Applied Energy 278 (2020) 115660

Contents lists available at ScienceDirect

Applied Energy

journal homepage: www.elsevier.com/locate/apenergy

Optimal carbon storage reservoir management through deep reinforcement learning

Alexander Y. Sun

Bureau of Economic Geology, Jackson School of Geosciences, The University of Texas at Austin, Austin, TX, United States of America

ARTICLE INFO

Keywords:
Reinforcement learning
Multistage decision-making
Deep autoregressive model
Deep Q network
Surrogate modeling
Markov decision process
Geological carbon sequestration

ABSTRACT

Model-based optimization plays a central role in energy system design and management. The complexity and high-dimensionality of many process-level models, especially those used for geosystem energy exploration and utilization, often lead to formidable computational costs when the dimension of decision space is also large. This work adopts elements of recently advanced deep learning techniques to solve a sequential decision-making problem in applied geosystem management. Specifically, a deep reinforcement learning framework was formed for optimal multiperiod planning, in which a deep Q-learning network (DQN) agent was trained to maximize rewards by learning from high-dimensional inputs and from exploitation of its past experiences. To expedite computation, deep multitask learning was used to approximate high-dimensional, multistate transition functions. Both DQN and deep multitask learning are pattern based. As a demonstration, the framework was applied to optimal carbon sequestration reservoir planning using two different types of management strategies: monitoring only and brine extraction. Both strategies are designed to mitigate potential risks due to pressure buildup. Results show that the DQN agent can identify the optimal policies to maximize the reward for given risk and cost constraints. Experiments also show that knowledge the agent gained from interacting with one environment is largely preserved when deploying the same agent in other similar environments.

1. Introduction

Model-based optimization plays a central role in many areas of energy system design and management, such as building performance optimization [1,2]; renewable energy system design [3]; plant and industrial process optimization [4,5]; hybrid electric vehicle energy management [6,7]; and carbon capture [8], storage [9], and utilization systems [10,11]. In general, computer models are used to examine the decision and parameter space under a variety of scenarios to achieve optimal energy system performance while minimizing total system costs and conforming with energy and environmental policy [12]. So far, a wide array of optimization methods have been applied, including integer and mixed-integer linear programming [5,13–17], evolutionary algorithms [1,18–20], and heuristic search methods (e.g., tabu search, simulated annealing) [21,22]. The efficacy and applicability of these optimization methods largely depend on (a) characteristics of the design task, for example, whether objective functions and constraints are linear or nonlinear, and whether the decision space is stochastic; (b) dimensionality of the decision space; and (c) complexity of the underlying computational model.

With the wide adoption of the Internet-of-Things technology in recent years [23,24], requirements on the resolution (both spatial and temporal) and granularity of system representation have also increased for many energy system simulation models. Oftentimes the physics behind many of these systems is complex, involving multiscale spatial and temporal processes. As a result, the computational models can be high-dimensional and coupled, making the optimization process extremely time consuming if not infeasible. In the past, two broad strategies have been taken to mitigate the computational burden, namely, simplifying the optimization process and reducing/replacing the physics-based models themselves. The first strategy aims to reduce the number of actual model runs by storing and exploiting the results of model runs, such that solutions of new trial points can be quickly estimated [25,26] or new search directions can be approximated [27–29]. The second strategy, commonly known as surrogate modeling, aims to develop a proxy of an otherwise computationally expensive model. A useful surrogate model should not only provide the benefits of low-cost function evaluations while maintaining numerical fidelity requirements, but also help to make quantitative assessments of alternative scenarios and their tradeoffs, and to perform sensitivity analyses and uncertainty quantification.

Geosystems, which are intricately linked to fossil energy exploration, climate change mitigation, and various renewable energy

E-mail address: alex.sun@beg.utexas.edu.

https://doi.org/10.1016/j.apenergy.2020.115660
Received 25 March 2020; Received in revised form 7 July 2020; Accepted 3 August 2020
Available online 10 August 2020

Table 1
A survey of surrogate modeling works in subsurface applications.

Technique	Applications	References
Multivariate linear regression	Monitoring network design, enhanced oil recovery, groundwater modeling	Chen et al. [34], Dai et al. [35], Mo et al. [36]
Sparse grid, stochastic collocation method	Risk assessment, Monte Carlo simulation, groundwater modeling, well schedule optimization	Oladyshkin and Nowak [37], Sun et al. [38], Laloy et al. [39], Liao et al. [40], Zhang et al. [41], Babaei and Pan [42]
Gaussian regression (kriging)	Contaminant source inversion, data assimilation	Zhang et al. [43], Zheng et al. [44]
Artificial neural networks	Well placement, reservoir production prediction	Golzari et al. [45], Jang et al. [46], Jeong et al. [47]

generation applications, represent a type of such high-dimensional complex systems. Like other aforementioned energy applications, model-based optimization is critical to exploration and utilization planning of geosystems, and there is a strong need for surrogate modeling because of the high computational cost of running large-scale system models. Unlike many other applications, modern subsurface models are characterized by high-dimensional distributed input variables that are also rife with uncertainty because of limited data availability, especially during the planning stage. The development of surrogate models to assist high-dimensional geosystem optimization under uncertainty is still a challenging task.

Table 1 provides a brief survey of existing surrogate modeling methods as related to subsurface applications, with a primary focus on methods not requiring code modifications (i.e., noninvasive methods). More comprehensive reviews can be found in topical reviews [e.g., 30–32]. Many existing surrogate modeling methods listed in Table 1 are built for specific input parameter distributions and are generally only applicable when the dimensionality of parameter space is relatively low. Although parameter dimension reduction methods are available to cope with the so-called curse of dimensionality, the resulting parameter dimensions may still be too high for the purpose of surrogate modeling [33].

In recent years, the arrival of the deep learning (DL) era has sparked new interests in surrogate modeling. Unlike their counterparts in conventional machine learning, these DL algorithms are designed to extract and learn hierarchical representations of high-dimensional input data without requiring the user to go through the feature selection and dimension reduction steps. Importantly, most DL algorithms are data driven and invoke few assumptions on input distributions. In a typical deep surrogate modeling setting, training samples of input–output pairs, which are prepared by running the original physics-based computational model, are presented to a DL algorithm to teach it to learn the approximate mapping between input(s) and output(s). Once trained, the DL model may be used as a surrogate model to predict system states.

Several recent studies have already demonstrated the use of DL-based surrogate modeling for geosystems. For example, Zhu et al. [48] developed a fully convolutional neural network (CNN) model to learn the forward mapping between a high-dimensional input field (permeability) and an output field (pressure), leading to the so-called end-to-end or image-to-image regression model. Sun [49] proposed a state-parameter identification model to learn not only the forward mapping, but also the inverse mapping between high-dimensional model input (hydraulic conductivity) and output (hydraulic head) by using an approach based on the cyclic generative adversarial network (CycleGAN) [50]. Mo et al. [51] extended the work of Zhu et al. [48] to transient multiphase flows by using prediction time as an additional input label during training. Similarly, Zhong et al. [52] presented a conditional convolutional GAN model for tracking the evolution of CO_2 plumes in carbon storage reservoirs. Zhong et al. [53] used CycleGAN to learn the forward and inverse mappings between CO_2 saturation (plume) and seismic data by using petrophysical models as constraints. It is often necessary to map multiple state variables in multiphase flow problems. For that purpose, Mo et al. [51] concatenated pressure and saturation fields into a single tensor variable, and Tang et al. [54] trained two recurrent residual U-Net (R-U-Net) surrogate models with the same architecture but different loss functions to approximate pressure and saturation fields.

Most of the aforementioned studies on dynamic systems focus on predicting the system states at an arbitrary time t for given inputs. Many other applications, for example, sequential optimization or data assimilation, are concerned with state transitions in a dynamic system. Recently, Mo et al. [33] developed a deep autoregressive model for approximating state transition functions. Given the system state at time t, the surrogate model can be trained to evolve the system state from t to $t + \Delta t$, thus providing a DL approach for learning and predicting state transitions in a high-dimensional dynamic system. They demonstrated the use of such a deep autoregressive model for tracking contaminant transport (i.e., concentration is the state variable) in groundwater aquifers. Geneva and Zabaras [55] presented a generic approach for developing deep autoregressive models that have stochastic initial states.

In this work, I adopted elements of the recently developed DL algorithms to tackle a model-based sequential optimization problem, which can be found in many applied energy applications such as microgrid energy demand management [56,57], energy–water nexus management [58,59], and reservoir production planning [60]. Specifically, the sequential optimization problem is formulated as a Markov Decision Process (MDP), which involves an agent interacting with a dynamic environment through a sequence of actions, observations, and rewards. The agent is not told what to do, but instead seeks to maximize the cumulative future reward by discovering an optimal policy through interactions. This type of problem falls under reinforcement learning, which is a general class of algorithms in machine learning that aims to help an agent learn how to behave in a dynamic environment, where the only feedback consists of a scalar reward [61,62].

The sequential behaviors/actions of the agent comprise a policy. One of the most popular methods for learning the optimal policy is Q-learning. The essence of Q-learning, which has been around for a long time [63], is to learn a nonlinear action–value function, or Q function, incrementally, based on rewards from different steps (see next section). But existing methods for nonlinear function approximation, especially those that are based on traditional artificial neural networks, are known to be highly unstable, making Q-learning hard to apply in practice [64]. A major breakthrough came recently when a group of researchers from DeepMind (now a Google company) introduced the original deep Q-learning (DQL) algorithm, which was shown to reach human-level performance on many Atari games [65,66]. The success of DQL is largely attributed to (a) the use of a target network sharing the same architecture with the deep Q-function network but with delayed update and (b) experience replay of past observations (see also Section 2). Together, the two mechanisms help to improve training stability. The DQL has quickly made its way beyond gaming to other areas, including microgrid energy trading and demand management [67,68], building energy management [69], electric vehicle energy management [6,69–71], and identification of sustainable management pathways in earth system models [72].

The main contributions of this work include (a) developing a DQL-based framework for optimal multiperiod planning involving high-dimensional, multistate geosystem models and (b) formulating a deep multitask learning (DeepMTL) approach to approximating multistate transition functions under variable forcing conditions, thus reducing total computational costs. As a case study, I apply the DQL framework to

multiperiod carbon storage planning, which is a geosystem-based measure for greenhouse gas emission reduction that has received significant renewed interest in the U.S. because of the recently passed carbon tax incentives, Section 45Q [73]. Globally, carbon sequestration constitutes an important tool, as the major energy producers and governments are moving toward net-zero carbon emission goals. The DQL framework itself is rather general and can be applied to many other applied energy multiperiod planning problems involving computationally expensive simulation models.

This paper is organized as follows: First, I introduce the DQL and DeepMTL methods in Section 2. Under Section 3, the problem setup and details of network training are provided. Section 4 provides results and discussions, followed by conclusions.

2. Methodology

2.1. Deep reinforcement learning

Simply speaking, reinforcement learning is about an agent (e.g., a reservoir operator) learning how to match system states to best actions over time in order to obtain the maximum reward [64]. A Markov Decision Process (MDP) provides a formal framework for modeling the agent's sequential interactions with an environment. The basic elements of an MDP are given by the tuple $(S, \mathcal{A}, T, r)$, in which S is a set of states or state space, $\mathcal{A}$ is set of actions or action space, $T : S \times \mathcal{A} \times S \to [0, 1]$ defines the state transition probability, and $r : S \times \mathcal{A} \to \mathbb{R}$ is a reward function that defines a scalar reward for given state and action. A policy is a mapping from state to action, $\pi : S \to \mathcal{A}$, that is subject to $\pi(s, a) \geq 0$ for $s \in S$ and $a \in \mathcal{A}$. The goal of reinforcement learning is to learn an optimal policy π that maximizes the expected return.

Starting from an initial state s_0, at step t the agent receives a state observation s_t and takes an action according policy $\pi(s_t, a_t)$. The system state evolves according to the transition probability, $T(s, a, s') = P(s_{t+1} = s' \mid s_t = s, a_t = a)$, and the agent receives a reward $r_t(s_t, a_t)$. Thus, execution of the MDP under policy π leads to a sequence, $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$. To identify the optimal policy, one approach is to learn a value function that expresses how good it is for an agent to be in a certain state. Toward such a goal, the state–action value function, or Q-function, is defined as the expected return from all future rewards starting from the state s and after taking action a under policy π

$$Q^{\pi}(s, a) = \mathbb{E}\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a\right), \tag{1}$$

in which $\gamma \in (0, 1]$ is a discount factor used to express the preference of the agent to future rewards. Eq. (1) may be written in the following recursive form known as the Bellman equation [64]

$$Q^{\pi}(s, a) = \mathbb{E}\left(r_t + \gamma \mathbb{E}\left(Q^{\pi}(s_{t+1}, a_{t+1})\right) \mid s_t = s, a_t = a\right). \tag{2}$$

The solution to MDP now becomes finding the optimal value of the state–action function, denoted here by $Q^*(s, a)$. The simplest action rule is one that picks an action (out of all possible actions defined in the action space) to maximize the reward in each state, without worrying about future states [62],

$$Q^*(s, a) = \max_a \sum_{s'} T(s, a, s')\left(r(s, a) + \gamma \max_{a'} Q^*(s', a')\right), \tag{3}$$

$$\pi^* = \arg\max_a Q^*(s, a), \tag{4}$$

in which the expected return is calculated using the transition probabilities as defined before. The resulting policy π^* in Eq. (4) is known as the greedy policy.

Knowledge on transition probability and reward function is not always available. The Q-learning algorithm, originally proposed by Watkins and Dayan [63], provides an iterative way to estimate the Q-function without requiring such knowledge (i.e., a model-free method)

$$Q_{k+1}(s_t, a_t) = Q_k(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q_k(s_{t+1}, a) - Q_k(s_t, a_t)\right), \tag{5}$$

in which $\alpha \in (0, 1]$ is a learning rate that determines the step size of update and k is an index on periods or steps in a planning horizon. In a typical Q-learning implementation, the value of the Q-function is updated according to Eq. (5) for all steps until a terminal state is reached. For each step, a uniform random number is drawn from $\mathcal{U}(0, 1)$. If the random number is less than ϵ, an action is randomly sampled from $\mathcal{A}$, otherwise the best action $\arg\max_a Q(s, a)$ is taken, in which ϵ is a user-specified parameter. This so-called ϵ-greedy policy tries to balance between exploitation (sampling an existing pool of good actions) and exploration (trying new actions) in the policy search. A common practice is to start with $\epsilon = 1$ and then gradually lower the value of ϵ such that the policy search transitions from exploration-dominated searches in the beginning to exploitation-dominated searches in later stages. The whole planning process is replayed many times, with each time constituting an episode. Theoretically, it can be shown that Q-learning will converge to the optimal solution when the number of episodes goes to infinity [63].

To speed up Q-learning, it is tempting to train and use a function approximator to estimate the action–state function $Q^{\pi}(s, a)$. Historically, the training of such function approximators had been difficult and unstable. A major breakthrough came when DQL was introduced [65,66]. In DQL, two neural networks are trained in parallel. The deep Q network (DQN) is used as a function approximator $\tilde{Q}(s, a; \theta)$ to approximate the value of the Q function, in which θ denotes DQN parameters. The second network is a target network used to calculate future rewards $Y(\theta^-) = r + \gamma \max_{a'} \tilde{Q}(s', a'; \theta^-)$, with θ^- being the target network parameters. Both networks are initialized with the same weights. During training, the following mean square error loss function is minimized by updating θ iteratively using a gradient-descent solver,

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left(Y(\theta^-) - \tilde{Q}(s, a; \theta)\right)^2\right], \tag{6}$$

in which $\mathcal{L}$ is the loss function. To improve training stability, the weights of the target network are only updated periodically during iterations. One approach is copying the values of θ to θ^- for every other L iterations defined by the user. The other approach, known as soft update, is using θ^- to slowly track the Q-function network [74]

$$\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^-, \tag{7}$$

in which $\tau \ll 1$ is a user-specified parameter. In this latter approach, the target network parameters are constrained to change slowly, which greatly improves the training stability. For exploitation, DQN uses an experience replay mechanism that uniformly samples batches of transition tuples (s, a, s', r) from a pool of stored state transitions. The use of experience replay or memory buffer removes correlation among samples and reduces the variance of updates, thus also improving training stability significantly [66].

2.2. Autoregressive model for state prediction

DQL still requires evaluation of a large number of system transitions, which can be computationally expensive for high-dimensional physics-based models, as mentioned under Introduction. The main motivation in this work is thus to develop a DL-based surrogate model to expedite the policy search process. My starting point is a general forward model written in the form [32]

$$\mathcal{F}(\mathbf{s}, \mathbf{q}, \boldsymbol{\Lambda}, \boldsymbol{\xi}) = \mathbf{0}, \tag{8}$$

in which $\mathcal{F}(\cdot)$ is a model operator; $\mathbf{s}$, written in boldface here, is used to represent all state variables that may be a function of both space and time; Λ denotes a set of model parameters; $\mathbf{q}$ represents forcing terms; and $\boldsymbol{\xi}$ represents boundary and initial conditions. For state–action predictions, the system in Eq. (8) may be cast in a discrete-time, state-space form as

$$\mathbf{s}_{t+1} = f(\mathbf{s}_t, \mathbf{q}_t, \Lambda, \boldsymbol{\xi}_t) + \boldsymbol{\eta}_t, \tag{9}$$

in which the function f describes system dynamics, and $\boldsymbol{\eta}_t$ is system noise. Without loss of generality, I considered the system of partial differential equations (PDEs) governing flow and transport of CO_2 plumes in saline aquifers, which is used as a demonstration case in this study,

$$\mathbf{q}_\alpha = -\frac{k_{r,\alpha}\mathbf{k}}{\mu_\alpha}(\nabla P_\alpha - \rho_\alpha g \nabla z), \qquad \alpha = w, g, \tag{10}$$

$$\frac{\partial(\phi S_\alpha)}{\partial t} = \nabla \cdot \mathbf{q}_\alpha + q_{f,\alpha}, \qquad \alpha = w, g, \tag{11}$$

$$P_g = P_w + P_{c,w}, \tag{12}$$

in which the subscript α denotes phases, which include liquid (w) and gas (g) for this case; $\mathbf{q}_\alpha$ is Darcy's flux for phase α; $k_{r,\alpha}$ is relative permeability; $\mathbf{k}$ is absolute permeability; ρ_α is density; μ_α is viscosity; $q_{f,\alpha}$ denotes sink/source terms; g is the gravity acceleration constant; z is the vertical coordinate; P_α is phase pressure; and S_α is phase saturation. The phase pressures are related through the capillary pressure $P_{c,w}$ as indicated in Eq. (12), and the phase saturations are constrained by $S_w + S_g = 1$. The primary state variables thus consist of liquid pressure P_w and gas saturation S_g and are denoted by $\mathbf{s} = \{P_w, S_g\}$.

The DL-based surrogate modeling aims to find an approximate autoregressive model, $\hat{\mathbf{s}}_{t+1} = \hat{f}(\hat{\mathbf{s}}_t, \Lambda, \mathbf{q}_t, \boldsymbol{\xi}_t; \theta)$, in which θ represents the network parameters. Previous works using DL techniques mainly perform single-task learning by predicting a single state variable such as the CO_2 saturation field [52] or by training separate neural networks to predict different state variables such as the pressure and saturation [54]. In many real-world problems, different learning targets may share similar information, which is particularly true for coupled PDEs. In the case of carbon sequestration, for example, pressure and saturation variations are coupled and are related to the same forcings (e.g., injection/production) and static model parameters (e.g., permeability and porosity). Thus, I am motivated to apply the multitask learning (MTL) principle to learn the pressure and saturation simultaneously. MTL was originally introduced to improve generalization performance of machine learning models, which it does so by sharing the domain-specific information contained in the training samples of related tasks [75].

Fig. 1 shows the DeepMTL architecture, in which the basic skeleton is a U-Net model widely used in image segmentation [76]. Convolution neural net (CNN) is the building block of U-Net (the solid green color blocks in Fig. 1). A CNN block consists of one or more convolutional layers that perform convolution operations on inputs from the previous layer [77]

$$\begin{aligned} \mathbf{x}_c^l &= \sigma\left(\sum_{c'} \mathbf{W}_{c',c}^l \otimes \mathbf{x}_{c'}^{l-1} + b_c^l\right), \\ x_{i,j,c}^l &= \sigma\left(\sum_m \sum_n \sum_{c'} w_{m,n,c',c}^l x_{i+m,j+n,c'}^{l-1} + b_c^l\right), \\ & \quad i = 1, \ldots, H,\ j = 1, \ldots, W,\ c = 1, \ldots, C_f \end{aligned} \tag{13}$$

in which $\mathbf{x}^{l-1}$ and $\mathbf{x}^l$ are the input and output tensors of the lth layer; subscripts m and n denote indices along the width and height dimensions of a kernel; c' is the index along the channel dimension; c represents the index of output channel dimension C_f, which is equal to the number of kernels used for convolving the lth layer; $\otimes$ is a convolution operator defined in the second line of Eq. (13); $\mathbf{W}_{c',c}^l = \{w_{m,n,c',c}^l\}$ represents the weight matrix of the cth kernel for the input channel c', $\mathbf{b}^l = \{b_c^l\}$ represents a bias vector, and both are trainable parameters (i.e., $\theta = \{\mathbf{W}, \mathbf{b}\}$); and σ represents the activation function. U-Net consists of a downsampling and then an upsampling segment. The downsampling segment (also called an encoder) is designed to capture fine-scale image contexts by using repeated convolutional blocks to progressively extract fine-scale feature maps, whereas the upsampling segment (also known as a decoder) is designed to progressively enlarge the feature maps until the desired image dimension is restored. In addition to its U-shaped architecture, the other distinct feature of U-Net is its use of skip connections (dashed lines in Fig. 1) that combine downsampled and upsampled feature maps at the same level to further improve representation learning [76].

For the carbon sequestration use case, the input tensor includes the system state variable at time t, $\mathbf{s}_t = \{P_{w,t}, S_{g,t}\}$, the spatially heterogeneous log-permeability map $\ln(\mathbf{k})$, and one or more action variables. The output tensor is the system state at the next step $\mathbf{s}_{t+1}$ (Fig. 1). Thus, the MTL design here follows the so-called hard parameter sharing, in which all tasks share the same hidden layers and only differ at the output layer to allow generation of task-specific images [78]. The ReLU activation function is used for all hidden layers, and tanh activation function is used at the output layer. The two learning tasks are trained using separate loss functions (see Section 3.3 below).

Fig. 2 summarizes the overall DQL workflow that is facilitated by the DeepMTL surrogate model. Both the deep Q network and target network share the same architecture, which uses CNN blocks for the input and hidden layers and fully connected dense layers for the output layer. The inputs to the network models are $\mathbf{s}_t$ and a_t, which are stacked along the channel dimension of the input tensor. The output from DQN is a Q-value vector having the same dimensions as the action space. ReLU is used as the activation function for all hidden layers, and the linear function is used at the output layer to generate Q values. DeepMTL is used as a state-transition approximator, and the selected actions from each planning step are stored in a memory buffer, which is implemented as a queue data structure. This workflow is repeated for all planning steps and then for all episodes. In the following section, the potential use of this DQL framework is demonstrated on optimal carbon sequestration reservoir management.

3. Demonstration problem setup

A key to safe commercial-scale geological carbon storage operations is an understanding of how much and how long the injected CO_2 can be stored in host geological formations, such as saline aquifers or depleted oil and gas reservoirs, without unintended migration [79,80]. Injecting large volumes of CO_2 may cause large-scale pressure perturbations and displacement of native formation fluids, affecting a subsurface volume that can be significantly larger than the CO_2 plume itself [81]. While recent subsidies and tax incentives on carbon sequestration have attracted interested parties to invest in carbon capture and storage projects, the potential risks must be factored into considerations during planning [82]. Besides operational costs, other types of costs also need to be considered, including monitoring costs and legal costs. In general, two types of carbon reservoir management strategies may be identified. The passive reservoir management strategy aims to design and deploy a comprehensive monitoring program upfront to track CO_2 plume movement and demonstrate plume containment, while the active reservoir management strategy combines CO_2 injection with brine extraction to mitigate far-field pressure buildup, thus increasing injectivity and constraining brine and CO_2 leakage [83]. So far, cost optimization of storage reservoir pressure management strategies has been conducted mainly using traditional model-based optimization tools [34,84,85], with only a few studies considering formation spatial heterogeneity in monitoring network designs [15,17,86].

As mentioned previously, sequential carbon reservoir management planning represents a case of high-dimensionality in both model parameter space and design space, which motivates the use of DQL in this study. For demonstration, I considered a single-layer carbon storage reservoir with lateral dimensions 1280 m × 1280 m and a uniform thickness of 15 m. The grid is discretized uniformly into 10 m×10 m cells. The reservoir is assumed to be an infinite-acting reservoir (i.e., no lateral boundary effect during the simulation period), and the top and bottom boundaries are no-flow boundaries. The permeability distribution is log-normal, with a mean $\ln k$ value of 4.6 (i.e., a geometric mean of 100 mD) and a standard deviation of 1.5. The variogram type is Gaussian

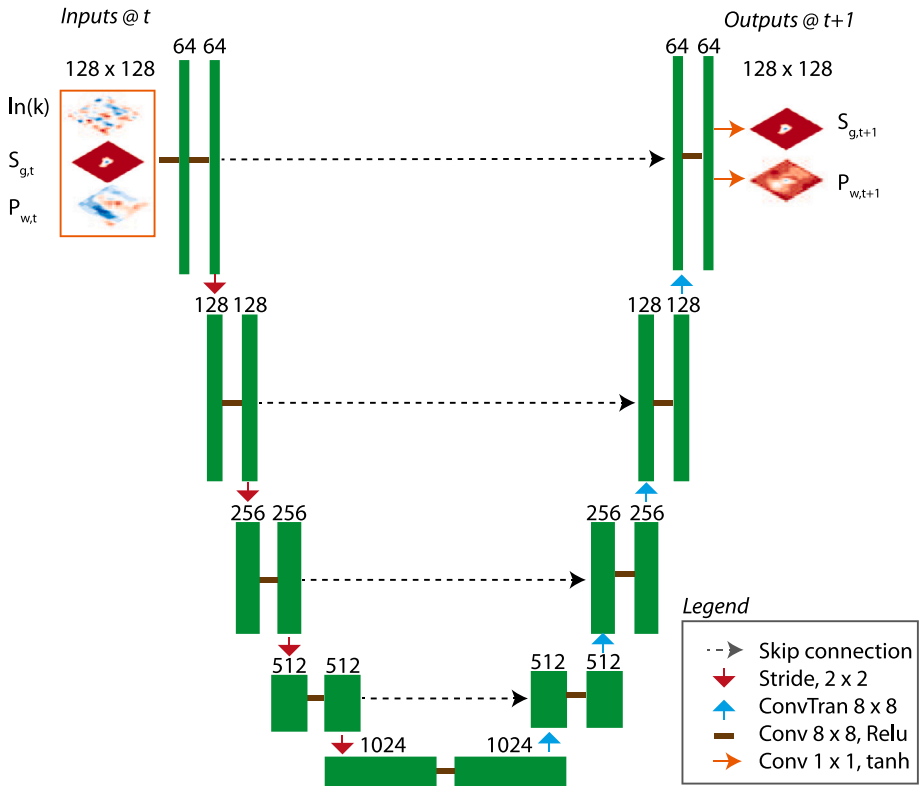

Fig. 1. A U-Net architecture is adopted for deep multitask learning (DeepMTL). Here the inputs include stacked log-transformed permeability ($\ln k$), gas saturation (S_g), and liquid pressure (P_w) at t, and the outputs include gas saturation and liquid pressure at $t+1$ that are learned by using different loss functions.

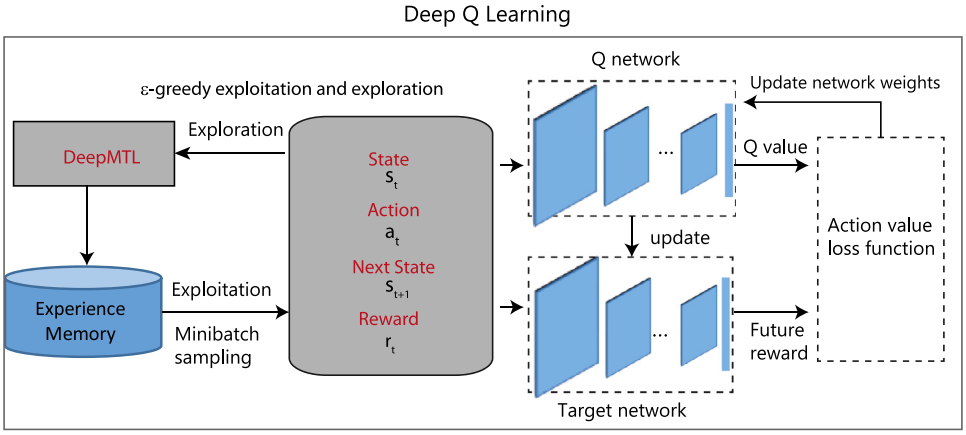

Fig. 2. A deep reinforcement learning workflow facilitated by deep multitask learning for state transition approximation and a deep Q network for Q-function learning. An experience or memory buffer is used to store past state–action pairs for exploitation.

with maximum and minimum correlation ranges of 250 m and 100 m, respectively, and the azimuth angle is 60°. A total of 400 realizations of the log-permeability fields were generated using the sequential Gaussian simulation module from the open-source geostatistical software, SGeMS [87]. The multiphase flow problem setup and the number of realizations are consistent to our previous studies [52].

The planned injection duration is 1800 d. To cast the reservoir management problem into an MDP, the total injection period is divided uniformly into 10 intervals (or stages), each having a duration of 180 d and requiring either the same or different actions. The optimal policy π^* is one that maximizes the total reward for the reservoir operator. In other words, this work deals with a multistage decision-making process described in Bellman [88]—regardless of the initial state and initial decision, the optimal policy must ensure that the remaining decisions are optimal with regard to the first decision.

Table 2 summarizes all major model parameters and their values used in this study.

3.1. Passive management scenario

In the passive management scenario, a variable-rate injector is located at the center of the domain. The maximum injection rate is 5×10^4 m^3/d (at surface conditions) and is discretized into 5000 m^3/d intervals. Thus, the discrete action space consists of 10 actions corresponding to different injection rates in increments of 5000 m^3/d. The optimal management problem is to identify the optimal injection

Table 2
Parameters used in the optimal carbon storage reservoir management examples.

Parameter	Value	Parameter	Value
Total duration [d]	1800	Log(k), mean	4.6
Planning step length [d]	180	Log(k), std. dev	1.5
Action space dimension	10	Rock compressibility [1/kPa]	1e−6
Max injection rate [m^3/d]	5×10^4		
Brine density [kg/m^3]	1100	Initial reservoir pressure [MPa]	15.1

rate for each period. For demonstration purposes, the following reward model is proposed

$$r(\mathbf{s}_t, a_t) = \begin{cases} (1.0 - c_{op} - c_{mon})\Delta M_{co2}(\mathbf{s}_t, a_t)v_{co2}, & \text{when } \Delta P^{mon} \le \Delta P_{max} \\ 0, & \text{when } \Delta P^{mon} > \Delta P_{max} \end{cases} \quad (14)$$

$$c_{mon} = \beta_1 \exp(\beta_2 \Delta P^{mon}/\Delta P_{max}), \quad (15)$$

in which $v_{co2} = \$50$ is the tax credit per ton of CO_2 injected, $\Delta M_{co2}(\mathbf{s}_t, a_t)$ is the total mass of CO_2 injected during period t under action a_t. The operational cost c_{op} is assumed to be fixed at 5% of the total tax credit (i.e., $c_p = 0.05$). The reservoir is monitored through a set of monitoring wells. Here the agent is assumed to be risk averse—as the pressure buildup increases, the agent would increase the monitoring cost nonlinearly to mitigate potential risks and liability. For demonstration, the monitoring cost, c_{mon}, is modeled as an exponential function of the maximum monitoring well pressure buildup ΔP^{mon}, which is defined as,

$$\Delta P^{mon} = \min\left\{ [\Delta P^{mon}_{iw}]_{iw=1}^{N_w}, \Delta P_{max} \right\}, \quad (16)$$

in which iw is the index of monitoring wells, N_w is the total number of monitoring wells, and ΔP_{max} is the maximum pressure buildup that can be tolerated based on the considerations of reservoir conditions (e.g., fault activation, induced seismicity) and the operator's own risk policy. Operation is halted when ΔP_{max} is exceeded at any of the monitoring wells and the reward becomes zero. In the literature, the reported pore pressure perturbations that can trigger earthquakes vary greatly, from ~1 kPa [89,90] to ~1 MPa [91]. I set $\beta_1 = 0.01$, $\beta_2 = 4.0$, and $\Delta P_{max} = 0.2$ MPa in the base case. The resulting monitoring cost curve, shown in Supporting Information (SI) 1, has a maximum of 0.545 at ΔP_{max}.

3.2. Active reservoir management scenario

Under the active reservoir management scenario, a set of brine production wells are added to actively reduce reservoir pressure. Here the production rate is constrained by the bottom-hole pressure and is estimated using Peaceman's well model [92]

$$q_w = \frac{2\pi\rho k k_r(S_w)\Delta D}{\mu} \frac{P - P_{bhp}}{\ln(r_e/r_w)}, \quad (17)$$

in which $k_r(S_w)$, the relative permeability, is a function of water saturation S_w; ΔD is reservoir thickness, which is 15 m; P_{bhp} is the well bottom-hole pressure, which is set to the initial reservoir pressure of 15.1 MPa; P is reservoir pressure at the well block; the well radius r_w is 0.05715 m; and the equivalent radius r_e is calculated using

$$r_e = \frac{1}{4}\exp(-\gamma_o)\left(\Delta x^2 + \Delta y^2\right)^{1/2}, \quad (18)$$

in which $\gamma_o = 0.5772$ is Euler's constant, and the block sizes $\Delta x = \Delta y =$ 10 m in the current problem. All other parameters are the same as before.

The proposed reward function under this scenario is

$$r(\mathbf{s}_t, a) = (1.0 - c_{op})\Delta M_{co2}(\mathbf{s}_t, a)v_{co2} - c_{brine}\Delta M_{brine}(\mathbf{s}_t, a) \quad (19)$$

in which ΔM_{brine} is the total mass of brine extracted in period t for taking action a, and c_{brine} is the brine extraction and disposal cost per unit mass. The operational cost c_{op} is assumed to be fixed at 10% of the total tax revenue. In the literature, brine treatment costs are estimated to be in the range of \$0.5–2.0 per m^3, not including disposal and other costs [93,94]. The sensitivity of the total reward to brine extraction cost will be examined as part of the sensitivity study.

3.3. Network implementation and training

3.3.1. DeepMTL

All deep neural network models are implemented in PyTorch. Reservoir simulations are performed using the compositional reservoir simulator CMG–GEM, hereafter referred to as CMG for short [95]. Each simulation run uses a different permeability realization from the 400-realization ensemble, and the simulation results are saved for every 30 d. For each 180-day planning step, the injection/production rates are randomly varied by performing Latin hypercube sampling on the discrete action space. This leads to 400 × 60 input–output data pairs: the inputs are permeability and time-varying injection rates and the outputs are pressure and saturation fields. The total data samples, grouped by realizations, are divided into three parts: 70% are used for training, 15% for validation, and 15% for testing.

The efficacy of training can be sensitive to the type of scaling/normalization techniques used to prepare the training samples. For pressure scaling, the procedure suggested by Tang et al. [54] is used, in which the ensemble mean at each time step is subtracted from all data samples corresponding to that step, and the resulting pressure residuals are then linearly scaled to $[-1, 1]$ before training. After training, the above procedure is reversed to obtain the predicted pressure values in the original domain. For saturation scaling, a residual learning procedure used in Sun et al. [96] is adopted (not to be confused with the popular residual neural nets by He et al. [97]), in which the training target is temporal saturation change, $S_{g,t+1} - S_{g,t}$ instead of $S_{g,t+1}$. After training, the predicted saturation residual is added to $S_{g,t}$ to generate the desired output $S_{g,t+1}$. The rationale is that the saturation between adjacent steps may be too similar to learn; thus, it is more effective to learn the saturation changes instead. Preliminary analyses show that using residual learning on saturation changes performs better than predicting $S_{g,t+1}$ directly.

Training of DeepMTL has two subtasks: saturation learning and pressure learning. For saturation, the mean square error (MSE) is used as the loss function,

$$\mathcal{L}_s = \frac{1}{N_T}\sum_{i=1}^{N_T} \|\mathbf{y}_i^{(S)} - \hat{\mathbf{y}}_i^{(S)}\|_2^2, \quad (20)$$

in which $\mathbf{y}$ and $\hat{\mathbf{y}}$ are simulated and predicted training samples, respectively; the superscript S denotes the saturation map; and N_T is the total number of training samples. For pressure, the mean absolute error (MAE) is used,

$$\mathcal{L}_P = \frac{1}{N_T}\sum_{i=1}^{N_T} |\, \mathbf{y}_i^{(P)} - \hat{\mathbf{y}}_i^{(P)} \,|, \quad (21)$$

in which superscript P denotes pressure. The total loss function is a weighted sum of the two, $\mathcal{L} = w\mathcal{L}_s + \mathcal{L}_P$, in which w is set to 80.0 based on preliminary analyses. During training, the samples are randomly shuffled to improve model generalization and reduce overfitting.

All network models are trained using the ADAM solver [98], with a learning rate of 0.0005, first-moment decay rate of 0.5, and second-moment decay rate of 0.999. The number of training epochs used is 100, and the minibatch size is 50. The total number of training samples is 16,800 (i.e., 70% of the 400 × 60 training pairs). Training was carried out on a dual-processor node equipped with 128 Gb of RAM and an Nvidia 1080-TI GPU running the CentOS Linux operating system. On average, each epoch took about 2 min 50 s to complete.

Two common metrics are used to measure the network performance on testing data: MSE and structural similarity index (SSIM). The latter is defined as

$$\text{SSIM}(u, v) = \frac{(2\mu_u\mu_v + c_1)(2\sigma_{uv} + c_2)}{(\mu_u^2 + \mu_v^2 + c_1)(\sigma_u^2 + \sigma_v^2 + c_2)}, \quad (22)$$

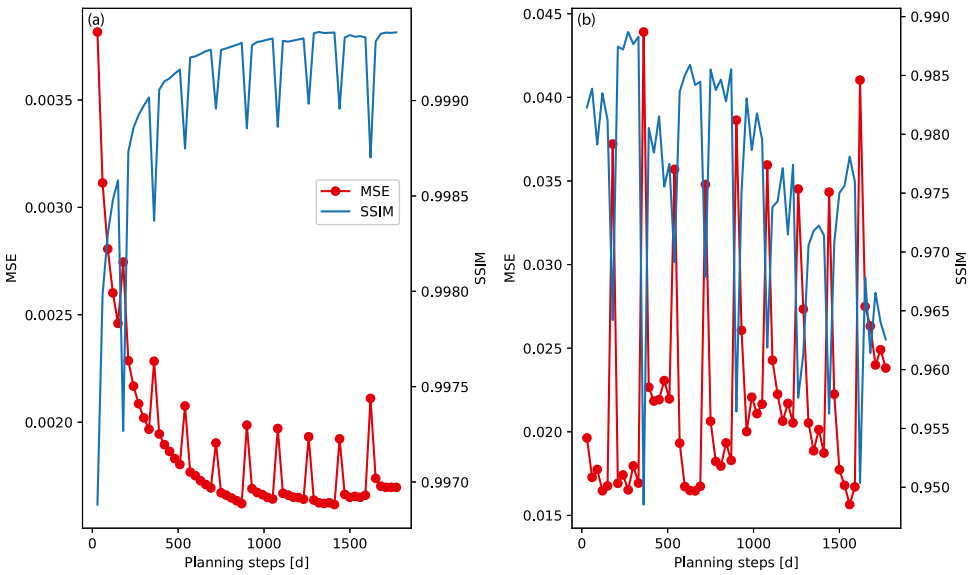

Fig. 3. Test performance metrics of the DeepMTL autoregressive model for the passive reservoir management scenario on (a) gas saturation S_g and (b) pressure P_w. The left axis of each subplot is MSE and the right axis is SSIM. The test is performed on the results from 60 models which include 60 output samples each.

in which μ and σ represent the mean and standard deviation of image patches falling in two sliding windows, and c_1 and c_2 are small constants introduced to avoid numerical instability [99]. The global SSIM is obtained by averaging the patch SSIM values, and its value falls in the range $[-1, 1]$, with higher values indicating better pattern matches. The sizes of the sliding windows used are 11×11.

3.3.2. DQN

The DQN is trained by minimizing the loss function in Eq. (6) using the ADAM solver with a learning rate of 0.0003. The value of discount factor γ is 0.99. The size of the memory buffer is set to 10,000, the minibatch size is 32, and the τ value used for soft updating DQN weights is set to 0.01 (see Eq. (7)). In the ϵ-greedy optimization, the ϵ value decays with the progressing of episodes as

$$\epsilon = \epsilon_f + (\epsilon_i - \epsilon_f) \exp\left(-\text{episode}/\lambda_D\right), \tag{23}$$

in which the initial value of ϵ used is $\epsilon_i = 1$, the final value is $\epsilon_f = 0.005$, and the decay factor λ_D is set to 600 episodes. As mentioned before, the strategy used in Eq. (23) encourages more exploration during the beginning of training and then more exploitation during the later stages of training. All DQN models are trained using 5000 episodes for this study, unless otherwise specified. The computing time for the 5000-episode training is about 3.5 h on the computing node mentioned in the above.

4. Results and discussion

4.1. Passive management scenario

After the DeepMTL autoregressive model is trained, it is tested on a set of 60 CMG model runs not used during training. Recall that each model run is generated using a different permeability realization and includes outputs from 60 output steps.

Fig. 3 shows the test metrics, MSE and SSIM, for gas saturation and liquid pressure, respectively. Results shown in Fig. 3a suggest that gas saturation prediction achieves relatively high accuracy, especially at late times, which is important for planning. On average, the MSE on gas saturation prediction is smaller than 0.004 (calculated on the scaled saturation maps, see Section 3.3), and the SSIM is above 0.995. The reason for increased prediction accuracy at late times is because the saturation changes slow down (except near the edges) as the plume size grows bigger, which helps the deep learning algorithm to learn the shape of the plume. Pressure prediction is more challenging than saturation prediction because of the rapid speed of pressure propagation in porous media, which leads to fast changing patterns, especially when the rate is also allowed to vary. Fig. 3b, which is also calculated on scaled pressure outputs, suggests that the performance metrics oscillate with time because of step changes in injection rates. The pressure MSE is around 0.02 during non-rate change times but jumps to about 0.04 when rate change occurs. The pressure SSIM exhibits a similar pattern but generally stays above 0.95. Note the saturation metrics are also affected by rate changes but to a lesser extent.

As an example of actual model outputs, Fig. 4 compares DeepMTL and CMG results side by side on a test realization at four different prediction times: 150, 300, 1050, and 1500 d. For reference, the corresponding log-transformed permeability field and the normalized injection rate history are plotted at the top. The injection rate history is normalized by the maximum injection rate for visualization purposes. In this particular example, the maximum injection rate ($5 \times 10^4 \text{m}^3/\text{d}$) happens to be imposed right at the beginning of the first 180-d period, and the injector is located close to a low-permeability zone (dark red color), leading to relatively large initial pressure responses. The predicted (third row) and CMG-simulated (second row) saturation maps show strong similarity. The largest saturation residuals, defined as the difference between CMG simulations and DeepMTL results, are mainly found near the plume edges but are generally limited to less than 0.01, except for the 150-d snapshot in which the maximum error residual is around 0.09. The similarity between predicted and simulated pressure maps is not as close as that between the saturation maps. The absolute difference is around 200 kPa, which occurs near the injector during the first period when the maximum injection rate is imposed. The realization selected here is used as the base case for the rest of this scenario.

Predicted pressure values at the monitoring well locations serve as indicators of risk levels in this scenario. In the carbon storage literature, the monitoring network optimization problem has been investigated [15,17]. A general principle for monitoring network design is that monitoring well locations should be sensitive to pressure anomalies. Another consideration is that the approximation error should be small when surrogate models are used for the design. In Fig. 5, the pressure approximation error due to surrogate modeling is quantified at four potential monitoring well locations (labeled by open green circles in Fig. 4) by calculating error residuals between the CMG-simulated and

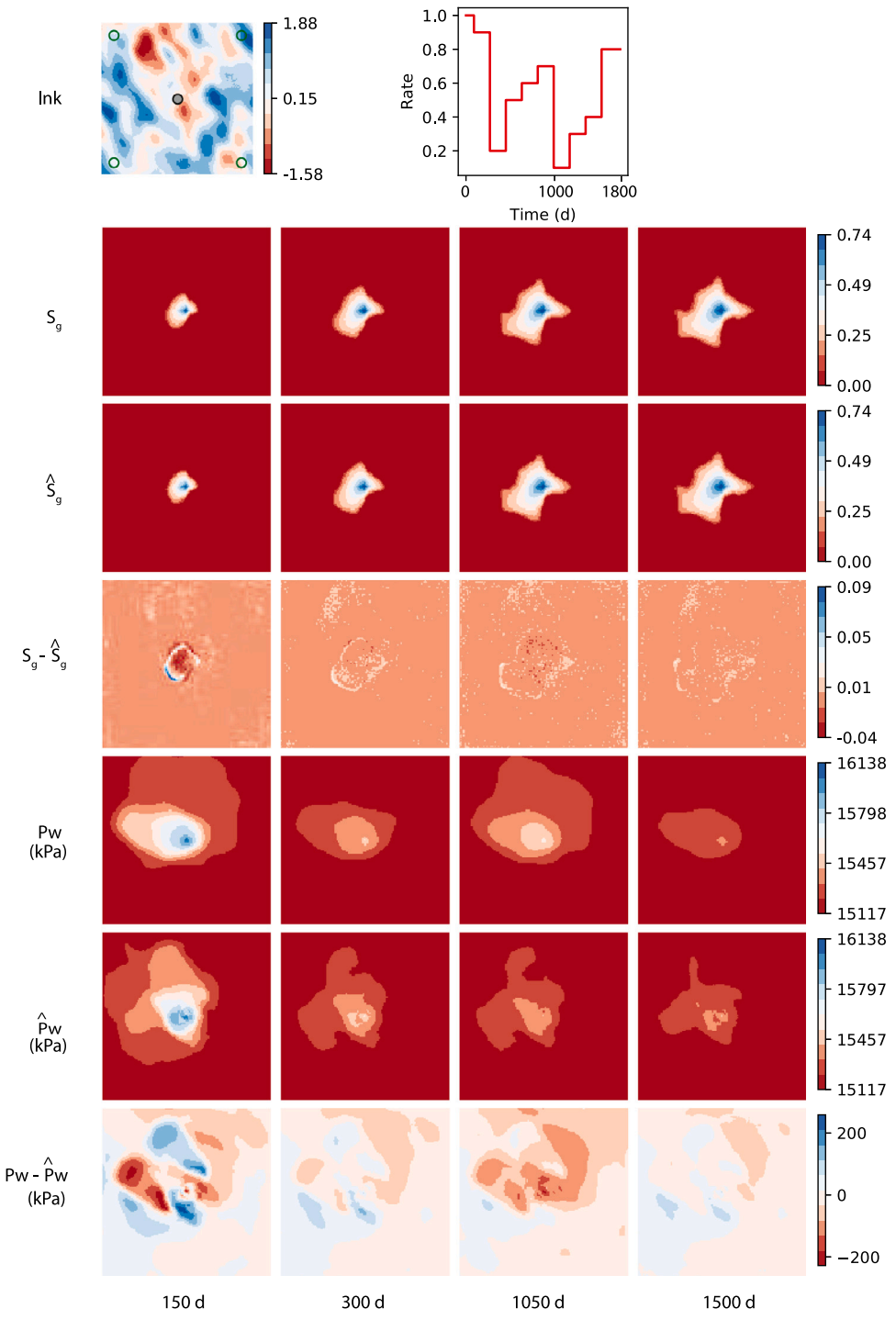

Fig. 4. DeepMTL results for a test realization used in the passive reservoir management scenario. First row: left, permeability field (log transformed) and locations of monitoring wells (origin in left upper corner); right, normalized injection rate as a function of time. Second to fourth rows: simulated gas saturation, predicted gas saturation, and difference between the two. Fifth to seventh rows: simulated liquid pressure, predicted pressure, and the difference between the two. The output times are (from left to right) 150, 300, 1050, and 1500 d.

DeepMTL results of training data. In this example, reservoir pressure tends to be more responsive to injection rate changes along one diagonal of the model domain than the other; this is because of the anisotropy of correlation ranges used in generating permeability realizations. Previous experience suggests that sensitivity to external forcing adds more patterns to the training data, thus improving the efficacy of pattern-based DL while reducing approximation errors [49]. Fig. 5 suggests that approximation errors at wells (10,10) and (118,118) are generally less than 5 kPa except for short periods immediately after rate changes. The errors at the other two well locations, (10,118) and (118,10), are around 50 kPa. Thus, wells (10,10) and (118,118) are used as monitoring wells. To account for the surrogate model error during DQN training, the pressure check given in Eq. (16) is modified to $\Delta P^{mon} = \min\{[\Delta P_{iw}^{mon} + \Delta P_{iw}^{95}]_{iw=1}^{2}, \Delta P_{max}\}$, in which ΔP_{iw}^{95} is the 95th confidence bound derived from training results, as shown in Fig. 5.

The trained DeepMTL model is then incorporated into the DQL framework shown in Fig. 2 to optimize injection strategies. As a test, I first considered a case in which the monitoring cost is ignored in the

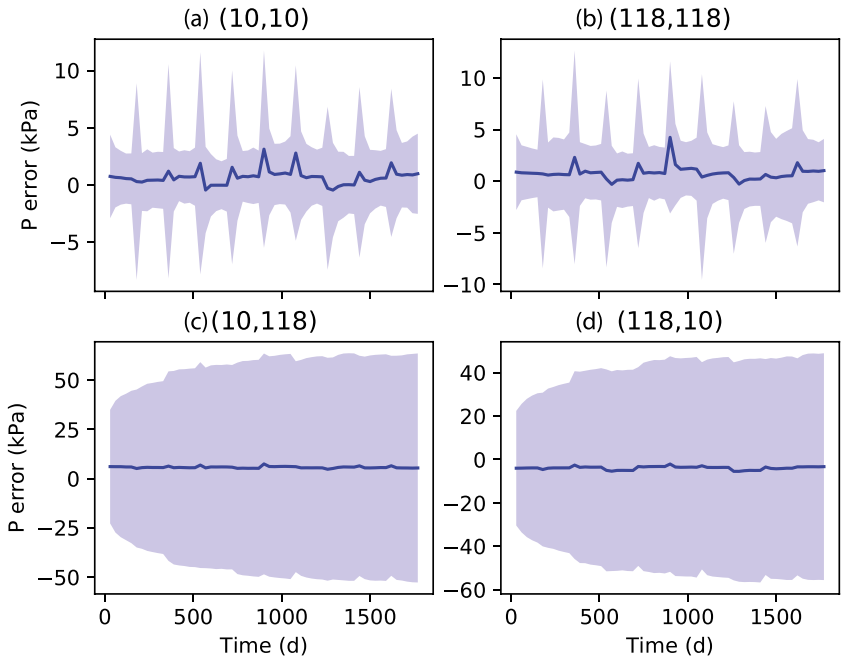

Fig. 5. Mean (solid blue line) and 95% confidence bounds (light-blue shaded area) of pressure approximation errors (i.e., simulated surrogate model) obtained at grid blocks (a) (10,10), (b) (118,118), (c) (10,118), and (d) (118, 10).

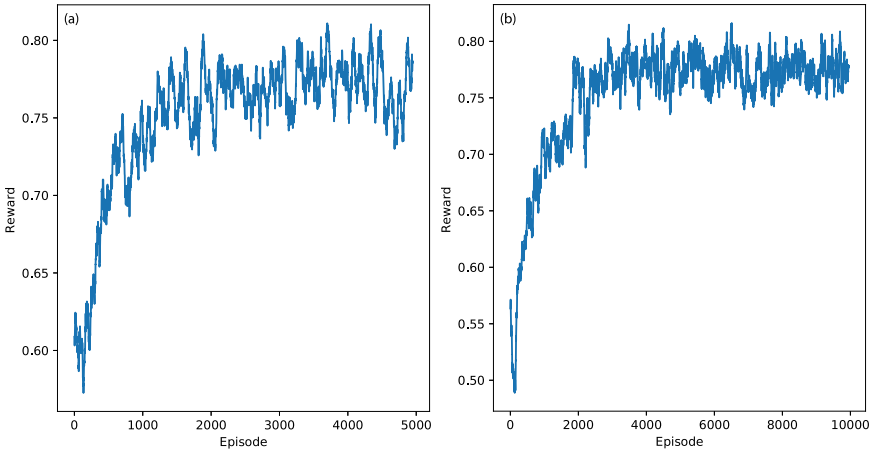

Fig. 6. Average total reward as a function of training episodes for (a) 5000 episodes and (b) 10,000 episodes. Total reward for each episode is the sum of reward from all action steps and is scaled by the maximum possible tax credit for the duration. The total reward is calculated as the moving average of past 50 episodes.

reward function Eq. (14). The optimal policy is trivial for this case, which is to impose the maximum possible injection rate for all steps regardless of pressure buildups. The total reward of each episode is the cumulative sum of all rewards from the 10 consecutive action steps, and the maximum tax credit for this case is $8.46 million, assuming a CO_2 density of 1.98 kg/m^3 at surface conditions. Fig. 6a shows the training results after 5000 episodes. Each average total reward is calculated as the moving average of the previous 50 episodes and is scaled by the maximum possible tax credit. During DQN training, the total reward gradually increases until it reaches an equilibrium range between 0.75 and 0.80. The essence of reinforcement learning is to train the agent to try different actions and to progressively favor those that appear to be the best. In this example, the shape of the average total reward profile suggests that the agent is able to learn from its repeated interactions with the environment to pick higher-reward actions, and the performance of the training becomes stable at large episodes—an observation that is consistent with the original DQN work [66]. The total reward always oscillates somewhat, which is normal because of the stochastic nature of the problem. Increasing the number of episodes further, from 5000 to 10,000, did not seem to change the results significantly (Fig. 6b).

To evaluate the trained DQN, I ran it on 500 test episodes by fixing the ϵ value in the ϵ-greedy search to a small value (0.05) according to [66] and finding the optimal policy corresponding to the maximum reward found during testing. For the base case environment, the DQN successfully identified the maximum rate injection as the best action for all 10 steps; namely, the optimal policy recommended by the DQN was to do constant-rate injection at the maximum rate available.

Design under uncertainty has been a challenge for monitoring optimization problems [15,47,100,101]. In the current context, the permeability is uncertain because of spatial heterogeneity which, in turn, makes the reinforcement learning environment uncertain. One question is whether the agent can transfer the experience gained from interacting with one environment to other similar environments. This has been demonstrated as feasible by Mnih et al. [66], who successfully trained an agent to play different Atari games. For this study, I applied the trained DQN on nine other permeability realizations, each presenting a slightly different environment not seen during DQN training. Note

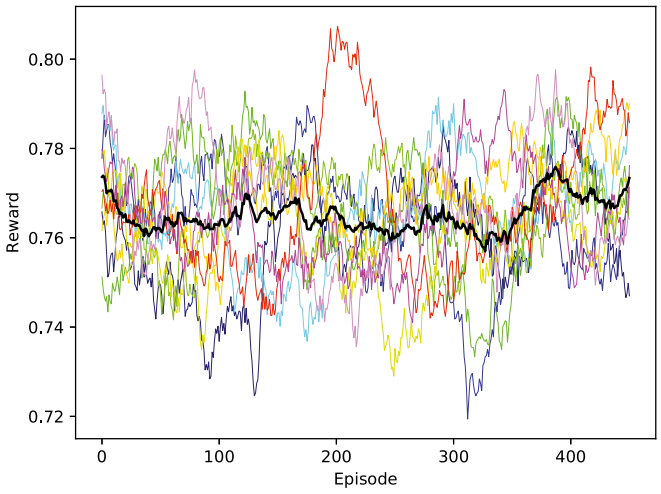

Fig. 7. Evaluation of trained DQN for 10 different environments (i.e., permeability realizations), each running over 500 testing episodes. The thin color lines show the 50-episode moving average of total rewards for each realization, and the thick dark line shows the ensemble average.

that because DQN is a model-free method, the effect of environment is reflected indirectly through the approximate state transitions as predicted using DeepMTL.

Fig. 7 shows the average total reward of 500 episodes for all 10 realizations, including the base case realization used for training. The ensemble average is shown by the dark solid line. The plot suggests that the performance of the agent stayed relatively robust, even in environments not seen during training. For all environments considered in this test, the trained agent was able to identify the constant-rate injection at the maximum rate as the best policy. This result is particularly encouraging in terms of reinforcement learning because it suggests that the trained agent can recognize similarities in states and take similar actions to achieve similar awards, despite uncertainty in the environments. Equivalently, this suggests that samples from the original memory buffer (i.e., tuples consisting of state, action, reward, and next states) can still be applied to solving MDP problems, as long as the new environments are not perturbed too significantly from the original (i.e., in the sense of equally likely environments).

Now the effect of monitoring cost on the optimal policy is investigated by considering the full reward model given in Eq. (14). Fig. 8 shows the optimal actions identified for pressure thresholds (ΔP_{max}) of 200, 150, 130, and 120 kPa, respectively. In all cases, the environment is fixed to the base case realization. In the case of ΔP_{max} = 200 kPa, the identified optimal policy is still constant injection at the maximum rate (Fig. 8a), but the average award is smaller (see SI 2). Inspection of the episode history reveals that injector shut-in due to overpressure never occurs for this case. For ΔP_{max} = 150 kPa (Fig. 8b), injector shut-in instances start to appear. The DQN-identified optimal policy is to lower the injection rate to 40,000 m^3/d in the second period but otherwise maintain the maximum injection rate. When ΔP_{max} is decreased to 130 kPa (Fig. 8c), the number of injector shut-in instances increases. The identified optimal policy is to use the maximum injection rate only during the initial period and then use the lowered rate of 45,000 m^3/d for the rest of the operations. Finally, when ΔP_{max} is dropped further to 120 kPa, the optimal policy is to use the schedule from Fig. 8c but reduce the rate in the first and eighth periods (Fig. 8d). Thus, results shown here suggest that the inclusion of ΔP_{max} and monitoring costs has a significant impact on the selected policy. When the rewards gained from maximum injection are not sufficient to offset losses due to increased monitoring costs and injector shut-ins, the agent learns to systematically adjust the injection schedule to avoid excessive periods of injector shut-in while minimizing the monitoring cost.

In summary, tests done under this passive management scenario range from an elementary proof-of-concept example to more realistic cases involving multiple constraints and even different environments. The DQN agent achieves stable performance in these test cases.

4.2. Active management scenario

For the active management scenario, CMG model runs are performed using the same ensemble of the 400 permeability realizations but now with four production wells located at the four corners of the domain. A separate DeepMTL autoregressive model is trained by following the same procedure as that used for the passive management scenario. Fig. 9 shows the metrics obtained on the 60 model runs that are set aside for testing. In general, the results are comparable to those

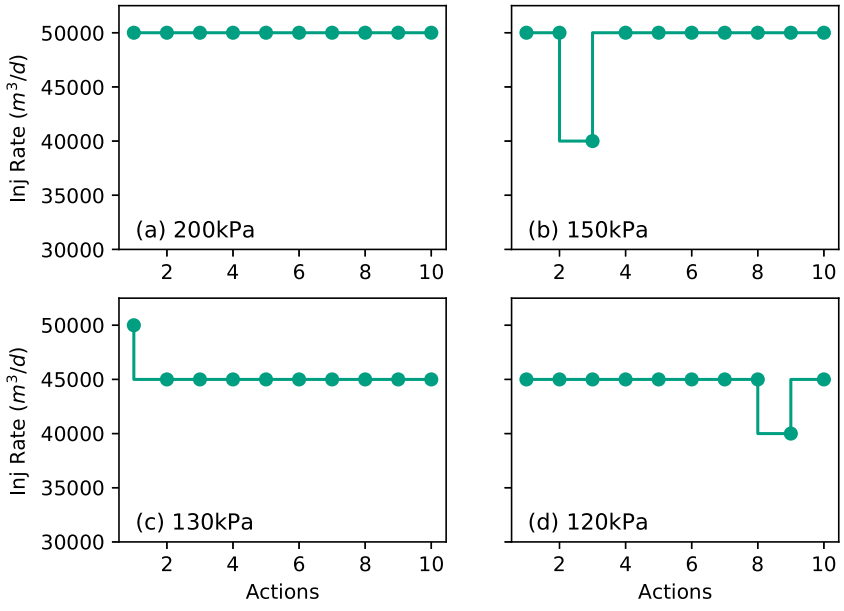

Fig. 8. Optimal policies under different maximum pressure thresholds: (a) 200 kPa, (b) 150 kPa, (c) 130 kPa, and (d) 120 kPa.

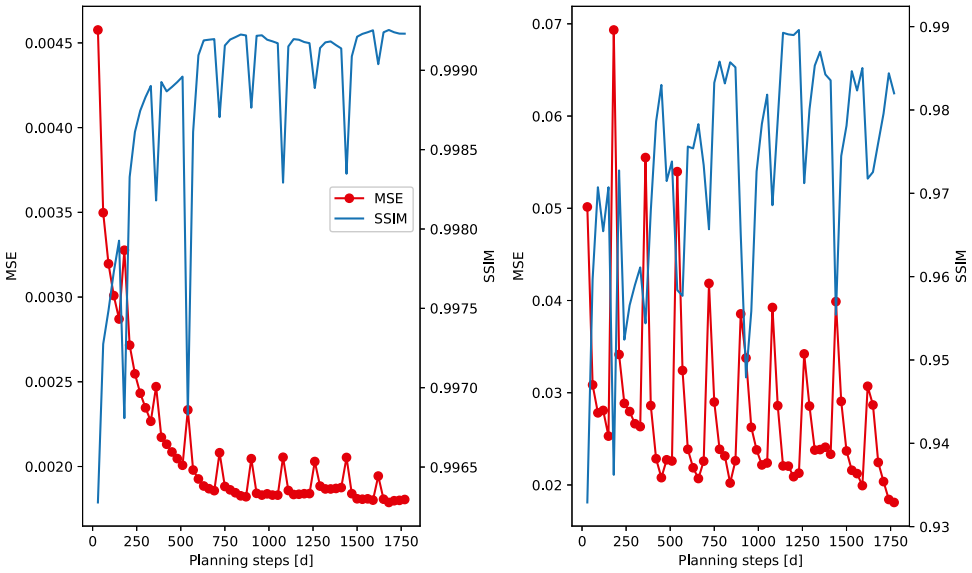

Fig. 9. Test performance metrics of the DeepMTL autoregressive model for the active reservoir management scenario on (a) gas saturation S_g and (b) pressure P_w, both scaled to [0,1]. The left axis of each subplot is MSE, and the right axis is SSIM. The test is performed on results from 60 models (each having 60 steps) not used during training.

obtained under the passive management scenario, even though the patterns now become more complex. The saturation maps are learned relatively accurately—the MSE is below 0.002 at large times, and the SSIM is above 0.995 at all times. The MSE of the pressure stays in the 0.02–0.04 range for most of the planning steps, and the pressure SSIM stays above 0.93.

Fig. 10 compares DeepMTL and CMG results for prediction times at 150, 300, 1050, and 1500 d for a selected test realization, which is shown at the top row. The scaled injection history is also shown. In this particular example, the injection rate starts low, increases to the maximum at 1440 d, and then decreases. Like the base case used in the passive management scenario (Fig. 4), large saturation error residuals occur near the plume edges and are less than 0.06. The pressure contours show similar patterns near the injector but differ in orientation, with the deepMTL results showing a north–south pattern and CMG results showing an east–west pattern.

Fig. 11 compares the mean water production rates obtained using the Peaceman's model Eq. (17) for the four production wells and over the entire test dataset. CMG results are shown as red dashed lines and DeepMTL results as solid blue lines. Overall, the match is satisfactory for the two wells located in blocks (10,10) and (118,118), but the two other wells show overestimates as high as 12%, especially well (118,10).

The trained DeepMTL is combined with the DQN to search for optimal policies under the active management scenario. Like in the previous scenario, the costs are nonlinearly dependent on the state variables because of the bottom-hole pressure constrained extraction rates. Higher injection rates increase the reservoir pressure which leads to higher brine production.

Fig. 12 shows the training history of three models with unit brine treatment costs of $1–3 per ton. In general, brine extraction lowers the average total reward; however, the best policy identified for all three cases is constant-rate injection at the maximum rate. Paradoxically, higher injection rates lead to higher brine production, but the negative costs are offset by higher rewards obtained from the higher injection rates. If the brine treatment cost continues to rise though, the optimal policy will eventually be affected. For the specific reward model chosen here, the changes start to occur at a unit price of about $6, as the profit margin becomes thin. Fig. 13 shows the optimal policies for higher costs at unit treatment costs of $6, $7, and $7.5, respectively, which are obtained by running the trained model on 500 test episodes. For a unit treatment cost of $6, the agent suggests that the injection rate be dropped sharply to 10,000 m^3/d in the ninth period. For unit costs of $7 and $7.5, the agent suggests sustained lower-rate injection to make the total reward positive.

5. Summary and conclusions

The need for system optimization is prevalent in energy system design and management applications. Model-based optimization under uncertainty, however, has been a challenging problem, especially when it comes to geosystem exploration and utilization planning. The main challenges stem from the high-dimensionality of geosystem process-level models, as well as high-dimensionality of the design space. Traditionally, dynamic programming and other heuristic search methods have been applied, but they are often limited to low dimensions. In this work, a deep learning–based reinforcement learning framework is formulated for optimal geosystem management. The deep reinforcement learning framework uses a deep multitask learning approach to develop surrogate models of the multistate, distributed process-level models and then uses a deep Q-learning algorithm to develop function approximators of the Q function, which is a value function that specifies what is the best policy in the long run. Both deep Q-learning network (DQN) and deep multitask learning (DeepMTL) are pattern based, allowing the agent to make decisions based on the patterns shown in the distributed state variables, thus increasing both the range and granularity of potential applications.

As a demonstration, the deep learning framework was applied to carbon storage reservoir management. Results were obtained for both a passive management scenario and an active management scenario, with the main difference being the use of brine extraction wells in the latter case. The multiperiod decision problems involved identifying the optimal injection rate schedule. I show, via both trivial and non-trivial examples, that the deep Q-learning agent can learn to identify the optimal policies to maximize the reward for given risk and cost constraints. I also show that the knowledge gained by an agent from interacting with one environment is largely preserved when deploying the same agent in other similar environments. Although only a 2D reservoir was considered in this study, the DeepMTL is general and can

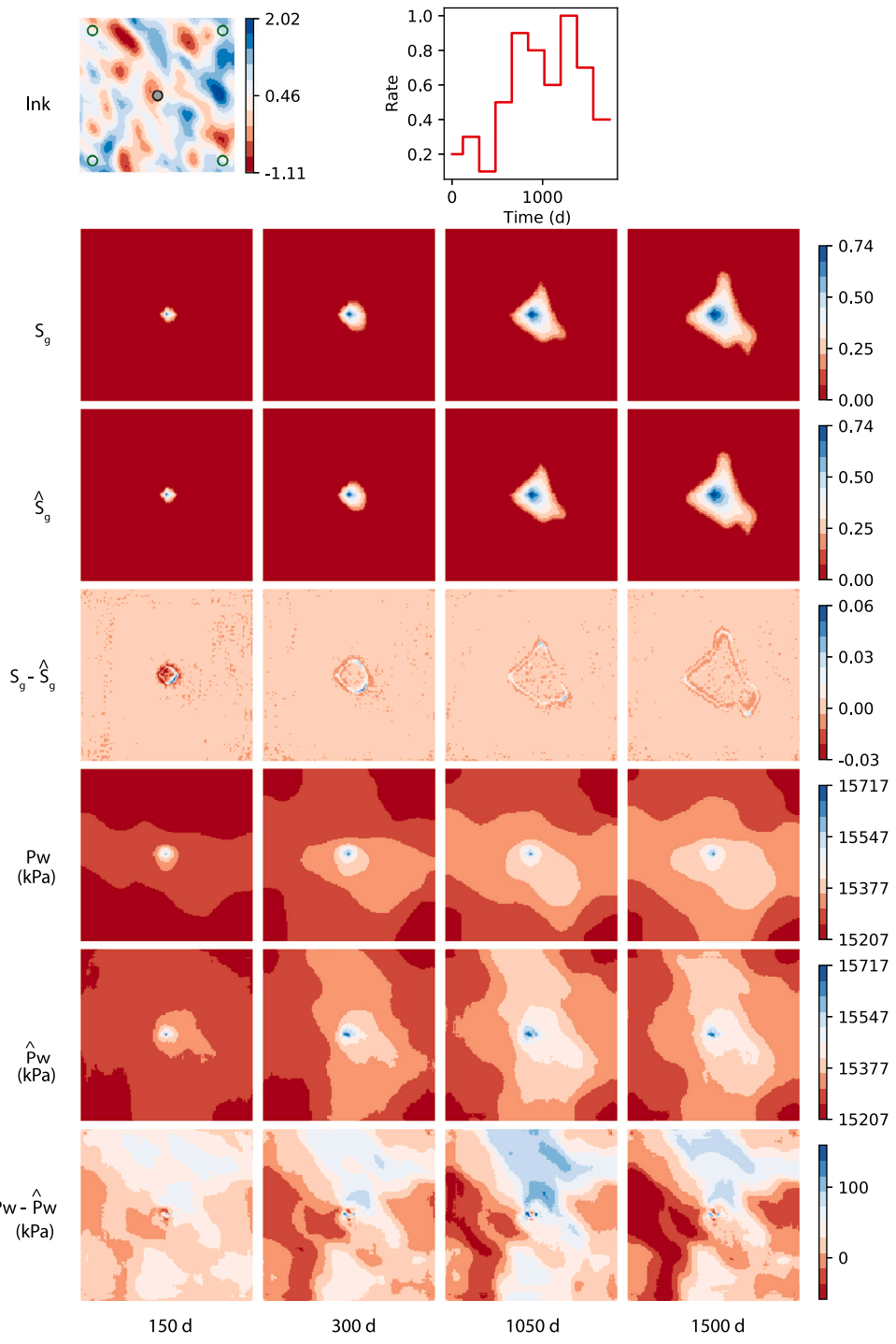

Fig. 10. DeepMTL results for a test realization used in the active reservoir management scenario. First row: left, permeability field (log transformed); right, normalized injection rate as a function of time. Second to fourth rows: simulated gas saturation, predicted gas saturation, and the difference between the two. Fifth to seventh rows: simulated liquid pressure, predicted pressure, and the difference between the two. The output times are (from left to right): 150, 300, 1050, and 1500 d.

be easily extended to 3D reservoirs by switching the reservoir model and adding one more dimension to the tensor arrays.

In this work, the experiments are limited to a discrete action space and episodic learning, in which each episode has a start and an ending state. The field of deep reinforcement learning is evolving quickly, and many new variants of DQN have been developed. The action space is limited to a single type of action (i.e., the injection rate), although extending it to multiple types of actions (i.e., production and injection rates) is feasible. Even for the 10-step action space considered in the demonstration examples, the potential saving in computing cost is sizable, considering that the cost of running each instance of the full numerical model is more than 2 min and the design is subject to uncertainty in permeability.

Thus, the deeplearning framework presented here can serve as a practical tool for solving a wide range of sequential optimization problems involving high-dimensional, process-level models.

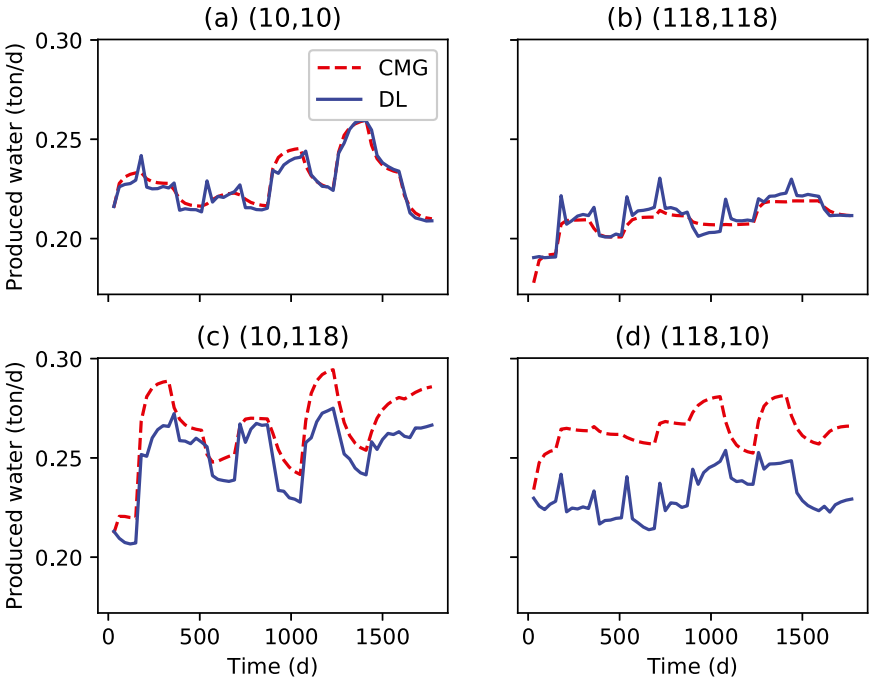

Fig. 11. Comparison of mean water production rate obtained at the four wells for CMG (red dashed lines) and DeepMTL (solid blue lines).

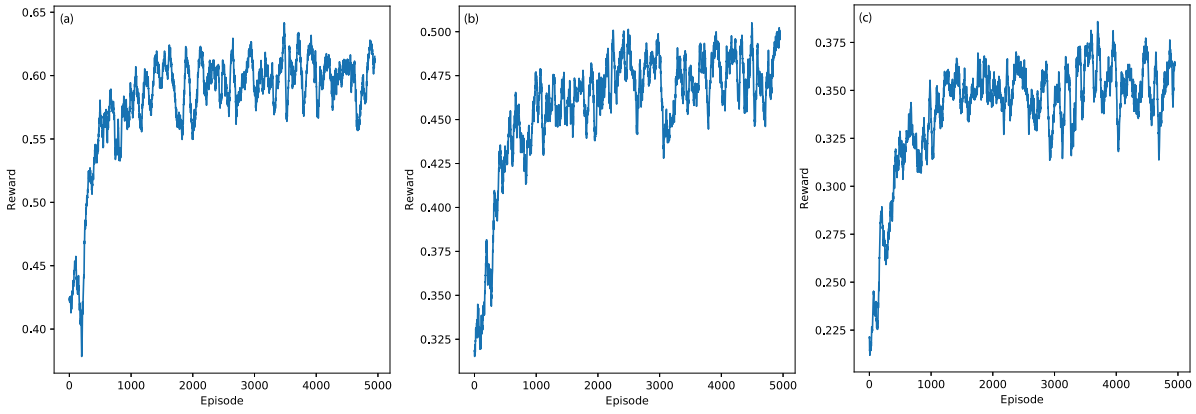

Fig. 12. Average total reward as a function of training episodes for brine extraction cost of (a) $1.0, (b) $2.0 and (3) $3.0 per ton. Each average total reward is calculated as the moving average of the previous 50 episodes.

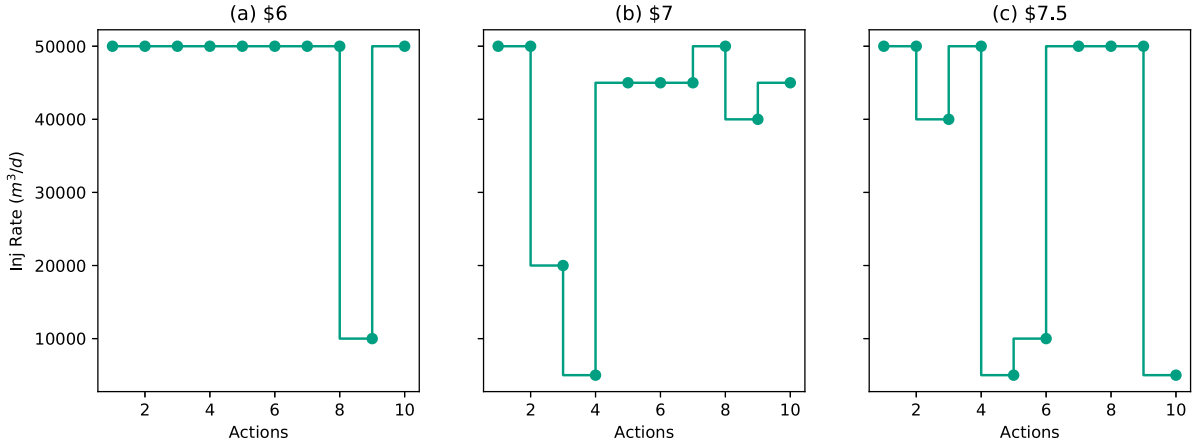

Fig. 13. Optimal policy for brine extraction cost of (a) $6.0, (b) $7.0, and (3) $7.5 per ton.

CRediT authorship contribution statement

Alexander Y. Sun: Conceptualization, Methodology, Software, Formal analysis, Data curation, Writing the draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The author is grateful to the two anonymous reviewers and the editor for their constructive comments. The work was partly supported by the U.S. Department of Energy, United States of America, National Energy Technology Laboratory, United States of America (NETL) under grant number DE-FE0026515. The author is grateful to the Texas Advanced Computer Center at The University of Texas at Austin for providing computing resources.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.apenergy.2020.115660.

References

[1] Nguyen A-T, Reiter S, Rigo P. A review on simulation-based optimization methods applied to building performance analysis. Appl Energy 2014;113:1043–58.
[2] Fan C, Xiao F, Zhao Y. A short-term building cooling load prediction method using deep learning algorithms. Appl Energy 2017;195:222–33.
[3] Chehouri A, Younes R, Ilinca A, Perron J. Review of performance optimization techniques applied to wind turbines. Appl Energy 2015;142:361–88.
[4] Rudolfsson M, Stelte W, Lestander TA. Process optimization of combined biomass torrefaction and pelletization for fuel pellet production–a parametric study. Appl Energy 2015;140:378–84.
[5] Shen F, Zhao L, Du W, Zhong W, Qian F. Large-scale industrial energy systems optimization under uncertainty: A data-driven robust optimization approach. Appl Energy 2020;259:114199.
[6] Wu J, He H, Peng J, Li Y, Li Z. Continuous reinforcement learning of energy management with deep q network for a power split hybrid electric bus. Appl Energy 2018;222:799–811.
[7] Xu B, Rathod D, Zhang D, Yebi A, Zhang X, Li X, et al. Parametric study on reinforcement learning optimized energy management strategy for a hybrid electric vehicle. Appl Energy 2020;259:114200.
[8] Li B, Duan Y, Luebke D, Morreale B. Advances in co_2 capture technology: A patent review. Appl Energy 2013;102:1439–47.
[9] Ampomah W, Balch R, Cather M, Will R, Gunda D, Dai Z, et al. Optimum design of co_2 storage and oil recovery under geological uncertainty. Appl Energy 2017;195:80–92.
[10] Tapia JFD, Lee J-Y, Ooi RE, Foo DC, Tan RR. Optimal co_2 allocation and scheduling in enhanced oil recovery (eor) operations. Appl Energy 2016;184:337–45.
[11] Wang X, van't Veld K, Marcy P, Huzurbazar S, Alvarado V. Economic co-optimization of oil recovery and co_2 sequestration. Appl Energy 2018;222:132–47.
[12] DeCarolis J, Daly H, Dodds P, Keppo I, Li F, McDowall W, et al. Formalizing best practice for energy system optimization modelling. Appl Energy 2017;194:184–98.
[13] Middleton RS, Bielicki JM. A scalable infrastructure model for carbon capture and storage: Simccs. Energy Policy 2009;37(3):1052–60.
[14] Pękala ŁM, Tan RR, Foo DC, Jezowski JM. Optimal energy planning models with carbon footprint constraints. Appl Energy 2010;87(6):1903–10.
[15] Sun AY, Nicot J-P, Zhang X. Optimal design of pressure-based, leakage detection monitoring networks for geologic carbon sequestration repositories. Int J Greenh Gas Control 2013;19:251–61.
[16] Fernández D, Pozo C, Folgado R, Guillén-Gosálbez G, Jiménez L. Multiperiod model for the optimal production planning in the industrial gases sector. Appl Energy 2017;206:667–82.
[17] Jeong H, Sun AY, Zhang X. Cost-optimal design of pressure-based monitoring networks for carbon sequestration projects, with consideration of geological uncertainty. Int J Greenh Gas Control 2018;71:278–92.
[18] Sun AY, Nicot J-P. Inversion of pressure anomaly data for detecting leakage at geologic carbon sequestration sites. Adv Water Resour 2012;44:20–9.
[19] Nwankwor E, Nagar AK, Reid D. Hybrid differential evolution and particle swarm optimization for optimal well placement. Comput Geosci 2013;17(2):249–68.
[20] Li B, Roche R, Miraoui A. Microgrid sizing with combined evolutionary algorithm and milp unit commitment. Appl Energy 2017;188:547–62.
[21] Fong K, Yuen S, Chow CK, Leung SW. Energy management and design of centralized air-conditioning systems through the non-revisiting strategy for heuristic optimization methods. Appl Energy 2010;87(11):3494–506.
[22] Nguyen TT, Vo DN, Truong AV. Cuckoo search algorithm for short-term hydrothermal scheduling. Appl Energy 2014;132:276–87.
[23] Wei M, Hong SH, Alam M. An iot-based energy-management platform for industrial facilities. Appl Energy 2016;164:607–19.
[24] Silva BN, Khan M, Han K. Towards sustainable smart cities: A review of trends, architectures, components, and open challenges in smart cities. Sustainable Cities Soc 2018;38:697–713.
[25] Queipo NV, Haftka RT, Shyy W, Goel T, Vaidyanathan R, Tucker PK. Surrogate-based analysis and optimization. Prog Aerosp Sci 2005;41(1):1–28.
[26] Akhtar T, Shoemaker CA. Multi objective optimization of computationally expensive multi-modal functions with rbf surrogates and multi-rule selection. J Global Optim 2016;64(1):17–32.
[27] Chen Y, Oliver DS, Zhang D. Efficient ensemble-based closed-loop production optimization. SPE J 2009;14(04):634–45.
[28] Chen B, Reynolds AC. Ensemble-based optimization of the water-alternating-gas-injection process. SPE J 2016;21(03):786–98.
[29] Zhao J, Wang J, Guo Z, Guo Y, Lin W, Lin Y. Multi-step wind speed forecasting based on numerical simulations and an optimized stochastic ensemble method. Appl Energy 2019;255:113833.
[30] Forrester AI, Keane AJ. Recent advances in surrogate-based optimization. Prog Aerosp Sci 2009;45(1–3):50–79.
[31] Razavi S, Tolson BA, Burn DH. Review of surrogate modeling in water resources. Water Resour Res 2012;48(7).
[32] Sun N-Z, Sun A. Model calibration and parameter estimation: For environmental and water resource systems. Springer; 2015.
[33] Mo S, Zabaras N, Shi X, Wu J. Deep autoregressive neural networks for high-dimensional inverse problems in groundwater contaminant source identification. Water Resour Res 2019;55(5):3856–81.
[34] Chen B, Harp DR, Lin Y, Keating EH, Pawar RJ. Geologic co_2 sequestration monitoring design: A machine learning and uncertainty quantification based approach. Appl Energy 2018;225:332–45.
[35] Dai Z, Viswanathan H, Middleton R, Pan F, Ampomah W, Yang C, et al. Co_2 accounting and risk analysis for co_2 sequestration at enhanced oil recovery sites. Environ Sci Technol 2016;50(14):7546–54.
[36] Mo S, Lu D, Shi X, Zhang G, Ye M, Wu J, et al. A taylor expansion-based adaptive design strategy for global surrogate modeling with applications in groundwater modeling. Water Resour Res 2017;53(12):10802–23.
[37] Oladyshkin S, Nowak W. Data-driven uncertainty quantification using the arbitrary polynomial chaos expansion. Reliab Eng Syst Saf 2012;106:179–90.
[38] Sun AY, Zeidouni M, Nicot J-P, Lu Z, Zhang D. Assessing leakage detectability at geologic co_2 sequestration sites using the probabilistic collocation method. Adv Water Resour 2013;56:49–60.
[39] Laloy E, Rogiers B, Vrugt JA, Mallants D, Jacques D. Efficient posterior exploration of a high-dimensional groundwater model from two-stage markov chain monte carlo simulation and polynomial chaos expansion. Water Resour Res 2013;49(5):2664–82.
[40] Liao Q, Zhang D, Tchelepi H. A two-stage adaptive stochastic collocation method on nested sparse grids for multiphase flow in randomly heterogeneous porous media. J Comput Phys 2017;330:828–45.
[41] Zhang G, Lu D, Ye M, Gunzburger M, Webster C. An adaptive sparse-grid high-order stochastic collocation method for bayesian inference in groundwater reactive transport modeling. Water Resour Res 2013;49(10):6871–92.
[42] Babaei M, Pan I. Performance comparison of several response surface surrogate models and ensemble methods for water injection optimization under uncertainty. Comput Geosci 2016;91:19–32.
[43] Zhang J, Li W, Zeng L, Wu L. An adaptive gaussian process-based method for efficient bayesian experimental design in groundwater contaminant source identification problems. Water Resour Res 2016;52(8):5971–84.
[44] Zheng Q, Zhang J, Xu W, Wu L, Zeng L. Adaptive multifidelity data assimilation for nonlinear subsurface flow problems. Water Resour Res 2019;55(1):203–17.
[45] Golzari A, Sefat MH, Jamshidi S. Development of an adaptive surrogate model for production optimization. J Pet Sci Eng 2015;133:677–88.
[46] Jang I, Oh S, Kim Y, Park C, Kang H. Well-placement optimisation using sequential artificial neural networks. Energy Explor Exploit 2018;36(3):433–49.
[47] Jeong H, Sun AY, Lee J, Min B. A learning-based data-driven forecast approach for predicting future reservoir performance. Adv Water Resour 2018;118:95–109.
[48] Zhu Y, Zabaras N, Koutsourelakis P-S, Perdikaris P. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. J Comput Phys 2019;394:56–81.

[49] Sun AY. Discovering state-parameter mappings in subsurface models using generative adversarial networks. Geophys Res Lett 2018;45(20):11–37.

[50] Zhu J-Y, Park T, Isola P, Efros AA. Unpaired image-to-image translation using cycle-consistent adversarial networks. In: Proceedings of the IEEE international conference on computer vision. 2017. p. 2223–32.

[51] Mo S, Zhu Y, Zabaras N, Shi X, Wu J. Deep convolutional encoder–decoder networks for uncertainty quantification of dynamic multiphase flow in heterogeneous media. Water Resour Res 2019;55(1):703–28.

[52] Zhong Z, Sun AY, Jeong H. Predicting co_2 plume migration in heterogeneous formations using conditional deep convolutional generative adversarial network. Water Resour Res 2019;55(7):5830–51.

[53] Zhong Z, Sun AY, Wu X. Inversion of time-lapse seismic reservoir monitoring data using cyclegan: A deep learning based approach for estimating dynamic reservoir property changes. J Geophys Res: Solid Earth 2020. e2019JB018408.

[54] Tang M, Liu Y, Durlofsky LJ. A deep-learning-based surrogate model for data assimilation in dynamic subsurface flow problems. 2019, arXiv preprint arXiv:1908.05823.

[55] Geneva N, Zabaras N. Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. J Comput Phys 2020;403:109056.

[56] Lu R, Hong SH, Zhang X. A dynamic pricing demand response algorithm for smart grid: reinforcement learning approach. Appl Energy 2018;220:220–30.

[57] Zia MF, Elbouchikhi E, Benbouzid M. Microgrids energy management systems: A critical review on methods, solutions, and prospects. Appl Energy 2018;222:1033–55.

[58] Nanduri V, Saavedra-Antolínez I. A competitive markov decision process model for the energy–water–climate change nexus. Appl Energy 2013;111:186–98.

[59] Zhang X, Vesselinov VV. Energy-water nexus: Balancing the tradeoffs between two-level decision makers. Appl Energy 2016;183:77–87.

[60] Wen Z, Durlofsky LJ, Van Roy B, Aziz K. Use of approximate dynamic programming for production optimization. In: SPE reservoir simulation symposium. Society of Petroleum Engineers; 2011.

[61] Sutton RS. Learning to predict by the methods of temporal differences. Mach Learn 1988;3(1):9–44.

[62] Van Otterlo M, Wiering M. Reinforcement learning and markov decision processes. In: Reinforcement learning. Springer; 2012, p. 3–42.

[63] Watkins CJ, Dayan P. Q-learning. Mach Learn 1992;8(3–4):279–92.

[64] Sutton RS, Barto AG. Reinforcement learning: An introduction. MIT Press; 2018.

[65] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al. Playing atari with deep reinforcement learning. 2013, arXiv preprint arXiv:1312.5602.

[66] Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. Nature 2015;518(7540):529–33.

[67] François-Lavet V, Taralla D, Ernst D, Fonteneau R. Deep reinforcement learning solutions for energy microgrids management. In: European workshop on reinforcement learning. 2016.

[68] Xiao L, Xiao X, Dai C, Pengy M, Wang L, Poor HV. Reinforcement learning-based energy trading for microgrids. 2018, arXiv preprint arXiv:1801.06285.

[69] Mocanu E, Mocanu DC, Nguyen PH, Liotta A, Webber ME, Gibescu M, et al. On-line building energy optimization using deep reinforcement learning. IEEE Trans Smart Grid 2018;10(4):3698–708.

[70] Liu T, Hu X, Li SE, Cao D. Reinforcement learning optimized look-ahead energy management of a parallel hybrid electric vehicle. IEEE/ASME Trans Mechatronics 2017;22(4):1497–507.

[71] Han X, He H, Wu J, Peng J, Li Y. Energy management based on reinforcement learning with double deep q-learning for a hybrid electric tracked vehicle. Appl Energy 2019;254:113708.

[72] Strnad FM, Barfuss W, Donges JF, Heitzig J. Deep reinforcement learning in world-earth system models to discover sustainable management strategies. Chaos 2019;29(12):123122.

[73] Internal Revenue Service. Section 45q. 2018, URL https://www.irs.gov/pub/irs-pdf/f8933.pdf. [Accessed 2 February 2020].

[74] Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, et al. Continuous control with deep reinforcement learning. 2015, arXiv preprint arXiv:1509.02971.

[75] Caruana R. Multitask learning. Mach Learn 1997;28(1):41–75.

[76] Ronneberger O, Fischer P, Brox T. U-net: Convolutional networks for biomedical image segmentation. In: International conference on medical image computing and computer-assisted intervention. Springer; 2015, p. 234–41.

[77] Goodfellow I, Bengio Y, Courville A, Bengio Y. Deep learning. MIT Press Cambridge; 2016.

[78] Ruder S. An overview of multi-task learning in deep neural networks. 2017, arXiv preprint arXiv:1706.05098.

[79] Jenkins C, Chadwick A, Hovorka SD. The state of the art in monitoring and verification–ten years on. Int J Greenh Gas Control 2015;40:312–49.

[80] Pawar RJ, Bromhal GS, Carey JW, Foxall W, Korre A, Ringrose PS, et al. Recent advances in risk assessment and risk management of geologic co_2 storage. Int J Greenh Gas Control 2015;40:292–311.

[81] Birkholzer JT, Zhou Q, Tsang C-F. Large-scale impact of co_2 storage in deep saline aquifers: A sensitivity study on pressure response in stratified systems. Int J Greenh Gas Control 2009;3(2):181–94.

[82] Sun AY, Jeong H, González-Nicolás A, Templeton TC. Metamodeling-based approach for risk assessment and cost estimation: Application to geological carbon sequestration planning. Comput Geosci 2018;113:70–80.

[83] Buscheck TA, Sun Y, Chen M, Hao Y, Wolery TJ, Bourcier WL, et al. Active co_2 reservoir management for carbon storage: Analysis of operational strategies to relieve pressure buildup and improve injectivity. Int J Greenh Gas Control 2012;6:230–45.

[84] Bielicki JM, Pollak MF, Deng H, Wilson EJ, Fitts JP, Peters CA. The leakage risk monetization model for geologic co_2 storage. Environ Sci Technol 2016;50(10):4923–31.

[85] González-Nicolás A, Cihan A, Petrusak R, Zhou Q, Trautz R, Riestenberg D, et al. Pressure management via brine extraction in geological co_2 storage: Adaptive optimization strategies under poorly characterized reservoir conditions. Int J Greenh Gas Control 2019;83:176–85.

[86] Cameron DA, Durlofsky LJ. Optimization of well placement, co_2 injection rates, and brine cycling for geological carbon sequestration. Int J Greenh Gas Control 2012;10:100–12.

[87] Remy N, Boucher A, Wu J. Applied geostatistics with SGeMS: A user's guide. Cambridge University Press; 2009.

[88] Bellman R. Dynamic programming. Science 1966;153(3731):34–7.

[89] Ferreira JM, Oliveira RTD, Assumpção M, Moreira JA, Pearce RG, Takeya MK. Correlation of seismicity and water level in the açu reservoir–an example from northeast brazil. Bull Seismol Soc Am 1995;85(5):1483–9.

[90] Shapiro SA, Huenges E, Borm G. Estimating the crust permeability from fluid-injection-induced seismic emission at the ktb site. Geophys J Int 1997;131(2):F15–8.

[91] Dinske C, Shapiro S, Häring M. Interpretation of microseismicity induced by time-dependent injection pressure. In: SEG technical program expanded abstracts 2010. Society of Exploration Geophysicists; 2010, p. 2125–9.

[92] Peaceman D. Interpretation of well-block pressures in numerical reservoir simulation. part 3 some additional well geometries. In: SPE annual technical conference and exhibition. 1987. p. 457–71.

[93] Sullivan EJ, Chu S, Stauffer PH, Middleton RS, Pawar RJ. A method and cost model for treatment of water extracted during geologic co_2 storage. Int J Greenh Gas Control 2013;12:372–81.

[94] Breunig HM, Birkholzer JT, Borgia A, Oldenburg CM, Price PN, McKone TE. Regional evaluation of brine management for geologic carbon sequestration. Int J Greenh Gas Control 2013;14:39–48.

[95] Computer Modeling Group. Cmg-gem. Calgary, Alberta Canada: 2019.

[96] Sun AY, Scanlon BR, Zhang Z, Walling D, Bhanja SN, Mukherjee A, et al. Combining physically based modeling and deep learning for fusing grace satellite data: Can we learn from mismatch? Water Resour Res 2019;55(2):1179–95.

[97] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016. p 770–8.

[98] Kingma DP, Ba J. Adam: A method for stochastic optimization. 2014, arXiv preprint arXiv:1412.6980.

[99] Wang Z, Bovik AC, Sheikh HR, Simoncelli EP. Image quality assessment: from error visibility to structural similarity. IEEE Trans Image Process 2004;13(4):600–12.

[100] Li H, Sarma P, Zhang D. A comparative study of the probabilistic-collocation and experimental-design methods for petroleum-reservoir uncertainty quantification. SPE J 2011;16(02):429–39.

[101] Zeng L, Shi L, Zhang D, Wu L. A sparse grid based bayesian method for contaminant source identification. Adv Water Resour 2012;37:1–9.