Language and Statistics II

Lecture 14: Practical Dynamic Programming
Noah Smith
Where we left off …

- Shieber, Schabes, and Pereira: logic programming (deduction) as a way to think about and implement parsers.
- Goodman: add weights!
- No implementation.

Meanwhile, in the “real” world of parsing …
- People were actually building weighted parsers!
- Crucial: good search strategy.
Beyond Goodman (1999)

- Goodman’s Algorithm: carry out deduction to build the chart (i.e., fill in items with nonzero value); then compute their values.
  - Tough part: **efficient** ordering of items.
  - For Forward/Viterbi: order by position
  - For CKY: order by width
  - In general?

- Would like efficient **execution strategy** for arbitrary programs.
  - Key idea: avoid unnecessary work and reproagation.
Indeed!

• The logic programs *don’t tell us how to implement the parser*!

• Is there a **generic** way to go about “compiling” a weighted logic program into a dynamic programming algorithm?
  – Yes: agenda-based DP (for Viterbi).
  – Does this generalize to arbitrary semirings?
Agenda

S → .8 NP VP

VP → .6 RB VB

RB → .04 never

VB → .006 win

NN → .00002 win

NNS → .002 quitters

...
Agenda

NP → .1
VP → .6 RB VB
S → .8 NP VP

NNS → .0002 quitters
NP → .1 NNS
RB → .04 never

VB → .006 win
NN → .00002 win

never

win

chart
Agenda

S → 0.8 NP VP
VP → 0.6 RB VB
RB → 0.04 never
VB → 0.006 win
NP → 0.1 NNS
NNS → 0.002 quitters
NN → 0.0002 win

chart

quitters

never

win
Agenda

never

chart

S →^8^ NP VP

NNS →^0.002^ quitters

VP →^6^ RB VB

NP →^1^ NNS

RB →^0.04^ never

VB →^0.006^ win

NN →^0.0002^ win

win

1

quitters

2

never

...
Agenda

S → .8 NP VP
VP → .6 RB VB
RB → .04 never
VB → .006 win

NNS → .0002 quitters
NP → .1 NNS

NN → .00002 win

...
win 3

S → .8 NP VP
VP → .6 RB VB
RB → .04 never
VB → .006 win

NNS → .0002 quitters
NP → .1 NNS
NN → .00002 win

Agenda

quitters

never

chart
Agenda

S → .8 NP VP

NNS → .0002 quitters

VP → .6 RB VB

NP → .1 NNS

RB → .04 never

VB → .006 win

NN → .00002 win

...
Agenda

S → .8 NP VP

NNS → .0002 quitters

VP → .6 RB VB

NP → .1 NNS

RB → .04 never

VB → .006 win

NN → .00002 win

...
Agenda

NP → .1

VP → .6 RB VB

S → .8 NP VP

NNS → .0002 quitters

RB → .04 never

VB → .006 win

NN → .00002 win

...
Agenda

- quitters 1
- never 2
- win 3

S → 0.8 NP VP
VP → 0.6 RB VB

NNS → 0.002 quitters
NP → 0.1 NNS

RB → 0.04 never
VB → 0.006 win
NN → 0.00002 win

...
Agenda

NP → .1 NNS

NNS → .002 quitters

RB → .04 never

VB → .006 win

NN → .0002 win

S → .8 NP VP

VP → .6 RB VB

quitters

never

win

chart
Agenda

```
S → .8 NP VP
VP → .6 RB VB
NP → .1 NNS

NNS → .0002 quitters

RB → .04 never
VB → .006 win
NN → .00002 win

...```

chart
Agenda

RB $\rightarrow$.04 never

NNS $\rightarrow$.0002 quitters

VB $\rightarrow$.006 win

NN $\rightarrow$.00002 win

chart
Agenda

- RB → .04 never
- NNS → .0002 quitters
- VB → .006 win
- NN → .00002 win
- S → .8 NP/VP
- VP → .6 RB VB
- NP → .1 NNS

chart
Agenda

S → .8 NP VP
VP → .6 RB VB
NP → .1 NNS
RB → .04 never

NNS → .0002 quitters

VB → .006 win
NN → .00002 win

…
Agenda

Chart

1. quitters
   S → 0.8 NP VP
   VP → 0.6 RB VB
   NP → 0.1 NNS
   RB → 0.04 never

2. never

3. win

NNS → 0.0002 quitters

VB → 0.006 win

NN → 0.00002 win

...
Agenda

chart

NNS → .0002 quitters

NN → .00002 win

VB → .006 win

...
Agenda

- VB → .006 win
- NNS → .0002 quitters
- NN → .00002 win

- S → .8 NP VP
- VP → .6 RB VB
- NP → .1 NNS
- RB → .04 never
- win

chart
Agenda

chart

NNS → .0002 quitters

NN → .00002 win

S → .8 NP VP

VP → .6 RB VB

NP → .1 NNS

VB → .006 win

RB → .04 never

2RB2 .04

3VB3 .006
Agenda

1. S → .8 NP VP
2. VP → .6 RB VB
3. NP → .2 NNS
   VB → .006 win
   RB → .04 never

NNS → .0002 quitters
NN → .00002 win
Agenda

NNS → .0002 quitters

NN → .00002 win

chart
...
Agenda chart

NN → .00002 win

S → .8 NP VP

NNS → .0002 quitters

NP → .1 NNS

VB → .006 win

RB → .04 never

NP → .1 NNS

VB → .006 win

RB → .04 never

NN → .00002 win
Agenda

- **quitters**: 0.0002
- **never**: 0.04
- **win**: 0.006

**Chart**

- **S**: 0.8
- **NP**: 0.0002
- **VP**: 0.000144

- **NN**: 0.0002
- **NNS**: 0.0002
- **RB**: 0.04
- **VB**: 0.006

...
Agenda

chart

NN \rightarrow 0.00002 \text{win}

NNS \rightarrow 0.0002 \text{quitters}

NP \rightarrow 0.006 \text{win}

RB \rightarrow 0.04 \text{never}

S \rightarrow 0.8 \text{NP VP}

NP \rightarrow 1 \text{NNS}

VB \rightarrow 0.006 \text{win}

RB \rightarrow 0.04 \text{never}

NNS \rightarrow 0.0002 \text{quitters}

VP \rightarrow 0.00144
Unnecessary Work

• If you only want the best derivation, you don’t want to build items that aren’t in it!
• But you don’t know which items to build until you have the best parse.
• Key idea in the agenda:
  – Intelligently order updates to items’ weights.
  – Roughly analogous to trading depth and breadth in search.
• Note: for exact inside/outside, all of the work is necessary!
Unnecessary Work

best parse:

S

NP

VP
Repropagation

• Suppose \((\text{NP}, 4, 7)\) currently has a weight of 0.3, constructed by \((\text{DT}, 4, 5) \otimes (\text{NP}, 5, 7)\).

• Now suppose we find that a better way to build \((\text{NP}, 4, 7)\): \((\text{DT}, 4, 5) \otimes (\text{NNP}, 5, 6) \otimes (\text{NNP}, 6, 7)\) with value 0.31.

• Maybe now we have a better way to build \((\text{VP}, 3, 7)\)! (Or anything else that used \((\text{NP}, 4, 7)\).

• Have to re-build all of those consequents, and compare again, and recursively repropagate to consequents of any item whose value changes.

• May not be \(O(n^3)\) anymore!
Repropagation

So any consequent of \((NP, j, k)\) might also increase.
Best-First Parsing

  - Cf. Goodman, build the chart and fill in weights at the same time.

- Many parsers in practice: prune, prune, prune.

- Alternative: order items by their weights.
  - “Uniform cost search”
  - Guarantee: the first time **goal** is popped from the agenda, you have the optimal parse.

- Charniak et al., 1998: heuristics to speed this up. “Figures of Merit” (big speed payoff).
Priorities

\[
priority(I) = \text{weight of the best parse that uses } I
\]
\[
= \text{inside}(I) \otimes \text{outside}(I)
\]
\[
\leq \text{inside}(I) \otimes \text{estimate}(\text{outside}(I))
\]

uniform cost search: 1

• Klein and Manning (2003): Blocked more than 90% of edges!
• Generalization of A* search (for hypergraphs instead of graphs).
• Heuristics? Computed by simpler, cheaper dynamic programs!
• Caveat emptor: only for Viterbi (max) semirings!
Dyna (Eisner et al., 2005)

- **Dyna** is a high-level programming language (like Prolog) for weighted deduction.
- Source code looks like Prolog.
- Compiles into C++.
- Core algorithms:
  - Generalized weighted, **prioritized** agenda.
    - Allows the use of heuristics, including A*
    - Handles repropagation if required
  - Efficient “tape” mechanism for **reverse** computation.
    - Very similar to backpropagation.
Dyna Programs

\[
\text{constit}(X,I,J) \quad +\quad \text{word}(W,I,J) \times \text{rewrite}(X,W).
\]
\[
\text{constit}(X,I,J) \quad +\quad \text{constit}(Y,I,\text{Mid}) \times \text{constit}(Z,\text{Mid},J) \times \text{rewrite}(X,Y,Z).
\]
\[
\text{goal} \quad +\quad \text{constit}(\text{“s”},0,N) \text{ whenever length}(N).
\]

\[
\text{constit}(X,I,J) \quad \text{max} = \quad \text{word}(W,I,J) \times \text{rewrite}(X,W).
\]
\[
\text{constit}(X,I,J) \quad \text{max} = \quad \text{constit}(Y,I,\text{Mid}) \times \text{constit}(Z,\text{Mid},J) \times \text{rewrite}(X,Y,Z).
\]
\[
\text{goal} \quad \text{max} = \quad \text{constit}(\text{“s”},0,N) \text{ whenever length}(N).
\]

\[
\text{constit}(X,I,J) \quad \text{max} = \quad \text{word}(W,I,J) \times \text{rewrite}(X,W).
\]
\[
\text{constit}(X,I,J) \quad \text{max} = \quad \text{constit}(Y,I,\text{Mid}) \times \text{inter}(X, Z, \text{Mid}, J).
\]
\[
\text{inter}(X, Z, \text{Mid}, J) \quad \text{max} = \quad \text{constit}(Z,\text{Mid},J) \times \text{rewrite}(X,Y,Z).
\]
\[
\text{goal} \quad \text{max} = \quad \text{constit}(\text{“s”},0,N) \text{ whenever length}(N).
\]
Dyna Programs

\[
\text{constit}(X,I,J) \quad += \quad \text{word}(W,I,J) \times \text{rewrite}(X,W).
\]
\[
\text{constit}(X,I,J) \quad += \quad \text{constit}(Y,I,Mid) \times \text{constit}(Z,Mid,J) \times \text{rewrite}(X,Y,Z).
\]
\[
\text{goal} \quad += \quad \text{constit}("s",0,N) \text{ whenever length}(N).
\]

\[
\text{reverseconstit}(Y,I,Mid) \quad += \quad \text{reverseconstit}(X,I,J) \times \text{constit}(Z,Mid,J) \times \text{rewrite}(X,Y,Z).
\]
\[
\text{reverseconstit}(Z,Mid,J) \quad += \quad \text{constit}(Y,I,Mid) \times \text{reverseconstit}(X,I,J) \times \text{rewrite}(X,Y,Z).
\]
\[
\text{reverseconstit}("s",0,N) \quad += \quad 1.
\]
Dyna Debugger
Parting Shots

- Weighted deduction as a convenient way to design, improve, understand, analyze, unify, transform, and implement otherwise tricky dynamic programming algorithms.