2017 American Control Conference
Sheraton Seattle Hotel
May 24–26, 2017, Seattle, USA

Deep Reinforcement Learning for Partial Differential Equation Control

Amir-massoud Farahmand, Saleh Nabi, Daniel N. Nikovski

Abstract— **This paper develops a data-driven method for control of partial differential equations (PDE) based on deep reinforcement learning (RL) techniques. We design a Deep Fitted Q-Iteration (DFQI) algorithm that works directly with a high-dimensional representation of the state of PDE, thus allowing us to avoid the model order reduction step common in the conventional PDE control design approaches. We apply the DFQI algorithm to the problem of flow control for time-varying 2D convection-diffusion PDE, as a simplified model for heating, ventilating, air conditioning (HVAC) control design in a room. We also study the transfer learning of a policy learned for a PDE to another one.**

I. INTRODUCTION

This paper develops a data-driven method for control of partial differential equations (PDE) based on deep reinforcement learning (RL) techniques. This work is motivated by two recent developments in machine learning and control. The first is that the PDE control problem can be formulated as a reinforcement learning problem [1]. Reinforcement learning is the problem of adaptively finding an optimal policy (i.e., controller) for an unknown nonlinear stochastic dynamical system without the knowledge of the dynamics—using only interaction data [2], [3]. The second motivating development is the recent successes of deep learning-based approaches to RL, which has been applied to solve complex problems such as playing Atari games [4], the board game of Go [5], and the visual control of robotic arms [6].

We describe a deep learning-based RL algorithm, called Deep Fitted Q-Iteration (DFQI), that can directly work with the state of PDE, a theoretically infinite-dimensional vector, thus allowing us to potentially overcome the limitations of classical approaches to PDE control. As an example, we consider the problem of optimal control for time-varying 2D convection-diffusion PDE, as a simplified model for heating, ventilating, air conditioning (HVAC) control design in a room, our motivating application. The proposed approach is general and can be applied to other PDE control problems too.

To motivate the reason behind formulating the PDE control problem as an RL problem, we first briefly summarize the conventional approaches to PDE control. These approaches can be classified into two categories. The first category is called *reduce-then-design* approach, in which the PDE is numerically approximated by a finite-dimensional ordinary differential equation (ODE). The resulting ODE is high-dimensional, so a model order reduction method is applied at this stage. A linear controller is then designed based on the reduced-order ODE model. An optimal controller is typically designed based on optimizing a quadratic cost functional [7]. The other alternative is the *design-then-reduce* approach, in which one directly designs a controller for the PDE, for example using distributed parameter LQR theory, and then later use numerical approximations to find the control gains [8], [9], [10]. Also refer to [11] for a survey on many methods for the turbulence control problem.

Even though the conventional approaches are elegant, they have some drawbacks, especially when they must be deployed to control a real-world system such as the temperature of a room in an HVAC system. The first drawback is that the controller is typically only valid in a small neighbourhood of the nominal PDE for which the controller has been designed. If the boundary conditions are significantly changed, for instance because some furniture is added to the room, the controller should be re-designed to maintain the performance. This is often impractical as this re-designing requires the knowledge of a control engineer who has expertise in designing PDE controllers. The second drawback is that the linear control design ignores potentially useful nonlinear phenomenon inherent in fluid dynamics problems [12]. Designing a controller based on this simplified model might lead to a suboptimal solution.

It is desirable to have a controller design procedure that has minimal assumptions about the PDE, does not need to know the model of the PDE, and is completely data-driven. In other words, it is desirable to have a method that finds a reasonably good controller only based on data observed from the plant. These desiderata can be achieved by formulating the PDE control problem as an RL problem, as has recently been shown [1], or some other flexible data-driven approaches such as genetic programming-based method of [13]. The focus of this work is on the RL-based approach.

Nonetheless due to the curse of dimensionality, solving an RL problem with a high-dimensional state space is difficult. The PDE control problem is particularly challenging as the state of a PDE is an infinite-dimensional vector (or a very high-dimensional vector in simulations). This makes the value function approximation, a step required by many RL algorithms, quite different from the usual problems for which RL algorithms have been applied so far, as almost all of them are designed to deal with problems with finite, and often low, dimensional state spaces. The challenge is to design a method that can easily deal with very high-dimensional state spaces, e.g., 2500 dimensions in our experiments. Generally speaking, the only way to deal with the curse of

All authors are with Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA, USA. {farahmand, nabi, nikovski}@merl.com

978-1-5090-5992-8/$31.00 ©2017 AACC

dimensionality is to exploit the intrinsic regularities of the problem, such as the smoothness of the value function or its sparsity in a certain set of basis functions, or any other type of structure or regularities that can make the learning problem easier. Refer to [14] for results on difficulty of learning in the context of supervised learning, and [15] for the role of regularities in RL and some sample-complexity results. So despite the fact that the value function of a PDE control problem resides in a very high-dimensional space, an RL algorithm that can exploit the value function's regularities might still perform well.

One source of regularities that can be exploited to solve PDE control problems comes from noticing the similarity of an infinite-dimensional object such as the 2D/3D temperature scalar field in a convection-diffusion problem (or more generally, any scalar/vector fields for other types of PDEs) with a 2D/3D image in computer vision problems. Both of these objects are scalar (or vector) fields, one defining the solution of a PDE and the other defining the colour of an image. They often have much spatial regularities, such as local smoothness and other spatial patterns, which can potentially be exploited by a learning algorithm. This connection can be seen most clearly by noticing that visualizing the solution of a PDE is indeed nothing but representing the state of the PDE as an image.[1]

This similarity has motivated us to design RL algorithms that directly work with the PDE's infinite-dimensional state vector and treat it as if the state is an image. In a previous work [1], we suggested to use the Regularized Fitted Q-Iteration (RFQI) algorithm [16] with a reproducing kernel Hilbert space (RKHS), as the value function approximator, for the PDE control problem. An RKHS is a suitable choice because it only requires us to define a kernel function (i.e., a measure of similarity) between two states of a PDE. Seeing the state of a PDE as an image means that we only need to define a kernel function between two images, which is not a difficult task.

This paper takes a different approach. We use a deep neural network (DNN) [17], particularly a deep convolutional network (ConvNet), as the estimator of the value function. The motivation is that ConvNets are quite suitable to extract problem-dependent features from image-like inputs. They have mitigated the need to manually design features for many computer vision problems. They have also been successfully used for learning to play computer games [4], perform visual-servoing [6] (both formulated as RL problems) and imitating a human driver using camera input [18] (formulated as a supervised learning problem). In this work, we describe the DFQI algorithm and apply it to two related PDE control problems. We empirically compare the performance of DFQI and RFQI. We also investigate the transfer learning problem within the context of RL and PDE control.

II. PDE CONTROL AS A MARKOV DECISION PROCESS

We describe how a PDE control problem can be formulated as a Markov Decision Process (MDP), which is a mathematical framework to define an RL problem [2].[2]

A *finite-action discounted* MDP is a 4-tuple $(\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{X}$ is a measurable state space, $\mathcal{A}$ is a finite set of actions, $P : \mathcal{X} \times \mathcal{A} \to \mathcal{M}(\mathcal{X})$ is the transition probability kernel, and $\mathcal{R} : \mathcal{X} \times \mathcal{A} \to \mathcal{M}(\mathbb{R})$ is the immediate reward distribution. The constant $0 \leq \gamma < 1$ is the discount factor. We use $r(x, a)$ to denote the expected value of the random variable $R \sim \mathcal{R}(\cdot|x, a)$.

We identify these quantities within the PDE control context. For concreteness, we consider a time-varying convection-diffusion PDE as a model to describe spatio-temporal evolution of temperature field in presence of a time-varying velocity field, but the basic idea behind formulating the PDE control problem as an RL problem is more general and can be applied to other PDEs too.

We denote the domain of PDE by $\mathcal{Z} \subset \mathbb{R}^2$ (or $\mathbb{R}^3$), which might represent a confined region, e.g., a room. Its boundary is denoted by $\partial\mathcal{Z}$. The time set is denoted by I, e.g., $I = \mathbb{R}_+$. Let us denote the temperature field by a time-dependent scalar field $T : \mathcal{Z} \times I \to \mathbb{R}$, and $v : \mathcal{Z} \times I \to \mathbb{R}^2$ (or $\mathbb{R}^3$) as the time-varying velocity field, e.g., an airflow field. Denote $S : \mathcal{Z} \times I \to \mathbb{R}$ as the time-varying source. The convection-diffusion equation is

$$\frac{\partial T(z,t)}{\partial t} = \frac{1}{\text{Pe}} \nabla^2 T - \nabla \cdot (vT) + S(z,t), \qquad (1)$$

in which $\text{Pe} = \frac{L v_c}{D}$ is the Péclet number with L being the characteristic length of the domain, v_c being the characteristic velocity, and $D : \mathcal{Z} \to \mathbb{R}$ being the thermal diffusivity constant. The velocity field is divergence-free to respect the continuity (conservation of mass), i.e., $\nabla \cdot v = 0$. For a given velocity and source field, the temperature field is $T(\cdot, t) \in \mathcal{T}$, in which $\mathcal{T}$ is the space of all temperature fields for a fixed time. We also use $\mathcal{S}$ to denote the space of all $S(\cdot, t)$. If there is no chance of confusion, we may simply use T, v, and S to refer to $T(\cdot, t)$, $v(\cdot, t)$, and $S(\cdot, t)$ for a particular t.

Let us partition the boundary ∂Z to ∂Z_1 and ∂Z_2, and impose the Dirichlet and Neumann boundary conditions:

$$\begin{aligned} T(z,t) &= T_b(z,t), \qquad && \forall z \in \partial Z_1 \\ \vec{n} \cdot \nabla T(z,t) &= 0. && \forall z \in \partial Z_2 \end{aligned}$$

The Neumann boundary condition signifies an insulated temperature surface and the Dirichlet boundary condition defines a prescribed temperature surface, e.g., provided by the HVAC unit.

We consider a PDE control problem in which the dynamics is controlled by changing the boundary temperature $T_b(z, t)$ (for $z \in \partial Z_1$) and flow velocity v. For example, the boundary temperature can be changed by turning on/off heaters or coolers on the walls of a room. The flow can be controlled

[1] Even though an image has a finite-dimensional representation (e.g., a 500×500-dimensional vector in an Euclidean space), it is effectively, from the numerical computation perspective, as high-dimensional as e.g., the temperature and/or flow field of a PDE.

[2] This section closely follows the same section from [1] for the convenience of the reader.

by using fans that induce a flow field in the room. For simplicity of our simulations, we assume that v can be chosen from a given set of divergence-free flows (we consider two different types of airflow fields in our simulations, as will be explained), but in a real physical system v is actually determined by the Navier-Stokes equations.

We only consider the case that the control commands (T_b and v) belong to a finite action (i.e., control) set $\mathcal{A}$ with $|\mathcal{A}| < \infty$:

$$\mathcal{A} = \{\, (T_b^a, v^a) \,:\, a = 1, \dots, |\mathcal{A}| \,\}\,.$$

This should be interpreted as choosing action a at time t leads to setting the boundary condition as $T_b(\cdot, t) = T_b^a$ and the flow velocity as $v(\cdot, t) = v^a(\cdot)$.

The dynamics of the PDE at time t is fully described when v, T, and S are all known.[3] The function v is a part of the action that we choose. So the rest defines the state of the system, which we denote by an infinite-dimensional vector $x = (T, S)$. The state space is $\mathcal{X} \triangleq \{\, x = (T, S) \,:\, T \in \mathcal{T}, S \in \mathcal{S} \,\}$.

We can compactly write the PDE as

$$\frac{\partial x}{\partial t} = g(x(t), a(t)),$$

in which both the domain and its boundary condition are implicitly incorporated in the definition of function g. To simplify and make it compatible with the MDP framework, we deal with a discrete-time version of the previous set of equations, which can be obtained by integration from time t to time $t + 1$. So we have

$$x_{t+1} = f(x_t, a_t).$$

The choice of 1 as the time step is arbitrary and could be replaced by any Δt, but for simplicity we assume it is indeed equal to 1. This deterministic dynamics can be generalized to stochastic dynamics by describing the temporal evolution of the PDE by a transition probability kernel:

$$X_{t+1} \sim \mathcal{P}(\cdot | X(t), a(t)).$$

We use X instead of x in order to emphasize that it is a random variable. For deterministic dynamics, $\mathcal{P}(x|X(t), a(t)) = \delta\left(x - f(X(t), a(t))\right)$, in which δ is Dirac's delta function.

We now describe how the reward function $r : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ might be specified. That function should reflect the desirability of the current state of the system as well as the cost of a selected action. We provide an example: Consider that the comfort zone of people in the room is denoted by $\mathcal{Z}_p \subset \mathcal{Z}$, and let T^* be the desirable temperature scalar field. This might be a constant or a spatially-varying temperature profile. One possible definition of the reward function is $r(x, a) = -\left[\int_{\mathcal{Z}_p} |T(z) - T^*(z)|^2 \mathrm{d}z + c_{\text{action}}(a)\right]$, in which $c_{\text{action}}(a)$ is the cost of choosing action a. This might include the cost of heater or cooler operation and the cost of using the fan(s) that generates the airflow field. In general, the reward can be any function of x, a, and the next state x'.

After identifying the state, action, and the reward function for a typical PDE control problem, we briefly introduce the concepts of policy and value functions from the MDP and RL literature. A measurable mapping $\pi : \mathcal{X} \to \mathcal{A}$ is called a deterministic Markov stationary policy, or simply *policy* in short. Following a policy π in an MDP means that at each time step t, we have $A_t = \pi(X_t)$.

For a policy π, the action-value function Q^π is defined as follows: Let $(R_t)_{t \geq 1}$ be the sequence of rewards when the Markov chain starts from a state-action (X_1, A_1) drawn from a positive probability distribution over $\mathcal{X} \times \mathcal{A}$ and the agent follows policy π. Then the action-value function $Q^\pi : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ at state-action (x, a) is defined as

$$Q^\pi(x, a) \triangleq \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} R_t \,\middle|\, X_1 = x, A_1 = a\right].$$

For a discounted MDP, we define the *optimal action-value* functions by $Q^*(x, a) = \sup_\pi Q^\pi(x, a)$ for all $(x, a) \in \mathcal{X} \times \mathcal{A}$. A policy π^* is *optimal* if it achieves the best values in every state, i.e., if $Q^{\pi^*} = Q^*$.

We say that a policy π is *greedy* with respect to (w.r.t.) an action-value function Q if $\pi(x) = \operatorname{argmax}_{a \in \mathcal{A}} Q(x, a)$ for all $x \in \mathcal{X}$. We define function $\hat{\pi}(x; Q) \triangleq \operatorname{argmax}_{a \in \mathcal{A}} Q(x, a)$ (for all $x \in \mathcal{X}$) that returns a greedy policy of an action-value function Q (If there exist multiple maximizers, a maximizer is selected in an arbitrary deterministic manner). Greedy policies are important because a greedy policy w.r.t. the optimal action-value function Q^* is an optimal policy. Hence, knowing Q^* is sufficient for behaving optimally.

The Bellman optimality operator $T^* : B(\mathcal{X} \times \mathcal{A}) \to B(\mathcal{X} \times \mathcal{A})$ is defined as

$$(T^* Q)(x, a) \triangleq r(x, a) + \gamma \int_{\mathcal{X}} \max_{a'} Q(y, a') \mathcal{P}(\mathrm{d}y | x, a). \tag{2}$$

The Bellman optimality operator has a property that its fixed point is the optimal action-value function, i.e., $Q^* = T^* Q^*$. Because of this property, value-based approaches to MDP and RL aim to find a good approximation to the fixed point of the Bellman optimality operator. Suggesting a method for doing so is the subject of the next section.

III. Deep Fitted Q-Iteration

The Deep Fitted Q-Iteration algorithm (DFQI) is an instance of the family of Approximate Value Iteration (AVI) (or Fitted Q-Iteration) algorithms [19], [20], [21], [16], [22], [23], [4]. These algorithms are based on approximately performing the Value Iteration (VI) algorithm. A generic VI algorithm iteratively assigns

$$Q_{k+1} = T^* Q_k.$$

Since T^* is a contraction mapping with Q^* being its fixed point, in the limit $Q_k \to Q^*$. For MDPs with large state

[3] Here we assume that the evolution of $S(z, t)$ can be described by another PDE with S as its state. If the dynamics of S depends on other variables too, they should be included as a part of the state too.

space (i.e., finite, but large number of states; or high-dimensional continuous), performing the exact VI is often impractical. In such cases, we can try AVI instead, that results in

$$Q_{k+1} \approx T^* Q_k,$$

in which Q_{k+1} is represented by a function from a function space $\mathcal{F}^{|\mathcal{A}|} : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$. The function space $\mathcal{F}^{|\mathcal{A}|}$ is often much smaller than the space of all measurable functions on $\mathcal{X} \times \mathcal{A}$.

In addition to the challenge of dealing with a large state space, we may also face the problem that the integral in the definition of $T^* Q_k$ (2) cannot be computed easily. For instance this might be because we do not have a direct access to $\mathcal{P}$, but instead we only have data from interacting with the dynamical system in the form of $\{(X_i, A_i, R_i, X'_i)\}_{i=1}^n$ with $(X_i, A_i) \sim \nu$ and $R_i \sim \mathcal{R}(\cdot|X_i, A_i)$ and $X'_i \sim \mathcal{P}(\cdot|X_i, A_i)$ (RL setting). To handle this setting, a key observation is that for any fixed measurable function Q, we have

$$\mathbb{E}\left[R(x,a) + \gamma \max_{a' \in \mathcal{A}} Q(X', a') \mid X = x, A = a\right] = (T^* Q)(x,a),$$

which means that the conditional expectation of samples in the form of $R(x,a) + \gamma \max_{a' \in \mathcal{A}} Q_k(X', a')$ is the same as $T^* Q_k$. Estimating this expectation given samples is the problem of regression, which is well-studied in the machine learning and statistics literature [24], [14], [25].

The difference between various AVI methods boils down to the choice of the function approximator $\mathcal{F}^{|\mathcal{A}|}$ and the way they perform regression to fit a function $Q_{k+1} \in \mathcal{F}^{|\mathcal{A}|}$ to $T^* Q_k$. Some choices for $\mathcal{F}^{|\mathcal{A}|}$ that have been proposed and studied are collection of trees [19], RKHS with regularized regression [16], and deep ConvNet, in an architecture called Deep Q-Network (DQN) [4]. These algorithms have also been studied theoretically. For error propagation analysis of AVI procedures, which relates the size of errors $\|Q_{k+1} - T^* Q_k\|$ to the performance of the outcome policy compared to the optimal policy, refer to [26], [27]. For statistical analysis of some variants of AVI, refer to [21], [16], [15].

In this work we propose an AVI algorithm that uses a deep neural network [17], particularly a ConvNet, as $\mathcal{F}^{|\mathcal{A}|}$. As already mentioned, one reason for this choice is that ConvNets have been successful in extracting features from image-like inputs, especially for supervised computer vision problems [28]. Since the state of a PDE is a scalar field, it suggests that the choice of ConvNet might be suitable. Another reason is that similar architectures have been successfully applied to solving problems in computer games with image inputs [4].

Algorithm 1 describes the DFQI algorithm. The function space $\mathcal{F}^{|\mathcal{A}|}$ is represented by a deep neural network, and in particular a deep ConvNet for the PDE control problem. The algorithm works as follows. At iteration k, we are given a dataset $\mathcal{D}_n^{(k)} = \{(X_i, A_i, R_i, X'_i)\}_{i=1}^n$ with $X_i \sim \nu_{\mathcal{X}}$, a sampling distribution over the state space $\mathcal{X}$, the action $A_i \sim \pi_b(\cdot|X_i)$, a behaviour policy, the reward $R_i \sim \mathcal{R}(\cdot|X_i, A_i)$, and the next state $X'_i \sim \mathcal{P}(\cdot|X_i, A_i)$. This dataset might be generated a priori or it might be generated at each iteration (as shown in the algorithm).

Algorithm 1 Deep Fitted Q-Iteration ($\mathcal{F}^{|\mathcal{A}|}$,$K$,$N_{\text{epoch}}$, b)

```
// $\mathcal{F}^{|\mathcal{A}|}$: Deep neural network representing the action-value function space
// $K$: Number of iterations of DFQI
// $N_{\text{epoch}}$: Epochs per each iteration
// $b$: mini-batch size
Initialize the neural network $\hat{Q}_0 \in \mathcal{F}^{|\mathcal{A}|}$
for $k = 0$ to $K - 1$ do
    Generate samples $\mathcal{D}_n^{(k)} = \{(X_i, A_i, R_i, X'_i)\}_{i=1}^n$
    $Y_i = R_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}_k(X'_i, a')$ for $i = 1, \dots, n$
    Define regression dataset $\mathcal{D}'_n = \{((X_i, A_i), Y_i)\}_{i=1}^n$
    $\hat{Q}_{k+1} \leftarrow \hat{Q}_k$
    for $N_{\text{epoch}}$ times do
        Construct a random batch of $\mathcal{D}_b = \{((X_j, A_j), Y_j)\}_{j=1}^b$ from $\mathcal{D}'_n$
        Define $L_b = \frac{1}{b} \sum_{(X_j, A_j, Y_j) \in \mathcal{D}_b} \left|\hat{Q}_{k+1}(X_j, A_j) - Y_j\right|^2$.
        Update $\hat{Q}_{k+1}$ in the direction of gradient of $L_b$
    end for
end for
return $\hat{Q}_K$ and $\pi_K(\cdot) = \hat{\pi}(\cdot; \hat{Q}_K)$
```

Each iteration of DFQI performs a regression estimation (though only approximately). We define the regression targets $Y_i = R_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}_k(X'_i, a')$ for $i = 1, \dots, n$. The regression targets are defined based on the current estimate of the action-value function $\hat{Q}_k$. So we obtain a dataset $\mathcal{D}'_n = \{((X_i, A_i), Y_i)\}_{i=1}^n$.

In most Fitted Q-Iteration algorithms, such as RFQI [16], we solve the regression problem exactly. For example, in the RFQI algorithm we solve

$$\hat{Q}_{k+1} \leftarrow \operatorname*{argmin}_{Q \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n |Q(X_i, A_i) - Y_i|^2 + \lambda_{Q,n} \|Q\|_{\mathcal{H}}^2,$$

with $\mathcal{H}(= \mathcal{F}^{|\mathcal{A}|})$ being an RKHS and $\lambda_{Q,n} > 0$ being the regularization coefficient.

Solving such an optimization problem up to the machine precision with a DNN is impractical and not even necessary. It is impractical as 1) convergence of a DNN is only guaranteed to a local minimum, and 2) the convergence rate is slower than solving a system of linear equations, which has to be solved in the RKHS-based formulation. But it is also not necessary because having an optimization error close to zero is not required for the good generalization error of a learning algorithm [29].

Instead of solving the regression problem exactly at each iteration, DFQI only partially minimizes the loss function. At each iteration of DFQI, it follows the direction of gradient of the regression loss only for a relatively small number of iterations N_{epoch}. Moreover, we follow the common practice

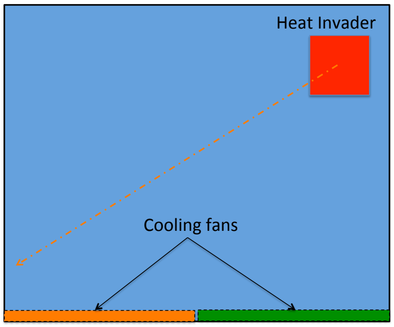

Fig. 1. Heat Invader Domain. A heat source starts from a random initial position and moves towards the bottom of the room. The agent can turn two cooling fans ON or OFF.

in training DNN of using mini-batches of size b to estimate the gradient, instead of using the whole dataset $\mathcal{D}'_n$. To be more concrete, for N_{epoch} times we randomly choose a mini-batch (i.e., a subset) of size b from $\mathcal{D}'_n$. Afterwards, we define the empirical loss function L_b, a squared error one, only based on these selected tuples, and compute the gradient of the empirical loss function L_b w.r.t. the DNN's parameters. We update the network based on the gradient. The exact way to use the gradient to update the network can vary. One may choose different weight adaptation rules to update the network: vanilla stochastic gradient descent, Adam [30], RMSprop, etc.

Note that in the inner loop, we optimize $\hat{Q}_{k+1}$, but we do not change the value of Y_is. This makes it different from the Q-learning algorithm. This is the same as using two separate networks, as is done by [4].

This computation is performed for K iterations to obtain the estimate $\hat{Q}_K(x,a)$ of the optimal action-value function Q^*. The obtained policy is the greedy policy of $\hat{Q}_K$, i.e., $\hat{\pi}(x;\hat{Q}_K) = \text{argmax}_{a\in\mathcal{A}}\,\hat{Q}_K(x,a)$.

DFQI is similar to DQN [4] and Neural Fitted Q-Iteration (NFQ) [20]. All of them are essentially AVI (or Fitted Q-Iteration) algorithms that use a neural network (DNN for DFQI and DQN, a shallow NN for the implementation of NFQ) as the underlying function space $\mathcal{F}^{|\mathcal{A}|}$. A common feature of DFQI and DQN is that at each iteration of AVI, neither of them attempts to converge to the minimum of the empirical loss function; instead, they only partially move towards the minimum before applying the Bellman operator to the current estimate. The DFQI algorithm is presented as a batch algorithm, so its connection to the other AVI algorithms might be more clear than DQN, which is presented as an online algorithm. Note that the authors of [22] present an algorithm with the same name Deep Fitted Q-iteration (DFQ). Their algorithm is, however, quite different from DFQI or DQN. DFQ uses an auto-encoder to find a low-dimensional representation of the state, which is then used by the tree-based Fitted Q-Iteration algorithm of [19].

IV. EXPERIMENTS

We use *Heat Invader*, a time-varying convection-diffusion problem, to conduct our experiments (cf. Figure 1). In the heat invader problem, introduced by [1], a heat source enters a room at a randomly chosen location at the top half of the room. This heat source travels the room with a constant speed (downward and leftward) until it leaves the room. As a result of this heat disturbance, the temperature field of the room, which is governed by a 2D convection-diffusion PDE, changes. The heat invader might be thought of as a heat disturbance entering the room after briefly opening the window in a hot day. To fight off the heat invader, two cooling fans on the floor can be turned ON or OFF. The goal is to choose when each of them should be ON or OFF in order to make the room "comfortable" while minimizing the energy used by the cooling system. Turning each of the cooling fans induces an airflow field in the room, depending on which fan is ON. The airflow field in the work of [1] was uniformly upward in the left and/or right section of the room. Here we also perform experiments with more complex circular airflow fields, which are more physically realistic, as shall be described soon.[4]

The state of the system, x, at time t depends on the temperature field $T(\cdot,t)$. T is a scalar field, so in theory it is an infinite-dimensional object. In our simulations, we use a finite volume solver (FiPy by [31]) with a 50×50 grid to represent the square room, so T is represented by a 2500 dimensional real-valued vector.[5] To fix the length scale, assume that the width and the height of the room is in fact 50 units of length. The heat invader defines the source S in (1). At the location of the heat invader, a square whose width is 1/5th of the room's (so it is 10×10), we set the value of $S(z,t) = 1$, and we set it to zero elsewhere. The heat invader moves with a downward/leftward velocity of $-(4e_x + 2e_y)$ per time step ($e_{x/y}$ are the unit vectors in the x or y direction), so this defines a time-varying source. We choose the Péclet number $\text{Pe} = 500$, as it is close to typical values for a room [32], [33].

The action set $\mathcal{A} = \{\,(T_b^a, v^a) : a = 1,\dots,|\mathcal{A}|\,\}$ in our experiments has four elements corresponding to a being one of OFF/OFF, ON/OFF, OFF/ON, and ON/ON settings of the left/right cooling fans. When a side of the room has an ON action, the Dirichlet boundary condition of the temperature, T_b^a, at the corresponding side of the floor is set to -0.5; and it is set to 0 when the action is OFF. Moreover, the fan induces an airflow. We consider two types of airflow, namely "uniform" and "circular". In the uniform airflow, we have a constant upward airflow on the same side of the room as the fan is. That is, $v^a(x,y) = 0e_x + 5e_y$ for all (x,y) in the left or right side of the room, and zero on the opposite side. When both of them are ON, the whole floor has the temperature of -0.5 and the airflow is $v^a(x,y) = 0e_x + 5e_y$ for all (x,y) in the room.

The circular airflow is defined as follows. When only the right side fan is ON, the airflow field is $v^{\text{OFF/ON}}(x,y) =$

[4] The name of Heat Invader is inspired from the Space Invader game on Atari 2600. Heat Invader is the PDE version of that game.

[5] We performed some simple grid studies. A finer 100×100 discretization has essentially the same behaviour, so to save the computation time we performed our experiments only on 50×50 grid.

$-5\cos\left(\frac{2\pi x}{50}\right)\sin\left(\frac{2\pi y}{50}\right)e_x + 5\sin\left(\frac{2\pi x}{50}\right)\cos\left(\frac{2\pi y}{50}\right)e_y$. When only the left side fan is ON, the airflow has the opposite flow direction, i.e., $v^{\text{ON/OFF}} = -v^{\text{OFF/ON}}(x,y)$. When both of them are ON, the airflow is a constant upward flow, i.e., $v^{\text{ON/ON}} = 5e_y$, which is the same as $v^{\text{ON/ON}}$ of the uniform airflow. All airflow fields for both uniform and circular cases are divergence-free.

By considering circular airflows, we have made the PDE more realistic compared to the uniform flow of [1]. We should mention that a real fan's induced airflow can be more complicated, but for simplicity of our simulations and to avoid solving the Navier-Stokes equations we focus on the current model. Also note that since we effectively control both the temperature on the boundary and the velocity field, the current formulation is indeed a nonlinear (bilinear) control problem due to $\nabla \cdot (vT)$ term in the convection-diffusion equation (1).

The reward function (the negative of cost) encodes our belief about what a comfortable room setting should be, in addition to the cost of operating the cooling fans. If the temperature field at any given point in the room is within a specific threshold of the desired temperature, there would not be any cost associated. Otherwise, that point is considered undesirable, and it contributes to the cost. The part of the reward due to the uncomfortable temperature is the average value of this criteria over the whole room. The part related to the operation cost is linear in the number of cooling fans that are ON. More precisely,

$$r(T,a) = -\int_{\mathcal{Z}} \mathbb{I}\{|T(z) - T^*(z)| > \Delta T_{\text{threshold}}\}\mathrm{d}\mu(z) - \begin{cases} 0 & a = \text{(OFF,OFF)} \\ c_{\text{actuator}} & a \in \{\text{(ON,OFF)}, \text{(OFF, ON)}\} \\ 2c_{\text{actuator}} & a = \text{(ON,ON)} \end{cases} \quad (3)$$

We set the desired temperature profile $T^*(z) = 0$, the threshold $\Delta T_{\text{threshold}} = 0.5$, and $c_{\text{actuator}} = 0.025$. Here μ is a uniform probability measure, i.e., the volume of $\mathcal{Z}$ according to μ is 1. Notice that this reward function is not linear or quadratic in T. The discount factor is set to $\gamma = 0.9$. In all our experiments, the left, right, and top walls have the Neumann boundary condition. The floor has a Dirichlet boundary condition, defined based on the selected action.

We compare the performance of DFQI, as described in the previous section, and RFQI, as described by [1]. Both of these algorithms have access to the whole scalar field T. This is a close approximation of the true state of the system. From now on, we call T the state of the system. We also compare with some manually-designed policies, namely "Simple controller" and "Smart controller" [1]. Briefly speaking, Simple controller turns the fan ON on the side of the room that is warmer, but it does not consider the operation cost. The Smart controller considers the operation cost too. Both of them are myopic policies.

We generate the batch of data $\mathcal{D}_n = \{(X_i, A_i, R_i, X_i')\}_{i=1}^n$, to be used by the DFQI and RFQI algorithms, as follows: We randomly generate the initial position of the heat invader at time $t = 0$ close to the top of the room. The initial temperature $T(\cdot;0)$ is set to zero at time $t = 0$. We let the temperature diffuse according to the convection-diffusion equation for one time step (since there is no convection field, it is only the diffusion term that acts at this time step). This results in the temperature field $T(\cdot;1)$. We use the temperature scalar field $T(\cdot;t)$ as the state X_t that is fed to the algorithms (for $t \geq 1$). We choose action A_1 uniformly random from the set of all possible actions $\mathcal{A} = \{\text{(OFF/OFF)}, \text{(ON/OFF)}, \text{(OFF/ON)}, \text{(ON/ON)}\}$ to obtain the new temperature field at time $t = 2$. Depending on the airflow type (uniform or circular), the effect would be different. The reward at time $t = 1$ is $R_1 = r(T(\cdot;2), A_1)$ with r being defined in (3). Meanwhile the heat invader moves with a constant velocity. After 40 steps, which defines one episode, we reset the state of the environment to its initial value of zero and then pick a new random location for the heat invader.[6] This procedure is repeated until we obtain n data points.

In the first set of experiments, we generate a dataset with $n = 20000$. After obtaining the data, we only use a subset of it (i.e., $n \in \{1240, 2000, 3160, 5000, 7960, 12600, 20000\}$) to obtain the policy using the DFQI and RFQI algorithms. We run each of those algorithms for $K = 50$ iterations. For each choice of n, we evaluate the greedy policy w.r.t. $\hat{Q}_K$ from 50 random initial states. We use two evaluation metrics. The first is the average reward per episode (i.e., $\sum_{t=1}^{T_f} R_t$) and the second is the return (i.e., $\sum_{t=1}^{T_f} \gamma^{t-1} R_t$) with T_f being the episode's length, which is 40 in our experiments.[7] We then increase the size of the dataset (e.g., from 1240 to 2000)—reusing the data in the previous set. We initialize the value function to the value obtained after processing the previous dataset. Again, we perform $K = 50$ iterations of RFQI/DFQI. We continue this procedure until we evaluate the policy obtained with $n = 20000$. We repeat this whole procedure 20 times, with an independent dataset in each run, for each of the uniform and circular airflow fields. For non-RL-based algorithms, the empirical average is computed based on 100 independent runs.

We use a ConvNet architecture similar to the one used by [4]. The network has three consecutive 2D convolutional layers followed by a fully connected hidden layer and 4 output layers. The first convolutional layer has 32 filters of 8×8 convolutions with the stride of 4. The second convolutional layer has 64 filters of 4×4 convolutions with the stride of 2. The third convolutional layer has 64 filters of 3×3 convolutions with the stride of 1. The fourth layer is a fully connected one with the output dimension of 500. All these layers use Rectified Linear units (ReLu). The final layer has 4 outputs (corresponding to $\hat{Q}(\cdot, a)$ for each choice of action $a \in \mathcal{A}$) with linear units. We use RMSprop algorithm

[6] The heat invader takes at most 25 steps to start from the top of the room and then leave it. The value of $t = 40$ is chosen accordingly, so that the state of the system gets approximately settled if no action is taken.

[7] Even though the discounted MDP formulation is for infinite-horizon problems, for evaluation we truncate the episode at the horizon T_f, mainly to save the computation cost.

for weight update with the learning rate of 0.00025, the momentum term of 0.95, and the normalizer constant of 0.01. We use the mini-batch size $b = 64$ and the number of epochs per DFQI iteration $N_{\text{epoch}} = 20$. To implement the ConvNet in DFQI, we use Keras package [34] on top of Theano [35] in Python. To implement the RFQI algorithm, we use scikit-learn [36] in Python.

Figure 2 compares the performance of DFQI and RFQI as a function of the number of samples n used for training, when the RL algorithms are trained (and evaluated) on the uniform airflow field. Figure 3 presents the same comparison when the airflow field (for both training and evaluation) is circular. The Simple and Smart controllers do not depend on n. The graphs depict the empirical mean of the average reward per episode (left) and return (right). The error bars around the curve show one standard error around the mean.

The graphs clearly indicate the progress in the quality of both DFQI and RFQI policies as n increases. We see that both DFQI and RFQI surpass the performance of the Simple and Smart controllers.

In the uniform airflow case, they outperform the Simple controller from $n = 1240$ samples, and they outperform the Smart controller after $n = 3160$ (DFQI) and $n = 7960$ (RFQI) samples (measured according to the average return; similar conclusion holds for the average reward). In the case of circular airflows, the Simple controller performs poorly (average reward is -0.393 ± 0.003 and the average return is -2.60 ± 0.025, in which we are presenting the standard error), so we do not show it in the graph. We observe that both DFQI and RFQI outperform the Smart controller rather quickly.

What is more interesting is that DFQI also outperforms RFQI for both uniform and circular airflows. In the uniform case, the superiority is mostly in the smaller range of samples, but eventually RFQI catches up. The difference is more striking in the circular airflow, in which even after $n = 20000$ the performance of RFQI is inferior. We hypothesize that this superiority of DFQI over RFQI is because the ConvNet in DFQI could learn PDE-specific function space $\mathcal{F}^{|\mathcal{A}|}$, while the RKHS used in RFQI is not being adapted to the data. We also see that the uniform airflow provides the possibility of having higher performing controls compared to the circular one, as reflected in the average reward and return in all controllers.

Finally we briefly investigate the issue of transfer learning. The question that we ask is whether learning a policy for one PDE (e.g., the one with uniform airflow) can be used to accelerate the learning of a good policy for different, but related, PDE (e.g., the one with circular airflow). We study three different scenarios. The first is that we learn a policy using DFQI only on the target domain (circular airflow) from scratch. There is no transfer learning in this case. We then consider a scenario in which we first find a policy on a source domain (uniform airflow) using DFQI, but then use it as an initial DNN to train for the target domain. We consider two sub-cases: When DFQI uses $n = 5000$ data points from the source domain and when it uses $n = 10000$. Obviously the latter one provides a better policy for the source domain, but we wonder whether it also provides a better initialization for the target domain. We perform this comparison for 20 independent runs. The results are shown in Figure 4. We see that transfer learning indeed helps a lot compared to the case without any transfer. Moreover, when we trained based on 10000 data points in the source domain, the performance is better, particularly when we do not have much data points in the target domain.

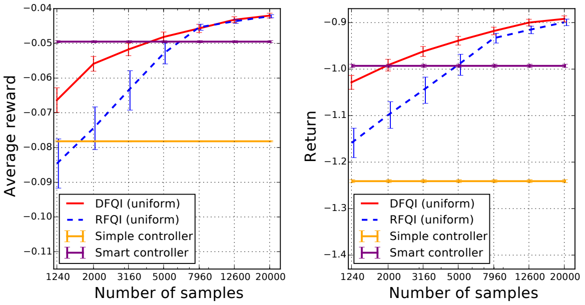

Fig. 2. (Uniform Airflow) The average reward per episode (left) and the return (right) of several policies as a function of the number of training points. The results are for uniform airflow. The red curve shows the DFQI-based policy and the blue one shows the RFQI-based policy. The performances of two manually-designed controllers are shown too. The error bars depict one standard error.

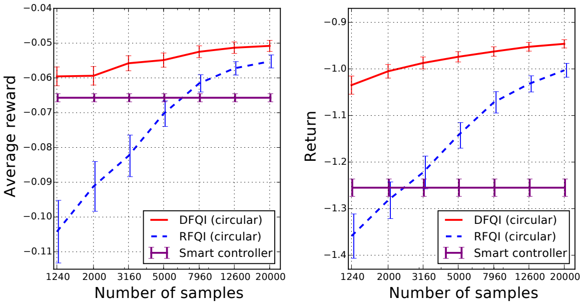

Fig. 3. (Circular Airflow) The average reward per episode (left) and the return (right) of several policies as a function of the number of training points. The results are for circular airflow. The red curve shows the DFQI-based policy and the blue one shows the RFQI-based policy. The performances of a manually-designed controller is shown too. The error bars depict one standard error.

V. CONCLUSIONS AND FUTURE WORK

In this work, we followed [1] in formulating a class of PDE control problems as reinforcement learning problems. We suggested to consider the state of a PDE as an image and to use deep convolutional neural networks as the value function estimator. We developed a Deep Fitted Q-Iteration algorithm and empirically showed its good performance.

In our simulations we considered that the airflow field v is given. Performing a more realistic simulation that finds v itself by solving the Navier-Stokes equations would be

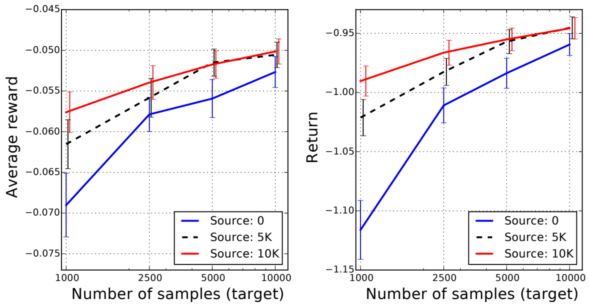

Fig. 4. The average reward per episode (left) and the return (right) of several source-to-target transfer learning scenarios. The horizontal axis is the number of training points in the target domain. The blue curve shows the scenario when no training on the source domain (with uniform airflow) has been done. The black dashed curve is when the policy is first trained on the source domain with 5K of data points, and then trained on the target domain (with circular airflow). The red one is for 10K of data points in the source domain. The error bars depict one standard error.

interesting. Eventually our aim is to perform a real physical system experiment. It is also interesting to apply the proposed method to other PDE control problems.

Although a main advantage of the proposed method is that it is completely data-driven, so the knowledge of the PDE is not required, it is curious to see whether we might leverage any prior knowledge about the PDE, however approximate, to accelerate the learning process. Another research direction is developing a method that is not restricted to finite action spaces, and can work with high-dimensional action spaces too.

ACKNOWLEDGMENT

We would like to thank Piyush Grover for discussions at the earlier stages of this work. We also acknowledge helpful comments by the anonymous reviewers.

REFERENCES

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

[2] Cs. Szepesvári, *Algorithms for Reinforcement Learning*. Morgan Claypool Publishers, 2010.

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

[4] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 02 2015.

[5] D. Silver *et al*, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 01 2016.

[6] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” *Journal of Machine Learning Research (JMLR)*, vol. 17, no. 39, pp. 1–40, 2016.

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

[8] J. Borggaard, J. A. Burns, A. Surana, and L. Zietsman, “Control, estimation and optimization of energy efficient buildings,” in *American Control Conference (ACC)*, 2009, pp. 837–841.

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

[10] J. A. Burns and W. Hu, “Approximation methods for boundary control of the Boussinesq equations,” in *IEEE Conference on Decision and Control (CDC)*, 2013, pp. 454–459.

[11] S. L. Brunton and B. R. Noack, “Closed-loop turbulence control: Progress and challenges,” *Applied Mechanics Reviews*, vol. 67, no. 5, 2015.

[12] D. Foures, C.-c. Caulfield, and P. J. Schmid, “Optimal mixing in two-dimensional plane poiseuille flow at finite Péclet number,” *Journal of Fluid Mechanics*, vol. 748, pp. 241–277, 2014.

[13] T. Duriez, S. L. Brunton, and B. R. Noack, *Machine Learning Control–Taming Nonlinear Dynamics and Turbulence*, ser. Fluid mechanics and its applications. Springer, 2016, vol. 116.

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

[15] A.-m. Farahmand, “Regularization in reinforcement learning,” Ph.D. dissertation, University of Alberta, 2011.

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

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[18] M. Bojarski *et al*, “End to end learning for self-driving cars,” *CoRR*, vol. abs/1604.07316, 2016.

[19] D. Ernst, P. Geurts, and L. Wehenkel, “Tree-based batch mode reinforcement learning,” *Journal of Machine Learning Research (JMLR)*, vol. 6, pp. 503–556, 2005.

[20] M. Riedmiller, “Neural fitted Q iteration – first experiences with a data efficient neural reinforcement learning method,” in *16th European Conference on Machine Learning*, 2005, pp. 317–328.

[21] R. Munos and Cs. Szepesvári, “Finite-time bounds for fitted value iteration,” *Journal of Machine Learning Research (JMLR)*, vol. 9, pp. 815–857, 2008.

[22] S. Lange and M. Riedmiller, “Deep auto-encoder neural networks in reinforcement learning,” in *International Joint Conference on Neural Networks (IJCNN)*, 2010.

[23] A.-m. Farahmand and D. Precup, “Value pursuit iteration,” in *Advances in Neural Information Processing Systems (NIPS - 25)*, 2012, pp. 1349–1357.

[24] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

[25] L. Wasserman, *All of Nonparametric Statistics (Springer Texts in Statistics)*. Springer, 2007.

[26] A.-m. Farahmand, R. Munos, and Cs. Szepesvári, “Error propagation for approximate policy and value iteration,” in *Advances in Neural Information Processing Systems (NIPS - 23)*, 2010, pp. 568–576.

[27] R. Munos, “Performance bounds in L_p norm for approximate value iteration,” *SIAM Journal on Control and Optimization*, pp. 541–561, 2007.

[28] A. Krizhevsky, I. Sutskever, and G. E. Sutskever, “ImageNet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems (NIPS - 25)*, 2012, pp. 1097–1105.

[29] L. Bottou and O. Bousquet, “The tradeoffs of large scale learning,” in *Advances in Neural Information Processing Systems (NIPS - 20)*. MIT Press, 2008, pp. 161–168.

[30] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” in *International Conference on Learning Representations (ICLR)*, 2015.

[31] J. E. Guyer, D. Wheeler, and J. A. Warren, “FiPy: Partial differential equations with Python,” *Computing in Science and Engineering*, vol. 11, no. 3, pp. 6–15, 2009. [Online]. Available: http://www.ctcms.nist.gov/fipy

[32] P. F. Linden, “The fluid mechanics of natural ventillation,” *Annual Review of Fluid Mechanics*, vol. 31, pp. 201–238, 1999.

[33] S. Nabi, “Buoyancy-driven exchange flow with applications to architectural fluid mechanics,” Ph.D. dissertation, University of Alberta, 2015.

[34] F. Chollet, “Keras,” https://github.com/fchollet/keras, 2015.

[35] Theano Development Team, “Theano: A Python framework for fast computation of mathematical expressions,” vol. abs/1605.02688, May 2016.

[36] F. Pedregosa *et al*, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research (JMLR)*, vol. 12, pp. 2825–2830, 2011.