Deep Reactive Policies for Planning in Stochastic Nonlinear Domains

Thiago P. Bueno,[1] **Leliane N. de Barros,**[1] **Denis D. Mauá,**[1] **Scott Sanner**[2]
[1]Department of Computer Science, University of São Paulo, Brazil
[2]Industrial Engineering, University of Toronto, Canada
tbueno@ime.usp.br, leliane@ime.usp.br, ddm@ime.usp.br, ssanner@mie.utoronto.ca

Abstract

Recent advances in applying deep learning to planning have shown that Deep Reactive Policies (DRPs) can be powerful for fast decision-making in complex environments. However, an important limitation of current DRP-based approaches is either the need of optimal planners to be used as ground truth in a supervised learning setting or the sample complexity of high-variance policy gradient estimators, which are particularly troublesome in continuous state-action domains. In order to overcome those limitations, we introduce a framework for training DRPs in continuous stochastic spaces via gradient-based policy search. The general approach is to explicitly encode a parametric policy as a deep neural network, and to formulate the probabilistic planning problem as an optimization task in a stochastic computation graph by exploiting the re-parameterization of the transition probability densities; the optimization is then solved by leveraging gradient descent algorithms that are able to handle non-convex objective functions. We benchmark our approach against stochastic planning domains exhibiting arbitrary differentiable nonlinear transition and cost functions (e.g., Reservoir Control, HVAC and Navigation). Results show that DRPs with more than 125,000 continuous action parameters can be optimized by our approach for problems with 30 state fluents and 30 action fluents on inexpensive hardware under 6 minutes. Also, we observed a speedup of 5 orders of magnitude in the average inference time per decision step of DRPs when compared to other state-of-the-art online gradient-based planners when the same level of solution quality is required.

Introduction

Deep Learning (DL) methods have achieved remarkable success over the past few years in a large variety of complex tasks, ranging from perception problems (Hinton et al. 2012; Krizhevsky, Sutskever, and Hinton 2012; Sutskever, Vinyals, and Le 2014) to control systems (Mnih et al. 2015; Silver et al. 2016). Inspired by these successful applications, recent advances in Deep Reactive Policies (DRPs) have shown promising results in applying DL to model-known planning.

A distinguished feature of DRPs concerning applications involving autonomous agents is that these policies are particularly suited to fast decision making, in contrast to state-of-the-art planners typically based on online simulation-based methods (Yoon et al. 2008; Keller and Helmert 2013; Raghavan et al. 2017) that can require non-trivial computational time per decision step. However, an important limitation of current approaches to learning DRPs is either the need of optimal planners to be used as supervisors, or the sample complexity of methods based on high-variance policy gradients. Both issues can be troublesome for planning in continuous spaces, especially for domains exhibiting nonlinear dynamics and costs. In the case of training DRPs by supervised learning, no efficient domain-independent optimal planner is currently available to be used as ground truth for the class of stochastic nonlinear domains. On the other hand, policy gradient methods for continuous action spaces can considerably exacerbate the need of huge amounts of data and processing power, which can be prohibitively expensive for common users in most applications.

So, in order to overcome those limitations, we propose a general framework of planning as optimization in continuous spaces. The basic idea is to leverage automatic differentiation in stochastic computation graphs to estimate gradients used to optimize the parameters of DRPs. We exploit the fact that several distributions used in model-known planning are amenable to the re-parameterization trick, which allows for optimizing a surrogate cost function by the application of adaptive stochastic gradient descent methods commonly used in DL. We compare our approach to a stochastic online extension of the recently proposed approach Planning through Backpropagation (Wu, Say, and Sanner 2017).

We are particularly interested in shedding light on the following questions. Is it possible to efficiently train DRPs for stochastic nonlinear domains without supervision or policy gradients? How do offline learned DRPs compare in terms of quality of solution and computational time to online gradient-based planning? How do different DRP architectures (i.e., in terms of number of layers, number of units) compare amongst themselves? Our experiments on stochastic nonlinear domains (e.g., Navigation (Faulwasser and Findeisen 2009), HVAC (Heating, Ventilation, and Air Conditioning) (Agarwal et al. 2010), and Reservoir Control (Yeh 1985)) highlight the flexibility of our approach over diverse differentiable transition and cost functions.

The results show that DRPs trained by gradient-based optimization in stochastic computation graphs can be evaluated within a fraction of the time needed by state-of-the-

art online planners even when the same level of solution quality is required. Indeed, we observed a speedup of 5 orders of magnitude in the average inference time per decision step of DRPs. Moreover, our experiments show that DRPs with more than 125,000 continuous action parameters can be learned by gradient-based optimization for stochastic nonlinear problems with 30 state fluents and 30 action fluents on inexpensive hardware under 6 minutes.

The paper is organized as follows. First, we present the motivations, related work and the formal background of the work. Then, we introduce our main contributions related to planning as optimization as a viable training method for DRPs. We continue with a presentation of the empirical analysis. We conclude with our final remarks and ideas for future work.

Related Work

DRPs have recently received considerable attention in the planning community (Toyer et al. 2018; Issakkimuthu, Fern, and Tadepalli 2018; Groshev et al. 2018). Most works on DRPs, however, have exclusively focused on discrete action spaces and stochastic policies. In these works, the policy network is usually trained by supervised learning. The basic idea is to leverage optimal planners as supervisors from which a target action for each state can be obtained. The DRPs can then be optimized via minimization of the cross-entropy error between the action distribution of the stochastic policy and the ground truth action, akin to an imitation learning approach, but for which the action to be imitated is given by the optimal planner instead of a human. Even though these works have managed to train policies enabling fast decision-making, their main contributions highlight the opportunities of exploiting the structure of the domain, and consequently how transfer learning can be accomplished across instances of the same domain. Conversely, our work focuses on deterministic policies in continuous spaces to which not much attention has been given in the area of model-known planning.

Other methods for training reactive policies have been proposed in the past and have led to the development of efficient planners. The Factored Policy Gradient (FPG) planner (Aberdeen 2005; Buffet and Aberdeen 2007) performs stochastic local search on the policy space using policy gradients, a technique inspired by Reinforcement Learning (RL). Policy gradients have been long studied in RL and many important works have achieved great success in practical applications. We do not revise RL-related works here since we are most concerned with model-known planning. Suffice it to say that policy gradient is a viable option to derive gradient estimators when the model dynamics is unknown or non-differentiable, which is typically the case in model-free RL applications. In contrast, we propose exactly the opposite, i.e., to learn a policy by leveraging gradients backpropagated through the model dynamics itself.

The most closely related prior work in terms of learning parametric policies via gradient-based optimization is the PEGASUS (Policy Evaluation-of-Goodness And Search Using Scenarios) planner (Ng and Jordan 2000). The training method used in PEGASUS is similar to ours, but their approach based on deterministic simulative models relies on i.i.d. noise variables pre-sampled from the standard uniform distribution, which seems unnecessarily restrictive when compared to the more general case based on distribution re-parameterization presented here. Also, differently from our proposed method leveraging flexible architectures of DRPs, the empirical evaluations of PEGASUS on continuous domains rely on hand-engineered features used as inputs to logistic models. Finally, it is assumed that the state space forms a d_S-dimensional unitary hyper-cube (i.e., $S = [0, 1]^{d_S}$), which avoids special considerations regarding normalization of the inputs. We make no restrictive assumptions regarding the state space.

In this work we focus on nonlinear domains. To the best of our knowledge, the only other efficient planner accepting arbitrary factored transition and cost functions in high dimensional continuous spaces exhibiting nonlinear dynamics is tf-plan (Wu, Say, and Sanner 2017).[1] Similarly to the approach presented here, tf-plan uses gradient-based optimization on models with large number of action parameters to solve complex nonlinear problems. However, their formulation is only directly applicable to deterministic problems. Here, we extend the planning as optimization approach to the more general case of stochastic transitions.

Background

Markov Decision Processes

We consider sequential decision-making problems in which an agent is supposed to interact with a discrete-time, stochastic environment modeled by a Markov Decision Process (MDP) (Puterman 2014). We are particularly concerned with continuous state-action MDPs (CSA-MDPs) with exogenous events.

We define a discrete time, finite horizon, **continuous state-action MDP** by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \Omega, \mathcal{T}, p_\omega, R, \mathcal{H}, \mathbf{s}_0)$, in which $\mathcal{S} \subseteq \mathbb{R}^n$ is the state space, $\mathcal{A} \subseteq \mathbb{R}^n$ is the action space, $\Omega \subseteq \mathbb{R}^m$ is the set of exogenous events, $\mathcal{T} \colon \mathcal{S} \times \mathcal{A} \times \Omega \times \mathcal{S} \to [0, 1]$ is the Markovian transition kernel given by the conditional probability density $p(\mathbf{s}'|\mathbf{s}, \mathbf{a}, \omega)$ over next state $\mathbf{s}'$ given the current state $\mathbf{s}$, action $\mathbf{a}$ and event ω, p_ω is the probability density over the set of exogenous events, $C \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}_{\geq 0}$ is the cost function that specifies the immediate return received in the current state $\mathbf{s}$ after applying action $\mathbf{a}$, $\mathcal{H} = 0, 1, ..., H - 1$ is the finite set of decision timesteps, and $\mathbf{s}_0$ is the start state.

CSA-MDPs are factored MDPs (Boutilier, Dean, and Hanks 1999) where the state $\mathbf{s}$ is represented by an n-dimensional vector whose dimensions are independent given the previous state and action. Hence, the transition kernel decomposes over the set of probability density functions of the state fluents: $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \omega_t) = \prod_{j=1}^{n} p(s^j_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \omega_t)$. Also, CSA-MDPs naturally allow concurrency of actions.

[1] No name was suggested in (Wu, Say, and Sanner 2017); we propose denoting their approach as "tf-plan", as it is built on top of TensorFlow's deep learning framework (Abadi et al. 2016).

A **Markovian deterministic policy** is given by a function $\pi \colon \mathcal{S} \times \mathcal{H} \to \mathcal{A}$ that prescribes a single action to each state in a given decision timestep. We abuse notation in order to denote the prescribed action by $\pi(\mathbf{s_t})$ instead of $\pi(\mathbf{s}, t)$.

We define a state **value function** $V^{\pi}(\mathbf{s})$ as the expected total cost received by following policy π from state $\mathbf{s}$ for a finite horizon H , i.e., $V^{\pi}(\mathbf{s}) = \mathbb{E}\left[\sum_{t=0}^{H-1} C(\mathbf{s}_t, \pi(\mathbf{s}_t)) \;\middle|\; \mathbf{s}\right]$. The agent's goal is to find an optimal policy π^* such that:

$$\pi^* = \arg\min_{\pi \in \Pi} V^{\pi}(\mathbf{s}_0)\,.$$

Gradients in Stochastic Computation Graphs

Stochastic computation graphs (Schulman et al. 2015) are a formalism used to specify models that mix deterministic computations with random variables drawn from distributions whose parameters depend on the results of previous computations. Its purpose is to define all dependencies between the variables of an acyclic generative model in order to allow the development of efficient sampling and gradient estimators.

Formally, a **stochastic computation graph** $\mathcal{G} = (V, E)$ is a directed acyclic graph defined over three disjoint sets of nodes: (i) input nodes Θ, assumed directly observable; (ii) deterministic nodes D, corresponding to functions of its parent nodes; and (iii) stochastic nodes S, representing random variables conditionally distributed accordingly to a function whose parameters depend on its parent nodes. The set of nodes V is partitioned as $V = \Theta \cup D \cup S$. An edge (u, v) belongs to E if node v or its probability distribution depends on node u. Let v and w be nodes of a stochastic computation graph $\mathcal{G}$. Then, we denote by $v \prec w$ the property that it exists a dependency path from node v to node w in $\mathcal{G}$. Additionally, we denote by $v \prec^D w$ the property that a dependency path from node v to node w traverses only deterministic nodes. Conversely, $v \prec^S w$ denotes the property that a dependency path from node v to node w traverses at least one stochastic node.

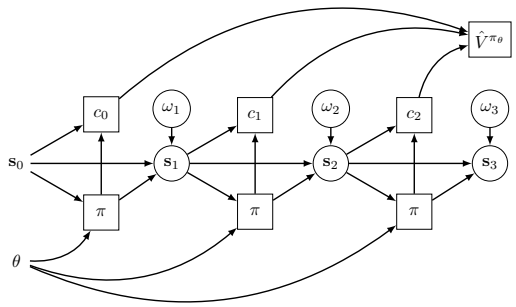

Figure 1: Stochastic computation graph of CSA-MDPs: the inputs nodes are $\mathbf{s}_0$ and θ representing the start state and the policy parameters shared across timesteps; stochastic nodes correspond to state variables $\mathbf{s}_{t+1} \sim p(\cdot|\mathbf{s}_t, \mathbf{a}_t, \omega_t)$ and exogenous events $\omega_t \sim p_\omega(\cdot)$; deterministic nodes consist of action variables $\mathbf{a}_t = \pi_\theta(\mathbf{s}_t)$ and costs $c_t = C(\mathbf{s}_t, \mathbf{a}_t)$. The cost node corresponds to the value function estimate $\hat{V}^{\pi_\theta}$.

The objective function of a stochastic computation graph is given by $J(\theta) = \mathbb{E}_{p_\theta}[\sum \hat{c}]$, where $\hat{c}$ are leaf nodes known as cost nodes. Input nodes are root nodes typically used to define the parameters we would like to differentiate J with respect to. Note that this objective function is given by an expectation whenever the model contains stochastic nodes. Therefore, we cannot directly use automatic reverse-mode differentiation algorithms (Griewank and Walther 2008) to optimize J in the general case where there is a path in graph such that the condition $\theta \prec^S J(\theta)$ holds.

Figure 1 shows the stochastic computation graph of CSA-MDPs with exogenous events. We follow the graphical notation presented in (Schulman et al. 2015) where square and rounded nodes represent deterministic functions and stochastic variables, respectively. Also, nodes depicted without border are considered observed, i.e., inputs nodes. Note that there are multiple paths between the parameters θ and the cost node $\hat{V}^{\pi_\theta} = \sum_t c_t$. For each timestep t, the only path satisfying $\theta \prec^D \hat{V}^{\pi_\theta}$ traverses the graph through the current cost c_t, but paths satisfying $\theta \prec^S \hat{V}^{\pi_\theta}$ all go through the next state $\mathbf{s}_t$. So, direct application of automatic differentiation is out of limits without additional work for MDPs with stochastic transitions.

An effective way to circumvent this difficulty in estimating gradients is to exploit the probability density **reparameterization trick**, which allows to sample a random variable $\mathbf{y} \sim p_\theta(\cdot)$ by transforming it into a deterministic function $\mathbf{y} = \phi(\theta, \xi)$ and using an independent marginal density to sample an auxiliary random variable $\xi \sim p(\cdot)$. For the location-scale family of distributions (e.g., Normal, Gamma, Uniform, Cauchy) the function $\phi(\theta, \xi)$ is given by $\phi(\theta, \xi) = \mu_\theta + \sigma_\theta \cdot \xi$, where μ_θ and σ_θ are functions defining the location and scale of probability density $p_\theta(\cdot)$.

For the purpose of estimating gradients in stochastic computation graphs, the re-parameterization trick is the key ingredient to obtain low-variance estimators of $\nabla_\theta J(\theta)$ (Schulman et al. 2015). Indeed, re-parameterizing the distributions to which the expectation is taken allows us to push the gradient operator into the expectation. Let $J(\theta) = \mathbb{E}_{\mathbf{y} \sim p_\theta}[f(\mathbf{y})]$ be the objective function depending on the function f defined over the stochastic node $\mathbf{y}$ drawn from the distribution p_θ. Then:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\mathbf{y} \sim p_\theta}[f(\mathbf{y})] \\
&= \nabla_\theta \mathbb{E}_{\xi \sim p}[f(\phi(\theta, \xi))] \\
&= \nabla_\theta \int p(\xi) f(\phi(\theta, \xi)) d\xi \\
&= \int p(\xi) \nabla_\theta f(\phi(\theta, \xi)) d\xi \\
&= \mathbb{E}_{\xi \sim p}[\nabla_\theta f(\phi(\theta, \xi))]\,.
\end{aligned}$$

Therefore, after re-parameterizing the distributions, we can estimate the gradient $\nabla_\theta J(\theta)$ by any approximation method applicable to expectations. We refer to $f(\phi(\theta, \xi))$ as the surrogate cost function for $J(\theta)$.

Figure 2 shows the result of the re-parameterization trick for the computation graph in Figure 1. Note that now all paths from the input node θ to the cost node $\hat{V}^{\pi_\theta}$ satisfy

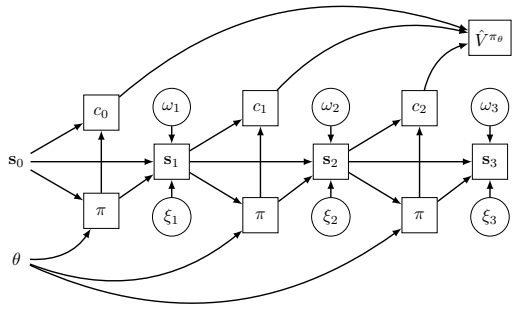

Figure 2: Stochastic computation graph of CSA-MDPs after re-parameterization of the transition distribution: note that for all timesteps $t > 0$ each state $\mathbf{s}_t$ becomes a deterministic function of previous states $\mathbf{s}_{t-1}$, current action $\mathbf{a}_t$, exogenous event ω_t, and the auxiliar noise variable ξ_t.

the property $\theta \prec^D \hat{V}^{\pi_\theta}$. Therefore, automatic differentiation can now be applied to a fully-differentiable cost function.

Deep Reactive Policies

A reactive policy is a model that attempts to explicitly represent a mapping from states to actions. It should be an efficient mechanism to select a valid action given any valid state. Moreover, a DRP is a parametric model built from the stacked layers of feed-forward neural nets.

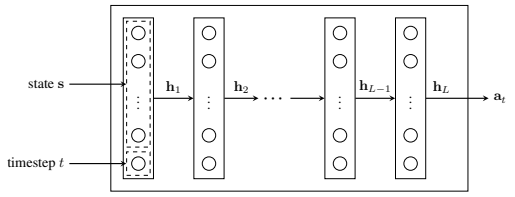

Figure 3: Deep Reactive Policy: the input layer of the policy network corresponds to the current state $\mathbf{s}_t$ and the output layer corresponds to the distribution over actions $\mathbf{a}_t$ (in the case of discrete action spaces) or the action variables themselves (in the case of continuous action spaces).

A **feed-forward neural net** is a parametric function approximator $f_\theta \colon \mathbb{R}^p \to \mathbb{R}^q$ consisting of compositional (hidden) layers $l = 1, \ldots, L$ that implement affine transformations $\mathbf{z}^{(l+1)} = W^{(l)} \cdot \mathbf{h}^{(l)} + b^{(l)}$ interleaved with nonlinear functions $\mathbf{h}^{(l+1)} = f^{(l+1)}(\mathbf{z}^{(l+1)})$. Vectors $\mathbf{z}^{(l)}$ and $\mathbf{h}^{(l)}$ are called logits and activations, respectively. The set of parameters θ comprise the kernel matrices $W^{(l)}$ and bias vectors $b^{(l)}$. Typical nonlinear functions used are the sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$, the hyperbolic tangent $\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp -x}$, and the rectified linear unit $\mathrm{ReLU}(x) = \max(0, x)$.

Probabilistic Planning as Optimization

In this section, we aim at formulating a probabilistic planning problem as an optimization problem over a stochastic computation graph. Our goal is to optimize the parameters of a DRP in order to minimize an approximation of the expected finite-horizon total cost from the given start state.

Policy Search via Gradient-Based Optimization

The overall approach of policy search via gradient-based optimization is to: (1) explicitly encode a policy using a parametric function π_θ; (2) embed π_θ in a stochastic computation graph approximating $V^{\pi_\theta}(\mathbf{s}_0)$; (3) define a suitable objective function $J(\theta)$; and (4) optimize the parameters via stochastic gradient descent over $J(\theta)$.

Parametric policies Representing a policy by a parametric function can have a huge impact both on the training time and in the capacity of approximating the optimal behavior in the given MDP. A number of options can be used in the context of policy search via gradient-based optimization. Usual choices vary from simple linear models to radial basis functions and neural nets. However, the only requirements are that the policy must be quickly evaluated to select an action and that the function's outputs must be differentiable w.r.t. its parameters for all states.

Objective function In principle, any non-decreasing function of $\sum_t C(\mathbf{s}_t, \pi_\theta(\mathbf{s}_t))$ could be used as the objective function $J(\theta)$ so as to minimize $V^{\pi_\theta}(\mathbf{s}_0)$. In this work, we shall use the Mean-Square Error (MSE):

$$J(\theta) = \mathbf{MSE}[V^{\pi_\theta}(\mathbf{s}_0), 0] = \mathbb{E}\left[\left(\sum_{t=0}^{H-1} C(\mathbf{s}_t, \pi_\theta(\mathbf{s}_t))\right)\right]^2. \tag{1}$$

The MSE objective function can considerably accelerate the optimization process as the gradients of the objective function w.r.t. the policy parameters will have their norms conveniently scaled up in the beginning of the training when the total cost is most-likely far from its minimum value. Moreover, we shall use Monte-Carlo sampling as a tractable approximation of the expectation in the objective function:

$$J(\theta) \approx \hat{J}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{t=0}^{H-1} C(\mathbf{s}_{i,t}, \pi_\theta(\mathbf{s}_{i,t}))\right)^2, \tag{2}$$

where i is the index of the state-action trajectory, N is the number of trajectories, t is the timestep and H is the horizon.

Stochastic gradient descent In order to optimize the policy parameters θ in the stochastic computation graph, we shall use an optimization method based on stochastic gradient descent:

$$\theta_{(k+1)} = \theta_{(k)} - \eta \, \nabla_\theta J(\theta)|_{\theta=\theta_{(k)}}, \tag{3}$$

where η is the learning rate.

A number of modern optimizers have been developed in recent years (Ruder 2016). For the purposes of the experiments, we used the adaptive optimizer **RMSProp** that implements momentum by dividing the learning rate η by an exponentially decaying average of squared gradients, thus maintaining independent learning rates for each parameter:

$$\mathbb{E}[g^2]_{(k)} = \gamma \mathbb{E}[g^2]_{(k-1)} + (1-\gamma) g^2_{(k)} \tag{4}$$

$$\theta_{(k+1)} = \theta_{(k)} + \frac{\eta}{\sqrt{\mathbb{E}[g^2]_{(k)} + \epsilon}}\, g_{(k)}, \tag{5}$$

where $g = \nabla_\theta J(\theta)$ and ϵ is a small non-negative smoothing term that avoids division by zero.

Deep Reactive Policies for Continuous Spaces

Currently, most works on DRPs in model-known planning have focused on discrete stochastic policies (Issakkimuthu, Fern, and Tadepalli 2018; Toyer et al. 2018; Groshev et al. 2018). Instead, we aim here at learning continuous deterministic DRPs. Even though the general idea of an efficient action selection mechanism enabling fast decision-making remains the same, some special care is needed in order to extend DRPs to continuous domains.

Action fluent constraints It is common in continuous action spaces to constrain the valid values of the action fluents. Therefore, at the very least, a DRP should be able to accommodate action fluents whose range are either an open, a closed or a half-open/half-closed interval of the reals. An alternative to constrain $\mathbf{a}_t$ to be within the bounds $[\mathbf{l}, \mathbf{u}]$ is to apply a transformation on the logits z^L of the output layer of the DRP such that:

$$\mathbf{a}_t = \begin{cases} \mathbf{l} + (\mathbf{u} - \mathbf{l})\, \sigma(z^{(L)}), & \text{if } \mathbf{l} > -\infty \text{ and } \mathbf{u} < +\infty \\ \mathbf{l} + \exp(z^{(L)}), & \text{if } \mathbf{l} > -\infty \text{ and } \mathbf{u} = +\infty \\ \mathbf{u} - \exp\left(-z^{(L)}\right), & \text{if } \mathbf{l} = -\infty \text{ and } \mathbf{u} < +\infty \\ z^{(L)} & \text{otherwise,} \end{cases} \tag{6}$$

where $\sigma(x)$ is the sigmoid function and $\mathbf{l}$ and $\mathbf{u}$ denote lower and upper bound vectors, respectively.

Layer normalization In addition to handle continuous outputs representing continuous action fluents, a DRP in continuous domains shall pay special attention to the range of the state fluents. It is a well-known issue in training neural nets that the distribution of a layer's activation can vary widely as a function of its previous inputs, which can significantly slow down the training, especially in the case of saturating nonlinearities. This phenomenon often referred to as internal covariate shift can be particularly pronounced in DRPs for continuous spaces given that state fluents can vary by many orders of magnitude, which is in direct contrast to discrete domains that typically deal with state boolean variables. Recently, layer normalization (Ba, Kiros, and Hinton 2016) addressed this issue by normalizing the summed inputs of a given layer.[2] Its effects are to re-center and re-scale the inputs by using simple statistics of the distribution of activations:

$$h^{(l)} = f\left(\frac{g}{\sigma^{(l)}}(z^{(l)} - \mu^{(l)}) + b\right), \tag{7}$$

where g and b the gain and bias parameters of the same dimension as $h^{(l)}$, and $\mu^{(l)}$ and $\sigma^{(l)}$ are the empirical mean and standard deviation of the pre-activations $z^{(l)}$.

[2]We only used layer normalization at the input layer. Experimentally, we saw no benefit to apply it to all DRP's layers.

Experiments

Benchmark domains We extended three domains previously proposed (e.g., Navigation (Faulwasser and Findeisen 2009), HVAC (Heating, Ventilation and Air Conditioning) (Agarwal et al. 2010), and Reservoir Control (Yeh 1985)) in order to incorporate stochastic transitions and additional nonlinearities with the overall goal of making the problems more appealing and challenging to our proposed approach.

Navigation is a path planning problem in a 2-dimensional space in which an agent is supposed to get to a goal position from a start position as fast as possible while avoiding deceleration zones. The position of the agent at timestep t is given by the state variable $\mathbf{p}_t \in \mathbb{R}^2$. The agent's movement is given by the action fluent $\mathbf{a}_t \in [-1, 1]^2$. The dynamics of the agent's movement is defined by:

$$\lambda = \prod_j \frac{2}{1 + \exp(-\alpha_j ||\mathbf{p}_t - \mathbf{c}_j||_2)} - 1\,,$$

$$\mathbf{p}_{t+1} \sim \text{Normal}(\mu = \mathbf{p}_t + \lambda \mathbf{a}_t, \sigma = \frac{\sigma_{\text{max}}}{\sqrt{2}} ||\mathbf{a}_t||)\,,$$

where each deceleration zone j is characterized by its center position $\mathbf{c}_j$ and decay constant α_j, and its effect depends upon the Euclidean distance between its center and the agent's position. The joint deceleration factor λ is given by the multiplicative effect of each independent deceleration zone. The cost function is simply the Euclidian distance from the current position to the goal position:

$$C_t = ||\mathbf{p}_t - \mathbf{g}||_2\,.$$

HVAC is a centralized continuous decision problem in which the objective is to control the temperature of multiple rooms within a comfortable interval subject to energy costs. The transition dynamics is defined by the nonlinear heat transfer through walls between adjacent spaces (e.g., rooms, hallway, or outside area). Each room i has a state variable $\delta^i_t \in \mathbb{R}$ denoting its temperature at timestep t. The action $a^i_t \in [0, a_{\text{max}}]$ corresponds to the volume of heated air sent via vent actuation to room i at each timestep t. The transition function of the problem is given by the following dependencies:

$$\delta^{\text{outside}}_t \sim \text{Normal}(\mu_{\text{out}}, \sigma_{\text{out}})\,,$$

$$\delta^i_{t+1} = \delta^i_t + \text{Normal}(\mu_a = a^i_t(\delta_a - \delta^i_t), \sigma_a) + \sum_{\text{ADJ}(i,j)} \text{Normal}(\mu_{ij} = (\delta^j_t - \delta^i_t)^3 / r_{ij}, \sigma_{ij})\,,$$

where δ_a is the temperature of the air being sent by the central actuator and r_{ij} is the thermal resistance between room i

and the adjacent space j. Standard deviations σ_a and σ_{ij} are constants of the domain. The cost function takes into consideration the temperature comfort zone $[l^i, u^i]$ of each room i and the energy cost k:

$$C_t = \sum_i \left| \delta_t^i - \frac{(l^i + u^i)}{2} \right| + k\mathbf{a}_t^i \,.$$

Reservoir Control is a system to maintain the level of interconnected water reservoirs within a nominal range. Each reservoir i has a state fluent denoting its level $l_t^i \in [0, \mathrm{l}_{\mathrm{max}}^i]$ and a corresponding action fluent a_t^i related to the water flow to a downstream reservoir (or to the sea). Note that even though a given reservoir has a single downstream reservoir, it can have zero or more upstream reservoirs. The dynamics of the system is given by the following dependencies:

$$\begin{aligned} r_t^i &\sim \text{Gamma}(k, \alpha)\,, \\ e_t^i &= 0.5 \log(l_t^i + 1) \left(\frac{l_t^i}{\mathrm{l}_{\mathrm{max}}^i} \right)^2 , \\ l_{t+1}^i &= \max\Big(0,\, l_t^i + r_t^i - e_t^i - a_t^i + \sum_{\text{DOWN(j, i)}} a_t^j \Big)\,, \end{aligned}$$

where r_t^i and e_t^i denote the rain and evaporation over reservoir i, and DOWN(j, i) is a topology predicate indicating reservoir i is the downstream reservoir of j. The cost function is such that a penalty is given for each reservoir out of its nominal range:

$$C_t = \sum_i \begin{cases} \delta_{\text{lower}}(l_{\text{min}}^i - l_t^i)^2 & \text{if } l_t^i < l_{\text{min}}^i\,, \\ \delta_{\text{upper}}(l_{\text{max}}^i - l_t^i)^2 & \text{if } l_t^i > l_{\text{max}}^i\,, \\ 0 & \text{otherwise.} \end{cases}$$

DRP architectures As there is an unbounded number of architectures to investigate, we set ourselves to evaluate two architectures representatives of distinct classes of models. The first DRP, denoted "tf-mdp (1)", is a shallow, but wide model. The second model, denoted "tf-mdp (2)", is a deep, but narrow model. Table 1 shows the number of hidden layers and units in each layer. In both models, we use the ELU activation function (Clevert, Unterthiner, and Hochreiter 2015) and layer normalization (Ba, Kiros, and Hinton 2016) in the input layer. Table 2 shows the number of parameters for each model/architecture.

DRP	Hidden Layers	Number of Units
tf-mdp (1)	1	2048
tf-mdp (2)	4	256, 128, 64, 32

Table 1: DRP architectures

Implementation We implemented tf-mdp in TensorFlow (Abadi et al. 2016).[3] We specified the domains/instances using RDDL (Relational Dynamic Influence Diagram Language) (Sanner 2010) and compiled the models to stochastic computation graphs in TensorFlow using a compiler specifically built for this work.[4] We conducted all experiments on a single 2.4 GHz Intel Core i5 8GB RAM machine.

Domain	tf-plan	tf-mdp (1)	tf-mdp (2)
Nav2	40	10,246	44,070
HVAC3	120	14,345	44,361
HVAC6	240	26,642	45,234
Res10	400	43,038	46,398
Res20	800	84,028	49,308
Res30	1,200	125,018	52,218

Table 2: Number of parameters per domain/instance

Methodology Training neural nets and especially deep neural nets such as DRPs can be especially sensitive to the choice of training hyperparameters (e.g, learning rate, batch size, number of training epochs). Our objective with the experiments is not necessarily to achieve the best possible outcome by carefully fine-tuning hyperparameters, but instead to provide a reasonable comparison between the models. Hence, we selected the sensible default values shown in Table 3 and fix them for all training runs.

Domain	Batch	Learning rate	Epochs	Horizon
Nav	256	0.001	200	20
HVAC	256	0.0001	200	40
Res	256	0.001	200	40

Table 3: Training hyperparameters for tf-mdp

We report average and standard deviation values over 10 training runs for each model/architecture in terms of quality of solution (i.e., average total cost from start state) and computational times. Additionally, in order to avoid hazardous fluctuations and divergences during training, we keep the best policy so far as a way to smooth out the gradient-based policy search.

We compare the results of "tf-mdp (1)" and "tf-mdp (2)" to an online extension of tf-plan implemented as an open-loop feedback controller that attempts to optimize the average of sampled state-action trajectories. We run tf-plan for 10 and 25 training epochs per timestep, denoted "tf-plan (10)" and "tf-plan (25)", respectively. Note that 10 and 25 training epochs per step of tf-plan corresponds to 4 and 10 times the total number of training epochs given to tf-mdp. We benchmark the results for the domains/instances: Nav2 (Navigation with 2 deceleration zones), HVAC3 (3 rooms) and HVAC6 (6 rooms), and Res10/20/30 (Reservoir Control with 10, 20 and 30 reservoirs, respectively).

Results Figures 4, 5 and 6 show that DRPs can achieve comparable or better results than the online tf-plan planner. Additionally, we notice that tf-mdp (2) (i.e., deep and narrow DRP) performed better in Navigation and HVAC problems.

[3] https://github.com/thiagopbueno/tf-mdp

[4] https://github.com/thiagopbueno/rddl2tf

However, tf-mdp (1) (i.e., shallow and wide DRP) was able to achieve better results for the number of training epochs pre-defined in Reservoir instances, even though it seems that tf-mdp (2) would be able to catch up if more training epochs were given. Also, we observe that for bigger problems in the Reservoir Control domain (as shown in Figure 6), both architectures presented a considerable variance across different training runs.

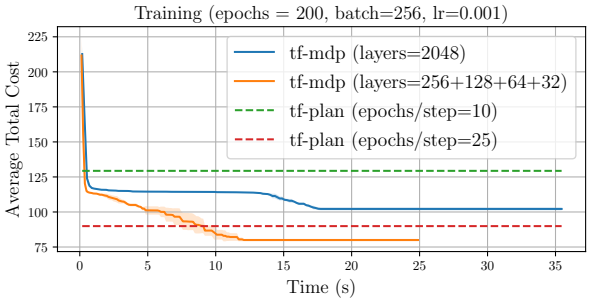

Figure 4: Navigation2: average total cost vs. training time

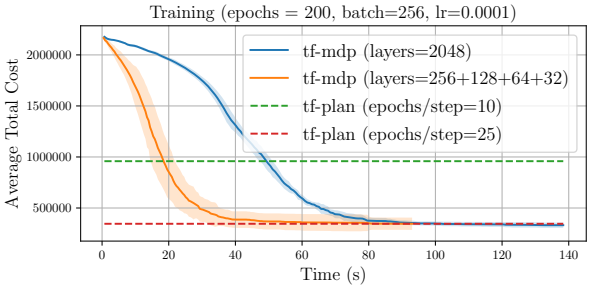

Figure 5: HVAC3: average total cost vs. training time

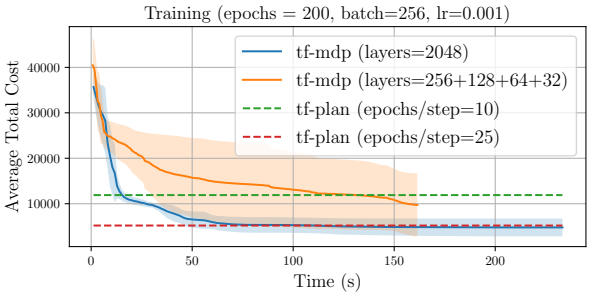

Figure 6: Reservoir20: average total cost vs. training time

Tables 4 and 5 show the empirical time estimates of inference (i.e., average time to compute the next action) and total computing time for each model and domain instance. We can observe that the average inference time of tf-mdp is within a fraction of the time required by the online planner tf-plan, which empirically validates the claim that DRPs can be useful for fast decision-making. Note that tf-mdp is on average 5 orders of magnitude faster than tf-plan in terms of inference time. Also, even when amortizing the offline computational cost over the timesteps of the horizon, tf-mdp remains competitive in terms of total computing time.

Domain	tf-plan (10)	tf-plan (25)	tf-mdp (1)	tf-mdp (2)
Nav2	1.2 ± 0.2	1.7 ± 0.3	$2.4 \cdot 10^{-5}$	$1.8 \cdot 10^{-5}$
HVAC3	3.2 ± 0.3	4.5 ± 0.5	$2.3 \cdot 10^{-5}$	$1.6 \cdot 10^{-5}$
HVAC6	3.6 ± 0.5	5.1 ± 1.3	$3.2 \cdot 10^{-5}$	$1.9 \cdot 10^{-5}$
Res10	2.9 ± 0.5	4.1 ± 0.2	$3.2 \cdot 10^{-5}$	$2.6 \cdot 10^{-5}$
Res20	4.8 ± 0.3	7.3 ± 2.2	$3.8 \cdot 10^{-5}$	$3.2 \cdot 10^{-5}$
Res30	7.4 ± 1.5	14.6 ± 6.1	$5.5 \cdot 10^{-5}$	$4.1 \cdot 10^{-5}$

Table 4: Average inference time per step (sec)

Domain	tf-plan (10)	tf-plan (25)	tf-mdp (1)	tf-mdp (2)
Nav2	26.6 ± 1.9	34.2 ± 2.3	35.4 ± 1.7	24.8 ± 1.0
HVAC3	126.4 ± 2.8	183.6 ± 4.7	138.3 ± 2.2	93.0 ± 2.5
HVAC6	138.9 ± 3.7	198.8 ± 6.5	150.5 ± 1.7	106.4 ± 2.7
Res10	117.7 ± 2.8	164.0 ± 4.7	190.7 ± 3.4	134.9 ± 4.5
Res20	194.4 ± 3.6	295.6 ± 5.6	233.2 ± 4.7	161.4 ± 2.7
Res30	296.3 ± 4.5	587.7 ± 5.8	344.9 ± 3.1	244.1 ± 3.7

Table 5: Total computing time (sec)

Conclusion

We investigated a promising approach for optimizing DRPs by leveraging non-convex optimization techniques commonly used in Deep Learning, but not frequently applied to model-known planning. We presented a gradient-based optimization approach for training DRPs that exploits the re-parameterization of the stochastic computation graph of MDPs. More general than previous approaches, we focused on continuous stochastic domains with concurrent actions and exogenous events exhibiting nonlinear transition and cost functions. We showed that training large DRPs with hundred of thousands of continuous action parameters can be carried out within minutes without the need of high-performance hardware. Finally, comparing the DRPs trained by our approach with online state-of-the-art gradient-based planners, we observed a speedup of several orders of magnitude on the time to select actions, which highlights the potential of DRPs for fast decision-making in continuous domains. As future work, we shall seek efficient ways to extend gradient-based policy search to hybrid (mixed discrete and continuous) domains for which more sophisticated methods for gradient backpropagation over the stochastic computation graph are required.

Acknowledgments

We thank the anonymous reviewers for their thoughtful comments on the draft version of this paper. This work was partially supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, CNPq (grants 870666/1998-3, 308433/2014-9, 303920/2016-5, 420669/2016-7), and FAPESP (grant 2015/01587-0).

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Aberdeen, D. 2005. Policy-Gradient Methods for Planning. In *Advances in Neural Information Processing Systems 18, NIPS 2005*, 9–16.

Agarwal, Y.; Balaji, B.; Gupta, R.; Lyles, J.; Wei, M.; and Weng, T. 2010. Occupancy-driven energy management for smart building automation. In *Proceedings of the 2nd ACM workshop on embedded sensing systems for energy-efficiency in building*, 1–6. ACM.

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research* 11:1–94.

Buffet, O., and Aberdeen, D. 2007. FF + FPG: guiding a policy-gradient planner. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS 2007*, 42–48.

Clevert, D.; Unterthiner, T.; and Hochreiter, S. 2015. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *CoRR* abs/1511.07289.

Faulwasser, T., and Findeisen, R. 2009. Nonlinear model predictive path following control. *Nonlinear Model Predictive Control* 384:335–343.

Griewank, A., and Walther, A. 2008. *Evaluating derivatives: principles and techniques of algorithmic differentiation*, volume 105. Siam.

Groshev, E.; Goldstein, M.; Tamar, A.; Srivastava, S.; and Abbeel, P. 2018. Learning Generalized Reactive Policies Using Deep Neural Networks. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018*, 408–416.

Hinton, G.; Deng, L.; Yu, D.; Dahl, G. E.; Mohamed, A.-r.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T. N.; et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* 29(6):82–97.

Issakkimuthu, M.; Fern, A.; and Tadepalli, P. 2018. Training Deep Reactive Policies for Probabilistic Planning Problems. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018*, 422–430.

Keller, T., and Helmert, M. 2013. Trial-Based Heuristic Tree Search for Finite Horizon MDPs. In *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

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.

Ng, A. Y., and Jordan, M. 2000. PEGASUS: A policy search method for large MDPs and POMDPs. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, 406–415. Morgan Kaufmann Publishers Inc.

Puterman, M. L. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Raghavan, A.; Sanner, S.; Khardon, R.; Tadepalli, P.; and Fern, A. 2017. Hindsight Optimization for Hybrid State and Action MDPs. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, 2017*, 3790–3796.

Ruder, S. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Sanner, S. 2010. Relational Dynamic Influence Diagram Language (RDDL): Language Description. `http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf`.

Schulman, J.; Heess, N.; Weber, T.; and Abbeel, P. 2015. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015*, 3528–3536.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T. P.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112.

Toyer, S.; Trevizan, F. W.; Thiébaux, S.; and Xie, L. 2018. Action Schema Networks: Generalised Policies with Deep Learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, 2018*.

Wu, G.; Say, B.; and Sanner, S. 2017. Scalable planning with tensorflow for hybrid nonlinear domains. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, 6276–6286.

Yeh, W. W.-G. 1985. Reservoir management and operations models: A state-of-the-art review. *Water resources research* 21(12):1797–1818.

Yoon, S. W.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic Planning via Determinization in Hindsight. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, 1010–1016.