Contents lists available at ScienceDirect

Computers and Chemical Engineering

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

Deep reinforcement learning control of hydraulic fracturing

Mohammed Saad Faizan Bangi[a,b], Joseph Sang-Il Kwon[a,b,*]

[a] Artie McFerrin Department of Chemical Engineering, Texas A&M University, College Station, TX 77845, USA
[b] Texas A&M Energy Institute, Texas A&M University, College Station, TX 77845, USA

ARTICLE INFO

Article history:
Received 19 April 2021
Revised 25 June 2021
Accepted 9 August 2021
Available online 11 August 2021

Keywords:
Deep reinforcement learning
Dimensionality reduction
Transfer learning
Hydraulic fracturing

ABSTRACT

Hydraulic fracturing is a technique to extract oil and gas from shale formations, and obtaining a uniform proppant concentration along the fracture is key to its productivity. Recently, various model predictive control schemes have been proposed to achieve this objective. But such controllers require an accurate and computationally efficient model which is difficult to obtain given the complexity of the process and uncertainties in the rock formation properties. In this article, we design a model-free data-based reinforcement learning controller which learns an optimal control policy through interactions with the process. Deep reinforcement learning (DRL) controller is based on the Deep Deterministic Policy Gradient algorithm that combines Deep-Q-network with actor-critic framework. Additionally, we utilize dimensionality reduction and transfer learning to quicken the learning process. We show that the controller learns an optimal policy to obtain uniform proppant concentration despite the complex nature of the process while satisfying various input constraints.

1. Introduction

Hydraulic fracturing is performed for extraction of oil and gas from rocks that have low porosity and low permeability (Economides et al., 1998). This is achieved by first carrying out controlled explosions within the formation to create initial fracture paths. Next, a fluid called pad is injected at high pressures to extend the initial fracture paths, which is followed by injection of a fracturing fluid which consists of water, additives, and proppant at high pressures in order to further extend these fractures into the rock formation. Once pumping is stopped, these fractures close due to the natural stresses in the rock formation as the fracturing fluid leaks into the reservoir, leaving behind proppant in the fractures. The trapped proppant acts as a highly conductive medium for easier extraction of oil and gas. The proppant concentration and the fracture geometry are the two main factors that affect the efficiency of the hydraulic fracturing process. To achieve the desired values for these attributes, it is necessary to design an optimal pumping schedule. Many works have been conducted in this direction (Nolte, 1986; Gu and Desroches, 2003; Yang et al., 2017). These works consider this control problem in an open-loop formulation. Additionally, there have been efforts to design model predictive control (MPC) schemes for hydraulic fracturing processes after advances in real-time measurement techniques such as downhole pressure analysis and microseismic monitoring (Gu and Hoo, 2015; Siddhamshetty et al., 2018). However, an MPC system requires a model which is difficult to build given the various uncertainties in the rock formation and is an ongoing research area (Narasingam et al., 2017; Narasingam and Kwon, 2017; Narasingam et al., 2018; Sidhu et al., 2018; Narasingam and Kwon, 2018; Bangi et al., 2019; Bangi and Kwon, 2020). Additionally, the performance of a MPC system depends on its tuning parameters, and the accuracy of the process model. It is a common practice to continuously monitor the controller's performance and begin a model re-identification process or re-tune the parameters of the controller in case the controller performance degrades, which is time-consuming and is resource intensive. To summarize, there are two challenges with designing a model-based controller for hydraulic fracturing process: (a) Its first principles model involves highly-coupled partial differential equations (PDEs) with moving boundaries which are computationally expensive to solve at each sampling time, and (b) Regular re-tuning of controller and model parameters.

To address the limitations of a model-based controller, we design a model-free data-based controller suitable for nonlinear chemical processes, specifically for hydraulic fracturing, by combining concepts from reinforcement learning (RL) and deep learning (DL). RL is a branch of machine learning which deals with solving complex decision making problems, and it involves an agent which interacts with the environment (process) to derive an optimal policy in order to reach the desired target (Sutton and Barto, 1998; Suglyama, 2015). A schematic of the RL process is shown in Fig. 1. RL has delivered tremendous success in computer games

* Corresponding author at: Artie McFerrin Department of Chemical Engineering, Texas A&M University, College Station, TX 77845, USA.
E-mail address: kwonx075@tamu.edu (J.S.-I. Kwon).

https://doi.org/10.1016/j.compchemeng.2021.107489
0098-1354/© 2021 Elsevier Ltd. All rights reserved.

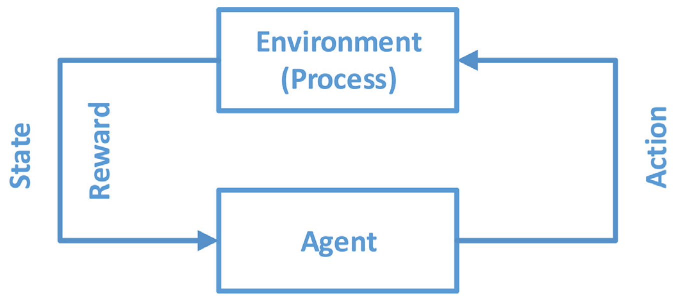

Fig. 1. A schematic of a standard RL algorithm.

(Mnih et al., 2013; 2015), board games (Tesauro, 1995; Silver et al., 2016), robotics (Lehnert and Precup, 2015; Lillicrap et al., 2015), etc. Furthermore, developments in DL have enabled its combination with RL, which has achieved huge success such as AlphaGo defeating human champions in the game Go (Silver et al., 2017). Another RL algorithm called deep-Q-network (DQN) (Mnih et al., 2013) has achieved the human level performance in Atari video games. Despite its success in other domains, application of RL to process control has been very limited (Brujeni et al., 2010; Badgwell et al., 2018; Shin et al., 2019; Kim et al., 2020; Yoo et al., 2021; Pistikopoulos et al., 2021) even though many process control problems can be defined as Markov decision processes (MDPs) (Lee and Lee, 2006). The challenge is the lack of RL algorithms that can efficiently deal with continuous state and action spaces, which is usually the case with process control applications. It is possible to discretize and use Dynamic programming (DP) to obtain a solution to the RL problem in such cases, but there is an exponential growth in computational complexity with respect to the number of states and actions which is referred to as the curse of dimensionality (Bertsekas, 2005). Approximate dynamic programming (ADP) was proposed, which utilizes simulations and function approximators to overcome this challenge (i.e., curse of dimensionality). In addition to ADP, many other RL algorithms have been proposed for continuous-time nonlinear systems (Vrabie and Lewis, 2009; Vamvoudakis and Lewis, 2010). But these algorithms require high-accuracy models that are either available or that can be identified using system identification methods. Given the limitations of model-based RL methods, several data-based RL methods have been proposed, which come with their own limitations (Lee and Lee, 2006). The recent success of combining RL with DL has led to a resurgence of interest in data-based RL for continuous state and action spaces.

In this article, we design a deep reinforcement learning (DRL) controller which is based on actor-critic approach and temporal difference (TD) learning (Spielberg et al., 2019; Ma et al., 2019). The actor (controller) interacts with the process iteratively and implements control actions that give maximum rewards. The critic, as the name suggests, evaluates the control policy followed by the actor and modifies it to achieve the optimal policy. In the DRL controller, the actor and the critic are both represented by two deep neural networks (DNNs) in order to effectively generalize them for continuous-time variables. The DRL controller also utilizes deep deterministic policy gradient (DDPG) algorithm (Silver et al., 2014) which is usually used for continuous action spaces. We also utilized concepts such as replay memory (RM), target networks and constrained action spaces to make learning more suitable for complex systems like hydraulic fracturing (Lillicrap et al., 2015). Replay memory (RM) is used to break the temporal correlation between two consecutive experiences obtained from the process. Without the RM, the DRL controller will learn from temporally correlated tuples of online data which will lead to inefficient learning. Taking random experiences from the RM breaks this correlation, and hence, speeds up the learning process. Target networks are separate networks used to stabilize the learning process by providing stable targets to the actor and the critic networks. At the beginning of the learning process, these networks are initialized as the copies of the actor and the critic networks, and during the learning process their parameters are constrained to change slowly, which enhances their stability. Action spaces are to RL what control input spaces are to process control. The relationship between the controller's action in the DRL framework and the control input from the process control perspective is direct. The DRL controller traverses the action space to obtain an optimal control policy for the defined control problem. In order to enforce constraints on the action space, we included an action-based reward function in the overall reward calculation. Additionally, we utilized Principal Component Analysis to reduce the dimension of the RL state before using it in the learning of the actor and the critic. Transforming the RL state to the reduced PCA space helped in faster learning of the control policy. Moreover, we utilized transfer learning wherein the DRL controller learns offline using a data-based reduced-order model (ROM) first and then learns online from the process. We summarize the novelty of our framework as follows: (1) We propose to use a data-based model-free DRL controller for control of a hydraulic fracturing process which is a complex moving-boundary system and is difficult to develop a highly accurate model; (2) We propose to utilize PCA in the DRL control framework to reduce the dimension of the RL state; (3) We propose a cumulative reward function to handle various process constraints of the hydraulic fracturing process; and (4) We propose the use of transfer learning for the DRL controller to reduce the online learning time.

The remainder of this text is organized as follows: Section 2 provides a brief background on RL and actor-critic approach. Section 3 provides the DRL framework for data-based control of discrete-time nonlinear processes. Section 4 presents the first principles model of hydraulic fracturing. Section 5 presents the design of a DRL controller for a hydraulic fracturing process including a ROM of a hydraulic fracturing process utilized in this work. Section 6 presents the results and analysis of the designed DRL conroller. Section 7 presents a few conclusions related to the closed-loop simulation results using the controller.

2. Background

2.1. Reinforcement learning

In RL framework, an agent interacts with the environment E at its current state s_t by implementing control action a_t and receiving a reward of r_t. The cumulative future discounted reward is given by

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{1}$$

where the discount factor is $0 < \gamma \leq 1$. The expected return Q after implementing action a_t on state s_t is defined as

$$Q(s,a) = \mathbb{E}[R_t \big| s_t = s, a_t = a] \tag{2}$$

The optimal action-value is defined as the maximum expected return after implementing action a_t on state s_t, and is given as

$$Q^*(s,a) = \max \mathbb{E}[R_t \big| s_t = s, a_t = a] \tag{3}$$

The optimal action-value Q^* can be calculated by iteratively solving the Bellman equation which is shown below:

$$Q^*(s,a) = \max \mathbb{E}[r + \gamma \max Q^*(s', a') \big| s, a] \tag{4}$$

where s' and a' are the subsequent state and action, respectively. Overall, the solution to a RL problem is obtained by implementing control actions on the environment and by learning an optimal control policy by receiving data from it.

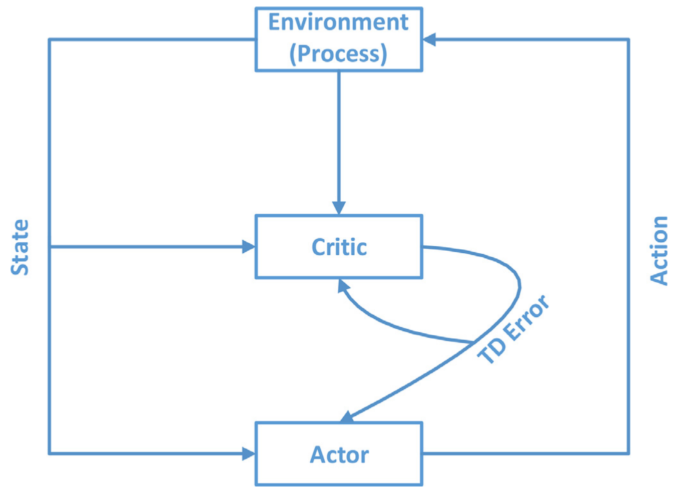

Fig. 2. A schematic of the actor-critic framework.

2.2. Actor-critic framework

The actor-critic framework is a widely used RL algorithm as it can be generalized to systems with continuous spaces. It has two components, i.e., the actor and the critic. The actor finds an optimal policy, and the job of the critic is to evaluate the policy calculated by the actor. With each iteration of learning, the actor is updated by the policy gradient theorem by adjusting the parameters of the policy function which can be represented using function approximators like neural networks. The critic estimates the action-value function Q which can be represented using a neural network and its parameters are updated using stochastic gradient descent (SGD) method such that the Bellman optimality condition is reached. A schematic diagram of the actor-critic algorithm is shown in Fig. 2.

3. Deep reinforcement learning (DRL) controller

The DRL controller (Spielberg et al., 2019) is a model-free controller based on the actor-critic framework. It utilizes two DNNs; one to generalize the actor in the continuous state space, and the other to generalize the critic in the continuous state and action spaces.

3.1. States and actions

Let u_t and y_t be the input applied on the system at time t and output from the system, respectively. The RL action a_t is same as the input u_t as understood from a control perspective. Therefore, $a_t = u_t$. But the relationship between RL state $s_t \in S$ and the state of the system is different as the RL state s_t should contain information about the system deemed necessary for the successful working of the RL controller. An example of a RL state definition can contain past outputs and the current deviation from the set-point as shown below:

$$s_t := [y_t, y_{t-1}, \dots y_{t-d_y}, (y_t - y_{sp})] \tag{5}$$

where d_y is the number of past outputs to be included in the DRL controller. The state of the system is initialized as y_0 in every episode, and y_{sp} is the defined set-point for the controller. During the learning process, in every episode, and at every time step, the RL state s_t is updated as we obtain measurements from the system. Additionally, the RL controller computes a deterministic control policy μ for each state $s_t \in S$ such that $\mu : S \to A$, where $a_t \in A$.

3.2. Reward functions

The goal of the RL controller is to reach the set-point using an optimal policy μ that maximizes the aggregate reward that the agent gains from the system. Two examples of reward functions $r : S \times A \times S \to \Re$ are shown below:

$$r(s_t, a_t, s_{t+1}) = \begin{cases} c & \text{if } |y_{i,t} - y_{i,sp}| \le \epsilon \\ -\sum_{i=1}^{n} |y_{i,t} - y_{i,sp}| & \text{otherwise} \end{cases} \tag{6}$$

$$r(s_t, a_t, s_{t+1}) = \begin{cases} 0 & \text{if } |y_{i,t} - y_{i,sp}| > |y_{i,t+1} - y_{i,sp}| \quad \forall i \in \{1, \dots n\} \\ -1 & \text{otherwise} \end{cases} \tag{7}$$

where $y_{i,t}$ is the i^{th} output, $y_{i,sp}$ is the set-point for the i^{th} output, $y_{i,t+1}$ is the subsequent i^{th} output, and ϵ is the user defined tolerance. As per Eq. (6), the agent receives a reward of $c > 0$ only if all the outputs are within the tolerance limit. The reward function of Eq. (6) leads to faster tracking but has the disadvantage of obtaining an aggressive control strategy. On the other hand, Eq. (7), a polar reward function, assigns a reward of 0 if the deviation from the set-point is monotonically decreasing for all n outputs at each sampling time, and assigns a reward of -1 otherwise. This reward function incentivizes gradual improvement towards the set-point, which results in a smoother tracking performance and a less aggressive control strategy.

3.3. DNNs as function approximators

The DRL controller utilizes two DNNs to approximate the policy and the Q functions. The actor utilizes a DNN with parameters W_a to generalize the policy function over the continuous action space such that given the state s_t, it produces control actions $a_t = \mu(s_t, W_a)$. Likewise, the critic utilizes a DNN with parameters W_c to generalize the Q function such that given the state s_t and action a_t, it produces Q values as network outputs, i.e., $Q^\mu(s_t, a_t, W_c)$.

3.4. DRL training

The objective of DRL controller training is to calculate the parameter values W_a and W_c such that, once the networks are trained, the actor network can be used to obtain the optimal control actions for any given state.

The DDPG algorithm (Lillicrap et al., 2015) was proposed in order to improve the trainability of the existing policy gradient theorem. DDPG borrows a concept of mini-batch training from DQN method which involves learning in mini-batches rather than learning directly from online data. DQN utilizes a RM to store historical data in the form of $[s^{(i)}, a^{(i)}, r^{(i)}, s'^{(i+1)}]$, and during the training process, mini-batches of data are sampled from the RM which are used to update the parameters of the DNNs that represent the actor and the critic. Using mini-batches of size M from the RM helps in stabilizing the training process, and randomly sampling tuples from the RM helps in breaking the temporal correlations between the samples. At each sampling time, the latest tuple is stored in the RM, and an old tuple is discarded from the RM in order to keep its size constant. The parameters of the critic are updated using SGD and samples from the RM so that the TD error is minimized, and the update equation is given as follows:

$$W_c \leftarrow W_c + \frac{\alpha_c}{M} \sum_{i=1}^{M} \Big[(\tilde{y}^{(i)} - Q(s^{(i)}, \mu(s^{(i)}, W_a), W_c)) * \nabla_{W_c} Q^\mu(s^{(i)}, \mu(s^{(i)}, W_a), W_c) \Big] \tag{8}$$

where

$$\tilde{y}^{(i)} \leftarrow r^{(i)} + \gamma Q^\mu(s'^{(i)}, \mu(s'^{(i)}, W_a), W_c), \forall i = 1, \dots, M \tag{9}$$

The parameters of the actor are updated in a batch-wise manner using M samples from the RM, following the DPG theorem, and the update equation is as follows:

$$W_a \leftarrow W_a + \frac{\alpha_a}{M}\sum_{i=1}^{M}[\nabla_{W_a}\mu(s^{(i)},W_a)\nabla_a Q^{\mu}(s^{(i)},a,W_c)\big|_{a=a^{(i)}}] \tag{10}$$

To further stabilize the learning process, target networks are used to provide stable targets to the critic. In Eq. (8), the parameters W_c are utilized to calculate $\tilde{y}$ in Eq. (9), which is a target for the critic network. If the target updates are erratic then the parameter updates are also erratic, which may cause the network to diverge. Hence, separate neural networks called target networks are utilized to provide stable targets to be used in Eq. (9). Suppose the parameters of the target networks are W'_a and W'_c, then, Eq. (9) changes as follows:

$$\tilde{y}^{(i)} \leftarrow r^{(i)} + \gamma Q^{\mu}(s'^{(i)}, \mu(s'^{(i)}, W'_a), W'_c), \forall i = 1,\ldots,M \tag{11}$$

where

$$W'_c \leftarrow \tau W_c + (1-\tau)W'_c \tag{12}$$

$$W'_a \leftarrow \tau W_a + (1-\tau)W'_a \tag{13}$$

where τ is the target network update rate. As per Eqs. (12) and (13), the parameters of the target networks slowly track the critic and the actor network, which ensures that the targets are changed slowly, thereby, stabilizing the learning process.

In practical applications, even though the action space is usually continuous, it is bounded too. This is because the control action is usually a representative of physical quantities like flow rate, pressure, and these physical quantities have limits. In order to ensure that the control actions are predicted within the set control limits, it is necessary to bound the actor network. If the actor network is not bounded, then the critic will continue to push the actor to predict control actions outside the control limits. To avoid this scenario, gradient clipping is used to bound the output layer of the actor network. This is done by multiplying the gradient used in the update step, Eq. (10), with an appropriate factor. Suppose the action space is bounded in the interval $[a_L, a_H]$ such that $a_L < a_H$, then the gradient clipping is done as follows:

$$\nabla_a Q^{\mu}(s,a,w) \leftarrow \nabla_a Q^{\mu}(s,a,w) * \begin{cases} (a_H - a)/(a_H - a_L) & \text{if } \nabla_a Q^{\mu}(s,a,w) \text{ increases } a \\ (a - a_L)/(a_H - a_L) & \text{otherwise} \end{cases} \tag{14}$$

With gradient clipping in place, the control actions will saturate at the upper bound a_H when the critic continually recommends increasing the control actions. On the other hand, the gradient clipping will ensure that the control actions do not decrease beyond the lower limit a_L when the critic continually recommends decreasing the control actions. Combining the above described concepts, the algorithm for training the DRL controller is provided in Algorithm 1.

As presented in the algorithm, RL training is started by first initializing the parameters of the actor and the critic network. Then the target networks are initialized as outlined in Line 3 of the algorithm. In Line 4, RM is initialized with tuples of historical data. Then, for each episode, a set-point is defined (Line 5). Now for each time step of the episode, the RL state s is defined (Line 8), a control action a_t is obtained from actor, which is implemented on the system to obtain a new output y_{t+1} and reward r_t (Lines 9 and 10), and the latest tuple (s, a_t, s', r_t) is stored in the RM (Line 12). Now, M samples are uniformly drawn from the RM to update the actor network, the critic network, and the target networks (Lines 13–23). Lines 8–23 are repeated until the end of the episode.

Algorithm 1 DRL controller training.

```
1:  Output: Optimal control policy $\mu(s, W_a)$
2:  Initialize $W_a$, $W_c$
3:  Initialize $W'_a \leftarrow W_a$, $W'_c \leftarrow W_c$
4:  Initialize RM with historical data
5:  for each episode do
6:    Set the set-point for episode as $y_{sp}$
7:    for each step $t = 0, 1, \ldots T-1$ do
8:      Set $s \leftarrow [y_t, y_{t-1}, \ldots y_{t-d_y}, (y_t - y_{sp})]$
9:      Set $a_t \leftarrow \mu(s, W_a)$
10:     Implement $a_t$ and obtain $y_{t+1}$ and $r_t$
11:     Set $s' \leftarrow [y_t, y_{t-1}, \ldots y_{t-d_y}, (y_t - y_{sp})]$
12:     Store tuple $(s, a_t, s', r_t)$ in RM
13:     Obtain $M$ tuples from RM
14:     for $i = 1, \ldots M$ do
15:       $\tilde{y}^{(i)} \leftarrow r^{(i)} + \gamma Q^{\mu}(s'^{(i)}, \mu(s'^{(i)}, W'_a), W'_c)$
16:     end
17:     Update $W_c$ using Eq. (8)
18:     for $i = 1, \ldots M$ do
19:       Calculate $\nabla_a Q^{\mu}(s^{(i)}, a, W_c)\big|_{a=\mu(s^{(i)}, W_a)}$
20:       Clip gradient using Eq. (14)
21:     end
22:     Update $W_a$ using Eq. (10)
23:     Update $W'_c$ and $W'_a$ using Eq. (12) and (13), respectively
24:   end
25: end
```

4. Hydraulic fracturing process

For simulation purposes, we utilize a first-principles model of hydraulic fracturing which comprises of 3 subprocesses: (a) Fracture propagation; (b) Proppant transport; and (c) Proppant bank formation. They are briefly discussed in the following subsections.

4.1. Fracture propagation

We assume the fracture propagation follows the Perkins, Kern and Nordgren (PKN) model (Perkins and Kern, 1961; Nordgren, 1972). The assumptions included are as follows: (a) the fracture length is assumed to be greater than the fracture width, and hence, the fluid pressure in the vertical direction remains constant; (b) the fracture is confined to a single layer because of large stresses in the rock layers above and below the fractures; and (c) the fracturing fluid is assumed to be incompressible, and the rock properties such as Young's modulus and Poisson's ratio are also assumed to be constant. Following the assumptions, the fracture geometry and fracture cross-sectional area take an elliptical and rectangular shape, respectively. Lubrication theory is utilized to explain the fluid momentum inside the fracture which relates the fluid flow rate, q_z, in the horizontal direction to the pressure gradient, $-\frac{\partial P}{\partial z}\hat{z}$, as follows:

$$q_z = -\frac{\pi H W^3}{64\mu}\frac{\partial P}{\partial z} \tag{15}$$

where H is the fracture height, W is the fracture width, and μ is the fracturing fluid viscosity. The maximum width of the fracture can be calculated using the net pressure exerted by the fracturing fluids as follows:

$$W = \frac{2PH(1-\nu^2)}{E} \tag{16}$$

where ν is the Poisson's ratio, and E is the Young's modulus of the rock formation. The local mass conservation of an incompressible fluid leads to the continuity equation which is:

$$\frac{\partial A}{\partial t} + \frac{\partial q_z}{\partial z} + HU = 0 \tag{17}$$

where A is the cross-sectional area of the fracture (calculated as $A = \pi WH/4$), U is the fluid leak-off rate per unit height, t is the time elapsed since the beginning of pumping, and z is the spatial coordinate in the horizontal direction. Plugging Eqs. (15) and (16) into Eq. (17) results in the following partial differential equation of W:

$$\frac{\pi H}{4}\frac{\partial W}{\partial t} - \frac{\pi E}{128\mu(1-\nu^2)}\left[3W^2\left(\frac{\partial W}{\partial z}\right)^2 + W^3\frac{\partial^2 W}{\partial z^2}\right] + HU = 0 \tag{18}$$

The two boundary conditions and an initial condition for the process are shown below (Gu and Hoo, 2015):

$$q_z(0,t) = Q_0 \qquad W(L(t),t) = 0, \tag{19}$$

$$W(z,0) = 0 \tag{20}$$

where Q_0 is the fracturing fluid rate at the wellbore, and $L(t)$ is the length of the fracture varying with time.

4.2. Proppant transport

In this model, it is assumed that the proppant moves at a rate equal to the superficial velocity of the fracturing fluid in the horizontal direction, and with the settling velocity relative to the fracturing fluid in the vertical direction. The other assumptions utilized are: (1) the size of proppant particle is large enough that its diffusive flux is neglected and only its convective flux is considered; (2) the proppant particle-particle interactions are neglected while only drag and gravity effects are considered; and (3) the uniform size of proppant particles is considered. Utilizing the assumptions with regards to proppant transport as explained in (Siddhamshetty, 2020), the advection of proppant can be defined in the following manner:

$$\frac{\partial(WC)}{\partial t} + \frac{\partial}{\partial z}(WCV_p) = 0 \tag{21}$$

$$C(0,t) = C_0(t) \quad \text{and} \quad C(z,0) = 0 \tag{22}$$

where $C(z,t)$ is the proppant concentration at distance z in the horizontal direction from the wellbore at time t, and $C_0(t)$ is the proppant concentration at the wellbore varied with time. V_p is the net velocity of proppant and is calculated using the following equation (Adachi et al., 2007):

$$V_p = V - (1-C)V_s \tag{23}$$

where V is the superficial fluid velocity, and V_s is the gravitational settling velocity which is calculated using the following equation (Daneshy, 1978):

$$V_s = \frac{(1-C)^2(\rho_{sd} - \rho_f)gd^2}{10^{1.82C}18\mu} \tag{24}$$

where ρ_f is the pure fluid density, ρ_{sd} is the proppant particle density, g is the gravitational constant, d is the proppant particle diameter, and μ is the fracture fluid viscosity which is dependent on the proppant concentration, C (Barree and Conway, 1995):

$$\mu(C) = \mu_0\left(1 - \frac{C}{C_{\max}}\right)^{-\alpha} \tag{25}$$

where μ_0 is the pure fluid viscosity, α is an exponent in the range of 1.2 to 1.8, and $C_{\max}$ is the maximum theoretical concentration calculated using the equation $C_{\max} = (1-\phi)\rho_{sd}$ where ϕ is the proppant bank porosity.

4.3. Proppant bank formation

When proppant settles down in a fracture, a proppant bank is formed whose height δ can be calculated using the following equations (Gu and Hoo, 2014; Novotny, 1977):

$$\frac{d(\delta W)}{dt} = \frac{CV_sW}{(1-\phi)} \tag{26}$$

$$\delta(z,0) = 0 \tag{27}$$

where Eq. (27) is the initial condition for Eq. (26). A point to note here is that this model comprising of Eqs. (15)-(27) was used to generate data for the RM and also as a simulation environment in designing the DRL controller.

5. Design of DRL controller for hydraulic fracturing

In our work, we design a DRL controller with the objective of obtaining an optimal control policy that leads to a uniform proppant concentration profile at the end of the pumping process.

5.1. RL state definition and dimensionality reduction

In order to obtain a uniform concentration profile at the end of the pumping process, we define a set-point C_{sp} for the concentration at 6 different locations along the length of the fracture and use these concentration variables as the outputs to be controlled. Let these be variables be represented as C_i where $i = 1, \ldots, 6$. Another important parameter in the hydraulic fracturing process is the total amount of proppant injected into the fracture A_t by time t. The total amount to be injected in the entire pumping process is prefixed, and this criteria has to be met by the DRL controller at the end of the pumping process. In order to ensure that this constraint is satisfied, we include A_t in the output definition with the prefixed value as its set-point A_{sp}. Therefore, the output vector at time t is as follows:

$$y_t = [C_{1t} \quad C_{2t} \quad C_{3t} \quad C_{4t} \quad C_{5t} \quad C_{6t} \quad A_t]^T \tag{28}$$

In our work, unlike Eq. (5), we propose to use a simpler definition of RL state as follows:

$$s_t := [y_t] \tag{29}$$

In order to quicken the learning process, we propose to reduce the dimension of the RL state by using Principal Component Analysis (PCA) on the concentration vector, i.e., $[C_{1t}\, C_{2t}\, C_{3t}\, C_{4t}\, C_{5t}\, C_{6t}]$. We use historical data to obtain the Principal Components (PCs) and select the most dominant one to calculate the corresponding PCA score. Therefore, the reduced output vector at time t is as follows:

$$y_{r,t} = [C_{r,t}\; A_t]^T \tag{30}$$

Using the reduced output vector, the RL state definition is changed as follows:

$$s_t := [y_{r,t}] \tag{31}$$

5.2. Action

Action a_t is the concentration of proppant injected into the fracture at time t. The unit of a_t is in terms of ppga which is a concentration unit used in petroleum engineering that refers to one pound of proppant added to a gallon of water. The range of a_t in ppga is [0 10] but is normalized to [0 1]. The DRL controller implements a control action every 100 s, and we assume that the measurements of the concentrations at the selected 6 locations will be available by then.

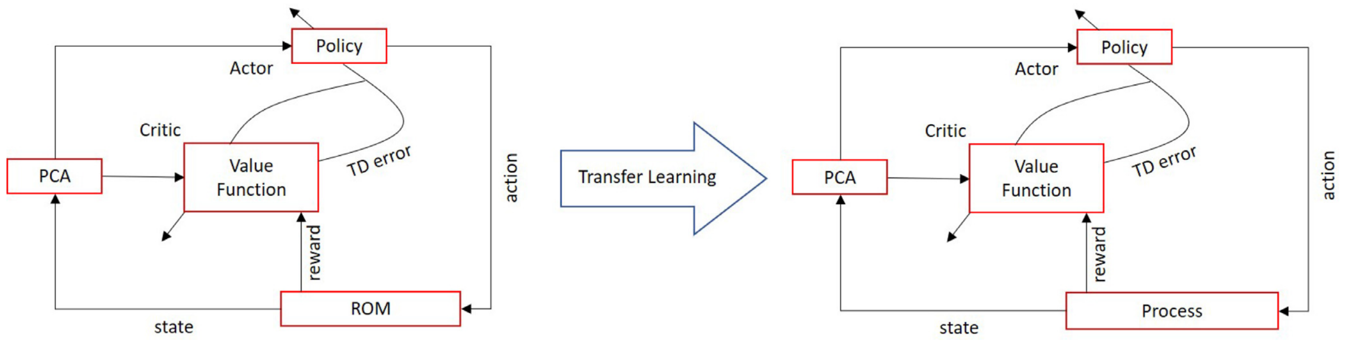

Fig. 3. A schematic of the proposed learning strategy.

5.3. Reward function

At each time step, the controller receives a reward r_t from the process whose aggregate value is to be maximized by the controller. Since the objective of the controller in set-point tracking is to minimize the tracking error, we incorporate this in the reward function r_1 as follows:

$$r_1(t) = \begin{cases} 1 - r_{tracking}(t) & \text{if } t < t_{end} \\ 1 & \text{if } |y_{i,end} - y_{i,sp}| \le \epsilon \quad \forall i \in \{1, 2 \dots 7\} \\ penalty * [1 - r_{tracking}(t)] & \text{if } |y_{i,end} - y_{i,sp}| > \epsilon \quad \forall i \in \{1, 2 \dots 7\} \end{cases} \tag{32}$$

where $r_{tracking}$ is the squared euclidean distance of the elements of the output vector from their respective set-points, and is defined as:

$$r_{tracking}(t) = \omega_1 * \sum_{i=1}^{6} \frac{|C_{it} - C_{sp}|^2}{6} + \omega_2 * |A_t - A_{sp}|^2 \tag{33}$$

where since the variables C_i and A are normalized between 0 and 1, the range of r_1 is [0 1]. The weights ω_1 and ω_2 indicate the significance of the variables C_i and A, respectively, to the reward function r_1.

Additionally, in our work, we do not use the gradient clipping technique to ensure that the control actions are within the feasible range $[a_L \; a_H]$ as specified in Algorithm 1. Instead we use a reward function r_2 to achieve this goal. The reward function r_2 is defined as follows:

$$r_2(t) = \begin{cases} 0 & \text{if } a(t) \in [0 \; 1] \\ -\frac{a(t)*(a(t)-1)}{0.25} & \text{otherwise} \end{cases} \tag{34}$$

Since the range of $[a_L \; a_H]$ is normalized to [0 1], we use the product term $a(t) * (a(t) - 1)$ in r_2 to penalize the controller if the control actions predicted are outside the feasible range.

Also, based on the knowledge about the process from the literature, we know that the optimal solution should follow a monotonically increasing profile (Nolte, 1986). But Nolte's power law pumping schedule is practically infeasible to implement. Hence, a stepwise increasing profile which is practical to implement is desired from the DRL controller. This can be obtained by enforcing a constraint on the amount of change in two consecutive inputs. We include this information in the form of a reward function r_3, which is defined as follows:

$$r_3(t) = \begin{cases} 0 & \text{if } \Delta a(t) \in [0 \; 0.3] \\ -\frac{\Delta a(t)*(\Delta a(t)-0.3)}{0.0225} & \text{otherwise} \end{cases} \tag{35}$$

where $\Delta a(t) = a(t) - a(t-1)$. The reward function r_3 ensures that the controller learns a control policy which is monotonically increasing with an increment less than 3 ppga/stage. The upper limit of 3 ppga/stage when normalized is equal to 0.3.

Therefore, considering all the constraints, the net reward r_t that the controller receives at time t is the cumulative sum of the rewards obtained using Eqs. (32)–(35) as shown below:

$$r_t = r_1(t) + r_2(t) + r_3(t) \tag{36}$$

Considering the reduced RL state definition and the tailor-made reward function, the algorithm for training the DRL controller for the hydraulic fracturing process is shown below in Algorithm 2.

Remark 1. Theoretically, the DRL controller is able to obtain an optimal control policy without any process knowledge. Practically, incorporating any process knowledge in the DRL controller, as in Eq. (35), helps in reducing training time, and achieving faster convergence to a solution. This is because we are restricting the input space for the DRL controller to obtain a solution.

5.4. DRL controller learning

In our work, we use two stages of learning. In the first stage, the controller learns using a reduced-order-model (ROM) of the process and the learning process is terminated when the controller learns a sub-optimal policy. In the second stage of learning, transfer learning is used wherein the sub-optimal controller parameters are used as initial values, and the controller continues to learn by interacting with the process directly. A schematic of our learning strategy for the DRL controller is shown in Fig. 3.

5.5. DRL controller hyperparameters

The actor and the critic are each represented using a DNN. Each of the DNNs has two hidden layers, where the first hidden layer consists of 400 neurons and the second hidden layer consists of 300 neurons. A large number of neurons are utilized because these networks have to represent complex policy and value functions for continuous state and action spaces. Rectified linear unit and linear activation functions were used in the hidden layers and output layer, respectively. For the first stage of learning, the parameters of the actor and the critic network, i.e., the weights and the biases, were initialized using Xavier initialization as this helps in maintaining constant variance in the outputs from the neurons across every layer. This constant variance helps prevent vanishing or exploding gradients. Also, batch normalization (Ioffe and Szegedy, 2015) was used in the hidden layers in order to ensure that the training is effective as different variables could have different units and could vary on different scales. Finally, we used Adam optimizer (Kingma and Ba, 2015) in order to train the networks as it is computationally efficient and well-suited for our optimization problem with a large number of parameters. Adam optimizer combines the advantages of AdaGrad and RMSProp, two popular stochastic optimization methods, by computing adaptive learning rates for each parameter using estimates of the first and the second moments of the first-order gradients (Kingma and Ba, 2015). The hyperparameters used in our work are given in Table 1.

Fig. 4. Training input for building ROMs.

Table 1
Hyperparameter values for the DRL controller.

Actor learning rate	0.01
Critic learning rate	0.01
Target network update rate	0.001
Minibatch size	16
RM size	4110
Reward discount factor	0.9
Control action limits	[0, 1]

5.6. ROM for hydraulic fracturing

In our work, we developed a ROM by applying the multivariate output error state space (MOESP) algorithm (Van Overschee and De Moor, 1996) to regress a linear time-invariant state-space model of the hydraulic fracturing process, which is presented in the following form:

$$x(t_{k+1}) = Ax(t_k) + Bu(t_k) \tag{37}$$

$$y(t_k) = Hx(t_k) \tag{38}$$

where $y(t_k)$ is the concentration of proppant at 25 different locations, i.e., $[cc_w, cc_9, \ldots, cc_{216}]$, where cc_{z_i} is the concentration at location z_i with $z_i - z_{i-1} = 0.5$ m, and $u(t_k) = [cc_w(t_k), \ldots cc_0(t_k - \theta_{216})]$ is the input to the state space model where cc_w is the concentration at the wellbore, and θ_{z_i} is the input time-delay due to the time required for the proppant to travel from the wellbore to location z_i. In order to obtain the ROM, we obtained training data from the first principles model presented in the previous section by giving input as shown in Fig. 4.

The locations that are of interest for the purpose of designing the DRL controller are $z = 36, 72, 108, 144, 172, 216$ which are included in the output of the ROM. Fig. 5 shows the comparison between the predictions from the ROM and from the first principles model at the wellbore and at these 6 locations. It can be observed that the predictions of the proppant concentration at these locations across the fracture are fairly accurate.

6. DRL controller results

6.1. Initializing the learning process

To briefly summarize the hydraulic fracturing process, a fracturing fluid along with proppant is injected at high pressures

Fig. 5. Output predictions from the ROMs at the wellbore and 6 other locations.

Fig. 6. Net reward gained in each episode during the ROM learning.

Fig. 7. Net reward gained in each episode during the entire learning process of the DRL controller. Please note that the learning curve of the second stage continues from Fig. 6, and corresponds to the episodes between 603 and 724 in this figure.

Table 2
Hyperparameter values used in rewards calculation.

ω_1	0.1
ω_2	0.9
penalty	0.1

to extend the fracture and to deposit proppant inside the fracture which acts as an artificial medium for the easier extraction of oil and gas. The objective of the DRL controller is to learn a control policy with injected proppant concentration as the manipulated variable and the concentration at 6 locations, i.e., $z = 36, 72, 108, 144, 172, 216$ from the wellbore as the controlled outputs with the objective of obtaining a uniform concentration of 10 ppga at these locations. The limits for the control actions are 0 and 10 ppga. The total fracking time considered is 1220 s. During the first 220 s of the injection process, called pad time, no proppant is injected, and thereafter, proppant injection begins. The pad time of 220 s was fixed in order to prevent premature termination of the hydraulic fracturing process, due to tip screen-out. So the DRL controller learns a control policy from 220 s onwards. The injection process occurs over 10 stages with a constant fracturing fluid rate of $Q = 0.03$ m^3/s in order to reach a fracture length of 135 m.

For the construction of the RM, we generated simulation data by implementing 411 input profiles on the first-principles model and collected 4110 snapshots of input-output data. PCA was used on the RM data to calculate the dominant PC in order to reduce the RL state during the learning process. In each episode of learning, the set-point for C_i is 10 ppga and for A is 24,000 kgs but after normalization these values change to 1. The tolerance for C_i is 0.08 and for A is 0.0417 after normalization. These tolerance values should be selected carefully as stricter tolerances will require longer training times, and laxer tolerances will result in poor performance of the DRL controller. The parameters used in the rewards calculation are tabulated in Table 2.

6.2. First stage of learning

In the first stage of the learning process, the parameters of the DRL controller are initialized as described in the previous section, and the learning process using the ROM was started. The DRL controller implements the control action as predicted by the actor and implements it on the ROM to obtain the outputs. The rewards are calculated using Eq. (36), and the tuple (s, a, s', r) is stored in the RM. Then $M = 16$ tuples are randomly selected from the RM and used to update the actor, critic, and the target networks as presented in Algorithm 2. The learning process is terminated when the following criteria are satisfied: (a) the net reward gained in an episode is 0.75 times the theoretical maximum (obtained from literature); and (b) the total amount of proppant injected is within the tolerance.

The DRL controller reaches the above-mentioned criterion at 603 episodes and in order to track the learning process, the net reward gained in each episode is plotted as shown in Fig. 6. Initially, since the weights and the biases of the DNNs are randomly ini-

Algorithm 2 DRL algorithm for hydraulic fracturing.

```
1:  **Output**: Optimal control policy $\mu(s, W_a)$
2:  Initialize $W_a$, $W_c$
3:  Initialize $W_a' \leftarrow W_a$, $W_c' \leftarrow W_c$
4:  Initialize RM with historical data
5:  Calculate the dominant PC
6:  Set the set-point for DRL controller as $y_{sp}$
7:  **for** each episode **do**
8:    **for** each step $t = 0, 1, \ldots T-1$ **do**
9:      Calculate $y_{r,t}$ using $y_t$ and $PC$
10:     Set $s \leftarrow [y_{r,t}]$
11:     Set $a_t \leftarrow \mu(s, W_a)$
12:     Implement $a_t$ and obtain $y_{t+1}$
13:     Calculate $y_{r,t+1}$ using $y_{t+1}$ and $PC$
14:     Set $s' \leftarrow [y_{r,t+1}]$
15:     Calculate $r_t$ using Eq. (36)
16:     Store tuple $(s, a_t, s', r_t)$ in RM
17:     Obtain $M$ tuples from RM
18:     **for** $i = 1, \ldots M$ **do**
19:       $\tilde{y}^{(i)} \leftarrow r^{(i)} + \gamma Q^{\mu}(s'^{(i)}, \mu(s'^{(i)}, W_a'), W_c')$
20:     **end**
21:     Update $W_c$ using Eq. (8)
22:     **for** $i = 1, \ldots M$ **do**
23:       Calculate $\nabla_a Q^{\mu}(s^{(i)}, a, W_c)\big|_{a=\mu(s^{(i)}, W_a)}$
24:     **end**
25:     Update $W_a$ using Eq. (10)
26:     Update $W_c'$ and $W_a'$ using Eqs. (12) and (13), respectively
27:   **end**
28: **end**
```

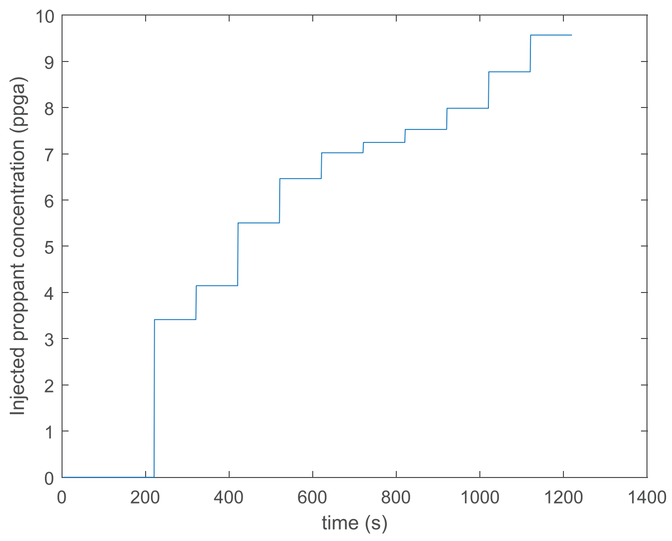

Fig. 8. The input profile implemented in the last episode.

tialized, the DRL controller shows poor performance. From episode 17, the controller performs reasonably well as observed in Fig. 6, but does not meet the criterion for termination until episode 603 wherein it gains a net reward of 3.481.

6.3. Second stage of learning

In the second stage of the learning process, the weights and the biases are initialized using the parameters obtained from the last episode of learning from the previous stage. The learning process is repeated as mentioned in Algorithm 2 and is terminated when all the states (i.e., the concentrations at 6 locations and the total amount of proppant injected) reach their respective set-points. The DRL controller is able to meet the criteria by episode 724. The learning curve in terms of net rewards per episode for both the stages is shown in Fig. 7. Initially, the curve undergoes fluctuations as the parameters are randomly initialized in the first stage, and thereafter, the controller performance improves until episode 603 where the criteria for the first stage learning is satisfied. The DRL controller continues to improve even in the second stage until episode 724 wherein the controller reaches the desired control objectives.

Fig. 8 shows the input profile implemented by the controller in the last episode of learning, and Fig. 9 shows the evolution of the concentrations at the selected locations in the same episode. The input profile obtained is a step-wise increasing profile with the injected proppant concentration values within the specified control limits of 0–10 ppga, and a maximum step increase of 3 ppga between two control actions. Additionally, as seen in Fig. 9, all the states are within their specified tolerance limits from their respective set-points. Hence, learning was terminated at the end of this episode.

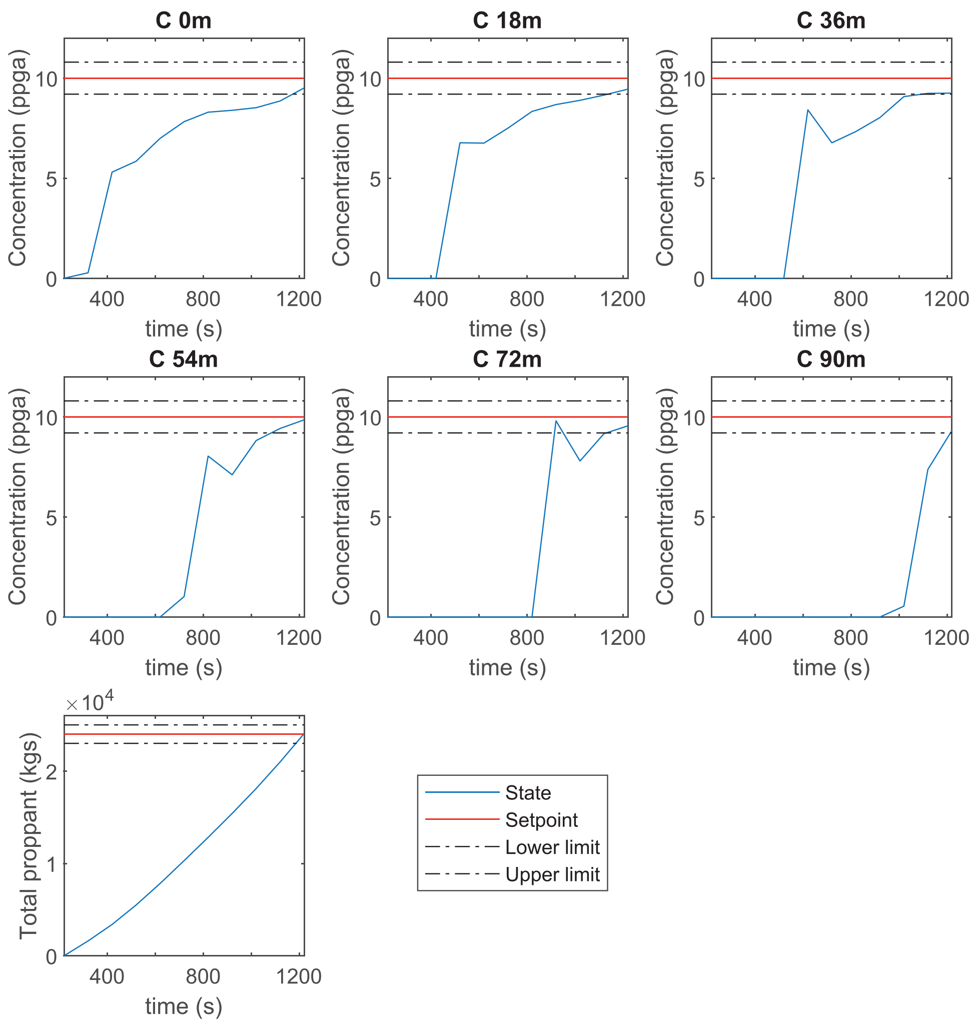

Fig. 9. Evolution of states in the last episode.

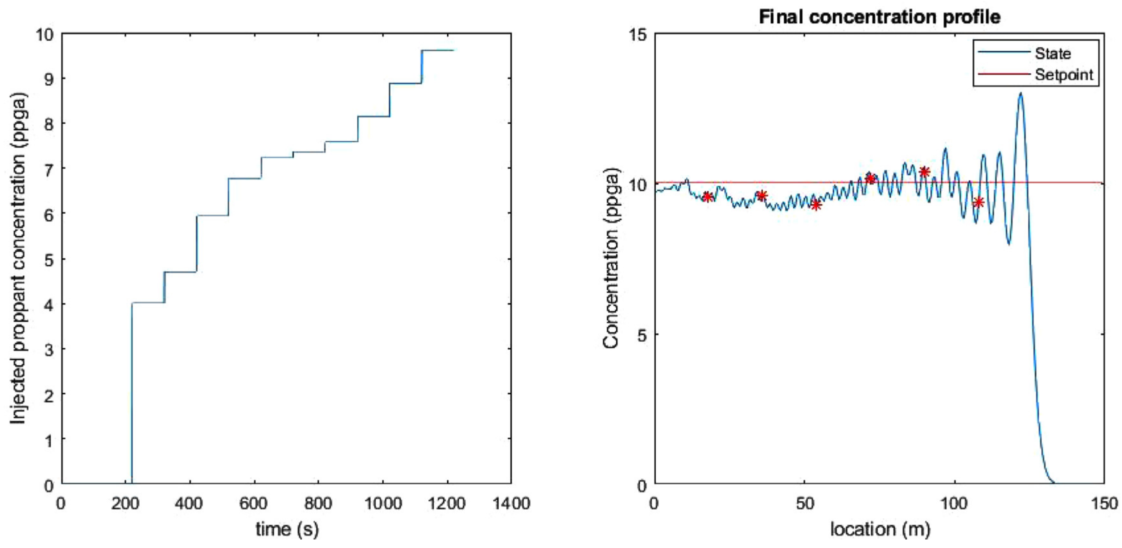

Fig. 10. The input profile obtained from the DRL controller (left) and the concentration profile at the end of pumping process (right) are presented.

In order to test the DRL controller's performance, we stop the learning process, utilize the actor to predict control actions, and obtain the corresponding outputs. Fig. 10 shows the inputs predicted by the DRL controller, and the concentration profile across the fracture length at the end of the proppant injection process. The input profile predicted by the controller satisfies the constraints, and the concentrations at the 6 selected locations are within the tolerance limits from the set-point.

7. Conclusions

This paper presents a DRL control framework based on DDPG and dimensionality reduction via PCA. This framework is able to learn a control policy for a high-dimensional and complex system, i.e., hydraulic fracturing. We demonstrate that multiple input constraints can be enforced on the controller through a careful design of the reward function. In order to quicken the learning process, we used a ROM to learn a sub-optimal policy offline and then used the learned parameters to re-initialize the learning process online. Our proposed framework enables the DRL controller to quickly learn a control policy in a continuous state and action space via self-learning. The advantage of the DRL controller over traditional controllers is the inherent ability to learn from data/measurements in an automated manner. On the contrary, careful initialization of various controller parameters is necessary, and using informative RL states along with a clear reward function that aligns with the objectives of designing the controller are both important factors too.

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.

CRediT authorship contribution statement

Mohammed Saad Faizan Bangi: Conceptualization, Methodology, Validation, Formal analysis, Investigation, Visualization, Writing – original draft, Writing – review & editing. **Joseph Sang-Il Kwon:** Conceptualization, Formal analysis, Resources, Writing – review & editing, Supervision, Project administration, Funding acquisition.

Acknowledgments

The authors gratefully acknowledge financial support from the National Science Foundation (CBET-1804407), the Department of Energy (DE-EE0007888-10-8), the Texas A&M Energy Institute, and the Artie McFerrin Department of Chemical Engineering. Also, the authors would like to thank Dr. Jong Woo Kim for helpful discussions.

References

Adachi, J., Siebrits, E., Pierce, A., Desroches, J., 2007. Computer simulation of hydraulic fractures.. Int. J. Rock Mech. Min. Sci. 44, 739–757.
Badgwell, T.A., Lee, J.H., Liu, K.-H., 2018. Reinforcement learning - overview of recent progress and implications for process control. Comput. Aid. Chem. Eng. 44, 71–85.
Bangi, M.S.F., Kwon, J.S.-I., 2020. Deep hybrid modeling of chemical process: application to hydraulic fracturing. Comput. Chem. Eng. 134, 106696. doi:10.1016/j.compchemeng.2019.106696.
Bangi, M.S.F., Narasingam, A., Siddhamshetty, P., Kwon, J.S.-I., 2019. Enlarging the domain of attraction of the local dynamic mode decomposition with control technique: application to hydraulic fracturing. Ind. Eng. Chem. Res. 58 (14), 5588–5601. doi:10.1021/acs.iecr.8b05995.
Barree, R., Conway, M., 1995. Experimental and numerical modeling of convective proppant transport.. J. Pet. Technol. 47, 216–222.
Bertsekas, D.P., 2005. Dynamic Programming and Optimal Control. MA: Athena Scientific.
Brujeni, L.A., Lee, J.M., Shah, S.L., 2010. Dynamic tuning of PI-controllers based on model-free reinforcement learning methods. In: Proceedings of the International Conference on Control, Automation and Systems (ICCAS), pp. 453–458.
Daneshy, A., 1978. Numerical solution of sand transport in hydraulic fracturing.. J. Pet. Technol. 30, 132–140.
Economides, M.J., Watters, L.T., Dunn-Normall, S., 1998. Petroleum Well Construction. Wiley, Chichester.
Gu, H., Desroches, J., 2003. New pump schedule generator for hydraulic fracturing treatment design. In: Proceedings of the SPE Latin American and Caribbean Petroleum Engineering Conference, Port-of-Spain, Trinidad and Tobago doi:10.2118/81152-ms.
Gu, Q., Hoo, K.A., 2014. Evaluating the performance of a fracturing treatment design.. Ind. Eng. Chem. Res. 53 (25), 10491–10503.
Gu, Q., Hoo, K.A., 2015. Model-based closed-loop control of the hydraulic fracturing process.. Ind. Eng. Chem. Res. 54 (5), 1585–1594.
Ioffe, S., Szegedy, C., 2015. Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on Machine Learning (ICML). In: JMLR Workshop and Conference Proceedings, 37, pp. 448–456. Lille, France
Kim, J.W., Park, B.J., Yoo, H., Oh, T.H., Lee, J.H., Lee, J.M., 2020. A model-based deep reinforcement learning method applied to finite-horizon optimal control of nonlinear control-affine system. J. Process Control 87, 166–178. doi:10.1016/j.jprocont.2020.02.003.
Kingma, D.P., Ba, J., 2015. Adam: a method for stochastic optimization. In: Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA.

Lee, J.H., Lee, J.M., 2006. Approximate dynamic programming based approach to process control and scheduling. Comput. Chem. Eng. 30, 1603–1618. doi:10.1016/j.compchemeng.2006.05.043.

Lehnert, L., Precup, D., 2015. Policy gradient methods for off-policy control. arXiv Preprint, arXiv:1512.04105

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. arXiv Preprint, arXiv:1509.02971

Ma, Y., Zhu, W., Benton, M.G., Romagnoli, J., 2019. Continuous control of a polymerization system with deep reinforcement learning. J. Process Control 75, 40–47. doi:10.1016/j.jprocont.2018.11.004.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602

Mnih, V., Kavukcuoglu, K., Silver, D., et al., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533.

Narasingam, A., Kwon, J.S., 2017. Development of local dynamic mode decomposition with control: application to model predictive control of hydraulic fracturing. Comput. Chem. Eng. 106, 501–511.

Narasingam, A., Kwon, J.S.-I., 2018. Data-driven identification of interpretable reduced-order models using sparse regression. Comput. Chem. Eng. 119 (2), 101–111. doi:10.1016/j.compchemeng.2018.08.010.

Narasingam, A., Siddhamshetty, P., Kwon, J.S., 2017. Temporal clustering for order reduction of nonlinear parabolic PDE systems with time-dependent spatial domains: application to a hydraulic fracturing process. AIChE J. 63 (9), 3818–3831.

Narasingam, A., Siddhamshetty, P., Kwon, J.S., 2018. Handling spatial heterogeneity in reservoir parameters using proper orthogonal decomposition based ensemble Kalman filter for model-based feedback control of hydraulic fracturing. Ind. Eng. Chem. Res. 57 (11), 3977–3989.

Nolte, K.G., 1986. Determination of proppant and fluid schedules from fracturing–pressure decline. SPE Prod. Eng. 1 (04), 255–265.

Nordgren, R., 1972. Propagation of a vertical hydraulic fracture. Soc. Petrol. Eng. J. 12, 306–314.

Novotny, E.J., 1977. Proppant transport. In: Proceedings of the 52nd SPE Annual Technical Conference and Exhibition, Denver, CO.

Perkins, T.K., Kern, L.R., 1961. Widths of hydraulic fractures. J. Pet. Technol. 13, 937–949.

Pistikopoulos, E.N., Barbosa-Povoa, A., Lee, J.H., Misener, R., Mitsos, A., Reklaitis, G.V., Venkatasubramanian, V., You, F., Gani, R., 2021. Process systems engineering - the generation next? Comput. Chem. Eng. 147, 107252. doi:10.1016/j.compchemeng.2021.107252.

Shin, J., Badgwell, T.A., Lee, J.H., Liu, K.-H., 2019. Reinforcement learning - overview of recent progress and implications for process control. Comput. Chem. Eng. 127, 282–294. doi:10.1016/b978-0-444-64241-7.50008-2.

Siddhamshetty, P., 2020. Modeling of Hydraulic Fracturing and Design of Online Optimal Pumping Schedule for Enhanced Productivity in Shale Formations. Texas A&M University Ph.D. thesis.

Siddhamshetty, P., Yang, S., Kwon, J.S.-I., 2018. Modeling of hydraulic fracturing and designing of online pumping schedules to achieve uniform proppant concentration in conventional oil reservoirs. Comput. Chem. Eng. 114, 306–317. FOCAPO/CPC 2017

Sidhu, H.S., Narasingam, A., Siddhamshetty, P., Kwon, J.S., 2018. Model order reduction of nonlinear parabolic PDE systems with moving boundaries using sparse proper orthogonal decomposition: application to hydraulic fracturing. Comput. Chem. Eng. 112, 92–100.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D., 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489. doi:10.1038/nature16961.

Silver, D., Lever, G., Hess, N., Degris, T., Wierstra, D., Riedmiller, M., 2014. Deterministic policy gradient algorithms. In: Proceedings of the 31st International Conference on Machine Learning, PMLR, Beijing, China, pp. 387–395.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., Hassabis, D., 2017. Mastering the game of Go without human knowledge. Nature 550, 354–359. doi:10.1038/nature24270.

Spielberg, S., Tulsyan, A., Lawrence, N.P., Loewen, P.D., Gopaluni, R.B., 2019. Toward self-driving processes: a deep reinforcement learning approach to control. AIChE. J. 65 (10). doi:10.1002/aic.16689.

Suglyama, M., 2015. Statistical Reinforcement Learning: Modern Machine Learning Approaches. CRC Press, Florida.

Sutton, R.S., Barto, A.G., 1998. Reinforcement Learning: An Introduction. MIT Press, Cambridge.

Tesauro, G., 1995. Temporal difference learning and TD-gammon. Commun. ACM 38 (3), 58–68. doi:10.3233/icg-1995-18207.

Vamvoudakis, K.G., Lewis, F.L., 2010. Online actor-critic algorithm to solve the continuous-time infinite horizon optimal control problem. Automatica 46 (5), 878–888. doi:10.1016/j.automatica.2010.02.018.

Van Overschee, P., De Moor, B., 1996. Subspace Identification for Linear Systems: Theory - Implementation - Applications. Springer, New York.

Vrabie, D., Lewis, F., 2009. Neural network approach to continuous-time direct adaptive optimal control for partially unknown nonlinear systems. Neural Netw. 22 (3), 237–246. doi:10.1016/j.neunet.2009.03.008.

Yang, S., Siddhamshetty, P., Kwon, J.S., 2017. Optimal pumping schedule design to achieve a uniform proppant concentration level in hydraulic fracturing.. Comput. Chem. Eng. 101 (C), 138–147.

Yoo, H., Kim, B., Kim, J.W., Lee, J.H., 2021. Reinforcement learning based optimal control of batch processes using Monte–Carlo deep deterministic policy gradient with phase segmentation. Comput. Chem. Eng. 144, 107133. doi:10.1016/j.compchemeng.2020.107133.