Reinforcement Learning with Function-Valued Action Spaces for Partial Differential Equation Control

Yangchen Pan [1 2] **Amir-massoud Farahmand** [3 2] **Martha White** [1] **Saleh Nabi** [2] **Piyush Grover** [2] **Daniel Nikovski** [2]

Abstract

Recent work has shown that reinforcement learning (RL) is a promising approach to control dynamical systems described by partial differential equations (PDE). This paper shows how to use RL to tackle more general PDE control problems that have continuous high-dimensional action spaces with spatial relationship among action dimensions. In particular, we propose the concept of *action descriptors*, which encode regularities among spatially-extended action dimensions and enable the agent to control high-dimensional action PDEs. We provide theoretical evidence suggesting that this approach can be more sample efficient compared to a conventional approach that treats each action dimension separately and does not explicitly exploit the spatial regularity of the action space. The action descriptor approach is then used within the deep deterministic policy gradient algorithm. Experiments on two PDE control problems, with up to 256-dimensional continuous actions, show the advantage of the proposed approach over the conventional one.

1. Introduction

This paper develops an algorithmic framework for handling reinforcement learning (RL) problems with high-dimensional action spaces. We are particularly interested in problems where the dimension of the action space is very large or even infinite. These types of problems naturally appear in the control of Partial Differential/Difference Equations (PDE), which have attracted attention because of their potential applications spreading over physical dynamic system (Lions, 1971) and engineering problems, including the design of air conditioning systems (Popescu et al., 2008), modelling of flexible artificial muscles with many degrees of freedom (Kim et al., 2013), traffic control (Richards, 1956; Lighthill & Whitham, 1955), and the modelling of information flow in social networks (Wang et al., 2013).

Many dynamical systems can be described by a set of Ordinary Differential Equations (ODE). Some examples are the dynamics describing inverted pendulum, robotic arm manipulator (with inflexible joints), and electrical circuits (in low-frequency regime for which the electromagnetic radiation is negligible). The property of these systems is that their state can be described by a finite dimensional variable. The use of RL to control ODE-based problems, with either discretized action or continuous actions, has been widely presented in various RL works (Sutton & Barto, 1998; Kober et al., 2013; Deisenroth et al., 2013). The success is most significant for problems where the traditional control engineering approaches may not fare well, due to the complexity of the dynamics, either in the form of nonlinearity or uncertainty. There are, however, many other physical phenomena that cannot be well-described by an ODE, but can be described by a PDE. Examples are the distribution of heat in an object as a function of time and location, motion of fluids, and electromagnetic radiation, which are described by the heat, Navier-Stokes, and Maxwell's equations, respectively. The control of PDEs using data-driven approaches, including RL-based formulation, has just recently attracted attention and is a relatively an unexplored area (Farahmand et al., 2016b; 2017; Belletti et al., 2018; Duriez et al., 2016).

Control of PDEs has been investigated in conventional control engineering (Krstic & Smyshlyaev, 2008; Ahuja et al., 2011; Borggaard et al., 2009; Burns et al., 2016; Burns & Hu, 2013; Brunton & Noack, 2015). Despite the mathematical elegance of conventional approaches, they have some drawbacks that motivate investigating learning-based methods, in particular RL-based approaches. Many conventional approaches require the knowledge of the PDE model, which might be difficult to obtain. Designing a con-

[1]Department of Computing Science, University of Alberta, Edmonton, Canada [2]Mitsubishi Electric Research Laboratories (MERL), Cambridge, USA [3]Vector Institute, Toronto, Canada. Correspondence to: Yangchen Pan <pan6@ualberta.ca>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

troller for the model also requires a control engineer with an expertise in modeling and control of PDEs, which makes it even more challenging. Furthermore, a hand-designed controller is brittle to changes in the geometry and parameters of the PDE, requiring repeated redesign of the controller to maintain performance. This is impractical in many industrial applications, such as an air conditioning system that is deployed to a customer's home. Moreover, many of the controller design tools are based on a linear PDE assumption, which ignores potentially useful nonlinear phenomena inherent in PDEs, such as those in fluid dynamics problems (Foures et al., 2014). Designing a controller based on this simplified model might lead to a suboptimal solution. The optimal controller is also typically designed for the quadratic cost function, as opposed to a more general objective, such as a user's comfort level for an air conditioning system.

It is desirable to have a controller design procedure that has minimal assumptions, does not require the knowledge of the PDE, and is completely data-driven. This can be achieved by formulating the PDE control problem as an RL problem, as has recently been shown (Farahmand et al., 2016b; 2017), or some other flexible data-driven approaches such as the genetic programming-based method of Duriez et al. (2016).

The PDE control problem is challenging because the state of a PDE is in theory an infinite-dimensional vector and very high-dimensional in computer simulations. Moreover, many PDE control problems have infinite-dimensional continuous action, i.e., function-valued action. For example, if the PDE is controlled through its boundary condition, the action space is a function space defined over the continuous boundary. Even though one may spatially discretize the action space and treat it as a multi-dimensional action space, such a solution leads to a very high-dimensional continuous action space and does not explicitly incorporate the spatial regularity of the PDE's action space.

Previous work has addressed the problem of high-dimensionality of the state space in the RL-based PDE control (Farahmand et al., 2016b; 2017), but their solutions have been limited to PDEs with a finite number of actions. This work focuses on the infinite-dimensionality of the action space. We formulate the PDE control with RL within the Markov Decision Process (MDP) formalism with function-valued state and action spaces (Section 2). We introduce *action descriptors* as a generic method to model actions in PDE control problems (Section 3). Action descriptors allow us to scale to arbitrarily high dimensional continuous action spaces and capture the spatial regularities among action dimensions. By benefiting from action descriptors, we propose a neural network architecture that is not changed with the increase of action dimensions; rather, it simply queries from a set of action descriptors (Section 4). This is in contrast with conventional RL algorithms that directly output a high-dimensional action *vector*. We provide some theoretical insights on why the proposed approach might have a better sample complexity through a covering number argument (Section 5). Finally, we empirically verify the effectiveness of our architecture on two PDE control domains with up to 256 dimensional continuous actions (Section 6).

2. Reformulating PDE Control as an MDP

In this section, we first provide a PDE control example and briefly discuss how this can be viewed as an MDP. We also highlight the need to exploit the spatial regularities in the action space of PDEs. For more discussion, refer to Farahmand et al. (2016b).

2.1. Heat Invader: A PDE Example

PDE control is the problem of modifying the behaviour of a dynamical system that is described by a set of PDEs. An example of a PDE is the convection-diffusion equation, which describes the changes in the temperature in an environment, among several other other physical phenomena. Given a domain $\mathcal{Z} \subset \mathbb{R}^d$ and time t, the temperature at location $z \in \mathcal{Z}$ at time t is denoted by the scalar field $T(z,t)$. The convection-diffusion equation describes the evolution of the temperature as follows:

$$\frac{\partial T}{\partial t} = \nabla \cdot \frac{1}{P_e} \nabla T - \nabla \cdot (vT) + S. \tag{1}$$

Here $S = S(z,t)$ is the heat source/sink, $v = v(z,t)$ is the velocity field describing the airflow in the domain, and $\frac{1}{P_e}$ is a diffusivity constant. The gradient operator is ∇, and the divergence operator is $\nabla\cdot$, which measures the outflow of a vector field. These operators are with respect to (w.r.t.) z. We can further decompose the source term $S(z,t) = S(z,t;a) = S^o(z,t) + a(z,t)$, where $S^o(z,t)$ is a source term that cannot be controlled (e.g., a disturbance) and $a(z,t)$ is the action under our control. The Heat Invader problem is a particular example of this convection-diffusion PDE over a 2D domain $\mathcal{Z}$ with a time-varying source $S^o(z,t)$ (Farahmand et al., 2016b).

Since the location space $\mathcal{Z}$ is a continuous domain, the action $a(\cdot,t)$ is a function lying in some function space. The dimension of this action space, however, depends on how we actuate the PDE. As a concrete example, if we can control the temperature of the airflow going through an inlet in an air conditioning system, the action is the scalar temperature of the inflow air and the action space is one-dimensional real-valued vector space (assuming we can control the temperature with arbitrary precision). But we may also be able to finely control the temperature of walls

of a room by a distributed set of minuscule heaters/coolers within the wall (this technology might not exist today, but is perceivable). In this case, the action is the temperature of each tiny heater/cooler in the wall, so the action space would be an extremely high dimensional vector space.

Generally, one may control a PDE by setting its boundary condition or by defining a source/sink within its domain. The boundary of a PDE has a dimension one smaller than the domain of a PDE (under certain regularities of the geometry of the boundary). So for a 3D PDE (e.g., a room and its temperature field), the boundary is a 2D domain (e.g., the walls). An action defined over the boundary, without further constraint, is a function with a domain that is uncountably large. A similar observation holds when we control a source/sink within the domain.

2.2. PDE Control as an MDP

To relate the PDE control problem to the reinforcement learning setting, we first provide a brief overview of MDP (Sutton & Barto, 1998; Szepesvári, 2010; Bertsekas, 2013). An MDP consists of $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{X}$ denotes the state space, $\mathcal{A}$ denotes the action space, $\mathcal{P} : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to \mathcal{M}(\mathcal{X})$ is the transition probability kernel (with $\mathcal{M}(\mathcal{X})$ being the space of probability distributions defined over $\mathcal{X}$), $\mathcal{R} : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to \mathcal{M}(\mathbb{R})$ is the reward distribution, and $\gamma \in [0, 1)$ is the discount factor. At each discrete time step $t = 1, 2, 3, ...$, the agent selects an action according to some policy π and the environment responds by transitioning into a new state x_{t+1} sampled from $\mathcal{P}(\cdot|x_t, a_t)$, and the agent receives a scalar reward r_{t+1} samples from $\mathcal{R}(\cdot|x_t, a_t, x_{t+1})$. The agent's goal is to maximize discounted cumulative sum of rewards from the current state x_t. Typically in the RL literature, both the state space and action spaces are finite dimensional vector spaces, for example both are subsets of an Euclidean space, $\mathbb{R}^p$ and $\mathbb{R}^k$, respectively. The state of a PDE, however, is an infinite-dimensional vector space. Thus to describe such problems, we need to work the MDPs with infinite dimensional state and action spaces.[1]

Consider now the PDE control problem, and how it can be formalized as an agent finding an optimal policy for an MDP $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. In (1), the state and action are infinite dimensional vectors (functions) $x_t = T(\cdot, t)$ and $a_t = a(\cdot, t)$. For example in the Heat Invader problem, the state is the current temperature at all locations, and the action changes the temperature at each location. Both state and action are functions belonging to some function space defined over the domain $\mathcal{Z}$ of PDE, e.g., the space of continuous function $\mathcal{C}(\mathcal{Z})$, the Sobolev space, etc. The particular choice of function space depends on the PDE and its regularity; we denote the space by $\mathcal{F}(\mathcal{Z})$.

The dynamics of the MDP is a function of the dynamics of PDE. One difference between the MDP framework and PDE is that the former describes a discrete-time dynamical system whereas the latter describes a continuous-time one. One may, however, integrate the PDE over some arbitrary chosen time step Δ_t to obtain a corresponding partial *difference* equation $x_{t+1} = f(x_t, a_t)$ for some function $f : \mathcal{F}(\mathcal{Z}) \times \mathcal{F}(\mathcal{Z}) \to \mathcal{F}(\mathcal{Z})$, which depends on the PDE. For the convection-diffusion PDE (1) with the state being denoted by $x \in \mathcal{F}(\mathcal{Z})$ (instead of temperature T), we have $f(x, a) = \int_{t=0}^{\Delta_t} \nabla \cdot \frac{1}{P_e} \tilde{x} - \nabla \cdot (v\tilde{x}) + S(z, t; a) \mathrm{d}t$ with the initial state of $\tilde{x}$ at $t = 0$ being x and the choice of $S(z, t; a)$ depending on a.[2] So we might write $\mathcal{P}(x|x_t, a_t) = \delta(x - f(x_t, a_t))$, where $\delta(\cdot)$ is the Dirac delta function. More generally, if there is stochasticity in the dynamics, for example if the constants describing the PDE are random, the temporal evolution of the PDE can be described by a transition probability kernel, i.e., $x_{t+1} \sim \mathcal{P}(\cdot|x_t, a_t)$.

The remaining specification of the MDP, including the reward and the discount factor, is straightforward. For example, the reward function in the Heat Invader problem is designed based on the desire to keep the room temperature at a comfortable level while saving energy:

$$r(x_t, a_t, x_{t+1}) = -\text{cost}(a_t) - \int_{z \in \mathcal{Z}} \mathbb{I}(|T(z, t+1)| > T^*(z, t+1)) \mathrm{d}z, \tag{2}$$

where $\mathbb{I}(\cdot)$ is an indicator function, $T^*(\cdot)$ is a predefined function describing the acceptable threshold of deviation from a comfortable temperature (assumed to be 0), and $\text{cost}(\cdot)$ is a penalty for high-cost actions. Refer to Appendix B.3 for more detail.

Reinforcement learning algorithms, however, are not designed to learn with infinite-dimensional states and actions. We can overcome this problem by exploiting the spatial regularities in the problem, and beyond PDEs, exploiting general regularities between the dimensions of the states as well as actions. We provide more intuition for these regularities before introducing MDPs with action-descriptors, a general subclass of infinite-dimensional MDPs that provides a feasible approach to solving these infinite-dimensional problems.

[1] The discounted MDP framework works fine with general state spaces, under certain measurability conditions, cf., Sections 5.3 and 5.4 and Appendix C of Bertsekas (2013). The conditions would be satisfied for bounded Borel measurable reward function and the Borel measurable stochastic kernel.

[2] Note that this step requires the technical requirement of the existence of the solution of a PDE, which has not been proven for all PDEs, e.g., the Navier-Stokes equation. Also we assume that the action remains the same for that time period.

2.3. Exploiting Spatial Regularities in PDEs

One type of regularity particular to PDEs is the spatial regularity of their state. This regularity becomes apparent by noticing that the solution of a PDE, which is typically a 1D/2D/3D scalar or vector field, is similar to a 1D/2D/3D image in a computer vision problem. This similarity has motivated some previous work to design RL algorithms that directly work with the PDE's infinite dimensional state vector, or more accurately its very high-dimensional representation on a computer, by treating the state as an image (Farahmand et al., 2016b; 2017). Farahmand et al. (2016b) suggest using the Regularized Fitted Q-Iteration (RFQI) algorithm (Farahmand et al., 2009) with a reproducing kernel Hilbert space (RKHS) as a value function approximator. For that approach, one only needs to define a kernel between two image-like objects. Farahmand et al. (2017) suggest using a deep convolution network (ConvNet) as the estimator of the value function. ConvNets are suitable to exploit spatial regularities of the input and can learn domain-specific features from image-like inputs.

Even though these previous approaches can handle high-dimensional states in PDE control problems, they are limited to a finite number of actions. Their approach does not exploit the possible regularities in the *action space* of a PDE. For example, some small local changes in a controllable boundary condition may not change the solution of the PDE very much. In that case, it makes sense to ensure that the actions of nearby points on the boundary be similar to each other. The discretization-based approach (Farahmand et al., 2016b; 2017) ignores the possibility of having a spatial regularity in the action space. We next describe how we can exploit these regularities—as well as make it more feasible to apply reinforcement learning algorithms to these extraordinarily high-dimensional continuous action problems—by introducing the idea of action descriptors.

3. MDPs with Action Descriptors

We consider an MDP formulation where the state space $\mathcal{X}$ and action space $\mathcal{A}$ can be infinite dimensional vector spaces, e.g., the space of continuous functions over $\mathcal{Z}$. Procedurally, the agent-environment interaction is as usual: at each step, the agent selects a function $a_t \in \mathcal{A}$ at the current state $x_t \in \mathcal{X}$, traverse to a new state x_{t+1} according to the dynamics of the PDE, and receives reward r_{t+1}.

Even though the infinite dimensional state/action space MDPs provide a suitable mathematical framework to talk about control of PDEs, a practically implementable agent may not be able to provide an infinite dimensional action as its output, i.e., providing a value for all uncountably infinite number of points over $\mathcal{Z}$. Rather, it may only select the values of actions at a finite number of locations in $\mathcal{Z}$, and convert those values to an infinite dimensional action appropriate as the control input to the PDE. Consider the Heat Invader problem. There might be a fine, but finite, grid of heater/cooler elements on the wall whose temperature can be separately controlled by the agent. Each of the elements is spatially extended (i.e., each element covers a subset of $\mathcal{Z}$), and together they define a scalar field that is controlled by the agent. The result is an infinite dimensional action, an appropriate control input for a PDE, even though the agent only controls a finite, but possibly very large, number of values. In some cases, the set of controllable locations are not necessarily fixed, and might change. For example, if an air conditioner is moved to some other location which has never placed before, the agent should ideally still be able to control them.

We propose to model this selection of action-dimensions (or the location of where the agent can exert control) using *action descriptors*. For the Heat Invader problem, the action descriptors correspond to the spatial locations z of the air conditioners; more generally, they can be any vector describing an action-dimension in the infinite-dimensional action space. The action descriptors help capture regularities across action dimensions, based on similarities between these vectors.

To be concrete, let $\mathcal{Z}$ be the set of locations in the domain of PDE, e.g., $\mathcal{Z} = [0, 1]^2$ for the 2D convection-diffusion equation (1). An action descriptor is $c \in \mathcal{Z}$ and determines the location where the action can be selected by the agent. We use the more general term action descriptor, rather than actions locations, as in other PDEs, $\mathcal{Z}$ may represent other regularities between actions. The set of all action descriptors is the finite and ordered set $\mathcal{C} = (c_1, \dots, c_k)$, for finite k. Given each action descriptor c_i and the state x, the agent's policy $\pi : \mathcal{X} \times \mathcal{Z} \to \mathbb{R}$ (in case of deterministic policies) generates an action scalar $u^{(i)} \in \mathbb{R}$, i.e, $u^{(i)} = \pi(x, c_i)$. Recalling that the action of a PDE belongs to $\mathcal{F}(\mathcal{Z})$, an infinite dimensional action space, we have to convert these action scalars to an action that can be fed to the underlying PDE. We use an *adapter* for this purpose. An adapter is a mapping from the set of action scalars $u = (u^{(1)}, \dots, u^{(k)})$ and the action descriptors $\mathcal{C}$ to a function defined over $\mathcal{Z}$, i.e., $I : \mathcal{C} \times \mathbb{R}^k \to \mathcal{F}(\mathcal{Z})$. The infinite dimensional action given to the PDE is $a_t = I(\mathcal{C}, u_t)$.

There are many ways to define an adapter, which can be thought of as a decoder from a finite-dimensional code to a function-valued code. Linear interpolators are one particular class of adapters. Given a set of (fixed) weighting functions $w_i : \mathcal{Z} \to \mathbb{R}$, the linear interpolator is

$$I(\mathcal{C}, u) : z \mapsto \sum_{i=1}^{k} w_i(z) u^{(i)}. \tag{3}$$

For instance, we may define the weighting function as

Gaussians with the centres on the action descriptors, i.e., $w_i(z) = \exp(-\frac{\|z-c_i\|^2}{2\sigma^2})$.

Another choice, which we use in our experiments, is based on partitioning of the domain $\mathcal{Z}$ around the action descriptors. Given the action descriptors $\mathcal{C}$, define a partition of $\mathcal{Z}$ and denote it by $(A_1, \ldots, A_k)$, i.e., $\bigcup A_i = \mathcal{Z}$ and $A_i \cap A_j = \emptyset$ for $i \neq j$. For example, A_i might be a rectangular-shaped region in $\mathcal{Z}$ and c_i being its centre. Another example would be Voronoi diagram corresponding to centres in $\mathcal{C}$, which is commonly used in the finite element method. Having this partitioning, we define the weighting functions as $w_i(z) = \mathbb{I}\{z \in A_i\}$. With this choice of adapter, the action given to the MDP is

$$a_t = I(\mathcal{C}, u) = \sum_{i=1}^{k} \mathbb{I}\{z \in A_i\} \pi(x_t, c_i). \tag{4}$$

To build physical intuition, consider the Heat Invader problem again. The action descriptors $\mathcal{C}$ correspond to the locations of the (centre of) heater/cooler elements of the air conditioners. Furthermore, a partition A_i corresponds to the spatially-extended region where each element occupies. When we set the temperature at location c_i to a certain value $u^{(i)}$ in (4), the value of the whole region A_i takes the same value $u^{(i)}$.

The action descriptor formulation allows the agent to exploit the spatial regularity of the action. For example if the variations of the action applied within a subregion $\mathcal{Z}_0 \subset \mathcal{Z}$ does not cause much change in the dynamics, it is sufficient to learn a $\pi(\cdot, c)$ that varies slowly within $\mathcal{Z}_0$. The learning agent that has access to $\mathcal{C}$ and can learn a mapping $\pi(x, c_i)$, as in (3) or (4), can potentially exploit this regularity. This can be contrasted with a non-adaptive interpolation-only agent where the action is chosen as $a_t = \sum_{i=1}^{k} \mathbb{I}\{z \in A_i\} \pi_i(x_t)$. In this scheme we choose the action at each partition without considering their relative locations through a series of separately learned $\pi_i(x)$. Such an agent cannot explicitly benefit from the spatial regularity of the action. We develop this argument more rigorously in Section 5. Another benefit of the action description formulation is that as long as $I(\mathcal{C}, u)$ can work with variable-sized sets $\mathcal{C}$ and u, it allows variable number of spatial locations to be queried. This can be helpful when the physical actuators in the environment are added or removed.

This formulation is a strict generalization of more standard settings in reinforcement learning, such as a finite-dimensional action space $\mathcal{A} = \mathbb{R}^k$. The domain for actions is $\mathcal{Z} = \{1, \ldots, k\}$ and we do not subselect locations, $\mathcal{C} = \mathcal{Z}$, requiring instead that all action dimensions are specified. The adapter simply returns action $\pi_i(x_t)$ for action index i, using $A_i = \{i\}$ and $a_t = \sum_{i=1}^{k} \mathbb{I}\{z \in A_i\} \pi_i(x_t)$.

4. PDE Control with RL Algorithms

There has been some work addressing continuous action spaces, including both with action-value methods (Baird & Klopf, 1993; Gaskett et al., 1999; del R Millán et al., 2002; van Hasselt & Wiering, 2007) and policy-based methods (Schulman et al., 2015; Montgomery & Levine, 2016; Silver et al., 2014; Lillicrap et al., 2016; Schulman et al., 2016). These methods, however, do not scale with extremely high-dimensional action spaces, because they either have to optimize the action-value function over a high-dimensional action space or output a high-dimensional action vector. These approaches also do not explicitly exploit regularities between actions. Some work for high-dimensional discrete (finite) action spaces do extract action embeddings (Sunehag et al., 2015; He et al., 2015; Dulac-Arnold et al., 2015) or impose factorizations on the action space (Sallans & Hinton, 2004; Dulac-Arnold et al., 2012; Pazis & Parr, 2011). These methods, however, are specific to large sets of discrete actions. Existing methods cannot be directly applied to learning for these (extremely) high-dimensional continuous action problems with regularities. Next we discuss how to modify a policy gradient method to extend to this setting.

For our MDP problem formulation with action descriptors, we propose to learn a policy function that receives the state along with action descriptors and outputs actions or probabilities over actions. Consider the policy parameterization $\pi_{\theta^\mu} : \mathcal{X} \times \mathcal{Z} \to \mathbb{R}$. For a given state x_t, under the infinite-dimensional MDP formalism, the selected action $a_t = \pi_{\theta^\mu}(x_t, \cdot)$ which is function-valued. With the action descriptors set $\mathcal{C}$, the policy outputs the ith action component by evaluating $\pi_{\theta^\mu}(x_t, c_i), c_i \in \mathcal{C}$ and hence we are able to get action scalars $u_t \in \mathbb{R}^{|\mathcal{C}|}$. Although a distribution over such functions could be maintained, for simplicity in this preliminary work, we focus on deterministic policies.

The Deterministic Policy Gradient algorithm (Lillicrap et al., 2016) provides a method to learn such a policy. For finite-dimensional action spaces $\mathcal{A}$, let $\pi_{\theta^\mu}(\cdot) : \mathcal{X} \to \mathcal{A}$ be the actor network parameterized by θ^μ, $Q(\cdot, \cdot; \theta^Q) : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ be the critic network parameterized by θ^Q, and $d_{\pi_{\theta^\mu}}(\cdot)$ be corresponding stationary distribution. Similarly to the stochastic case (Sutton et al., 2000), the goal is to maximize the expected average reward, under that policy

$$J(\pi_{\theta^\mu}) = \int_{\mathcal{X} \times \mathcal{X}} d_{\pi_{\theta^\mu}}(x) r(x, \pi_{\theta^\mu}(x), x') \mathrm{d}x \mathrm{d}x'.$$

The Deterministic Policy Gradient theorem (Lillicrap et al., 2016; Silver et al., 2014, Theorem 1) shows that the gradient of this average reward objective, under certain conditions, is approximately the expected value, across states, of $\nabla_{\theta^\mu} Q(x, \pi_{\theta^\mu}(x); \theta^Q)$. Using the chain rule, this provides a straightforward gradient ascent update for θ^μ: $\nabla_a Q(x, a; \theta^Q)|_{a=\pi_{\theta^\mu}(x)} \nabla_{\theta^\mu} \pi_{\theta^\mu}(x)$, where

Algorithm 1 DDPG with Action Descriptors

Initialize a random process $\mathcal{N}$ for exploration
Initialize an empty buffer B for experience replay
Initialize actor and critic networks (e.g., with Xavier initialization) $\pi(\cdot,\cdot;\theta^\mu):\mathcal{X}\times\mathcal{Z}\to\mathbb{R}, Q(\cdot,\cdot;\theta^Q):\mathcal{X}\times\mathbb{R}^k\to\mathbb{R}$ and target actor and critic networks $\pi(\cdot,\cdot;\theta^{\mu'}), Q(\cdot,\cdot;\theta^{Q'})$
Define a set of action descriptors $\mathcal{C}, |\mathcal{C}|=k$
and target network update rate τ
for t = 1, 2, ... **do**
  Observe x_t, compute action scalars $u_t = [\pi(x_t,c_1;\theta^\mu),\dots,\pi(x_t,c_k;\theta^\mu)]+\mathcal{N}\in\mathbb{R}^k$
  Execute action $a_t = I(\mathcal{C},u_t)$, and transition to state x_{t+1} and get reward r_{t+1}
  Add sample $(x_t,u_t,x_{t+1},r_{t+1})$ to B
  $B_N \leftarrow$ a mini-batch of N samples from B
  for $(x_i,u_i,x_{i+1},r_{i+1})\in B_N$ **do**
    $u' = [\pi(x_i,c_1;\theta^{\mu'}),\dots,\pi(x_i,c_k;\theta^{\mu'})]$
    set target $y_i = r_{i+1}+\gamma Q(x_{i+1},u';\theta^{Q'})$
  end for
  Update the critic by minimizing the loss:
  $L=\frac{1}{N}\sum_{i=1}^N(y_i-Q(x_i,u_i;\theta^Q))^2$
  Update the actor by gradient ascent, with gradient:
  For $f(\theta^\mu)=[\pi(x_i,c_1;\theta^\mu),\dots,\pi(x_i,c_k;\theta^\mu)]$
  $\frac{1}{N}\sum_{i=1}^N\nabla_u Q(x,u;\theta^Q)|_{u=f(\theta^\mu)}\nabla_{\theta^\mu}f(\theta^\mu)$
  Update target network parameters:
  $\theta^{\mu'}\leftarrow(1-\tau)\theta^{\mu'}+\tau\theta^\mu$
  $\theta^{Q'}\leftarrow(1-\tau)\theta^{Q'}+\tau\theta^Q$
end for

$\nabla_a Q(x,a;\theta^Q)|_{a=\pi_{\theta^\mu}(x)}$ is the Jacobian matrix generated by taking gradient of each action component with respect to the actor network parameters.

We can extend this algorithm to use action descriptors as shown in Algorithm 1, which can efficiently scale to extremely high-dimensional continuous actions while capturing the intrinsic regularities. The change to DDPG is simply to modify the actor network, to input both states and action descriptors and output an action scalar. DDPG, on the other hand, would only receive the state and output a k-dimensional vector $u\in\mathbb{R}^k$ to specify the action scalars. To get the k action scalars with our actor network, the network is evaluated k times with each action descriptor.

5. Theoretical Insights

We provide theoretical evidence showing that learning a policy that explicitly incorporates the action location (or descriptor) z might be beneficial compared to learning many separate policies for each action location. The evidence for this intuitive result is based on comparing the covering number of two different policy spaces. The first is the space of policies with certain spatial regularity (Lipschitzness in z) explicitly encoded. The other is the policy space where the spatial regularity is not explicitly encoded. The covering number is a measure of complexity of a function space and appears in the estimation error terms of many error upper bounds, both in supervised learning problems (Györfi et al., 2002; Steinwart & Christmann, 2008) and in RL (Antos et al., 2008; Lazaric et al., 2016; Farahmand et al., 2016a). Note that the covering number-based argument is only one part of an error upper bound—even in the supervised learning theory (Mohri et al., 2012, page 61). Due to complications arising from exploration strategy and convergence issues, to the best of our knowledge, there is no estimation error bound for deep reinforcement learning algorithms so far. Therefore, our results only provide a possible mathematical insight rather than a complete picture of the sample efficiency bound.

To formalize, consider a policy $\pi_\theta:\mathcal{X}\times\mathcal{Z}\to\mathbb{R}$, parameterized by $\theta\in\Theta$. Let us denote this space by Π_L. We make the following assumptions regarding its regularities (refer to Appendix A for the definition of the covering number and the proof of the result).

Assumption A1 The following properties hold for the policy space Π_L:

- For any fixed action location $z\in\mathcal{Z}$, the ε-covering number of $\Pi_L|_z \triangleq \{\, x\mapsto\pi_\theta(x,z) \,:\, \theta\in\Theta\,\}$ is $\mathcal{N}(\varepsilon)$.
- The policy π_θ is L-Lipschitz in the action location z uniformly in $\theta\in\Theta$ and $x\in\mathcal{X}$, i.e., $|\pi_\theta(x,z_1)-\pi_\theta(x,z_2)|\le L\,\|z_1-z_2\|$ for any $z_1,z_2\in\mathcal{Z}$ and any $x\in\mathcal{X}$. The domain $\mathcal{Z}$ is a bounded subset of $\mathbb{R}^d$.

We think of Π_L as the policy space to which the optimal policy, or a good approximation thereof, belongs, but we do not know which member of it is the actual optimal policy. The role of any policy search algorithm, DDPG included, is to find that policy within Π_L. The stated assumptions on Π_L describe certain types of regularities of Π_L, which manifest themselves both in the complexity of the policy space for a fixed location z (through the covering number $\mathcal{N}(\varepsilon)$), and its Lipschitzness as the location parameter varies. Note that we have not proved that the optimal policy for the Heat Invader problem, or any PDE control problem for that matter, in fact satisfies these regularities.

We would like to compare the ε-covering number of Π_L with the ε-covering number of a policy space that does not explicitly benefit from the Lipschitzness of Π_L, but still can provide an ε-approximation to any member of Π_L. This policy might be seen as the extreme example of the policy used by the conventional DDPG (or any other policy search algorithm) where each action dimension is represented separately. In other words, for having N-dimensional action, we have N different function approximators. Let us introduce some notations in order to define this policy space more precisely.

Consider a set of locations $\{c_i\}_{i=1}^{M_\varepsilon}$ and their corresponding partition $\{A_i\}_{i=1}^{M_\varepsilon}$ with resolution $\frac{\varepsilon}{2L}$. This means that for each $z\in\mathcal{Z}$, there exists a $c_i\in A_i$ such that the distance of

z to c_i is less than $\frac{\varepsilon}{2L}$ and $z \in \mathcal{A}_i$. The number of required partition is $M_\varepsilon = c(\frac{2L}{\epsilon})^d$, for some constant $c > 0$, which depends on the choice of distance metric and the geometry of $\mathcal{Z}$ (but not ε). Define the following policy space (cf. (4)):

$$\underline{\Pi} = \Big\{ \pi_{\underline{\theta}}(x, z) = \sum_{i=1}^{M_\varepsilon} \pi_{\theta_i}(x, c_i) \mathbb{I}\{z \in A_i\} : \pi_{\theta_i} \in \Pi_L|_{c_i}, i = 1, \dots, M_\varepsilon \Big\}.$$

This is the policy space where each action location is modeled separately, and it is allowed to be as flexible as any policy in Π_L with a fixed action location. But this policy space does not restrict the policy to be Lipschitz in $\mathcal{Z}$, so it is more complex than Π_L. The following proposition compares the complexity of $\underline{\Pi}$ and Π_L in terms of the logarithm of their covering numbers (metric entropy).

Proposition 1. *Consider two policy spaces Π_L and $\underline{\Pi}$, as defined above. Suppose that Π_L satisfies Assumption A1. It holds that for any $\varepsilon > 0$, the policy space $\underline{\Pi}$ provides an ε-cover of Π_L. Furthermore, for some $c_1, c_2 > 0$, independent of ε, the following upper bounds on the logarithm of the covering number hold:*

$$\log \mathcal{N}(\varepsilon, \Pi_L) \le c_1 \left(\frac{L}{\varepsilon}\right)^d + \log \mathcal{N}(\varepsilon),$$
$$\log \mathcal{N}(\varepsilon, \underline{\Pi}) \le c_2 \left(\frac{L}{\varepsilon}\right)^d \log \mathcal{N}(\varepsilon/2).$$

The covering number of $\underline{\Pi}$ grows faster than that of Π_L. This is intuitive as the former imposes less restriction on its members, i.e., no Lipschitzness over $\mathcal{Z}$. To give a more tangible comparison between two results, suppose that $\log N(\varepsilon) = c\varepsilon^{-2\alpha}$ for some $c > 0$ and $0 \le \alpha < 1$.[3] In that case, the metric entropy of Π_L behaves as $\varepsilon^{-\max\{d, 2\alpha\}}$ whereas that of $\underline{\Pi}$ behaves as $\varepsilon^{-(d+2\alpha)}$.

Since there is no error bound for DDPG, we cannot compare the error bounds. But for some estimation problems such as regression or value function estimation with a function approximator with the metric entropy of $\log N(\varepsilon) = c\varepsilon^{-2\beta}$, the optimal error bound behaves as $O(n^{-\frac{1}{1+\beta}})$, with n being the number of samples (Yang & Barron, 1999; Farahmand et al., 2016a). Taking this as a rough estimate, the ratio of the error rates of Π_L to that of $\underline{\Pi}$ is

$$n^{\frac{-2\min\{2\alpha, d\}}{(2+2\alpha+d)(2+\max\{2\alpha, d\})}}$$

which becomes significant as α, the complexity of $\Pi_L|_z$, grows. For example, when $\alpha = 1$ and $d = 3$, the error rate of a method that uses Π_L as the function approximator is $n^{1/6}$ faster than the other's. This suggests the possible benefit of explicitly incorporating the spatial regularity in the $\mathcal{Z}$ space, in terms of sample complexity of learning.

[3] This choice of metric entropy holds for some nonparametric function spaces, such as Sobolev spaces and some RKHS. We do not show that this covering number result actually holds for the policy space $\Pi_L|_z$, so at this stage this is only an example.

6. Experiments

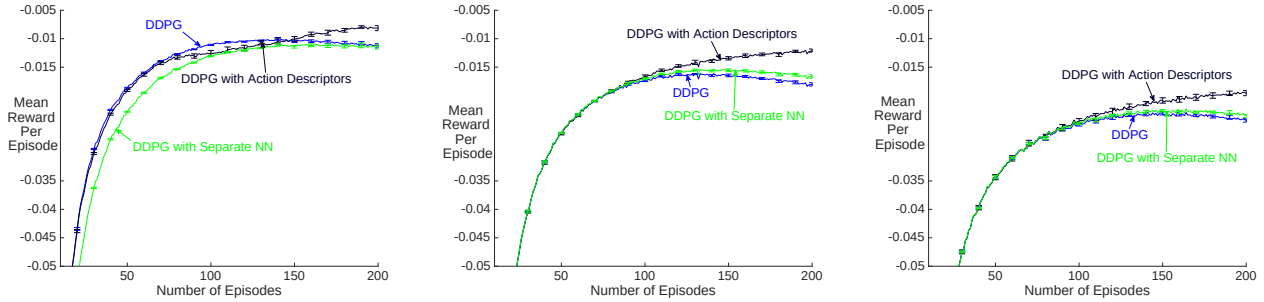
We empirically show that our approach, which can exploit spatial regularity, can be easily scaled to large continuous action dimensions while maintaining competitive performance. We compare DDPG and DDPG with separate Neural Networks (NN) for each action component, to our DDPG with Action Descriptors on two domains: a simple PDE Model domain, and the Heat Invader problem as described throughout this work. The latter is a more difficult problem with more action regularities.

6.1. Results on the PDE Model Domain

The PDE Model domain is using the 2D heat equation as the underlying transition dynamic (see Appendix B.2 for details). The infinite-dimensional state and action spaces are discretized to $\mathcal{X} \subset \mathbb{R}^{d \times d}$ and $\mathcal{A} \subset [-1, 1]^{d \times d}$. Figure 1 shows the comparison between our DDPG with Action Descriptors and the other two competitors, as the number of state and action dimension increase: $d^2 \in \{36, 100, 256\}$. One can see that using action descriptors consistently outperforms the other algorithms. Both DDPG and the DDPG with separate NNs begin to decrease in performance after about 100 episodes, for higher-dimensional actions $d^2 = 100, 256$. DDPG with Action Descriptors, however, does not display this degradation for any action dimension, and continues improving with more training time. Additionally, DDPG with separate NN shows slightly lower sample efficiency, as expected, though with higher action dimensions this effect is less obvious as all algorithms degrade and the regularity on this domain is not that strong.

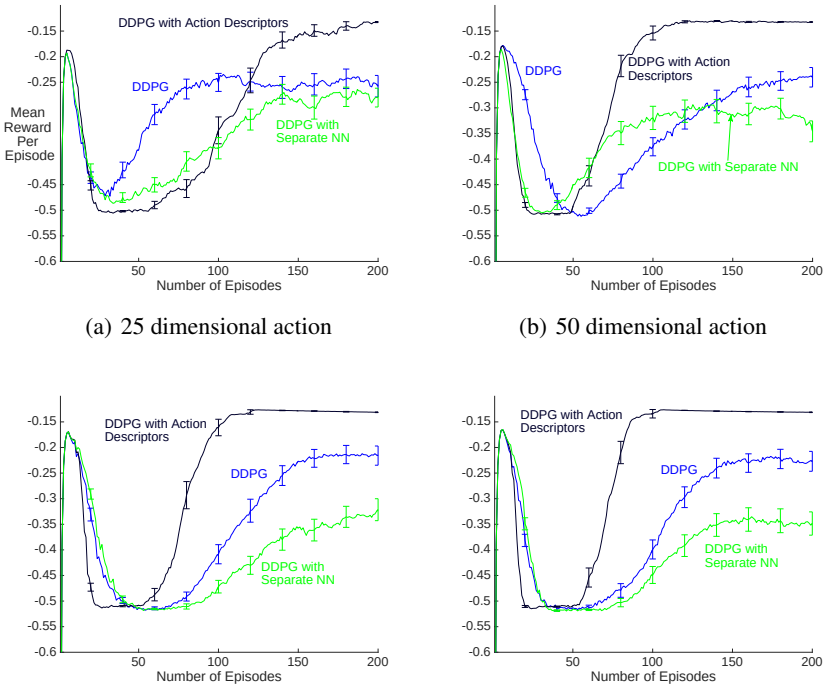
6.2. Results on the Heat Invader Domain

The underlying state transition dynamics of this domain is the PDE as described by Equation (1). In an ideal case, there are infinitely many air conditioners on the floor, and at each time step we can control the temperature for each air conditioner. In simulation, performed using the finite volume solver FiPy (Guyer et al., 2009), the state and action spaces are discretized. The state space is discretized to $\mathcal{X} \subset \mathbb{R}^{50 \times 50}$, giving a total of 2500 measurement locations on the floor. Figure 2 shows comparisons across different control dimensions $a_t \in [-0.5, 0]^k, k \in \{25, 50, 100, 200\}$. On this more difficult domain, it becomes clear that DDPG with separate NN has a lower sample efficiency than DDPG, and the former even shows divergence after the dimension reaches 50. For $k = 100$ and



(a) PDEModel: action dimension = 36 (b) PDEModel: action dimension = 100 (c) PDEModel: action dimension = 256

Figure 1. Mean reward per episode vs. episodes on PDE-Model. The results are averaged over 30 runs, with standard error displayed.



(a) 25 dimensional action (b) 50 dimensional action

(c) 100 dimensional action (d) 200 dimensional action

Figure 2. Mean reward per episode vs. episodes on the Heat Invader domain, with increasing action dimension. The results are averaged over 50 runs. DDPG with action descriptors consistently outperforms other algorithms across dimensions, and DDPG shows higher sample efficiency than the DDPG with separate NN. We suspect the latter cannot converge to a good policy after the dimension increased to 50 due to its inability to capture the regularities. The results suggest that 25 air conditioners may not provide sufficiently fine resolution to control the temperature well in the room, as even the solution under DDPG with Action Descriptors has more variability. Note that the sharp increase in the reward early in the graphs is due to the relatively large noise in the action, triggering the airflow to be ON and making the temperature decrease faster. As the noise decreases after a few episodes, the fan turns off and the agent is improving its policy.

200, DDPG with Action Descriptors learns a significantly better and more stable policy. Theoretically, the finer we can control the temperature, the higher reward we can potentially obtain. We suspect there is a tradeoff between how well the agent can learn the policy and how finely the temperature needs to be controlled. Once the dimension increases to 100 or 200, only DDPG with Action Descriptors is able to exploit this finer-grained control.

7. Conclusion

We presented a general framework of using reinforcement learning for PDE control, which has infinite dimensional state and action spaces (or very high-dimensional in practice). We proposed the concept of action descriptors to enable RL algorithms, such as deterministic policy gradient, to handle these problems. Theoretical evidence showed why our approach might have better sample efficiency compared to a conventional continuous-action RL approach. Our strategy enables the architecture to easily scale with increasing action dimension. We showed that the same neural network architecture, with the exception of the output layer, can obtain significantly better performance scaling as the dimension increases on two PDE domains. We believe that the proposed RL-based approach has the potential to be a powerful alternative to conventional control engineering methods to PDE control, as well as other RL/control problems with extremely high-dimensional action spaces with action regularities.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful feedback.

A. Theoretical Analysis: Proofs and Additional Discussions

In this appendix we first review the notion of covering number (Appendix A.1). Afterwards, we prove Proposition 1. The basic idea of the proof is to calculate the covering numbers of a policy space that has a Lipschitz continuity and that of another space without such property which still can provide an ϵ-approximation to any object in the former policy space (Appendix A.2). Finally in Appendix A.3, we relate the Lipschitz continuity of feedforward neural networks with their weights, thus showing that one can explicitly impose the Lipschitz continuity in DNN by controlling the norm of their weights.

A.1. Covering Number

We briefly introduce the notion of covering number, and refer the reader to standard references for more discussions, e.g., Chapter 9 of Györfi et al. (2002). Consider a normed function space $\mathcal{F}(\Omega)$ defined over the domain Ω with the norm denoted by $\|\cdot\|$. The set $S_\varepsilon = \{f_1, \dots, f_{N_\varepsilon}\}$ is the ε-covering of $\mathcal{F}$ w.r.t. the norm if for any $f \in \mathcal{F}$, there exists an $f' \in S_\varepsilon$ such that $\|f - f'\| \leq \varepsilon$. The covering number is the minimum N_ε such that there exists an ε-cover S_ε for $\mathcal{F}$. We use $\mathcal{N}(\varepsilon, \mathcal{F}(\Omega))$ or simply $\mathcal{N}(\varepsilon)$ if $\mathcal{F}(\Omega)$ is clear from the context, to denote it.

For simplicity of arguments in this paper, the covering number results use the supremum norm, which is defined as

$$\|f\|_\infty = \sup_{w \in \Omega} |f(w)|.$$

A.2. Proof for Proposition 1

Proof. First we show that $\underline{\Pi}$ provides an ε-approximation of Π_L. Pick any $\varepsilon > 0$. For each c_i ($i = 1, \dots, M_\varepsilon$), pick a minimal $\varepsilon/2$-covering set of $\Pi_L|_{c_i}$, and call it $S_{\varepsilon/2, c_i}$. Define the following function space:

$$\underline{\Pi}_{\varepsilon/2} = \Big\{ \pi_{\underline{\theta}}(x, z) = \sum_{i=1}^{M_\varepsilon} \pi_{\theta_i}(x, c_i) \mathbb{I}\{z \in A_i\} : \pi_{\theta_i} \in S_{\varepsilon/2, c_i}, i = 1, \dots, M_{\varepsilon/2} \Big\}.$$

This space is an $\varepsilon/2$-covering of $\underline{\Pi}$. Moreover, it provides an ε-covering of Π_L too. To see this, consider any $\pi_\theta \in \Pi_L$. For any fixed $(x, z) \in \mathcal{X} \times \mathcal{Z}$, the location z falls in one of the partitions A_i, so $\|z - c_i\| \leq \frac{\varepsilon}{2L}$. By construction of $\underline{\Pi}_{\varepsilon/2}$, there exists $\pi_{\underline{\theta}} \in \underline{\Pi}_{\varepsilon/2}$ such that $|\pi_\theta(x, c_i) - \pi_{\underline{\theta}}(x, c_i)| \leq \varepsilon/2$. Therefore, we have

$$\begin{aligned} \big|\pi_\theta(x, z) - \pi_{\underline{\theta}}(x, z)\big| \leq & |\pi_\theta(x, z) - \pi_\theta(x, c_i)| + \\ & \big|\pi_\theta(x, c_i) - \pi_{\underline{\theta}}(x, c_i)\big| + \\ & \big|\pi_{\underline{\theta}}(x, c_i) - \pi_{\underline{\theta}}(x, z)\big| \\ \leq & L \|z - c_i\| + \frac{\varepsilon}{2} + 0 \leq \varepsilon. \end{aligned}$$

As we have M_ε cells, the number of members of the covering of $\pi_{\underline{\theta}}(x, c_i)$ is

$$[\mathcal{N}(\varepsilon/2)]^{M_\varepsilon}. \tag{5}$$

Substituting the value of M_ε leads to the desired result for $\mathcal{N}(\varepsilon, \underline{\Pi})$.

To derive the covering number for Π_L, let $\varepsilon > 0$ and define the following function space:

$$\tilde{\Pi}_\varepsilon = \{\tilde{\pi}(x, z) = \sum_{i=1}^{M_\varepsilon} 0.5\varepsilon \left\lfloor \frac{\pi(x, c_i)}{0.5\varepsilon} \right\rfloor \mathbb{I}\{z \in A_i\}, \pi \in \Pi_L\}$$

We shortly show that $\tilde{\Pi}_\varepsilon$ is an ε-covering of Π_L. Also notice that $\tilde{\pi}$ only takes discrete values with resolution of $\varepsilon/2$.

Consider any $\pi_\theta \in \Pi_L$. As in the previous case, for any fixed $(x, z) \in \mathcal{X} \times \mathcal{Z}$, the location z falls in one of the partitions A_i. Because of the construction of $\tilde{\Pi}_\varepsilon$, there exists a $\tilde{\pi}$ that is $\varepsilon/2$-close to $\pi(x, c_i)$. Using this property and the Lipschitzness of $\pi_\theta \in \Pi_L$, we have

$$\begin{aligned} |\pi_\theta(x, z) - \tilde{\pi}(x, z)| \leq & |\pi_\theta(x, z) - \pi_\theta(x, c_i)| + \\ & |\pi_\theta(x, c_i) - \tilde{\pi}(x, c_i)| + \\ & |\tilde{\pi}(x, c_i) - \tilde{\pi}(x, z)| \\ \leq & L \|z - c_i\| + \\ & \left| 0.5\varepsilon \left\lfloor \frac{\pi_\theta(x, c_i)}{0.5\varepsilon} \right\rfloor - \pi_\theta(x, c_i) \right| + 0 \\ \leq & \varepsilon/2 + \varepsilon/2 = \varepsilon \end{aligned} \tag{6}$$

So $\tilde{\Pi}_\varepsilon$ provides Π_L with an ε-cover. It remains to count the number of elements of $\tilde{\Pi}_\varepsilon$.

We choose an arbitrary centre c_1. We let the function $\tilde{\pi}(x, c_1)$ to be one of possible $\mathcal{N}(\varepsilon)$ members of the minimal covering of $\Pi_L|_{c_1}$. Notice that for any c_i and c_j that are neighbour (so they have distance less than ε/L), using the Lipschitzness of π and the definition of $\tilde{\pi}$, we have

$$\begin{aligned} |\tilde{\pi}(x, c_i) - \tilde{\pi}(x, c_j)| \leq & |\tilde{\pi}(x, c_i) - \pi(x, c_j)| + \\ & |\pi(x, c_i) - \pi(x, c_j)| + \\ & |\pi(x, c_j) - \tilde{\pi}(x, c_j)| \\ \leq & 2\varepsilon. \end{aligned}$$

Consider two neighbour centres c_i and $c_j \neq c_i$. Since $\tilde{\pi}$ only takes discrete values (with the resolution of $\varepsilon/2$), the

value of $\tilde{\pi}(x, c_j)$ can only be one of 9 possible values, i.e., $\tilde{\pi}(x, c_i) - 4\varepsilon/2, \tilde{\pi}(x, c_i) - 3\varepsilon/2, \ldots, \tilde{\pi}(x, c_i) + 4\varepsilon/2$.

Choose $c_i = c_1$. One of its neighbour, let us call it $c_j = c_2$, can take at most 9 different values. Therefore, the total number of function $\tilde{\pi}$ defined over $\mathcal{X} \times \{c_1, c_2\}$ is $9\mathcal{N}(\varepsilon)$. We continue this argument with an arbitrary sweep through neighbourhood structure of the partition. As there are at most M_ε points, the number of possible functions $\tilde{\pi}_\varepsilon$ is

$$\mathcal{N}(\varepsilon) \times 9^{M_\varepsilon}. \tag{7}$$

Replacing the value of M_ε leads to the desired result. □

We would like to acknowledge that the argument for counting the members of $\tilde{\Pi}_\varepsilon$ is borrowed from a similar argument in Lemma 2.3 of van de Geer (2000).

A.3. Lipschitzness of Feedforward Neural Networks

The function space Π_L in Section 5 is L-Lipschitz in the action location z. To show how one might impose this condition in practice, we focus on NN and relate the Lipschitz constant of a feedforward NN to its weights. Consequently, by controlling the weights, we might control the Lipschitz constant of the NN. Note that the Lipschitz properties of NN has been studied before, either as a standalone result (e.g., Balan et al. 2017; Asadi et al. 2018) or as a part of proof for another result (e.g., Barron 1994).

A single layer of feedforward NN with ReLU or $\tanh$ nonlinearity is Lipschitz. To see this, consider an ℓ_p-norm and its induced matrix norm (for any $1 \leq p \leq \infty$). As ReLU or $\tanh$ are both 1-Lipschitz coordinate-wise, we have

$$\begin{aligned}\|\text{ReLU}(Wx_1) - \text{ReLU}(Wx_2)\|_p &\leq \|W(x_1 - x_2)\|_p \\ &\leq \|W\|_p \|x_1 - x_2\|_p .\end{aligned}$$

Therefore, its Lipschitz constant is $L = \|W\|_p$. As the composition of Lipschitz functions with Lipschitz constants L_1, L_2, etc. w.r.t. the same norm is also Lipschitz with the Lipschitz constant $L = L_1 L_2 \cdots$, one may obtain the overall Lipschitz constant of a multi-layer feedforward NN. So by controlling the norm of the weights, either by regularization or weight clipping, one can control the overall Lipschitz constant of the network.

B. Detail of Experiments

In this section, we first introduce concrete neural network and parameter settings followed by the details of how we optimize each algorithm and how we plot the learning curve. Afterwards, we provide details about the PDE Model and the Heat Invader domains. Additional experimental results are given for completeness.

B.1. Experimental Settings

Neural network architecture. We follow the same neural network architecture as introduced in the original DDPG paper (Lillicrap et al., 2016). We use TensorFlow (Abadi et al., 2015) for our implementations. The DNN has 3 convolutional layers without pooling, and each layer has 32 filters. The kernel sizes are 4×4 for the first two layers and 3×3 for the third one. Batch normalization (Ioffe & Szegedy, 2015) is used in the convolutional layers. The convolutional layers are followed by two fully connected layers and each has 200 ReLU units. In the actor network, the activation function in the output layer is *sigmoid* for the Heat Invader domain and *tanh* for the PDE Model domain. The critic network uses the same convolutional layer setting, but its activation function in the output layer is linear. The L_2 weight decay with parameter 0.001 is used after the convolutional layers in the critic network. The final layer parameters of both actor and critic networks were initialized from a uniform distribution $[-0.0003, 0.0003]$ and all other parameters are initialized using Xavier initialization (Glorot & Bengio, 2010).

As for the DDPG with separate NN, if the action is in $\mathbb{R}^k$, we have k neural networks with the same architecture as explained above, except that all of them share the same convolutional layers.

For DDPG with action descriptors, we use exactly the same architecture as DDPG, except that the output layer has only one output unit. The *action descriptor* comes into the neural network immediately after the convolutional layers by concatenating.

Parameter setting. Across all experiments, we used experience replay with the buffer size of $20,000$ and the batch size of 16. Each episode has 40 steps. The discount factor is 0.99. The exploration noise is from the Gaussian distribution $\mathcal{N}(u_t, \frac{1.0}{\text{episodes}})$. For each choice of action dimension, we sweep over parameters as follows. Actor learning rate is selected from $\alpha \in \{0.000001, 0.000005, 0.00001, 0.0001\}$, critic learning rate is from $\alpha \times \{1.0, 5.0, 10.0, 20.0, 40.0\}$. The intuition behind this selection is that the critic learning rate should be larger, and in fact our experimental results are consistent with this intuition.

Performance measure. We take a similar approach as in the previous work (Farahmand et al., 2017), computing the mean reward per step for each episode i averaged over N runs. Hence on our learning curve, given episode index i, the corresponding y-axis value is computed as

$$R^{(i)} = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{T} \sum_{t=1}^{T} r_t^{(n,i)},$$

where $T = 40$ across all experiments. To pick up the best

parameter, we compute Evaluate $= \sum_{i=m}^{n} R^{(i)}$. For the PDE Model, we use $m = 180$ and $n = 200$, whereas for the Heat Invader, we use $m = 150$ and $n = 200$. We choose these ranges to avoid a "divergent" learning curve which may happen when an algorithm picks a large learning rate and hence converges quickly at the beginning but later diverges.

B.2. Additional Details on PDE Model

The PDE Model domain is a simple model that allows easy experimentation. It is defined based on the controlled 2D heat equation

$$\frac{\partial h(z,t)}{\partial t} = \alpha(\nabla^2 h + a),$$

where h is the heat, a function of location $z = (x, y) \in \mathbb{R}^2$ and time t, a is a function-valued action (or control signal), and α is a constant that depends on the physical properties of the environment (thermal conductivity, etc.) and is called thermal diffusivity. The action a is a scalar field that we can control in order to modify the behaviour of the dynamical system. We let $\alpha = 1$ for simplicity.

We use the finite difference time domain (FDTD) method to solve this PDE, which we now describe. We discretize the 2D spatial domain into $d \times d$ elements with d being an integer value. This can be represented by a matrix, in which each entry indicates the heat value at a particular location. We can implement the $d \times d$-dimensional state transition (still in continuous time) as

$$\frac{\partial h(z,t)}{\partial t} = \alpha\Big(\frac{h(x+\delta,y,t)+h(x-\delta,y,t)+h(x,y-\delta,t)+h(x,y+\delta,t)}{c} - \frac{4h(x,y,t)}{c} + \text{action}(x,y,t)\Big),$$

where c is a constant to scale the change and the action(x, y, t) is the control command that is under our control at location (x, y). For each $x \in \mathcal{X}$, let $x^{(i,j)}$ denotes the element in the i-th row and j-th column of the matrix x. To turn this continuous-time finite difference approximation to discrete-time finite difference approximation (known as finite-difference time domain (FDTD) method), let us denote x_t and a_t as the state and action at time t. The state dynamics are described by the following equations,

$$\Delta x_t = \frac{\left(x_t^{(i-1,j)} + x_t^{(i+1,j)} + x_t^{(i,j-1)} + x_t^{(i,j+1)} - 4x_t^{(i-1,j-1)}\right)}{\delta_s} + a_t^{(i-1,j-1)}$$

$$x_{t+1}^{(i-1,j-1)} \leftarrow x_t^{(i-1,j-1)} + \delta_t \Delta x_t,$$

with the temporal discretization $\delta_t = 0.001$ and spatial discretization $\delta_s = 0.1$ for the finite difference method.

For boundary values, where $i - 1$ or $i + 1$ become smaller than 0 or larger than d, we set them to zero, i.e., $x_t^{(i,j)} = 0, \forall i, j \notin \{1, ..., d\}$. The reward function is defined as

$$r(x_t, a_t, x_{t+1}) = -\frac{||x_{t+1}||_2}{d} - \frac{||a_t||_2}{d}.$$

One can intuitively understand this equation as following. Since the heat always move from the high heat region to low heat region, for each entry in the matrix (i.e. a location on the 2-D room) we can take the sum of its neighbors as the heat value coming from nearby region. Then if that sum is larger than the entry's current value, we should expect the entry's heat increases. At the beginning of each episode, each entries' values are independently and uniformly initialized from $[0, 1]^{d\times d}$. Note that at each step, the update to the value in each entry happens simultaneously. The boundary values are set as 0 across all steps. Each episode is restricted to 40 steps and at each step, an action is repeatedly executed 100 times. This design is based on the intuition that the agent may not react quickly enough to change action every 0.001 time unit.

Our design of reward function is mainly based on the intuition that we want to keep the temperature low and action cost small. We believe other reasonable reward choices can also work.

We choose action descriptors simply by uniformly choosing from the space $[-0.5, 0.5]^2$. There is no particular reason for our choice, we believe other range such as $[-1, 1]$ can also work, as long as the descriptors are topologically distributed the same as the air conditioners.

B.3. Additional Details on Heat Invader

We include Figure 3 to describe the domain: Heat Invader. The action is discretized to four rows, lying in the middle of the room and each row has 50 air conditioners. The output action is repeated to 200 executable action dimensions if the output dimension is less than 200. Similar to Farahmand et al. (2017), we still design two fans symmetrically located on the left and right wall of the room. A fan will be trigged if the sum of absolute values of actions located on that corresponding half area of the room exceeds some threshold, which we set as 25. The reward function

$$r(x_t, a_t, x_{t+1}) = -\text{cost}(a_t) - \int_{z\in\mathcal{Z}} I(|T(z, t+1)| > T^*(z, t+1))\mathrm{d}z \tag{8}$$

is designed as following. $\mathcal{Z}$ is discretized to 50×50 on the floor and hence there are totally 2500 temperature measurements. We set $T^*(z, t + 1) = 0.501, \forall z \in \mathcal{Z}$. We

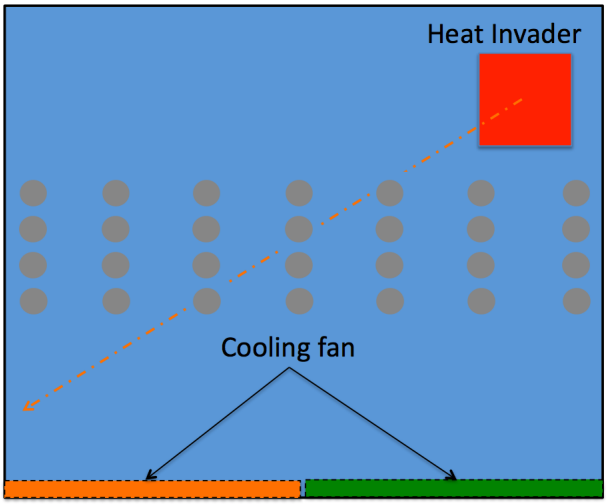


Figure 3. Heat invader domain. Image is modified from the paper (Farahmand et al., 2016b). The solid circle refers to the elements of the air conditioner. The actual number of air conditioners is different from what the figure shows.

further scaled the integration (summation over locations) by 2500. The cost from action is defined as $\text{cost}(a_t) = \frac{||a_t||}{d}, a_t \in \mathbb{R}^d$. The initial position of the heat invader is randomly sampled from $(i, j) \in [45, 50]^2, i, j \in \mathbb{Z}$. There are two types of airflow to choose, one is *uniform* and another one is *whirl*. We also include results by using *whirl* airflow as showed in Figure 4. Note that there is a common learning pattern when using either uniform or whirl airflow. At the beginning, the exploration noise is large and hence the fan is triggered frequently, allowing the temperature to be lowered faster. As the noise decreasing, the penalty coming from the "uncomfortable temperature" shows effect and hence the performance drops. Afterwards while the agent improves the policy, the performance starts to increases again.

We design the set of action descriptors as follows. When the action dimension $k \leq 50$, we think the air conditioners are located on a single line in the middle of the room, hence we pick the descriptors uniformly from the segment $[-1, 1]$. When the action dimension $k = 100$, we think there are two lines on the middle of the room and hence we design the descriptors as two dimensional vectors $(x, y) \in [-1, 1]^2$, where x takes two values $-1, 1$ while y takes 50 values uniformly along $[-1, 1]$. Similar approach applied to the case $k = 200$. Note that this domain has a fixed executable action dimension $a_t \in \mathbb{R}^{200}$, hence when the output action from the agent has dimension less than 200 (i.e., $1, 25, 50, 100$), the *adapter* $I(\mathcal{C}, u_t)$ would map the output action to 200 dimensions by repeating.

As a sanity check, we also conducted experiments when we use only 1 dimensional action as showed in figure 4. Neither of the two algorithms can learn a good policy for $k = 1$, because there is no sufficiently fine-grained control. In this case, all three algorithms are almost identical, except that DDPG with Action Descriptors has two additional input units, as the descriptor is in $\mathbb{R}^2$.

Due to computational resource restriction, to generate the figure using the uniform airflow, we use 10 runs to find best parameter settings and then do 50 runs for that best setting.

References

Abadi, M., Agarwal, A., Barham, P., and et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org. 10

Ahuja, S., Surana, A., and Cliff, E. Reduced-order models for control of stratified flows in buildings. In *American Control Conference (ACC)*, pp. 2083–2088. IEEE, 2011. 1

Antos, A., Szepesvári, Cs., and Munos, R. Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71:89–129, 2008. 6

Asadi, K., Misra, D., and Littman, M. L. Lipschitz continuity in model-based reinforcement learning. *arXiv:1804.07193*, 2018. 10

Baird, L. and Klopf, A. H. Reinforcement learning with high-dimensional, continuous actions. *Wright Laboratory, Wright-Patterson Air Force Base, Tech. Rep*, 1993. 5

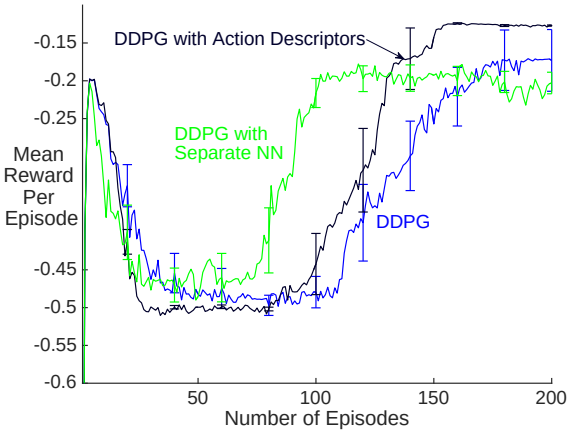
Balan, R., Singh, M., and Zou, D. Lipschitz properties for deep convolutional networks. *arXiv:1701.05217*, 2017. 10

Barron, A. R. Approximation and estimation bounds for artificial neural networks. *Machine Learning*, 14(1):115–133, Jan 1994. 10

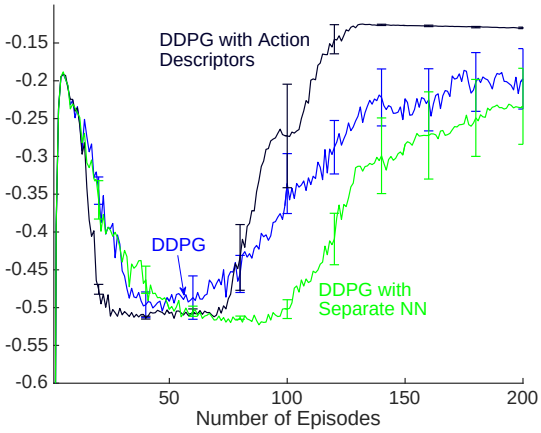
Belletti, F., Haziza, D., Gomes, G., and Bayen, A. M. Expert level control of ramp metering based on multi-task deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, 19(4):1198–1207, 2018. 1

Bertsekas, D. P. *Abstract dynamic programming*. Athena Scientific Belmont, 2013. 3

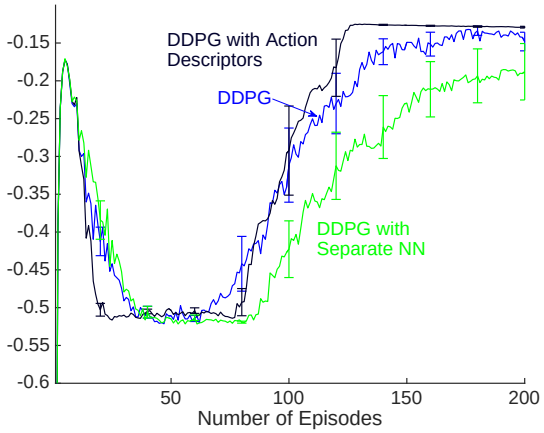
Borggaard, J., Burns, J. A., Surana, A., and Zietsman, L. Control, estimation and optimization of energy efficient buildings. In *American Control Conference (ACC)*, pp. 837–841, 2009. 1



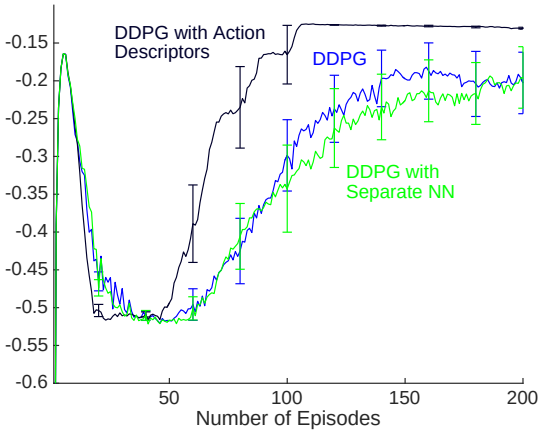
(a) 25 dimensional action



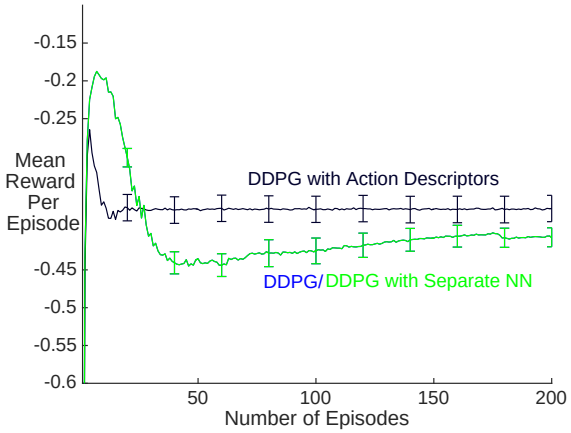
(b) 50 dimensional action



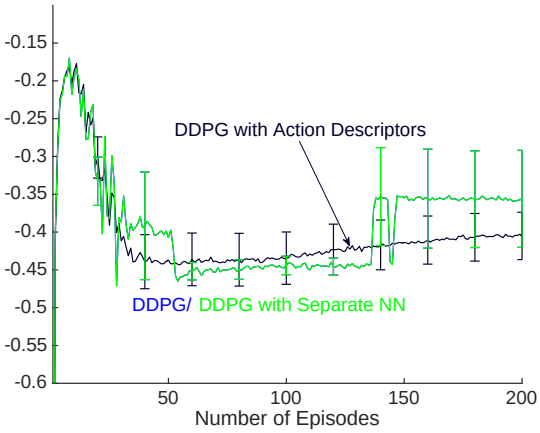
(c) 100 dimensional action



(d) 200 dimensional action



(e) 1 dimensional action, uniform



(f) 1 dimensional action, whirl

Figure 4. Results of mean reward per episode vs. episodes on the Heat Invader domain, with an increasing number of action dimensions. The results are averaged over 10 runs except Figure (e). This figure is generated by using *whirl* air flow unless otherwise specified. The results basically match with what we see in the experimental section, which was generated using *uniform* airflow. Figure (e) and (f) show the comparison when using 1d action dimension as a sanity check.

Brunton, S. L. and Noack, B. R. Closed-loop turbulence control: Progress and challenges. *Applied Mechanics Reviews*, 67(5), 2015. 1

Burns, J. A. and Hu, W. Approximation methods for boundary control of the Boussinesq equations. In *IEEE Conference on Decision and Control (CDC)*, pp. 454–459, 2013. 1

Burns, J. A., He, X., and Hu, W. Feedback stabilization of a thermal fluid system with mixed boundary control. *Computers & Mathematics with Applications*, 2016. 1

Deisenroth, M. P., Neumann, G., and Peters, J. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013. 1

del R Millán, J., Posenato, D., and Dedieu, E. Continuous-Action Q-Learning. *Machine Learning*, 2002. 5

Dulac-Arnold, G., Denoyer, L., Preux, P., and Gallinari, P. Fast reinforcement learning with large action sets using error-correcting output codes for mdp factorization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 180–194. Springer, 2012. 5

Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., Mann, T., Weber, T., Degris, T., and Coppin, B. Deep reinforcement learning in large discrete action spaces. *arXiv:1512.07679*, 2015. 5

Duriez, T., Brunton, S. L., and Noack, B. R. *Machine Learning Control–Taming Nonlinear Dynamics and Turbulence*, volume 116 of *Fluid mechanics and its applications*. Springer, 2016. 1, 2

Farahmand, A.-m., Ghavamzadeh, M., Szepesvári, Cs., and Mannor, S. Regularized fitted Q-iteration for planning in continuous-space Markovian Decision Problems. In *Proceedings of American Control Conference (ACC)*, pp. 725–730, June 2009. 4

Farahmand, A.-m., Ghavamzadeh, M., Szepesvári, Cs., and Mannor, S. Regularized policy iteration with nonparametric function spaces. *Journal of Machine Learning Research (JMLR)*, 17(139):1–66, 2016a. 6, 7

Farahmand, A.-m., Nabi, S., Grover, P., and Nikovski, D. N. Learning to control partial differential equations: Regularized fitted Q-iteration approach. In *IEEE Conference on Decision and Control (CDC)*, pp. 4578–4585, December 2016b. 1, 2, 4, 12

Farahmand, A.-m., Nabi, S., and Nikovski, D. N. Deep reinforcement learning for partial differential equation control. In *American Control Conference (ACC)*, 2017. 1, 2, 4, 10, 11

Foures, D., Caulfield, C.-c., and Schmid, P. J. Optimal mixing in two-dimensional plane poiseuille flow at finite Péclet number. *Journal of Fluid Mechanics*, 748: 241–277, 2014. 2

Gaskett, C., Wettergreen, D., and Zelinsky, A. Q-Learning in Continuous State and Action Spaces. In *Advanced Topics in Artificial Intelligence*. 1999. 5

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010. 10

Guyer, J. E., Wheeler, D., and Warren, J. A. FiPy: Partial differential equations with Python. *Computing in Science and Engineering*, 11(3):6–15, 2009. URL `http://www.ctcms.nist.gov/fipy`. 7

Györfi, L., Kohler, M., Krzyżak, A., and Walk, H. *A Distribution-Free Theory of Nonparametric Regression*. Springer Verlag, New York, 2002. 6, 9

He, J., Chen, J., He, X., Gao, J., Li, L., Deng, L., and Ostendorf, M. Deep reinforcement learning with an unbounded action space. *arXiv:1511.04636*, 2015. 5

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015. 10

Kim, K. J., Choi, H. R., and Tan, X. Biomimetic robotic artificial muscles. *World Scientific*, 2013. 1

Kober, J., Andrew Bagnell, J., and Peters, J. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32:1238–1274, 09 2013. 1

Krstic, M. and Smyshlyaev, A. *Boundary control of PDEs: A course on backstepping designs*, volume 16. SIAM, 2008. 1

Lazaric, A., Ghavamzadeh, M., and Munos, R. Analysis of classification-based policy iteration algorithms. *Journal of Machine Learning Research (JMLR)*, 17(19):1–30, 2016. 6

Lighthill, M. and Whitham, J. On kinematic waves. i: Flow movement in long rivers. ii: A theory of traffic flow on long crowded roads. pp. 229:281–345, 1955. 1

Lillicrap, T. P., J. Hunt, J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. 2016. 5, 10

Lions, J. L. Optimal control of systems governed by partial differential equations. 1971. 1

Mohri, M., Rostamizadeh, A., and Talwalkar, A. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN 026201825X, 9780262018258. 6

Montgomery, W. H. and Levine, S. Guided policy search as approximate mirror descent. In *Advances in Neural Information Processing Systems (NIPS - 29)*, pp. 4008–4016, 2016. 5

Pazis, J. and Parr, R. Generalized value functions for large action sets. In *Proceedings of the 28th International Conference on Machine Learning*, pp. 1185–1192, 2011. 5

Popescu, M. C., Petrisor, A., and Drighiciu, M. A. Modelling and simulation of a variable speed air-conditioning system. In *IEEE International Conference on Automation, Quality and Testing, Robotics*, volume 2, pp. 115–120, May 2008. 1

Richards, P. I. Shock waves on the highway. *Operations Research*, 4(1):42–51, 1956. 1

Sallans, B. and Hinton, G. E. Reinforcement learning with factored states and actions. *Journal of Machine Learning Research (JMLR)*, 5:1063–1088, 2004. 5

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015. 5

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *International Conference on Learning Representations*, 2016. 5

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on on Machine Learning (ICML)*, pp. 387–395. JMLR.org, 2014. 5

Steinwart, I. and Christmann, A. *Support Vector Machines*. Springer, 2008. 6

Sunehag, P., Evans, R., Dulac-Arnold, G., Zwols, Y., Visentin, D., and Coppin, B. Deep reinforcement learning with attention for slate markov decision processes with high-dimensional states and actions. *arXiv:1512.01124*, 2015. 5

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. The MIT Press, 1998. 1, 3

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS - 12)*, 2000. 5

Szepesvári, Cs. *Algorithms for Reinforcement Learning*. Morgan Claypool Publishers, 2010. 3

van de Geer, S. A. *Empirical Processes in M-Estimation*. Cambridge University Press, 2000. 10

van Hasselt, H. and Wiering, M. A. Reinforcement learning in continuous action spaces. In *IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL)*, pp. 272–279, 2007. 5

Wang, H., Wang, F., and Xu, K. Modeling information diffusion in online social networks with partial differential equations. *arXiv:1310.0505*, 2013. 1

Yang, Y. and Barron, A. R. Information-theoretic determination of minimax rates of convergence. *The Annals of Statistics*, 27(5):1564–1599, 1999. 7