A LEFT-TO-RIGHT TAGGER FOR WORD GRAPHS

Christer Samuelsson
Bell Laboratories, Lucent Technologies
600 Mountain Ave, Room 2D-339
Murray Hill, NJ 07974, USA
Email: christer@research.bell-labs.com

Abstract

An algorithm is presented for tagging input word graphs and producing output tag graphs that are to be subjected to further syntactic processing. It is based on an extension of the basic HMM equations for tagging an input word-string that allows it to handle word-graph input, where each arc has been assigned a probability. The scenario is that of some word-graph source, e.g., an acoustic speech recognizer, producing the arcs of a word graph, and the tagger will in turn produce output arcs, labelled with tags and assigned probabilities. The processing as done entirely left-to-right, and the output tag graph is constructed using a minimum of lookahead, facilitating real-time processing.

1 Background

Ever since [Church 1988], HMM-based part-of-speech (PoS) tagging has been very popular. The task of the particular PoS tagger described in the current article is to act as a module for lexical lookup that is capable of performing some local pruning, and the output is passed on to subsequent syntactic processing. For this reason, it is desirable to retain some ambiguity in the output, namely that which cannot be accurately removed at this stage of processing. This is very similar in spirit to the lexical-lookup and lexical-pruning stages of the SRI Core Language Engine, see [Rayner & Carter 1996], and the lexical-lookup phase and subsequent morphological disambiguation of the English Constraint-Grammar Parser of Helsinki, see [Karlsson *et al* (eds.) 1995].

The tagger is a component of a system employing a pipeline architecture with the design philosophy that at some processing level, a pruned search space, e.g., a word graph, is the input of one module, which produces an output pruned search space, e.g., a tag graph or preterminal chart, at the next processing level, which in turn serves as the input of a subsequent module, thus closing the circle.

It is also highly desirable, both for the goal of mimicking human behavior and for the more practical reason of achieving real-time processing that the tagger process the input left-to-right. This means that the tagger will continuously construct the arcs of the output tag graph as soon as enough input has been seen to do so. This in turn requires that the amount of lookahead employed by the tagger be kept to a minimum.

The work described in the current paper is based on previous theoretical work by the author on extending the basic HMM equations for word-string tagging to handle word-graph input, as described in detail in [Samuelsson 1997]. Section 2 below provides the final equations arrived at, together with some explanations; the reader is referred to the above publication for the underlying theory and their formal derivations. Section 3 constitutes the new body of work, describing the new left-to-right tagging algorithm and motivating the simplification made to achieve it. The algorithm has been implemented and works satisfactorily. The implementation itself is a straight-forward realization of the described algorithm, and of little interest. No serious empirical evaluation has yet been attempted, partly due to the lack of appropriate test material.

2 The Equations

The following is an extension of the basic HMM equations given in for example [Rabiner 1989], pp. 272–274, or [Krenn & Samuelsson 1997], pp. 42–46, to the case where the input is a word graph, rather than a word string.

By first producing a tag graph from the word graph, we reduce the problem of tagging an input word graph to that of applying an N-gram tag model to an input tag graph. This is done by defining

$$b_{rtj} \quad = \quad P(L_{rt} = l_j \mid G_r) \quad = \quad \sum_k P(L_{rt} = l_j \mid W_{rt} = k) \cdot P(W_{rt} = k \mid G_r) \tag{1}$$

where $P(W_{rt} = k \mid G_r)$ is the probability of word candidate k from node r to node t in the input word graph. The conditioning on G_r indicates that the probability distribution is the one of node r, i.e., that over the outgoing arcs of node r. $P(L_{rt} = l_j \mid W_{rt} = k)$ is the lexical tag probability, a model parameter estimated from training data.

We can thus recast the problem as that of applying an N-gram model to reestimate the arc probabilities of a labelled directed acyclic graph, where a probability distribution over the outgoing arcs has been associated with each node. We require that there be unique globally minimal and maximal elements, the start and end nodes of the graph. This is often referred to as a lattice, although strictly speaking it is not, as we only require a unique global minimal element, not one for any given node pair.

We introduce a(ny) complete ordering $<$ of the nodes of the graph which does not violate the topological ordering defined by the arcs. We refer to each node through its rank in this ordering, the node number, and we let 0 be the start node and T be the end node.

We recursively calculate the α, β, γ, δ and ϵ variables for each node (pair) $\{(r,t) : 0 \leq r < t = 1,\ldots,T\}$ and each possible state $i, j = 1,\ldots,n$. The states encode the N-gram model, and each state corresponds to a label sequence. In the bigram case, each state is a label, in the trigram case, a label pair, etc.

$$\alpha(t,j) \quad = \quad P(S_t = j \mid \mathbf{G}_{<t}) \quad = \quad \sum_r \epsilon(r,t,j) \tag{2}$$

This is the probability of being in state j at node t, given that you start from the start node.

$$\beta(r,i) \quad = \quad P(\langle end \rangle \mid S_r = i; \mathbf{G}_{\geq r}) \quad \approx \quad \sum_{t,j} q(r,t,i,j) \cdot \beta(t,j) \tag{3}$$

This is the probability of reaching the end node, given that you start in node r and that you are in state i.

$$\gamma(r,t,j) \quad = \quad P(S_t = j; \langle end \rangle \mid S_r = \bullet; \mathbf{G}) \quad \approx \quad \frac{\epsilon(r,t,j) \cdot \beta(t,j)}{\sum_{t,j} \epsilon(r,t,j) \cdot \beta(t,j)} \tag{4}$$

This is the joint probability of transiting from node r to node t, being in state j at node t, and reaching the end node, given that you start in node r.

$$\delta(t,j) \quad = \quad \max_{\mathbf{S}_{<t}} P(\mathbf{S}_{<t}; S_t = j \mid \mathbf{G}_{<t}) \quad \approx \quad \max_{r,i} \left[\delta(r,i) \cdot q(r,t,i,j)\right] \tag{5}$$

This is the joint probability of being in state j at node t and reaching it through the most likely node-state sequence, given that you start from the start node.

$$\epsilon(r,t,j) \quad = \quad P(S_r = \bullet; S_t = j \mid \mathbf{G}_{<t}) \quad \approx \quad \sum_i \alpha(r,i) \cdot q(r,t,i,j) \tag{6}$$

This is the joint probability of transiting from node r to node t and being in state j at node t, given that you start in the start node.

$$q(r,t,i,j) \quad = \quad P(S_t = j \mid S_r = i; G_r) \quad = \quad P(L_{rt} = l_j \mid S_r = i; G_r) \quad \approx \quad \frac{p_{ij} \cdot b_{rtj}}{p_j}$$

This is the joint probability of transiting from node r to node t and being in state j at node t, given that you start in node r and that you are in state i. Note that for N-grams with $N > 2$, l_i may equal l_j for distinct i and j.

$$b_{rtj} \quad = \quad P(L_{rt} = l_j \mid G_r)$$

This is the input arc probability of Eq. (1) as explained above.

$$p_{ij} \quad = \quad P(L_{x\bullet} = l_j \mid S_x = i)$$

This is the N-gram transition probability, a model parameter estimated from training data.

$$p_j \quad = \quad P(L_{\bullet\bullet} = l_j)$$

This is the label unigram probability, a model parameter estimated from training data.

We also need the initializations

$$\alpha(\bullet, j) \quad = \quad \delta(0, j) \quad = \quad \begin{cases} 1 & \text{if } j \text{ is the initial state} \\ 0 & \text{otherwise} \end{cases}$$

$$\beta(T, i) \quad = \quad \begin{cases} 1 & \text{if } i \text{ is a final state} \\ 0 & \text{otherwise} \end{cases}$$

Here $\mathbf{G}$ is the input graph, $\mathbf{G}_{<r}$ is the portion of the input graph from nodes 0 to $r-1$, $\mathbf{G}_{\geq t}$ is the portion of the input graph from nodes t to T, and G_r is the portion of the graph associated with node r, referring to the input probability distribution over the outgoing arcs of node r.

$S_t = j$ is the event of being in state j at node t, which will uniquely determine the label of the previous $N-1$ arcs leading to node t, and $S_r = \bullet$ is the event of being in some state at node r, i.e., of visiting node r. If in some expression $P(\ldots S_r = \ldots S_t = \ldots)$ or $P(\ldots S_t = \ldots \mid \ldots S_r = \ldots)$, where $r < t$, there is no node $s : r < s < t$, we will interpret this as implying an immediate transition from node r to node t.[1]

The δ variables are used to find the most probable node and state sequence $\mathbf{i}$, the latter which uniquely determines the most probable node and label sequence:

$$\mathbf{i} \quad = \quad \langle \tau_1, \iota_1 \rangle, \ldots, \langle \tau_m, \iota_m \rangle \quad = \quad \operatorname*{argmax}_{\mathbf{S}} P(\mathbf{S} \mid \mathbf{G})$$

This can be obtained by tracing the nodes and states for which the maximum was obtained in each application of the recurrence equation for the δ variables:

$$\langle \tau_m, \iota_m \rangle \quad = \quad \langle T, \iota_m \rangle \quad = \quad \operatorname*{argmax}_{j} \delta(T, j)$$

$$\langle \tau_{p-1}, \iota_{p-1} \rangle \quad = \quad \operatorname*{argmax}_{r,i} \left[\delta(r, i) \cdot q(r, \tau_p, i, \iota_p)\right]$$

In practice, the argmax will be constructed during the maximization procedure, and retained as a back pointer for each δ variable. In the interest of clarity and brevity, we have omitted these back pointers from Eq. (5).

The γ variables are used to estimate the probability of a particular label being assigned to a particular node pair:

$$P(L_{rt} = l_j \mid \mathbf{G}) \quad = \quad \sum_{i:l_i = l_j} P(S_r = \bullet : S_t = i \mid \mathbf{G}) \quad = \quad \sum_{i:l_i = l_j} \gamma(r, t, i) \tag{7}$$

This is the sum over all states whose last label is l_j. The γ variables allow us to retain ambiguity in the output, while pruning low-probability labels. A similar strategy is adopted for word-string input in [de Marcken 1990].

It is easy to verify, that using the above equations, the optimal label sequence and the reestimated arc probabilities can be constructed in $O(L^N T^3)$ time, where L is the cardinality of the label set, N is the "N" in "N-gram", and T is the number of input graph nodes. This is because in the worst case we have to scan to the beginning or end of the graph in the recurrence step, while constructing accumulators for each node pair. However, if there is a maximum arc length, which is a common situation, we recover the $O(L^N T)$ time complexity of the input-string case.

[1] If there is no arc from r to t in $\mathbf{G}$, the probability in question is simply zero.

3 The Algorithm

The problem with the traditional approach, i.e., using the δ variables to find the most likely tag sequence, is firstly that it is geared towards a single best tag sequence, and secondly requires scanning to the end of the current input before producing its output. A third problem is that of assigning probabilities to the arcs of the output tag graph. The first two problems can certainly be amended for example by maintaining multiple best candidates at each node and state, and start tracing back from the best candidates already before the end of the input is reached. The third problem can perhaps be solved by determining the output probabilities from the value of the relevant δ accumulators.

We will however choose another approach based on the γ variables, from which the output arc probabilities can be directly estimated. These variables too, as the equations stand, require that the input be processed in its entirety before any output arcs can be produced. The key observation here is that the β variables can be approximated with a reasonable degree of accuracy without scanning the entire input. We realize that without taking the N-gram model into account, the β variables will all be 1, if the input graph is consistent. A first, crude approximation would simply set them to 1, and only use the ϵ variables to calculate the γ variables. An improvement on this, that costs little, is to set them to 1 in the β-variable recurrence equation:

$$\beta(r,i) \quad = \quad P(\langle end \rangle \mid S_r = i; \mathbf{G}_{\geq r}) \quad \approx \quad \sum_{t,j} q(r,t,i,j) \cdot \beta(t,j) \quad \approx \quad \sum_{t,j} q(r,t,i,j) \cdot 1 \tag{8}$$

This is exactly what the tagging algorithm does.

When devising the tagging algorithm, we capitalize on the requirement that lexicographical ordering respect the topological one, and on the complete ordering introduced in Section 2.

The status of any node is one of EMPTY, PARTIAL, FULL, PENDING or FINISHED:

- EMPTY means that no inbound or outbound arcs have yet been seen.
- PARTIAL means that at least one inbound or outbound arc has been seen, but not all outbound arcs.
- FULL means that all outbound arcs have been seen, but that the β and ϵ accumulators have not yet been constructed.
- PENDING means that the β and ϵ accumulators have been constructed, using Eq. (8) and Eq. (6) respectively, but not the γ accumulators. Also, the contributions from the outgoing arcs to the α accumulators of their end nodes have been recorded according to Eq. (2).
- FINISHED means that the γ accumulators have been constructed using Eq. (4), and the reestimated arc probabilities of Eq. (7) have been passed on to the next processing step.

By the Last Node of some node r, we mean the greatest node t for which there exists an arc from node r to node t.

We define two distinguished nodes Front and Back with the following properties:

- All nodes prior to Front are either FINISHED or PENDING (or EMPTY but discarded).
- All nodes prior to Back are FINISHED (or EMPTY but discarded).
- Back is prior to Front.

If all nodes prior to some FULL node t are PENDING or FINISHED, then the α accumulators for node t are completed, and we can construct the ϵ accumulators for node t. This is also a convenient time to construct the β accumulators for node t, and to update the α accumulators for the end nodes of the arcs from node t.

1. Since all nodes prior to Front are PENDING or FINISHED, we can advance Front until a PARTIAL node is encountered, in the process converting the FULL nodes into PENDING nodes by constructing the ϵ and β accumulators, and updating subsequent α accumulators. Any EMPTY nodes encountered in the process can be discarded, since they are unreachable from the start node, due to the relationship between the node numbering and the topological ordering.

2. Since all nodes prior to Front have completed β accumulators and all PENDING nodes have completed ϵ accumulators, any PENDING node whose Last Node is prior to Front can be converted into a FINISHED node by constructing its γ accumulators and outputting the reestimated arc probabilities. We can thus scan to Front for such nodes, taking this action. Since all nodes prior to Back are FINISHED, we can start the scan from Back.

3. We now advance Back over all FINISHED nodes, again discarding EMPTY nodes, and await the arrival of more incoming arcs, converting EMPTY nodes to PARTIAL nodes and PARTIAL nodes to FULL nodes. Whenever Front becomes FULL, we goto 1.

We also need to know the start node to initialize Front, Back and the α variables. And we need to know the end node to allow breaking the recursion and proceeding to the next input graph, when the current one has been completely processed.

The algorithm allows the input arcs to arrive in any order they wish, and passes them on, appropriately processed, as quickly as the mathematical model allows. The mathematical model in question is the basic set of HMM equations found everywhere, extended to word-graph input, as described in [Samuelsson 1997], with the extra approximation in the β-variable recursion Eq. (8). The key issue is to introduce a complete node ordering that respects the topological ordering of the arcs of the input word graph. By doing this, the nodes can be processed in number order as soon as enough of the remaining input has been seen. The Back pointer indicates how far complete processing has progressed. The Front and Last Node pointers indicate how far sufficient information has been seen to complete processing.

4 Summary

The algorithm presented in the current article is based on an extension of the basic HMM equations for tagging an input word-string that allows it to handle word-graph input, where each arc has been assigned a probability. The produced output is a probabilistic tag graph, or chart, that is further processed in subsequent syntactic parsing steps.

The processing is done entirely left-to-right, and the output tag graph is constructed using a minimum of lookahead, facilitating real-time transduction of the input arcs of a word graph, generated by some word-graph source, e.g., an acoustic speech recognizer, into output arcs labelled with tags and assigned probabilities.

There are points of similarity with the presented approach and that of using weighted transductions to transduce a probabilistic word graph into a probabilistic (lexical) tag graph, and in a subsequent transduction step apply a probabilistic N-gram model, see [Pereira *et al* 1994]. There, the resulting tag graph is effectively the (probabilistic) intersection of the lexical tag graph with the tag N-gram model, where the nodes of the output graph encode the set of possible predecessor tags, thus changing the topology of the lexical tag graph, and generally drastically increasing its size. Their implementation allows realtime processing mimicking the behavior of the method presented in the current article. However, the method of the current article retains the topology of the lexical tag graph, and merely readjusts the arc probabilities, thus limiting the size of the output graph.

Future and ongoing work includes extending the scheme in both directions, i.e., to handle preceding phoneme-to-word transduction and subsequent phrasal parsing respectively.

References

[Church 1988] Kenneth W. Church. 1988. "A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text". In *Procs. 2nd Conference on Applied Natural Language Processing*, pp. 136–143, ACL, 1988.

[Karlsson *et al* (eds.) 1995] F. Karlsson, A. Voutilainen, J. Heikkilä and A. Anttila (eds.). 1995. *Constraint Grammar. A Language-Independent System for Parsing Unrestricted Text.* Berlin and New York: Mouton de Gruyter, 1995.

[Krenn & Samuelsson 1997] Brigitte Krenn and Christer Samuelsson. 1994–1997. *The Linguist's Guide to Statistics.* Version of May 21, 1997. `http://coli.uni-sb.de/~christer`.

[de Marcken 1990] Carl G. de Marcken. 1990. "Parsing the LOB Corpus". In *Procs. 28th Annual Meeting of the Association for Computational Linguistics*, pp. 243–251, ACL, 1990.

[Pereira *et al* 1994] Fernando Pereira, Michael Riley and Richard Sproat. 1994. "Weighted Rational Transductions and their Application to Human Language Processing". In *Procs. of the Human Language Technology Workshop*, pp. 249–254, Morgan Kaufmann, 1994.

[Rabiner 1989] Lawrence R. Rabiner. 1989. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition". In *Readings in Speech Recognition*, pp. 267–296. Alex Waibel and Kai-Fu Lee (eds). Morgan Kaufmann, 1990.

[Rayner & Carter 1996] Manny Rayner and David Carter. 1996. "Fast Parsing Using Pruning and Grammar Specialization". In *Procs. 34th Annual Meeting of the Association for Computational Linguistics*, pp. 223–230, ACL, 1996.

[Samuelsson 1997] Christer Samuelsson. 1997. "Extending N-gram Tagging to Word Graphs". In *Procs. 2nd International Conference on Recent Advances in Natural Language Processing*, Tzigov Chark, Bulgaria.