. Thuật toán học.

2.4. Lựa chọn đặc trưng.

2.5. Các kỹ thuật cải tiến hệ thống Transition-based.

2.1.Giới thiệu

- Transition-based là 1 phương pháp phổ biến cho bài toán Dependency Parsing
- Ý tưởng cơ bản của nó là dựa trên các Transition (SHIFT, REDUCE, LEFT-ARC, RIGHT-ARC)
- Khi đọc câu từ trái sang phải, hệ thống sẽ quyết định thực hiện transition nào, dãy các transition này giúp xác định được quan hệ phụ thuộc giữa các từ trong câu
- Các nhà nghiên cứu: Joakim Nivre, Yoav Goldberg, Yuji Matsumoto,...

2.2.Thuật toán phân tích cú pháp.

2.2.1. Giới thiệu

2.2.2. Các thuật ngữ.

2.2.3. Thuật toán.

2.2.4.Ví dụ

2.2.1. Giới thiệu.

- Ở đây trình bày thuật toán Nivre, là 1 dạng giải thuật greedy (giải thuật tham lam)
- Thực hiện duyệt câu từ trái qua phải, lựa chọn transition có khả năng cao nhất và đi tiếp theo transition đấy.
- Cho phép chỉ tìm ra cây thỏa mãn tính chất projective
- Luôn được đánh giá cao về mặt tốc độ, chỉ với độ phức tạp O(n) với n là độ dài của câu
- Có nhiều biến thể: Arc-Eager, Arc-Standard, Hybrid.
- Phần này chúng ta tìm hiểu hệ thống Arc-Eager Transition

2.2.2.Các thuật ngữ.

Với thuật toán này, ta có:

- 1 Stack **Σ** chứa các từ từng được xét, và sẽ có thể được xét tiếp.

- 1 Buffer **B** chứa các từ chưa được xét, hoặc mới xét đến.

- 1 tập **A** chứa các quan hệ phụ thuộc đã tìm ra

Ở mỗi thời điểm, hệ thống sẽ xem xét từ ở đỉnh Stack và từ ở đầu Buffer có quan hệ phụ thuộc gì không.

2.2.2.Các thuật ngữ.

- Configuration c = (**Σ**|s, b|**B**, **A**)
  - 1 configuration như trên thể hiện 1 trạng thái/cấu hình của hệ thống khi phân tích câu.
- Nó bao gồm 3 thành phần như trên:
  - Stack **Σ** chứa từ ở đỉnh stack là s
  - Buffer **B** chứa từ b ở đầu tiên (từ tiếp theo xét đến là b)
  - Tập **A** các quan hệ phụ thuộc đã tìm ra
- Quan hệ phụ thuộc (s, lb, b)
  - Quan hệ với head là s, dependent là b, nhãn quan hệ là lb.

2.2.2.Các thuật ngữ.

- Dựa trên configuration, người ta định nghĩa ra 4 Transition như sau:
  - SHIFT [(**Σ**, b|**B**, **A**)] = (**Σ**|b, **B**, **A**)
  - $RIGHT_{lb}$ [(**Σ**|s, b|**B**, **A**)] = (**Σ**|s|b, **B**, **A** ∪ {s, lb, b})
  - $LEFT_{lb}$ [(**Σ**|s, b|**B**, **A**)] = (**Σ**, b|**B**, **A** ∪ {b, lb, s})
  - REDUCE [(**Σ**|s, **B**, **A**)] = (**Σ**, **B**, **A**)
- Có thể mô tả tương ứng như sau:
  - SHIFT: chuyển từ ở đầu buffer lên đỉnh của stack, không thêm quan hệ
  - RIGHT: thêm từ ở đầu buffer vào đỉnh stack, thêm quan hệ phụ thuộc (s, lb, b) ((s->b) là hướng sang phải nên gọi là RIGHT)
  - LEFT: bỏ từ ở đỉnh stack ra, giữa nguyên buffer, thêm quan hệ phụ thuộc (b, lb, s) ((b->s) là hướng sang trái nên gọi là LEFT)
  - REDUCE: bỏ từ ở đỉnh Stack đi, không thêm quan hệ nào

2.2.3.Thuật toán.

- Câu đầu vào là W = w_1, w_2, ..., w_n. (w_i là từ thứ i trong câu)
- Cấu hình khởi tạo: c_{init} = (**Σ**, **B**, **A**)
  - **Σ**: Stack chỉ chứa 1 nút ROOT
  - **B**: **B** = w_1, w_2, ..., w_n (chứa tất cả các từ của câu)
  - **A**: tập rỗng
- Cấu hình kết thúc: $c_{terminal}$ = (**Σ**, **B**, **A**)
  - **Σ**: chỉ chứa ROOT
  - **B**: rỗng
  - **A**: tập chứa quan hệ phụ thuộc.

2.2.3.Thuật toán.

Các ký hiệu trong thuật toán:

- **INITIAL (W)**: hệ thống khởi tạo cấu hình ban đầu theo câu W
- **TERMINAL (C)**: kiểm tra xem C có phải là cấu hình kết thúc không
- **LEGAL (C)**: Tập các transition hợp lệ với cấu hình C
- $\mathbf{t_p}$: transition tiếp theo
- **w**: các tham số của mô hình quyết định transition (classifier), vector này cần được học và sẽ trình bày trong phần 2.3
- **ϕ(c, t)**: Các đặc trưng được tính toán tương ứng với cấu hình c sau transition t, sẽ trình bày trong phần 2.4.

2.2.3.Thuật toán.

Thuật toán được thực hiện như sau:

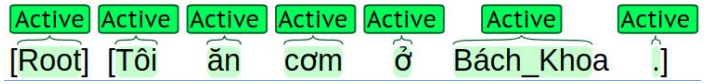
1: **Input**: sentence W , parameter-vector w
2: $c \leftarrow$ **INITIAL** (W)
3: **while** not **TERMINAL** (c) do
4: $t_p \leftarrow \arg\max_{t \in \text{LEGAL}(c)} w \cdot \phi(c, t)$
5: $c \leftarrow t_p(c)$
6: **return** A_c

2.2.3.Thuật toán.

- Có thể mô tả lại bằng lời như sau:
  - Đầu tiên khởi tạo configuration tương ứng với câu đầu vào
  - Sau đó, liên tục lựa chọn các transition có khả năng nhất và đi theo transition đấy cho tới khi gặp configuration kết thúc.
  - Trả về tập các quan hệ phụ thuộc.

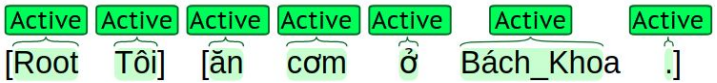
- Quá trình lựa chọn transition được quyết định bởi bộ classifier, sẽ được trình bày trong phần học.
- Độ phức tạp O(n). Dễ thấy bộ phân tích cứ đi lần lượt từ đầu đến cuối câu là tới kết quả cuối cùng.

2.2.4.Ví dụ.



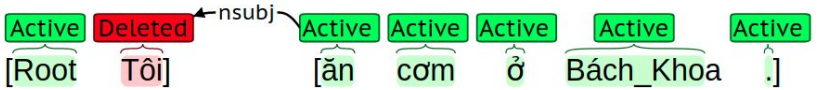
- Câu đầu vào: **Tôi ăn cơm ở Bách_Khoa** .
- Stack: Ngoặc vuông bên trái
- Buffer: Ngoặc vuông bên phải.
- A : tập quan hệ phụ thuộc đang rỗng
- Active: Nút vẫn đang còn được xét.
- Deleted: Nút đã xét xong, loại bỏ khỏi Stack

2.2.4.Ví dụ.



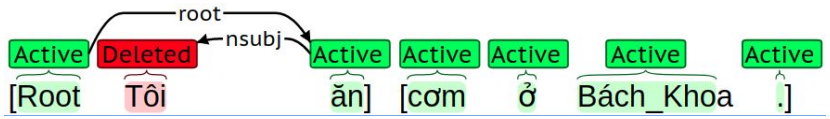
SHIFT: chuyển 'Tôi' từ Buffer sang Stack

2.2.4.Ví dụ.



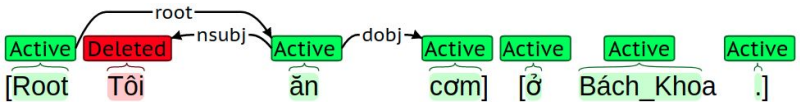
$LEFT_{nsubj}$: Xóa 'Tôi' khỏi Stack, thêm (ăn, nsubj, Tôi) vào tập A
(Đỉnh Stack hiện tại là Root)

2.2.4.Ví dụ.



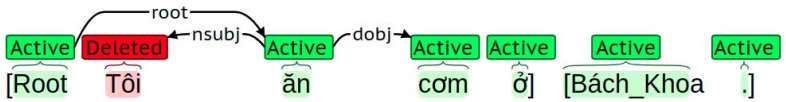
$RIGHT_{root}$: Thêm 'ăn' từ bufer vào stack, thêm (Root, root, ăn) vào tập A
(Đỉnh Stack hiện tại là 'ăn')

2.2.4.Ví dụ.



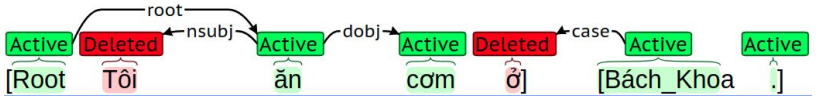
$RIGHT_{dobj}$: Thêm 'cơm' từ buffer vào stack, thêm (ăn, dobj, cơm) vào tập A
(Đỉnh Stack hiện tại là 'cơm')

2.2.4.Ví dụ.



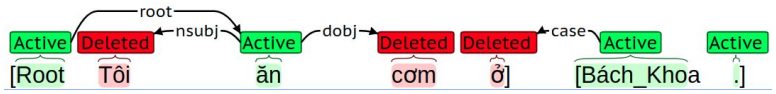
SHIFT: chuyển 'ở' từ buffer sang stack

2.2.4.Ví dụ.



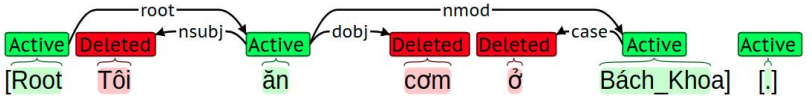
$LEFT_{case}$: Xóa 'ở' khỏi Stack, thêm (Bách_Khoa, case, ở) vào tập A
(Đỉnh Stack hiện tại là cơm)

2.2.4.Ví dụ.



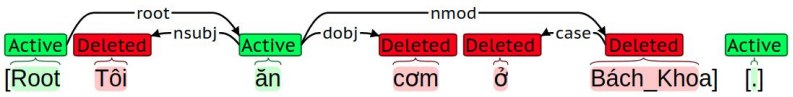
REDUCE: Xóa 'cơm' khỏi Stack
(Đỉnh Stack hiện tại là 'ăn')

2.2.4.Ví dụ.



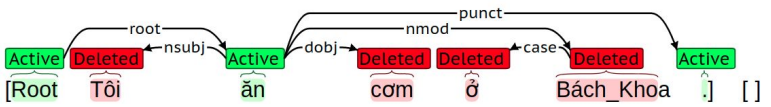
$RIGHT_{nmod}$: Thêm 'Bách_Khoa' từ buffer vào stack, thêm (ăn, nmod, Bách_Khoa) vào tập A
(Đỉnh Stack hiện tại là 'Bách_Khoa')

2.2.4.Ví dụ.



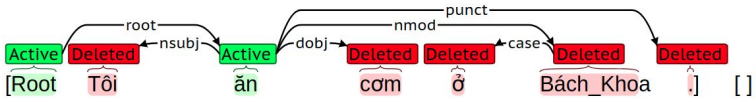
REDUCE: Xóa 'Bách_Khoa' khỏi Stack
(Đỉnh Stack hiện tại là 'ăn')

2.2.4.Ví dụ.



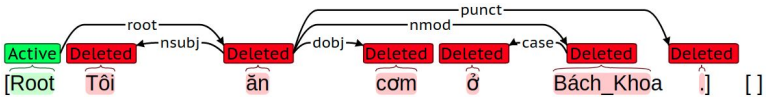
RIGHT$_{nmod}$: Thêm '.' từ buffer vào stack, thêm (ăn, punct, .) vào tập A
(Đỉnh Stack hiện tại là '.')

2.2.4.Ví dụ.



REDUCE: Xóa '.' khỏi Stack
(Đỉnh Stack hiện tại là 'ăn')

2.2.4.Ví dụ.

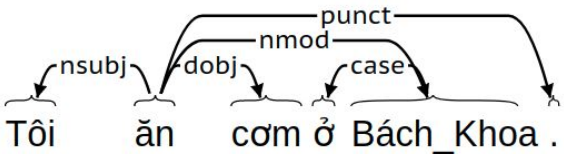


REDUCE: Xóa 'ăn' khỏi Stack
(Đỉnh Stack hiện tại là 'Root')

Hiện tại đã đến cấu hình cuối cùng, Stack chỉ có Root, Buffer rỗng. Hệ thống trả lại tập quan hệ phụ thuộc A.

2.2.4.Ví dụ.

Kết quả cuối cùng



2.3. Thuật toán học

2.3.1.Tổng quan.

- Trong giải thuật phân tích cú pháp, chúng ta cần phải xác định transition tiếp theo tương ứng với cấu hình hiện tại.
- Việc quyết định transition nào được lựa chọn sẽ do classifier đảm nhiệm. Vì vậy chúng ta cần học ra được các trọng số cho classifier này.
- Việc học này có thể sử dụng nhiều phương pháp như SVM hay neural network,...
- Để đơn giản, phần này chúng ta trình bày thuật toán học với mạng neural.

2.3.2.Dữ liệu học.

- Định dạng dữ liệu học: CoNLL-U Format
  - Mỗi từ được thể hiện trong 1 dòng
  - Mỗi dòng gồm 10 cột, chứa các thông tin: id, form, lemma, upos, xpos, feats, head, deprel, deps, misc
  - Các câu ngăn cách nhau bởi 1 dòng trắng
  - Thường được gọi là Treebank
- Trong bài toán này, chúng ta chỉ quan tâm đến các thông tin: id, form (từ), upos hoặc xpos (nhãn từ loại), head (id của head tương ứng với từ này), deprel (nhãn phụ thuộc)
- Xem thêm: http://universaldependencies.org/format.html

2.3.2.Dữ liệu học.

Ví dụ về dữ liệu học.

```
1   Nhưng       _   CC  CC  _   8   cc      _   _
2   có_vẻ       _   RB  RB  _   8   advmod  _   _
3   như         _   IN  IN  _   8   mark    _   _
4   rất         _   RB  RB  _   5   advmod  _   _
5   nhiều       _   JJ  JJ  _   6   amod    _   _
6   người       _   NN  NN  _   8   nsubj   _   _
7   chưa        _   RB  RB  _   8   neg     _   _
8   biết        _   VB  VB  _   0   ROOT    _   _
9   về          _   IN  IN  _   10  case    _   _
10  nấm         _   NN  NN  _   8   nmod    _   _
11  Agaricus    _   NNP NNP _   10  nmod    _   _
12  cùng        _   IN  IN  _   13  case    _   _
13  công_dụng   _   NN  NN  _   10  nmod    _   _
14  vượt_trội   _   JJ  JJ  _   13  amod    _   _
15  từ          _   IN  IN  _   16  case    _   _
16  nó          _   PRP PRP _   13  nmod    _   _
17  .           _   PUNCT   PUNCT   _   8   punct   _   _

1   Nhằm        _   TO  TO  _   2   mark    _   _
2   hưởng_ứng   _   VB  VB  _   0   ROOT    _   _
3   chương_trình    _   NN  NN  _   2   dobj    _   _
4   “           _   PUNCT   PUNCT   _   5   punct   _   _
5   Hành_trình  _   NN  NN  _   3   nmod    _   _
6   đó          _   JJ  JJ  _   5   amod    _   _
7   ”           _   PUNCT   PUNCT   _   5   punct   _   _
```


2.3.3.Thuật toán học.

Các ký hiệu trong thuật toán:

- **INITIAL (W)**, **TERMINAL (C), LEGAL(c)**, **w, ϕ(c, t)**: xem lại slide 29
- **T**: gold tree được xây dựng từ dữ liệu học
- **CORRECT (C)**: Tập các transition đúng với cấu hình c hiện tại , tức là các transition có thể dẫn đến gold tree.
- **o(t; c, T)**: trả về true nếu cấu hình c có thể dẫn đến cây gold tree T, còn lại False
- **UPDATE (w, ϕ(c, t_o), ϕ(c, t_p))**: Cập nhật trọng số w của mô hình với sự mất mát của transition đúng t_o và transition được mô hình dự đoán ra t_p.
- **NEXT (c, t_o)**: chuyển sang cấu hình tiếp theo bằng cách cho cấu hình c đi theo transition t_0

2.3.3.Thuật toán học.

for sentence W with gold tree T **in** corpus **do**
    c ← **INITIAL** (W)
    while not TERMINAL (c) **do**
        CORRECT (c) ← {t | **o**(t; c, T) = true}
        t_p ← **arg max** $_{t \in \text{LEGAL}(c)}$ w · ϕ(c, t)
        t_o ← **arg max** $_{t \in \text{CORRECT}(c)}$ w · ϕ(c, t)
        if $t_p \notin$ **CORRECT** (c) **then**
            UPDATE (w, ϕ(c, t_o), ϕ(c, t_p))
            c ← **NEXT** (c, t_o)
        else
            c ← t_p(c)

2.3.3.Thuật toán học.

- Có thể thấy, thuật toán học máy này cũng giống như các thuật toán thông thường, cập nhật lại trọng số khi mô hình dự đoán sai, còn dự đoán đúng thì không cập nhật.
- Các phiên bản đầu tiên của thuật toán chỉ chấp nhận duy nhất 1 transition tại mỗi lần quyết định là đúng (Static Oracle), sau này, thuật toán được cải tiến và cho phép nhiều hơn 1 transition là đúng (Dynamic Oracle)

2.4.Lựa chọn đặc trưng.

- Có thể thấy, trong phần học classifier để quyết định transition, chúng ta học ra được bộ tham số cho classifier này.
- Như vậy, cần xác định các đặc trưng cần thiết làm đầu vào cho classfier.
- Phần này sẽ nêu ra các đặc trưng được sử dụng phổ biến trong bài toán phân tích cú pháp phụ thuộc này.
- Các đặc trưng này được lựa chọn bởi các chuyên gia trong bài toán này, thông thường các nghiên cứu phía sau chỉ lấy dùng trong bộ các đặc trưng này

2.4.Lựa chọn đặc trưng.

- Đặc trưng gồm 2 phần:
  - **Core feature**: các đặc trưng lõi, dùng để cấu thành nên các feature template, ví dụ chủ yếu là word, pos tag, dependency label, …
  - **Template feature**: là các template kết hợp từ các core feature, dùng để làm đầu vào cho bộ phân loại
- Xem thêm về các template feature ở trang sau.
- Các template này được lấy ra từ bài báo: **Transition-based Dependency Parsing with Rich Non-local Features** của **Yue Zhang** và **Joakim Nivre**

2.4.Lựa chọn đặc trưng.

Baseline Feature Template

- *S*: Stack
- *N*: Buffer
- *Chỉ số 0, 1, ...*: từ thứ bao nhiêu trong Stack hoặc Buffer.
- *0l* hoặc *0r*: con trái nhất hoặc con phải nhất (dependent)
- *0h*: cha của từ này (head)

from single words
S_0wp; S_0w; S_0p; N_0wp; N_0w; N_0p; N_1wp; N_1w; N_1p; N_2wp; N_2w; N_2p;
from word pairs
S_0wpN_0wp; S_0wpN_0w; S_0wN_0wp; S_0wpN_0p; S_0pN_0wp; S_0wN_0w; S_0pN_0p N_0pN_1p
from three words
$N_0pN_1pN_2p$; $S_0pN_0pN_1p$; $S_{0h}pS_0pN_0p$; $S_0pS_{0l}pN_0p$; $S_0pS_{0r}pN_0p$; $S_0pN_0pN_{0l}p$

Table 1: Baseline feature templates.
w – word; p – POS-tag.

2.4.Lựa chọn đặc trưng.

New Feature Template

- *valency*: số lượng modifier (dependent) của từ này
- *label set*: tập các label của các quan hệ phụ thuộc nối với từ này
- *l, r*: tính theo bên trái hoặc bên phải.

distance
S_0wd; S_0pd; N_0wd; N_0pd; S_0wN_0wd; S_0pN_0pd;
valency
S_0wv_r; S_0pv_r; S_0wv_l; S_0pv_l; N_0wv_l; N_0pv_l;
unigrams
$S_{0h}w$; $S_{0h}p$; S_0l; $S_{0l}w$; $S_{0l}p$; $S_{0l}l$; $S_{0r}w$; $S_{0r}p$; $S_{0r}l$;$N_{0l}w$; $N_{0l}p$; $N_{0l}l$;
third-order
$S_{0h2}w$; $S_{0h2}p$; $S_{0h}l$; $S_{0l2}w$; $S_{0l2}p$; $S_{0l2}l$; $S_{0r2}w$; $S_{0r2}p$; $S_{0r2}l$; $N_{0l2}w$; $N_{0l2}p$; $N_{0l2}l$; $S_0pS_{0l}pS_{0l2}p$; $S_0pS_{0r}pS_{0r2}p$; $S_0pS_{0h}pS_{0h2}p$; $N_0pN_{0l}pN_{0l2}p$;
label set
S_0ws_r; S_0ps_r; S_0ws_l; S_0ps_l; N_0ws_l; N_0ps_l;

Table 2: New feature templates.
w – word; p – POS-tag; v_l, v_r – valency; l – dependency label, s_l, s_r – labelset.

2.5.Các kỹ thuật cải tiến.

2.5.1. Error Exploration
2.5.2. Beam search

2.5.1.Error Exploration.

- Kỹ thuật này sử dụng trong quá trình học bộ classifier
- Cho phép hệ thống đi theo 1 transition mà biết là sai (theo 1 xác suất nào đấy)
- Cố gắng tối ưu sao cho các transition tiếp theo đi càng về gần cây đúng (gold tree) càng tốt.
- Nhờ việc này, khi chạy thử nghiệm mô hình, nếu có một transition bị dự đoán sai, thì việc dự đoán các transition phía sau sẽ cố gắng sao càng về cây đúng càng tốt.

2.5.1.Error Exploration.

Xem hàm **CHOOSE_NEXT$_{EXP}$**

Algorithm 3 Online training with a dynamic oracle

```
w ← 0
for I = 1 → ITERATIONS do
    for sentence x with gold tree G_gold in corpus do
        c ← c_s(x)
        while c is not terminal do
            t_p ← arg max_t w · φ(c, t)
            ZERO_COST ← {t | o(t; c, G_gold) = true}
            t_o ← arg max_{t∈ZERO_COST} w · φ(c, t)
            if t_p ∉ ZERO_COST then
                w ← w + φ(c, t_o) − φ(c, t_p)
            t_n ← CHOOSE_NEXT(I, t_p, ZERO_COST)
            c ← t_n(c)
return w
```

```
function CHOOSE_NEXT_ARB(I, t, ZERO_COST)
    if t ∈ ZERO_COST then
        return t
    else
        return RANDOM_ELEMENT(ZERO_COST)
```

```
function CHOOSE_NEXT_EXP(I, t, ZERO_COST)
    if I > k and RAND() > p then
        return t
    else
        return CHOOSE_NEXT_ARB(I, t, ZERO_COST)
```

Goldberg and Nivre 2012.
(A Dynamic Oracle for Arc-Eager Dependency Parsing)

2.5.2.Beam Search.

- Kỹ thuật tìm kiếm chùm này sử dụng trong quá trình phân tích (Parsing)
- Thông thường các thuật toán phân tích hướng transition based chỉ theo 1 đường duy nhất dựa trên các transition tốt nhất mà nó chọn ra
- Beam Search cho phép lưu trữ và tìm kiếm theo một số lượng con đường tốt nhất (beam-size), nhờ vậy tăng khả năng tìm được cây đúng hơn, nhưng hạn chế là làm chậm quá trình phân tích
- beam-size=1: là thuật toán thông thường.

2.5.2.Beam Search.

Có thể thấy ở hình bên, tập BEAM lưu trữ Q con đường tốt nhất hệ thống tìm ra được.
Score ở đây do bộ phân loại quyết định

PARSE($x = (w_0, w_1, \ldots, w_n)$)
1 BEAM $\leftarrow \{(c_s(x), 0.0)\}$
2 **while** $\exists \langle c, s \rangle \in$ BEAM $: c \notin C_t$
3 NEWBEAM $\leftarrow \emptyset$
4 **for every** $(c, s) \in$ BEAM
5 **for every** $t \in T$
6 NEWBEAM $\leftarrow$ NEWBEAM $\cup \{\langle t(c), s + \text{SCORE}(c, t) \rangle\}$
7 BEAM $\rightarrow$ QBEST(NEWBEAM)
8 **return** $\leftarrow$ 1BEST(BEAM)

3.Phương pháp Graph-based

3.1. Giới thiệu.

3.2. Thuật toán phân tích cú pháp.

3.3. Thuật toán học.

3.4. Lựa chọn đặc trưng.

3.1. Giới thiệu.

- 1 phương pháp phổ biến trong bài toán phân tích cú pháp phụ thuộc.
- Ý tưởng của nó là bài toán tìm cây khung cực đại (MST)
- Với câu đầu vào, ta xem nó như là 1 đồ thị gồm n+1 đỉnh (n là độ dài câu + 1 nút ROOT), giữa 2 đỉnh bất kỳ luôn có các cạnh thể hiện cho các quan hệ và trọng số thể hiện khả năng giữa 2 từ này có quan hệ phụ thuộc.
- Trọng số được quyết định bởi 1 mô hình học máy.
- Tìm ra cây khung lớn nhất từ đồ thị trên (có thể có ràng buộc Projective hoặc không), do làm theo kiểu vét cạn nên có khả năng tìm được cây đúng nhiều hơn.
- Nhược điểm là độ phức tạp lớn, cỡ $O(|L|*n^2)$ với L là tập nhãn, n là độ dài câu

3.2.Thuật toán phân tích cú pháp.

3.2.1. Thuật toán Chu-Liu-Edmond

3.2.2. Thuật toán Eisner

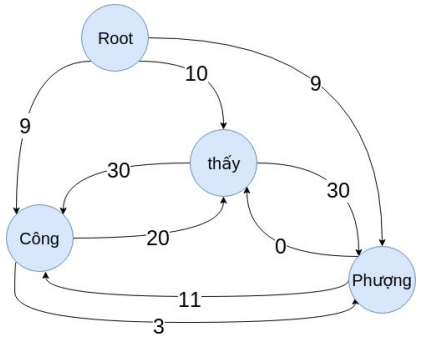
3.2.1.Thuật toán Chu-Liu-Edmond.

3.2.1.1.Tổng quan.

- Thuật toán Chu-Liu-Edmond là một thuật toán tìm cây khung lớn nhất cho đồ thị có hướng.
- Cây tìm ra có thể không thỏa mãn tính chất Projective
- Độ phức tạp: $O(|L|*n^2)$
  - L là tập nhãn
  - n là độ dài câu

3.2.1.2.Thuật toán.

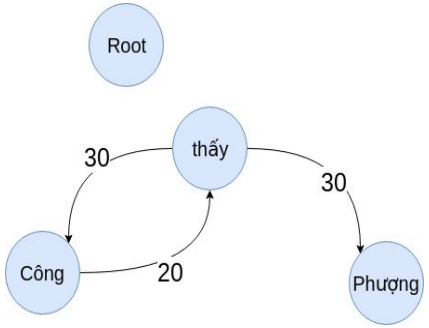
- Câu đầu vào: **Công thấy Phượng**
- Đồ thị được xây dựng như hình bên:
  - Thêm nút root vào
  - Trọng số trên các cạnh được cho như hình bên, thực tế nó được quyết định bởi mô hình học máy
- Ở đây ta xem như không có nhãn cho đơn giản



3.2.1.2.Thuật toán Chu-Liu-Edmond.

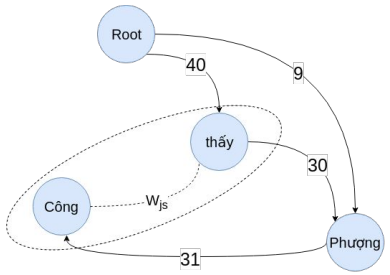
- Loại bỏ các cạnh đi từ root.
- Với mỗi đỉnh, tìm ra cạnh vào (mũi tên đi vào) có trọng số lớn nhất. Loại bỏ các cạnh còn lại
- Nếu đồ thị còn lại sau các thao tác trên là cây, thì đó là cây cần tìm

Ở đây, ta có 1 chu trình giữa 2 từ **Công** và **thấy**, nên chưa có đáp án, cần tiếp tục thực hiện.



3.2.1.2.Thuật toán Chu-Liu-Edmond.

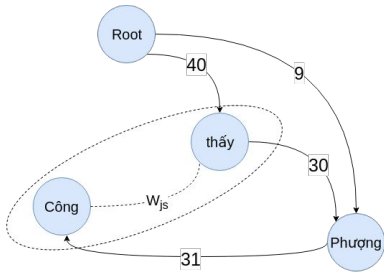
- Xét các chu trình có trong đồ thị mới, gộp chúng lại thành 1 đỉnh, tính toán lại trọng số các cạnh. Việc này được gọi là contract. Ở đây ta gộp 2 đỉnh **Công** và **thấy**.
- Nếu các cạnh đi từ các đỉnh trong chu trình đến 1 đỉnh ngoài chu trình, chọn cạnh có trọng số lớn nhất. Ở đây chọn **thấy->Phượng** với trọng số 30. (Do **Công->Phượng** là 3 < 30)



3.2.1.2.Thuật toán Chu-Liu-Edmond.

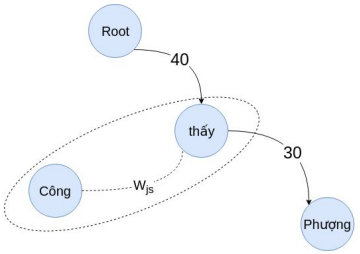
- Nếu các cạnh đi từ 1 đỉnh ngoài chu trình đến 1 đỉnh trong chu trình, chọn trọng số là tổng trọng số của cây khung lớn nhất chứa đỉnh ngoài chu trình đấy và các đỉnh trong chu trình
- Ví dụ ở đây với đỉnh **root** đến chu trình. Ta thấy:
  - **root->Công->thấy**: 9+20 = 29
  - **root->thấy->Công**: 10 + 30 = 40

Chọn **root->thấy->Công**: 40



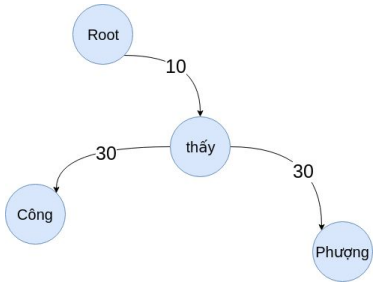
3.2.1.2.Thuật toán Chu-Liu-Edmond.

- Thực hiện gọi đệ quy giải thuật Chu-Liu-Edmond, tìm ra các cạnh đi tới có trọng số lớn nhất, loại bỏ các cạnh khác. Chú ý chu trình vẫn đang được xem là 1 đỉnh.



3.2.1.2.Thuật toán Chu-Liu-Edmond.

- Thực hiện recontract (lấy cây tạo bởi các đỉnh của chu trình) và trả về cây khung cần tìm.



3.2.1.2.Thuật toán.

Hình bên là mã giả của thuật toán này

Khi gặp chu trình, thuật toán kết nối các đỉnh và tính lại trọng số với hàm **contract** (dòng 5), sau đấy gọi lại hàm Chu-Liu-Emond (dòng 6)

Chu-Liu-Edmonds(G, s)
Graph $G = (V, E)$
Edge weight function $s : E \rightarrow \mathbb{R}$
1. Let $M = \{(x^*, x) : x \in V, x^* = \arg\max_{x'} s(x', x)\}$
2. Let $G_M = (V, M)$
3. If G_M has no cycles, then it is an MST: return G_M
4. Otherwise, find a cycle C in G_M
5. Let $G_C = \text{contract}(G, C, s)$
6. Let $\boldsymbol{y} = \text{Chu-Liu-Edmonds}(G_C, s)$
7. Find a vertex $x \in C$ s. t. $(x', x) \in \boldsymbol{y}, (x'', x) \in C$
8. return $\boldsymbol{y} \cup C - \{(x'', x)\}$

contract$(G = (V, E), C, s)$
1. Let G_C be the subgraph of G excluding nodes in C
2. Add a node c to G_C representing cycle C
3. For $x \in V - C : \exists_{x' \in C}(x', x) \in E$
   Add edge (c, x) to G_C with
   $s(c, x) = \max_{x' \in C} s(x', x)$
4. For $x \in V - C : \exists_{x' \in C}(x, x') \in E$
   Add edge (x, c) to G_C with
   $s(x, c) = \max_{x' \in C} [s(x, x') - s(a(x'), x') + s(C)]$
   where $a(v)$ is the predecessor of v in C
   and $s(C) = \sum_{v \in C} s(a(v), v)$
5. return G_C

Marina Valeeva.
(Slide Graph-based dependency parsing)

3.2.2.Thuật toán Eisner.

3.2.2.1. Tổng quan.

3.2.2.2. Các thuật ngữ.

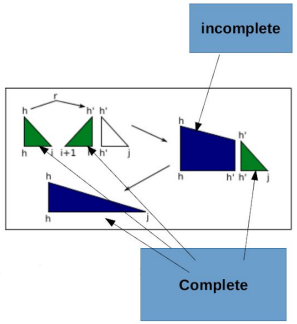
3.2.2.3. Thuật toán.

3.2.2.1.Tổng quan.

- Thuật toán Einser cũng là 1 thuật toán tìm cây khung lớn nhất cho đồ thị có hướng.
- Thuộc kiểu giải thuật quy hoạch động theo hướng bottom-up, xây các cây con nhỏ, rồi kết hợp chúng để xây dựng các cây lớn hơn. Tiếp tục đến khi có được cây tốt nhất và chứa toàn bộ từ trong câu
- Cho phép chỉ tìm ra cây thỏa mãn tính chất Projective
- Độ phức tạp thuật toán: $O(n^3)$
  - n: độ dài của câu

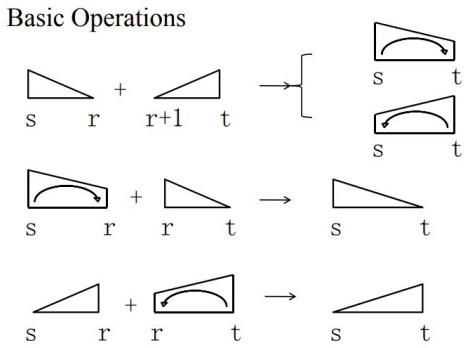
3.2.2.2.Các thuật ngữ.

- Các thuật ngữ:
  - Complete span
  - Incomplete span
- Hai thành phần này khó định nghĩa riêng, việc xây dựng có tính đệ quy.
  - Bắt đầu các từ trong câu là các complete span
  - Sau đấy, 2 complete span kết nối với nhau tạo ra 1 incomplete span.
  - Rồi 1 incomplete span kết nối với 1 complete span lại tạo ra 1 complete span



3.2.2.2.Các thuật ngữ.

Hình bên thể hiện các thao tác kết hợp các span



Wenliang Chen, Zhenghua Li, Min Zhang.

(Slide Dependency Parsing)

3.2.2.3.Thuật toán Eisner.

Algorithm 1 Eisner's Dynamic Programming Algorithm

Initialize the table: $C[s][s][d][c] = 0; \quad \forall s \in \{0, ..., n\}, d \in \{\leftarrow, \rightarrow\}, c \in \{0, 1\}$
for k in 1 to n **do**
  for s in 0 to n **do**
    $t = s + k$
    if $t > n$ **then**
      break
    end if
    //build the incomplete spans
    $C[s][t][\leftarrow][0] = \max_{s \le u < t}(C[s][u][\rightarrow][1] + C[u+1][t][\leftarrow][1] + s(t, s))$
    $C[s][t][\rightarrow][0] = \max_{s \le u < t}(C[s][u][\rightarrow][1] + C[u+1][t][\leftarrow][1] + s(s, t))$
    //build the complete spans
    $C[s][t][\leftarrow][1] = \max_{s \le u < t}(C[s][u][\leftarrow][1] + C[u][t][\leftarrow][0])$
    $C[s][t][\rightarrow][1] = \max_{s \le u < t}(C[s][u][\rightarrow][0] + C[u][t][\rightarrow][1])$
  end for
end for
return $C[0][n][\rightarrow][1]$ as the best parse.

3.2.2.3.Thuật toán Eisner.

C[s][t][d][c]: thể hiện 1 span từ từ ở vị trí thứ s đến vị trí t

d = $\rightarrow$: head bên phải và dependent bên trái.

d = $\leftarrow$: head bên trái và dependent bên phải.

c = 1 : nếu là complete span

c = 0 : nếu là incomplete span

3.2.2.3.Thuật toán Eisner.

Ý tưởng đấy có thể mô tả như sau:

- Đầu tiên khởi tạo các span chỉ chứa các từ độc lập với điểm bằng 0
- Sau đó xây dựng các span chứa 2 từ liên tiếp nhau từ các span là 1 từ ở bước trên, tính điểm dựa trên các cạnh kết nối.
- Tiếp tục xây dựng các span chứa 3 từ liên tiếp nhau từ các span 1 từ và 2 từ ở trên.
- Cứ tiếp xây dựng lên cho đến khi xây dựng được 1 complete span chứa tất cả các từ trong câu.
- Trong quá trình này, điểm của các span được xác định là tổng điểm các cạnh, điểm này được xác định bởi 1 mô hình học máy

3.3.Thuật toán học.

3.3.1. Tổng quan.

3.3.2. Dữ liệu học.

3.3.3. Thuật toán học.

3.3.1. Tổng quan.

- Trong phương pháp graph-based, chúng ta cần tìm ra cây khung lớn nhất.
- Để tìm được cây khung lớn nhất, ta cần tính được trọng số (hay điểm) cho các cạnh của đồ thị.
- Một mô hình học máy được huấn luyện để tính điểm cho các cạnh, ở phần này, ta tìm hiểu về cách học tham số cho mô hình này.
- Có nhiều phương pháp để học, như mạng neural, svm, ...

3.3.2. Dữ liệu học.

Giống như dữ liệu học của phần Transition based.

3.3.3. Thuật toán học.

- Với câu đầu vào x, cây thu được là y, điểm số của cây này được tính là :

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j) \in \boldsymbol{y}} s(i,j) = \sum_{(i,j) \in \boldsymbol{y}} \mathbf{w} \cdot \mathbf{f}(i,j)$$

- Trong đó
  - s(x, y) là điểm của cây cú pháp phụ thuộc y đối với câu đầu vào là x.
  - s(i, j) là điểm của cạnh i->j trong cây cú pháp phụ thuộc y
  - w là trọng số học được.
  - f(i,j) là giá trị tương ứng với các feature (xem phần feature extractor - 3.4)
- Việc của quá trình học là học ra bộ trọng số w của mô hình tính điểm

3.3.3. Thuật toán học.

- Dữ liệu huấn luyện: $T = \{(x_t, y_t)\}_{t=1}^{T}$
  - x_t: câu đầu vào thứ t
  - y_t: cây cú pháp phụ thuộc chính xác tương ứng của câu x_t
- N: số lần lặp
- w: vector trọng số cần học
- v: tổng các vector trọng số
- Mã giả của thuật toán xem ở phần sau.

3.3.3. Thuật toán học.

1. $w_o = 0; v = 0; i = 0$
2. **for** n : 1..N
3. **for** t : 1..T
4. $w^{(i+1)} =$ **update** $w^{(i)}$ **according to instance** (x_t, y_t)
5. $v = v + w^{(i+1)}$
6. $i = i + 1$
7. $w = v/(N * T)$

3.3.3. Thuật toán học.

- Có thể thấy ở thuật toán trên, trọng số của lần tiếp theo được cập nhật theo trọng số của lần trước và dữ liệu của ví dụ hiện tại (x_t, y_t)
- Bộ trọng số cuối cùng được tính toán bằng trung bình của tất cả các bộ trọng số được tìm ra ở mỗi lần lặp (dòng 7)
- Việc cập nhật trọng số (dòng 4) có thể thực hiện theo nhiều phương pháp khác nhau, một trong những phương pháp phổ biến nhất hiện tại là MIRA (Margin Infused Relaxed Algorithm)
- Việc học với MIRA cố gắng cực đại khoảng cách giữa điểm của cây đúng (Gold tree) với cây sai.

3.3.3. Thuật toán học - MIRA.

1. Training data: $T = (x_t, y_t)_{t=1}^{T}$
2. $w_o = 0; (v) = 0; i = 0$
3. **for** n : 1 · · · N **do**
4. **for** t : 1 · · · T **do**
5. **min** $||w^{i+1} - w^i||$
6. s.t. $\mathbf{s}(x_t, y_t) - \mathbf{s}(x_t, y') \geq \mathbf{L}(y_t, y'), \forall y' \in \Phi(x_t)$
7. $v = v + w^{i+1}$
8. $i = i + 1$
9. **end for**
10. **end for**
11. $w = v/(N*T)$

3.3.3. Thuật toán học - MIRA.

- Trong thuật toán trên:
  - **Φ**(x_t): Tập các cây cú pháp phụ thuộc có thể sinh ra từ x_t
  - **L**(y_t , y'): Giá trị mất mát (loss) của cây gold tree y_t và cây đang xét y', thường bằng số cạnh có trong y' mà không có trong y_t.
- Có thể thấy ở dòng 6, phương pháp MIRA cố gắng làm sao cho khoảng cách giữa cây đúng và cây sai luôn lớn hơn hoặc ít nhất bằng số cạnh sai trong cây sai.
- Phần đầu vào cho việc học tính điểm sẽ là các đặc trưng được lựa chọn từ dữ liệu đầu vào, sẽ được trình bày trong phần tiếp theo.

3.4. Lựa chọn đặc trưng.

3.4.1. First-order
3.4.2. Higher-order

3.4.1. First Order.

a)

Basic Uni-gram Features
p-word, p-pos
p-word
p-pos
c-word, c-pos
c-word
c-pos

b)

Basic Big-ram Features
p-word, p-pos, c-word, c-pos
p-pos, c-word, c-pos
p-word, c-word, c-pos
p-word, p-pos, c-pos
p-word, p-pos, c-word
p-word, c-word
p-pos, c-pos

c)

In Between POS Features
p-pos, b-pos, c-pos
Surrounding Word POS Features
p-pos, p-pos+1, c-pos-1, c-pos
p-pos-1, p-pos, c-pos-1, c-pos
p-pos, p-pos+1, c-pos, c-pos+1
p-pos-1, p-pos, c-pos, c-pos+1

Trên đây là các đặc trưng được đề xuất bởi McDonald trong bài báo năm 2005. (Online Large-Margin Training of Dependency Parsers)

3.4.1. First Order.

a)

Basic Uni-gram Features
p-word, p-pos
p-word
p-pos
c-word, c-pos
c-word
c-pos

b)

Basic Big-ram Features
p-word, p-pos, c-word, c-pos
p-pos, c-word, c-pos
p-word, c-word, c-pos
p-word, p-pos, c-pos
p-word, p-pos, c-word
p-word, c-word
p-pos, c-pos

c)

In Between POS Features
p-pos, b-pos, c-pos
Surrounding Word POS Features
p-pos, p-pos+1, c-pos-1, c-pos
p-pos-1, p-pos, c-pos-1, c-pos
p-pos, p-pos+1, c-pos, c-pos+1
p-pos-1, p-pos, c-pos, c-pos+1

p: parent(head)
c: child (dependent)
b: between (các từ đứng ở giữa head và dependent)

3.4.1. First Order.

- Sau này, ông và các nhà nghiên cứu còn bổ sung thêm nhiều đặc trưng khác, như:
  - Khoảng cách giữa head và dependent (số từ đứng giữa 2 từ này)
  - Hướng quan hệ phụ thuộc (trái/phải)
  - Nhãn quan hệ phụ thuộc
- Các đặc trưng này vẫn được sử dụng rộng rãi cho tới hiện nay.

3.4.2. Higher Order.

- Việc lựa chọn đặc trưng ở trên có 1 điểm yếu là chỉ tính điểm trên 1 cạnh độc lập mà không quan tâm đến cạnh khác.
- Vì vậy, sau này người ta phát triển thêm và các đặc trưng second-order (quan sát 2 cạnh 1 lúc) và third-order (quan sát 3 cạnh 1 lúc)
- Tuy nhiên, việc quan sát thêm nhiều đặc trưng giúp tăng độ chính xác nhưng cũng gia tăng độ phức tạp.
- Với các phương pháp gần đây, đa phần đều chỉ sử dụng đặc trưng first-order, độ chính xác có kém hơn 1 chút nhưng vượt trội về mặt tốc độ.

3.4.2.Higher Order.

Có thể thấy ở hình bên:

- Second-Order quan tâm đến cả sibling (cạnh có cùng cha là head của cạnh hiện tại), hoặc grand (cạnh có cha của head)
- Third-Order quan tâm đến 2 sibling, hoặc 1 sibling và 1 grand