MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

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

Pan, Y.; Farahmand, A.-M.; White, M.; Nabi, S.; Grover, P.; Nikovski, D.N.

TR2018-028 February 2018

Abstract

Recent work has shown reinforcement learning (RL) is promising to control partial differential equations (PDE) with discrete actions. This paper shows how to use RL algorithms to solve more general and common PDE control problems where the action can be in continuous high-dimensional space with spatial relationships amongst action dimensions. In particular, we propose the idea of action descriptors, which encodes regularities among spatially-extended action dimensions and enables the agent to control high-dimensional action PDEs. Based upon covering number argument, we provide theoretical evidence suggesting that this approach can be more sample efficient compared to a conventional approach that treat each action dimension separately and does not explicitly exploit the spatial regularity in the action space. The action descriptors approach is then used within the deep deterministic policy gradient algorithm, and experiments are conducted on two PDE control domains, with up to 256 dimensional continuous actions. The empirical results showthe advantage of the proposed approach over the conventional approach. We believe the action descriptor-based approach has the potential of solving various PDE control problems with high-dimensional action spaces, as well as some other classical high-dimensional action problems where the action dimensions have regularities among themselves.

arXiv

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

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

Abstract

Recent work has shown reinforcement learning (RL) is promising to control partial differential equations (PDE) with discrete actions. This paper shows how to use RL algorithms to solve more general and common PDE control problems where the action can be in continuous high-dimensional space with spatial relationships amongst action dimensions. In particular, we propose the idea of *action descriptors*, which encodes regularities among spatially-extended action dimensions and enables the agent to control high-dimensional action PDEs. Based upon covering number argument, we provide theoretical evidence suggesting that this approach can be more sample efficient compared to a conventional approach that treat each action dimension separately and does not explicitly exploit the spatial regularity in the action space. The action descriptors approach is then used within the deep deterministic policy gradient algorithm, and experiments are conducted on two PDE control domains, with up to 256 dimensional continuous actions. The empirical results show the advantage of the proposed approach over the conventional approach. We believe the action descriptor-based approach has the potential of solving various PDE control problems with high-dimensional action spaces, as well as some other classical high-dimensional action problems where the action dimensions have regularities among themselves.

[1]Mitsubishi Electric Research Laboratories (MERL), Cambridge, USA [2]Department of Computing Science, University of Alberta, Edmonton, Canada [3]Vector Institute, Toronto, Canada. Correspondence to: Amir-massoud Farahmand <farahmand@vectorinstitute.ai>.

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

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 Equation (PDE), which has attracted attention because of their potential applications spread over physical dynamic system (Lions, 1971) and engineering problems, including the design of air conditioning systems (Popescu et al., 2008), modeling of flexible artificial muscles with many degrees of freedom (Kim et al., 2013), of traffic control (Richards, 1956; Lighthill & Whitham, 1955), and the modeling of information flow among 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 electrical radiation is insignificant). The property of these systems is that their state can be described by a finite dimensional variable. The use of RL to control ODEs tasks such as mentioned above, 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 a 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 relatively an unexplored area (Farahmand et al., 2016b; 2017a; Belletti et al., 2017; Duriez et al., 2016).

Control of PDEs has been investigated in the 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 controller for the model also requires a control engineer with an expertise in modeling and control of PDEs, which makes it even more challenging. Further, 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. Furthermore, 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; 2017a), 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 theoretical state of a PDE is an infinite-dimensional vector. Moreover, many PDE control problems have infinite-dimensional continuous action—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. Spatial discretization of the action space leads to a very high-dimensional continuous action space with spacial regularities to be taken care of, conventional RL algorithms, as we will discussed in later section, may not be suitable for those problems.

In this work, we address PDE control with RL by using a Markov Decision Process (MDP) formalism with function-valued state and action. We then propose a generic method to model actions in PDE control problems by introducing *action descriptors*, which can naturally scale to exceptionally high dimensional continuous action and simultaneously capture regularities among action components. We provide some mathematical insight on why our proposed approach might have a better sample complexity through a covering number argument. By benefiting from action descriptors, we propose an innovative neural network architecture that is not changed with the increase of action dimensions; rather, it simply queries an expanded set of action descriptors. This contrasts standard RL algorithms, that directly output a high-dimensional action vector. Last, we empirically verify the effectiveness of our architecture on two PDE control domains with up to 256 dimensional continuous actions.

2. Reformulating PDE control as an MDP

In this section, we first provide a PDE control example, and then briefly discuss how this could 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 problems characterize a dynamical system using partial different equations. An example of such a set of equations is the convection-diffusion equation

$$\frac{\partial T(z,t)}{\partial t} = \nabla \cdot \frac{1}{P_e} \nabla T - \nabla \cdot (vT) + S(z,t), \quad (1)$$

where $S(z,t)$ is the is the source or sink at a specific location $z \in \mathcal{Z} \subset \mathbb{R}^d$ and time t; $T(z,t)$ is the temperature function at location z and t; $\nabla\cdot$ is the divergence operator measuring the difference between a sink and source in the vector field; $\frac{1}{P_e}$ is a diffusivity constant; and $v = v(z,t)$ is the velocity field, physically describing the airflow in the domain, which is also a function of location and time. This PDE describes how the temperature changes at each location at each time step. 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, and $a(z,t)$ is the action that we can 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 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 perceiv-

able). In this case, the action is the temperature of each tiny heater/cooler in the wall, so the action space would be a 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. Formulation as an MDP

To relate this to the reinforcement learning setting, we first provide a brief overview of this formalism, which involves trial-and-error interaction in an environment formalized by an 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, for some discount factor $\gamma < 1$. Typically in the RL literature, both the state space and action space are finite dimensional, for example with both being subsets of an Euclidean space, $\mathbb{R}^d$ and $\mathbb{R}^k$ respectively. However, this can be extended to infinite-dimensional vector spaces, which we describe below and which is needed to represent the PDE control problem as an MDP.[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 $a_t = a(\cdot, t)$. 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. As an example, for the Heat Invader problem, the state is the current temperature at all locations, and the action can increase or decrease the source value at each location. The particular choice of function space depends on the PDE and its regularity. For simplicity of exposition, 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 the following partial *difference* equation: $x_{t+1} = f(x_t, a_t)$ for some function $f : \mathcal{F}(\mathcal{X}) \times \mathcal{F}(\mathcal{Z}) \to \mathcal{F}(\mathcal{X})$, which depends on the PDE. As an example, for the convection-diffusion PDE (1) with the state being T, we have $f(T, a) = \int_{t=0}^{\Delta_t} \nabla \cdot \frac{1}{P_e} \nabla T - \nabla \cdot (vT) + S(z, t; a) \mathrm{d}t$ with the initial state at $t = 0$ determined by 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's 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.

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 states and actions. We provide more intuition for these regularities, and then introduce 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, c.f., 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 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; 2017a). 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. (2017a) 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 provide domain-specific features from image-like inputs.

Even though these work can handle high-dimensional states in PDE control problems, they are limited to a finite number of actions. Furthermore, 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; 2017a) 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 an infinite dimensional vector space, e.g., the space of continuous functions over $\mathcal{Z}$. Procedurally, the agent-environment interaction is no different: at each step, the agent selects a function $a_t \in \mathcal{A}$ at the current state $x_t \in \mathcal{X}$, then transit to next 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 through an "adapter" converts those values to an infinite dimensional action appropriately as an input of the PDE. As an example, 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 $\mathcal{Z}$), so 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. For some problems, the set of controllable locations are not necessarily fixed, and might change. For example, if one of the elements breaks or some additional ones are added, 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. The action descriptors can 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 a 2D convection-diffusion equation (1). An action location is $c \in \mathcal{Z}$ and determines the location where the action can be freely selected by the agent. The set of all action locations defines the finite and ordered set $\mathcal{C} = (c_1, \dots, c_k)$, called action descriptor. Here the number of actions k is finite. Given each action location c_i and the state x, the agent generates an action $u^{(i)} \in \mathbb{R}$. Given the set of actions $u = (u^{(1)}, \dots, u^{(k)})$ and the action descriptors $\mathcal{C}$, we have an "adapter" $I : \mathcal{C} \times \mathbb{R}^k \to \mathcal{F}(\mathcal{Z})$ which outputs a function defined over domain $\mathcal{Z}$. This function is the action given to the MDP with infinite dimensional action space, i.e., $a_t = I(\mathcal{C}, u_t)$.

The adapter can be thought of as a decoder from a finite-dimensional code to a function-valued code. The choice of I is not unique. One particular choice, which is useful in the context of PDE control, is defined as follows: Based on the action descriptors $\mathcal{C}$, define a partition of $\mathcal{Z}$ and denote it by $(A_1, \dots, A_k)$, 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 it centre. Another example would be Voronoi diagram corresponding to centres in $\mathcal{C}$. We then define

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

which assigns the value of $u^{(i)}$ to any point within A_i.

To build a physical intuition, consider the Heat Invader

problem, where the action descriptors $\mathcal{C}$ correspond to the spatial locations of the heater/cooler elements of the air conditioners. Furthermore, the partitions A_i corresponds to the region that each element takes. When we set the temperature at location c_i to a certain value $u^{(i)}$, in the model (3), the value of the whole region A_i takes the same value $u^{(i)}$. Note that this partitioning of the domain is somehow similar to the meshing in the finite element method.

Our action descriptor formulation allows exploiting the spatial regularity of the action as it explicitly encodes information that specify correlations among different action components, i.e., locations where different action components are applied. Moreover, 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.

The significance of this formulation parallels the gains observed when moving from tabular state representation to function approximators. Therefore, even though only a finite number of action dimensions will practically ever be queried, the generalization obtained across action-dimensions should result in significantly improved learning speed, even with high-dimensional continuous actions.

We would like to remark that this formulation encompasses more specific cases such as a finite-dimensional action space $\mathcal{A} = \mathbb{R}^k$. To see this, choose $\mathcal{Z} = \{1, \ldots, k\}$ and define $A_i = \{e_i\}$ for $i = 1, \ldots, k$ with e_i being the unit vector corresponding to the i-th dimension of $\mathbb{R}^k$.

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, then, cannot be directly applied to learning for these (extremely) high-dimensional continuous action problems with regularities. We now 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 get 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 is $a_t = \pi_{\theta^\mu}(x_t, \cdot)$ which is function-valued. With the descriptor 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 $u_t \in \mathbb{R}^{|\mathcal{C}|}$. Through the use of $I(\mathcal{C}, u_t)$, it is converted to the correct number of dimensional action whatever the domain needed. 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) is an appropriate choice to learn such a deterministic 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. 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 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 $\nabla_a Q(x, a; \theta^Q)|_{a=\pi_{\theta^\mu}(x)}$ 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 showed in Algorithm 1, which can efficiently scale to extremely high-dimensional continuous actions while capturing the intrinsic regularities. From implementation, the change to vanilla DDPG is to make the actor network have only one output unit, and hence we think of it as a function space. Given a state, we are able to specify a function within that space, and then it can be evaluated on k action descriptors to get the desired action vector $u \in \mathbb{R}^k$.

5. Theoretical Insights

We provide theoretical evidence that learning a policy that explicitly incorporates the action location 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 differ-

Algorithm 1 DDPG with Action Descriptors

Initialize a random process $\mathcal{N}$ for exploration
buffer B for experience replay
actor and critic networks:
$\pi(\cdot,\cdot;\theta^\mu) : \mathcal{X} \times \mathcal{Z} \mapsto \mathbb{R}, Q(\cdot,\cdot;\theta^Q) : \mathcal{X} \times \mathbb{R}^k \mapsto \mathbb{R}$
target actor and critic networks $\pi(\cdot,\cdot;\theta^{\mu'}), Q(\cdot,\cdot;\theta^{Q'})$
set of action descriptors $\mathcal{C}, |\mathcal{C}| = k$
target network update rate τ
for t = 1, 2, ... **do**
  Observe x_t, take action $u_t + \mathcal{N} \in \mathbb{R}^k$ where each component $u_t^{(j)} = \pi(x_t, c_j; \theta^\mu), \forall c_j \in \mathcal{C}$
  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
  // use B_N and corresponding targets
  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

ent policy spaces. The first is the space of policies with certain spatial regularity (Lipschitzness in z) explicitly encoded and 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 a 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 our best knowledge, there is no estimation error bound so far for deep reinforcement learning algorithms. Therefore, our results only provide a possible mathematical insight rather than a complete picture of the sample efficiency bound.

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 the appendix 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 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 then 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 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 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 $M_\varepsilon = c(\frac{\varepsilon}{2L})^d$, 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:

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

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:*

$$\log \mathcal{N}(\varepsilon, \Pi_L) \leq c_1 \left(\frac{L}{\varepsilon}\right)^d + \log \mathcal{N}(\varepsilon),$$

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

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 $0 \leq \alpha < 1$.[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 here either. 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 would be

$$n^{\frac{1+d+2\alpha}{1+\max\{2\alpha, d\}}},$$

which becomes significant as α, the complexity of $\Pi_L|_z$, grows. This suggests the possible benefit of explicitly incorporating the spatial regularity in the $\mathcal{Z}$ space, in terms of sample complexity of learning.

6. Experiments

We now empirically show that our approach can be easily scaled to large continuous action dimensions while maintaining competitive performance. We also verify that exploiting the regularity improves both speed of learning and stability, as the action dimension increases. We compare vanilla DDPG, and DDPG with separate Neural Networks (NN) for each action component, to our DDPG with Action Descriptors. The network architecture is the same for both vanilla DDPG and our DDPG with descriptor, except in the last layer, where DDPG outputs a k-dimensional action vector and DDPG with Action Descriptors only uses one output unit. We tested on two domains. The first is a simple PDE Model domain, and the second is the Heat Invader problem, which has been described throughout this work. It is a more difficult problem with a stronger requirement on regularities.

[3] This particular choice of metric entropy holds for some non-parametric function spaces, such as many RKHS. We should warn the reader that 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.

6.1. Results on the PDE Model domain

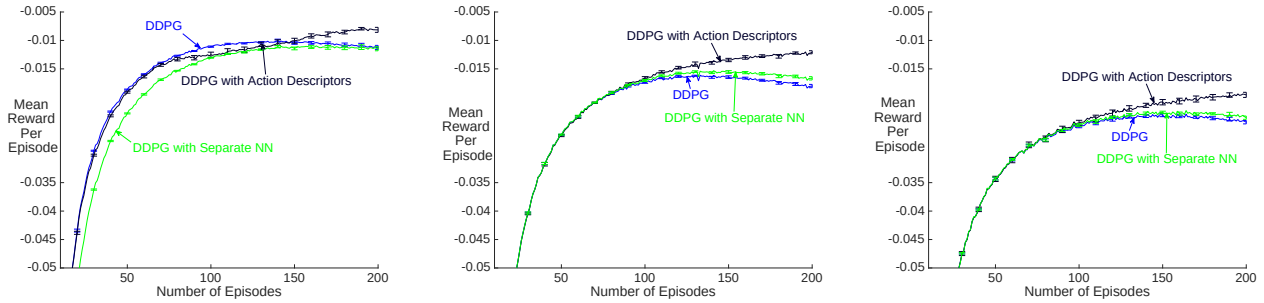
The PDE model domain is using the 2D heat equation as the underlying transition dynamic (please see A.4 for details). The infinite-dimensional state and action spaces are discretized to $\mathcal{X} \subset \mathbb{R}^{d \times d}$ and $\mathcal{A} \subset [-1, 1]^{d \times d}$ (we get a vector from algorithm and reshape it to a matrix). One can intuitively understand the domain as following. At any location (i, j) (an entry in the matrix $x_t \in \mathcal{X}$) which has high temperature, it likely transfers its heat content to nearby locations, and vice versa. Note that the boundary condition implies certain spatial regularity. Since all initial state values are positive, the 0 boundary value means the heat value near the boundary is likely to decrease.

Figure 1 shows the comparison between our DDPG with Action Descriptors and the other two competitors, as the number of state and action dimension increases in $d^2 \in \{36, 100, 256\}$. One can see that using action descriptors consistently outperforms the other algorithms. On this domain both vanilla DDPG, and the one with separate NN cannot work well, in fact, they showed certain level of divergence when dimension is high $d^2 = 100, 256$. Both of them converge when $d^2 = 36, a_t \in \mathbb{R}^{36}$; DDPG with separate NN shows slightly lower sample efficiency as we expected. However, in any action dimension, our DDPG with Action Descriptors continues improving.

6.2. Results on the Heat Invader domain

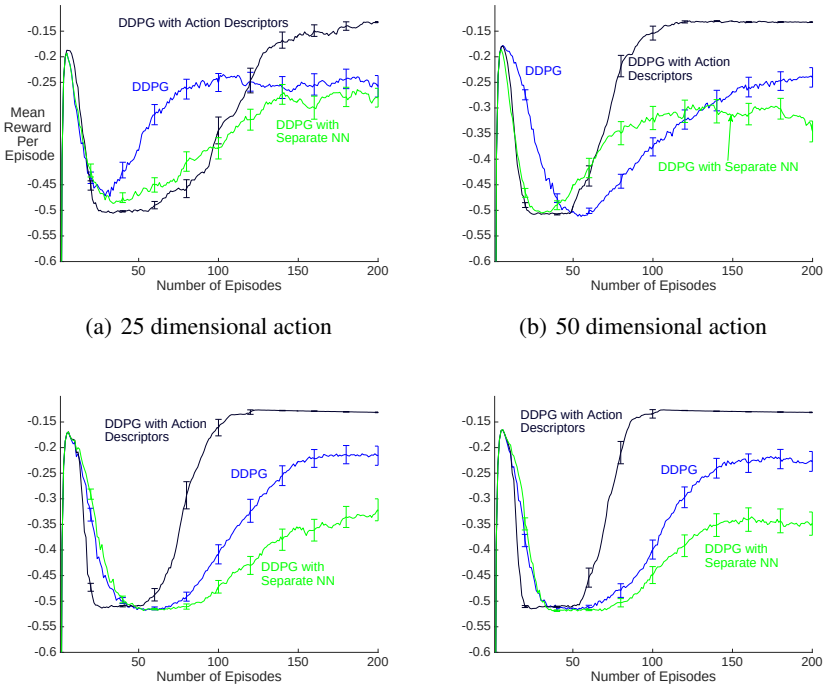
The underlying state transition dynamics of this domain is the PDE as described by Equation (1). In an ideal case, there are infinitely many air conditioners on the floor, and at each time step we can control the temperature for each air conditioner. In simulation, the state and action spaces are discretized. The state space is discretized to $\mathcal{X} \subset \mathbb{R}^{50 \times 50}$, giving a total of 2500 measurement locations on the floor. We experimented with a variety of discretization levels for the action space: $a_t \in [-0.5, 0]^k, k \in \{25, 50, 100, 200\}$. Because the air conditioners are uniformly distributed on the floor, in a square area, we can topologically think of it as the area $[-1, 1]^2$. Hence we can design the set of descriptors as $\mathcal{C} \subset [-1, 1]^2$, where $|\mathcal{C}| = k$ and the descriptors are uniformly distributed points in that area (please see A.5 for details).

Figure 2 shows comparisons across different control dimensions. On this more difficult domain, requring a stronger regularity, it becomes clear that DDPG with sepa-



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

Figure 1. Relative performance on the PDE-Model domain, with increasing dimension of the state and action space. The results are averaged over 30 runs, with standard error displayed. DDPG with Action Descriptors consistently outperforms DDPG and DDPG with separate NN. The latter two can converge to a policy only when the dimension is reasonably low, both them degrade after the action dimension reaches 100, even with a careful parameter optimization on performance in the last 10% of episodes.



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

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

Figure 2. Results of mean reward per episode vs. episodes on the Heat Invader domain, with an increasing action dimensions. The results are averaged over 50 runs. Our DDPG with action descriptors consistently outperform other algorithms across dimensions, and vanilla DDPG shows a higher sample efficiency than the one with 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 be sufficiently fine 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 decreasing in a few episodes, the fan turns off and the agent has to learn to improve its policy.

rate NN has a lower sample efficiency than vanilla DDPG, and the former even shows divergence after the dimension reaches 50. For $k = 100$ and 200, DDPG with Action Descriptors learns a significantly better and more stable policy, even though it only has one output unit, regardless of the dimension of action. DDPG, and its variant of using separate NN, on the other hand, has the number of outputs growing with k. 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 demonstrated a general framework of using reinforcement learning for PDE control, which theoretically could have infinite-dimensional state and action spaces and practically has very high-dimensional continuous states and actions with special regularities to handle. We then proposed the notion of action descriptors, to enable reinforcement learning algorithms—such as deterministic policy gradient algorithm—to be used for these problems. Theoretical evidences are provided to illustrate why our approach could

have better sample efficiency. Our strategy enables the architecture to easily scale with increasing action dimension. We show that with the same neural network architecture, except the output layer, we can obtain significantly better scaling as dimension increases as well as better empirical performance on two PDE domains. We believe our proposed strategy is a potential substitution for conventional methods of solving PDE control problems, and also a promising and effective way to solve more classical reinforcement learning control problems with extremely high-dimensional continuous action spaces.

References

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

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

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

Belletti, Francois, Haziza, Daniel, Gomes, Gabriel, and Bayen, Alexandre M/. Expert level control of ramp metering based on multi-task deep reinforcement learning. *CoRR*, abs/1701.08832, 2017. URL `http://arxiv.org/abs/1701.08832`.

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

Borggaard, Jeff, Burns, John A., Surana, Amit, and Zietsman, Lizette. Control, estimation and optimization of energy efficient buildings. In *American Control Conference (ACC)*, pp. 837–841, 2009.

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

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

Burns, John A, He, Xiaoming, and Hu, Weiwei. Feedback stabilization of a thermal fluid system with mixed boundary control. *Computers & Mathematics with Applications*, 2016.

Deisenroth, Marc Peter, Neumann, Gerhard, and Peters, Jan. A survey on policy search for robotics. *Found. Trends Robot*, 2(1–2):1–142, August 2013. ISSN 1935-8253.

del R Millán, José, Posenato, Daniele, and Dedieu, Eric. Continuous-Action Q-Learning. *Machine Learning*, 2002.

Dulac-Arnold, Gabriel, Denoyer, Ludovic, Preux, Philippe, and Gallinari, Patrick. 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.

Dulac-Arnold, Gabriel, Evans, Richard, van Hasselt, Hado, Sunehag, Peter, Lillicrap, Timothy, Hunt, Jonathan, Mann, Timothy, Weber, Theophane, Degris, Thomas, and Coppin, Ben. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

Duriez, Thomas, Brunton, Steven L, and Noack, Bernd R. *Machine Learning Control–Taming Nonlinear Dynamics and Turbulence*, volume 116 of *Fluid mechanics and its applications*. Springer, 2016.

E, W., Han, J., and Jentzen, A. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *ArXiv e-prints*, June 2017.

Farahmand, Amir-massoud, Ghavamzadeh, Mohammad, Szepesvári, Csaba, and Mannor, Shie. Regularized fitted Q-iteration for planning in continuous-space Markovian Decision Problems. In *Proceedings of American Control Conference (ACC)*, pp. 725–730, June 2009.

Farahmand, Amir-massoud, Ghavamzadeh, Mohammad, Szepesvári, Csaba, and Mannor, Shie. Regularized policy iteration with nonparametric function spaces. *Journal of Machine Learning Research (JMLR)*, 17(139):1–66, 2016a.

Farahmand, Amir-massoud, Nabi, Saleh, Grover, Piyush, and Nikovski, Daniel 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.

Farahmand, Amir-massoud, Nabi, Saleh, and Nikovski, Daniel N. Deep reinforcement learning for partial differential equation control. In *American Control Conference (ACC)*, 2017a.

Farahmand, Amir-massoud, Pourazarm, Sepideh, and Nikovski, Daniel N. Random projection filter bank for time series data. In *Advances in Neural Information Processing Systems (NIPS)*, 2017b.

Foures, DPG, Caulfield, Colm-cille, and Schmid, Peter J. Optimal mixing in two-dimensional plane poiseuille flow at finite Péclet number. *Journal of Fluid Mechanics*, 748:241–277, 2014.

Gaskett, Chris, Wettergreen, David, and Zelinsky, Alexander. Q-Learning in Continuous State and Action Spaces. In *Advanced Topics in Artificial Intelligence*. 1999.

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

Györfi, László, Kohler, Michael, Krzyżak, Adam, and Walk, Harro. *A Distribution-Free Theory of Nonparametric Regression*. Springer Verlag, New York, 2002.

He, Ji, Chen, Jianshu, He, Xiaodong, Gao, Jianfeng, Li, Lihong, Deng, Li, and Ostendorf, Mari. Deep reinforcement learning with an unbounded action space. *CoRR abs/1511.04636*, 2015.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

Kim, Kwang Jin, Choi, Hyouk Ryeol, and Tan, Xiaobo. Biomimetic robotic artificial muscles. *World Scientific*, 2013.

Kober, Jens, Andrew Bagnell, J, and Peters, Jan. Reinforcement learning in robotics: A survey. 32:1238–1274, 09 2013.

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

Lazaric, Alessandro, Ghavamzadeh, Mohammad, and Munos, Rémi. Analysis of classification-based policy iteration algorithms. *Journal of Machine Learning Research (JMLR)*, 17(19):1–30, 2016. URL `http://jmlr.org/papers/v17/10-364.html`.

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

Lillicrap, Timothy P., J. Hunt, Jonathan, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. 2016.

Lions, Jacques Louis. Optimal control of systems governed by partial differential equations. 1971.

Mohri, Mehryar, Rostamizadeh, Afshin, and Talwalkar, Ameet. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN 026201825X, 9780262018258.

Montgomery, William and Levine, Sergey. Guided policy search as approximate mirror descent. 2016.

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

Popescu, M. C., Petrisor, A., and Drighiciu, M. A. Modelling and simulation of a variable speed air-conditioning system. In *2008 IEEE International Conference on Automation, Quality and Testing, Robotics*, volume 2, pp. 115–120, May 2008. doi: 10.1109/AQTR.2008.4588805.

Richards, P.I. Shockwaves on the highway. 1956.

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

Schulman, Jhon, Levine, Sergey, Moritz, Philipp, Jordan, Michael, and Abbeel, Pieter. Trust region policy optimization. 2015.

Schulman, John, Moritz, Philipp, Levine, Sergey, Jordan, Michael, and Abbeel, Pieter. High-dimensional continuous control using generalized advantage estimation. *International Conference on Learning Representations*, 2016.

Silver, David, Lever, Guy, Heess, Nicolas, Degris, Thomas, Wierstra, Daan, and Riedmiller, Martin. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, pp. I–387–I–395. JMLR.org, 2014.

Steinwart, Ingo and Christmann, Andreas. *Support Vector Machines*. Springer, 2008.

Sunehag, Peter, Evans, Richard, Dulac-Arnold, Gabriel, Zwols, Yori, Visentin, Daniel, and Coppin, Ben. Deep reinforcement learning with attention for slate markov decision processes with high-dimensional states and actions. *arXiv preprint arXiv:1512.01124*, 2015.

Sutton, Richard S. and Barto, Andrew G. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

Sutton, Richard S., McAllester, David, Singh, Satinder, and Mansour, Yishay. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS - 12)*, 2000.

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

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

van Hasselt, Hado and Wiering, Marco A. Reinforcement Learning in Continuous Action Spaces. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007.

Wang, Haiyan, Wang, Feng, and Xu, Kuai. Modeling information diffusion in online social networks with partial differential equations. *CoRR*, abs/1310.0505, 2013.

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

A. Appendix

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 that is an ε-cover for $\mathcal{F}$, and 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

The proof for proposition 1 is as following.

Proof. First we show that $\underline{\Pi}$ provides an ε-covering 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 $\varepsilon/2$-cover 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. Because of the 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

$$[\mathcal{N}(\varepsilon/2)]^{M_\varepsilon}. \tag{4}$$

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

To provide 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

$$\begin{aligned} |\pi_\theta(x,z) - \tilde{\pi}(x,z)| \leq & |\pi_\theta(x,z) - \pi_\theta(x,c_i)| + \\ & |\pi_\theta(x,c_i) - \tilde{\pi}(x,c_i)| + \\ & |\tilde{\pi}(x,c_i) - \tilde{\pi}(x,z)| \\ \leq & L\|z - c_i\| + \\ & \left| 0.5\varepsilon \left\lfloor \frac{\pi_\theta(x,c_i)}{0.5\varepsilon} \right\rfloor - \pi_\theta(x,c_i) \right| + 0 \\ \leq & \varepsilon/2 + \varepsilon/2 = \varepsilon \end{aligned} \tag{5}$$

So $\tilde{\Pi}_\varepsilon$ provides an ε-cover for Π_L. 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 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., $\tilde{\pi}(x, c_i) - 4\varepsilon/2, \tilde{\pi}(x, c_i) - 3\varepsilon/2, \dots, \tilde{\pi}(x, c_i) + 4\varepsilon/2$.

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

$$\mathcal{N}(\varepsilon) \times 9^{M_\varepsilon}. \tag{6}$$

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. Experimental settings

Neural network architecture. The implementation tool is Tensorflow. We basically follow the same neural network architecture as introduced in the original DDPG paper (Lillicrap et al., 2016). We have 3 convolutional layers without pooling, and each has 32 filters. The kernel sizes are 4×4 for the first two layers and then 3×3 for the third one. Batch normalizations (Ioffe & Szegedy, 2015) are used in the convolutional layers. The convolutionary layers are followed by two fully connected layers and each has 200 relu units. The output layer in the actor network is sigmoid on the heat invader domain and is tanh on the PDE model domain. The output layer of ciritic network use the same convolutional layer setting with that of the actor network, but it does not has activation function in the output layer, and a L_2 weight decay factor as 0.001 was used after the convolutional layers. The final layer weights and biases of both the actor and critic were initialized from a uniform distribution $[-0.0003, 0.0003]$ and all other parameters are initialized by using Xavier (Glorot & Bengio, 2010).

As for the DDPG with separate NN, we used same convolutional layers across all neural networks, but replicate the full connected layers k times if the action is in $\mathbb{R}^k$.

For DDPG with action descriptors, we used exactly the same architecture as DDPG, execpt the output layer has only one output irrespective of action dimensions. The *action descriptor* comes into the neural network immediately after the convolutional layers.

Parameter Setting. Across all experiments, we used experience replay with buffer size $20,000$ and the batch size is 16. Each episode has at most 40 steps. The discount rate is 0.99. We have the exploration noise as gaussian distribution $\mathcal{N}(u_t, \frac{1.0}{episodes})$. For each action dimension, we sweep over parameters as following. Actor learning rate α from $\{0.000001, 0.000005, 0.00001, 0.0001\}$, critic learning rate is from $\alpha \times \{1.0, 5.0, 10.0, 20.0, 40.0\}$ (intuitively critic learning rate should be larger, and indeed we tried that with a smaller cirtic learning rate the algorithm cannot work well).

Performance measure. We take a similar approach with the previous work (Farahmand et al., 2017a), computing the mean reward per step for each episode i averaged over of number of runs N. 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 of our experiments. When pick up best parameters, we compute $Evaluate = \sum_{i=m}^{n} R^{(i)}$. On PDE model, we use $m = 180, n = 200$ while on heat invader, we use $m = 150, n = 200$. We choose those ranges to avoid fast converge at the begining but divergence later, which actually happened more obvious for our competitor algorithms.

A.4. Additional details on PDE model

The PDE model domain is our self-defined domain based on $2-$D heat equation.

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

where h is the heat, a function of location $z = (x, y) \in \mathbb{R}^2$ and time t, α is some constant depends on physical material, called thermal diffusitivity. We let $\alpha = 1$ for simplicity. Note that since we think the space as a 2-D plane, we use a matrix to discretize it and hence each entry in the matrix indicates the heat value at that particular location. Use finite difference method to approximate the second order derivative, we can implement the state $d \times d$ matrix transition as following.

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

c is some constant to scale the change and the $action$ results from control, the effect outside of the physical system. This is how our discretized state transition derived. For each $x \in \mathcal{X}$, let $x^{(i,j)}$ denotes the element in the ith row and j column of the matrix x. Let x_t and a_t be the state and action, respectively, at time step t. Then 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 $\delta_s = 0.1$ of the finite difference method. For boundary values, where $i - 1$ or $i + 1$ becomes 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

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

One can intuitively understand this as the heat always move from the high heat region to low heat region, and hence for each entry in the matrix we take the sum of four of its neighbors as the heat value coming from nearby region,

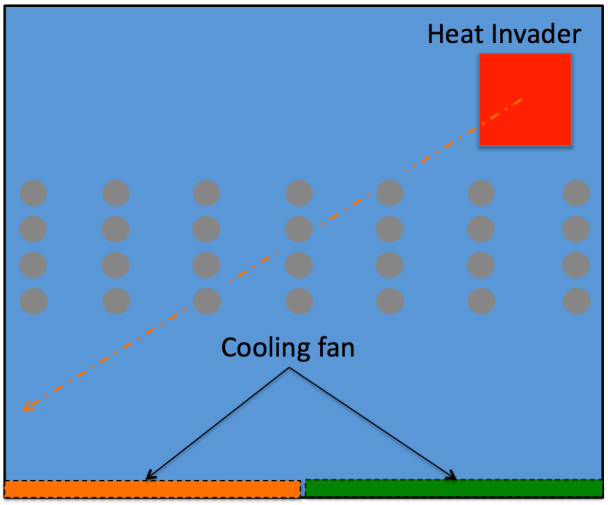


Figure 3. Heat invader domain. Image is modified from the paper (Farahmand et al., 2017b). The solid circle means the air conditioners. The actual number of air conditioners is different with what the figure showed.

then if this value is larger than the entry's current value, we should expect an increase in this entry. At each episode, the state values are uniformly initialized from $[0, 1]^{d\times d}$. Note that at each step, the update to each value in the state happens simultaneously. The boundary values are setted 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, which is the time discretization level.

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

The way of choosing action descriptors is to simply uniformly choose from the space $[-0.5, 0.5]^2$. There is no particular reason for our choice, we believe other way such as $[-1, 1]$ can also work, as long as the descriptors are uniformly chosen from a square area.

A.5. Additional details on Heat Invader

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

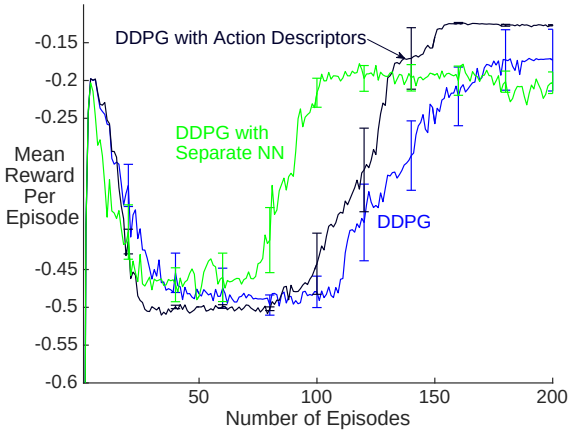
$$r(x_t, a_t, x_{t+1}) = -\text{cost}(a_t) - \int_{z\in\mathcal{Z}} I(|T(z, t+1)| > T^*(z, t+1))dz \tag{7}$$

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 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 of averaging less runs when 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 comes from the "uncomfortable temperature" shows effect and hence the performance drops. Later while the agent is learning a better policy, the performance starts improving again.

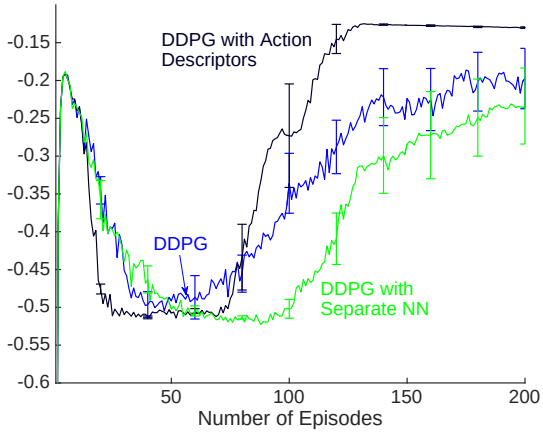
We design the set of action descriptors as following. When the action dimension $k \leq 50$, we think the air conditioners are located on a single line in the middle of the room, hence we pick the descriptors uniformly from the segment $[-1, 1]$. When the action dimension $k = 100$, we think there are two lines on the middle of the room and hence we design the descriptors as two dimensional denoted as $(x, y) \in [-1, 1]^2$, 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 while the output action from the agent has dimension less then 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 sanity check, we also conducted experiments when 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. When the dimension is 1, the two algorithms are almost identical, except that DDPG with Action Descriptors has two additional input units, as the descriptor is in $\mathbb{R}^2$.

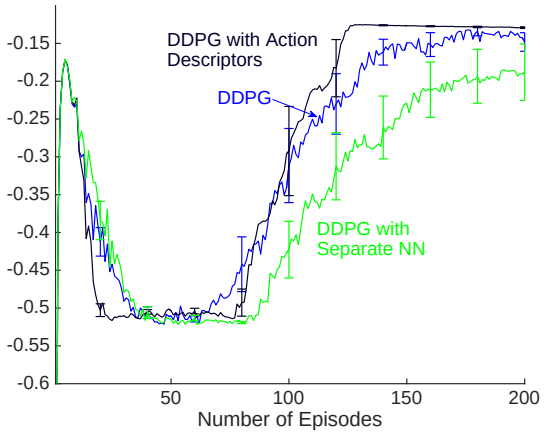
Due to computational resource restriction, to generate the figure using the uniform airflow, we use 10 runs to find best parameter settings and then do 50 runs for that setting.



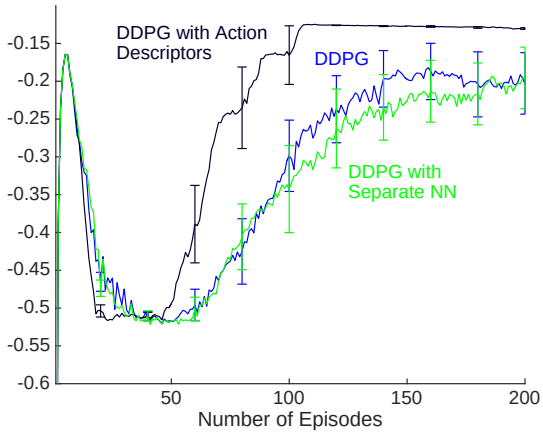
(a) 25 dimensional action



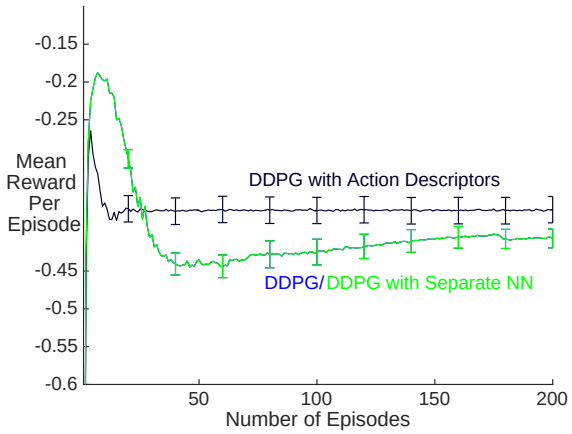
(b) 50 dimensional action



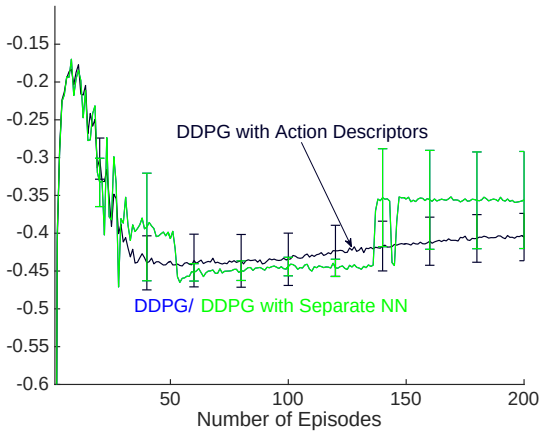
(c) 100 dimensional action



(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 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.