Linguistique LI6

May 3, 2013

Search problems in NLP

- Nowadays, most algorithms in computational linguistics assume weighted formalisms: weighted finite state automata/transducers, weighted grammars, weighted tree transducers...
- Weights are provided by some statistical estimation procedure (generally borrowed from Machine learning) as e.g. :
  - Maximum likelihood estimation
  - Perceptron
  - Logistic regression
  - Support Vector Machines
- Instead of computing every possible solution, we prefer to search for a best or k-best solutions in the space of solutions.

Review of some classical search problems

Many NLP problems can be cast as search problems in Directed Acyclic Graphs (DAGs) or Directed Acyclic Hypergraphs (DAHs)

Plan

The part of speech tagging problem

- Consider the sentences:
  1. La belle porte le voile
  2. La belle porte le chapeau
- (1) is ambiguous while (2) is not.
- Part of speech tagging will try to select a preferred reading among possible labellings in order to provide a preferred analysis among e.g.:

  La/D belle/N porte/V le/D voile/V
  La/D belle/A porte /N le/Cl voile/V

- Part of speech tagging, does not try to provide an analysis of the context in which the sentence is uttered in order to choose a preferred alternative (at least in what is usually done), instead it provides a best alternative provided previous experience.

Part of speech tagging as supervised learning problem

- In what follows, I suppose we have a corpus (or list of observations) of pairs of sequence of words and correct categories, where each observation is of the form:

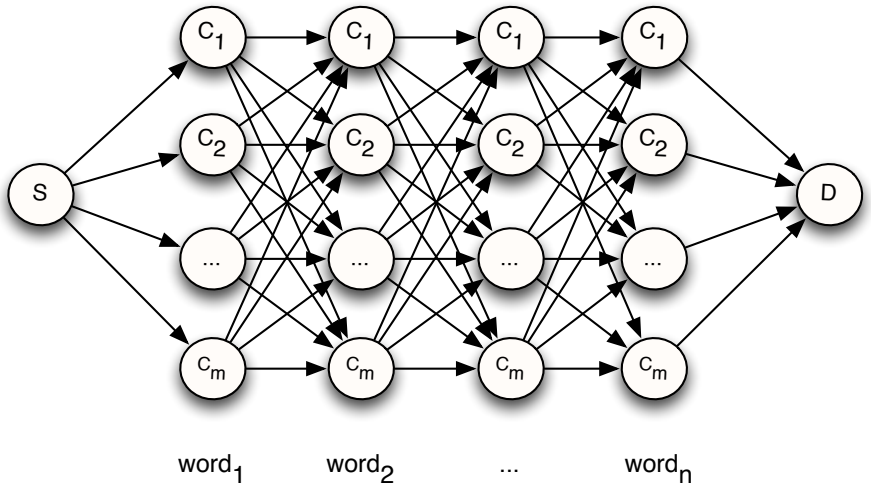
$$f\mathbf{w} = w_1 ::: w_n; \mathbf{c} = c_1 ::: c_n g$$

- The corpus can be used to count and estimate weights for some statistical model (*learning task*).
- Part of speech tagging amounts to predict a labelling (sequence of categories $\hat{\mathbf{c}} = c_1 ::: c_n$) on new observations (*prediction task*)

In what follows I focus on the prediction task: I assume sequences with higher weight to be preferred.

Part of speech tagging as a search problem

- The problem of part of speech tagging amounts to find the best sequence of grammatical categories $\hat{C} = c_1 \dots c_n$ to assign to a given word sequence $W = w_1 \dots w_n$
- The problem can be represented as a search problem (search for the best path) in the following (weighted) DAG:



where each w_i can in principle be mapped to any category $c \in C$ such that we have in principle $|C|^n$ possible category sequences for the string W.

Plan

Topological numbering/ordering

Let $G = hV; E i$, be a DAG:

- A topological numbering is a function $x : V \; 7! \; 1 : : : n$ such that:

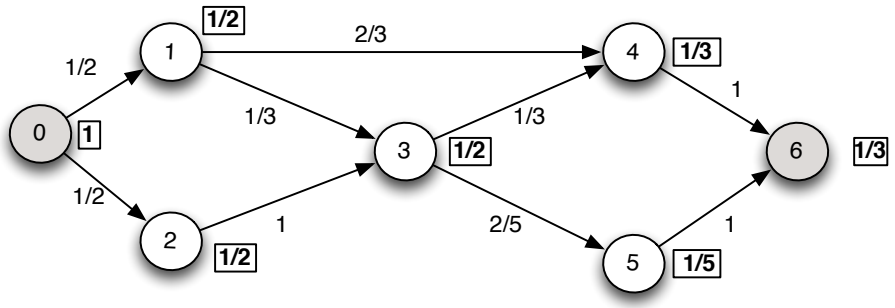
$$x(v) \quad x(u) \quad 8(u; v) \; 2 \; E$$

- A topological ordering is a function $x : V \; 7! \; 1 : : : n$ such that:

$$x(v) > x(u) \quad 8(u; v) \; 2 \; E$$

Example I

Suppose we want to compute the longest path from the source to all other nodes in a DAG. Here we suppose the weights are probabilities, we multiply them along a path and take the maximum of all incoming paths at every node.



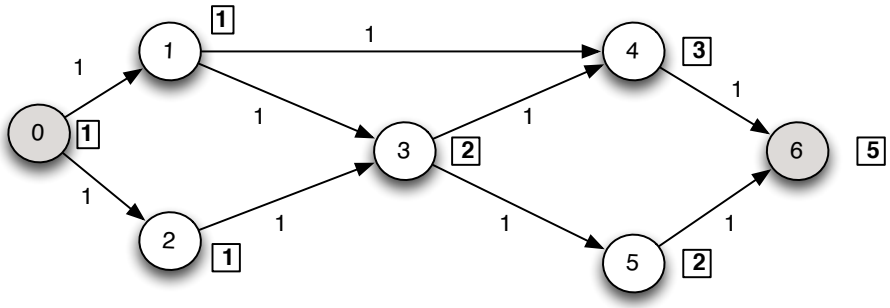
function LONGESTPATH($G = \langle V, E \rangle$)

Compute a topological ordering of G

$d[v] \leftarrow$

Example II

Suppose we want to compute for every node $v \in V$ the number of different paths from the source to v in a DAG. Here we suppose the weights are 1 on every edge, we multiply them along a path and add them at every node.



function NUMPATHS($G = \langle V, E \rangle$)
  Compute a topological ordering of G
  $d[v] \leftarrow 0 \qquad (\forall v \in V)$
  $d[s] \leftarrow 1$
  for all $v \in V$ in topological order **do**
    for all $u \in BS(v)$ **do**
      $d[v] \leftarrow d[v] + (d[u] \times w(u, v))$
    end for
  end for

Generalisation

- The introductory examples algorithms can be generalised by observing that we perform computations in **semi-ring** structures of the form $\langle A; \quad ; \quad ; \bar{0}; \bar{1} \rangle$ where:
  - A is a set
  - $(A; \quad)$ is a commutative monoid with identity element $\bar{0}$
  - $(A; \quad)$ is a monoid with identity element $\bar{1}$
  - Multiplication distributes over addition
  - Multiplication by 0 annihilates A

Comments

- **Example 1** used the probabilistic semiring $\langle [0; 1]; \max; \quad ; 0; 1 \rangle$
- **Example 2** used the counting semiring $\langle \mathbb{N}; +; \quad ; 0; 1 \rangle$

Some terminology/definitions

Definition

A **weighted directed graph** is a pair $G = hV; Ei$ where V is the set of vertices and E the set of Edges with a mapping $E \not\to A$ that assigns each edge a weight from the semiring $(A; \quad ; \quad ; \bar{0}; \bar{1})$

Definition

The **backward star** $BS(v)$ of a vertex x is the set of incoming edges, the **forward star** $FS(v)$ is the set of outgoing edges.

Some terminology/definitions continued

Definition

A **path** $= e_1 ::: e_n$ in a graph G is a sequence of consecutive edges where $e_i; e_{i+1}$ are connected by a vertex. The **weight** of a path $w(\ )$ is:

$$w(\ ) = \bigotimes_{i=1}^{n} w(e_i)$$

Definition

Given a distinguished vertex $s \ 2 \ V$ the source, the weight of the best path from the source to v is defined as:

$$(v) = \begin{cases} \bar{1} & v = s \\ \bigoplus_{\in \ (v)} w(\ ) & v \ 6= s \end{cases}$$

Best weights in a DAG

- Applies to a weighted directed acyclic graph
- Assumes a single source

function VITERBI($G = \langle V, E \rangle$)
    Compute a topological ordering of G
    $d[v] \leftarrow \bar{0} \qquad (\forall v \in V)$
    $d[s] \leftarrow \bar{1}$
    for all $v \in V$ in topological order **do**
        for all $u \in BS(v)$ **do**
            $d[v] \leftarrow d[v] \oplus (d[u] \otimes w(u, v))$
        end for
    end for
end function

Variations

- Here is a variant (forward update)

function VITERBI($G = \langle V, E \rangle$)
  Compute a topological ordering of G
  $d[v] \leftarrow \bar{0} \qquad (\forall v \in V)$
  $d[s] \leftarrow \bar{1}$
  for all $v \in V$ in topological order **do**
    for all $u \in FS(v)$ **do**
      $d[u] \leftarrow d[u] \oplus (d[v] \otimes w(v, u))$
    end for
  end for
end function

- Another variant is **memoized recursion**: starts on the destination state t and recursively solves subproblems (by storing intermediate results to avoid duplicate computations)

Exercises and examples

- In practice the probabilistic semi-ring is not really used since the multiplication of very low probabilities leads to numerical underflows. In place the probabilities are mapped to their logarithm. Which semi-ring would you use in this case ?
  Solution: $h[\quad 1\,; 0]; \max; +; \quad 1\,; 0i$
- Two standard techniques used in Machine Learning, the perceptron and logistic regression yield weights that are real numbers, and their combination along a path is addition, in this case one uses the real semi-ring: $h[\quad 1\,; +1\,]; \max; +; \quad 1\,; 0i$

Plan

Illustration with a classical statistical model

- A bigram HMM models the mapping as follows (markov independance hypothesis+word independance emission hypothesis):

$$P(\mathbf{c}, \mathbf{w}) = \prod_{i=1}^{n} P(c_i|c_{i-1})P(w_i|c_i)$$

- This model can be used to predict the categories given the observed words :

$$P(\mathbf{c}|\mathbf{w}) = \frac{1}{P(\mathbf{w})} \times \prod_{i=1}^{n} P(c_i|c_{i-1})P(w_i|c_i)$$

- Finding the best sequence of categories among all possible such sequences $|C|^n$ amounts to find :

$$\hat{\mathbf{c}} = \underset{c \in |C|^n}{\operatorname{argmax}} \frac{1}{P(\mathbf{w})} \times \prod_{i=1}^{n} P(c_i|c_{i-1})P(w_i|c_i)$$

- Since $P(\mathbf{w})$ is constant accross all hypotheses, it can be dropped:

$$\hat{\mathbf{c}} = \underset{c \in |C|^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(c_i|c_{i-1})P(w_i|c_i)$$

Estimating the parameters of an HMM

- We suppose to have a corpus of n observations of annotated text, where each sequence of word ($\mathbf{w}_i$) is mapped to its sequence of categories ($\mathbf{c}_i$)
- The vector of parameters $\tilde{}$ (probabilities of factors $P(c_i|c_{i-1})$ and $P(w_i|c_i)$) of an HMM are estimated in order to maximise the likelihood of the observed data:

$$\hat{\theta} = \underset{\tilde{}}{\operatorname{argmax}} \prod_{i=1}^{n} P(\mathbf{c}_i, \mathbf{w}_i; \vec{\theta})$$

- The solution to this optimisation problem is analytic and amounts to estimate the parameters using relative frequencies:

$$P(c_i|c_{i-1}) = \frac{\text{Count}(c_i, c_{i-1})}{\text{Count}(c_{i-1})} \quad P(w_i|c_i) = \frac{\text{Count}(w_i, c_i)}{\text{Count}(c_i)}$$

Comment

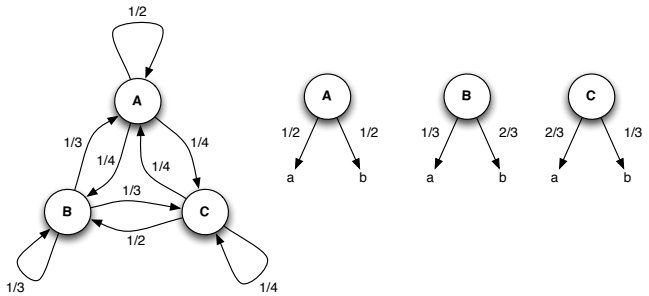
Since bigrams and words are zipf distributed, it is critical in practice to refine parameter estimation using some smoothing method.

Representation of an HMM

Definition

An HMM is a tuple $\langle C, T, \Pi, F, W, P \rangle$ where :

- C : is a state set $c_1 \dots c_n$
- T : is a probability transition matrix where t_{ij} is the probability to move from c_i to c_j, $P(c_j|c_i)$, with $\forall c_i \sum_j t_{ij} = 1$
- Π : $\pi_1, \dots \pi_n$ is the initial probability distribution with $\sum_i \pi_i = 1$
- F : is the set of final states ($F \subseteq C$)
- W : $w_1 \dots w_n$ is a set of symbols (observations or words)
- P : is the symbol emission probability function where $c_i(w_j)$ is the probability $P(w_i|c_i)$. We have that $\sum_j c_i(w_j) = 1$



Plugging it together

To summarize, we can solve the following equation:

$$\hat{c} = \underset{c \in |C|^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(c_i | c_{i-1}) P(w_i | c_i)$$

with the search method (Viterbi algorithm) described above. The weights of the edges are the products $P(c_i | c_{i-1}) P(w_i | c_i)$.

Instanciation of the Viterbi Algorithm for HMM

The Viterbi algorithm for computing the best path:

function VITERBI($G = \langle V, E \rangle$)
  Compute a topological ordering of G
  $d[v] \leftarrow \bar{0} \qquad (\forall v \in V)$
  $d[s] \leftarrow \bar{1}$
  for all $v \in V$ in topological order **do**
    for all $u \in FS(v)$ **do**
      $d[u] \leftarrow d[u] \oplus (d[v] \otimes \text{weight}(d[v], d[u]))$
    end for
  end for
end function

where the semi-ring is the probabilistic semi-ring. We further assume that weight($d[v]; d[u]$) is a function computing $P(c_i|c_{i-1}) \quad P(w_i|c_i)$

Exercise (Forward algorithm)

- Provided an HMM model of the form:

$$P(\mathbf{c}; \mathbf{w}) = \prod_{i=1}^{n} P(c_i | c_{i-1}) P(w_i | c_i)$$

  we want to compute $P(\mathbf{w}) = \sum_{c \in |C|^n} \prod_{i=1}^{n} P(c_i | c_{i-1}) P(w_i | c_i)$ for some $\mathbf{w} = w_1 ::: w_n$
- How to compute $P(\mathbf{w})$?

Conclusion

Comment

For other machine learning methods such as Multinomial Logistic regression, Perceptron... also try to maximise some score. What changes is the weight(;) function and the semi-ring actually used. Although some further algorithmic complications may rise at statistical estimation (estimating the weights).

When applied in prediction, models of the above form usually get 96%-98% correct predictions per word when compared with reference data.

Plan

Parsing as a search problem

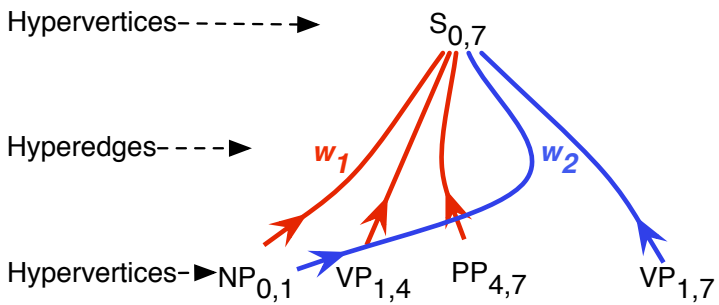
- In practice an exhaustive parser computing all parses for a treebank grammar returns commonly a shared forest encoding several thousands/millions of trees, which is:
  - Useless as it stands (practical point of view)
  - Ignoring the strong competence hypothesis (Bresnan 82) : human language competence also allows to rank parses.
- Since the grammar is weighted, parsing can also be viewed as a search problem for the best (or k-best solutions) in the space of all solutions (that is the shared forest).

Plan

1. Part of speech tagging
   - Searching the longest path in a DAG
   - HMM modelling
2. Parsing as a search problem
   - Search in an Hypergraph
   - Pcfg
   - Lexicalised parsing

The shared forest as an hypergraph

- The shared forest can be seen as an hypergraph whose hypervertices are equivalence classes (equivalent items) and whose hyperedges are weighted $\text{CFG}_\cap$ rules.
- Two items $hX; i; j\, i$,$hX'; i'; j'\, i$ are equivalent iff $X = X'; i = i'$ and $j = j'$.
- Illustration:



A fragment of a shared forest as an hypergraph

- In what follows, we transfer our graph based search algorithm to hypergraphs

Hypergraph (definitions)

Definition

A **weighted directed hypergraph** is a pair $H = \langle V; E \rangle$ with a set R where V is the set of (hyper) vertices, E the set of hyperedges and

Definitions (continued)

Definition

The **Backward Star** $BS(v)$ of a vertex v is the set of incoming hyperedges such that $fe\ 2\ Ejv = h(e)g$

Definition

The **Forward Star** $FS(v)$ of a vertex v is the set of outgoing hyperedges such that $fe\ 2\ Ejv\ 2\ T(e)g$

Definition

The graph projection of an hypergraph is a directed graph $G = hV; E'i$ where

$$E' = f(u; v)j9e\ 2\ BS(v) \quad s:t: \quad u\ 2\ T(e)g$$

An hypergraph is acyclic iff its projection is acyclic and a topological ordering of H is a topological ordering of V, that is a topological ordering of G.

Definitions (Derivation)

We also need to define a counterpart of a graph path:

Definition

A derivation D of a vertex v in a hypergraph H, its size jDj and its weight $w(D)$ are recursively defined as follows:

- if e 2 $BS(v)$ with $jej = 0$, then $D = he;\ i$ is a derivation of v, its size $jDj = 1$ and its weight $w(D) = f_e()$
- If e 2 $BS(v)$ where $e > 0$ and D_i is a derivation of $T_i(e)$ for 1 i jej then $D = he; D_1 ::: D_{|e|} i$ is a derivation of v, its size $jDj = 1 + \sum_{i=1}^{|e|} jD_i j$ and its weight $w(D) = f_e(w(D_1); : : : ; w(D_{|e|}))$

Ordering derivations and best derivation

- The ordering on weights R induces an ordering on derivations : $D \quad D'$ iff $w(D) \quad w(D')$

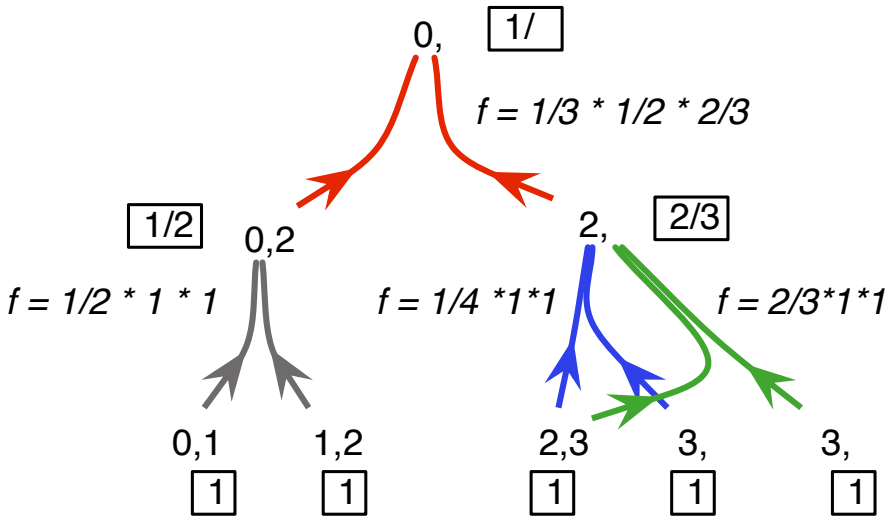
Definition

Let $D(v)$ be the set of derivations of v. The best weight (v) of a vertex v is the weight of the best derivation of v:

$$(v) = \begin{cases} \bar{1} & v \text{ is a source vertex} \\ \quad_{D \in \mathcal{D}(v)} w(D) & \text{otherwise} \end{cases}$$

Illustration

Suppose we want to compute the longest path from the sources vertices to all other nodes in a DAH. Here we suppose the weights are probabilities, we multiply them along the derivations by the probabilities of the hyperedges and take the maximum of all incoming derivations at every node.



The Viterbi algorithm for DAH

The Viterbi algorithm for computing the best derivation:

function VITERBI($H = \langle V, E \rangle$)
  Compute a topological ordering of H
  $d[v] \leftarrow \bar{0} \qquad (\forall v \in V)$
  $d[s] \leftarrow \bar{1}$ ▷ Init weights for all the sources
  for all $v \in V$ in topological order **do**
    for all hyperedge $e = \langle v, u_1, \ldots u_{|e|}, f_e \rangle \in BS(v)$ **do**
      $d[v] \leftarrow d[v] \oplus f_e(d(u_1) \ldots d(u_{|e|}))$
    end for
  end for
end function

Comments

In principle the algorithm can accomodate any weighting scheme: it requires to specify f_e and the semi-ring according to the interpretation of the weights required by the statistical method used. In practice, f_e is computationaly expensive (prohibitive) for discriminative learning methods : multinomial logistic regression, perceptron...

Plugging Viterbi in CKY

Since Viterbi requires the parse items to be processed in topological order wrt H, CKY is a straightforward choice for processing weighted grammars since it already processes items in topological order, hence Viterbi can be directly integrated in the recognition process. The weighted inference rules for CKY are the following:

- Goal :

$$< S; 0; n >$$

- Scan :

$$\frac{}{hA; i \quad 1; 1i : \overline{1}} A \; ! \; w_i \; 2 \; P$$

- Complete :

$$\frac{hB; i; `_1 i : w_1 \quad hC; i + `_1; `_2 i : w_2}{hA; i; `_1 + `_2 i : w_1 \quad w_2 \quad w(A \; ! \; BC)} \quad A \; ! \; BC \; 2 \; P$$

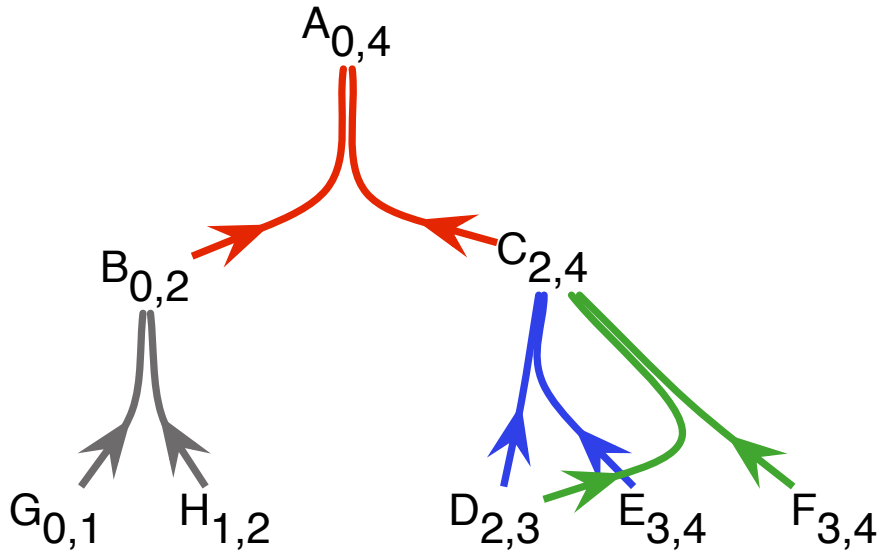
A forward variant of the Viterbi algorithm

Like for the graph case, a forward variant of the Viterbi algorithm can be designed for hypergraphs too. However the situation is not as straightforward as for the graph case, the following adaptation is incorrect:

function VITERBI($H = \langle V, E \rangle$)
  Compute a topological ordering of H
  $d[v] \leftarrow \bar{0} \qquad (\forall v \in V)$
  $d[s] \leftarrow \bar{1}$ ▷ Init weights for all the sources
  for all $v \in V$ in topological order **do**
    for all hyperedge $e \in FS(v)$ **do**
      $d[h(e)] \leftarrow d[h(e)] \oplus f_e(d(u_1) \dots d(u_{|e|}))$
    end for
  end for
end function

Illustration

- There is indeed a risk of firing an hyperedge before all its tail vertices have been assigned their best weight.



Suppose $v = B_{0,2}$ but the green hyperedge has not yet been processed. Firing the red hyperedge immediately does not guarantee its weight to be optimal.

- It requires to wait until the tail has been fully processed.

A solution

One can use a counter to delay the firing

function VITERBI($H = \langle V, E \rangle$)
  Compute a topological ordering of H
  $d[v] \leftarrow \bar{0} \qquad (\forall v \in V)$
  $d[s] \leftarrow \bar{1}$ ▷ Init weights for all the sources
  $r[e] \leftarrow |e|$
  for all $v \in V$ in topological order **do**
    for all hyperedge $e \in FS(v)$ **do**
      $r[e] \leftarrow r[e] - 1$
      if $r[e] == 0$ **then**
        $d[h(e)] \leftarrow d[h(e)] \oplus f_e(d(u_1) \dots d(u_{|e|}))$
      end if
    end for
  end for
end function

Plan

Illustration (PCFG)

- A Probabilistic Context free Grammar $G = hN; T; P; S; \Pi i$ is a CFG $hN; T; P; S i$ with a mapping $\Pi : P \;\; ! \;\; [0; 1]$
- A PCFG defines a probability distribution over a set of sentences $S = L(G)$ such that :

$$P(S) = 1$$

- The probability $P(s)$ of a sentence $s \; 2 \; S$ is the sum of the probabilities of its parse trees:

$$P(s) = \sum_{t \in \text{GEN}(s)} P(s; t)$$

where $\text{GEN}(s)$ denotes the set of parse trees of this sentence.

- The Probability of a parse tree $P(t)$ is the probability of the rules used to derive s

$$P(s; t) = P(t) = \prod_{P(\;\; \rightarrow \;\;) \in t} P(\quad ! \quad)$$

Inference

There are two main inference tasks for PCFGs:

- **Parameter estimation:** Given a collection of trees, called a *treebank* one can estimate the parameters of the grammar by counting, the relative frequency ratio is also the maximum likelihood estimate:

$$P(\quad ! \quad) = P(\ j\ ) = \frac{C(\quad ! \quad)}{\sum_{:\ \rightarrow \quad} C(\ \rightarrow\ )}$$

- **Prediction (parsing)**, which amounts to search for the parse $\hat{t}$ with maximal probability:

$$\hat{t} = \underset{t \in \textsc{Gen}(s)}{\operatorname{argmax}} \prod_{P(\ \rightarrow\ ) \in t} P(\quad ! \quad)$$

which we can solve with the Viterbi search for Directed acyclic hypergraphs (aka weighted CKY) using a probabilistic semi-ring.

Comments on search methods

- The Viterbi search method is not the only search method available...
- Other search methods can be used, for instance Dijkstra's shortest path algorithm (Knuth 77) or its extension A* search can be used as well. (Q: with which semi-rings ?)
- However using another search method requires:
  - The semi-ring to satisfy the properties of the search method: e.g. Dijkstra requires a superior semi-ring.
  - Since it is desirable to integrate the weighting computation into the recognition process, the ordering of the parsing method should be compatible with the ordering of the search method. (e.g. Earley cannot work with a search method requiring a topological ordering of the items)