

LCTG—ASSIGNMENT 4

Head Dependency-based Probabilistic DCG

due Nov. 23rd 2007, in class

1 Basic Probabilistic DCG

Here is a naive structure-building Probabilistic DCG (PDCG), of a kind discussed in the lectures. It uses the prolog DCG curly brackets facility to compute probabilities on the side during a parse. Probabilities of rules and words are respectively given by relations `p/2` and `p/3`, which are distributed throughout the program for ease of reading, and which have the effect of lookup tables. Note that these relations or tables constitute a *generative model*, inasfar as probabilities for all rules rewriting nonterminals such as `S` and `NP`, including those for preterminals like `N`, sum to 1.0.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%% Probabilistic Syntactic structure-building DCG in DCG notation    %%
%% Mark Steedman, November 2002, rev. November 2003, Nov. 05      %%
%%                                                                    %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Stuff to overcome Sictus' abbreviation of list terms

portray(Term) :- is_list(Term), write(Term).

is_list([]).
is_list(_|_).

%%% End of stuff

s( P0, [s,[NP, VP]] ) -->                                     %Rule r1
  np(P1, NP),
  vp(P2, VP),
  {p(r1, P), P0 is P*P1*P2}.

p(r1, 1.0).

np(P0, [np,[PN]]) -->                                       %Rule r2
  pn(P1, PN),
  {p(r2,P), P0 is P*P1}.

p(r2, 0.3).
```

```

np(P0, [np,[Det, N, Rel]]) --> %Rule r3
  det(P1, Det),
  n(P2, N),
  optrel(P3, Rel),
  {p(r3,P), P0 is P*P1*P2*P3}.

p(r3, 0.7).

ppwith(P0, [ppwith,[PREP, NP]]) --> %Rule r41
  pwith(P1, PREP),
  np(P2, NP),
  {p(r41,P), P0 is P*P1*P2}.

p(r41, 1.0).

ppon(P0, [ppon,[PREP, NP]]) --> %Rule r42
  pon(P1, PREP),
  np(P2, NP),
  {p(r42,P), P0 is P*P1*P2}.

p(r42, 1.0).

vp(P0, [vp,[TV, NP]]) --> %Rule r5
  tv(P1, TV),
  np(P2, NP),
  {p(r5,P), P0 is P*P1*P2}.

p(r5, 0.4).

vp(P0, [vp,[DTV, NP, PP]]) --> %Rule r61
  dtv(P1, DTV),
  np(P2, NP),
  ppwith(P3, PP),
  {p(r61,P), P0 is P*P1*P2*P3}.

p(r61, 0.4).

vp(P0, [vp,[DTV, NP, PP]]) --> %Rule r62
  dtv(P1, DTV),
  np(P2, NP),
  ppon(P3, PP),
  {p(r62,P), P0 is P*P1*P2*P3}.

p(r62, 0.05).

```

```

vp(P0, [vp,[IV]]) -->                                     %Rule r7
    iv(P1, IV),
    {p(r7,P), P0 is P*P1}.

p(r7, 0.15).

optrel(P0, [rel,[0]]) --> [], {p(r8,P), P0 is P}.         %Rule r8

p(r8, 0.8).

optrel(P0, [rel,[PP]]) --> ppwith(P1, PP), {p(r91, P), P0 is P*P1}. %Rule r91

p(r91, 0.1).

optrel(P0, [rel,[PP]]) --> ppon(P1, PP), {p(r92, P), P0 is P*P1}. %Rule r92

p(r92, 0.099).

optrel(P0, [rel,[WH, VP]]) -->                               %Rule r10
    relpro(P1, WH), vp(P2, VP), {p(r10, P), P0 is P*P1*P2}.

p(r10, 0.001).

%pn(P0, pn[harry]) --> [harry], {p(harry, pn, P1), P0 is P1.
%% etc. -- very tedious, therefore ....

pn(P0, [pn,[Word]]) --> [Word], {pn(Word), p(Word, pn, P1), P0 is P1}.
pn(gilbert).
pn(george).

p(gilbert, pn, 0.5).
p(george, pn, 0.5).

iv(P0, [iv,[Word]]) --> [Word], {iv(Word), p(Word,iv,P1), P0 is P1}.
iv(walks).
iv(talks).

p(walks, iv, 0.7).
p(talks, iv, 0.3).

tv(P0, [tv,[Word]]) --> [Word], {tv(Word), p(Word,tv,P1), P0 is P1}.
tv(sees).
tv(shoots).

```

p(sees, tv, 0.7).
 p(shoots, tv, 0.3).

dtv(P0, [dtv,[Word]]) --> [Word], {dtv(Word), p(Word,dtv,P1), P0 is P1}.
 dtv(sees).
 dtv(shoots).

p(sees, dtv, 0.6).
 p(shoots, dtv, 0.4).

det(P0, [det,[Word]]) --> [Word], {det(Word), p(Word,det,P1), P0 is P1}.
 det(a).
 det(the).

p(a, det, 0.5).
 p(the, det, 0.5).

n(P0, [n,[Word]]) --> [Word], {n(Word), p(Word, n, P1), P0 is P1}.
 n(frog).
 n(man).
 n(telescope).
 n(revolver).

p(frog, n, 0.3).
 p(man, n, 0.3).
 p(telescope, n, 0.2).
 p(revolver, n, 0.1).

pwith(P0, [pwith,[Word]]) --> [Word], {pwith(Word), p(Word, pwith, P1), P0 is P1}.
 pwith(with).

p(with, pwith, 1.0).

pon(P0, [pon,[Word]]) --> [Word], {pon(Word), p(Word, pon, P1), P0 is P1}.
 pon(on).

p(on, pon, 1.0).

relpro(P0, [relpro,[Word]]) --> [Word], {relpro(Word), p(Word, relpro, P1), P0 is P1}.
 relpro(that).

p(that, relpro, 1.0).

This PDCG embodies **unsound** independence assumptions, notably that the probability of a given PP (such as *with a telescope*) is independent of the verb (such as *sees* or *shoots*) or the noun (such as *frog* or *man*) that it is associated with syntactically (and hence semantically). The probability ratios of the two readings of the following two sentences should be different, because frogs with telescopes are less likely than men with telescopes.

```
%| ?- s(P, T, [gilbert, sees, a, frog, with, a, telescope], []).
%P = 2.4695999999999995E-05,
%T = [s, [[np, [[pn, [gilbert]]]],
%      [vp, [[tv, [sees]],
%            [np, [[det, [a]], [n, [frog]],
%                rel, [[ppwith, [[pwith, [with]],
%                                [np, [[det, [a]], [n, [telescope]], [rel, [0]]]]]]]]]]]] ? ;

%P = 0.000169344,
%T = [s, [[np, [[pn, [gilbert]]]],
%      [vp, [[dtv, [sees]],
%            [np, [[det, [a]], [n, [frog]], [rel, [0]]]],
%            [ppwith, [[pwith, [with]], [np, [[det, [a]], [n, [telescope]], [rel, [0]]]]]]]]]] ? ;
%no

%| ?- s(P, T, [gilbert, sees, a, man, with, a, telescope], []).
%P = 2.4695999999999995E-05,
%T = [s, [[np, [[pn, [gilbert]]]],
%      [vp, [[tv, [sees]],
%            [np, [[det, [a]], [n, [man]],
%                rel, [[ppwith, [[pwith, [with]],
%                                [np, [[det, [a]], [n, [telescope]], [rel, [0]]]]]]]]]]]] ? ;

%P = 0.000169344,
%T = [s, [[np, [[pn, [gilbert]]]],
%      [vp, [[dtv, [sees]],
%            [np, [[det, [a]], [n, [man]], [rel, [0]]]],
%            [ppwith, [[pwith, [with]], [np, [[det, [a]], [n, [telescope]], [rel, [0]]]]]]]]]] ? ;
%no
%| ?-
```

The homework asks you to fix this problem by making the PDCG into a **head-dependency based** PDCG along lines developed in the lectures, computing the probability of a constituent on the basis of the probabilities of the head-daughter dependencies that it embodies.

2 Head-dependency Based PDCG

Formally, a dependency is defined as a 4-tuple: $[R, H_r, S, H_s]$, where R is the rule identifier, H_r is the head word of the rule, S is the position of the argument in the rule, and H_s is the head

word of the argument filling that slot. For example, the following is the subject dependency yielded by applying rule r1 in the derivation of *Gilbert walks*:

(1) [r1, walks, 1, gilbert]

To compute the probability of the structure that results:

(2) [s, [[np, [[pn, [gilbert]]], [vp, [[iv, [walks]]]]]]]

—we must first add a rule that computes the probability of the start-symbol being realized by an *S* headed by “walk”.

(3) start(P0, T) --> s(P1,H,T), {p(start, H, Phead), P0 is P1*Phead}.%Rule r0

Then we must rewrite the first rule so that it looks up the probability $p1$ of this rule *given this head*, and that of this specific head-dependency of the argument, given this head and this rule

(4) s(P0, H2, [s, [NP, VP]]) --> %Rule r1
 np(P1, H1, NP),
 vp(P2, H2, VP),
 {p1(r1,H2, Prulegivenhead),
 p2(r1, H2, 1, H1, Pargumentheadgivenheadandrule),
 P0 is Prulegivenhead*Pargumentheadgivenheadandrule*P1*P2}.

In general you will find yourself introducing a Head variable just about everywhere we introduced a Probability variable to make the DCG into a PDCG.

$p1/3$ is a lookup like $p/2$ in the PCFG that computes the probability of a rule $R A \rightarrow \alpha$ applying given a parent A headed by H . This is

$$\frac{freq(R|H,A)}{\sum_{R'} freq(R'|H,A)}$$

$p2/4$ is a lookup like $p/2$ in the PCFG that computes the probability of the i th non-head daughter in a rule $R A \rightarrow \alpha$ being headed by I given a parent A headed by H . This is

$$\frac{freq(I|R,i,H,A)}{\sum_{I'} freq(I'|R,i,H,A)}$$

Notice that the above rule uses partial execution on H2 to pass the head of the VP (that is, the verb) as the head of the S. Lots of rules will need to do this.

This particular rule also looks up $p_{top}(H2,P_{top})$, the probability of the whole sentence being headed by that particular verb: **no other rule needs such a term, since they either inherit that head with probability 1 or get the relevant probability from the dependency tables defined below**. You will need to make a second version of the rule for embedded s. $p_{top}(H2,P_{top})$ can be defined by table lookup, where the probabilities are defined by counting the examples in the corpus, e.g.:

```
(5) ptop(walks, 0.25).
     ptop(talks, 0.25).
     ptop(sees, 0.25)}.
     ptop(shoots, 0.25).
```

For realistic wide coverage parsing we would want to **induce** a grammar of rules like (4) from a corpus of human-labeled trees like (2). But for this homework you provide such a grammar by modifying the toy hand-built PDCG.

Similarly we must provide corresponding probability table entries to assign probabilities to all possible dependencies for this and other rules, which must sum to 1.0 over each rule expanding S for each head (here there is only one, namely r1).

```
(6) p2(r1, walks, 1, gilbert, 0.3).
     p2(r1, walks, 1, george, 0.3).
     p2(r1, walks, 1, man, 0.4).
     p2(r1, walks, 1, frog, 0.0).
     p2(r1, walks, 1, telescope, 0.0).
     p2(r1, walks, 1, revolver, 0.0).

     p2(r1, sees, 1, gilbert, 0.2).
     p2(r1, sees, 1, george, 0.4).
     p2(r1, sees, 1, man, 0.4).
     p2(r1, sees, 1, frog, 0.0).
     p2(r1, sees, 1, telescope, 0.0).
```

etc. etc.

Such probabilities can be obtained from a labelled corpus of trees like (2) by counting all occurrences of r1 with a particular head on the S, and counting the occurrences of each dependent subject with that rule and head. The probabilities above are obtained by dividing the latter counts by the former – hence they too sum to 1.

Again, for a realistically sized corpus we would want to obtain such counts automatically, but for the homework you can obtain them either with a program or manually, from a small corpus on the class page called *hw4.newtreebank.txt*

Note that the head of a lexical word is always the word itself. itself. When you pass a head feature to a terminal word the probability of that word given the head is always 1.0; in effect the terminal word has already been generated as a head further up in the tree. Hence the lexical parts of the program look like this:

```
(7) %pn(1, harry, pn[harry]) --> [harry].
     %% etc. -- very tedious, therefore ....

     pn(1.0, Word, [pn,[Word]]) --> [Word], {pn(Word)}.
     pn(gilbert).
     pn(george).
```

3 Smoothing

If the corpus reflects the true probability of frogs with telescopes, seeing with frogs etc., such a parser will now return **different** probabilities for the two analyses of *Gilbert saw the man with the telescope*, *Gilbert saw the frog with the telescope*, etc., reflecting the readings that humans actually obtain for these sentences.

However, it will now assign a zero probability to the perfectly reasonable sentence *a frog walks*, simply because the corpus happens not to contain any sentence with frog as the subject of walking.

Not only is this unlike human performance. It is also unrealistic, because no corpus is ever big enough to exhibit every dependency pair that you will ever encounter.

We need to “smooth” the probability model, so that it assigns a small probability to unseen events.

4 Unseen Words

A related problem is that not only is no corpus ever going to exhibit every dependency, but also no corpus is ever going to include every word you encounter. So we need a way to guess the category of unseen words, as well as a way to smooth their probability. One crude way to do this is simply to guess that they are nouns, since that is the most frequent single category. Another tactic is to use a “Part of Speech tagger”—a low-level usually Markovian approximation embodying statistical regularities

5 Homework 4: What You Have to Do

Make sure you understand how the basic PDCG linked to on the web-page works using appropriate examples from the file `hw4.newtreebank.txt` of training data, also linked to there.

1. Transform the naive PDCG into a head-dependency-based PDCG, along the lines sketched above. Note: treat the head of the NP in a PPwith or PPOn—that is the noun—as the head of the PP (It is in order to allow you to do this that PPwith and PPOn are distinguished as different syntactic types.)
2. Complete the head-dependency probability model along the lines sketched above, using counts for the trees in `hw4.treebank.txt`. (You can do this by hand or with a simple prolog program.)
3. Think of a way of smoothing the probability model, while still keeping it a generative model, and implement it.
4. Think of a way of dealing with unknown words while still keeping the model generative, and implement that.
5. Use your revised PDCG to compute probabilities for all analyses of all sentences in the file `hw4.testdata`

6. As well as handing in the assignment on the due date in class, send it to the TA via email, with the file named 'hw4_NAMEofSTUDENT.pl'.