Combining Graph and Transition-based parsers

Nora Kumpikova

20. December

Table of contents

- Classification criteria:

- Classification criteria:
    1. explicit transition-actions - *"Shift"* ; *"Reduce"*

- Classification criteria:
  1. explicit transition-actions - *"Shift"* ; *"Reduce"*
  2. assign scores - dependency graphs/transition actions

- Classification criteria:
  1. explicit transition-actions - *"Shift"* ; *"Reduce"*
  2. assign scores - dependency graphs/transition actions
- Graph-based parsers - find highest scoring parse tree
- Transition-based parsers - build parse from sequence of actions

- Classification criteria:
  1. explicit transition-actions - *"Shift"* ; *"Reduce"*
  2. assign scores - dependency graphs/transition actions
- Graph-based parsers - find highest scoring parse tree
- Transition-based parsers - build parse from sequence of actions
- Beam-search framework:
  1. high accuracy
  2. wider range of features

- Classification criteria:
  1. explicit transition-actions - *"Shift"* ; *"Reduce"*
  2. assign scores - dependency graphs/transition actions
- Graph-based parsers - find highest scoring parse tree
- Transition-based parsers - build parse from sequence of actions
- Beam-search framework:
  1. high accuracy
  2. wider range of features
- Data:
  English and Chinese PennTreebank

MSTParser and MaltParser

MSTParser | **MaltParser**

MSTParser and MaltParser

MSTParser

- exact inference

MaltParser

- deterministic

MSTParser and MaltParser

MSTParser

- exact inference
- constrained features

MaltParser

- deterministic
- large feature range

MSTParser and MaltParser

MSTParser

- exact inference
- constrained features

- *Research:*

MaltParser

- deterministic
- large feature range

MSTParser and MaltParser

MSTParser

- exact inference
- constrained features

MaltParser

- deterministic
- large feature range

- *Research:*
  1. defining features for graph-based parsing
  2. add search to transition-based parsing
  3. combine both to utilize strengths

Graph-based Parser

- uses same features as MSTParser
- Problem:

$$F(x) = \underset{y \in \mathrm{GEN}(x)}{\arg\max}\, \mathrm{Score}(y)$$

- Score of output(linear model):

$$\mathrm{Score}(y) = \Phi(y) \cdot \vec{w}$$

Perceptron Algorithm

Inputs: training examples (x_i, y_i)
Initialization: set $\vec{w} = 0$
Algorithm:
// R training iterations; N examples
for $t = 1..R$, $i = 1..N$:
  $z_i = \arg\max_{y \in \text{GEN}(x_i)} \Phi(y) \cdot \vec{w}$
  if $z_i \neq y_i$:
    $\vec{w} = \vec{w} + \Phi(y_i) - \Phi(z_i)$
Outputs: $\vec{w}$

- train values of the weight vector
- $\Phi(y)$, $\Phi(z)$ - global feature vector

Summary

1. for each POS-tagged input sentence → loop through partial parse tree
2. works incrementally
3. build parse tree - word by word, adding links inbetween current/predecessors
4. stores best items for each processing stage
5. after processing - pick best output from storage
6. uses "early update" strategy

Notes:

- improve learning by avoiding irrelevant information

w - word
t - POS tag

1	leftmost (CLC) and rightmost (CRC) children of C	PtCtCLCt; PtCtCRCt
2	left (la) and right (ra) arity of P	Ptla; Ptra; Pwtla; Pwtra

Parent word (P)	Pw; Pt; Pwt
Child word (C)	Cw; Ct; Cwt
P and C	PwtCwt; PwtCw; PwCwt; PwtCt; PtCwt; PwCw; PtCt
A tag Bt between P, C	PtBtCt
Neighbour words of P, C, left (PL/CL) and right (PR/CR)	PtPLtCtCLt; PtPLtCtCRt; PtPRtCtCLt; PtPRtCtCRt; PtPLtCLt; PtPLtCRt; PtPRtCLt; PtPRtCRt; PLtCtCLt; PLtCtCRt; PRtCtCLt; PRtCtCRt; PtCtCLt; PtCtCRt; PtPLtCt; PtPRtCt
sibling (S) of C	CwSw; CtSt; CwSt; CtSw; PtCtSt;

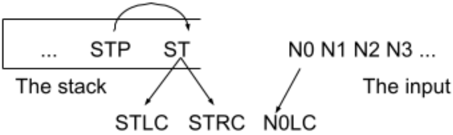
- uses the transition model of MaltParser
- deterministic - choses transition action for each step
- stack, 4 transition actions: *Shift, ArcRight, ArcLeft, Reduce*
- builds tree through repeated application of transition actions

Processing

1. input - processed left to right, word index maintained
2. stack - stores unfinished words
3. **Shift** - pushes current word to the stack
4. **ArcRight** - adds a dependency link from the stack top to the current word
5. **ArcLeft** action adds a dependency link from the current word to the stack top
6. **Reduce** - pops the stack

Notes:

- **Shift** and **ArcRight** - push a word on to the stack; read the next input word
- **ArcLeft** and **Reduce** - pop the stack
- **ArcLeft** and **ArcRight** - add a link to the output
- Major drawback - error propagation



(ST) - top of stack
(STP) - parent
(STLC) - left most child
(STRC) - right most child
N0 - current word; (N1,N2...) - next words from input
N0LC - left most child of current word

$$T(s) = \arg\max_{T \in \text{ACTION}} \text{Score}(T, s)$$

- *s* - context
- **T** - action
- ACTION = Shift, ArcRight, ArcLeft,Reduce

Error reduction:

- keep track of multiple candidate outputs
- avoid making decisions too early

$$F(x) = \underset{y \in \mathrm{GEN}(x)}{\arg\max} \sum_{T' \in \mathrm{act}(y)} \mathrm{Score}(T', s_{T'})$$

- GEN(x) - set of candidates
- x - input
- F(x) - best output
- T' - one action
- act(y) - sequence
- $s_{T'}$ - context

1. state item - contains a partial parse tree, a stack configuration
2. apply all possible actions to each existing state item
3. generate new items
4. store item with the highest overall score

Final state items: Requirements

- have fully built parse trees
- have only one root word left of the stack

Features

1	stack top	STwt; STw; STt
2	current word	N0wt; N0w; N0t
3	next word	N1wt; N1w; N1t
4	ST and N0	STwtN0wt; STwtN0w; STwN0wt; STwtN0t; STtN0wt; STwN0w; STtN0t
5	POS bigram	N0tN1t
6	POS trigrams	N0tN1tN2t; STtN0tN1t; STPtSTtN0t; STtSTLCtN0t; STtSTRCtN0t; STtN0tN0LCt
7	N0 word	N0wN1tN2t; STtN0wN1t; STPtSTtN0w; STtSTLCtN0w; STtSTRCtN0w; STtN0wN0LCt

Graph/Transition-based parsers

Goal:
Improve parsing accuracy → combination of graph/transition-based parser
Similarities:
- build parse tree incrementally
- keep memory of comparable state items
- rank state items by score
- use the averaged perceptron
- "early update" training

The parser:

- global linear model
- union of feature templates
- decoder from the transition-based parser

Score model

$$\begin{aligned} Score_{\mathrm{C}}(y) &= Score_{\mathrm{G}}(y) + Score_{\mathrm{T}}(y) \\ &= \Phi_{\mathrm{G}}(y) \cdot \vec{w_{\mathrm{G}}} + \Phi_{\mathrm{T}}(y) \cdot \vec{w_{\mathrm{T}}} \end{aligned}$$

- concatenating feature vectors $\Phi_G(y)$ and $\Phi_T(y) \rightarrow$ global vector $\Phi_C(y)$
- concatenating weight vectors $\vec{w}_G$ and $\vec{w}_T \rightarrow$ weight vector $\vec{w}_C$

$$Score_{\mathrm{C}}(y) = \Phi_{\mathrm{C}}(y) \cdot \vec{w}_{\mathrm{C}}$$

Score model

$$\begin{aligned} Score_{\mathrm{C}}(y) &= Score_{\mathrm{G}}(y) + Score_{\mathrm{T}}(y) \\ &= \Phi_{\mathrm{G}}(y) \cdot \vec{w_{\mathrm{G}}} + \Phi_{\mathrm{T}}(y) \cdot \vec{w_{\mathrm{T}}} \end{aligned}$$

- concatenating feature vectors $\Phi_G(y)$ and $\Phi_T(y)$ $\rightarrow$ global vector $\Phi_C(y)$
- concatenating weight vectors $\vec{w}_G$ and $\vec{w}_T$ $\rightarrow$ weight vector $\vec{w}_C$

$$Score_{\mathrm{C}}(y) = \Phi_{\mathrm{C}}(y) \cdot \vec{w}_{\mathrm{C}}$$

1. linear model

Score model

$$Score_C(y) = Score_G(y) + Score_T(y)$$
$$= \Phi_G(y) \cdot \vec{w_G} + \Phi_T(y) \cdot \vec{w_T}$$

- concatenating feature vectors $\Phi_G(y)$ and $\Phi_T(y) \rightarrow$ global vector $\Phi_C(y)$
- concatenating weight vectors $\vec{w}_G$ and $\vec{w}_T \rightarrow$ weight vector $\vec{w}_C$

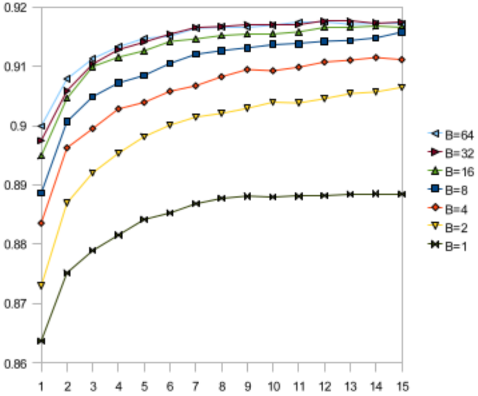
$$Score_C(y) = \Phi_C(y) \cdot \vec{w}_C$$

1. linear model
2. trained on perceptron algorithm

	Sections	Sentences	Words
Training	2–21	39,832	950,028
Dev	22	1,700	40,117
Test	23	2,416	56,684

Accuracy:
- precision of lexical heads
- percentage of complete matches

Beam size



- X-axis: number of training iterations
- Y-axis: precision of lexical heads

Accuracy comparison

	Word	Complete
MSTParser 1	90.7	36.7
Graph [M]	91.2	40.8
Transition	91.4	41.8
Graph [MA]	91.4	42.5
MSTParser 2	91.5	42.1
Combined [TM]	92.0	45.0
Combined [TMA]	92.1	45.4

word - precision of lexical head; complete - complete matches

- MSTParser 1/2 - first/second order MSTParsers
- Graph[M/MA] - graph-based parser
- Transition - transition-based parser
- Combined[TM/TMA] - combined parser

	Sections	Sentences	Words
Training	001–815; 1001–1136	16,118	437,859
Dev	886–931; 1148–1151	804	20,453
Test	816–885; 1137–1147	1,915	50,319

Table 6: Training, development and test data from CTB

	Non-root	Root	Comp.
Graph [MA]	83.86	71.38	29.82
Duan 2007	84.36	73.70	32.70
Transition	84.69	76.73	32.79
Combined [TM]	86.13	77.04	35.25
Combined [TMA]	86.21	76.26	34.41

Table 7: Test accuracies with CTB 5 data

Accuracy:

- percentage of non-root words with assigned correct head
- percentage of correctly identified root words
- percentage of complete matches

- successfully develop combined parser
- discriminative perceptron training and beam-search decoding
- significantly increased accuracy

References

- A Tale of Two Parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search (Yue Zhang, Stephen Clark)
- Characterizing the errors of data-driven dependency parsing models (Ryan McDonald, Joakim Nivre)