Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS 2021)

Data-Driven Decision-Theoretic Planning using Recurrent Sum-Product-Max Networks

Hari Tatavarti, [1] **Prashant Doshi,** [2] **and Layton Hayes** [1]
[1] Institute for AI, University of Georgia, Athens, GA 30602 USA
[2] Dept. of Computer Science & Institute for AI, University of Georgia, Athens, GA 30602 USA
pdoshi@uga.edu

Abstract

Sum-product networks (SPN) are knowledge compilation models and are related to other graphical models for efficient probabilistic inference such as arithmetic circuits and AND/OR graphs. Recent investigations into generalizing SPNs have yielded sum-product-max networks (SPMN) which offer a data-driven alternative for decision making that has predominantly relied on handcrafted models. However, SPMNs are not suited for decision-theoretic planning which involves *sequential* decision making over multiple time steps. In this paper, we present recurrent SPMNs (RSPMN) that learn from and model decision-making data over time. RSPMNs utilize a template network that is unfolded as needed depending on the length of the data sequence. This is significant as RSPMNs not only inherit the benefits of SPNs in being data driven and mostly tractable, they are also well suited for planning problems. We establish soundness conditions on the template network, which guarantee that the resulting SPMN is valid, and present a structure learning algorithm to learn a sound template. RSPMNs learned on a testbed of data sets, some generated using RDDLSim, yield MEUs and policies that are close to the optimal on perfectly-observed domains and easily improve on a recent batch-constrained RL method, which is important because RSPMNs offer a new model-based approach to offline RL.

Introduction

Decision-theoretic planning (DTP) views planning as a sequence of decisions, each optimizing the planner's combined immediate and longer-term utility. However, DTP relies on accurately specifying the planning problem – a difficult task in practice – and DTP is essentially intractable for all but the simplest problems. This motivates investigating new tractable models for DTP, which can be learned directly from data, thereby allowing the tremendous progress of machine learning techniques to cross over into planning.

Arithmetic circuits (Huang, Chavira, and Darwiche 2006) and sum-product networks (SPN) (Poon and Domingos 2011) directly learn a network polynomial that is graphically represented as a network of sum and product nodes from domain data. Evaluations of the polynomial provide the joint or conditional distributions under conditions of validity. These graphical models are appealing because most types of inference can be performed in time that is *linear* in the size of the network. On the other hand, inference in Bayesian networks is generally exponential. A limitation of SPNs is that the size of the learned network is not bounded. Given the benefit of these generative models, Melibari et al. (2016) introduced recurrent SPNs for modeling sequence data of varying length and to perform tractable inference on sequences.

Sum-product-max networks (SPMN) (Melibari, Poupart, and Doshi 2016) generalize SPNs by adding two new types of nodes: max and utility nodes. Max nodes correspond to decision variables and utility nodes to the reward function, which allow SPMNs to computationally represent a probabilistic decision-making problem. If the SPMN learned from data is valid by satisfying a set of properties, then it correctly encodes a function that computes the maximum expected utility given the partial order between the variables. As such, SPMNs represent a shift in paradigm for decision-making models: from being primarily handcrafted to enabling machine learning from decision-making data.

Motivated by these recent generalizations of the SPN, we present a new graphical model that extends the twin benefits of an SPN (tractable inference and directly learned from data) to a new class of problems. This new model, which we refer to as a recurrent SPMN (RSPMN) can be seen as a synthesis of a recurrent SPN and an SPMN: it allows extending the decision-making problem across multiple time steps thereby modeling DTP problems for the first time. Given temporal decision-making data of any finite length, we present an effective method for learning an RSPMN of any finite length from this data and evaluating it to obtain the maximum expected utility (MEU) and the corresponding policy. A key component of the learned model is the *template network*, whose repeated application makes the temporal generalization possible.

We prove that unfolding the learned RSPMN produces a valid SPMN, which, in combination with a result from Melibari et al. (2016), establishes that its evaluation is equivalent to using the *sum-max-sum* rule. On a testbed of decision-making datasets from simulations in perfectly-observed domains, learned RSPMNs generate MEUs that are near optimal. RSPMNs offer a model-based approach to DTP where sequential data has already been collected. It is also applicable to offline (batch) reinforcement learning; thus we compare the MEUs with those from a recent batch Q-learning method (Fujimoto et al. 2019) and report favorable results.

Preliminaries

We briefly review SPNs followed by its generalization to single-shot decision-making contexts, SPMNs. An SPN (Poon and Domingos 2011) over n random variables $X_1, \ldots, X_n$ is a rooted directed acyclic graph (DAG) whose leaves are the distributions of the random variables and whose internal nodes are sums and products. Each edge emanating from a sum node has a non-negative weight. The value of a product node is the product of the values of its children. The value of a sum node is the weighted sum of its children's values. The value of an SPN is the value of its root, which is the output of a polynomial whose variables are the indicator variables and the coefficients are the weights (Darwiche 2000). An SPN is *valid* iff the normalized polynomial represents the joint probability distribution over the variables and yields the correct marginals. Completeness and decomposability are sufficient conditions for validity. Both impose some conditions on the *scope* of a node.

Definition 1 (Scope) *The scope of a node is the union of scopes of its children, where the scope of a leaf node is the set of random variables whose distribution it holds.*

In other words, the scope is the set of variables that appear in the sub-SPN rooted at that node. Next, we define the conditions sufficient for an SPN to be valid.

Definition 2 (Sum-complete) *An SPN is complete iff all children of the same sum node have the same scope.*

Definition 3 (Decomposable) *An SPN is decomposable iff no variable appears in the scopes of more than one child of a product node.*

Various structure and parameter learning algorithms have been presented to learn valid SPNs from data (Poon and Domingos 2011; Adel, Balduzzi, and Ghodsi 2015; Gens and Domingos 2013; Lowd and Rooshenas 2013). Most types of inference on the structure thus learned is tractable in the size of the network.

SPMNs (Melibari, Poupart, and Doshi 2016) generalize SPNs by adding max nodes that represent decision variables and utility nodes to represent the utility function. An SPMN over decision variables $D_1, \ldots, D_m$, random variables $X_1, \ldots, X_n$, and utility functions $U_1, \ldots, U_k$ is a rooted directed acyclic graph. Its leaves are either distributions over random variables or utility nodes that hold constant values. An internal node is either a sum, product, or max node. Each max node corresponds to one of the decision variables and each outgoing edge from a max node is labeled with one of the possible values of the corresponding decision variable. Value of a max node i is $max_{j \epsilon Children(i)} v_j$, where $Children(i)$ is the set of children of i, and v_j is the value of the subgraph rooted at child j.

Recall the concepts of information sets and partial ordering in influence diagrams (Koller and Friedman 2009). Information sets $I_0, \ldots, I_m$ are subsets of the random variables such that the random variables in the information set I_{i-1} are observed before the decision associated with variable D_i, $1 \leq i \leq m$, is made. Any information set may be empty and variables in I_m need not be observed before some decision node. An ordering between the information sets may be established as follows: $I_0 \prec D_1 \prec I_1 \prec D_2, \ldots, \prec D_m \prec I_m$. This is a partial order, denoted by $P^{\prec}$, because variables within each information set may be observed in any order. Melibari et al. show that a set of properties are sufficient to ensure that an SPMN correctly encodes a function that computes the MEU given the partial order between the variables and some utility function U. An SPMN is valid if it satisfies Defs. 2 and 3, and two new additional properties:

Definition 4 (Max-complete) *An SPMN is max-complete iff all children of the same max node have the same scope, where the scope is as defined previously.*

Definition 5 (Max-unique) *An SPMN is max-unique iff each max node that corresponds to a decision variable D appears at most once in every path from root to leaves.*

An SPMN is solved by assigning values to the random variables that are consistent with the evidence. Then, we perform a bottom-up pass of the network during which operators at each node are applied to the values of the children. The optimal decision rule is found by tracing back (i.e., top-down) through the network and choosing the edges that maximize the decision nodes.

Recurrent SPMNs

Popular frameworks such as a Markov decision process (MDP) and languages such as dynamic influence diagrams (Shachter and Bhattacharjya 2010) model planning as a temporal sequence of decision-making steps. For our purposes, each of these steps can be modeled using a structure analogous to an SPMN. This yields a structure that is similar to a dynamic influence diagram, which unfolds an influence diagram with temporal links as many times as the number of steps in the extended problem thereby generating a larger influence diagram that models the complete sequence.

We take this perspective to modeling DTP, which involves sequential decision making and introduce a *recurrent SPMN* (RSPMN), which unfolds a *template network* as many times as the number of time steps in each sequence of data. While the template network is not rooted at a single node and is not a valid SPMN, we obtain these by learning an additional component: a top network that caps the unfolded templates, which, in conjunction with some properties on the structure of the template, then yields a valid SPMN.

An alternative approach to the RSPMN is to directly learn the SPMN from the sequence data using the LearnSPMN algorithm (Melibari, Poupart, and Doshi 2016). But, this poses two main challenges. First, an increase in the sequence length often leads to an exponential blow up of the size of the network and subsequently in evaluation time as we demonstrate later in our experiments. Second, the LearnSPMN algorithm requires a fixed number of variables in each data record. Hence, it may not be used when the sequence length varies between records as there may not always be an efficient way to either fill in the missing time steps for shorter sequences or eliminate extra sequences from the longer ones.

We begin by describing which domain attributes should be present in the data to allow the learning of RSPMNs followed by the components of the RSPMN.

Data Schema

Useful data for learning RSPMNs consists of a finite temporal sequence of values of state and utility variables, and decisions that are actions. More formally, consider a planning problem where the (fully observed) state of the environment is characterized by n variables, X_1, X_2, ..., X_n; actions by a combination of m decision variables, D_1, D_2, ..., D_m; and a single utility variable U. A candidate data record of at most T steps is then a sequence of T tuples of the form $\langle (I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^0$, $(I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^1, \ldots, (I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^{T-1} \rangle$. Recall from the previous section, $I_0, I_1, \ldots, I_m$ are information sets where I_{i-1}, $1 \leq i \leq m$ consists of values of the state variables in the information set of D_i. Additionally, u in each tuple is the value of utility variable U given the realizations of the state variables and decisions in that tuple.

RSPMN Properties and Validity

An RSPMN models sequences of decision-making data of varying lengths using a fixed set of parameters by unfolding a template network. In the context of a dynamic influence diagram, our template corresponds to an influence diagram with temporal links between nodes that are repeated in each time slice.

Definition 6 (Template network) *A template network is a DAG r root nodes and at least $n+1$ leaf nodes where n is the number of state variables and there is one utility function. The root nodes form a set of interface nodes Ir. The leaf nodes in the network hold the distributions over the random state variables $X_1, X_2, \ldots, X_n$, hold constant values as utility nodes, or are latent interface nodes. The root interface nodes and interior nodes can be sum, product, or max nodes. Let L denote the set of leaf latent interface nodes. Each latent node in L is related to a root interface node in Ir of the template network through a mapping $f : L \rightarrow Ir$.*

The f-mappings can be seen as time delay edges that link and let replace latent interface nodes at time step t with root interface nodes at $t+1$, thereby enabling recurrence of the template. The leaf latent node assumes a scope after unrolling the template network. The scope of a latent node at time step t is equal to the scope of a root interface node of the template at time step $t+1$, $scope(l_i^t) = scope(ir_i^{t+1})$. More formally, for any pair of latent nodes $l_i^t, l_j^t \in L$, let $f(l_i^t) = ir_i^{t+1}, f(l_j^t) = ir_j^{t+1}$, where $ir_i^{t+1}, ir_j^{t+1} \in Ir$, then $\left(scope(ir_i^{t+1}) = scope(ir_j^{t+1})\right) \Rightarrow \left(scope(l_i^t) = scope(l_j^t)\right)$.

Intuitively, the leaf latent interface nodes can be viewed as summarizing the latent information coming from the next template network. They pass information between templates of different steps. In other words, they pass up the information in a bottom-up evaluation of the RSPMN and pass down the information in a top-down pass. As such, the root and leaf latent nodes play a key role in linking the template networks during unfolding. To ensure that the unfolded network is a valid SPMN with a single root, we define another special network.

Definition 7 (Top network) *A top network is a rooted DAG consisting of sum and product nodes, and whose leaves are the latent interface nodes. Edges from a sum node are weighted as in a SPN.*

Of course, the bottom-most template network – corresponding to the final time step T of the planning – has its leaf interface nodes removed. Sum or product parents of these interface nodes with no other children are also pruned. We may effectively achieve this by setting the values of all these interface nodes as 1 (thus summing them out) and any utility values to pass set to 0.

Next, we seek to ensure that the SPMN formed after interfacing the top network and repeated templates is valid. One way to check for validity is to ensure that all the sum nodes in the unfolded SPMN are complete, the product nodes are decomposable, and max nodes are complete and unique as defined previously. However, can we define constraints on the top and template networks that will ensure validity of the unfolded SPMN? If so, we may establish the validity without checking the full network, which may grow to be quite large. To establish this, we first introduce a *sound* template.

Definition 8 (Soundness of the template) *A template network is sound iff all sum nodes in the template are sum-complete as defined in Def. 2; all product nodes in the template are decomposable as defined in Def. 3; all max nodes in the template are max-unique and max-complete as defined in Defs. 4 and 5; the scope of all the root interface nodes in Ir is the same, i.e., $scope(ir_i) = scope(ir_j) \quad \forall\, ir_i, ir_j \in Ir$; and, the scopes of the leaf latent interface nodes in L are related to that of the mapped root interface nodes in Ir,*

$$(scope(ir_i) = scope(ir_j)) \Rightarrow (scope(l_i) = scope(l_j)) .$$

Theorem 1 establishes that a sound template network combined with a valid top network is sufficient to generate a valid SPMN on unfolding the RSPMN. We provide the inductive proof in the technical Appendix included in the supplement.

Theorem 1 (Validity of RSPMN) *If, (a) in the top network, all sum nodes are complete and product nodes are decomposable, i.e., top network is valid, and (b) the template network is sound as defined in Def. 8, then the SPMN formed by interfacing the top network and the template network unfolded an arbitrary number of times as needed is valid.*

Learning of RSPMNs

We present a general algorithm for learning the structures and parameters of a valid top network and a sound template network from data whose schema was outlined in the previous section. Each data record of length T is a capture of an episode during which a decision-maker interacts with the environment for T steps (observing the state, acting, and obtaining reward). Let there be E such episodes. For convenience, we denote a tuple $(I_0, d_1, I_1, d_2, \ldots, I_{m-1}, d_m, I_m, u)^t$ as τ^t. Consequently, the data set has E records each consisting of T tuples $\langle \tau^0, \tau^1, \ldots, \tau^{T-1} \rangle_e$ where $e = 1, 2, \ldots, E$. Let us also note that all variables related to time step $t+1$ assume

Algorithm 1: LEARNRSPMN

Input : Dataset $\langle \tau^0, \tau^1, \ldots, \tau^{T-1} \rangle_e$ where $e = 1, 2, \ldots, E$; Partial Order $P^{\prec}$
Output : top network, template network

1 $\mathcal{S}^{t=2} \leftarrow$ Run LEARNSPMN on wrapped 2-time step data $\langle \tau^{t'}, \tau^{t'+1} \rangle_e \;\; e = 1, 2, \ldots, E; t' = 0, \ldots, T-2$
2 Create top network and set of root interface nodes Ir from $\mathcal{S}^{t=2}$
3 Create initial template network from Ir and $\mathcal{S}^{t=2}$
4 Revise initial template using sequence data to obtain the final template

their values after time step t. This is reflected in the *expanded* $P^{\prec}$, which now specifies the partial order not only among variables of a single time step but also includes variables of the next time step. This is sufficient because the partial order among variables of two consecutive steps does not change over time.

Algorithm 1 LEARNRSPMN presents the four main steps in learning the RSPMN from data. We describe and illustrate each of these steps below in more detail while deferring the pseudocode to the supplementary material. We also illustrate their applications on a simple 2×2 grid problem.

Each cell in the grid is represented by two binary state variables X, Y, which represent the x, y coordinates of the cell, respectively. Here, the top-left cell is $(0, 0)$ and $(1, 1)$ indexes the bottom-right cell. The agent can decide to either move in one of the four cardinal directions or perform a No-op, which is represented using a single decision variable, A. Let $(0, 0)$ be the start state, $(1, 0)$ a penalizing state with a reward of -10, and $(1, 1)$ the goal state with a reward of 10. All transitions are deterministic and cost -1. Reward is represented by the utility variable U. We simulated a random agent for $T = 4$ to generate sequence decision-making data of 10K records.

Step 1(Alg. 2 in the appendix). Learn an SPMN from 2-time step data Planning environments can often be modeled as Markovian. Thus, state transition probabilities and utility functions can be sufficiently learned from data spanning two time steps. Consequently, the first step of LEARNRSPMN is to use Melibari et al.'s LearnSPMN algorithm (2016) to learn a valid SPMN, $\mathcal{S}^{t=2}$, from 2-time step data. Subsequently, $\mathcal{S}^{t=2}$ serves as a basis for obtaining the template network.

However, which two time steps of a data record should we utilize? One may think that it would be sufficient to limit to tuples of the first two steps $\langle \tau^0, \tau^1 \rangle$ in each data record, or to tuples of any two consecutive steps. But, an agent often starts the episode at the same start state and is often at the same intermediate state in a subsequent time step. As such, data in the first two time steps, or for that matter, any fixed pair of time steps, is seldom fully representative of the transition probabilities and the dynamic situation. Hence, we consider each consecutive pair of tuples $\langle \tau^{t'}, \tau^{t'+1} \rangle \;\; t' = 0, 1, \ldots, T-2$ in each episode and wrap it to create a data set with $T \times E$ rows spanning two time steps.

As the expanded $P^{\prec}$ represents the partial order among the variables of two consecutive time steps, it is sufficient for use in LearnSPMN. Then, LearnSPMN is run over the wrapped data set with $P^{\prec}$ as the partial order. Figure 1 shows a learned 2-time step SPMN from the grid data.

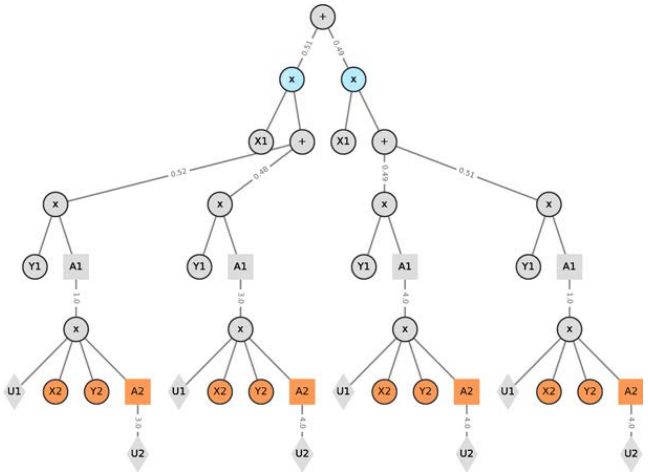

Figure 1: SPMN $\mathcal{S}^{t=2}$ learned on wrapped 2-time step data for the example grid problem. Xt, Yt, At, and Ut represent the corresponding variables for step t, where $t \in \{1, 2\}$. A single choice is shown from each decision node for clarity. (If not, each decision node A in the illustration would contain 5 branches each for one direction and No-op.)

Step 2(Alg. 3 in the appendix). Obtain top network and Ir nodes To obtain the nodes in Ir, we extract a 1-time step network $\mathcal{S}^{t=1}$ from $\mathcal{S}^{t=2}$. We point out that it is necessary to use a 2-time step SPMN to obtain $\mathcal{S}^{t=1}$. To realize this, let a state-action pair $\langle s_0, a_0 \rangle$ transition to state s_1 while $\langle s_2, a_0 \rangle$ transition to s_3. If $\mathcal{S}^{t=1}$ is learned from data of a single time step, the correlations between variables of different steps are obviously not ascertained. Due to this, both state values s_0 and s_2 may get included in a single substructure. However, the 2-time step data makes it clear that they effect differing transitions. This would be identified during independence testing, thereby modeling them separately by creating different clusters for each of these states in the first step as they result in a transition to a different next state. Importantly, this helps identify the behaviorally distinct states of the domain and helps create an interface node for each of them. Although the 2-time step SPMN structure can capture consecutive transitions, it is not desirable to use as a template. Intuitively, this is because 2-time step structure would enable us to unfold in even numbers $2, 4, 6, \ldots$ of time steps and not an arbitrary number of time steps. Moreover, the 1-time step structure is sufficient to model a template whereas a 2-time step structure would unnecessarily result in more nodes.

To generally obtain $\mathcal{S}^{t=1}$, we simply remove all those sum nodes whose immediate children have scopes that consist of subset of variables in the next time step $(I_0, D_1, I_1, D_2, \ldots, I_{m-1}, D_m, I_m, U)$[1]. If there are no such sum nodes, but instead variables of the next time step are directly linked to product nodes (as in Fig. 1 where the orange

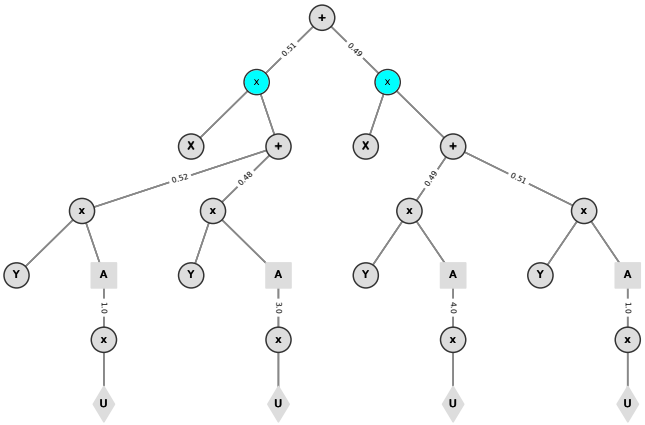

Figure 2: $\mathcal{S}^{t=1}$ obtained from the $\mathcal{S}^{t=2}$ of Fig. 1 with the orange nodes removed.

nodes are children of the product nodes), then we remove the nodes corresponding to these children. This results in $\mathcal{S}^{t=1}$ with no nodes whose scopes include variables of the next time step – yet appropriately modeling its impact on the first time step. Figure 2 illustrates $\mathcal{S}^{t=1}$ for the grid problem.

Now, we obtain the nodes in Ir using $\mathcal{S}^{t=1}$ albeit limiting the interface nodes to be products. We start by using breadth-first search to identify the top-most product nodes whose scopes equal the set of all variables of the first-time step $(I_0, D_1, I_1, D_2, \ldots, I_{m-1}, D_m, I_m, U)^0$. Starting from such top-most product nodes (the two blue product nodes in Fig. 2), the root interface nodes are obtained by identifying all the distinct state distributions from these product nodes. Identifying such distinct state distributions is particularly useful because having such states and their corresponding structures, we could use them to model the transitions from one state to the next. Ir is obtained from $\mathcal{S}^{t=1}$ by recursively traversing all the branches of the top-most product node until we find a product node without any sum node as a child. We will always find such nodes as these are product nodes whose children are either leaf distributions or max nodes. This corresponds to the four product nodes in Fig. 2 below the top two blue colored nodes. Each of these product nodes is returned as a set containing itself, and the leaf nodes of all the parent product nodes on path to this product node are added to this set. Each such set corresponds to a distinct distribution of I_0 and corresponding transitions and distributions in $(D_1, I_1, D_2, \ldots, I_{m-1}, D_m, I_m, U)$. A product node is created for *each* of these sets and the elements of the set are added as children of the product node as shown in Fig. 3. Each of these product nodes is a root interface node. Each of these interface nodes holds an SPMN that corresponds to a state distribution (there are four interface nodes for the four states in illustration). The interface nodes, for example, can help learn the probability of transitioning from one state s^t on taking a^t to some other state s^{t+1} in the next time step. *Observe that union of the scopes of the children of each interface node in Ir is identical. This makes the scopes of all the interface nodes Ir identical, which is needed to learn a sound template.*

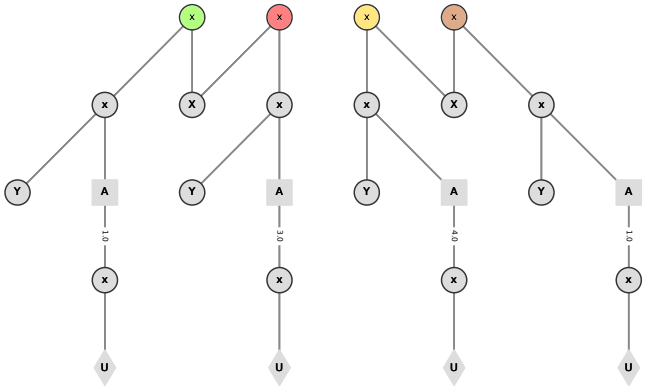

Figure 3: Illustrations of the root interface nodes obtained using the network in Fig. 2. The four colored product nodes function as the root interface nodes.

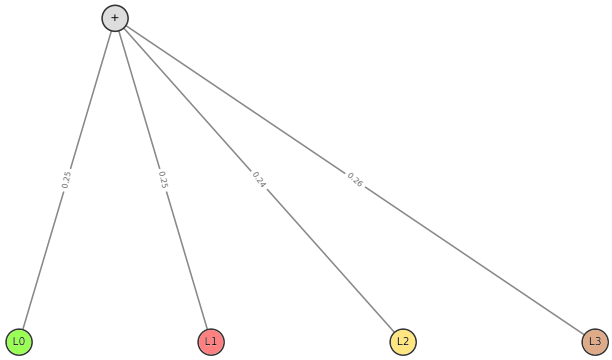

Figure 4: The top network with color-coded latent interface nodes indicating a bijective relationship with the similarly color-coded nodes in Ir.

The **top network** is then simply a sum node with as many children as the nodes in Ir. Each of these children is a latent interface node with a one-one relationship to a root interface node. The weights on the edges from the sum node are equal and correspond to a uniform distribution; these are adapted in a subsequent step. Alternately, the weights on these edges can be obtained by multiplying the weights on sum nodes through each of the paths that resulted in the specific ir node. We show the top network in Fig. 4.

Step 3(Alg. 5 in the appendix). Building an initial template Let $|Ir|$ denote the number of root interface nodes created in the previous step. We begin by creating a subnetwork S_L rooted at a sum node with as many children as $|Ir|$. The weights on the edges are equal and correspond to a uniform distribution. Each of the children is a leaf *latent interface node* observing the following relationship, $f(l_i) = ir_i$, $i = 1, 2, \ldots, |Ir|$, and f is a bijective map. As such, each latent interface node corresponds to a distinct root interface node. For our grid example, S_L includes a sum node with four children.

The network from Step 2 containing the root interface nodes forms a basis for creating the initial template. Begin-

ning at *each* root node in Ir, we traverse the graph to the *bottom-most* sum, product, or max node. In case of a product node, we add a new edge and link it to a new subnetwork S_L. In case of a sum or max node, each of its children nodes is now replaced by a product node with two outgoing edges. One of these edges links to the previous child node while the other edge links to S_L. Including all latent nodes in S_L can be effectively thought of as having observed a state and taken a decision at time step t results in reaching all the other states in the next step $t+1$ with equal probability.

As each latent interface node is related to a root interface node through the bijective mapping and the root interface nodes have identical scopes, the sum node of each S_L is complete. As S_L is added beside every leaf node, it does not impact the properties of other nodes. Therefore, the initial template network is sound.

Step 4(Alg. 6 in the appendix). Learning the final template network Parameters of the template network are the edge weights of the outgoing edges from the sum nodes including S_L. We adapt the hard expectation-maximization for SPNs (Poon and Domingos 2011; Peharz 2015) to the recurrent structure of the template to update the structure and parameters of the initial template. Broadly, it involves performing a bottom-up pass during which the likelihoods of each sum, product, and max node are calculated using a data record $\langle \tau^0, \tau^1, \ldots, \tau^{T-1} \rangle$. Then, a top-down (backpropagation) pass beginning at the rooted top network, which selects a maximum likelihood path and updates *integer counts* on the edges from sum nodes along that path.

Data from the last tuple τ^{T-1} is entered in the leaf random variable nodes of the bottom-most template (recall that the bottom-most template network has its leaf latent nodes removed). Likelihoods are propagated upwards through the network by performing the sum, product, and max operations represented by the nodes until we obtain a likelihood for each (product) node in Ir^{T-1}. Using the bijection function that relates the nodes in L with the nodes in Ir, we may propagate the likelihoods of the nodes in Ir^{T-1} to the corresponding latent nodes in L^{T-2}. The previously mentioned bottom-up pass is repeated using the likelihoods of the leaf latent nodes and data from tuple τ^{T-2} entered into the leaf random variable nodes of time step $T-2$ template, thereby yielding another set of likelihoods for the nodes. Regressing in data to time step 0, the bottom-up pass continues assigning a likelihood to each node in the template terminating when the root node of the top network is reached.

Given the likelihoods computed at each node for each time step, the top-down pass begins at the root node of the top network and at time step 0. It traverses downward visiting each node, selecting the child node with the highest likelihood for each sum node (including subnetwork S_L) and updating the count (initialized at zero) on the edge connecting the sum node to the child, selecting the child with the highest likelihood for the max node, and following each edge of the product node. The bijection mapping is used to go from the leaf latent nodes with maximum likelihood in time step $t-1$ to the mapped root interface nodes of time step t. An edge from a sum node chosen again has its previous count incremented by 1. The top-down pass terminates at the bottom-most network representing time step $T-1$.

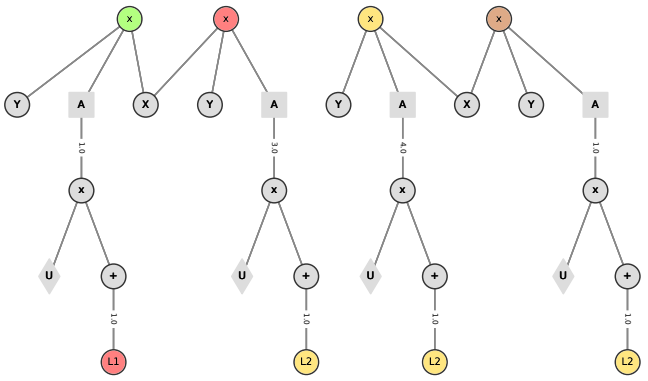

Figure 5: The final template network from the bottom and top-down passes for the grid problem. Notice that we retain the leaf latent nodes at each S_L from the initial template with the maximum likelihoods only.

New weights of outgoing edges from each sum node are obtained as: $\frac{\text{count on edge from sum} \rightarrow \text{child node}}{\text{\# sum node visited}}$. *Thereafter, any leaf latent nodes and indeed any children of a sum node with zero counts are pruned.* We may perform both the bottom-up and top-down passes without actually unfolding the template network by following the implicit links represented by the bijection mapping. Applying this step, the learned final template for our illustrative grid problem is shown in Fig. 5. As we prune just the latent interface nodes, the scope of S_L does not change and the template network remains sound.

MEU and Policy Evaluation

We may evaluate the RSPMN formed by interfacing the top network with the learned final template iterated as many times as the length of each data record or as desired to compute the MEU for each state and obtain a policy.

The MEU is obtained by evaluating the template network bottom-up as in an SPMN and passing the values between the templates of different time steps through the latent leaf nodes. The value at the root node of the top network gives the final desired value. The utility values of the leaf latent interface nodes of the bottom-most template (last time step) are set to zero. After evaluating the bottom-most template network, each root interface node of the template network holds a utility value. In the next iteration, the utility values of the leaf latent interface nodes are assigned the utility values of the mapped root nodes computed in the previous iteration, and the process is repeated until time step 0 and the top network is evaluated. Assuming that the template network learned a model of the true transitions well, each bottom-up pass through the template can be thought of as performing one Bellman update in the value iteration technique. This can be run until the desired length of the sequential problem is reached. Subsequently, each node of the template holds a corresponding expected utility value.

To additionally get the best action given an observation, the template network is interfaced with the top network and

Data set	X, D	#Episodes	T	#Cols	|SPMN|	|RSPMN|
GridUniverse[1]	1, 1	100K	8	24	138,492	(13, 210)
FrozenLake[1]	1, 1	100K	8	24	1,068,246	(18, 401)
Maze[1]	2, 1	100K	8	24	352,312	(11, 184)
Taxi[1]	4, 1	20K	50	150	–	(80, 1815)
SkillTeaching[2]	12, 4	100K	10	170	–	(137, 4878)
Elevators[2]	13, 4	200K	10	180	–	(143, 5390)
CrossingTraffic[2]	18, 4	100K	15	345	–	(82, 2349)

Table 1: Superscript 1 denotes simulations of Gym domains and [2] denotes simulations of RDDLSim (Sanner 2010) MDP domains. **X, D** give the numbers of state and decision variables in the domain, #Cols is the total number of columns in each data record. We also report the size of the learned structures. |RSPMN| gives the sizes of the (top, template) networks respectively. ' – ' denotes that the SPMN was not learned in 12 hrs on an Ubuntu Intel i7 64GB RAM PC.

variables are assigned values corresponding to the observed state. A top-down pass starting at the root node of the top network and choosing the decision with the MEU at each max node reached (as in an SPMN) yields the best action(s) for that state.

Experiments

We implemented LEARNRSPMN in the open-source SPFlow library (Molina et al. 2019) and evaluated it on a new testbed of sequential decision-making data sets that adhere to the data schema discussed previously. Our code is available on GitHub at https://github.com/c0derzer0/RSPMN.

Evaluation testbed As there are few existing data sets on simulations of decision-making domains, we created a new testbed listed in Table 1 and available for public use from the GitHub repository. Each data set is generated by using a random policy which interacts with the environment collecting the ⟨state, action, reward⟩ generated at each step. Each episode is run for T time steps, which is selected to be sufficient to reach the goal state. A new episode is started if either the last time step is reached or if the agent reaches the goal or another terminal state.

For each data set in the testbed, we learn an SPMN using the LearnSPMN algorithm. This was made possible by padding the sequences so that all sequences have the same length – on reaching a terminal state, the agent stayed in that state until the length of the sequence is T. The top and template networks of a RSPMN were learned using our LEARNRSPMN. We show the sizes of the learned networks as the total number of nodes in each, in the last two columns of Table 1. Notice the blow up in the sizes of the SPMNs learned for the sequential data sets. The SPMN has many repeated structures for the state distributions over time. For the larger RDDLSim domains, the sizes of the top and template networks also grow but we did not observe a disproportionate growth, while the SPMNs could not be learned.

MEU and policy comparisons Table 2, which reports the key results, compares the MEU from the start state of each domain as obtained by evaluating the learned RSPMNs and any learned SPMNs with the (near-)optimal values. We obtained the optimal values from converged deep Q-networks for the Gym domains and by solving the MDP using value iteration for the RDDLSim domains. Observe that RSPMNs yield MEUs that are very close to the optimal and significantly better than those from the learned SPMNs. Clearly, the sequential data sets do not have sufficient episodes to learn high-quality SPMNs.

RSPMNs and SPMNs also represent a model of the environment as present in the data, which then influences the MEU and the policy. *So, how well did the structure learning method capture the environment dynamics?* To answer this question, we simulated the policies obtained from the RSPMNs in their respective Gym environments and noted down the average rewards. Table 2 shows that these average rewards, recorded across 10K episodes, nearly match the MEUs obtained directly from the RSPMNs. This implies that the networks are modeling the environment accurately. We also compare with the policies learned by the discrete batch-constrained Q-learning (Fujimoto et al. 2019) with default parameters on the data sets as a baseline. This is a technique similar to deep Q-networks but constrained to learning from a data set. We report the average rewards obtained by simulating its learned policies on 10K episodes for all the domains. Clearly, Table 2 shows that BCQ expects far more data on several domains to learn a good policy.

Next, we report the deviation in policy learned by the RSPMN from the optimal one. This is the total number of states where the actions differ and reported as a percentage $\Delta\%$ of the total number of states. Notice the large deviations for FrozenLake, Taxi, and the RDDLSim domains although the learned policies show MEUs close to the optimal. This is likely due to the presence of multiple optimal policies in these large domains. For the sake of completeness, we also report the log likelihoods of the models in the last column.

Table 3 shows our final set of results on the clock time it takes for learning the initial template structure, learning the final template of the RSPMN, and learning the large SPMNs when possible. The time to learn an SPMN was capped at about 12 hrs. Observe that both learning and evaluating the large SPMNs takes a *few orders of magnitude* longer than learning the templates. However, the template learning times also increase for the larger RDDLSim domains with Elevators taking more than 4 hours. On the other hand, the MEU evaluation remains quick for all the domains taking less than a minute.

Related Work

SPNs are related to other graphical models for probabilistic inference such as arithmetic circuits (Park and Darwiche 2004) and AND/OR graphs (Dechter and Mateescu 2007). Bhattacharjya and Shachter (2007) proposed decision circuits as a representation that ensures exact evaluation and solution of influence diagrams in time linear in the size of the network. A decision circuit extends an arithmetic circuit with max nodes for optimized decision making, which is analogous to how SPMNs extend SPNs. However, decision circuits are obtained by compiling IDs. Previous work has shown that

Data set	MEU			Average reward		Δ %	LL (RSPMN)
	Optimal	RSPMN	SPMN	RSPMN	BCQ		
GridUniverse	6	6	6	5.9	5.9	0	-0.87
FrozenLake	0.8	0.818	0.13	0.8	0.3	62.5	-6.17
Maze	0.966	0.966	0.052	0.96	0.96	0	-0.86
Taxi	8.9	9	-	8.9	-200	60.25	-2.45
SkillTeaching	-3.022	-3.06	-	-3.009	-7.36	83.3	-2.09
Elevators	-7.33	-7.47	-	-7.357	-9.14	80	-4.8
CrossingTraffic	-4.428	-4.425	-	-4.427	-5.91	94.7	-8.44

Table 2: Our key results comparing the MEUs of the optimal policy, learned RSPMNs, SPMNs, and batch-constrained Q-learning. Δ % gives the policy deviations between the policies obtained from RSPMN and the optimal ones.

Data set	Template learning		SPMN learning	MEU eval	
	Initial	Final		RSPMN	SPMN
GridUniverse	2m 26.33s	1m 1.49s	4h 25m 31.72s	0.72s	8.8s
FrozenLake	1m 49.8s	2m 02.78s	12h 5m 40.77s	23.21s	1m 26.85s
Maze	2m 51.19s	54.84s	2h 31m 49.51s	0.62s	24.87s
Taxi	9m 21.79s	2h 28m 15.75s	–	18.45s	–
SkillTeaching	59m 5.87s	29m 28.49s	–	3.84s	–
Elevators	1h 19m 3.91s	4h 19m 29.53s	–	20s	–
CrossingTraffic	8m 46.14s	1h 37m 53.17s	–	18.45s	–

Table 3: Initial and final template learning times are shown in first two columns. These are significantly less than those learning the large SPMNs. Run times of MEU evaluations on the learned SPMNs and RSPMNs are shown next.

SPMNs are efficiently reducible to decision circuits in time that is linear in the size of the SPMN (Melibari, Poupart, and Doshi 2016). However, no dynamic extension of decision circuits has been presented nor any algorithms to learn decision circuits directly from data.

Attias (2003) posed planning using MDPs as a likelihood maximization problem where the "data" is the initial state and the final goal state or the maximum total reward. Toussaint et al. (2006) extended this to infer finite-state automata for infinite-horizon partially-observable MDPs. A most likely controller is inferred using expectation-maximization on a mixture of increasing-horizon DBNs. This approach is appealing because it allows the advances in inference to bear on planning. It differs from SPMNs by requiring a specification of the planning domain; these form the required parameters of the DBN. Furthermore, inference in DBNs is essentially intractable although this may be speeded up using some of the network compilation models. In contrast, SPMNs and RSPMNs offer a tractable approach to DTP that maximizes expected utility and is fundamentally different from planning as inference.

Offline (batch) learning seeks to derive an optimal policy from a given set of prior experiences. This set is analogous to our simulations, and may either be fixed or allowed to grow. While offline RL is not as well studied as its online counterpart, the general approach is to modify online techniques for use in batch contexts. Prominent methods, such as experience replay (Lin 1992) and fitted Q-iteration (Ernst, Geurts, and Wehenkel 2005), are model-free and utilize the Q-update rule synchronously over all data until convergence. Recently, methods based on deep neural networks such as BCQ (Fujimoto et al. 2019) have appeared.

Concluding Remarks

RSPMNs offer a model-based approach to DTP, learning the policy directly from data, which is provably tractable in the size of the network. We introduced RSPMNs and presented the first method to learn the structure and parameters of the model. We established RSPMN's favorable performance in comparison to batch RL. RSPMNs are also useful for off-policy evaluations (Gottesman et al. 2019) where the environment model is learned from the data, which is then used for policy evaluations.

An important future extension is to explore anytime techniques for learning compact networks where the number of nodes may grow with additional resources. Another extension would be to adapt online structure learning (Kalra et al. 2018) for RSPMNs. This can then be used to dynamically update the model while the agent is interacting with environment.

Acknowledgements

This research is supported in part by NSF grant #1815598 to PD. We thank Alejandro Molina for help with the open-source SPFlow codebase for SPNs. We also thank Pascal Poupart for discussions that helped shape the direction of this research.

References

Adel, T.; Balduzzi, D.; and Ghodsi, A. 2015. Learning the Structure of Sum-Product Networks via an SVD-based Algorithm. In *Conference on Uncertainty in AI (UAI)*.

Attias, H. 2003. Planning by Probabilistic Inference. In *Ninth International Workshop on AI and Statistics (AISTATS)*.

Bhattacharjya, D.; and Shachter, R. D. 2007. Evaluating influence diagrams with decision circuits. In *Proceedings of the conference on Uncertainty in artificial intelligence*, 9–16.

Darwiche, A. 2000. A Differential Approach to Inference in Bayesian Networks. In *UAI*, 123–132.

Dechter, R.; and Mateescu, R. 2007. AND/OR search spaces for graphical models. *Artificial intelligence* 171(2-3): 73–106.

Ernst, D.; Geurts, P.; and Wehenkel, L. 2005. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research* 6(Apr): 503–556.

Fujimoto, S.; Conti, E.; Ghavamzadeh, M.; and Pineau, J. 2019. Benchmarking Batch Deep Reinforcement Learning Algorithms. *arXiv preprint arXiv:1910.01708* .

Gens, R.; and Domingos, P. 2013. Learning the structure of sum-product networks. In *Proceedings of The 30th International Conference on Machine Learning*, 873–880.

Gottesman, O.; Liu, Y.; Sussex, S.; Brunskill, E.; and Doshi-Velez, F. 2019. Combining parametric and nonparametric models for off-policy evaluation. In *International Conference on Machine Learning*, 2366–2375.

Huang, J.; Chavira, M.; and Darwiche, A. 2006. Solving MAP Exactly by Searching on Compiled Arithmetic Circuits. In *AAAI*, volume 6, 3–7.

Kalra, A.; Rashwan, A.; Hsu, W.; Poupart, P.; Doshi, P.; and Trimponias, G. 2018. Online structure learning for feed-forward and recurrent sum-product networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 6944–6954.

Koller, D.; and Friedman, N. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.

Lin, L.-J. 1992. Self-Improving Reactive Agents based on Reinforcement Learning, Planning and Teaching. *Machine Learning* 8(293–321).

Lowd, D.; and Rooshenas, A. 2013. Learning Markov networks with arithmetic circuits. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, 406–414.

Melibari, M.; Poupart, P.; and Doshi, P. 2016. Sum-Product-Max Networks for Tractable Decision Making. In *Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI)*, 1846–1852.

Melibari, M.; Poupart, P.; Doshi, P.; and Trimponias, G. 2016. Dynamic Sum Product Networks for Tractable Inference on Sequence Data. In *International Biennial Conference on Probabilistic Graphical Models (PGM), JMLR: Workshop & Conference Proceedings, Vol 52*, 345–355.

Molina, A.; Vergari, A.; Stelzner, K.; Peharz, R.; Subramani, P.; Mauro, N. D.; Poupart, P.; and Kersting, K. 2019. SPFlow: An Easy and Extensible Library for Deep Probabilistic Learning using Sum-Product Networks.

Park, J. D.; and Darwiche, A. 2004. A differential semantics for jointree algorithms. *Artificial Intelligence* 156(2): 197–216.

Peharz, R. 2015. *Foundations of Sum-Product Networks for Probabilistic Modeling*. Ph.D. thesis, Aalborg University.

Poon, H.; and Domingos, P. 2011. Sum-product networks: A new deep architecture. In *12th Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2551–2558.

Sanner, S. 2010. Relational Dynamic Influence Diagram Language (RDDL): Language Description. Specification in IPPC 2011.

Shachter, R.; and Bhattacharjya, D. 2010. Dynamic programming in infuence diagrams with decision circuits. In *Twenty-Sixth Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 509–516.

Toussaint, M.; and Storkey, A. J. 2006. Probabilistic inference for solving discrete and continuous state Markov Decision Processes. In *International Conference on Machine Learning (ICML)*, 945–952.