Mathematics of Control, Signals, and Systems (2021) 33:775–795
https://doi.org/10.1007/s00498-021-00300-3

ORIGINAL ARTICLE

Recurrent neural networks for stochastic control problems with delay

Jiequn Han[1] · **Ruimeng Hu**[2,3]

Received: 5 January 2021 / Accepted: 9 July 2021 / Published online: 22 July 2021

Abstract

Stochastic control problems with delay are challenging due to the path-dependent feature of the system and thus its intrinsic high dimensions. In this paper, we propose and systematically study deep neural network-based algorithms to solve stochastic control problems with delay features. Specifically, we employ neural networks for sequence modeling (e.g., recurrent neural networks such as long short-term memory) to parameterize the policy and optimize the objective function. The proposed algorithms are tested on three benchmark examples: a linear-quadratic problem, optimal consumption with fixed finite delay, and portfolio optimization with complete memory. Particularly, we notice that the architecture of recurrent neural networks naturally captures the path-dependent feature with much flexibility and yields better performance with more efficient and stable training of the network compared to feedforward networks. The superiority is even evident in the case of portfolio optimization with complete memory, which features infinite delay.

Keywords Deep learning · Stochastic control with delay · Recurrent neural networks · Stochastic differential delay equations

R.H. was partially supported by the NSF grant DMS-1953035, the Faculty Career Development Award and the Research Assistance Program Award, University of California, Santa Barbara.

✉ Ruimeng Hu
rhu@ucsb.edu

Jiequn Han
jiequnhan@gmail.com

[1] Department of Mathematics, Princeton University, Princeton, NJ 08544-1000, USA

[2] Department of Mathematics, University of California, Santa Barbara, CA 93106-3080, USA

[3] Department of Statistics and Applied Probability, University of California, Santa Barbara, CA 93106-3080, USA

1 Introduction

Stochastic control problems study the agent's rational behavior with the existence of uncertainty in observations or in the noise that drives the evolution of the system. Inclusion of delay in stochastic control problems is important for realistic applications, e.g., in economics for time-to-build problems [2,43], in marketing for modeling the "carryover" or "distributed lag" advertising effect [22,24], and in finance for portfolio selection under the market with memory and delayed responses [15,17,46,50]. See also [40, Chapter 1] for modeling systems with aftereffect in mechanics and engineering, biology, and medicine. To model the delay feature, the dynamics of the controlled system will depend not only on the current state but also on the history of δ time units prior to the current time, where δ is a fixed number and can be infinity. This makes the problem path-dependent and, thus, infinite-dimensional.

The challenge brought by the path-dependence feature in stochastic control problems with delay has attracted rich theoretical studies in the literature. For example, [22–24] proposed to reformulate them as infinite-dimensional Markovian problems and analyze the associated Hamilton–Jacobi–Bellman (HJB) equations. The dynamic programming principle was proved in [11,44]. In [53], the authors studied the problem using anticipated backward stochastic differential equations (ABSDEs), and [10,28,51] proved a stochastic maximum principle using the ABSDEs. [3] employed the so-called randomization method, to list a few. Meanwhile, except for the special cases where problems can be reduced to finite-dimensional ones [4,15,16,45,50], the stochastic control problems with delay remain practically intractable, and one needs to resort to numerical methods for possible solutions; see [18,19,41,42] for probabilistic approaches which analyzed the corresponding discretized control problems, and [9] for an analytical approach which focused on finite difference methods of the infinite-dimensional HJB equation. In both approaches, one has to temporally and spatially discretize the stochastic control problems with delay, yielding a finite-dimensional setup with dimensionality proportional to both the number of discretized timestamps and spatial grids. Therefore, all aforementioned existing algorithms can only work in the low-dimensional setting but encounter demanding challenges or become unfeasible when faced with high-dimensional cases.

This paper aims to address the aforementioned numerical challenges by deep learning-based algorithms, with the model described by a stochastic differential delay equation (SDDE). Observing deep neural networks' remarkable performance in representing high-dimensional functions in numerical computations in many fields [5,6,14,32–36,54], we naturally leverage them in the context of stochastic control problems with delay. Specifically, motivated by [29], we shall approximate the controls using neural networks of various architectures at each time, stack these subnetworks together to form a deep network, and train them simultaneously. The optimal parameters are obtained by minimizing the loss function, which is the proxy of the cost functional in the control problem. Although using deep neural networks to direct parameterize the strategy in optimal control problems is not new, e.g., in [8,20,29,38], our work has the following merits: Firstly, we develop deep learning algorithms with the focus on the feature of delay, which is by nature infinite-dimensional; To the best of the authors' knowledge, this is the first work in the literature that systematically

leverages neural networks to solve stochastic control problems with delay beyond the linear-quadratic case. Secondly, we systematically study the strengths and weaknesses of different neural network architectures by testing them on three typical examples with benchmark solutions. The carefully selected benchmark problems with open-sourced code facilitate the further study of numerical algorithms for stochastic control problems with delay features. Our main findings include:

(1) The algorithm based on recurrent neural networks can naturally capture the path-dependent feature, i.e., not requiring a priori knowledge of the lag time δ, and yield better performance with more efficient and stable training compared to feedforward neural networks;
(2) The former one is capable of dealing with more complex problems efficiently and accurately, e.g., problems with infinite delay $\delta = \infty$, and problems with state constraints;
(3) Both algorithms perform better when training with state processes (corresponds to closed-loop controls) than with background noise (corresponds to open-loop controls), especially for problems with state constraints.

The rest of the paper is organized as follows. In Sect. 2, we introduce the mathematical formulation of the stochastic control problems with delay in continuous time. We describe the deep learning-based algorithms in Sect. 3 and three benchmark examples in Sect. 4, followed by a systemic numerical study in Sect. 5. We make conclusive remarks and describe future works in Sect. 6.

2 The stochastic control problem with delay

On a complete probability space $(\Omega, \mathcal{F}, \mathbb{P})$, we consider a stochastic control problem in which the state process $X \in \mathbb{R}^n$ is characterized by a stochastic differential delay equation (SDDE):

$$\begin{cases} \mathrm{d}X(t) = b(t, X_t, \pi(t))\,\mathrm{d}t + \sigma(t, X_t, \pi(t))\,\mathrm{d}W(t), & t \in [0, T], \\ X(t) = \varphi(t), & t \in [-\delta, 0]. \end{cases} \tag{1}$$

Here, $\delta \geq 0$ is the fixed delay, π is the control process taking values in $\mathcal{A} \in \mathbb{R}^m$ and to be chosen in some admissible set $\mathbb{A}$ and $W(t)$ is an ℓ-dimensional standard Brownian motion. Throughout the paper, we denote by P_t the trajectory of a process P from time $t-\delta$ to t, and $P(t)$ the process value at time t, i.e., $P_t(s) = P(t+s)$, for $-\delta \leq s \leq 0$.

Remark 1 The delay parameter δ here is assumed to be deterministic and known. However, in a more realistic setting δ might be *unknown* a priori. We shall see that the recurrent neural networks (discussed in Sect. 3.2) can naturally take care of the unknown δ implicitly due to its architecture design.

Let $C := C([-\delta, 0], \mathbb{R}^n)$ be the Banach space of all continuous functions with the supremum norm:

$$\|y\|_C = \sup_{-\delta \leq s \leq 0} |y(s)|, \ \forall y \in C.$$

The drift b and volatility σ coefficients are deterministic functionals:

$$(b, \sigma) : [0, T] \times C \times \mathcal{A} \to (\mathbb{R}^n, \mathbb{R}^{n\times \ell}).$$

Denote by $L^2(\Omega, C)$ the space of all $\mathcal{F}$-measurable stochastic processes, i.e.,

$$\Omega \ni \omega \to X(\omega) \in C \text{ is in } L^2(\Omega, C), \text{ iff. } \int_\Omega \|X(\omega)\|_C^2 \, \mathrm{d}\mathbb{P}(\omega) < \infty,$$

then $L^2(\Omega, C)$ is complete with the semi-norm $\|X\|_{L^2(\Omega,C)} := [\int_\Omega \|X(\omega)\|_C^2 \, \mathrm{d}\mathbb{P}(\omega)]^{1/2}$. We assume that the initial path $\varphi \in L^2(\Omega, C)$ and is independent of the Brownian motion $W(t)$, and the existence of solution X to the SDDE (1) is considered in $L^2(\Omega, C([-\delta, T], \mathbb{R}^n))$. Let $(\mathcal{F}_t)_{t\geq 0}$ be the filtration supporting $W(t)$ and φ, and let $C([0, T], L^2(\Omega, C))$ be the space of all L^2-continuous C-valued $\mathcal{F}_t$-adapted process $P : [0, T] \ni t \to P_t \in L^2(\Omega, C)$ with the semi-norm:

$$\|P\|_{C([0,T],L^2(\Omega,C))} := \sup_{0\leq t\leq T} \|P_t\|_{L^2(\Omega,C)}.$$

The trajectory X_t of SDDE (1) is considered in $C([0, T], L^2(\Omega, C))$.

The agent aims to minimize her expected cost:

$$\mathbb{E}_\varphi \left[\int_0^T f(t, X_t, \pi(t)) \, \mathrm{d}t + g(X_T) \right], \tag{2}$$

for a given distribution of the initial condition φ and over all admissible strategies π in $\mathbb{A}$:

$$\begin{aligned} \mathbb{A} := \Big\{ &\{\mathcal{F}_t\}\text{-progressively measurable process } \pi : [0, T] \times \Omega \to \mathcal{A} \subset \mathbb{R}^m : \\ &\int_0^T \mathbb{E}[\pi(t)^2] \, \mathrm{d}t < \infty \Big\}. \end{aligned} \tag{3}$$

where the running cost f and terminal cost g are deterministic functionals, $f : [0, T] \times C \times \mathcal{A} \to \mathbb{R}$, $g : C \to \mathbb{R}$.

Usually, one requires uniform Lipschitz conditions in the second variable of b and σ to ensure the existence and uniqueness of strong solutions to SDDE (1), that is,

$$\begin{aligned} \|(b, \sigma)(t, y_1, \pi) - (b, \sigma)(t, y_2, \pi)\|_{L^2} \leq L\|y_1 - y_2\|_{L^2(\Omega,C)}, \\ \forall t \in [0, T] \text{ and } y_1, y_2 \in L^2(\Omega, C). \end{aligned}$$

See detailed analysis in Mohammed's monographs [48,49]. Assumptions on f and g would ensure the expected cost (2) is finite.

In this paper, instead of discussing necessary conditions for the admissibility, we aim at providing a systematic numerical study of deep learning algorithms for finding

optimal strategies to stochastic control problems with delay. That is, we focus on the deep neural networks' (DNNs) architecture design in order to handle the high-dimensionality arising from the delay and comparing their performance based on some tractable examples. In Sect. 4, we present three examples with tractability to benchmark our numerical schemes and support our findings. Note that the third example has an infinite history dependence (i.e., $\delta = \infty$), and the numerical results further illustrate the advantage of using recurrent neural networks in this more general framework.

3 Deep learning algorithm

Our numerical algorithm builds on the temporal discretization of (1)–(2) and approximating $\pi(t) \in \mathbb{R}^m$ using neural networks. More precisely, let $N_T \in \mathbb{N}$ and $0 = t_0 < t_1 < \ldots < t_{N_T} = T$ be a partition of size N_T on $[0, T]$. Without loss of generality, we assume they are equidistributed and the fixed delay $\delta < \infty$ covers N_δ subintervals:

$$h \equiv t_{k+1} - t_k, \ \forall k = 0, \ldots, N_T - 1, \ \text{and } \delta = N_\delta h.$$

Consequently, we can extend the partition to $[-\delta, 0]$:

$$-\delta = t_{-N_\delta} \leq t_{-N_\delta+1} \leq \ldots t_0 = 0, \ \text{with } t_{k+1} - t_k \equiv h, \ \forall k = -N_\delta, \ldots, -1.$$

We then consider the discretized version of (1)–(2):

$$X(t_{k+1}) = X(t_k) + b(t_k, X_{t_k}, \pi(t_k))h + \sigma(t_k, X_{t_k}, \pi(t_k))\Delta W(t_k), \tag{4}$$

$$\inf_{\{\pi(t_k)\}_{k=0}^{N_T-1}} \mathbb{E}\left[\sum_{k=0}^{N_T-1} f(t_k, X_{t_k}, \pi(t_k))h + g(X_T)\right], \tag{5}$$

where X_{t_k} represents the path with N_δ lags and $\Delta W(t_k)$ is the increment in Brownian motions:

$$X_{t_k} = (X(t_{k-N_\delta}), \ldots, X(t_k)), \quad \Delta W(t_k) = W(t_{k+1}) - W(t_k).$$

Regarding the discretized system (4), one would expect results similar to [41, Section 4]. That is, as the mesh size $h \to 0$, the value function associated to (4)–(5) converges to the one of the original problem (1)–(2); and the near-optimal control associated to (4) (which now as functions of X_{t_k}) is also near-optimal to the original one. Below we propose two architectures in deep learning for approximating $\pi(t_k) \in \mathbb{R}^m$.

3.1 Feedforward neural network

A feedforward neural network is a composition of several fully connected layers $F_{d_1,d_2}(x)$:

$$F_{d_1,d_2}(x) = \rho(Ax + b) : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2},$$

where $x \in \mathbb{R}^{d_1}$ and $F_{d_1,d_2}(x) \in \mathbb{R}^{d_2}$ are the input and output of this layer, $A \in \mathbb{R}^{d_2 \times d_1}$ and $b \in \mathbb{R}^{d_2}$ are the weight matrix and bias vector, and $\rho(\cdot)$ is the activation function applied on each element of the vector individually. Common choices of the activation function include rectified linear unit (ReLU), identity, sigmoid, hyperbolic tangent:

$$\rho_{\text{ReLU}}(x) = \max\{x, 0\}, \quad \rho_{\text{Id}}(x) = x, \quad \rho_{\text{s}}(x) = \frac{1}{1 + e^{-x}}, \quad \rho_{\tanh}(x) = \tanh(x).$$

Motivated by the path-dependent structure of the considered problems (the change of current state only depends on the history up to lag δ), a natural idea is to approximate $\pi(t_k)$ by a feedforward neural network taking the state history up to lag $\bar{\delta}$ as the input. Note here it could be $\bar{\delta} \neq \delta$ since we may not know the underlying true δ a priori. Without loss of generality, we assume $\bar{\delta} = N_{\bar{\delta}} h$ ($N_{\bar{\delta}} \in \mathbb{N}^+$) and define $\bar{X}_{t_k} \equiv (X(t_{k-N_{\bar{\delta}}}), \ldots, X(t_k), t_k) \in \mathbb{R}^{n \times (N_{\bar{\delta}}+1)+1}$. Then, we represent the policy as

$$\pi(t_k) \approx F_{d_I,m} \circ F_{d_{I-1},d_I} \cdots F_{d_1,d_2} \circ F_{n\times(N_{\bar{\delta}}+1)+1,d_1}(\bar{X}_{t_k}), \tag{6}$$

where I is the number of hidden layers. In this case, the algorithm will produce feedback controls, i.e., controls that are adapted to the canonical filtration of X, denoted by $\mathcal{F}_t^X$. Also, with a fixed input dimension and the added time variable in the input, we are able to share the parameters of sub-neural networks, thus reducing the parameter number by a factor of N_T compared to N_T different networks at each timestamp.

3.2 Recurrent neural network

The idea of recurrent neural networks (RNNs) [55] is to make use of sequential information. They have shown great success in natural language processing, handwriting recognition, etc. [25–27]. The most common RNN is long short-term memory (LSTM) [37]. The advantage of an LSTM is the ability to deal with the vanishing gradient problem and data with lags of unknown duration.

An LSTM is composed of a series of units, each of which corresponds to a timestamp, and each unit consists of a cell and three gates: input gate, output gate, and forget gate. Among these components, the cell keeps track of the information received so far, the input gate captures to which extent new input information flows into the cell, the forget gate captures to which extent the existing information remains in the cell, and the output gate controls to which extent the information in the cell will be used to compute the output of the unit. In our case, the k^{th} unit is responsible for approximating $\pi(t_k)$:

$$\begin{aligned}
&\text{forget gate: } f_k = \rho_{\mathrm{s}}(W_f x_k + U_f h_{k-1} + b_f),\\
&\text{input gate: } i_k = \rho_{\mathrm{s}}(W_i x_k + U_i h_{k-1} + b_i),\\
&\text{output gate: } o_k = \rho_{\mathrm{s}}(W_o x_k + U_o h_{k-1} + b_o),\\
&\text{cell: } c_k = f_k \odot c_{k-1} + i_k \odot \rho_{\tanh}(W_c x_k + U_c h_{k-1} + b_c),\\
&\text{output of the } k^{th} \text{ unit: } h_k = o_k \odot \rho_{\tanh}(c_k),
\end{aligned} \tag{7}$$

where the operator $\odot$ denotes the Hadamard product, x_k denotes the k^{th} input, c_k stores the cell information, and h_k, f_k, i_k, o_k and c_k are all d_h-dimensional vectors. We take $(X(t_0), t_0), (X(t_1), t_1), (X(t_2), t_2), \dots$ as the input sequence $x_0, x_1, x_2, \dots$ in practice, and specify the initial information of h_k, c_k according to the discretized initial condition φ (see details in Sect. 3.4). Then, we take an affine transformation of h_k as the proxy of $\pi(t_k)$:

$$\pi(t_k) \approx W h_k + b. \tag{8}$$

Although for both schemes (8) and (6), the input dimensions keep constant as k changes, using (6) requires prior knowledge of δ. That is, for (6) which we feed the discretized state values (4) of length $N_{\bar{\delta}} + 1$, to obtain the best performance, one needs to get a good estimate $\bar{\delta}$ of δ first; while for (8) we only need to provide the current state value $X(t_k)$. Notice that in an LSTM all input information up to time t_k is summarized by the k^{th} cell, but if the optimal control only depends on the past up to δ, the forget gates are designed for dropping out the unneeded information. This dropout is characterized by NN's parameters, which are determined by supervised learning. We shall detail the learning part in the next section.

We remark that there are many variations of LSTM, for instance, gated recurrent units (GRUs) [13] that do not have output gates, peephole LSTM [21] where h_{k-1} is mostly replaced by c_{k-1} in all gates, etc. The numerical experiments will be conducted using the standard LSTM introduced above, and extensions to the variants are straightforward.

3.3 Choice of input data: $X(t_k)$ or $W(t_k)$

In the previously proposed networks, we use the data consisting of the state $X(t_k)$ as the input. On one side, using $X(t_k)$ as the input data may lead to sub-optimal controls, as $(\mathcal{F}_t^X)_{t\geq 0}$ might be smaller than $(\mathcal{F}_t)_{t\geq 0}$ in general, and thus the resulting policy in the feedback form forms a strict subset of (3). On the other side, in many scenarios, there exists an optimal control in (3) of the feedback form. For instance, see examples of problems with delay, among many others, in [10,12,50]. This is also the case in control problems without delay, if the solution to the HJB equation is smooth enough, which provides the decoupling field of the corresponding forward–backward stochastic differential equations [7, Chapter 4]. So we do not lose much by searching within a smaller set.

A naive idea to enforce the progressive measurability imposed in (3) is to take the data consisting of the Brownian motion $\{W(t_k)\}_{k=1}^{N_T}$ as input. However, in numerical

experiments, we observed that the resulting networks become much more challenging to optimize, leading to much worse performance. One possible explanation is that when there are additional constraints on the admissible set $\mathbb{A}$, it is usually directly related to the state process $X(t)$. The constraints are hardly satisfied if NNs only know background noises W and need to infer the state X. Another drawback of using $W(t_k)$ as the input data in the feedforward model is that the whole input variable becomes $W_k = (W(t_0), \cdots, W(t_k))$, whose dimension increases dramatically as k approaches to N_T. In addition, due to different input dimensions across k, we cannot let sub-neural networks share parameters, which will further increase the number of parameters. Based on the above reasons, we only report results with the input data consisting of $X(t_k)$.

3.4 Implementation

To summarize the algorithms proposed in the above subsections, we essentially have

$$\pi(t_k) = \psi_k(\text{Data}; \theta),$$

where Data can be either $(X(t_{k-N_{\bar{\delta}}}), \ldots, X(t_k), t_k)$ in the feedforward model or $(X(t_k), t_k)$ in the LSTM model, and θ denotes all parameters appearing in (6) or (8).

The internal states h_k, c_k in the LSTM model (7) need to be initialized properly according to the initial segment φ. In our case, we start the LSTM model from the timestamp $t_{-N_\delta} = -\delta$ (when $\delta = \infty$, we choose a properly truncated time), and the initial states h_{-N_δ} and c_{-N_δ} are both initialized as $(X(-\delta = t_{-N_\delta}), 0, \cdots, 0)$. Here, we have assumed $d_h \geq n$ (recall that n is the dimension of $X(t)$) such that there is no information lost at the beginning, and additional zeros are added to match the dimension d_h. Then, we feed in the input sequence $(X(t_{-N_\delta+1}), t_{-N_\delta+1}), (X(t_{-N_\delta+2}), t_{-N_\delta+2}), \ldots$ to evolve the model according to (7) and output controls starting at t_0.

The optimal parameters θ^* are then be obtained by minimizing the following expected discretized loss using stochastic gradient descent algorithms:

$$\inf_{\{\psi_k \in \mathcal{N}_k\}_{k=0}^{N_T-1}} \mathbb{E}\left[\sum_{k=0}^{N_T-1} f(t_k, X_{t_k}, \psi_k(\text{Data}; \theta))h + g(X_T)\right].$$

Note that in some numerical examples, we may need to deal with constraints involving the control policy and/or state variables. When the policy π taking values in $\mathcal{A}$ is required to be nonnegative, we apply a ReLU activation function before the final output of the policy network to ensure such a property:

$$\pi(t_k) \leftarrow \rho_{\text{ReLU}}(\pi(t_k)).$$

When a 1-dimensional state variable $X(t)$ is required to be nonnegative, we add the corresponding penalty term in the cost function:

$$f(t_k, X_{t_k}, \psi_k(\text{Data}; \theta)) \leftarrow f(t_k, X_{t_k}, \psi_k(\text{Data}; \theta)) + \xi \max\{-X(t_{k+1}), 0\}$$

where ξ is a penalty coefficient. More complicated constraints can be dealt with by the penalty method in a similar way (see, e.g., [29]). Details on the choices of ψ_k, algorithm to obtain θ^*, N_T, etc., are presented at the beginning of Sect. 5.

4 Benchmark examples

This section presents three tractable examples: a linear-quadratic regulator problem with delay in engineering, an optimal consumption problem in a financial market with delayed dynamics, and a portfolio optimization problem with infinite delay. They together serve as preparation of numerical experiments in Sect. 5. With specific formulas of (b, σ, f, g) depending on the current state $X(t)$, the weighted average of X_t and $X(t-\delta)$, the first two problems turn out to be essentially finite-dimensional and admit solutions expressed in a simple form. This allows us to benchmark and compare our proposed two deep learning schemes. A third example, where the portfolio performance depends on the exponential average of all the historical value ($\delta = \infty$), is presented with analytical solutions as well, to further evidence the superiority of the LSTM model. In the sequel, we will focus on describing the model and keep technical details minimal. The optimal control π^* and cost V_0 to the problem (1)–(2):

$$V_0 := \sup_{\pi \in \mathbb{A}} \mathbb{E}_\varphi \left[\int_t^T f(s, X_s, \pi(s))\, \mathrm{d}s + g(X_T) \middle| X_0 = \varphi \right]$$

are provided in Propositions 1–3 and we give references on their proofs.

For the first two examples, we take a special dependence form for the functionals (b, σ, f, g). To be specific, the path dependence is characterized by distributed delay $Y(t)$ and discrete delay $Z(t)$:

$$Y(t) := \int_{-\delta}^0 e^{\lambda s} X(t+s)\, \mathrm{d}s \text{ and } Z(t) := X(t-\delta).$$

Note that $Y(t)$ is also called the exponentially decayed weighted moving average. Problems with such explicit structures have been studied in [4,15,16,45,50,51]. Some allow general analysis, and some have tractable examples. We choose two examples from [4] and present them in Sects. 4.1 and 4.2.

4.1 Linear-quadratic problem with delay

Stochastic linear-quadratic (LQ) problems were extensively studied in the literature. They have appeared in many contexts and have been used to benchmark various numerical algorithms due to their tractability. LQ problems with delay were first

investigated by Kolmanovskiĭ and Shaĭkhet [40]. The delay version can be stated as:

$$(LQ)\begin{cases} \mathrm{d}X(t) = (A_1(t)X(t) + A_2(t)Y(t) + A_3Z(t) + B(t)\pi(t))\,\mathrm{d}t + \sigma(t)\,\mathrm{d}W(t), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad t \in [0,T] \\ \min_{\pi(t)\in\mathbb{R}^m} \mathbb{E}_\varphi \Big[\displaystyle\int_0^T (X(t) + e^{\lambda\delta}A_3Y(t))^{\mathrm{T}} Q(t)(X(t) + e^{\lambda\delta}A_3Y(t)) \\ +\pi(t)^{\mathrm{T}} R(t)\pi(t)\,\mathrm{d}t + (X(T) + e^{\lambda\delta}A_3Y(T))^{\mathrm{T}} G(X(T) + e^{\lambda\delta}A_3Y(T)) \Big], \end{cases} \tag{9}$$

where $X_0 = \varphi \in L^2(\Omega, C)$ is a given initial segment, $A_1(t)$, $A_2(t)$, $Q(t) \in \mathbb{R}^{n\times n}$, $B(t) \in \mathbb{R}^{n\times m}$, $R(t) \in \mathbb{R}^{m\times m}$ are deterministic matrix-valued functions in $L^\infty[0,T]$, $\sigma(t) \in \mathbb{R}^{n\times \ell}$ is a deterministic matrix-valued function in $L^2[0,T]$, A_3, $G \in \mathbb{R}^{n\times n}$ are deterministic matrices. We assume that $Q(t)$, G are positive semi-definite and $R(t)$ is positive definite for all $t \in [0,T]$ and continuous on $[0,T]$. To have a tractable solution, we further prescribe the relation:

$$A_2(t) = e^{\lambda\delta}(\lambda I_n + A_1(t) + e^{\lambda\delta}A_3)A_3, \tag{10}$$

where I_n is the identity matrix with rank n. This example was studied in [4, Section 4], and we summarize the main results as follows for completeness.

Proposition 1 *Consider the stochastic control problem (LQ) with the initial segment φ. Assume that $A_3 \neq 0$ and (10) holds, then the optimal control is given by*

$$\pi^*(t) = -R^{-1}(t)B(t)^{\mathrm{T}}P(t)(X^*(t) + e^{\lambda\delta}A_3Y^*(t)), \text{ where } Y^*(t) = \int_{-\delta}^0 e^{\lambda s}X^*(t+s)\,\mathrm{d}s,$$

$X^*(t)$ solves the SDDE (9) with the optimal control π^*, and $P(t)$ solves the Riccati equation

$$\begin{aligned} \dot{P}(t) = {} & P(t)B(t)R^{-1}(t)B(t)^{\mathrm{T}}P(t) - (A_1(t) + e^{\lambda\delta}A_3)^{\mathrm{T}}P(t) \\ & - P(t)(A_1(t) + e^{\lambda\delta}A_3) - Q(t), \quad P(T) = G. \end{aligned}$$

The optimal cost V_0 to problem (LQ) is given by

$$\begin{aligned} V_0 \equiv V(0,X_0) = {} & (X(0) + e^{\lambda\delta}A_3Y(0))^{\mathrm{T}}P(0)(X(0) + e^{\lambda\delta}A_3Y(0)) \\ & + \int_0^T \mathrm{Tr}(\sigma(s)\sigma^{\mathrm{T}}(s)P(s))\,\mathrm{d}s. \end{aligned}$$

4.2 Optimal consumption in a delayed financial market

In this section, we consider $X(t)$ described in the SDDE (1) as the wealth process, and study the utility maximization problem from both consumption and terminal wealth.

Here, we do not specify dynamics of tradable assets, but directly model the wealth return and volatility, both depending on the sliding average $Y(t) = \int_{-\delta}^{0} e^{\lambda s} X(t+s)\,\mathrm{d}s$ and the past value $Z(t) = X(t-\delta)$. This can be interpreted as investing on of some path-dependent options. Let $c(t)$ be the investor's consumption rate at time t, then the optimal consumption problem is stated as:

$$(C)\quad \begin{cases} \mathrm{d}X(t) = (\mu(t, X(t), Y(t)) + aZ(t) - c(t))\,\mathrm{d}t + \sigma(t, X(t), Y(t))\,\mathrm{d}W(t), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad t \in [0, T], \\ \max_{c(t)\in\mathbb{R}^+} \mathbb{E}_{\varphi}\left[\int_0^T e^{-\beta t} U_1(c(t))\,\mathrm{d}t + U_2(X(T) + ae^{\lambda\delta} Y(T))\right], \\ \text{subject to } X(t) \geq 0,\ t \in [0, T], \end{cases} \tag{11}$$

where U_1, U_2 are utility functions on the consumption and the terminal wealth, and a is a positive real number. In this example, all processes are scalars, i.e., $n = m = \ell = 1$. To have a tractable solution, we take the power utility $U_1(x) = U_2(x) = \frac{1}{\gamma}x^{\gamma}$ and require

$$\mu(t, x, y) = ae^{\lambda\delta}(ae^{\lambda\delta} + \lambda)y + \mu_t T(x, y), \quad \sigma(t, x, y) = \sigma_t T(x, y), \tag{12}$$

where $T(x, y) = x + ae^{\lambda\delta} y$, and μ_t and σ_t are some positive continuous functions. This example has been studied in [4, Section 5], and a special case $a = -\lambda e^{-\lambda\delta}, \lambda < 0$ was treated in [15]. We now summarize the results as follows.

Proposition 2 *Consider the optimal consumption problem (C) with the initial wealth segment $X_0 = \varphi$, and assume (12) holds. Then, the problem value V_0 is given by*

$$V_0 \equiv V(0, X_0) = \frac{1}{\gamma} p(0)^{1-\gamma} (X(0) + ae^{\lambda\delta} Y(0))^{\gamma},$$

where $p(t)$ solves the following equation:

$$\dot{p}(t) = \left(\frac{1}{2}\gamma\sigma_t^2 - \frac{\gamma}{1-\gamma}(\mu_t + ae^{\lambda\delta})\right) \cdot p(t) - e^{-\frac{\beta t}{1-\gamma}}, \quad p(T) = 1.$$

The optimal consumption $c^*(t)$ is

$$c^*(t) = e^{-\frac{\beta t}{1-\gamma}} \frac{1}{p(t)} (X^*(t) + ae^{\lambda\delta} Y^*(t)),$$

with $X^*(t)$ following the SDDE (11) associated with c^*.

Remark 2 If the terminal utility is changed to $e^{-\beta_2 T} U_2(X(T) + ae^{\lambda\delta} Y(T))$, then Proposition 2 still holds except for the terminal condition of $p(t)$. It then becomes

$$p(T) = e^{-\beta_2 T/(1-\gamma)}.$$

If the running utility is changed to $U_1(x) = \frac{\eta x^\gamma}{\gamma}$, the ordinary differential equation for $p(t)$ then becomes

$$\dot{p}(t) = (\frac{1}{2}\gamma\sigma_t^2 - \frac{\gamma}{1-\gamma}(\mu_t + ae^{\lambda\delta})) \cdot p(t) - \eta^{\frac{1}{1-\gamma}} e^{-\frac{\beta t}{1-\gamma}}.$$

4.3 Portfolio optimization with complete memory

In the last example, we study a portfolio optimization problem with an infinite delay feature. More precisely, in the SDDE (1) and cost functional (2), the dependence is on $X(t)$ and the exponential average of all the history value $Y(t)$,

$$Y(t) := \int_{-\infty}^{0} e^{\lambda s} X(t+s)\,\mathrm{d}s.$$

Let $X(t)$ be the wealth process as in Sect. 4.2. Here, we consider both investment $\pi(t)$ and consumption $c(t)$ and parameterize them proportional to the wealth $X(t)$, i.e., $c(t)$ denotes the fraction of wealth consumed at time t. Then, the problem reads:

$$(P)\quad \begin{cases} \mathrm{d}X(t) = [((\mu_1 - r)\pi(t) - c(t) + r)X(t) + \mu_2 Y(t)]\,\mathrm{d}t + \sigma\pi(t)X(t)\,\mathrm{d}W(t), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad t \in [0,T] \\ \max\limits_{\substack{c(t)\in\mathbb{R}^+ \\ \pi(t)\in\mathbb{R}}} \mathbb{E}_\varphi \left[\int_0^T e^{-\beta t} U_1(c(t)X(t))\,\mathrm{d}t + e^{-\beta T} U_2(X(T), Y(T)) \right]. \end{cases} \tag{13}$$

The above problem was introduced in [52]. They derived explicit solutions under exponential, power and log utilities. Given the history up to time 0, $\varphi(\cdot) \in L^2(\Omega, C((-\infty, 0], \mathbb{R}))$, the problem value turns out depending only on $X(0) \equiv \varphi(0)$ and $Y(0) \equiv \int_{-\infty}^{0} e^{\lambda s}\varphi(t+s)\,\mathrm{d}s$.

Remark 3 For different choices of utility functions, we may or may not require $X(t) \geq 0$, for $t \in [0, T]$. Theoretically, if the strategy $(\pi(t), c(t))$ satisfies

$$|\pi(t)X(t)| \leq \Lambda_0 |X(t) + Y(t)|,$$
$$|c(t)X(t)| \leq \Lambda_0 |X(t) + Y(t)|,$$

for some constant Λ_0, then $X(t)$ in (13) stays positive. For proof, see [52, Lemma 2.1].

Proposition 3 *Consider the portfolio optimization problem with infinite delay (P) with the complete history $X_0 = \varphi \in L^2(\Omega, C((-\infty, 0], \mathbb{R}))$ under log utility:*

$$U_1(x) = \log(x), \quad U_2(x,y) = \frac{1}{\beta}\log(x + \eta y), \quad \eta = \frac{1}{2}\left(\sqrt{(r+\lambda)^2 + 4\mu_2} - (r+\lambda)\right).$$

and require a state constraint $X(t) \geq 0$, $\forall t \in [0, T]$. *Then, the optimal investment and consumption rates are*

$$\pi^*(t) = \frac{(\mu_1 - r)(X^*(t) + \eta Y^*(t))}{\sigma^2 X^*(t)},$$
$$c^*(t) = \frac{\beta(X^*(t) + \eta Y^*(t))}{X^*(t)},$$

where $X^*(t)$ *solves the SDDE* (13) *associated to* π^* *and* c^*. *The value function is of the form*

$$V(t, x, y) = p(t) + \frac{1}{\beta}\log(x + \eta y)$$

with

$$p(t) = \frac{\Lambda_2}{\beta}(1 - e^{-\beta(T-t)}), \quad \Lambda_2 = \frac{1}{2}\frac{(\mu_1 - r)^2}{\beta\sigma^2} + \log(\beta) - 1 + \frac{1}{\beta}(r + \eta),$$

and the optimal cost $V_0 = V(0, X(0), Y(0))$.

5 Numerical results

In this section, we present numerical results on the three benchmark examples introduced above: linear-quadratic problem (Sect. 5.1), optimal consumption (Sect. 5.2), and portfolio optimization (Sect. 5.3). We choose $n = 10$ in the first example, while $n = 1$ is fixed in the settings of the other two examples. In all three experiments, we observe consistently good performance compared to the benchmark solutions. The important hyperparameters and running time of these problems are summarized in Table 1, and other problem-dependent parameters will be introduced in the corresponding subsections. The initial path $\varphi \in L^2(\Omega, C)$ is modeled as a fixed deterministic path perturbed by some white noise.

In numerical solutions, the distributed delay $Y(t)$ is approximated by the midpoint quadrature rule, in both calculating the benchmark solution and simulating the dynamics of $X(t)$ (cf. Eqs. (9), (11) and (13)). The activation function in the feedforward model is ReLU in all the layers except that in the final output layer, it is an identity if there is no constraint on the control. In the learning process, to guarantee a fair comparison, in each problem, we use the same learning rate and choose the size of the feedforward network and the LSTM properly such that they have a similar number of parameters. We adopt Adam optimizer [39] to optimize the parameters in neural networks. In each update step, we simulate 128 (batch size) paths to compute the stochastic gradient through backpropagation. For each example, we run the algorithm three times and report the average result.

Table 1 Some hyperparameters and running time of the numerical examples

Parameter/problem	Linear-quadratic	Optimal consumption	Portfolio optimization
T	1	2	5
δ	1	0.8	$\infty^{\dagger}$
N_T	40	60	100
ξ (constraint penalty)	0 (no constraint)	10^5	10
Running time (s)	4294	1429	941

†The initial segment φ is constant when $t \leq -4$ to ease the computation of the distributed delay $Y(t)$. Both neural network models are initialized at $t = -4$

The algorithm is implemented in Python by using the machine learning library TensorFlow [1]. The code can be found in a public GitHub repository[1] upon publication, and thus, the results presented here can be straightforwardly reproduced and further developed.

5.1 Linear-quadratic problem

We consider a 10-dimensional example in Sect. 4.1, in which $n = 10$, $m = 10$, and $\lambda = 0.1$. In Eq. (9), A_1, A_3, B, σ are random generated constant coefficient matrices, Q, R, G are constant matrices proportional to identity matrices, and A_2 is determined by (10). The dimension of the hidden state in the LSTM model is $d_h = 200$, which gives 171,610 parameters in total. The feedforward model takes the state history as inputs up to lag $\bar{\delta} = \delta$ with $N_{\bar{\delta}} = 40$, and it has 2 hidden layers with a width of 300, which gives 108,910 parameters in total. The LSTM model is trained with 16,000 steps, in which the learning rate is 0.005 for the first 8000 steps and 0.0005 for the second 8000 steps. The feedforward model is trained with 32,000 steps, in which the learning rate is 0.005 for the first 8000 steps and 0.0005 for the remaining 24,000 steps. We calculate the total cost on the validation data every 200 steps and plot the curve against training time in Fig. 1. We can see that with the same learning rates and roughly the same number of parameters, the LSTM model converges to a solution with a lower cost compared to the feedforward model. We further test the learned policy on a new set of sample paths, and the average cost of the LSTM model and feedforward model are 2.8740 and 2.8812. Note that the cost obtained by the policy discretized from the analytical optimal control is 2.8723 (cf. Proposition 1). Figures 2 and 3 depict one sample path (first 5 dimensions only) of the optimal state X and control π provided by two neural networks in comparison with the analytical solution, in which the LSTM architecture presents a better agreement.

In the above experiment, the lag time $\bar{\delta}$ processed by the feedforward model is chosen to be the same as δ. One main drawback of the feedforward model is that it requires knowing the true lag time δ a priori to determine the network's size. If the chosen lag time $\bar{\delta}$ is smaller than the actual lag δ time, which means there is a loss of information when the feedforward network processes the data, the final performance

[1] https://github.com/frankhan91/RNN-ControlwithDelay.

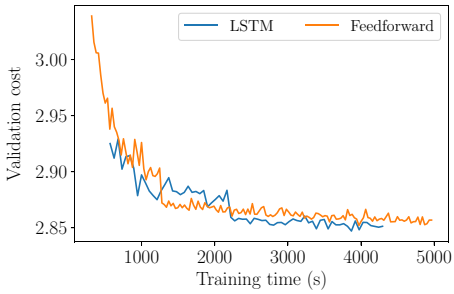

Fig. 1 Training curve of two models in the example of linear-quadratic problem

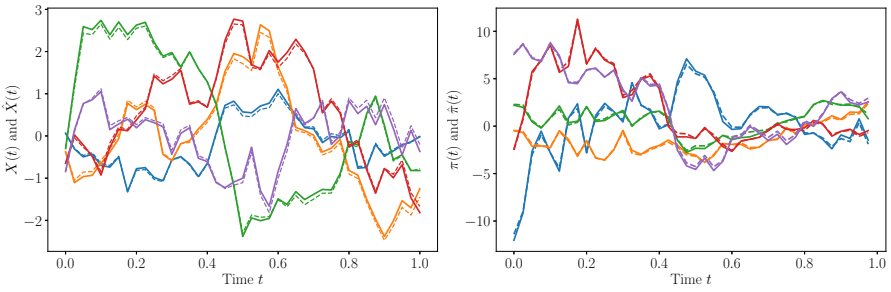

Fig. 2 A sample path of the first 5 dimensions of the state $X(t)$ and control $\pi(t)$ obtained from the LSTM model. Left: the optimal state process discretized from the analytical solution $X_i(t)$ (solid lines) and its approximation $\hat{X}_i(t)$ (dashed lines) provided by the approximating control, under the same realized path of Brownian motion. Right: comparisons of the optimal control $\pi_i^*(t)$ (solid lines) and $\hat{\pi}_i(t)$ (dashed lines)

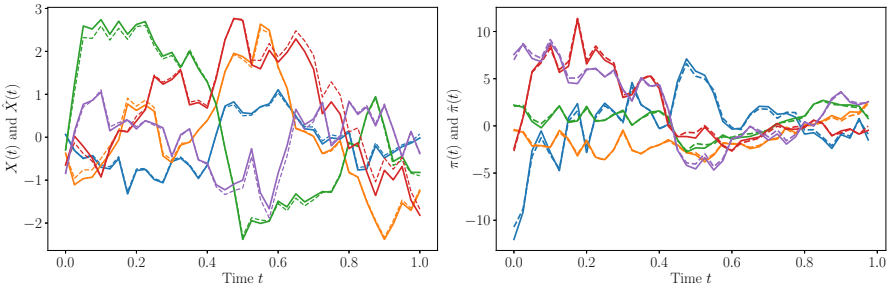

Fig. 3 A sample path of the first 5 dimensions of the state $X(t)$ and control $\pi(t)$ obtained from the feedforward model. Left: the optimal state process $X_i(t)$ discretized from the analytical solution (solid lines) and its approximation $\hat{X}_i(t)$ (dashed lines) provided by the approximating control, under the same realized path of Brownian motion. Right: comparisons of the optimal control $\pi_i^*(t)$ (solid lines) and $\hat{\pi}_i(t)$ (dashed lines)

might be compromised. To quantify this effect, we test the feedforward model with different processed lag time $\bar{\delta}$ from 0.2 to 1 with step size 0.1, while the actual lag $\delta = 1$. The corresponding optimized costs are shown in Fig. 4. As expected, we observe that the cost increases as the lag time processed by the feedforward model decreases. A higher optimized cost indicates that the model can only find a sub-optimal but not an optimal strategy due to the lack of information.

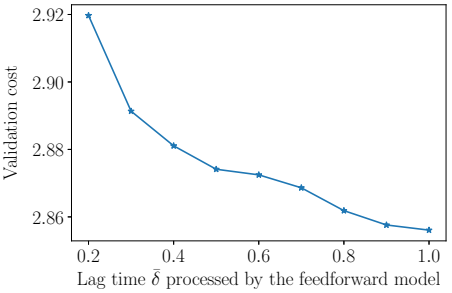

Fig. 4 The effect of lag time $\bar{\delta}$ processed by the feedforward model in the example of linear-quadratic problem. The lag time δ in the actual system is 1

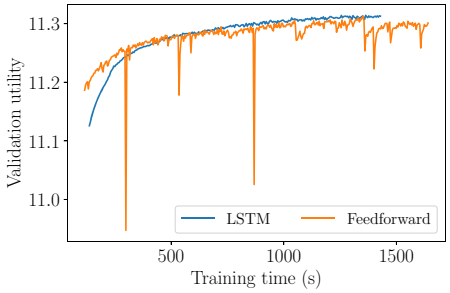

Fig. 5 Training curve of two models in the example of optimal consumption

5.2 Optimal consumption

We consider the example introduced in Sect. 4.2, with $\lambda = 0.1$, $\beta = 0.1$, $a = 0.2$, $\gamma = 0.7$, $\mu_t = 0.1$, and $\sigma_t = 0.5$. The dimension of the hidden state in the LSTM model is $d_h = 30$, which gives 3991 parameters in total. The feedforward model takes the state history as input up to lag $\bar{\delta} = \delta$ with $N_{\bar{\delta}} = 24$, and it has 2 hidden layers with a width of 54, which gives 4483 parameters in total. The LSTM model and feedforward model are trained with 60,000 and 80,000 steps, respectively, with a learning rate of 0.0001. We calculate the total utility on the validation data every 200 steps and plot the curve against training time in Fig. 5. We can see that with the same learning rates and a similar number of parameters, the LSTM model still converges to a higher utility (which corresponds to a better control policy) than the feedforward model. Furthermore, the training process of the feedforward model is much more unstable than the LSTM model, probably due to its insufficient capability to handle state constraints. We further test the learned policy on a new set of sample paths, and the average utility of the LSTM model and feedforward model are 10.7202 and 10.7100. Note that the utility obtained by the policy discretized from the analytical optimal control is 10.7142 (cf. Proposition 2). Figures 6 and 7 depict three sample paths provided by two models in comparison with the analytical solution, in which the LSTM model presents a better agreement.

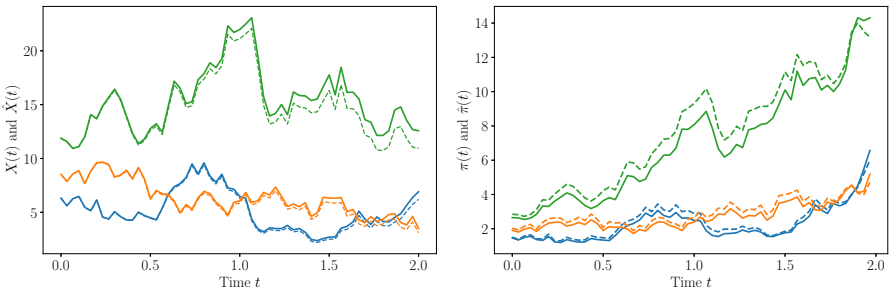

Fig. 6 Three sample paths of the state $X(t)$ and control $\pi(t)$ obtained from the LSTM model. Left: the optimal state process $X(t)$ discretized from the analytical solution (solid lines) and its approximation $\hat{X}(t)$ (dashed lines) provided by the approximating control, under the same realized paths of Brownian motion. Right: comparisons of the optimal control $\pi_i^*(t)$ (solid lines) and $\hat{\pi}_i(t)$ (dashed lines)

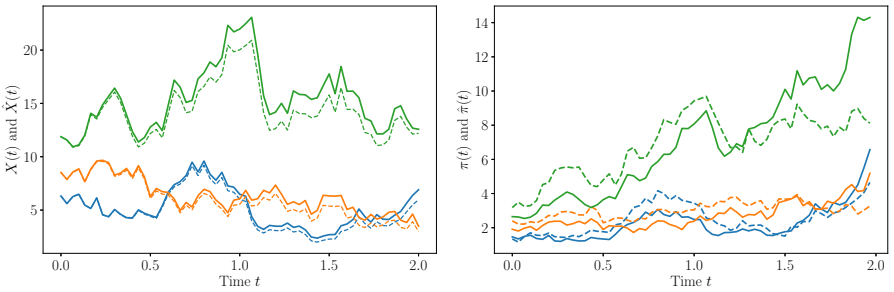

Fig. 7 Three sample paths of the state $X(t)$ and control $\pi(t)$ obtained from the feedforward model. Left: the optimal state process $X(t)$ discretized from the analytical solution (solid lines) and its approximation $\hat{X}(t)$ (dashed lines) provided by the approximating control, under the same realized paths of Brownian motion. Right: comparisons of the optimal control $\pi_i^*(t)$ (solid lines) and $\hat{\pi}_i(t)$ (dashed lines)

5.3 Portfolio optimization

We now consider the example introduced in Sect. 4.3, with $\lambda = 0.3$, $\beta = 0.2$, $\mu_1 = 0.1$, $\mu_2 = 0.2$, $r = 0.05$, and $\sigma = 0.4$. In this example, the model has an infinite memory. We choose the initial segment φ to be constant for $t \leq -4$ and the white noise perturbation only presents for $t \in [-4, 0]$. Under this setting, the computation of the distributed delay $Y(t)$, defined as an integral running from $-\infty$ to t, becomes tractable. Both neural network models are initialized at $t = -4$ for the sake of fairness. The dimension of the hidden state in the LSTM model is $d_h = 60$, which gives 15242 parameters in total. The feedforward model takes the state history as input up to lag $\bar{\delta} = 4 < \delta = \infty$ with $N_{\bar{\delta}} = 80$, and it has 2 hidden layers with a width of 96, which gives 17474 parameters in total. The LSTM model and the feedforward model are trained with 8000 and 20,000 steps, respectively, both with a learning rate of 5×10^{-5}. We calculate the total utility in (13) on the validation data every 100 steps and plot the curve against the training time in Fig. 8. We can see that in this example, the LSTM model converges to a higher utility much faster than the feedforward model, and the

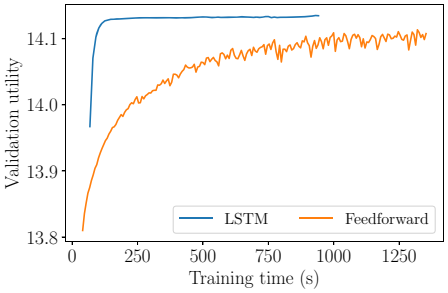

Fig. 8 Training curve of two models in the example of portfolio optimization

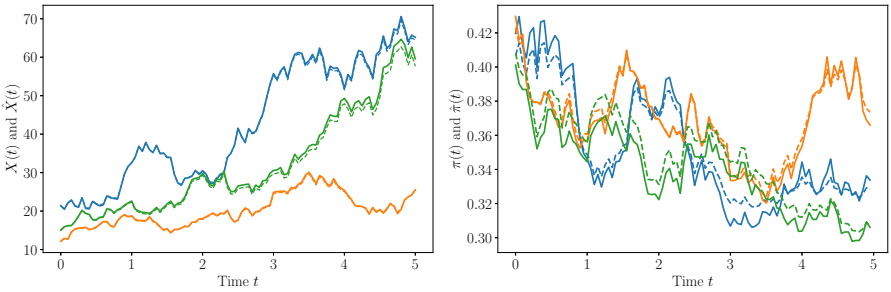

Fig. 9 Three sample paths of the state $X(t)$ and control $\pi(t)$ obtained from the LSTM model. Left: the optimal state process $X(t)$ discretized from the analytical solution (solid lines) and its approximation $\hat{X}(t)$ (dashed lines) provided by the approximating control, under the same realized paths of Brownian motion. Right: comparisons of the optimal control $\pi_i^*(t)$ (solid lines) and $\hat{\pi}_i(t)$ (dashed lines)

convergence is much stable. We further test the learned policy on a new set of sample paths, and the average utilities obtained from the LSTM model and feedforward model are 14.5326 and 14.4979. Note that the utility obtained by the policy discretized from the analytic optimal control is 14.5316 (cf. Proposition 3). Figures 9 and 10 depict three sample paths provided by two models in comparison with the analytical solution, in which the LSTM model presents a much better agreement.

We believe that the remarkable outperformance of the LSTM model is due to the infinite memory feature of this problem, which can be naturally captured by the LSTM model but not the feedforward one. In particular, for the feedforward model, the first input at $t = 0$ is $(X(-4), \ldots, X(0), 0)$, and later inputs keep the same sliding window length ($N_{\bar{\delta}} = 80$) of the state history and gradually drop the initial information. We believe that such information loss is the main reason why the feedforward model performs significantly sub-optimal. Numerically, we also observed that the choice of $\bar{\delta} < 4$ will even worsen the performance due to the information loss at the initial step.

6 Conclusion

In this paper, we propose and systematically study deep neural network-based algorithms to solve stochastic control problems with delay features. The challenge is

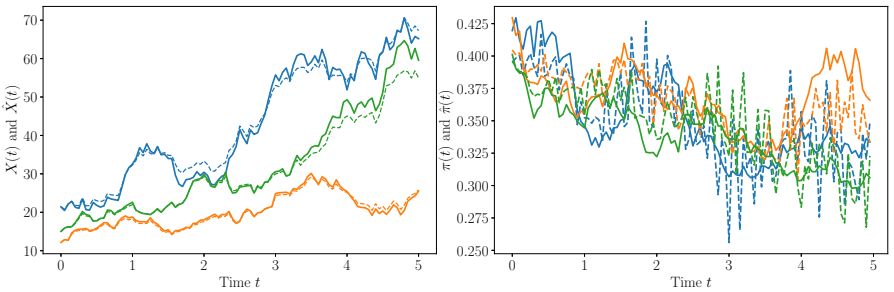

Fig. 10 Three sample paths of the state $X(t)$ and control $\pi(t)$ obtained from the feedforward model. Left: the optimal state process $X(t)$ discretized from the analytical solution (solid lines) and its approximation $\hat{X}(t)$ (dashed lines) provided by the approximating control, under the same realized paths of Brownian motion. Right: comparisons of the optimal control $\pi_i^*(t)$ (solid lines) and $\hat{\pi}_i(t)$ (dashed lines)

brought by the path-dependent property in the SDDE system and thus its intrinsic high dimensions. Viewing the optimal policy as a function of state process path $(X(t))_{t\geq 0}$ or background noise path $(W(t))_{t\geq 0}$, we naturally start with feedforward neural networks and take the discretized process as inputs. With the path-dependent feature, we then employ recurrent neural networks for sequence modeling to parameterize the policy and optimize the objective function. On the architecture design level, we point out that recurrent neural networks such as the LSTM model do not require prior knowledge of lag time δ. The models are then tested on three benchmark examples: (1) linear-quadratic problem with fixed delay; (2) optimal consumption in a financial market with fixed finite delay; (3) portfolio optimization with complete memory. Numerically, we found that the architecture of recurrent neural networks can naturally capture the path-dependent feature with much flexibility, resulting in a better performance with more efficient and stable training compared to the feedforward architectures. The superiority is even evident for infinite delay $\delta = \infty$, supported by our third example. Moreover, the carefully selected benchmark problems with open-sourced code will facilitate the further study of numerical algorithms for stochastic control problems with delay features. Remark that the numerical techniques developed in this paper can be naturally generalized to the studies on the deep fictitious play algorithms [30,31,38,47,56] for finding the Nash equilibrium in stochastic differential games by including delay in the games. This will be investigated in future work.

Acknowledgements J.H. and R.H. are grateful to the reviewers for their valuable and constructive comments.

References

1. Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M et al (2016) Tensorflow: a system for large-scale machine learning. In: 12th USENIX symposium on operating systems design and implementation (OSDI 16), pp 265–283
2. Asea PK, Zak PJ (1999) Time-to-build and cycles. J Econ Dyn Control 23(8):1155–1175

3. Bandini E, Cosso A, Fuhrman M, Pham H (2018) Backward SDEs for optimal control of partially observed path-dependent stochastic systems: a control randomization approach. Ann Appl Probab 28(3):1634–1678
4. Bauer H, Rieder U (2005) Stochastic control problems with delay. Math Methods Oper Res 62(3):411–427
5. Bengio Y (2009) Learning deep architectures for AI. Found® Trends Mach Learn 2(1):1–127
6. Carleo G, Troyer M (2017) Solving the quantum many-body problem with artificial neural networks. Science 355(6325):602–606
7. Carmona R (2016) Lectures on BSDEs, stochastic control, and stochastic differential games with financial applications. SIAM
8. Carmona R, Laurière M (2019) Convergence analysis of machine learning algorithms for the numerical solution of mean field control and games: II—the finite horizon case. arXiv preprint arXiv:1908.01613
9. Chang MH, Pang T, Pemy M (2008) Finite difference approximation for stochastic optimal stopping problems with delays. J Ind Manag Optim 4(2):227
10. Chen L, Wu Z (2010) Maximum principle for the stochastic optimal control problem with delay and application. Automatica 46(6):1074–1080
11. Chen L, Wu Z (2012) Dynamic programming principle for stochastic recursive optimal control problem with delayed systems. ESAIM Control Optim Calc Var 18(4):1005–1026
12. Chen L, Wu Z, Yu Z (2012) Delayed stochastic linear-quadratic control problem and related applications. J Appl Math 6:66
13. Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using RNN encoder–decoder for statistical machine translation. arXiv preprint arXiv:1406.1078
14. Eiginan W, Han J, Jentzen A (2017) Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Commun Math Stat 5(4):349–380
15. Elsanosi I, Larssen B (2001) Optimal consumption under partial observations for a stochastic system with delay. Preprint series. Pure mathematics http://urn. nb. no/URN: NBN: no-8076
16. Elsanosi I, Øksendal B, Sulem A (2000) Some solvable stochastic control problems with delay. Stoch Int J Probab Stoch Process 71(1–2):69–89
17. Federico S (2011) A stochastic control problem with delay arising in a pension fund model. Finance Stoch 15(3):421–459
18. Fischer M, Nappo G (2008) Time discretisation and rate of convergence for the optimal control of continuous-time stochastic systems with delay. Appl Math Optim 57(2):177–206
19. Fischer M, Reiss M (2007) Discretisation of stochastic control problems for continuous time dynamics with delay. J Comput Appl Math 205(2):969–981
20. Fouque JP, Zhang Z (2020) Deep learning methods for mean field control problems with delay. Front Appl Math Stat 6:11
21. Gers FA, Schraudolph NN, Schmidhuber J (2002) Learning precise timing with LSTM recurrent networks. J Mach Learn Res 3:115–143
22. Gozzi F, Marinelli C, Savin S (2009) On controlled linear diffusions with delay in a model of optimal advertising under uncertainty with memory effects. J Optim Theory Appl 142(2):291–321
23. Gozzi F, Masiero F (2017) Stochastic optimal control with delay in the control I: solving the HJB equation through partial smoothing. SIAM J Control Optim 55(5):2981–3012
24. Gozzi F, di Roma S, Marinelli C (2005) Stochastic optimal control of delay equations arising in advertising models. Stoch Part Differ Equ Appl VII:133–148
25. Graves A (2013) Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850
26. Graves A, Mohamed Ar, Hinton G (2013) Speech recognition with deep recurrent neural networks. In: 2013 IEEE international conference on acoustics, speech and signal processing. IEEE, pp 6645–6649
27. Graves A, Schmidhuber J (2009) Offline handwriting recognition with multidimensional recurrent neural networks. In: Advances in neural information processing systems, pp 545–552
28. Guatteri G, Masiero F (2020) Stochastic maximum principle for problems with delay with general dependence on the past. arXiv preprint arXiv:2002.03953
29. Han J (2016) Deep learning approximation for stochastic control problems. arXiv preprint arXiv:1611.07422 (2016)
30. Han J, Hu R (2020) Deep fictitious play for finding Markovian Nash equilibrium in multi-agent games. In: Mathematical and scientific machine learning. PMLR, pp 221–245

31. Han J, Hu R, Long J (2020) Convergence of deep fictitious play for stochastic differential games. arXiv preprint arXiv:2008.05519
32. Han J, Jentzen A, Weinan E (2018) Solving high-dimensional partial differential equations using deep learning. Proc Natl Acad Sci 115(34):8505–8510
33. Han J, Lu J, Zhou M (2020) Solving high-dimensional eigenvalue problems using deep neural networks: a diffusion Monte Carlo like approach. J Comput Phys 423:109792
34. Han J, Ma C, Ma Z, Weinan E (2019) Uniformly accurate machine learning-based hydrodynamic models for kinetic equations. Proc Natl Acad Sci 116(44):21983–21991
35. Han J, Zhang L, Weinan E (2019) Solving many-electron Schrödinger equation using deep neural networks. J Comput Phys 66:108929
36. Hermann J, Schätzle Z, Noé F (2020) Deep-neural-network solution of the electronic Schrödinger equation. Nat Chem 12(10):891–897
37. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
38. Hu R (2021) Deep fictitious play for stochastic differential games. Commun Math Sci 19(2):325–353
39. Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. In: Proceedings of the international conference on learning representations (ICLR)
40. Kolmanovskiĭ VB, Shaĭkhet LE (1996) Control of systems with aftereffect, vol 157. American Mathematical Society
41. Kushner H (2008) Numerical methods for controlled stochastic delay systems. Springer
42. Kushner HJ (2006) Numerical approximations for stochastic systems with delays in the state and control. Stoch Int J Probab Stoch Process 78(5):343–376
43. Kydland FE, Prescott EC (1982) Time to build and aggregate fluctuations. Econom J Econom Soc 66:1345–1370
44. Larssen B (2002) Dynamic programming in stochastic control of systems with delay. Stoch Int J Probab Stoch Process 74(3–4):651–673
45. Larssen B, Risebro NH (2001) When are HJB-equations for control problems with stochastic delay equations finite dimensional? Preprint series. Pure mathematics http://urn. nb. no/URN: NBN: no-8076
46. Li K, Liu J (2018) Portfolio selection under time delays: a piecewise dynamic programming approach. Available at SSRN 2916481
47. Min M, Hu R (2021) Signatured deep fictitious play for mean field games with common noise. In: International conference on machine learning (ICML). PMLR. arXiv:2106.03272
48. Mohammed SEA (1984) Stochastic functional differential equations, vol 99. Pitman Advanced Publishing Program
49. Mohammed SEA (1998) Stochastic differential systems with memory: theory, examples and applications. In: Stochastic analysis and related topics VI. Springer, pp 1–77
50. Øksendal B, Sulem A (2000) A maximum principle for optimal control of stochastic systems with delay, with applications to finance. Preprint series. Pure mathematics http://urn. nb. no/URN: NBN: no-8076
51. Øksendal B, Sulem A, Zhang T (2011) Optimal control of stochastic delay equations and time-advanced backward stochastic differential equations. Adv Appl Probab 43(2):572–596
52. Pang T, Hussain A (2017) A stochastic portfolio optimization model with complete memory. Stoch Anal Appl 35(4):1–25
53. Peng S, Yang Z (2009) Anticipated backward stochastic differential equations. Ann Probab 37(3):877–902
54. Pfau D, Spencer JS, Matthews AG, Foulkes WMC (2020) Ab initio solution of the many-electron Schrödinger equation with deep neural networks. Phys Rev Res 2(3):033429
55. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagating errors. Nature 323(6088):533–536
56. Xuan Y, Balkin R, Han J, Hu R, Ceniceros HD (2021) Optimal policies for a pandemic: a stochastic game approach and a deep learning algorithm. In: Mathematical and scientific machine learning (MSML). arXiv:2012.06745