Learning Stationary Temporal Probabilistic Networks

Daniel Nikovski

The Robotics Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213
danieln@cs.cmu.edu

Abstract

The paper describes a method for learning representations of partially observable Markov decision processes in the form of temporal probabilistic networks, which can subsequently be used by robotic agents for action planning and policy determination. A solution is provided to the problem of enforcing stationarity of the learned Markov model. Several preliminary experiments are described that confirm the validity of the solution.

1 Introduction

Robotic systems must reason, plan, and learn under the conditions of significant uncertainty due to unexpected influences and imprecise sensors and effectors. Decision-theoretic planning (DTP) has emerged as the paradigm of choice for planning and sequential decision making under uncertainty, because of its solid foundations in probability theory and normative decision science [DW91]. DTP has been applied with much success to several robotic tasks such as navigation ([CKK96, KS96], sensor monitoring ([NB94]), people following ([BDKL92]), and autonomous driving [FHKR95, FOPR97].

In all of these applications, the environment and the effects of the agent's actions on it are modeled by means of temporal probabilistic networks (TBN), also known as dynamic belief nets (DBN). In control theory terms, a TPN is a model of the dynamics of a system expressed by means of a Bayesian network with specific structure [DW91]. Unlike classical control theory, however, the state of the system is expressed as a truth assignment over discrete random variables, and not as a real-valued vector. These random variables roughly correspond to propositions in classical planning, and the set of assignments on them corresponds to the state set of a Markov decision process (MDP).

TPNs can either be constructed manually or learned from data. Since TPNs are essentially Bayesian networks with specific structure, one way to learn them autonomously from data is to adapt a general algorithm for learning Bayesian nets, such as the ones described in [Heck95, RBKK95]. However, when doing so, one should keep in mind that a TPN is a model of a real dynamical system, which imposes further constraints on the parameters of the learned network. Ignoring these constraints might result in higher sample complexity, and even worse, in learning a wrong model of the system.

The paper discusses one such additional constraint, previously ingored when learning TPNs from data: the requirement for state stationarity of the dynamical system. The rest of the paper is organized as follows. Section 2 reviews two algorithms for learning in TPNs. Section 3 discusses the stationarity problem in TPNs and describes a solution to that problem. Section 4 describes several preliminary experiments that confirm the validity of the solution, and section 5 discusses future experiments and concludes.

2 Learning in temporal probabilistic networks

Fig. 1 shows the generic structure of a TPN. The network can have two or more time slices, each of which has very similar structure. Each time slice has a set $\mathcal{S}$ of state nodes S_i, $i = 1, n$, a set $\mathcal{O}$ of observation nodes O_i, $i = 1, m$, and an action node A that represents several actions a_i, $i = 1, p$. Each of the state nodes can be in a number of discrete states s_j. If the state nodes are Boolean (Bernoulli random variables), they correspond to propositions in a classical planning domain. Probabilistic networks with action nodes have traditionally been called influence diagrams [Pear88], and correspondingly a TPN with actions can also be viewed as a dynamic influence diagram.

Each of the observation nodes in its turn can register a number of observations o_j. The probabilities $P(O_i = o_j | S_k = s_l)$ represent the *observation model* of the underlying POMDP, which is typically assumed to be the same for all time slices.

The *state evolution model* is represented by means of the probabilities $P[S_i(t+1) = s_j | S_1(t) = s_{k_1}, \ldots, S_n(t) = s_{k_n}, A_q(t)]$, stored in the CPT of node $S_i(t+1)$. The dependency does not have to be on all nodes from the previous time slice. However, the parents of state nodes in state $t+1$ can be only state nodes in the previous time slice t, which is equivalent to the assumption that the underlying process is Markovian.

Learning in TPNs can be performed by means of general algorithms for learning probabilistic networks from data, such as those proposed in [Heck95, RBKK95]. The goal of learning is to find values for the entries in the conditional probability tables (CPT) of a TPN, which maximize the log-likelihood $\ln P(\mathbf{D}|\mathbf{w})$ that the training data set $\mathbf{D}$ was generated by the network with parameters $\mathbf{w}$. Each case D_l from the data set $\mathbf{D}$ consists of assignments for the observable nodes in O. Two such algorithms will be considered: the first one based on gradient ascent, and the other one on the expectation-maximization (EM) algorithm.

2.1 Gradient ascent

Russell et al. ([RBKK95]) proposed a particularly simple learning rule which performs gradient ascent in the space of CPT entries:

$$\Delta w_{ijk} = \eta \sum_l \frac{\partial P(D_l|\mathbf{w})/\partial w_{ijk}}{P(D_l|\mathbf{w})} = \sum_l \frac{P(S_{ij}, \mathbf{U}_{ik}|D_l, \mathbf{w})}{w_{ijk}},$$

$$w_{ijk} = P(S_{ij}|\mathbf{U}_{ik}) = P(S_i = s_{ij}|\mathbf{U}_i = \mathbf{u}_{ik})$$

where w_{ijk} is the entry of the CPT of state node S_i that designates the probability that this node S_i will be in its j-th state s_{ij} given that its parents $\mathbf{U}_i$ are in their k-th configuration $\mathbf{u}_k$ [RBKK95]. The summations index l runs over all data cases D_l in $\mathbf{D}$. As pointed out in [RBKK95], the quantity $P(S_{ij}, \mathbf{U}_{ik}|D_l, \mathbf{w})$ can be obtained by means of belief updating in a general-purpose belief net reasoning system.

2.2 Baum-Welch

Since TPNs model a Markov decision process, it is possible to adapt algorithms for learning MDPs to the task of learning TPNs. In particular, the algorithm from [KS96] can be adapted, which in its turn has been adapted from the Baum-Welch algorithm ([Rabi89]). The latter is a variant of the EM algorithm popular in statistics. The learning rule of the Baum-Welch algorithm for the two-slice DPN is:

$$\hat{w}_{ijk}^{T+1} = \frac{\sum_l \xi_{i,l}^T(j,k)}{\sum_l \gamma_{i,l}^T(j)},$$

where $\hat{w}_{ijk}^{T+1}$ is the estimate of the transition probability $P[S_i(t+1) = s_{ik}|\mathbf{U}_i(t) = \mathbf{u}_{ij}]$ after iteration $T+1$, and

$$\xi_{i,l}^T(j,k) = P[S_i(t+1) = s_{ik}, \mathbf{U}_i(t) = \mathbf{u}_{ij}|D_l, \hat{\mathbf{w}}^k]$$

$$\gamma_{i,l}^T(j) = P[\mathbf{U}_i(t) = \mathbf{u}_{ij}|D_l, \hat{\mathbf{w}}^k].$$

It can be seen that the quantity $\xi_{i,l}(j,k)$ is in fact used in Russell's algorithm too. For a TPN, the learning rule of that algorithm can also be written as:

$$\hat{w}_{ijk}^{T+1} = \hat{w}_{ijk}^T + \eta \frac{\sum_l \xi_{i,l}^k(j,k)}{\hat{w}_{ijk}^T}$$

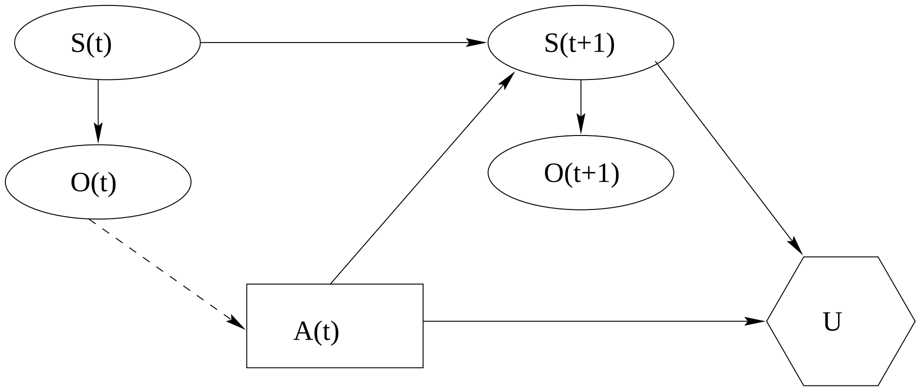

Figure 1: A temporal probabilistic network is an influence diagram that has chance nodes for states and observations, decision nodes for actions, and utility nodes for specifying goal states. The state at time $t+1$ depends on the state and action at time t, and the observations depend on the state in their respective time slice. The link from action nodes (A) to utility nodes (U) can be used to include the cost of actions in policy determination. The dotted link from percepts to actions is the policy that has to be determined by the planning algorithm.

That is, Baum-Welch and the gradient ascent algorithm make use of the same quantities, but in quite a different manner: while the gradient ascent algorithm performs small steps in parameter space around the previous estimate of the parameter, Baum-Welch completely re-estimates the parameter value.

3 Enforcing stationarity

Several amendments are necessary when using these algorithms for learning temporal probabilistic nets. The observation models for all time slices have to be the same. Russell et al. proposed that this constraint be enforced by sharing parameters between time slices. Similarly, the state evolution model can be shared among time slices [RBKK95]. However, one important detail remains. Markov processes have the property of being stationary, if the state evolution probabilities are constant over time [Papo91]:

$$\mathcal{P}[\mathcal{S}(t)] = \mathcal{P}[\mathcal{S}(t+1)],$$

where $\mathcal{P}[\mathcal{S}(t)]$ and $\mathcal{P}[\mathcal{S}(t+1)]$ are the prior probability distributions for time slices t and $t+1$ respectively. The algorithm proposed in [RBKK95] enforces a different condition:

$$\mathcal{P}[\mathcal{S}(t+1)|\mathcal{S}(t), A(t)] = \mathcal{P}[\mathcal{S}(t+2)|\mathcal{S}(t+1), A(t)]$$

The latter condition does not imply the former. Violating the condition $\mathcal{P}[\mathcal{S}(t)] = \mathcal{P}[\mathcal{S}(t+1)]$ results in wrong prior beliefs for states. By casting Bayes rule in likelihood form, we can see that all errors in prior odds $Odds(S)$ are directly magnified to much bigger errors in posterior odds $Odds(S|O)$ when observations are made:$Odds(S|O) = L(O|S)Odds(S)$, where $L(O|S) = P(O|S)/P(O|\bar{S})$ is the likelihood of observing O in state S. Typically, the sensor model $P(O|S)$ is designed in such a way that the observations are highly diagnostic of the underlying state, which implies high values for the likelihood $L(O|S)$, which in its turn increases proportionally the error in posterior beliefs.

The key to the solution of the problem is to observe that the conditional distribution $\mathcal{P}[\mathcal{S}(t+1)|\mathcal{S}(t), A]$ uniquely determines the prior distribution $\mathcal{P}[\mathcal{S}(t)]$. However, in order to learn the distribution $\mathcal{P}[\mathcal{S}(t+1)|\mathcal{S}(t), A]$, one has to know the prior distribution $\mathcal{P}[\mathcal{S}(0)]$ for the first time slice. There seems to be a vicious circle, but in fact there exists a simple solution. We can start with an initial guess about the prior distribution $\mathcal{P}[\mathcal{S}(0)]$ and alternate estimations of the conditional distribution $\mathcal{P}[\mathcal{S}(t+1)|\mathcal{S}(t), A]$ (by means of the learning algorithm) with re-estimations of the prior distribution $\mathcal{P}[\mathcal{S}(0)]$ (by means of nested iterations to a fixed point, described below).

Let us consider first a network without actions, i.e., one that represents a hidden Markov chain. We have to solve for the probability distribution $\Omega^* = \mathcal{P}[\mathcal{S}]$ such that when multiplied by the transition matrix M, represented by the conditional distribution $\mathcal{P}[\mathcal{S}(t+1)|\mathcal{S}(t)]$, the same distribution results: $\Omega^* = \Omega^* M$. The distribution Ω^* is also called an *invariant distribution* of the Markov chain. It is hard to find this distribution analytically. However, a simple method is described in [BT89] for finding Ω^* by means of iterating to a fixed point. Starting with an initial guess $\Omega(0)$, we iterate $\Omega(T+1) = \Omega(T)M$, or, equivalently, $\Omega(T+1) = \Omega(0)M^T$.

We can implement the former formula quite easily by means of a TPN: we start with an initial guess for the priors $\mathcal{P}[\mathcal{S}(t)]$ in the first slice, do belief updating, and read off the values of the state nodes $\mathcal{P}[\mathcal{S}(t+1)]$ in the second slice. Those are transferred back to the first slice, and the process is repeated until the distributions in the two slices converge to the same value.

For the case of POMDPs with action nodes, we can do a similar process, but using the *expected* transition matrix of the POMDP. We need to set priors $P(a_i)$ on the actions — unless we have special reasons to prefer one action to another, we can make them equally likely. In system identification terms, this is a way of providing persistent excitation for the learning algorithm. The transition matrices M_i for all actions a_i can be weighted by those priors in order to compute the effective transition matrix M of the POMPD: $M = \sum_{i=1}^{p} M_i P(a_i)$. Substituting in the fixed-point iteration formula, we get

$$\begin{aligned}\Omega(T+1) = \quad & \Omega(T)M = \\ & \Omega(T)\textstyle\sum_{i=1}^{p} M_i P(a_i) = \\ & \textstyle\sum_{i=1}^{p}[\Omega(T)M_i]P(a_i).\end{aligned}$$

That is, we can set the action node to all actions in turn, do belief updating, read off the resulting distributions in the second slice and weight them individually by the priors for the respective actions. After all actions are tried in turn, the weighted distribution is moved to the first slice and iterations continue until convergence.

4 Experiments with the algorithm

Gradient ascent and Baum-Welch were tested first on an HMM with two states, s_1 and s_2, and two observations o_1 and o_2, and later on a POMDP. A data set with 500 data cases was generated by logical sampling of a known temporal belief net. The net to be learned was initialized with the correct sensor model for both time slices. The objective of learning was to find the transition probabilities $P[s_1(t+1)|s_1(t)]$ and $P[s_2(t+1)|s_1(t)]$, and implicitly the prior probability $P(s_1)$ by using the fixed-point iteration method from the previous section.

A number of trial runs were performed, with different sensor and transition models. It was found that in general Baum-Welch was much more successful than gradient ascent in learning the parameters of this particular architecture. For the sake of comparison, the increase in log-likelihood of the data for the two algorithms is plotted against the iteration number in Fig. 2. The observation model for this experiment was $P(o_1|s_1) = P(o_2|s_2) = 0.99$, i.e., the observations indicate very well which state the HMM is in. The dynamics of the HMM was also almost completely deterministic: $P[s_2(t+1)|s_1(t)] = P[s_1(t+1)|s_2(t)] = 0.99$. It can be seen that the increase in log-likelihood for Baum-Welch is more than that for Russell's algorithm. Furthermore, typically Russell's algorithm converges to a set of parameters much worse than those found by Baum-Welch, as shown in Fig. 3 for another sample run.

Adding actions to an HMM turns it into a POMDP. Learning the dynamics in this case simply means learning transition probabilities for each action. For another set of experiments two actions were added, A and B, such that A kept the POMDP in the same state with high probability, while B transferred it into the other state with high probability. Several experiments were

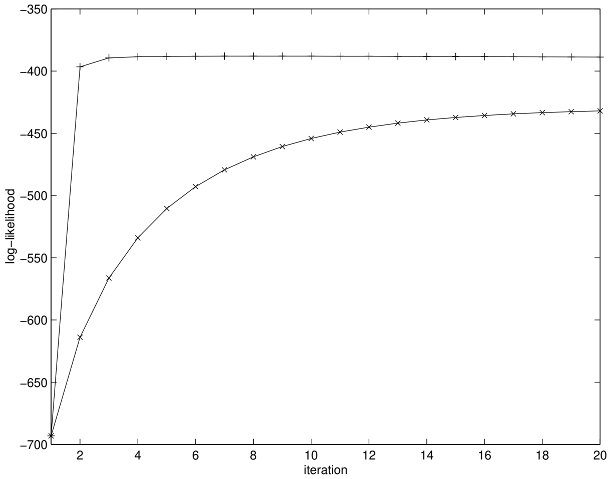

Figure 2: Comparison between Russell's algorithm (x) and Baum-Welch (+) with the same data set, initial configuration of parameters, and learning rate.

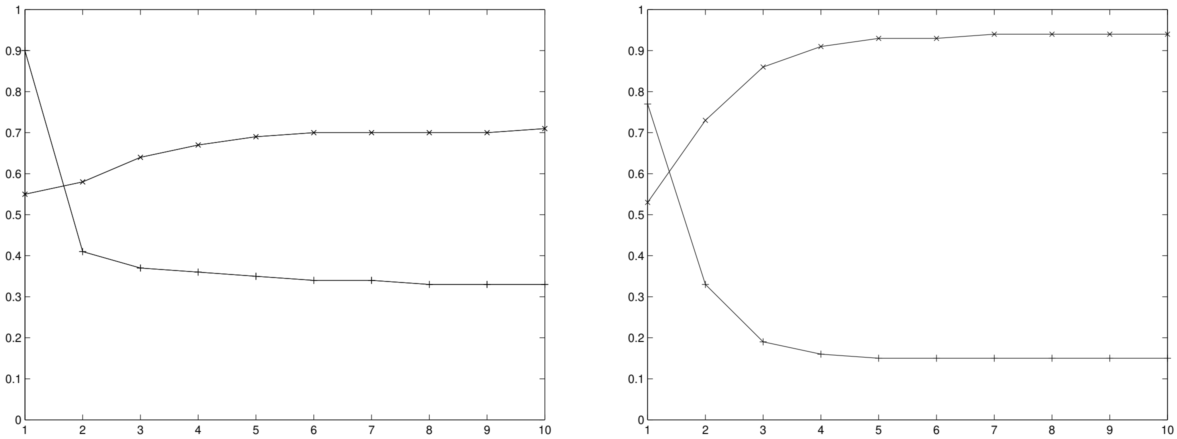

Figure 3: Convergence of the parameters $P[s_2(t+1)|s_1(t)]$ and $P[s_2(t+1)|s_2(t)]$ with (a) Russell's algorithm and (b) Baum-Welch. The true probabilities are 0.99 and 0.01.

performed with various level of noise in the dynamics and observation models. The goal state was set to s_2 by giving utility 100 for it and utility 0 for s_1. The correct policy in this case is $\pi(O_1) = B$, $\pi(O_2) = A$.

Fig. 4 shows a trial run with low noise (1% only). The algorithm clearly learns very well the transition probabilities for both actions. Certainly, a model needs only be good enough so that the correct policy can be determined. For the case of 1% noise only, the correct policy is found immediately after the first iteration of the learning algorithm. In another experiment with 20% noise in observations ($P(o_1|s_1) = P(o_2|s_2) = 0.8$), the correct policy could be determined in the seventh iteration.

5 Conclusion and future work

The main result reported in the paper is a method for enforcing stationarity in learned models by means of iterations to a fixed point to determine invariant prior distributions on state. The experiments demonstrated that the method works on simple example problems. Further experimental work should verify if the same algorithm can be extended to more than one state nodes.

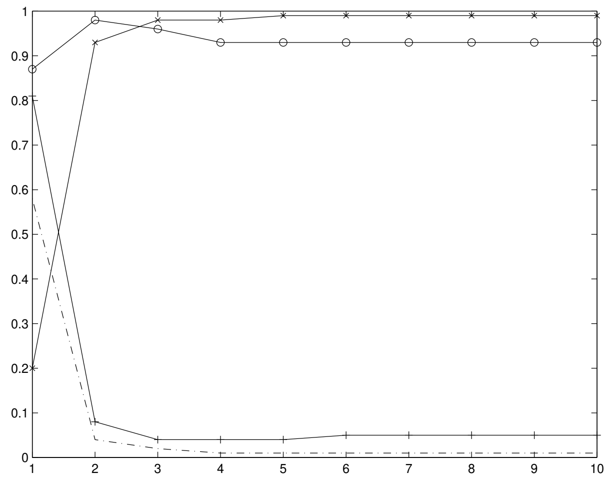

Figure 4: Convergence of Baum-Welch for the action models of actions A and B.

References

[BDKL92] Basye, K., Dean, T., Kirman, J., Lejter, M. (1992). A decision-theoretic approach to planning, perception, and control. IEEE Expert, August 1992, pp. 58-65.

[BT89] Bertsekas, D.P., and Tsitsiklis, J.N. (1989). *Parallel and Distributed Computation*, Prentice-Hall.

[CKK96] Cassandra, A.R., Kaelbling, L.P., and Kurien, J.A. (1996). Acting under uncertainty: Discrete Bayesian models for mobile robot navigation, in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, 1996.*

[DW91] Dean, T, and Wellman, M. (1991). *Planning and Control*, Morgan Kaufman, 1991.

[FHKR95] Forbes, J., Huang, T., Kanazawa, K., and Russell, S. (1995) The BAT-mobile: Towards a Bayesian automated taxi. In *Proc. Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.

[FOPR97] , Forbes, J., Oza, N., Parr, R., and Russell, S. (1997). Feasibility Study of Fully Automated Traffic Using Decision-Theoretic Control. California PATH Research Report UCB-ITS-PRR-97-18, Institute of Transportation Studies, University of California, Berkeley. April 1997.

[Heck95] . Heckerman, D. (1995). *A Tutorial on Learning Bayesian Networks*, Technical Report, Microsoft Research, Number MSR-TR-95-06, March 1995.

[KS96] S. Koenig and R.G. Simmons (1996). Unsupervised learning of probabilistic models for robot navigation, in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).*

[NB94] Dynamic belief networks for discrete monitoring, *IEEE Transactions on Systems, Man and Cybernetics*, 24(11), pp. 1593-1610, November 1994.

[Pear88] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems*. San Mateo: Morgan Kaufmann.

[Papo91] Papoulis, A. (1991). *Probability, Random Variables, and Stochastic Processes*, 3rd edition, McGraw Hill.

[Rabi89] Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition, *Proceedings of the IEEE*, 77(2), pp. 257-285, February 1989.

[RBKK95] Russell, S., Binder, J., Koller, D., and Kanazawa, K. (1995). Local learning in probabilistic networks with hidden variables, in *Proc. Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada.