# Training Deep Reactive Policies for Probabilistic Planning Problems

**Murugeswari Issakkimuthu, Alan Fern, Prasad Tadepalli**
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331, USA

**Abstract**

State-of-the-art probabilistic planners typically apply look-ahead search and reasoning at each step to make a decision. While this approach can enable high-quality decisions, it can be computationally expensive for problems that require fast decision making. In this paper, we investigate the potential for deep learning to replace search by fast reactive policies. We focus on supervised learning of deep reactive policies for probabilistic planning problems described in RDDL. A key challenge is to explore the large design space of network architectures and training methods, which was critical to prior deep learning successes. We investigate a number of choices in this space and conduct experiments across a set of benchmark problems. Our results show that effective deep reactive policies can be learned for many benchmark problems and that leveraging the planning problem description to define the network structure can be beneficial.

## Introduction

Many real-world planning problems involve large factored state spaces with highly stochastic exogenous and endogenous dynamics. The Relational Dynamic Influence Diagram Language (RDDL) was designed to model such problems by compactly defining large Dynamic Bayesian Networks (DBNs) over state and action variables. Current state-of-the-art planners for RDDL problems are based on online search, where at each step some combination of search and reasoning is used to select an action. For example, there are planners based on sample-based tree search (Keller and Eyerich 2012; Kolobov et al. 2012; Bonet and Geffner 2012), symbolic variants (Cui et al. 2015; Raghavan et al. 2015; Anand et al. 2016), and those that construct and solve integer linear programs at each step (Issakkimuthu et al. 2015). These planners can require non-trivial computation time per step, which can make them inapplicable to problems that require fast decisions.

One approach to support fast decisions is via reactive policies that can be applied online to quickly select actions. Offline Symbolic Dynamic Programming (SDP) has recently been explored for producing such policies for RDDL problems (Raghavan et al. 2012; 2013). SDP (Hoey et al. 1999) uses symbolic operations to produce a symbolic policy representation that can be efficiently evaluated at any state. Unfortunately, while there have been significant advances, scalability is still an issue with SDP. Reactive policies can also be produced via supervised learning or reinforcement learning. Most recently, state-of-the-art results have been achieved in a variety of domains by learning deep neural networks (DNNs) to represent reactive policies. Examples include learning to play Atari games directly from pixel input (Mnih et al. 2015), robotic control (e.g. (Levine et al. 2016)), and the game of Go (Silver et al. 2016; 2017). These results motivate the investigation of learning such Deep Reactive Policies (DRPs) for planning problems described in RDDL. We note that the impressive successes of DRPs are not due to the blind application of off-the-shelf tools and DNN architectures. Rather, the successes were enabled by significant expertise and manual exploration of architectures and training methods. The objective of this paper is to present an initial exploration of the DRP design space for RDDL benchmark problems via an extensive empirical investigation covering five domains with some theoretical guarantees about the expressiveness of the architectures.

We describe three classes of architectures that support problem-specific DRPs by leveraging the RDDL problem definition. We train our DRPs to imitate the action choices of more expensive non-reactive planners by supervised learning. We consider two different choices for generating data and two different ways to optimize DRPs based on the data. Our experiments shed light on the following questions. Can we learn DRPs that are competitive with the planners that they are learned from? Can the RDDL problem definition be used to define more effective network architectures? Are there any consistently superior DRP architecture choices across RDDL problems? Are some supervised training signals and loss functions more effective in general than others? We note that this study is focused on learning DRPs for individual planning problems using supervised learning. It is an interesting future direction to consider learning DRPs that generalize across problems within an entire planning domain. However, such a step requires additional architectural considerations, which we believe should be informed by the study of individual problems. We also note that other training mechanisms such as reinforcement learning will be interesting to consider in future work.

Copyright © 2018, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

## Related Work

There is a long history of work on integrating machine learning and automated planning (Minton 1993; Zimmerman and Kambhampati 2003; Jiménez et al. 2012). Much work focuses on learning control knowledge (heuristics and pruning rules) to speed up a planner and/or improve the plan quality. While these approaches have shown promise, they are not guaranteed to reduce planning times and can even result in net slow down of a planner (Minton et al. 1989). An important exception is the prior work on learning reactive policies in the form of relational rule lists for deterministic STRIPS domains (Khardon 1999). Extensions to the work include using richer rule representations (Martin and Geffner 2000; De La Rosa, Celorrio, and Borrajo 2008), iterative learning algorithms (Fern, Yoon, and Givan 2006), and application to probabilistic STRIPS (Yoon, Fern, and Givan 2002). While these approaches are promising in many domains, it has been a challenge to demonstrate their robustness across a wide range of domains. One difficulty is that the rule languages, once selected, are inflexible and can not always capture key concepts. This motivates investigating DNNs for planning, since, in principle, they can induce deep features and concepts as needed.

The most closely related prior work is the Factored Policy Gradient (FPG) planner (Buffet and Aberdeen 2009), which represents reactive policies using simple neural networks, most commonly a linear network per action for computing action probabilities. The network parameters are tuned using policy-gradient reinforcement learning where each learning episode begins from the starting state of the problem being considered. Promising results were demonstrated for a number of planning domains including probabilistic PDDL (PPDDL) benchmark problems. Interestingly, for most problems there was no perceived benefit to using multi-layer networks over simple linear networks. Our experiments also show that for some RDDL benchmarks linear networks are as good as or better than more complex networks. Most recently, concurrent work (Toyer et al. 2018) is the first to learn deep networks for relational generalization across problems of PPDDL planning domains. PPDDL and RDDL are qualitatively different languages, however, which makes it difficult to apply that approach directly to many RDDL domains.

## RDDL Planning Problems

We assume familiarity with the basic framework of Markov Decision Processes (MDPs). A factored MDP describes the state space by a finite set of binary variables $(x_1, x_2, \dots, x_n)$ and the action space by a finite set of binary variables $(a_1, a_2, \dots, a_m)$. In this work, we focus on the case where actions are constrained to have exactly one of the action variables set at any time and view each $a_i$ as a distinct ground action. Most current RDDL benchmarks already have this constraint. The reward function $R$ specifies a mapping from the state and action variables to real-valued rewards. We assume that the transition function $T$ is compactly described as a DBN which specifies the probability distribution over each state variable $x_i$ in the next time step, denoted $x'_i$, given the values of a subset of the state and action variables $parents(x'_i)$ in the current time step, in particular, $T(s,a,s') = \prod_i Pr(x'_i|parents(x'_i))$. RDDL (Sanner 2010) is a high-level specification language for compactly representing such DBN domains in a relationally-factored form using parameterized state and action variables. Individual problem instances then specify a set of domain objects that instantiate the state and action variables. A policy $\pi$ is a mapping, possibly stochastic, from the state space to actions. We focus on optimizing the expected finite-horizon total reward of a policy.

## Architectures for Deep Reactive Policies

A Deep Reactive Policy (DRP) is a policy encoded as a deep neural network. DRPs are reactive in the sense that they can be quickly evaluated in a single feed-forward pass. Our DRP architectures are organized into $L+1$ layers of nodes. $Z^l = \{z^l_k\}$ denotes layer $l$, where $z^l_k$ is the $k$'th node in layer $l$ and its value is denoted by $o(z^l_k)$. The input layer $Z^0$ contains $n$ nodes, each taking the value of one state variable, i.e., $o(z^0_i) = x_i$. Layers 1 through $L-1$ are hidden layers each containing $C \times n$ nodes, where $C$ parameterizes the number of *channels* which allows for scaling the DRP size with the number of state variables. The output layer $Z^L = \{z^L_0, z^L_1, z^L_2, \dots, z^L_m\}$ contains $m+1$ nodes, where $z^L_1, \dots, z^L_m$ correspond to the $m$ RDDL actions and $z^L_0$ corresponds to the NOOP action. In all the architectures the final hidden layer $Z^{L-1}$ is fully connected to the output layer $Z^L$.

For **single-channel** networks ($C = 1$) we have $Z^l = \{z^l_1, z^l_2, ..., z^l_n\}$ for all the hidden layers. Each hidden node $z^l_k$ is connected to a set of nodes $(I^l_k)$ in the previous layer $Z^{l-1}$ via real-valued weights, where $w^l_{i,k}$ is the weight from $z^{l-1}_i$ to $z^l_k$ and $b^l_k$ is the bias parameter to $z^l_k$. We use *ReLU* activation functions as the non-linearity for hidden nodes so that the value of $z^l_k$ is

$$o(z^l_k) = \text{ReLU}\left(\sum_{p \in I^l_k} o(z^{l-1}_p) w^l_{p,k} + b^l_k\right). \quad (1)$$

The output layer is a softmax layer computing a probability distribution over the $m+1$ actions

$$o(z^L_k) = \frac{exp(\sum_{p=1}^{|Z^{L-1}|} o(z^{L-1}_p) w^L_{p,k} + b^L_k)}{\sum_{j=0}^{m} exp(\sum_{p=1}^{|Z^{L-1}|} o(z^{L-1}_p) w^L_{p,j} + b^L_j)} \quad (2)$$

For **multi-channel** networks ($C > 1$) $Z^l = \{X^l_1, X^l_2, ..., X^l_n\}$ for all the hidden layers. The set of nodes in $Z^l$ is partitioned into $n$ subsets, where subset $X^l_k$ corresponds to $z^l_k$ in a single-channel network with $|X^l_k| = C$. If the hidden layers of single-channel networks are (column) vectors of size $n$ then those of multi-channel networks are matrices with $C$ such (column) vectors. The $k^{th}$ row of the matrix corresponds to nodes in $X^l_k$. All nodes in $X^l_k$ receive the same set of input connections from the previous layer though with different weights. If $z^{l-1}_p$ is connected to node $z^l_k$ in a single-channel network then the entire subset of nodes $X^{l-1}_p$ are connected to $z^l_k$ in a multi-channel network.

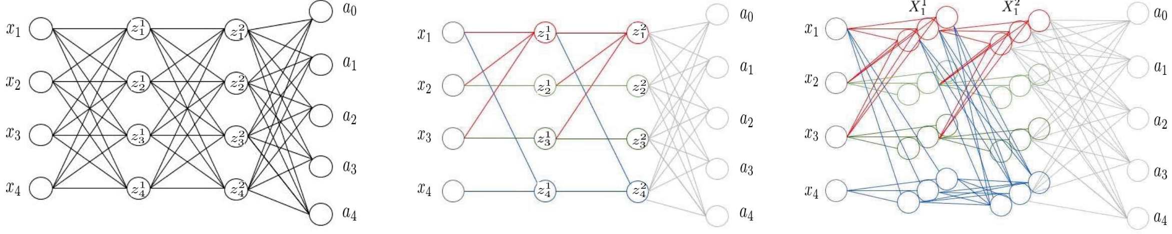


Figure 1: (Left to Right) A single-channel FC-DRP, single-channel S-DRP, partially illustrated multi-channel S-DRP

**Fully-Connected DRPs (FC-DRPs).** FC-DRPs are the traditional fully-connected networks in which each hidden node at layer $l$ is connected to each hidden node in layer $l-1$, i.e., for all $l > 1$ and $k$, $I_k^l = \{1, \ldots, |Z^{l-1}|\}$. We use $FC(L, C)$ to denote a FC-DRP architecture with layer and channel parameters $L$ and $C$.

**Sparse DRPs (S-DRPs).** The large number of FC-DRP parameters raises the potential for overfitting. CNNs reduce the number of parameters, while remaining expressive by attaching spatial semantics to hidden nodes and only allowing connections to spatially close nodes. In analogy, S-DRPs associate hidden nodes with state variables and only connect nodes whose variables have probabilistic dependencies. $S(L, C)$ denotes an S-DRP with the associated layer and channel parameters. The nodes in hidden layer $Z^l$ are partitioned into $n$ sets $X_1^l, X_2^l, \ldots, X_n^l$, each having $C$ hidden units. We interpret the nodes in $X_i^l$ as being associated with state variable $x_i$. A hidden node $z_k^l \in X_i^l$ is connected only to nodes in $Z^{l-1}$ that are associated with state variables that $x_i$ depends on in the transition function. In particular, $I_k^l = \bigcup_{j \in parents(x_i')} X_j^{l-1}$, where $parents(x_i')$ is the set of state variables in the current time step that can influence the transition probability of $x_i$. Thus, the S-DRP connectivity mirrors the DBN local dependency structure across time steps. Figure 1 illustrates an FC-DRP and S-DRP with single and multiple channels.

**Representation Capacity of S-DRPs.** The potential advantage of sparsity is better generalization, while the potential disadvantage is representation capacity. Consider an MDP with two binary state variables $x_1$ and $x_2$ with independent transition dynamics. Let policy $\pi(x_1, x_2) = XOR(x_1, x_2)$, which is not linearly representable. The hidden layers for any S-DRP will not be able to compute features that combine $x_1$ and $x_2$ and hence the final linear softmax layer will not be able to represent $\pi$. In general, when policies involve complex dependencies among state variables that have independent transition dynamics, S-DRPs may be inadequate. We provide an initial result that characterizes a class of policies that is S-DRP representable subject to MDP restrictions and also describe a small S-DRP modification that supports any policy.

The Q-function $Q^{\pi}(s, a)$ of $\pi$ gives the value of executing action $a$ from state $s$ and then following policy $\pi$. We say that a policy $\pi$ is *Q-representable* if there is a policy $\pi'$ such that $\pi(s) = \arg\max_a Q^{\pi'}(s, a)$ and that for each state the maximizing Q-value is unique. Examples of Q-representable policies include optimal policies that have unique optimal actions in each state, policies computed by the Rollout algorithm for any base rollout policy $\pi'$, or any policy from the standard policy iteration sequence. A reward function $R$ is said to be *independently additive* if $R(s) = \sum_i R_i(x_i)$. A transition function is *DBN-representable* if it has the form $T(s, a, s') = \prod_i Pr(x_i' | parents(x_i'))$.

**Theorem 1.** *For any MDP with independently additive rewards and DBN-representable transition function, if $\pi$ is Q-representable, then $\pi$ can be represented as a finite S-DRP.*

*Proof.* The proof uses the concept of Krylov basis for MDPs (Petrik 2007). Let $P$ be the transition probability matrix over ground MDP states for $\pi$ and $R$ be the reward vector over ground states. The Krylov basis function of order $t$ is given by $b^t = P^t R$. The component of vector $b^t$ for state $s$ gives the expected reward at step $t$ if $\pi$ is followed from $s$. For a finite $K$ the value function $V^{\pi}$ of $\pi$ can be represented as a linear combination of the basis functions $b^0, \ldots, b^K$ (Ipsen and Meyer 1998). It follows that for any policy $\pi'$ the basis can also linearly represent $Q^{\pi'}(s, a)$ for any fixed action $a$ as a function of $s$, in particular the $\pi'$ that shows $\pi$ is Q-representable. For any given $s$, the maximum over these functions yields $\pi(s)$.

To relate the Krylov basis to S-DRPs, it is possible to show that for DBN-representable transition functions and independently additive rewards, each Krylov basis function has the form $b^t(s) = \sum_i R_i(x_i) P\left(x_i^t | parents_i^t\right)$, where $x_i^t$ is the value of state variable $x_i$ after $t$ steps from the initial state being conditioned on $parents_i^t$, the set of state variables at the initial time step that influence $x_i^t$, i.e. the *t-step influencers* of $x_i$. Thus, the Krylov basis decomposes linearly into a set of functions that depend on the $t$-step influencers of each $x_i$. Now consider any S-DRP hidden node $z \in X_i^{L-1}$ in the last layer that is associated with $x_i$. It is easy to see that the output $o(z)$ is a function of only the input state variables that are $L$-step influencers of $x_i$ under any policy. Thus, each such $z$ can be viewed as computing a potentially complex non-linear feature of the $L$-step influencers of $x_i$. For large enough $L$ and $C$ this allows for the S-DRP to represent the above decomposition of the Krylov basis and applying a softmax layer will then return the actions of $\pi$. □

By changing the activation function of the output layer we

can represent any policy under mild conditions. In particular, an RBF S-DRP is a DRP where the softmax output layer is replaced by having a radial basis activation function (RBF) for each output node. The only constraint on the RBF is that it is maximized when the affine transformation of its input is zero (e.g. a Gaussian). An RBF S-DRP selects the action in the output layer with the highest node activation. In the following a policy $\pi$ is said to be Q-distinct if for any state $s$ and any $a \neq \pi(s)$, $Q^{\pi}(s, \pi(s)) \neq Q^{\pi}(s, a)$.

**Theorem 2.** *For any MDP with independently additive rewards and DBN-representable transition function, if $\pi$ is Q-distinct then $\pi$ can be represented via a finite RBF S-DRP.*

*Proof.* (Sketch) The proof is similar to the previous theorem. Since $V^{\pi}$ and $Q^{\pi}(s, a)$ for any fixed $a$ are linearly representable via Krylov basis functions, so is $V^{\pi} - Q^{\pi}(s, a)$ for any $a$. Since $\pi$ is Q-distinct, this expression is zero iff $\pi(s) = a$. This means that applying an RBF to the above difference for each $a$ will identify $\pi(s)$. The rest of the proof follows along the same lines as above. □

When the reward function is not independently additive, it may decompose into factors over groups of state variables. In such cases, we can get a similar result by extending the S-DRPs to include one or more fully connected layers between the last sparse hidden layer and the output layer.

**Relational Weight Sharing DRPs (R-DRPs).** An R-DRP is constructed by constraining all weights in the S-DRP that are relational matchings to have the same value. Intuitively, $(s, t)$ and $(u, v)$ are relational matchings when the probabilistic dependency between $s$ and $t$ is structurally similar to that from $u$ to $v$. Sharing is limited to weights between state-fluent nodes in adjacent layers. The bias parameters and the weights of the fully-connected layer at the end are not shared. Two connections are similar if the (start-node, end-node) pairs of the connections are similar. For example, in the blocksworld domain, the pairs (clear(A), on(A, B)) and (clear(C), on(C, D)) are semantically similar. Since nodes in the input and hidden layers of R-DRPs represent state fluents the (start-node, end-node) pairs $(z_u^{l-1}, z_v^l)$ are instantiated first-order predicates. Consider weights $w_{j,k}^l$ and $w_{u,v}^l$ between $(z_j^{l-1}, z_k^l)$ and $(z_u^{l-1}, z_v^l)$ respectively, where $(z_j^{l-1}, z_k^l) = (q_j(j_1, j_2, ...j_{n_j}), q_k(k_1, k_2, ...k_{n_k}))$, and $(z_u^{l-1}, z_v^l) = (q_u(u_1, u_2, ...u_{n_u}), q_v(v_1, v_2, ...v_{n_v}))$. Let $J = (j_1, j_2, ...j_{n_j}), K = (k_1, k_2, ...k_{n_k}), U = (u_1, u_2, ...u_{n_u})$, and $V = (v_1, v_2, ...v_{n_v})$. Weights $w_{j,k}^l$ and $w_{u,v}^l$ are constrained to be the same if (1) $q_j = q_u$ and $q_k = q_v$ and (2) $J \times K = U \times V$. $|J \times K| = n_j n_k$ and $J \times K$ is defined as the cross-product of ordered tuples $J$ and $K$ giving an ordered tuple of binary values. The $1^{st}$ $n_k$ entries of $J \times K$ are computed by comparing $j_1$ with each element of $(k_1, k_2, ..., k_{n_k})$. The $2^{nd}$ $n_k$ entries of $J \times K$ are computed by comparing $j_2$ with each element of $(k_1, k_2, ..., k_{n_k})$. The last $n_k$ entries of $J \times K$ are computed by comparing $j_{n_j}$ with each element of $(k_1, k_2, ..., k_{n_k})$. The components of both $J$ and $K$ are strings representing objects in the problem and when $j_1$ is compared to $k_1$ the result is 1 if the strings match and 0 otherwise. For example, if $z_j^{l-1} = clear(A)$ and $z_k^l = on(A, B)$ then $J \times K = (1, 0)$ because $A \neq B$ and the second component is 0.

## Supervised Training of DRPs

We use supervised learning to train DRPs for individual RDDL problems. This involves generating training data and optimizing the parameters of a chosen DRP architecture.

**Training Data Generation.** Following prior work (e.g. (Khardon 1999; Martin and Geffner 2000; Yoon, Fern, and Givan 2002)) we generate training data using *imitation learning*, which aims to learn a policy that imitates the actions of an expert. In our case, the expert is a non-reactive online planner that can select an action at any state. More precisely, given a planning problem with initial state $s_0$ and horizon $H$, we use the planner to generate multiple trajectories, each one starting in $s_0$ and then following a sequence of actions selected by the planner until the horizon. Each of the stochastic trajectories gives a sequence of state-action pairs $(s_0, a_0), (s_1, a_1), \ldots, (s_{H-1}, a_{H-1})$, which can be combined to create a standard supervised training set. A disadvantage of learning from just state-action pairs is that the learning algorithm is unable to make informed trade-offs when perfect accuracy is not possible. To address this, we can augment the training examples with Q-value estimates for each action when available from the planner. Here the Q-value of a state action pair $Q(s, a)$ is the expected finite horizon reward of starting in state $s$, taking action $a$, and then acting optimally thereafter. This idea of leveraging Q-values for supervised policy learning has been shown to be effective in prior work, e.g. (Fern, Yoon, and Givan 2006).

**Expert Planners.** We consider imitation learning from two RDDL planners. The first is the state-of-the-art planner, Prost (Keller and Eyerich 2012) (IPPC-2011), which is based on Monte-Carlo Tree Search with various heuristics and pruning mechanisms. Prost does not generate Q-value estimates for all actions in a state due to pruning mechanisms. Thus, when using Prost, the training data only contains state-action pairs. The second planner is Rollout, which performs policy rollout (Tesauro and Galperin 1997) using a random base policy. Given a state $s$, Rollout produces a very rough estimate of $Q(s, a)$ for each action $a$ as follows. Simulate $N$ trajectories that each start at $s$, then select action $a$ followed by random actions until a fixed horizon. $Q(s, a)$ is estimated to be the average cumulative reward across the trajectories and the Rollout planner returns the action that maximizes $Q(s, a)$. Since Rollout produces Q-value estimates for all actions, we include those values in the training data. Rollout can be viewed as computing a policy that is equivalent to performing one step of policy iteration starting from a random policy. In practice, Rollout is often surprisingly effective and it is often competitive or better than Prost, especially for larger planning problems.

**Parameter Optimization.** For each problem we use both Prost and Rollout to generate a training data set of size 10,000 state-action pairs for three domains and up to 32,000 for the other two domains (Sysadmin and Game-of-life) depending on the problem size. The data was generated by producing trajectories with horizon $H = 40$. Given one such

Table 1: Sysadmin, Game of Life and Skill Teaching Results

| Problem # | Planners | | TRN(Rollout, 0/1) | | | | TRN(Rollout, Q) | | | | TRN(Prost, 0/1) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Prost | Rollout | $\pi_{lin}$ | $\pi^*$ | $\pi_L$ | $\pi_A$ | $\pi_{lin}$ | $\pi^*$ | $\pi_L$ | $\pi_A$ | $\pi_{lin}$ | $\pi^*$ | $\pi_L$ | $\pi_A$ |
| **Sysadmin** | | | | | | | | | | | | | | |
| 1 | 339 | 332 | 342 | 346 | 344 | 344 | 341 | 346 | 341 | 341 | 342 | 347 | 344 | 341 |
| | | | | R310 | F110 | F110 | | R110 | F310 | Linear | | S310 | F310 | F15 |
| 2 | 301 | 290 | 302 | 315 | 309 | 313 | 313 | 319 | 311 | 313 | 311 | 321 | 321 | 316 |
| | | | | S510 | S110 | S15 | | S110 | S310 | R110 | | S310 | S310 | F110 |
| 3 | 553 | 523 | 562 | 575 | 559 | 559 | 559 | 576 | 557 | 562 | 570 | 570 | 561 | 554 |
| | | | | F110 | S110 | S15 | | S110 | S310 | S55 | | Linear | F110 | S35 |
| 4 | 489 | 463 | 502 | 504 | 492 | 504 | 495 | 510 | 501 | 501 | 496 | 513 | 490 | 486 |
| | | | | F110 | S110 | F110 | | F110 | S310 | S35 | | F11 | S110 | F15 |
| 5 | 573 | 588 | 625 | 645 | 631 | 618 | 625 | 649 | 628 | 634 | 620 | 650 | 637 | 638 |
| | | | | S15 | S110 | R110 | | R310 | S310 | S110 | | S15 | S110 | F110 |
| 6 | 527 | 532 | 583 | 598 | 590 | 598 | 583 | 597 | 597 | 595 | 576 | 601 | 578 | 573 |
| | | | | S15 | S110 | S15 | | S310 | S310 | R110 | | S310 | S110 | F15 |
| 7 | 618 | 658 | 724 | 734 | 733 | 714 | 727 | 737 | 730 | 737 | 709 | 730 | 723 | 711 |
| | | | | S310 | S15 | R35 | | S310 | S110 | S310 | | R35 | S15 | S35 |
| 8 | 498 | 522 | 589 | 591 | 589 | 569 | 596 | 600 | 584 | 579 | 583 | 591 | 591 | 583 |
| | | | | S15 | Linear | R11 | | F11 | S110 | S15 | | S15 | S15 | Linear |
| 9 | 728 | 811 | 872 | 889 | 875 | 872 | 884 | 893 | 883 | 877 | 832 | 849 | 833 | 849 |
| | | | | R35 | S15 | Linear | | R310 | S110 | R35 | | F11 | S15 | F11 |
| 10 | 546 | 580 | 643 | 645 | 641 | 643 | 639 | 655 | 643 | 624 | 608 | 624 | 608 | 624 |
| | | | | F11 | S15 | Linear | | F11 | S110 | R15 | | S11 | Linear | S11 |
| %Δ Prost | 0 | 1.62 | 9.82 | 11.72 | 10.21 | 9.81 | 10.22 | 12.55 | 10.43 | 10.21 | 8.32 | 11.18 | 9.23 | 8.82 |
| %Δ Rollout | -1.32 | 0 | 8.00 | 9.88 | 8.37 | 8.03 | 8.38 | 10.69 | 8.59 | 8.40 | 6.63 | 9.44 | 7.50 | 7.06 |
| **Game of Life** | | | | | | | | | | | | | | |
| 1 | 210 | 188 | 77 | 196 | 196 | 196 | 70 | 202 | 197 | 199 | 49 | 191 | 191 | 188 |
| | | | | F310 | F310 | F310 | | F510 | S510 | S310 | | S310 | S310 | F310 |
| 2 | 130 | 122 | 96 | 125 | 125 | 125 | 98 | 135 | 135 | 121 | 85 | 129 | 126 | 129 |
| | | | | F510 | F510 | F510 | | F510 | F510 | F55 | | F510 | F310 | F510 |
| 3 | 150 | 134 | 128 | 146 | 146 | 141 | 121 | 148 | 148 | 148 | 119 | 149 | 148 | 149 |
| | | | | F510 | S510 | F310 | | R510 | S510 | S510 | | F510 | S310 | F510 |
| 4 | 347 | 347 | 225 | 331 | 331 | 331 | 227 | 339 | 338 | 338 | 206 | 321 | 304 | 321 |
| | | | | S310 | S310 | S310 | | S55 | S510 | S510 | | S310 | S35 | S310 |
| 5 | 309 | 295 | 240 | 285 | 285 | 280 | 234 | 304 | 304 | 304 | 229 | 299 | 287 | 299 |
| | | | | S35 | S35 | S510 | | S510 | S510 | S510 | | S510 | S310 | S510 |
| 6 | 283 | 266 | 253 | 268 | 267 | 263 | 252 | 277 | 276 | 274 | 245 | 277 | 275 | 277 |
| | | | | S510 | S310 | S55 | | F35 | S35 | S510 | | S310 | S35 | S310 |
| 7 | 486 | 500 | 330 | 455 | 449 | 447 | 308 | 481 | 481 | 481 | 280 | 435 | 421 | 435 |
| | | | | S510 | S35 | S310 | | S510 | S510 | S510 | | S510 | S35 | S510 |
| 8 | 435 | 450 | 330 | 431 | 431 | 431 | 337 | 449 | 446 | 446 | 313 | 408 | 408 | 406 |
| | | | | S55 | S55 | S55 | | S55 | S510 | S510 | | S35 | S35 | S510 |
| 9 | 410 | 412 | 340 | 416 | 414 | 416 | 344 | 429 | 419 | 419 | 335 | 402 | 399 | 402 |
| | | | | S310 | S35 | S55 | | S510 | S55 | S55 | | S55 | S35 | S55 |
| 10 | 575 | 602 | 263 | 488 | 486 | 488 | 252 | 531 | 531 | 531 | 280 | 513 | 483 | 476 |
| | | | | S510 | S310 | S510 | | S510 | S510 | S510 | | S510 | S35 | S310 |
| %Δ Prost | 0 | -2.58 | -29.91 | -5.06 | -5.33 | -5.93 | -31.18 | -0.72 | -1.36 | -2.37 | -35.31 | -5.20 | -7.43 | -6.02 |
| %Δ Rollout | 2.98 | 0 | -27.74 | -2.20 | -2.45 | -3.12 | -29.09 | 2.23 | 1.56 | 0.49 | -33.43 | -2.27 | -4.51 | -3.07 |
| **Skill Teaching** | | | | | | | | | | | | | | |
| 1 | 67 | 65 | 66 | 67 | 67 | 66 | 64 | 67 | 66 | 64 | 67 | 68 | 65 | 65 |
| | | | | F11 | S510 | F31 | | F15 | S53 | S55 | | R110 | F55 | F55 |
| 2 | 80 | 76 | 76 | 78 | 76 | 78 | 76 | 78 | 75 | 77 | 78 | 80 | 77 | 77 |
| | | | | R35 | R510 | R35 | | F110 | S53 | S31 | | F31 | F310 | F310 |
| 3 | 74 | 85 | 83 | 94 | 85 | 82 | 80 | 98 | 92 | 78 | 87 | 106 | 89 | 89 |
| | | | | S15 | R55 | F31 | | R31 | S55 | F11 | | F15 | S510 | S510 |
| 4 | 101 | 84 | 62 | 104 | 101 | 82 | 56 | 110 | 91 | 89 | 114 | 114 | 93 | 91 |
| | | | | R51 | R55 | R15 | | R31 | R510 | S35 | | F110 | F55 | R35 |
| 5 | 10 | -10 | -28 | -4 | -23 | -39 | -14 | -4 | -42 | -19 | 17 | 36 | -1 | -1 |
| | | | | R51 | R53 | S310 | | R55 | R310 | R53 | | R31 | F35 | F35 |
| 6 | -11 | -11 | 31 | 33 | -1 | -6 | 17 | 24 | -25 | 9 | 5 | 21 | -4 | -4 |
| | | | | S51 | R310 | F110 | | F51 | R510 | S110 | | F310 | F35 | F35 |
| 7 | -48 | -83 | -68 | -46 | -89 | -49 | -59 | -40 | -60 | -51 | -44 | -23 | -62 | -62 |
| | | | | R53 | R55 | S310 | | R53 | R510 | S510 | | F31 | R310 | R310 |
| 8 | -141 | -210 | -191 | -142 | -163 | -212 | -155 | -139 | -156 | -156 | -154 | -109 | -144 | -134 |
| | | | | F310 | R55 | F110 | | F110 | R55 | R55 | | F510 | S310 | S55 |
| 9 | -145 | -155 | -160 | -138 | -161 | -162 | -146 | -122 | -155 | -155 | -167 | -122 | -156 | -172 |
| | | | | F510 | R35 | R55 | | S35 | R35 | R35 | | S15 | F15 | F510 |
| 10 | -214 | -212 | -216 | -194 | -226 | -240 | -247 | -188 | -268 | -279 | -228 | -178 | -214 | -194 |
| | | | | F310 | R53 | R31 | | F11 | R510 | F15 | | R15 | F110 | F53 |
| %Δ Prost | 0 | -34.46 | -13.40 | 28.75 | -35.04 | -54.41 | -9.29 | 24.70 | -72.14 | -18.76 | 21.96 | 71.06 | -8.41 | -8.12 |
| %Δ Rollout | 29.01 | 0 | 19.23 | 57.26 | -0.21 | -20.93 | 20.93 | 52.84 | -38.46 | 12.03 | 50.17 | 94.30 | 22.03 | 22.11 |

dataset we optimize the parameters of a DRP by defining a loss function over the training data and applying stochastic gradient descent. In this work, for all problems and networks we use the Adam optimizer built into the Tensorflow framework with a batch size of 40 and initial learning rate of $10^{-5}$. We train for 2000 iterations and compute the ac-

Table 2: Tamarisk Results

| Problem # | Planners: Prost | Planners: Rollout | TRN(Rollout, 0/1): $\pi_{lin}$ | TRN(Rollout, 0/1): $\pi^*$ | TRN(Rollout, 0/1): $\pi_L$ | TRN(Rollout, 0/1): $\pi_A$ | TRN(Rollout, Q): $\pi_{lin}$ | TRN(Rollout, Q): $\pi^*$ | TRN(Rollout, Q): $\pi_L$ | TRN(Rollout, Q): $\pi_A$ | TRN(Prost, 0/1): $\pi_{lin}$ | TRN(Prost, 0/1): $\pi^*$ | TRN(Prost, 0/1): $\pi_L$ | TRN(Prost, 0/1): $\pi_A$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | -137 | -160 | -177 | -124 | -142 | -142 | -173 | -127 | -145 | -145 | -162 | -123 | -142 | -142 |
| | | | | F35 | S310 | S310 | | S55 | S35 | S35 | | F15 | S310 | S310 |
| 2 | -469 | -524 | -587 | -469 | -485 | -475 | -571 | -473 | -502 | -484 | -532 | -427 | -486 | -472 |
| | | | | R15 | S35 | R35 | | R15 | S35 | S310 | | F31 | S510 | S310 |
| 3 | -210 | -243 | -244 | -198 | -211 | -211 | -274 | -207 | -207 | -207 | -256 | -186 | -209 | -200 |
| | | | | S15 | S310 | S310 | | S310 | S310 | S310 | | R15 | S310 | S35 |
| 4 | -744 | -783 | -786 | -650 | -705 | -705 | -805 | -669 | -719 | -719 | -782 | -694 | -701 | -694 |
| | | | | R110 | S35 | S35 | | S15 | R310 | R310 | | R310 | S35 | R310 |
| 5 | -568 | -646 | -671 | -560 | -615 | -560 | -640 | -547 | -588 | -588 | -645 | -526 | -558 | -579 |
| | | | | S35 | S310 | S35 | | S510 | R310 | R310 | | R510 | S310 | S35 |
| 6 | -1005 | -977 | -940 | -834 | -883 | -886 | -969 | -866 | -882 | -891 | -1100 | -893 | -938 | -938 |
| | | | | S110 | S310 | R310 | | S55 | S310 | R310 | | S55 | S35 | S35 |
| 7 | -862 | -829 | -834 | -662 | -669 | -669 | -809 | -677 | -687 | -677 | -875 | -679 | -709 | -709 |
| | | | | S510 | S310 | S310 | | S53 | S310 | S53 | | S110 | S310 | S310 |
| 8 | -1380 | -1229 | -1210 | -1087 | -1139 | -1165 | -1203 | -1104 | -1131 | -1144 | -1361 | -1228 | -1243 | -1243 |
| | | | | R510 | S35 | S310 | | R110 | S55 | S35 | | F51 | S35 | S35 |
| 9 | -1010 | -827 | -803 | -686 | -797 | -736 | -867 | -681 | -735 | -681 | -961 | -752 | -818 | -821 |
| | | | | F11 | S310 | S510 | | R510 | S510 | R510 | | R310 | S310 | S53 |
| 10 | -1548 | -1228 | -1259 | -1064 | -1124 | -1095 | -1254 | -1057 | -1201 | -1201 | -1528 | -1375 | -1394 | -1595 |
| | | | | S110 | S35 | S310 | | F11 | F510 | F510 | | S53 | S35 | F510 |
| %Δ Prost | 0 | -0.72 | -3.27 | 15.39 | 8.99 | 10.73 | -4.39 | 14.14 | 8.87 | 9.71 | -7.56 | 12.48 | 6.38 | 5.52 |
| %Δ Rollout | -1.16 | 0 | -2.15 | 15.87 | 9.52 | 11.28 | -3.33 | 14.69 | 9.40 | 10.30 | -7.70 | 11.91 | 6.15 | 4.90 |

curacy on a validation set every 500 iterations and stop if there is no improvement in two successive stages. Training times vary significantly for different problems and architectures, which can be improved with additional hardware and further optimizations.

*Training with 0/1 Loss.* Our first loss is defined over just state-action pairs. Given a state $s$ a DRP produces a probability distribution over actions, which we will denote by the vector $\hat{P}(s)$. Given a training state-action pair $(s, a)$, let $t(s)$ denote the 0/1 target probability distribution over actions that assigns probability 1 to action $a$. We measure the 0/1 cross entropy loss of a prediction $\hat{P}(s)$ as the cross-entropy $H(\hat{P}(s), t(s))$ between $\hat{P}(s)$ and $t(s)$, where for probability vectors $P$ and $Q$, $H(P, Q) = -\sum_i P_i \log(Q_i)$. $H(P, Q)$ is minimized when $P = Q$ and hence the 0/1 loss encourages $\hat{P}(s)$ to increase the probability of the action.

*Training with Q-Loss.* When Q-values are available in the training data, we incorporate them by defining a Q-Loss function that prefers predictions $\hat{P}(s)$ that assign higher probabilities to actions with higher Q-values. In particular, we use the Q-values for a state $s$ to define a Boltzmann probability distribution over actions $P(a \mid s) = \frac{\exp(Q(s,a))}{\sum_{a'} \exp(Q(s,a'))}$ with temperature equal to one. Here $P$ assigns higher probability to actions with higher Q-values. Our Q-loss function for a training example is then simply $H\left(\hat{P}(s), P(\cdot|s)\right)$, which is minimized when the predicted probabilities match the Boltzmann probabilities.

**Doing Better than the Expert.** In our experiments, we will sometimes see the learned DRPs outperforming the expert planners. The exact reasons for this is not fully clear, however, results from imitation-learning theory offer a potential explanation. First, it is important to note that Prost and Rollout are both stochastic planners due to running Monte-Carlo simulations. One way to model the stochasticity is by starting with a deterministic policy $\pi^*$, which captures the typical action choices of the planner and then creating a stochastic policy $\hat{\pi}$ that follows $\pi^*$ with $1-\epsilon$ probability and uses a randomized action choice with $\epsilon$ probability. It has been shown (Ross and Bagnell 2010) that the finite-horizon reward of $\hat{\pi}$ can be worse than $\pi^*$ by as much as $\epsilon H^2$, where $H$ is the horizon. Thus, even if Prost and Rollout typically select actions according to a high-quality $\pi^*$, their actual performance can be substantially worse. We can now think of the training data as being generated by $\pi^*$, but corrupted with some amount of noise. If our learning procedure is robust to the noise, then it is possible for the learned DRP to provide a better approximation of $\pi^*$ than the planners. In particular, if the learned approximation has an error rate of $\epsilon' < \epsilon$, then the learned policy has the potential to achieve a performance closer to $\pi^*$ than the planner.

## Experiments

**Benchmark Problems and Architectures.** We selected five RDDL benchmark domains: Sysadmin, Game-of-Life, Skill Teaching, Tamarisk, and Wildfire. Each domain comes with a standard set of ten problems ranging from quite small to quite large. While computational constraints prevented including additional domains, there are some benchmark domains that are not a good match for DRPs. For example, the Navigation domain contains state variables that only provide the robot location. To be successful a planner needs to reason about the probabilistic navigation grid to eventually find a deterministic optimal path. In such domains, there is no room to benefit from the generalization ability of a DNNs. All the selected domains appear to offer non-trivial opportunities to learn policies that generalize across states.

We trained FC-DRPs, S-DRPs, and R-DRPs for all combinations of $L = 1, 3, 5$ and $C = 1, 5, 10$ along with a linear policy (no hidden layers). Each architecture was trained using three strategies: Rollout as the planner with 0/1 loss,

Table 3: Wildfire Results

| | Planners | | TRN(Rollout, 0/1) | | | | TRN(Rollout, Q) | | | | TRN(Prost, 0/1) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Problem # | Prost | Rollout | $\pi_{lin}$ | $\pi^*$ | $\pi_L$ | $\pi_A$ | $\pi_{lin}$ | $\pi^*$ | $\pi_L$ | $\pi_A$ | $\pi_{lin}$ | $\pi^*$ | $\pi_L$ | $\pi_A$ |
| 1 | -275 | -439 | -481 | -256 | -603 | -603 | -522 | -368 | -368 | -368 | -950 | -159 | -238 | -208 |
| | | | | S110 | S35 | S35 | | R510 | F15 | F15 | | F35 | R15 | S510 |
| 2 | -8856 | -8913 | -9466 | -8621 | -8900 | -8783 | -8989 | -8674 | -9078 | -9078 | -8807 | -8428 | -9034 | -9034 |
| | | | | S55 | F15 | S15 | | R110 | S15 | S15 | | F11 | F15 | F15 |
| 3 | -1899 | -1547 | -1354 | -1131 | -1131 | -1373 | -2037 | -976 | -1285 | -1517 | -1355 | -802 | -1747 | -1490 |
| | | | | R510 | R510 | F11 | | S15 | F11 | R310 | | F15 | S35 | R310 |
| 4 | -8756 | -8986 | -8572 | -7808 | -8136 | -8459 | -8840 | -7757 | -8040 | -8040 | -8888 | -7693 | -7693 | -9121 |
| | | | | R110 | R510 | S55 | | R310 | S35 | S35 | | S35 | S35 | F31 |
| 5 | -3220 | -585 | -1331 | -467 | -716 | -1100 | -800 | -497 | -497 | -723 | -1552 | -1552 | -2959 | -2517 |
| | | | | R15 | S11 | R510 | | F11 | F11 | S35 | | Linear | S110 | F510 |
| 6 | -15878 | -7079 | -7370 | -6548 | -7465 | -6820 | -7132 | -6480 | -7221 | -7221 | -15313 | -10948 | -14056 | -11975 |
| | | | | F15 | F11 | S35 | | R15 | R55 | R55 | | F15 | S110 | F510 |
| 7 | -7731 | -6169 | -5483 | -4885 | -5169 | -6479 | -5452 | -5178 | -5648 | -5882 | -7327 | -6270 | -9259 | -9259 |
| | | | | S15 | S11 | F510 | | S310 | S15 | F510 | | R31 | F510 | F510 |
| 8 | -13673 | -10192 | -9975 | -9389 | -9389 | -10840 | -9411 | -9305 | -9529 | -9828 | -13053 | -11235 | -16661 | -16661 |
| | | | | S11 | S11 | S53 | | S110 | F11 | S15 | | R11 | F510 | F510 |
| 9 | -16129 | -5551 | -4941 | -4152 | -6662 | -6662 | -4317 | -4310 | -5911 | -5911 | -17962 | -17036 | -17556 | -17556 |
| | | | | R310 | F510 | F510 | | R51 | F510 | F510 | | F35 | F510 | F510 |
| 10 | -25459 | -12049 | -11238 | -9763 | -12030 | -12030 | -10586 | -9683 | -11113 | -11113 | -31343 | -29047 | -30061 | -30061 |
| | | | | R35 | F510 | F510 | | F11 | F510 | F510 | | F53 | F510 | F510 |
| %Δ Prost | 0 | 25.68 | 24.26 | 40.90 | 23.44 | 18.39 | 22.30 | 37.24 | 32.04 | 29.59 | -18.67 | 21.66 | -1.74 | 1.74 |
| %Δ Rollout | -91.75 | 0 | -9.91 | 18.54 | -2.45 | -13.43 | -3.35 | 15.99 | 7.12 | 1.09 | -81.34 | -44.79 | -93.72 | -82.48 |

Rollout with Q-loss, and Prost with 0/1 loss. The strategies are denoted by *TRN(Rollout,0/1)*, *TRN(Rollout, Q)*, and *TRN(Prost, 0/1)* respectively. In total this resulted in training $27 \times 3$ networks for each of the 50 problems.

**Description of Results Tables.** Tables 1-3 contain our main set of experimental results for each domain. Throughout our experiments, the expected total reward of a policy or planner is estimated using a horizon of 40 averaged over 100 simulations. In each table the top 10 rows give results for individual problems, where larger problem numbers tend to correspond to larger problems. The second and third columns give the average reward of Prost and Rollout. The next three blocks of columns correspond to one of the three training methods. For each training method the first column gives the average reward for the trained linear policy $\pi_{lin}$. The next column, labeled $\pi^*$, gives the maximum reward achieved over all architectures trained for the problem (all combinations of F(L,C), S(L,C), and R(L,C)). Below this maximum reward is the name of the architecture that achieved the maximum reward, e.g. S110 is an S-DRP with $L = 1$ and $C = 10$. This maximum reward is what we would achieve in practice if we performed DRP model selection via simulation of the learned policies, which will be practical in some settings.

The final two columns in each block are included to assess our ability to perform model selection using validation data, rather than simulations as for $\pi^*$. In particular, for each problem in addition to the training set we generated a set of validation data containing 2000 state-action pairs from the appropriate planner. Given a learned DRP we can evaluate the loss it achieves on the validation data (either 0/1 or Q loss as appropriate) and the accuracy of selecting the actions in the validation set. For each problem, the column $\pi_L$ ($\pi_A$) gives the average reward and name of the architecture that minimized (maximized) the validation loss (accuracy). For large RDDL benchmarks and moderate number of DRP architectures it will often be much cheaper to select models using the validation set measures compared to using simulation to estimate expected reward. Thus, the $\pi_L$ and $\pi_A$ columns are included to help evaluate how effective this cheaper form of model selection might be.

Finally, the last two rows of each table aggregate results across problems. The row labeled $\%\Delta Prost$ ($\%\Delta Rollout$) gives the average percentage improvement over Prost (Rollout) across problems for each column. For example, in Sysadmin, the Rollout planner achieves a negligible average improvement over Prost of 1.62%. Negative values indicate an average decrease in performance.

**Comparison to Expert Planners.** First we consider the performance of the simple linear policy. We note that for Sysadmin and Wildfire that on averge $\pi_{lin}$ is able to achieve a non-trivial average performance improvement over both Prost and Rollout for all of the training regimes, with the exception of TRN(Prost, 0/1) for Wildfire. This indicates that it is possible to represent good policies in these two domains using simple functions. This is also an example of where a learned policy outperforms the expert that it was learned from, for which, we presented a potential explanation. For other domains, $\pi_{lin}$ performs worse than the experts, which indicates that good policies in these domains require more complex representations or that a good linear policy was not learnable using this training data. Now consider the performance of $\pi^*$, which is the best we would hope to do when using simulation for model selection. For all domains, with the exception of Game-of-Life, $\pi^*$ had better average performance than both expert planners in all three training regimes. One exception was a decrease in performance relative to Rollout in Wildfire for TRN(Prost, 0/1). This decrease is understandable since the performance of Prost in this domain is quite poor compared to Rollout (average performance reduction of -91.75%), which means learning from Prost is unlikely to yield good performance. In Game-of-

Table 4: FC-DRP vs S-DRP vs R-DRP Results

| Domain | %$\Delta$ | TRN(Rollout, 0/1) | | | | TRN(Rollout, Q) | | | | TRN(Prost, 0/1) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $\pi^*$ | $\pi_F$ | $\pi_S$ | $\pi_R$ | $\pi^*$ | $\pi_F$ | $\pi_S$ | $\pi_R$ | $\pi^*$ | $\pi_F$ | $\pi_S$ | $\pi_R$ |
| Sysadmin | Prost | 11.72 | 10.85 | 11.11 | 10.56 | 12.55 | 11.74 | 11.96 | 11.83 | 11.18 | 10.30 | 10.80 | 9.91 |
| | Rollout | 9.88 | 9.04 | 9.27 | 8.73 | 10.69 | 9.89 | 10.13 | 9.96 | 9.44 | 8.59 | 9.07 | 8.18 |
| Game of Life | Prost | -5.06 | -8.01 | -5.49 | -12.32 | -0.72 | -4.48 | -1.64 | -6.79 | -5.20 | -10.01 | -5.45 | -13.17 |
| | Rollout | -2.20 | -5.07 | -2.66 | -9.68 | 2.23 | -1.44 | 1.25 | -3.90 | -2.27 | -6.96 | -2.53 | -10.42 |
| Skill Teaching | Prost | 28.75 | -10.93 | 18.95 | 6.87 | 24.70 | 17.90 | -1.89 | 18.55 | 71.06 | 62.72 | 61.09 | 53.67 |
| | Rollout | 57.26 | 19.89 | 48.39 | 36.19 | 52.84 | 46.27 | 28.15 | 47.23 | 94.30 | 86.50 | 85.68 | 79.50 |
| Tamarisk | Prost | 15.39 | 12.19 | 13.95 | 13.06 | 14.14 | 11.98 | 13.56 | 13.26 | 12.48 | 10.88 | 10.42 | 11.90 |
| | Rollout | 15.87 | 12.74 | 14.38 | 13.67 | 14.69 | 12.69 | 14.00 | 13.86 | 11.91 | 10.28 | 10.05 | 11.33 |
| Wildfire | Prost | 40.90 | 36.26 | 39.41 | 39.76 | 37.24 | 35.08 | 35.42 | 35.63 | 21.66 | 17.63 | 11.98 | 12.12 |
| | Rollout | 18.54 | 11.82 | 15.79 | 17.15 | 15.99 | 12.97 | 11.44 | 13.16 | -44.79 | -60.91 | -75.68 | -69.14 |

Life $\pi^*$ is on average worse than both expert planners by a small percentage in all three training regimes. We note that the relatively small drop in performance compared to the planners, comes with a dramatic improvement in decision making time. Finally, we note that for all benchmarks and all training regimes, $\pi^*$ is able to significantly outperform the linear policy on average, except in Sysadmin, where $\pi^*$ is better by only a small margin. This shows that there is indeed benefit to considering deeper non-linear architectures for these RDDL benchmarks.

**Comparison of Model Selection Strategies.** Here we consider the impact of using the validation set loss $\pi_L$ and validation accuracy $\pi_A$ for model selection instead of using simulations as done for $\pi^*$. We first observe that for Sysadmin, Game-of-Life, and Tamarisk, the drop in performance of $\pi_L$ and $\pi_A$ compared to $\pi^*$ for all training regimes is relatively small in terms of average percent improvement over the planners. For Wildfire the performance drop is more significant when learning based on 0/1 loss, especially for TRN(Prost, 0/1), which may again be due to the low quality of Prost's data in this domain. In Skill Teaching the drop in average performance of both $\pi_L$ and $\pi_A$ is substantial in all training regimes. This indicates a poor match between the validation loss and actual reward accumulated during planning. The reasons for this are currently unclear. We also observe that except for Skill Teaching, in all cases where $\pi^*$ showed an average positive improvement over an expert planner, $\pi_L$ and $\pi_A$ were also able to achieve an improvement. Thus, in most cases if we were satisfied with the performance of the expert planner, we would also be satisfied with the much faster DRPs selected by $\pi_L$ and $\pi_A$. Finally, there does not appear to be a clear winner between $\pi_L$ and $\pi_A$, nor does there appear to be a training regime where model selection based on validation data performs best.

**Comparison of Training Methods.** For Sysadmin, Game-of-Life, and Tamarisk we see that Prost and Rollout achieve similar average performance across problems. In these cases, we see that the performance of $\pi^*$ is also similar across the three training regimes. In Skill Teaching, Rollout is significantly worse on average than Prost (by -34.46%) and we see that $\pi^*$ trained with TRN(Rollout, 0/1) and TRN(Rollout, Q) is significantly worse on average than with TRN(Prost, 0/1). This agrees with the intuition that learning from a lower quality planner should result in a worse learned policy. For Wildfire the situation is reversed and we see that training from Prost data is significantly worse than training from rollout with TRN(Rollout, 0/1) and TRN(rollout, Q) performance similarly on average. Overall these results indicate that for these experiments the quality of the planner used to generate data is the dominating factor in training, compared to using 0/1 or Q-based loss. This is in contrast to prior studies (Fern, Yoon, and Givan 2006; Anthony, Tian, and Barber 2017), where Q-based loss improved performance. This suggests investigating improved ways to incorporate Q-values into loss functions.

**Comparison of Architectures.** From the tables we can observe for each problem and training regime, which architecture was selected by $\pi^*$. Overall, from this data we do not see a consistent trend that would favor particular architectural properties. In particular, we see FC-DRPs, S-DRPs, and R-DRPs all appearing with reasonable frequencies. We do see that in Game-of-Life, which is perhaps the most complex policy to learn based on the difficulty of competing with the planners, we see that for large problems S-DRPs with larger values of $L$ and $C$ tend to be chosen. It is also difficult to spot an overall trend in terms of $L$ or $C$. We did find that sparse architectures suffer much more than FC-DRPs when $C = 1$, which requires further investigation. We note that these results are at best suggestive, since the tables do not indicate how close other architectures were to the performance of $\pi^*$.

Table 4 summarizes the performances of the best FC-DRP, S-DRP, and R-DRP across the domains. Each row gives the averaged % improvement over Prost or Rollout for each domain. The first column in each training regime, copies results for $\pi^*$ from the previous table and represents the best performance over all architectures. The next three columns record the average improvement of the best architecture restricted to FC-DRPs ($\pi_F$), S-DRPs ($\pi_S$), and R-DRPs ($\pi_R$). Again we see that there is not a consistently best single top performing class. We do see that at least one of the sparsely connected DRPs, $\pi_S$ and $\pi_R$, always outperform the fully connected architectures ($\pi_F$), with the exception of Wildfire and Tamarisk for TRN(Prost, 0/1). This is suggestive that the sparse architectures can leverage the RDDL definition to realize a benefit. We have also seen that frequently the sparse architectures are able to achieve similar results to FC-DRPs using many fewer parameters. It is also encouraging to see that the weight sharing approach of $\pi_R$ is usually competitive on average even though it uses dramatically fewer parameters. This suggests the potential effectiveness of relational generalization across planning problems

in a domain. Finally, we see that $\pi^*$ sometimes significantly outperforms the others. This indicates that within a single problem domain the best architecture class differs across the problems.

## Acknowledgements

This work was supported by NSF grant IIS-1619433 and DARPA contract N66001-17-2-4030. We thank Intel for assisting with compute support for this work.

## References

Anand, A.; Noothigattu, R.; Singla, P.; et al. 2016. Oga-uct: on-the-go abstractions in uct. In *Twenty-Sixth International Conference on Automated Planning and Scheduling*.

Anthony, T.; Tian, Z.; and Barber, D. 2017. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems 30*, 5364–5374.

Bonet, B., and Geffner, H. 2012. Action selection for MDPs: Anytime AO* versus UCT. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Buffet, O., and Aberdeen, D. 2009. The factored policy-gradient planner. *Artificial Intelligence* 173(5):722–747.

Cui, H.; Khardon, R.; Fern, A.; and Tadepalli, P. 2015. Factored mcts for large scale stochastic planning. In *AAAI*, 3261–3267.

De La Rosa, T.; Celorrio, S. J.; and Borrajo, D. 2008. Learning relational decision trees for guiding heuristic planning. In *International Conference on Automated Planning and Scheduling*, 60–67.

Fern, A.; Yoon, S. W.; and Givan, R. 2006. Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *Journal of AI Research (JAIR)* 25:75–118.

Hoey, J.; St-Aubin, R.; Hu, A.; and Boutilier, C. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Ipsen, I. C., and Meyer, C. D. 1998. The idea behind krylov methods. *American Mathematical Monthly* 889–899.

Issakkimuthu, M.; Fern, A.; Khardon, R.; Tadepalli, P.; and Xue, S. 2015. Hindsight optimization for probabilistic planning with factored actions. In *International Conference on Automated Planning and Scheduling*, 120–128.

Jiménez, S.; De la Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review* 27(4):433–467.

Keller, T., and Eyerich, P. 2012. PROST: Probabilistic Planning Based on UCT. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Khardon, R. 1999. Learning action strategies for planning domains. *AIJ* 113(1-2):125–148.

Kolobov, A.; Dai, P.; Mausam; and Weld, D. 2012. Reverse iterative deepening for finite-horizon mdps with large branching factors. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Levine, S.; Pastor, P.; Krizhevsky, A.; Ibarz, J.; and Quillen, D. 2016. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research* 0278364917710318.

Martin, M., and Geffner, H. 2000. Learning generalized policies in planning domains using concept languages. In *KRR*.

Minton, S.; Carbonell, J.; Knoblock, C. A.; Kuokka, D. R.; Etzioni, O.; and Gil, Y. 1989. Explanation-based learning: A problem solving perspective. *AIJ* 40:63–118.

Minton, S., ed. 1993. *Machine Learning Methods for Planning*. Morgan Kaufmann.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Petrik, M. 2007. An analysis of laplacian methods for value function approximation in mdps. In *IJCAI*, 2574–2579.

Raghavan, A.; Joshi, S.; Fern, A.; Tadepalli, P.; and Khardon, R. 2012. Planning in factored action spaces with symbolic dynamic programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Raghavan, A.; Khardon, R.; Fern, A.; and Tadepalli, P. 2013. Symbolic opportunistic policy iteration for factored-action MDPs. In *Advances in Neural Information Processing Systems*.

Raghavan, A.; Khardon, R.; Tadepalli, P.; and Fern, A. 2015. Memory-effcient symbolic online planning for factored mdps. In *UAI*, 732–741.

Ross, S., and Bagnell, D. 2010. Efficient reductions for imitation learning. In *International Conference on Artificial Intelligence and Statistics*, 661–668.

Sanner, S. 2010. Relational dynamic influence diagram language (rddl): Language description.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *Nature* 550(7676):354–359.

Tesauro, G., and Galperin, G. R. 1997. On-line policy improvement using monte-carlo search. In *Advances in Neural Information Processing Systems*, 1068–1074.

Toyer, S.; Trevizan, F.; Thiébaux, S.; and Xie, L. 2018. Action schema networks: Generalised policies with deep learning. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Yoon, S.; Fern, A.; and Givan, R. 2002. Inductive policy selection for first-order MDPs. In *Conference on Uncertainty in Artificial Intelligence*.

Zimmerman, T., and Kambhampati, S. 2003. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine* 24(2)(2):73–96.