Viterbi training in PRISM

Taisuke Sato *sato@mi.cs.titech.ac.jp*
Keiichi Kubota *kubota@mi.cs.titech.ac.jp*
Tokyo Institute of Technology, Ookayama 2-12-1, Meguro, Tokyo, Japan

Abstract

VT (Viterbi training), or hard EM, is an efficient way of parameter learning for probabilistic models with hidden variables. Given an observation y, it searches for a state of hidden variables x that maximizes $p(x, y \mid \theta)$ by coordinate ascent on parameters θ. In this paper we introduce VT to PRISM, a logic-based probabilistic modeling system for generative models. VT improves PRISM's probabilistic modeling in two ways. First although generative models are said to be inappropriate for discrimination tasks in general, when parameters are learned by VT, models often show good discrimination performance. We conducted two parsing experiments with probabilistic grammars while learning parameters by a variety of inference methods, i.e. VT,EM,MAP and VB. The result is that VT achieves the best parsing accuracy among them in both experiments. Second since VT always deals with a single probability of a single explanation, Viterbi explanation, the exclusiveness condition imposed on PRISM programs is no more required when we learn parameters by VT. PRISM with VT thus allows us to write inclusive clause bodies, learn parameters and compute Viterbi explanations.

1. Introduction

VT (Viterbi training) has been used for long time as an efficient parameter learning method in various research problems such as machine translation based on word alignment (Brown et al., 1993), speech recognition to estimate parameters of Hidden Markov Models (Juang & Rabiner, 1990; Strom et al., 1999), image analysis for supervised and unsupervised learning of multi-dimensional HMMs (hidden Markov Models) (Joshi et al., 2006), parsing by unsupervised learning of dependency models (Spitkovsky et al., 2010), and gene finding by unsupervised learning of HMMs (Lomsadze et al., 2005). Although VT is NP-hard even for PCFGs (probabilistic context free grammars), which is proved by encoding the 3-SAT problem into PCFGs (Cohen & Smith, 2010), and is not a consistent estimation method unlike MLE (maximum likelihood estimation)(Lember & Koloydenko, 2007), it often outperforms and runs faster than the conventional EM algorithm.

We introduce this VT to PRISM which is a probabilistic extension of Prolog (Sato & Kameya, 2001; 2008). There are already multiple parameter learning methods available in PRISM. One is the EM algorithm, or more generally MAP (maximum a posteriori) estimation (Sato & Kameya, 2001). Another is VB (variational Bayes) (Sato et al., 2009) which approximately realizes Bayesian inference and learns pseudo counts assuming Dirichlet priors over parameters. They are implemented on PRISM's data structure called *explanation graph*s representing AND/OR boolean formulas made up of probabilistic ground atoms. Probabilities used in EM, MAP and VB are all computed by running the generalized IO algorithm or its variant on explanation graphs.

VT in PRISM runs on explanation graphs just like EM, MAP and VB but always computes a single probability of a single explanation called *Viterbi explanation* or most probable explanation. Not only this results in faster computation but thanks to the nature of VT's objective function which differs from the likelihood of data, VT tends to learn parameters yielding better prediction performance in tasks such as parsing as we see later.

In addition VT brings about a favorable side effect on PRISM. It makes unnecessary *the exclusiveness condition* imposed on PRISM programs to ensure efficient probability computation. This is because it always

Appearing in *Proceedings of the 29th International Conference on Machine Learning*, Edinburgh, Scotland, UK, 2012.

deals with a single probability of Viterbi explanation and hence there is no need for summing up probabilities of non-exclusive explanations. Consequently PRISM can learn parameters by VT from programs that do not satisfy the exclusiveness condition and compute Viterbi explanations using the learned parameters.

2. Viterbi training and PRISM

2.1. Viterbi training

Let x be hidden variables, y observed ones and $p(x, y \mid \theta)$ their joint distribution with parameters θ[1]. MLE estimates parameters θ from y as the maximizer of the (log) likelihood function $L_{EM}(y \mid \theta)$:

$$L_{EM}(y \mid \theta) \stackrel{\text{def}}{=} \log \sum_x p(x, y \mid \theta).$$

In the case of MAP (maximum a posteriori) estimation, we add a prior distribution $p(\theta)$ and use $L_{MAP}(y \mid \theta)$ as an objective function:

$$L_{MAP}(y \mid \theta) \stackrel{\text{def}}{=} \log \sum_x p(x, y \mid \theta)p(\theta).$$

What VT does is similar to MLE and MAP but it uses a different objective function $L_{VT}(y \mid \theta)$ defined as

$$L_{VT}(y \mid \theta) \stackrel{\text{def}}{=} \log \max_x p(x, y \mid \theta)p(\theta).$$

VT estimates parameters as the maximizer of $L_{VT}(y \mid \theta)$ by coordinate ascent that alternates the maximization of $\log p(x, y \mid \theta)$ w.r.t. x and the maximization of $\log p(x, y \mid \theta)$ w.r.t. θ:

$$x^{(n)} = \operatorname{argmax}_x \log p(x, y \mid \theta^{(n)}) \quad (1)$$

$$\theta^{(n+1)} = \operatorname{argmax}_\theta \log p(x^{(n)}, y \mid \theta) \quad (2)$$

Starting with appropriate initial parameters $\theta^{(0)}$, VT iterates the above two steps and terminates when $x^{(n+1)} = x^{(n)}$ holds (recall we assume random variables are discrete). Proving the convergence property of VT is straightforward.

$$\begin{aligned} L_{VT}(y \mid \theta^{(n+1)}) &= \log p(x^{(n+1)}, y \mid \theta^{(n+1)})p(\theta^{(n+1)}) \\ &\geq \log p(x^{(n)}, y \mid \theta^{(n+1)})p(\theta^{(n+1)}) \\ &\geq \log p(x^{(n)}, y \mid \theta^{(n)})p(\theta^{(n)}) \\ &= L_{VT}(y \mid \theta^{(n)}) \end{aligned}$$

So $L_{VT}(y \mid \theta^{(n)}) \leq L_{VT}(y \mid \theta^{(n+1)}) \leq 0$ for every $n = 0, 1, \ldots$ Since $\{L_{VT}(y \mid \theta^{(n)})\}_n$ is a monotonically increasing sequence with an upper bound, it converges as n goes to infinity.

[1] In this paper, we assume distributions are discrete.

2.2. VT for PRISM

Here we reformulate VT in the context of PRISM. We skip an introductory explanation of PRISM. The reader is referred to (Sato & Kameya, 2001; 2008) for its semantics, tabled search, explanation graph and probability computation.

Let DB be a PRISM program with parameters $\boldsymbol{\theta}$ and $P_{DB}(\cdot \mid \boldsymbol{\theta})$ a probability measure defined by DB. Also let $\boldsymbol{G} = G_1, \ldots, G_T$ be a list of observed iid goals, and $\mathrm{expl}(G_t)$ $(1 \leq t \leq T)$ the set of explanations ϵ_t (conjunctions of `msw` atoms) such that $\epsilon_t, DB \vdash G_t$. $\boldsymbol{G}$ corresponds to observed variables y and $\{\epsilon_t\}_{t=1}^T$ to hidden variables x in $p(x, y \mid \theta)$ in the previous subsection. We use $\theta_{i,v}$ for *parameter*s (probabilities) of multi-valued switch `msw(i,v)` $(v \in V_i)$ where i is a switch identifier (ground term) and V_i is a finite set of possible outcomes (ground terms) declared by `values/3` declaration in DB. So $\sum_{v \in V_i} \theta_{i,v} = 1$ holds. We put $\boldsymbol{\theta}_i = \{\theta_{i,v}\}_{v \in V_i}$ and $\boldsymbol{\theta} = \bigcup_i \boldsymbol{\theta}_i$ where i ranges over possible switch identifiers.

We introduce a prior distribution $p(\boldsymbol{\theta}_i) \propto \prod_{v \in V_i} \theta_{i,v}^{\alpha_{i,v}-1}$, Dirichlet distribution with hyper parameters $\{\alpha_{i,v}\}_{v \in V_i}$, over $\boldsymbol{\theta}_i$. In the following, to avoid the difficulty of zero-probability encountered in parameter learning, we assume *pseudo count* $\delta_{i,v} \stackrel{\text{def}}{=} \alpha_{i,v} - 1 > 0$ and use $\delta_{i,v}$ instead of $\alpha_{i,v}$.

Finally we define *Viterbi explanation* ϵ_t^* for a goal G_t as a most probable explanation for G_t given by

$$\epsilon_t^* = \operatorname{argmax}_{\epsilon_t \in \mathrm{expl}(G_t)} P_{DB}(\epsilon_t \mid \boldsymbol{\theta})p(\boldsymbol{\theta}). \quad (3)$$

The objective function $L_{VT}(\boldsymbol{G} \mid \boldsymbol{\theta})$ for VT in PRISM is now computed as follows.

$$\begin{aligned} &L_{VT}(\boldsymbol{G} \mid \boldsymbol{\theta}) \\ &= \log \max_{\epsilon_t \in \mathrm{expl}(G_t), 1 \leq t \leq T} \prod_{t=1}^T P_{DB}(\epsilon_t, G_t \mid \boldsymbol{\theta})p(\boldsymbol{\theta}) \\ &= \log \prod_{t=1}^T \max_{\epsilon_t \in \mathrm{expl}(G_t)} P_{DB}(\epsilon_t \mid \boldsymbol{\theta})p(\boldsymbol{\theta}) \\ &= \log \prod_{i,v} \theta_{i,v}^{\sum_{t=1}^T \sigma_{i,v}(\epsilon_t^*) + \delta_{i,v}} \\ &= \sum_{i,v} \left(\sum_{t=1}^T \sigma_{i,v}(\epsilon_t^*) + \delta_{i,v} \right) \log \theta_{i,v} \end{aligned}$$

where "i, v" ranges over those satisfying `msw(i,v)` in ϵ_t^* and $\sigma_{i,v}(\epsilon_t^*)$ is the number of occurrences of `msw(i,v)` in ϵ_t^*.

By substituting various definitions and formulas into equations (1) and and (2), we obtain the VT algorithm

for PRISM which alternately executes (4) and (5) where $\boldsymbol{\theta}^{(n)}$ stands for the set of parameters $\{\theta_{i,v}^{(n)}\}_{i,v}$ at step n.

$$\epsilon_t^{*(n)} = \mathrm{argmax}_{\epsilon_t \in \mathrm{expl}(G_t)} \log P_{DB}(\epsilon_t \mid \boldsymbol{\theta}^{(n)}) \quad (4)$$

$$\theta_{i,v}^{(n+1)} \propto \sum_{t=1}^{T} \sigma_{i,v}(\epsilon_t^{*(n)}) + \delta_{i,v} \quad (5)$$

Here (4) corresponds to (1) and (5) to (2) respectively.

Given observed goals $\boldsymbol{G} = G_1, \ldots, G_T$, we first perform tabled search for all explanations to build explanations graphs representing $\mathrm{expl}(G_t)$ for each $t (1 \leq t \leq T)$. Then starting from the initial parameters $\boldsymbol{\theta}^{(0)}$[2], we repeat (4) and (5) alternately while efficiently computing the Viterbi explanations (4) based on the explanation graphs until $\epsilon_t^{*(n+1)} = \epsilon_t^{*(n)}$ holds for all t $(1 \leq t \leq T)$. $\{\theta_{i,v}^{(n+1)}\}_{i,v}$ are then learned parameters.

One of the problems of VT is the sensitivity to the initial condition. So we need to carefully choose $\boldsymbol{\theta}^{(0)}$. Uniform distributions for $\boldsymbol{\theta}^{(0)}$ (Spitkovsky et al., 2010) and $\epsilon_t^{*(0)} (1 \leq t \leq T)$ (Cohen & Smith, 2010) are possible choices. In practice, we further add random restart to alleviate the sensitivity problem.

3. Learning experiments with probabilistic grammars

In this section[3] we apply VT to parsing tasks in natural language processing where observable variables are sentences and hidden variables are parse trees. We predict true parse trees for given sentences using probabilistic grammars (PCFG and PLCG) whose parameters are learned by VT and compare the parsing performance with EM, MAP and VT.

3.1. VT for PCFGs

Prior to describing the parameter learning experiment with a PCFG by VT, we briefly review how to write PCFGs in PRISM. In PCFGs, sentence derivation is carried out probabilistically. When there are k PCFG rules $\theta_1 : A \to \beta_1, \ldots, \theta_k : A \to \beta_k$ for a nonterminal A with probabilities $\theta_1, \ldots, \theta_k$ $(\theta_1 + \cdots + \theta_k = 1)$, A is expanded by $A \to \beta_i$ into β_i with probability θ_i. The probability of a parse tree τ is the product of probabilities associated with occurrences of CFG rules in τ and the probability of a sentence is the sum of probabilities of parse tress for the sentence.

```
values('S',[['S','S'],[a],[b]],[0.4,0.3,0.3]).

pcfg(L):- pcfg(['S'],L,[]).

pcfg([A|R],L0,L2):-
  ( get_values(A,_) ->  % msw(A,_) exists, so
      msw(A,RHS),       % A is a nonterminal
      pcfg(RHS,L0,L1)
  ; L0=[A|L1] ),
  pcfg(R,L1,L2).
pcfg([],L,L).
```

Figure 1. A PCFG program

Writing PCFG programs is easy in PRISM. Figure 1 is a PRISM program for a PCFG { `0.4:S→S S, 0.3:S→a, 0.3:S→b` }. In general, PCFG rules such as $\{ \theta_1 : A \to \beta_1, \ldots, \theta_k : A \to \beta_k \}$ are encoded by `values/3` declaration as

`values('A',[`$\beta_1, \ldots, \beta_k$`],[`$\theta_1, \ldots, \theta_k$`])`

where β_i is a Prolog list of terminals and nonterminals.

We wrote a PCFG program like Figure 1 for ATR corpus (Uratani et al., 1994) using an associated CFG[4]. The corpus contains labeled parse trees for 10,995 Japanese sentences whose average length is about 10. The associated hand-crafted CFG comprises 860 CFG rules (172 non-terminals and 441 terminals) and yields 958 parses/sentence on average.

We applied four learning algorithms, i.e. VT, EM, MAP and VB (Sato et al., 2009) available in PRISM to the PCFG program for ATR corpus[5] and compared their parsing performance. We conducted eight-fold cross validation for each algorithm to evaluate the quality of learned parameters in terms of three perfor-

[2] We may start from some $\epsilon_t^{*(0)}$ as well.

[3] Experiments are done using PRISM augmented with preliminarily implemented VT.

[4] In the experiment, to speed up parsing, we partially evaluated the PCFG program with individual CFG rules and obtained a specialized set of clauses representing PCFG rules. We ran them as a PRISM program.

[5] In PRISM, EM is a special case of MAP inference. We used random but almost uniform initialization of parameters and set uniformly pseudo counts $\delta_{i,v}$ to 1.0^{-9} for EM and 1.0 for MAP and VT, respectively. Similarly we uniformly set hyper parameters $\alpha_{i,v}$ to 1.0 for VB. The number of candidates for re-ranking in VB (Sato et al., 2009) was set to 5. In all cases, we set the number of random restarts to 50 and used the best parameter set that gave the largest value of objective functions, i.e. L_{EM} for EM, L_{MAP} for MAP and L_{VT} for VT. For the case of VB that learns pseudo counts, we chose the best set of pseudo counts giving the highest free energy (Sato et al., 2009).

mance measures, i.e. LT(labeled tree), BT(bracketed tree) and 0-CB(zero crossing brackets)[6]. The entire corpus is partitioned into eight sections. In each fold, one section is used as a test corpus and sentences in the remaining sections are used as training data. For each of EM, MAP, VT and VB, parameters (or pseudo counts) are learned by unsupervised learning from the training data. A parse tree is predicted, i.e. Viterbi explanation is computed for each sentence in the test corpus using learned parameters or using the approximate a posterior distribution learned by VB. The predicted trees are compared to answers, i.e. the labeled trees in the test corpus to compute LT, BT and 0-CB respectively. The performance figures are calculated as averages over eight folds. The results are summarized in Table 1.

Table 1. Parsing performance by PCFG

PCFG	Learning method			
Measure	VT(%)	EM(%)	MAP(%)	VB(%)
LT	**74.69**	70.02	70.31	72.13
BT	**77.87**	73.10	73.45	75.46
0-CB	83.78	84.44	84.89	**87.08**

The table shows that VT clearly outperforms the other three in terms of LT and BT by a considerable margin but it is the worst as far as 0-CB is concerned. This is understandable if we recall that VT looks for the best parameters for the best parse tree and hence widens the gap between the probability of the best tree and those not, which is thought to negatively affect less stringent criteria such as 0-CB.

Another thing to note is that the objective functions for EM, MAP and VB are similar in the sense that they all sum out hidden variables whereas the objective function for VT retains them. This fact together with Figure 1 seems to suggest that parsing performance is more affected by the difference among objective functions than the difference among learning methods.

[6] A parse tree is represented as a set of terms of the form (NP, 1, 3) which means a nonterminal NP spans from the 1st word to the 3rd word in the given sentence. NP is called a label. If two parse trees have the same representation, they are identical. LT is the ratio of correctly predicted trees in the test corpus, i.e. those that exactly match the labeled trees in the test corpus. BT is the same as LT but only cares about brackets and ignores labels. 0-CB is the ratio of predicted trees with non-conflicting bracketing with the test corpus.

```
values(lc('S','S'),[rule('S',['S','S'])]).
values(lc('S',a),[rule('S',[a])]).
values(lc('S',b),[rule('S',[b])]).
values(first('S'),[a,b]).
values(att('S'),[att,pro]).

plcg(L):- g_call(['S'],L,[]).

g_call([],L,L).
g_call([G|R],[Wd|L],L2):-
   ( G = Wd -> L1 = L     % shift operation
   ; msw(first(G),Wd),lc_call(G,Wd,L,L1) ),
   g_call(R,L1,L2).

lc_call(G,B,L,L2):-       % B-tree is completed
   msw(lc(G,B),rule(A,[B|RHS2])),
   ( G = A -> true ; values(lc(G,A),_) ),
   g_call(RHS2,L,L1),     % complete A-tree
   ( G = A -> att_or_pro(A,Op),
     ( Op = att -> L2 = L1 ; lc_call(G,A,L1,L2) )
   ; lc_call(G,A,L1,L2) ).

att_or_pro(A,Op):-
   ( values(lc(A,A),_) -> msw(att(A),Op) ; Op=att ).
```

Figure 2. A PLCG program

3.2. VT for PLCG

PCFGs assume top-down parsing. Contrastingly, there is a class of probabilistic grammars based on bottom-up parsing for CFGs called PLCGs (probabilistic left-corner grammars)(Manning, 1997; Roark & Johnson, 1999; Van Uytsel et al., 2001). Although they use the same set of CFG rules as PCFGs but attach probabilities not to expansion of nonterminals but to three elementary operations in bottom-up parsing, i.e. shift, attach and project. As a result they define a different class of distributions from PCFGs.

Programs for PLCGs look very different from those for PCFGs. Figure 2 is a PLCG program which is a dual version of the PCFG program in Figure 1 with the same underlying CFG {S→S S, S→a, S→b}. It generates sentences using the first set of 'S' and the left-corner relation for this CFG[7].

The program works as follows. Suppose nonterminals G and B are in the left-corner relation and G is waiting for a B-tree, i.e. a subtree with the root node labeled B, to be completed. When a B-tree is completed, the program probabilistically chooses a CFG rule of the form A → Bβ to further grow the B-tree using this rule. Upon the completion of the A-tree and if G=A, the attach operation or the projection is probabilistically

[7] We assume the reader is familiar with parsing theory.

chosen. By replacing `values/3` declarations appropriately, this program is applicable to any PLCG.

We have developed a PLCG program similarly to the PCFG program for ATR corpus and applied VT, EM, MAP and VB to measure their parsing performance in terms of LT, BT and 0-CB by eight-fold cross validation. We obtained Table 2. As expected VT outperformed EM, MAP and VB in terms of LT and BT but it was the second best in terms of 0-CB. Also we see that the PCLG achieved better parsing performance than the PCFG with VT,EM and MAP but no so with VB.

Table 2. Parsing performance by PLCG

PLCG	Learning method			
Measure	VT(%)	EM(%)	MAP(%)	VB(%)
LT	**76.26**	71.81	71.17	71.15
BT	**78.86**	75.17	74.28	74.28
0-CB	87.45	**95.92**	86.03	86.04

4. Removing the exclusiveness condition

PRISM assumes the exclusiveness condition on programs to simplify probability computation. It says that the clause bodies must be exclusive when successfully proved (Sato & Kameya, 2001). Although most of generative probabilistic models such as BNs (Bayesian networks), HMMs and PCFGs are naturally described as PRISM programs satisfying the condition, removing it certainly gives us more freedom of probabilistic modeling. Theoretically it is possible to remove it by introducing BDDs (binary decision diagrams) as ProbLog (De Raedt et al., 2007; Kimmig et al., 2008) and PITA (Riguzzi & Terrance Swift, 2011) do. If, however, we are only interested in obtaining Viterbi explanations determined by parameters learned from data as we are in many cases, we may forget about BDDs and the exclusiveness condition because VT is applicable to any programs regardless of the exclusiveness condition and learning parameters by VT and computing Viterbi explanations using the learned parameters is enough. Moreover, as we have seen in the previous section, VT can learn parameters that exhibit excellent performance in prediction tasks.

However, there is one caveat: While VT allows us to use probabilistic inclusive-or in the clause bodies, parameters learned by VT might cause a heavily biased selection of disjuncts. For example look at Figure 3.

This program represents a probabilistic version of inclusive-or "`b` $\vee$ `c`" as a goal "`a`". Note that each

```
values(cl(b),[yes,no]).
values(cl(c),[yes,no]).
a:- (b ; c).
b:- msw(cl(b),yes).
c:- msw(cl(c),yes).
```

Figure 3. Probabilistic inclusive or

of `msw(cl(b),yes)` and `msw(cl(c),yes)` occurs only once and hence their parameters are independently adjustable. Consequently learning parameters by VT from goals like `[a,a,...]` would inevitably result in $P_{DB}(\texttt{msw(cl(b),yes)}) = 1$ or $P_{DB}(\texttt{msw(cl(b),yes)}) = 1$ and never give, say, $P_{DB}(\texttt{msw(cl(b),yes)}) = P_{DB}(\texttt{msw(cl(b),yes)}) = 0.5$.

In general VT chooses best parameters for the Viterbi explanations. It therefore prefers to assign probability 1 to `msw`s that occur only once in a program and hence often gives extreme probabilities to disjuncts. If one considers such a behavior of VT undesirable, one way to remedy it is to set large values to pseudo counts $\delta_{i,v}$ in (5).

5. Discussion

In this paper we discussed VT for PRISM that runs on explanation graphs, a single data structure in PRISM. VT thus implemented requires time for all solution search (by tabling) and also space to store discovered explanation graphs. It is possible, however, to implement VT without explanation graphs, and to realize much more memory efficient VT by repeating search for a Viterbi explanation in each cycle of VT. We note this approach particularly fits well with mode-directed tabling (Zhou et al., 2010). In mode-directed tabling, we can search for partial Viterbi explanations for subgoals efficiently without constructing explanation graphs and put them together to form a larger Viterbi explanation for the goal. Currently however mode-directed tabling is not available in PRISM. We are planing to incorporate it in PRISM in the near future.

6. Conclusion

We enhanced PRISM's probabilistic modeling by introducing VT (Viterbi training) to PRISM. Although it has already been used in various models under various names (Brown et al., 1993; Juang & Rabiner, 1990; Strom et al., 1999; Joshi et al., 2006; Spitkovsky et al., 2010; Lomsadze et al., 2005),

we made the following contributions to VT. One is a generalization by deriving a generic VT algorithm for PRISM, thereby making it applicable to a very wide class of discrete models described by PRISM programs that range from BNs to probabilistic grammars. The other is an empirical evaluation of VT that complements the scarcity of literature on applications of VT to pure PCFGs and PLCGs. We conducted two learning experiments with them using VT preliminarily implemented in PRISM and confirmed VT's excellent parsing performance compared to EM, MAP and VB.

References

Brown, P.F., Pietra, V.J.D., Pietra, S.A.D., and Mercer, R.L. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19:263–311, 1993.

Cohen, S.B. and Smith, N.A. Viterbi Training for PCFGs: Hardness Results and Competitiveness of Uniform Initialization. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL'10)*, pp. 1502–1511, 2010.

De Raedt, L., Kimmig, A., and Toivonen, H. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pp. 2468–2473, 2007.

Joshi, D., Li, J., and Wang, J.Z. A computationally efficient approach to the estimation of two- and three-dimensional hidden markov models. *IEEE Transactions on Image Processing*, 15(7):1871–1886, 2006.

Juang, B.H. and Rabiner, L.R. The segmental K-means algorithm for estimating parameters of hidden Markov models. *IEEE Transactions on Signal Processing*, 38:1639 – 1641, 1990.

Kimmig, A., Costa, V., Rocha, R., Demoen, B, and De Raedt, L. On the efficient execution of problog programs. In *Proceedings of the 24th International Conference on Logic Programming (ICLP'08)*, pp. 175–189, 2008.

Lember, J. and Koloydenko, A. Adjusted viterbi training. *Probability in the Engineering and Informational Sciences*, 21(3):451–475, 2007.

Lomsadze, A., Ter-Hovhannisyan, V., Chernoff, Y.O., and Borodovsky, M. Gene identification in novel eukaryotic genomes by self-training algorithm. *Nucleic Acids Research*, 33:6494–6506, 2005.

Manning, C.D. Probabilistic parsing using left corner language models. In *Proceedings of the 5th International Conference on Parsing Technologies (IWPT-97)*, pp. 147–158. MIT Press, 1997.

Riguzzi, F. and Terrance Swift, T. The PITA system: Tabling and answer subsumption for reasoning under uncertainty. *Theory and Practice of Logic Programming (TPLP)*, 11(4-5):433–449, 2011.

Roark, B. and Johnson, M. Efficient probabilistic top-down and left-corner parsing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pp. 421–428, 1999.

Sato, T. and Kameya, Y. Parameter learning of logic programs for symbolic-statistical modeling. *Journal of Artificial Intelligence Research*, 15:391–454, 2001.

Sato, T. and Kameya, Y. New Advances in Logid-Based Probabilistic Modeling by PRISM. In De Raedt, L., Frasconi, P., Kersting, K., and Muggleton, S. (eds.), *Probabilistic Inductive Logic Programming*, pp. 118–155. LNAI 4911, Springer, 2008.

Sato, T., Kameya, Y., and Kurihara, K. Variational Bayes via Propositionalized Probability Computation in PRISM. *Annals of Mathematics and Artificial Intelligence*, 54:135–158, 2009.

Spitkovsky, V.I., Alshawi, H., Jurafsky, D., and Manning, C.D. Viterbi training improves unsupervised dependency parsing. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pp. 9–17, July 2010.

Strom, N., Hetherington, L., Hazen, T.J., Sandness, E., and Glass, J. Acoustic modeling improvements in a segment-based speech recognizer. In *Proceedings of IEEE ASRU Workshop*, 1999.

Uratani, N., Takezawa, T., Matsuo, H., and Morita, C. ATR integrated speech and language database. Technical Report TR-IT-0056, ATR Interpreting Telecommunications Research Laboratories, 1994. In Japanese.

Van Uytsel, D.H., Van Compernolle, D., and Wambacq, P. Maximum-likelihood training of the PLCG-based language model. In *Proceedings of the IEEE Automatic Speech Recognition and Understanding Workshop 2001 (ASRU'01)*, 2001.

Zhou, N.-F., Kameya, Y., and Sato, T. Mode-directed tabling for dynamic programming, machine learning, and constraint solving. In *Proceedings of the 22th International Conference on Tools with Artificial Intelligence (ICTAI-2010)*, 2010.