Quantitative Finance

ISSN: 1469-7688 (Print) 1469-7696 (Online) Journal homepage: www.tandfonline.com/journals/rquf20

Efficient pricing and hedging of high-dimensional American options using deep recurrent networks

Andrew S. Na & Justin W. L. Wan

To cite this article: Andrew S. Na & Justin W. L. Wan (2023) Efficient pricing and hedging of high-dimensional American options using deep recurrent networks, Quantitative Finance, 23:4, 631-651, DOI: 10.1080/14697688.2023.2167666

To link to this article: https://doi.org/10.1080/14697688.2023.2167666

Quantitative Finance, 2023
Vol. 23, No. 4, 631–651, https://doi.org/10.1080/14697688.2023.2167666

Efficient pricing and hedging of high-dimensional American options using deep recurrent networks

ANDREW S. NA * and JUSTIN W. L. WAN

David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada

(*Received 11 May 2022; accepted 6 January 2023; published online 30 January 2023*)

We propose a deep recurrent neural network (RNN) framework for computing prices and deltas of American options in high dimensions. Our proposed framework uses two deep RNNs, where one network learns the continuation price and the other learns the delta for each timestep. Our proposed framework yields prices and deltas for the entire spacetime, not only at a given point (e.g. $t = 0$). The computational cost of the proposed approach is linear in N, which improves on the quadratic time seen for feedforward networks that price American options. The computational memory cost of our method is constant in N, which is an improvement over the linear memory costs seen in feedforward networks. Our numerical simulations demonstrate these contributions and show that the proposed deep RNN framework is computationally more efficient than traditional feedforward neural network frameworks in time and memory.

Keywords: American option pricing; Deep recurrent neural networks; Stochastic differential equations; Delta hedging

JEL Classifications: C15, C45

1. Introduction

Neural network solutions to the American option pricing problem are some of the most studied topics in neural networks applications in finance. This is because the capabilities of neural networks to overcome the curse of dimensionality are attractive to practitioners who price and hedge American options. The pricing and hedging of American options present interesting practical and academic challenges, such as accurate and efficient pricing. However, many papers that look at neural networks in option pricing do not account for the extended training time required to train a network for each timestep to achieve good price estimates. This becomes more important when the trader on the trading floor needs to purchase an American option but needs an acceptable price for the contract. It also turns out the trader will require the derivative of the option price with respect to the underlying asset, called the delta of option, to hedge their position (Hull 2003). This is often done within a span of a few minutes and with limited computing resources. This is what motivated our research into recurrent neural network solutions to the American options problem.

There are numerous classical approaches to solve the American option problem such as the binomial tree method (Hull 2003), numerical solutions to partial differential equations (PDE) with free boundaries or penalty terms (Forsyth and Vetzal 2002, Achdou and Pironneau 2005, Duffy 2006), regression based methods (Tsitsiklis and Van Roy 1999, Longstaff and Schwartz 2001, Kholer *et al.* 2001), stochastic mesh methods (Broadie and Glasserman 2004), and so on. The PDE method is the most accurate method to compute option prices and deltas throughout the options horizon. However, when the dimension of an American option, i.e. the number of underlying assets, is too large numerical solution of PDEs is practically infeasible as the complexity grows exponentially with the dimension. When the dimension, d, is moderate (e.g. $3 < d \leq 20$), the regression-based Longstaff–Schwartz method (Longstaff and Schwartz 2001) is widely considered as the state-of-the-art approach for computing option prices. However, figure 1 in Bouchard and Warin (2012) shows that using such regressed values as the spacetime solution is inaccurate. This is reinforced in figure 8 in Chen and Wan (2021), which shows the clear advantages of using a neural network framework for pricing and hedging.

Outside of classical methods, there is a growing number of applications of neural networks on the American option problem. Some of the early approaches that solves the American option problem with neural networks use simple feedforward networks to approximate the price of the option for restricted

*Corresponding author. Email: andrew.na@uwaterloo.ca

number of assets ($d \leq 10$) as seen in Kholer *et al.* (2001) and Haugh and Kogan (2004). Later works extend the scope and incorporate chained deep neural networks to solve the American option problem (Han 2016, Fujii *et al.* 2017, Han and Jentzen 2017, Beck *et al.* 2018). The work of Guler and Laignelet (2019), follows the line of work of Han (2016) and Han and Jentzen (2017), but adds a regularizing term for added stability. The use of a neural network grid has been proposed by Sirignano and Spiliopoulos (2018). The work of Salvador *et al.* (2019) uses a neural network to learn option prices of the linear complementary problem of option pricing in an unsupervised manner. More recently, authors of Herrera *et al.* (2021) proposed a randomized reinforcement learning using recurrent networks to solve the American option problem. However, the methods mentioned do not provide methodology to determine the delta of an American option using neural networks. Delta and price approximation for all spacetime using a deep residual network was proposed by Chen and Wan (2021). In our work, we refer to the method of Chen and Wan (2021) as the deep residual learning (DRL) method. The work of Hure *et al.* (2020) provides theoretical convergence results for universal approximators. Note that RNNs are shown to be universal approximators in Schäfer and Zimmermann (2006). The DRL method solves many of the issues of some earlier work. One major drawback is that it is both computationally expensive in time and memory because the weights need to be trained and stored at each timestep.

In this paper, we propose to use two deep recurrent neural network regression framework to solve high-dimensional American style option problems. One of our networks is used to approximate the continuation price and the other is used to approximate the delta. This is in line with the general formulation outlined in Hure *et al.* (2020). In our proposed method, we introduce a least squares formulation that approximates the solution to the underlying backward stochastic differential equation (BSDE). Our method solves for price and delta in all of space and time. We outline the contributions of this paper:

- We extend the framework of Chen and Wan (2021) and propose a deep Recurrent neural network (RNN) framework that solves the American option problem. Using deep RNNs, the number of networks that need to be trained is reduced, which means we can speed up the computation time of high-dimensional option pricing. We show that our proposed framework achieves a better time and memory complexity for ($d > 20$). Our method computes option prices and deltas faster and with less memory compared to the work presented in Chen and Wan (2021).
- We introduce the approximation of deltas in all spacetime using our deep RNN framework, this has not been done by any other RNN option pricing paper to the best of our knowledge. The delta is computed using the pathwise derivative approach which is extended to general time t. We incorporate the domain knowledge of American options into our deep RNN framework to approximate accurate, time and memory efficient prices and deltas.

In contrast to the approach in Chen and Wan (2021), our work presents a few key differences.

- *Different Neural Network Architecture*: The most obvious difference in our approach is the use of *two* deep recurrent network. One deep recurrent network learns the continuation price and the other learns the delta. By using deep recurrent networks to approximate the continuation price and delta of American option through time, allows us to effectively reduce the number of networks that needs to be trained and stored to two networks as opposed to multiple networks.
- *Different Optimization criteria*: Our loss function directly learns the continuation price and delta functions at each timestep. This is then used to determine the value and delta of the option.

2. American options

In this paper, we use capital and lowercase letters to distinguish random and deterministic variables respectively. Furthermore, to distinguish vectors from scalars we will use an over line for vectors and bold text for matrices. Suppose we have a basket of d stocks; with price processes $\vec{S}(t) = [S_1(t), \ldots, S_d(t)]^\top \in \mathbb{R}^d$. Note that S is a random variable and $\vec{S}$ is a vector of random variables. Let $t \in [0, T]$ be the time up to maturity T, and let r be the interest rate. Let δ_i, $\mu_i = (r - \delta_i)$ and σ_i be the dividend, drift and volatility of stocks $i = 1, \ldots, d$. Let $\boldsymbol{\rho} \in \mathbb{R}^{d \times d}$ be the correlation matrix and we define the correlated random variable as

$$\mathrm{d}W_i(t) = \sum_{j=1}^{d} L_{i,j}\phi_j(t)\sqrt{\mathrm{d}t},$$

where $\phi_j(t) \sim N(0, 1)$ are i.i.d, and $\mathbf{L}$ is the Cholesky factor of $\boldsymbol{\rho}$, i.e. $\boldsymbol{\rho} = \mathbf{L}\mathbf{L}^\top$. Given a vector of initial prices, $\vec{s}(0) \in \mathbb{R}^d$, the price vector $\vec{S}(t)$ follows the dynamics given by geometric Brownian motion. For each element i, we have

$$\mathrm{d}S_i(t) = \mu_i S_i(t)\,\mathrm{d}t + \sigma_i S_i(t)\,\mathrm{d}W_i(t), \tag{1}$$

with initial condition $S_i(0) = s_i(0)$. For simplicity, we assume that the market has sufficient liquidity and we do not consider market frictions. Let $f(\vec{s}(t))$ be the payoff of the American option at the realized state $\vec{s}(t)$. This payoff function is a function of $g(\vec{s}(t))$, where $g(\vec{s}(t))$ varies depending on the payoff style, i.e. the commonly seen max call option has a payoff of the form $\max_{i=1,\ldots,d}\{s_i(t) - K\}$, and the general payoff has the form:

$$f(\vec{s}(t)) := \max\{g(\vec{s}(t)), 0\}. \tag{2}$$

The American option is more complicated to price than the European option with the same payoff because it can be exercised at any point t. To determine if we want to exercise or not, we need the continuation price.

Let $c(\vec{s}(t), t)$ be the continuation price of the American call option, i.e. the discounted payoff of the option when it is not

exercised at time t at price $\vec{s}(t)$. We denote the optimal stopping time by $\tau \in [t, T]$, which is the best time to exercise the option between time t and maturity T. Then the continuation price can be written as

$$c(\vec{s}(t), t) = \max_{\tau \in [t,T]} \mathbb{E}[e^{-r(\tau - t)} f(\vec{S}(\tau)) \mid \vec{S}(t) = \vec{s}(t)]. \quad (3)$$

Note that we exercise the option when we determine that the payoff is greater than or equal to the continuation price. Then the value of the American style option is given by

$$\begin{aligned} v(\vec{s}(t), t) &= \max\{f(\vec{s}(t)),\ c(\vec{s}(t), t)\} \\ &= \begin{cases} c(\vec{s}(t), t) & c(\vec{s}(t), t) \geq f(\vec{s}(t)) \\ f(\vec{s}(t)) & \text{otherwise} \end{cases}, \end{aligned} \quad (4)$$

The optimal exercise boundary is the price of the asset such that

$$c(\vec{s}(t), t) = f(\vec{s}(t)). \quad (5)$$

In practical applications of hedging, we are interested in evaluating the delta of the American option. We denote the vector of delta as, $\nabla v(\vec{s}(t), t) = [\frac{\partial v}{\partial s_1}(\vec{s}(t), t), \ldots \frac{\partial v}{\partial s_d}(\vec{s}(t), t)]^\top$. We compute the delta of the American style option as

$$\nabla v(\vec{s}(t), t) = \begin{cases} \nabla c(\vec{s}(t), t) & c(\vec{s}(t), t) \geq f(\vec{s}(t)) \\ \nabla f(\vec{s}(t)) & \text{otherwise} \end{cases}. \quad (6)$$

Our goal is to compute both the option price $v(\vec{s}(t), t)$ and delta $\nabla v(\vec{s}(t), t)$ on the entire space-time in a time and memory efficient way.

3. BSDE formulation of the American option pricing problem

3.1. BSDE formulation

The American option pricing problem is often formulated as a Hamilton–Jacobi–Bellman (HJB) equation. This non-linear PDE is difficult to solve as we need to solve the PDE and the optimal control. Instead of working with the HJB equation, we reformulate the American option pricing problem into a BSDE with the following theorem:

THEOREM 3.1 (BSDE formulation) *Assume that an American option is not exercised at time* $[t, t + dt]$. *Then the continuation price of the American option at time* t *satisfies the following BSDE:*

$$\mathrm{d}c(\vec{S}(t)) = rc(\vec{S}(t), t)\,\mathrm{d}t + \sum_{i=1}^{d} \sigma_i \frac{\partial c}{\partial s_i}(\vec{S}(t), t)\,\mathrm{d}W_i(t), \quad (7)$$

where $\vec{S}(t)$ *satisfies* (1) *and* r, σ_i *and* $dW_i(t)$ *are defined as in* (1).

Proof Interested readers to the proof are referred to El Karoui *et al.* (1997), which uses Ito's lemma. ■

The significance of the BSDE formulation is two fold; one, it couples $c(\vec{S}(t), t)$ and $\nabla c(\vec{S}(t), t)$. If the approximation for the price is correct, then the approximation for the delta will also be correct, which allows us to perform a complete hedging process (Chen and Wan 2021). The second significance of the BSDE formulation is that it allows us to design a less expensive neural network compared to the HJB PDE (Sirignano and Spiliopoulos 2018). This is because using the HJB PDE requires us to compute and store the Hessian tensor, which is an $\mathcal{O}(Md^2)$ tensor, where M is the number of samples used to train the neural network. Compared to the HJB formulation, the BSDE formulation only requires the Jacobian tensor of size $\mathcal{O}(Md)$.

3.2. Discretization of BSDE

In this paper, we use the Euler method as in Hure *et al.* (2020) and Chen and Wan (2021) to simulate SDE (1). Let $m = 1, \ldots, M$ be the indices of simulation paths and $n = 1, \ldots, N$ be the indices of the discrete timesteps from 0 to T. Let $\Delta t = T/N$ be the timestep size and let $t^n = n\Delta t$. We discretize $\mathrm{d}W_i(t)$ as

$$(\Delta W_i)_m^n = \sum_{j=1}^{d} L_{i,j} (\phi_j)_m^n \sqrt{\Delta t}.$$

We discretize (1) as

$$(S_i)_m^0 = (s_i)_m^0, \quad i = 1, \ldots, d \quad (8)$$

$$(S_i)_m^{n+1} = (1 + (r - \delta_i)\Delta t)(S_i)_m^n + \sigma_i (S_i)_m^n (\Delta W_i)_m^n, \quad i = 1, \ldots, d. \quad (9)$$

We can discretize the payoff function at $n + 1$ as

$$f(\vec{S}_m^{n+1}) = \max\{g(\vec{S}_m^{n+1}), 0\}.$$

For $n = N - 1, \ldots, 0$, we discretize (7) as

$$\begin{aligned} c^{n+1}(\vec{S}_m^{n+1}) &= (1 + r\Delta t) c^n(\vec{S}_m^n) \\ &\quad + \sum_{i=1}^{d} \sigma_i (S_i)_m^n \frac{\partial c^n}{\partial s_i}(\vec{S}_m^n)(\Delta W_i)_m^n, \end{aligned} \quad (10)$$

where $c^n(\vec{S}_m^{n+1})$ is the discrete approximation to the continuation price $c(\vec{S}_m^n, t^n)$, and $\frac{\partial c^n}{\partial s_i}(\vec{S}_m^{n+1})$ the discrete approximation to the delta $\frac{\partial c}{\partial s_i}(\vec{S}_m^n, t^n)$.

We can then find the exercise boundary, $\mathcal{E}_m^{n+1}$, defined as

$$\mathcal{E}_m^{n+1} = \begin{cases} 1, & c^{n+1}(\vec{S}_m^{n+1}) \geq f(\vec{S}_m^{n+1}) \\ 0, & \text{otherwise.} \end{cases}$$

Finally, the value of the American option is discretized as

$$v(\vec{S}_m^{n+1}, t^{n+1}) = (1 - \mathcal{E}_m^{n+1}) f(\vec{S}_m^{n+1}) + \mathcal{E}_m^{n+1} c^{n+1}(\vec{S}_m^{n+1}),$$

with terminal condition:

$$v(\vec{S}_m^N, t^N) = f(\vec{S}_m^N). \quad (11)$$

For high dimensions when d is large, solving (10) directly can be time consuming or even intractable. Instead, we use (10) to

construct a least squares formulation which will be presented in section 3.3. This is then used to compute the option price (4) and delta (6).

3.3. Least squares formulation to the BSDE

Consider the nth timestep t^n. For easier exposition, we will drop the dependence on $\vec{S}_m^n$ to simplify notation. Let y^n and $\frac{\partial y^n}{\partial s_i}$ be the approximation for the discrete continuation price c^n and the discrete delta $\frac{\partial c^n}{\partial s_i}$, respectively. We let $\nabla y^n = [\frac{\partial y^n}{\partial s_1}, \ldots, \frac{\partial y^n}{\partial s_d}]^\top$ and $\nabla c^n = [\frac{\partial c^n}{\partial s_1}, \ldots, \frac{\partial c^n}{\partial s_d}]^\top$. To derive the least-squares formulation of the BSDE (10), we first define the approximation error as

$$e^{n+1} = c^{n+1} - y^{n+1}.$$

To approximate c^{n+1}, y^{n+1} must satisfy (10). In other words, y^{n+1} can be defined as

$$y^{n+1} = (1 + r\Delta t)y^n + \sum_{i=1}^{d} \sigma_i \frac{\partial y^n}{\partial s_i} (\Delta W_i)_m^n.$$

Now the error can be written as

$$e^{n+1} = (1 + r\Delta t)(c^n - y^n) + \sum_{i=1}^{d} \sigma_i \left(\frac{\partial c^n}{\partial s_i} - \frac{\partial y^n}{\partial s_i} \right) (\Delta W_i)_m^n.$$

Taking the expected value, the error equation becomes

$$\mathbb{E}[e^{n+1}] = \mathbb{E}\left[(1 + r\Delta t)(c^n - y^n) + \sum_{i=1}^{d} \sigma_i \left(\frac{\partial c^n}{\partial s_i} - \frac{\partial y^n}{\partial s_i} \right) (\Delta W_i)_m^n \right].$$

Discounting the process, we have that $\mathbb{E}[e^{n+1}] = (1 + r\Delta t)\mathbb{E}[e^n]$. Consequently, we have

$$\mathbb{E}[e^n] = \mathbb{E}\left[(c^n - y^n) + \frac{1}{1 + r\Delta t} \sum_{i=1}^{d} \sigma_i \left(\frac{\partial c^n}{\partial s_i} - \frac{\partial y^n}{\partial s_i} \right) (\Delta W_i)_m^n \right].$$

Then the mean squared error at n is given by

$$\begin{aligned}
\mathbb{E}[|e^n|^2] &= \mathbb{E}\left[\left| (c^n - y^n) + \frac{1}{(1 + r\Delta t)} \times \sum_{i=1}^{d} \sigma_i \left(\frac{\partial c^n}{\partial s_i} - \frac{\partial y^n}{\partial s_i} \right) (\Delta W_i)_m^n \right|^2 \right] \\
&= \mathbb{E}\left[\left| (c^n - y^n) + \frac{\sqrt{\Delta t}}{(1 + r\Delta t)} \sum_{i=1}^{d} \sum_{j=1}^{d} L_{i,j}\sigma_i \times \left(\frac{\partial c^n}{\partial s_i} - \frac{\partial y^n}{\partial s_i} \right) (\phi_j)_m^n \right|^2 \right] \\
&\approx \mathbb{E}\left[|(c^n - y^n)|^2 + \frac{\Delta t}{(1 + r\Delta t)^2} \times \left| \sum_{i=1}^{d} \sum_{j=1}^{d} L_{i,j}\sigma_i \left(\frac{\partial c^n}{\partial s_i} - \frac{\partial y^n}{\partial s_i} \right) (\phi_j)_m^n \right|^2 \right].
\end{aligned}$$

Finally, the loss function at n is given by

$$\mathcal{L}[y^n, \nabla y^n] = \mathbb{E}[|c^n - y^n|^2] + \frac{\Delta t}{(1 + r\Delta t)^2} \mathbb{E} \times \left[\left| \sum_{i=1}^{d} \sum_{j=1}^{d} L_{i,j}\sigma_i \left(\frac{\partial c^n}{\partial s_i} - \frac{\partial y^n}{\partial s_i} \right) \right|^2 \right], \quad (12)$$

where $\mathbb{E}[|(\phi_j)_m^n|^2] = 1$. Note that the authors of Hure *et al.* (2020) show that the loss function (12) converges to the viscosity solution of the original BSDE (7).

We remark that we use the loss (12) over the loss of Chen and Wan (2021) for two reasons:

- The loss (12) does not require integration over time steps, reducing the need to store the results of each time step. This also allows us to train the recurrent network for all time in one step.
- In our approach, the delta is approximated by a neural network, which allows us to compute delta at $t = 0$ for any stopping time τ using the pathwise derivative method without having to worry if $\partial S/\partial s_0$ is evaluable.

Given the loss function $\mathcal{L}$ we compute the approximations y^n and ∇y^n by minimizing (12):

$$\{y^{n*}, \nabla y^{n*}\} = \arg\min_{y^n, \nabla y^n} [\mathcal{L}[y^n, \nabla y^n]]. \quad (13)$$

Since the BSDE is a backward process, we start our computation at $n = N$, where the value of the option is given by the payoff (2). This means that the risk-neutral price of the option at $n = N - 1$, must be the discounted value of the option. To train our network, we must find the continuation price and delta at time n such that the value of the option at n is given by (4).

3.4. Continuation price and delta at time t

To compute the minimizers y^{n*} and $\frac{\partial y^{n*}}{\partial s_i}$, we need to determine the continuation price, c^n, and delta, $\frac{\partial c^n}{\partial s_i}$, in (12). To determine c^n, we will solve the optimization problem (3). To determine $\frac{\partial c^n}{\partial s_i}$ is not as straight forward. Our derivation follows the work of Broadie and Glasserman (1996) which shows a framework for estimating the delta of a European call options using the pathwise derivative. This was later extended to the American options in Thom (2009) and applied to neural networks in Chen and Wan (2021).

We see in Broadie and Glasserman (1996) that when the underlying asset evolves following the process (1), $\frac{\partial S_i}{\partial s_0} = \frac{S_i}{s_0}$. We can extend this notion to the increment $[t, \tau]$ under (1) since:

$$\frac{\partial S_i(\tau)}{\partial s_i(t)} = e^{(\mu - \frac{\sigma^2}{2})(\tau - t) + \sigma\sqrt{(\tau - t)}W_t} = \frac{S_i(\tau)}{s_i(t)}. \quad (14)$$

Finally, given τ from (3), we can define the delta of the continuation price as

$$\frac{\partial c}{\partial s_i}(\vec{s}_i(t)) = \mathbb{E}\left[e^{-r(\tau - t)} \frac{\partial f}{\partial S_i}(\vec{S}(\tau)) \frac{\partial S_i(\tau)}{\partial s_i(t)} \right], \quad (15)$$

where $\frac{\partial S_i(\tau)}{\partial s_i(t)}$ is determined from (14). The pathwise derivative approach may not be applicable if the derivative $\partial S_i/\partial s_i$ cannot be evaluated. This may occur if the underlying asset does not follow the process (1). However, we are always able to evaluate (15). This is because our framework uses a deep RNN to approximate the continuation price of the option, and using autodifferentiation on the computation graph for the input variable $S_i(\tau)$ with respect to the realised asset price $s_i(t)$ we can always evaluate the derivative $\partial S_i(\tau)/\partial s_i(t)$.

Note, if we denote τ as $\tau = \tilde{n}\Delta t$, then the continuation price can be discretized using (3) as

$$c^n \approx \mathbb{E}[e^{-r(\tilde{n}-n)\Delta t} f^{\tilde{n}}]. \tag{16}$$

Using (14) the delta can be discretized as

$$\nabla c^n \approx \mathbb{E}\left[e^{-r(\tilde{n}-n)\Delta t} \nabla f^{\tilde{n}} \frac{S^{\tilde{n}}}{S^n}\right]. \tag{17}$$

4. Deep recurrent neural network formulation

To solve (13) is non-trivial. This problem can be solved using some parameterized function, such as polynomials, to approximate y^n, which allows us to solve the optimization problem in the parameter space. Classical methods such as the Longstaff–Schwartz method are built on this principle. More modern methods as seen in Herrera *et al.* (2021) can be used to alleviate some of this by using neural networks to learn such functions. We note that methods such as the Longstaff–Schwartz method and the method proposed by Herrera *et al.* (2021) do not solve the BSDE problem outlined in (7).

A main drawback of the deep neural network approach in Chen and Wan (2021) is that they require a feedforward neural network for every time step. It may be somewhat improved by skipping every J time timsteps, but it still requires to train N/J networks where N is the total number of timesteps. When N is large, training time can be very costly.

Our proposed approach is to use two deep recurrent networks to compute the approximate continuation price y^n and delta ∇y^n. The advantage of our deep Recurrent network approach is that we only require two networks. This leads to tremendous savings in time and memory for large N.

4.1. Sequence of neural networks

The main advantage of using a Recurrent network over traditional feedforward networks is that the network is able to learn time-dependent dynamic problems (Elman 1990). There exist many different types of RNNs and we refer the reader to Hochreiter and Schmidhuber (1997) and Cho *et al.* (2014). In our approach, we use the gated recurrent unit (GRU) of Cho *et al.* (2014) to solve the American option pricing problem.

In this paper, we present two recurrent networks, rolled out through time, $\{\mathcal{F}^n(\mathbf{s};\Omega)\,|\,n = N-1,\ldots,0\}$ and $\{\mathcal{G}^n(\mathbf{s};\Theta)\,|\,n = N-1,\ldots,0\}$, where $\{\Omega,\Theta\}$ are trainable set of parameters. We will first present our recurrent network used to approximate the price, $\{y^n(\mathbf{s};\Omega)\,|\,n = N-1,\ldots,0\}$.

The arbitrage free price of an option should differ from the previous price and delta by a function with magnitude $\mathcal{O}(\Delta t)$. As shown in Chen and Wan (2021), using a linear combination of the neural network output and the discounted value provides an estimate to the continuation price with lower variation, which improves the accuracy of regression.

We note that our problem is solved backwards in time, and we use 'previous', 'current' and 'next' to refer to the $(n+1)$th, nth, and $(n-1)$th timesteps respectively. We define the user-defined coefficient $\alpha \in [0,1]$. The coefficient α determines how much of the approximation is given by the discounted value and the deep RNN approximation $\mathcal{F}^n(s;\Omega)$. Mathematically, we approximate the continuation price as

$$\begin{aligned} y^N(\mathbf{s}) &= f^N(\mathbf{s}) \\ y^n(\mathbf{s};\Omega) &= (1-\alpha)e^{-r(N-n)\Delta t} f^N + \alpha\mathcal{F}(\mathbf{s};\Omega), \\ n &= N-1,\ldots,0. \end{aligned} \tag{18}$$

The set of parameters Ω includes all the weights and biases used in algorithm 1.

Algorithm 1 Continuation price network using a single GRU unit

Require: $\mathbf{h}^0, \mathbf{W}_e^n, \mathbf{W}_o^n, \mathbf{f}^N$
  for $n = N-1,\ldots,0$ **do**
    GRU $\leftarrow$ (19)–(22)
    $\mathbf{e}^n \leftarrow \text{swish}(\mathbf{W}_e^{n\top}\text{GRU} + b_e)$
    $\mathbf{o}_p^n \leftarrow \text{softplus}(\mathbf{W}_o^\top \mathbf{e}^n + b_o)$
    $\mathcal{F}(s;\Omega) \leftarrow \mathbf{o}_p^n$
    $\mathbf{y}^n \leftarrow (1-\alpha)e^{-r(N-n)\Delta t}\mathbf{f}^N + \alpha\mathcal{F}(\mathbf{s};\Omega)$

Our deep RNN, $\mathcal{F}(\mathbf{s};\Omega)$, is shown in figure 1. We parameterize it by an L-layer deep RNN with GRU units with a linear embedding layer and output layer, as shown in figure 2. A typical GRU unit is illustrated in figure 3. Let the input of the deep RNN be $\mathbf{X} \in \mathbb{R}^{M\times 2d}$ and previous hidden state be $\mathbf{h}^{n-1} \in \mathbb{R}^{M\times 2d}$. The GRU unit is composed of the update gate $\mathbf{z}^n$ which decides if the a new hidden state should be stored, the reset gate $\mathbf{r}^n$, decides which information to keep from the previous sequence, the new hidden state $\hat{\mathbf{h}}^n$ stores potential information from a filtered previous state and current state. Mathematically, the GRU is given by the set of equations

$$\mathbf{r}^n = \hat{\sigma}(\mathbf{W}_r^{n\top}[\mathbf{h}^{n+1}, \mathbf{X}^n] + b_r) \tag{19}$$

$$\mathbf{z}^n = \hat{\sigma}(\mathbf{W}_z^{n\top}[\mathbf{h}^{n+1}, \mathbf{X}^n] + b_z) \tag{20}$$

$$\hat{\mathbf{h}}^n = \tanh(\mathbf{W}_{\hat{h}}^{n\top}[\mathbf{r}^n \odot \mathbf{h}^{n+1}, \mathbf{X}^n] + b_{\hat{h}}) \tag{21}$$

$$\mathbf{h}^n = (1-\mathbf{z}^n)\odot\mathbf{h}^{n+1} + \mathbf{z}^n \odot \hat{\mathbf{h}}^n, \tag{22}$$

where $\hat{\sigma}$, tanh are the typical sigmoid and hyperbolic tangent activation functions seen in Goodfellow *et al.* (2016). The parameters $\mathbf{W}_r^n, \mathbf{W}_z^n, \mathbf{W}_h^n \in \mathbb{R}^{d\times d}$ are trainable weights and $b_r, b_z, b_h \in \mathbb{R}^d$ are trainable offsets.

In the following part, we temporarily use a square bracket subscript for the layer index $l = 1,\ldots,L$. We construct the L-layer deep RNN for a continuation price as

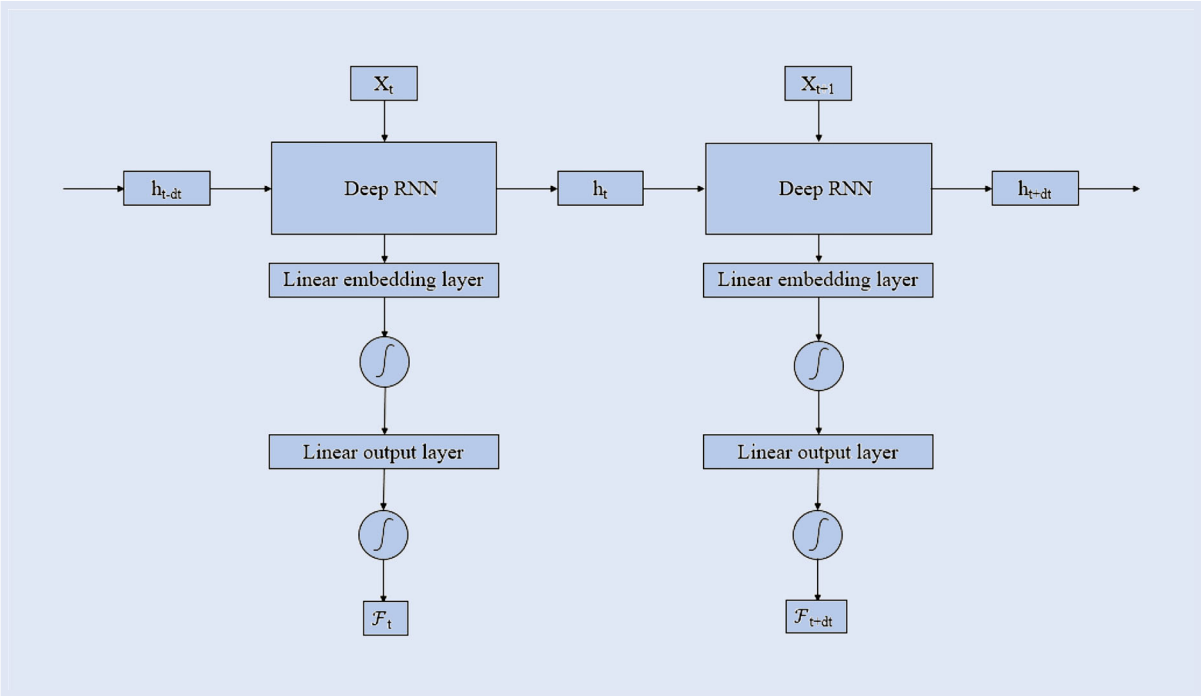

Figure 1. Deep recurrent network for price rolled out through time. Our network is composed of a recurrent unit, an embedding linear layer and an output linear layer. Takes in a matrix of inputs $\mathbf{x}$, hidden information $\mathbf{h}$ and outputs $\mathcal{F}$.

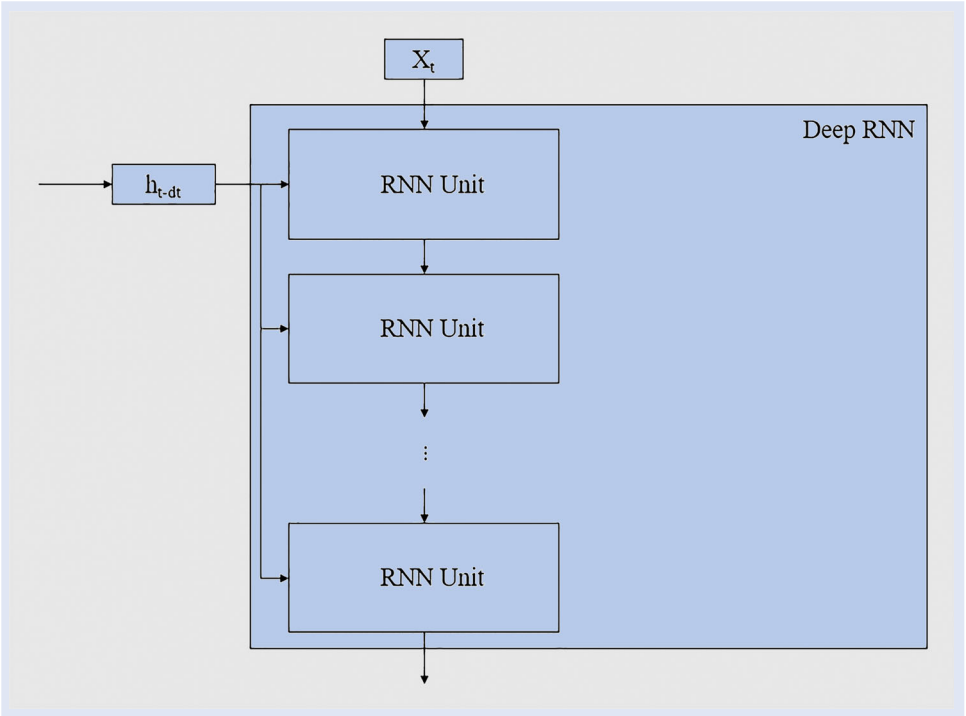

Figure 2. Stacked RNN units within the deep RNN.

- For layers, $l = 1$:

$$\mathbf{r}^n_{[l]} = \hat{\sigma}(\mathbf{W}_r^{n\top}[\mathbf{h}^{n+1}, \mathbf{X}^n] + b_r)$$
$$\mathbf{z}^n_{[l]} = \hat{\sigma}(\mathbf{W}_z^{n\top}[\mathbf{h}^{n+1}, \mathbf{X}^n] + b_z)$$
$$\hat{\mathbf{h}}^n = \tanh((\mathbf{W}_{\hat{h}}^{n\top})_{[l]}[\mathbf{r}^n_{[l]} \odot \mathbf{h}^{n+1}, \mathbf{X}^n] + b_{\hat{h}})$$
$$\mathbf{h}^n_{[l]} = (1 - \mathbf{z}^n_{[l]}) \odot \mathbf{h}^{n+1} + \mathbf{z}^n_{[l]} \odot \hat{\mathbf{h}}^n_{[l]}.$$

- For layers, $l = 2, \ldots, L$:

$$\mathbf{r}^n_{[l]} = \hat{\sigma}(\mathbf{W}_r^{n\top}[\mathbf{h}^{n+1}, \mathbf{h}^n_{[l-1]}] + b_r)$$
$$\mathbf{z}^n_{[l]} = \hat{\sigma}(\mathbf{W}_z^{n\top}[\mathbf{h}^{n+1}, \mathbf{h}^n_{[l-1]}] + b_z)$$
$$\hat{\mathbf{h}}^n = \tanh((\mathbf{W}_{\hat{h}}^{n\top})_{[l]}[\mathbf{r}^n_{[l]} \odot \mathbf{h}^{n+1}, \mathbf{h}^n_{[l-1]}] + b_{\hat{h}})$$
$$\mathbf{h}^n_{[l]} = (1 - \mathbf{z}^n_{[l]}) \odot \mathbf{h}^{n+1} + \mathbf{z}^n_{[l]} \odot \hat{\mathbf{h}}^n_{[l]}.$$

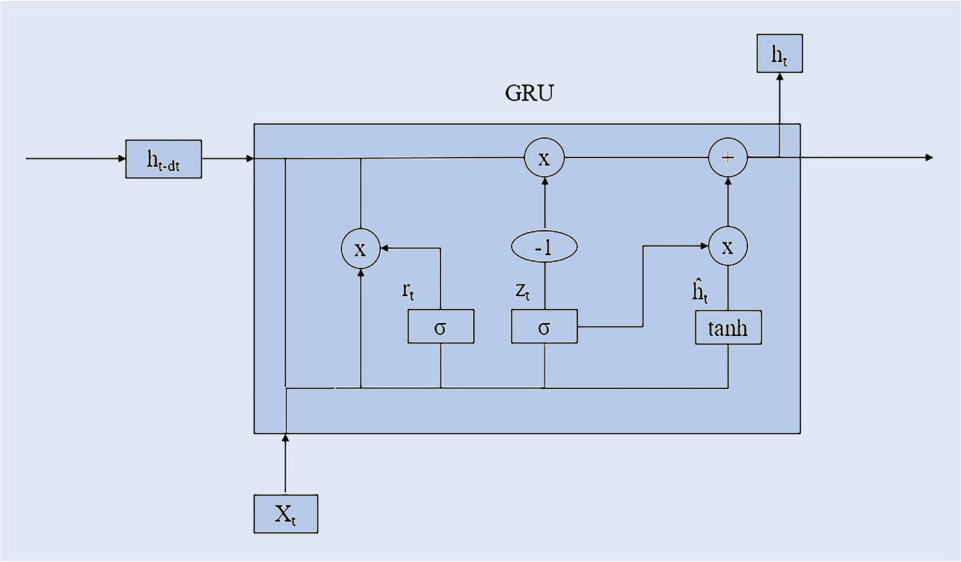

Figure 3. Typical GRU cell with gates. Operations in circles are element wise operations. The σ and tanh are the sigmoid and hyperbolic tangent activation functions as seen in Goodfellow *et al.* (2016).

- For the embedding layer:

$$\mathbf{e}^n = \text{swish}(\mathbf{W}_e^{n\top}\mathbf{h}_{[L]}^n + b_e^n),$$

  where $\mathbf{W}_e^n \in \mathbb{R}^{d\times d}$ are trainable weights and $b_e^n \in \mathbb{R}^d$ is the trainable bias.
- For the output layer:

$$\mathcal{F}(\mathbf{s};\Omega^n) = \text{softplus}(\mathbf{W}_{\mathcal{F}}^{n\top}\mathbf{e}^n + b_{\mathcal{F}}^n),$$

  where $\mathbf{W}_{\mathcal{F}}^n \in \mathbb{R}^{d\times d}$ are trainable weights and $b_{\mathcal{F}}^n \in \mathbb{R}^d$ is the trainable bias. The swish activation function is given in (24).

We construct the deep RNN to compute the deltas $\mathcal{G}(\mathbf{s};\Theta)$, in a similar way to $\mathcal{F}(\mathbf{s};\Omega)$. The main difference between the two networks is in the output layers, where

$$\mathcal{G}(\mathbf{s};\Theta^n) = \hat{\sigma}\,(\mathbf{W}_{\mathcal{G}}^{n\top}\mathbf{e}^n + b_{\mathcal{G}}^n),$$

where $\mathbf{W}_{\mathcal{G}}^n \in \mathbb{R}^{d\times d}$ are trainable weights and $b_{\mathcal{G}}^n \in \mathbb{R}^d$ is the trainable bias. Note that the sigmoid function is used here as it is the derivative of the softplus function with respect to the input.

To compute deltas, we define the coefficient $\beta \in [0,1]$. β, like α determines how much of the approximation is given by the discounted delta and the deep RNN approximation $\mathcal{G}(s;\Theta)$. Mathematically, we approximate the option delta as

$$\begin{aligned}\nabla y^N(\mathbf{s}) &= \nabla f^N(\mathbf{s})\\ \nabla y^n(\mathbf{s};\Theta) &= (1-\beta)e^{-r(N-n)\Delta t}\nabla f^N\frac{S^N}{S^n} + \beta\mathcal{G}(\mathbf{s};\Theta),\\ n &= N-1,\ldots,0.\end{aligned} \tag{23}$$

The set of parameters Θ includes all the weights and biases used in algorithm 2.

Our deep Recurrent networks are structured as a one way stacked recurrent units as shown in figure 2. For $n = N-1, \ldots, 0$ our input, $\mathbf{X}^n$ passes through the first Recurrent unit. The subsequent Recurrent units use the output of the previous unit as the input of the next, while the previous hidden state, $\mathbf{h}^{n+1}$, is passed to all the recurrent units. The output of the deep RNN is passed through a linear embedding layer, an activation function, then a linear output layer, and a final activation function which is chosen based on the domain information of the problem.

Algorithm 2 Delta network using a single GRU unit

Require: $\mathbf{h}_\Delta^0, \mathbf{W}_e^n, \mathbf{W}_o^n, \nabla\mathbf{f}^N$
  for $n = N-1,\ldots,0$ **do**
    GRU $\leftarrow$ (19)–(22)
    $\mathbf{e}^n \leftarrow \text{swish}(\mathbf{W}_e^{n\top}\text{GRU} + b_e)$
    $\mathbf{o}_\Delta^n \leftarrow \hat{\sigma}\,(\mathbf{W}_o^\top\mathbf{e}^n + b_o)$
    $\mathcal{G}(s;\Theta) \leftarrow \mathbf{o}_\Delta^n$
    $\nabla\mathbf{y}^n \leftarrow (1-\beta)e^{-r(N-n)\Delta t}\nabla\mathbf{f}^N\frac{\mathbf{s}^N}{\mathbf{s}^n} + \beta\mathcal{G}(\mathbf{s};\Theta)$

4.2. Recurrent unit selection

A recurrent network is a sequential learning model based on neural networks (Jordan 1990). The key differentiating point of a recurrent network as opposed to a neural network is the use of hidden states (Elman 1990). The hidden states of a recurrent network, allows us to retain information as the network is trained over a sequence.

The GRU is an extension of RNN and alleviates the problem of vanishing/exploding gradients (Cho *et al.* 2014). This is done by feeding through a combination of the previous state and the current state. The GRU is a variation of a popular LSTM (Long short term memory) (Hochreiter and Schmidhuber 1997) and uses gates (activation functions) to transform the input. The GRU reduces the complexity of the LSTM by reducing the number of gates through which information passes.

A typical GRU is shown in figure 3. The linear structure of the GRU output allows us to generalize the payoff and delta of the option with better accuracy than non-linear structure of the LSTM. Past studies have shown that the linear structure of the output preserves the most information, as observed in Chen and Wan (2021) and studied in He *et al.* (2016). We found that the non-linear output of LSTMs can force the output to achieve extreme negative values, which is not ideal for pricing. Another reason is that it has fewer parameters to optimize as it only has three gates over the four gates seen in LSTMs. This reduces the number of operations we need to perform and the number of parameters that need to be optimized, which further increases our computational efficiency. We refer interested readers to Cho *et al.* (2014) for a more detailed review of GRUs and (Hochreiter and Schmidhuber 1997) for a review of LSTMs.

4.3. *Activation function selection for output and embedding layers*

The choice of suitable activation functions leads to better predictive capabilities for any network (Goodfellow *et al.* 2016). As shown in figure 1, our network adds two linear layers to the output of the deep recurrent network, a linear embedding layer, and a linear output layer. We denote the embedding layer as $\mathbf{e}^n$, $\mathbf{o}_p^n$ for the output layer of the price approximation, and $\mathbf{o}_\Delta^n$ for the output layer of the delta approximation. For continuation prices, the natural activation function used for the output layer is the popular rectilinear unit (ReLU) activation function (Goodfellow *et al.* 2016). The ReLU function has the form:

$$\text{ReLU}(x) = \max\{x, 0\},$$

which has the same form as our payoff function $f(x)$. However, the ReLU activation function suffers from vanishing gradients if a network is trained for too many iterations. Instead, we use the softplus activation function given by

$$\text{softplus}(x, \kappa) = \frac{1}{1 + e^{(-\kappa x)}}.$$

This activation function approaches the ReLU activation as $\kappa \to \infty$. This gives us a good smoothing of the payoff and it's exponential behaviour as it evolves through time. For the output layers of the network that approximate the option deltas, we use a sigmoid activation function. This selection is natural as the sigmoid function is the derivative of the softplus function.

For the embedding layer, we use the swish activation function given by

$$\text{swish}(x) = x\hat{\sigma}(x). \tag{24}$$

The swish activation function is empirically shown to perform better than the ReLU function, as shown in Ramachandran *et al.* (2017) and helps our method reach more accurate solutions.

4.4. *Feature selection and network summary*

Feature selection in neural network has a significant impact on the accuracy of the network model (Goodfellow *et al.* 2016), the correct choice of inputs in any regression problem allows for good predictions and approximations. We use the asset price, and $g(\mathbf{s})$ from (2). Chen and Wan (2021) found that $g(\mathbf{s})$ performs better than using the payoff, $f(\mathbf{s})$ due to the additional information it carries. We concatenate the inputs, but we do not perform normalization. Our concatenated input matrix is

$$\mathbf{x} = [\mathbf{s}, g(\mathbf{s})]^T \in \mathbb{R}^{M\times 2d}.$$

Unlike a traditional feedforward network, the Recurrent network uses the hidden state to carry information through time. In practical applications of Recurrent networks the hidden state is initialized as zero or given some random numbers following a uniform or normal distribution. In our approach, we initialize the hidden state using $\mathbf{f}^N$ for $\mathcal{F}(\mathbf{s};\Omega)$ and $\nabla\mathbf{f}^N$ for $\mathcal{G}(\mathbf{s};\Theta)$. We pad the initial hidden state with itself to match the number of GRU layers.

To summarize sections 4.2–4.4, we can express our price approximating network for $n = N-1,\ldots,0$ by algorithm 1 and similarly we approximate the delta of the option we use algorithm 2.

4.5. *Training and evaluation*

Consider training a recurrent network, recurrent networks can be trained with a full sequence of events at once. For clarity, in this paper we will look at how to construct the data required for training at each timestep, n and all simulations m.

Case: $\mathbf{n} = \mathbf{N}$. At the final timestep N, we let $y^N(\mathbf{S}^N) = f^N(\mathbf{S}^N)$ and $\nabla y^N(\mathbf{S}^N) = \nabla f^N(\mathbf{S}^N)$. We consider a smoothed payoff function given by using a scaled softplus activation function with a user defined parameter κ given by

$$y^N(\mathbf{S}^N) = f_\kappa^N(\mathbf{S}^N) \equiv \frac{1}{\kappa}\ln(1 + e^{\kappa g(\mathbf{S}^N)}), \tag{25}$$

where $g(\mathbf{S}_m^N)$ is the same as in (2), let $\nabla g(\mathbf{S}^N) = [\frac{\partial g(\mathbf{S}^N)}{\partial \vec{S}_1^N}, \ldots, \frac{\partial g(\mathbf{S}^N)}{\partial \vec{S}_d^N}]$, then the derivative of the smoothed payoff is

$$\nabla y^N(\mathbf{S}^N) = \nabla f_\kappa^N(\mathbf{S}^N) \equiv \hat{\sigma}(\kappa g(\mathbf{S}^N))\nabla g(\mathbf{S}^N). \tag{26}$$

The hidden states are initialized as $f_\kappa^N(\mathbf{S}^N)$ for $\mathcal{F}(\mathbf{S}^N;\Omega)$ and $\nabla f_\kappa^N(\mathbf{S}^N)$ for $\mathcal{G}(\mathbf{S}^N;\Theta)$. The initial value of the continuation price is given by $c^N(\mathbf{S}_m^N) = f_\kappa^N(\mathbf{S}_m^N)$, which is the payoff at $t = T$. Similarly, the initial delta is given by $\nabla c^N(\mathbf{S}_m^N) = \nabla f_\kappa^N(\mathbf{S}_m^N)$. The initial hidden states are given by

$$\mathbf{h}^N = [f_\kappa^N(\mathbf{S}^N)],$$

for the pricing network and

$$\mathbf{h}^N = [\nabla f_\kappa^N(\mathbf{S}^N)],$$

for the delta network.

Case: $\mathbf{n} = \mathbf{N}-\mathbf{1},\ldots,\mathbf{0}$. We use the deep RNN framework to compute the continuation price and delta of the option at

timesteps $n = N-1,\ldots,0$. We construct the training input as

$$\mathbf{X}^n = [\mathbf{S}^n, g(\mathbf{S}^n)]^T,.$$

Then we determine the optimal stopping time $\tilde{n}$ as

$$\tilde{n} = \max_{\tilde{n}\in[N,n]} \{e^{-r(N-n)\Delta t}\mathbb{E}[f^n]\}.$$

Finally, we update the continuation price and delta of the option using (16) and (17).

To obtain the optimal set of parameter $\{\Omega^*, \Theta^*\}$, we can rewrite the loss function as a function of $\{\Omega, \Theta\}$ as

$$\mathcal{L}(\Omega,\Theta) = \mathbb{E}[|c - y(\mathbf{S};\Omega)|^2] + \frac{\Delta t}{(1+r\Delta t)^2}\mathbb{E} \times \left[\left|\sum_{i=1}^{d}\sum_{j=1}^{d} L_{i,j}\sigma_i\left(\frac{\partial c}{\partial s_i} - \frac{\partial y}{\partial s_i}(\mathbf{S};\Theta)\right)\right|^2\right]. \quad (27)$$

Minimizing the loss functions gives us the optimal set of parameters, we can express this as

$$\{\Omega^*, \Theta^*\} = \arg\min_{\{\Omega,\Theta\}} \{\mathcal{L}(\Omega,\Theta)\}. \quad (28)$$

To solve (28), an optimization algorithm like Adam (Kingma and Ba 2017) is used. The set of optimal parameters naturally gives us $\mathbf{y}^{n*} = y^n(\mathbf{S}^n;\Omega^*)$, and $\nabla\mathbf{y}^{n*} = \nabla y^n(\mathbf{S}^n;\Theta^*)$ which solves the problem (12).

The full algorithm for training is presented in Algorithm 3. We let $\mathcal{O}$ be the optimizer that minimizes (12) and we let $\mathbf{h}_\Delta^N$ be the hidden state of the delta approximating network. The max function is the element wise maximum between its inputs. To simplify some notation, we let $\mathbf{y}_m^n \equiv y^n(\mathbf{S}^n;\Omega)$, and let $\nabla\mathbf{y}_m^n \equiv \nabla y^n(\mathbf{S}^n;\Theta)$.

Algorithm 3 Training the recurrent network

Require: $\mathbf{S}^n, \mathcal{O}$
initialize $\mathbf{c}$,$\nabla\mathbf{c}$,$\mathbf{y}$, and $\nabla\mathbf{y}$ of shape ($N+1$,d,batch size)
for $\mathbf{S}^n$ in Batch **do**
$n = N$
$\mathbf{c}^N$,$\nabla\mathbf{c}^N \leftarrow f^N(\mathbf{S}^N)$,$\nabla f^N(\mathbf{S}^N)$
$\mathbf{y}^N$,$\nabla\mathbf{y}^N \leftarrow f^N(\mathbf{S}^N)$,$\nabla f^N(\mathbf{S}^N)$
$\{\mathbf{h}^N, \mathbf{h}_\Delta^N\} \leftarrow \{f^N(\mathbf{S}^N), \nabla f^N(\mathbf{S}^N)\}$
for $n = N-1,\ldots,0$ **do**
$\mathbf{X}^n \leftarrow [\mathbf{S}^n, g(\mathbf{S}^n)]^\top$
$\tilde{n} \leftarrow \max_{\tilde{n}\in[N-1,\ldots,n+1]} \mathbb{E}[f(\mathbf{S})]$
$\mathbf{c}^n$,$\nabla\mathbf{c}^n \leftarrow e^{-r(\tilde{n}-n)\Delta t} f^{\tilde{n}}(\mathbf{S}^{\tilde{n}})$,$e^{-r(\tilde{n}-n)\Delta t}\nabla f^{\tilde{n}}(\mathbf{S}^{\tilde{n}})\frac{S^{\tilde{n}}}{S^n}$
$\mathbf{y}^{N-1,\ldots,0} \leftarrow$ Algorithm 1
$\nabla\mathbf{y}^{N-1,\ldots,0} \leftarrow$ Algorithm 2
$\{\mathbf{y}^*, \nabla\mathbf{y}^*\} \leftarrow \mathcal{O}(\mathcal{L}(\mathbf{y}, \nabla\mathbf{y}))$

Once training is complete, we evaluate the trained network with a forward pass of the network. Similar to the training phase, the recurrent networks can evaluate the full sequence of events at once. The initial setup at time $n = N$ is the same and will not be repeated.

In the evaluation phase, we directly construct the evaluation input 4.5. With the hidden state at $n = N$ we use the trained network algorithm 1 to approximate $y^n(\mathbf{S}^n;\Omega^*)$ and we use the trained network algorithm 2 to approximate $\nabla y^n(\mathbf{S}^n;\Theta^*)$ for $n = N-1,\ldots,0$. For the evaluation phase, we use the learned price, $y^n(\mathbf{S}^n;\Omega^*)$ to construct the exercise boundary which is given by

$$\mathcal{E}^{n*} = \begin{cases} 1 & \text{if } y^n(\mathbf{S}^n;\Omega^*) \geq f_\kappa^n(\mathbf{S}^n) \\ 0 & \text{otherwise.} \end{cases}$$

Then the value update is given by

$$\mathbf{v}^n \approx \max\{f^n(S^n), y^n(\mathbf{S}^n;\Omega^*)\} \quad (29)$$

and the delta is updated as

$$\nabla\mathbf{v}^n \approx (1-\mathcal{E}^{n*})\nabla f^n(\mathbf{S}^n) + \mathcal{E}^{n*}\nabla y^n(\mathbf{S}^n;\Theta^*). \quad (30)$$

The full algorithm for evaluation is presented in algorithm 4.

Algorithm 4 Evaluation of Recurrent network

Require: $\mathbf{S}^n, \epsilon$
initialize $\mathbf{v}$,$\nabla\mathbf{v}$,$\mathbf{y}$, and $\nabla\mathbf{y}$ of shape ($N+1$,d,batch size)
for $\mathbf{S}^n$ in Batch **do**
$n = N$
$\mathbf{v}^N, \nabla\mathbf{v}^N \leftarrow f^N(\mathbf{S}^N), \nabla f^N(\mathbf{S}^N)$
$\mathbf{y}^N$,$\nabla\mathbf{y}^N \leftarrow f^N(\mathbf{S}^N)$,$\nabla f^N(\mathbf{S}^N)$
$\{\mathbf{h}^N, \mathbf{h}_\Delta^N\} \leftarrow \{f^N(\mathbf{S}^N), \nabla f^N(\mathbf{S}^N)\}$
for $n = N-1,\ldots,0$ **do**
$\mathbf{X}^n \leftarrow [\mathbf{S}^n, g(\mathbf{S}^n)]^\top$
$\mathbf{y}^{*[N-1,\ldots,0]} \leftarrow$ algorithm 1
$\nabla\mathbf{y}^{*[N-1,\ldots,0]} \leftarrow$ algorithm 2
$\mathcal{E}^* \leftarrow (\mathbf{y}^* \geq f(\mathbf{S}))$
$\mathbf{v} \leftarrow \max\{f(\mathbf{S}), \mathbf{y}^*\}$
$\nabla\mathbf{v} \leftarrow (1-\mathcal{E}^*)\nabla f(\mathbf{S}) + \mathcal{E}^*\nabla\mathbf{y}^*$

5. Computational costs

In this section, we analyze the computational time and memory cost of our algorithm and compare the proposed method with the DRL method. The DRL method and other feedforward methods require the neural networks to be initialized and trained at each time step. The method in Chen and Wan (2021) attempts to reduce the RAM load by pricing at each time step and storing the results. However, this means that they must write to a hard drive at each time step, which introduces additional time inefficiencies and large memory overhead.

The time complexity of training and pricing for the DRL method has been shown to be $\mathcal{O}((\frac{c_1 N}{J} + c_2)NMLd^2)$ (Chen and Wan 2021), where N is the number of time steps, M is the number of simulations, L is the number of neural network layers, d is the dimensions, J is the number of timestep skips, and $\{c_1, c_2\}$ are constants related to the batch size and number of epochs. Note that J is a small constant, typically around $2-5$. Thus the complexity is quadratic in N.

The memory complexity is calculated to be $\mathcal{O}(NMd)$ (Chen and Wan 2021), for storing the asset prices, price

approximation y^n and the delta approximation ∇y^{n+1}. However, this does not count the number of network weights and trainable parameters that must be computed and stored at each timestep. Taking into account all network parameters, the total memory complexity is $\mathcal{O}(NMLd^2)$. Our proposed method allows for time and memory savings since we only need to train one set of weights rather than N/J sets of weights. In our analysis, we drop small constant factors on the time and memory complexities.

5.1. Time cost

We will separate the analysis of two cases: one at the final time and one for the other time steps.

Case 1: $\mathbf{n} = \mathbf{N}$. At final time when $n = N$, the proposed method needs to compute $\mathcal{E}^N$, v^N and ∇v^N, each of which is an $M \times d$ matrix. This is performed twice, once for algorithm 3 and once for algorithm 4. For each algorithm 4 and algorithm 3, the time complexity for $\mathcal{E}^N$ is of constant order. The evaluation of $f^N(S^N)$ and $\nabla f^N(S^N)$ over an array of size $M \times 2d$ both require addition which is $\mathcal{O}(Md)$. Thus the time complexity requirement for N is $\mathcal{O}(Md)$.

Case 2: $\mathbf{n} = \mathbf{N} - \mathbf{1}, \ldots, \mathbf{0}$. At timesteps $n = N - 1, \ldots, 0$, we need to construct the input X^n an $M \times 2d$ matrix, initialize and optimize the Recurrent networks y^n and ∇y^n. The construction of the input X^n and the initialization of the Recurrent network is done in constant time. Most of the operations required in algorithm 3 are performed during optimization. For input size $M \times 2d$ our network has weights of size $2d \times 2d$ and an output weight of size $2d \times d$ for each of the L layers. The dominant operation in algorithms 3 and 4 is multiplication of weights and inputs which requires $\mathcal{O}(MLd^2)$ operations. Thus over N timesteps the total time complexity for algorithms 3 and 4 is bounded by $\mathcal{O}(NMLd^2)$.

Using the Python library time, we can track the time requirements of the DRL method and compare it to our method; the total times used by both methods are presented in table 1. The complexity of the methods was measured by taking the logarithm of the timing results from table 1 and presented in figure 4. Figure 4 shows that the DRL method has worse than linear growth with respect to N and a near linear growth in time in our method. As shown in table 1, we can clearly see the absolute advantage we achieve by reducing the complexity by a factor of N, and our method at $N = 100$ is already 10 times faster than the DRL method. Also the time results did not vary significantly between dimension 2, 5 and 10, but showed more variation when the number of timesteps was increased.

5.2. Memory cost

Another contribution of our work is the reduction of memory complexity by a factor of N due to the use of a Recurrent network. This is a large reduction, since N is the dominant number in our complexity, since a large N is required for more accurate solutions.

The initialized asset prices, option payoff $f^n(S^n)$ and delta $\nabla f^n(S^n)$ require $\mathcal{O}(NMd)$ floating points of storage, which is the same as the DRL method (Chen and Wan 2021). Our method differs in terms of the network; thus we only need to compare the storage requirements of the network. Since the proposed method only stores the training parameters in the final timestep, it is not required to store the training weights and gradient information in each timestep. Thus, for the input of size $M \times 2d$, hidden layers of size $2d \times 2d$ and output layer of size $2d \times d$, the Recurrent networks requires $\mathcal{O}(Md^2)$ floating points. This is performed over L layers, thus the final memory complexity is $\mathcal{O}(MLd^2)$ which is constant in N.

As for comparison, using the python library tracemalloc we measured the total peak memory requirements of the DRL method and compared it to our method. This was done by measuring the peak memory of each function and adding the accumulated peak memories. The total peak memory used by both methods is presented in table 1. The memory complexity of the methods were measured by taking the logarithm of the peak memory measured for a given number of timesteps N and presented in figure 5. Figure 5 shows the near linear growth of memory in N for the DRL method and a constant growth in our method. Furthermore, the memory results did not vary significantly between dimensions 2, 5, and 10, but showed more variation when the number of time steps was increased.

6. Numerical results

In this section, we solve the high-dimensional American option problem using our algorithms 3 and 4. We compute the price $v(\vec{s}_0, 0)$ and the delta $\nabla v(\vec{s}_0, 0)$ at $t = 0$ for given $\vec{s}_0 = (s_0^1, \ldots, s_0^d)$ where $s_0^1 = \cdots = s_0^d = 0.9\,K, K$ or $1.1\,K$. We also show that our method can be used for delta hedging across spacetime, by demonstrating our hedging positions in a two-dimensional case. We use two separate types of experiments. In the first type we constrain all computational runtime such that the runtime for our proposed method and the runtime for the DRL method are the same. In the second type of experiment, we run the experiment until the pricing errors produced by our method matches the pricing errors of the DRL method.

In our experiments, we set the strike $K = 100$. The smoothing parameter in (25) is given by $\kappa = 2/\Delta t$. We use the parameters $\alpha = 0.5$ and $\beta = 0.5$. The network used in our experiments is a $L = 5$ deep RNN with GRU units. The batch size for our proposed method was given as 10, 000 . Our computation was performed using a 6GB-NVIDIA GTX 2060 GPU, a 6 core AMD Ryzen 5 3600X processor, and 16 GB of RAM.

In experiment 1, we look at the American style geometric average option. We compare the accuracy of the proposed method to the DRL method for both price and delta at time $t = 0$ for $S_0 = s_0$ for all simulations, while under a similar runtime constraint. In experiment 2, we compare the delta hedging results calculated using the price and delta computed by the proposed method and the DRL methods, we present the portfolio mean and standard deviations for dimensions $[2, 5, 30, 100]$. In experiments 3, we compare the price at $t = 0$, delta at $t = 0$ and delta hedging results for the American style max call option under the similar runtime constraints. Experiments 4 and 5 evaluate how long

Table 1. Time and memory comparison between methods.

(a) $d = 2$: Proposed method				
N	Total time (s)	Training time (s)	Memory (MB)	Price ($)
50	39.48	29.2	5.62	6.6601
100	76.35	56.73	5.63	6.4266
1000	516.07	379.94	5.63	6.5961
(b) $d = 2$: DRL method				
N	Total time (s)	Training time (s)	Memory (MB)	Price ($)
50	113.75	113.62	3701.59	6.6702
100	316.3	316.06	7705.53	6.4817
1000	25590.97	25586.84	151944.60	6.5822
(c) $d = 5$: Proposed method				
N	Total time (s)	Training time (s)	Memory (MB)	Price ($)
50	39.32	28.91	5.62	5.9069
100	78.82	57.77	5.63	5.9524
1000	812.96	577.02	5.63	5.8701
(d) $d = 5$: DRL method				
N	Total time (s)	Training time (s)	Memory (MB)	Price ($)
50	134.8	134.64	4548.33	5.9218
100	410.92	410.62	9408.30	5.9518
1000	36739.60	36734.08	168054.00	5.8697
(e) $d = 10$: Proposed method				
N	Total time (s)	Training time (s)	Memory (MB)	Price ($)
50	38.55	28.15	5.62	5.9883
100	76.44	56.05	5.63	5.9087
1000	785.49	552.97	5.63	5.7964
(f) $d = 10$: DRL method				
N	Total time (s)	Training time (s)	Memory (MB)	Price ($)
50	251.40	251.02	6847.64	5.9791
100	726.46	725.86	14124.60	5.9053
1000	65499.84	65491.55	215701.40	5.7364

Note: The American style geometric average call option with $d = 2$, 5, and 10, priced using $M = 50,000$. We used a Recurrent layer size of $L = 7$ for the proposed method. We sued a timestep skip, $J = 4$ and $L = 7$ for the DRL method.

the proposed method needs to run in order to match the accuracy of the DRL method. For experiments 1, 2, and 3, finite difference solutions with very fine grids are used as exact solutions. We note that this is feasible only if $d \leq 3$. We make comparison results to the unsupervised method of Salvador *et al.* (2019), and also look at the deep out-the-money (OTM) and deep in-the-money (ITM) performance of our method, the DRL method and the Longstaff–Schwartz method in experiment 3. In the case of ITM, at-the-money (ATM) and OTM, we do not make a comparison with the Longstaff–Schwartz method. This is because the Longstaff–Schwartz method has been compared with the DRL method in Chen and Wan (2021). It has been shown in Chen and Wan (2021) that the DRL method is more efficient than the Longstaff–Schwartz method. In addition, we remark that the comparison is not made with the other methods referenced in the introduction, such as Han (2016) Han and Jentzen (2017), Sirignano and Spiliopoulos (2018), Beck *et al.* (2018), Guler and Laignelet (2019) Salvador *et al.* (2019). This is because the work of Sirignano and Spiliopoulos (2018) was compared in Chen and Wan (2021). Although the works of Han (2016), Han and Jentzen (2017) and Beck *et al.* (2018) compute European option problems using neural networks based on BSDEs, these works do not discuss the more challenging American option problems. The work of Guler and Laignelet (2019) looks at solving the Black–Scholes and HJB equations using a BSDE approach; however, they look at solving the general PDE and not the American option problem. Their approach is also simulated with very limited number of timesteps, simulation size and the dimensions are limited to $d = 2$.

We note that when finite difference solutions are available, we can evaluate the absolute and percent errors of computed prices and deltas. More specifically, denote the finite difference solutions by v_{exact}. Then the percent error of the price at

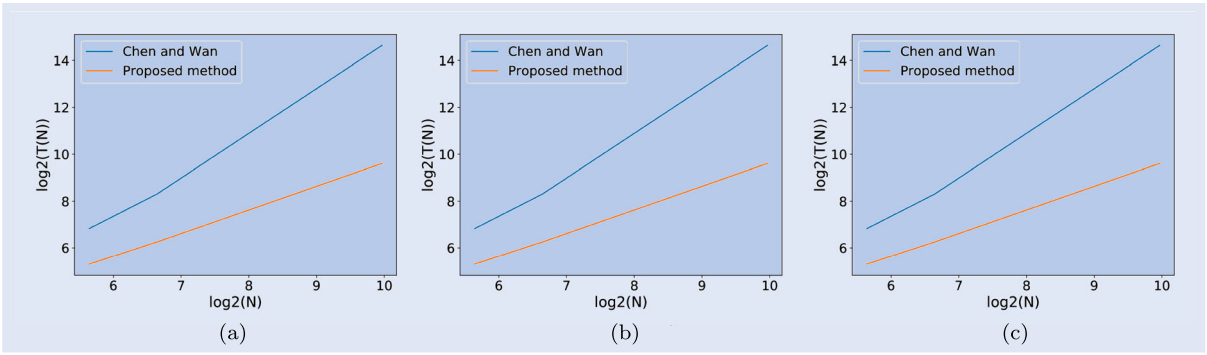

Figure 4. Logarithm of the total time, T(N), against the number of time steps, N, for the DRL method (blue line) and the proposed method (orange line) for dimensions (a) $d = 2$, (b) $d = 5$ and (c) $d = 10$.

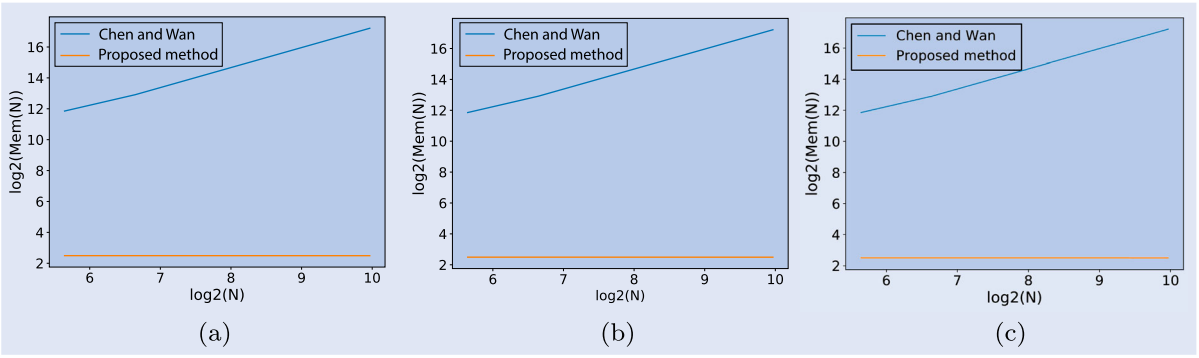

Figure 5. Logarithm of total peak memory versus the number of time steps, N, for the DRL method (blue line) and the proposed method (orange line) for dimensions (a) $d = 2$, (b) $d = 5$, and (c) $d = 10$.

$t = 0$ is

$$\frac{|v(\vec{s}_0, 0) - v_{\text{exact}}|}{|v_{\text{exact}}|} \times 100\%, \tag{31}$$

and the percent errors of the delta at $t = 0$ is

$$\frac{\|\nabla v(\vec{s}_0, 0) - \nabla v_{\text{exact}}\|_{L_2}}{\|\nabla v_{\text{exact}}\|_{L_2}} \times 100\%. \tag{32}$$

In addition, we evaluate the quality of computed exercise boundaries. In the first set of experiments, we look at the exercise boundary, i.e. a point $(\vec{S}^n, t^n)$ is 'exercised' or 'continued', computed by our proposed method and the DRL method, under run time constraint. The true exercise points is determined by finite difference method without any run time constraints. Let the 'exercised' class be the positive class and denote the numbers of true positive, true negative, false positive, and false negative samples as TP, TN, FP, FN, respectively. Then the quality of the exercise boundaries can be evaluated as

$$\text{f1} - \text{score} = \frac{TP}{TP + 0.5(FP + FN)}. \tag{33}$$

The best (or worst) case of the f1-score is 1 (or 0), respectively. Another common metric to evaluate the quality of classification problems is accuracy; however, since all of our experiments are skewed to the positive class, the F1 score would be a better metric than the accuracy (Chen and Wan 2021).

6.1. *American style geometric average call option*

We consider a d-dimensional American style 'geometric average' call option, where $\rho_{ij} = \rho$ for $i \neq j$, $\sigma_i = \sigma$ for all i's, and the payoff function is given by $f(\vec{s}) = \max\{(\prod_{i=1}^{d} s_i)^{1/d} - K, 0\}$. Although, geometric average options are rarely seen in practice, they have semi-analytical solutions for benchmarking the performance of our algorithm in high dimensions, as shown in Glasserman and Yu (2004), Sirignano and Spiliopoulos (2018) and Chen and Wan (2021). More specifically, it is shown in Glasserman and Yu (2004), that the d-dimensional can be reduced to a one-dimensional American call option with the variable $s' = (\prod_{i=1}^{d} s_i)^{1/d}$, where the effective volatility is $\sigma' = \sqrt{(1 + (d-1))/d}\sigma$, and the effective drift, $\mu' = r - \delta + \frac{1}{2}(\sigma'^2 - \sigma^2)$. Thus, by solving the equivalent one-dimensional option using finite difference method, one can compute the d-dimensional option prices and under special circumstances, deltas† accurately.

In the following experiments 1 and 2, we evaluated the American style geometric average call option under run time constraints. For each $s_0 = 90, 100, 110$ and $d = 2, 5, 30, 100$ we run our method and the DRL method for a fixed run time in seconds. Through this experiment we show that under time constraints, our method approximates the solution more accurately than the DRL method. We computed price of the American style geometric average call option as described

† We note that solving the equivalent one-dimensional option is not sufficient for computing the d-dimensional delta except at the symmetric points $s_1 = \cdots = s_d$. Interested readers can verify this by straightforward algebra.

Table 2. Two-dimensional geometric average call option: computed prices and deltas at $t = 0$.

	(a) Finite difference method				
s_0	Price	Percent error (%)	Delta	Percent error (%)	Run time (s)
90	6.7000	–	19.3136	–	–
100	11.2502	–	25.8041	–	–
110	16.7708	–	31.3886	–	–
	(b) Proposed method				
s_0	Price	Percent error (%)	Delta	Percent error (%)	Run time (s)
90	6.6520	0.71348	19.0308	1.46425	36.39
100	11.0801	1.93303	25.3053	1.93303	36.74
110	16.8600	0.5313	31.1031	0.90957	36.21
	(c) DRL method				
s_0	Price	Percent error (%)	Delta	Percent error (%)	Run time (s)
90	5.2920	21.0117	23.0260	19.2220	36.01
100	10.0960	18.2110	30.5032	18.2110	36.25
110	16.2260	3.24612	35.3606	12.6540	36.09

Table 3. Five-dimensional geometric average call option: computed prices and deltas at $t = 0$.

	(a) Finite difference method				
s_0	Price	Percent error (%)	Delta	Percent error (%)	Run time (s)
90	6.0685	–	7.3985	–	–
100	10.3173	–	10.0894	–	–
110	16.1580	–	12.8505	–	–
	(b) Proposed method				
s_0	Price	Percent error (%)	Delta	Percent error (%)	Run time (s)
90	6.0820	0.2241	7.3345	0.86504	37.19
100	10.4070	0.8733	9.9467	1.41436	38.69
110	15.9818	1.08926	12.3168	4.153145792	37.54
	(c) DRL method				
s_0	Price	Percent error (%)	Delta	Percent error (%)	Run time (s)
90	5.6789	6.41345	7.9648	7.6543	37.06
100	9.1800	11.0281	12.6274	25.155	38.23
110	13.1910	18.3589	17.9543	32.713	37.07

in Sirignano and Spiliopoulos (2018), with the parameters $\{T = 2, r = 0, \sigma = 0.25, \rho = 0.75\}$.

Experiment 1: American style geometric average call option

In the first experiment, we compare the computed prices at $t = 0$ between our proposed method and the deep residual network of Recurrent under a run time constraint, see tables 2–5. Each of our results in tables includes: the exact prices computed by the Crank–Nicolson finite difference method with 1000 timesteps and 16, 385 space grid points: the computed prices, the corresponding percent errors, and the run time of the DRL method; the computed prices, the percent errors, and the run time of our proposed method. For the proposed method, the percent errors range from 0.29% to 2.29% for the calculated prices. Our method performs noticeably better than the DRL method at lower dimensions as total sample sized used for $d = 2$, 5, 30, 100 was 200, 000 . For comparison, for the DRL method, the percent errors range from 2.04% to 21.01%. To ensure the computation did not exceed the runtime limit, the timestep was set to $N = 10$ and the skip parameter $J = 8$. This led to a serious degradation of accuracy for the DRL method.

The DRL method gives us deltas in all spacetime. Tables 2–5† also show the deltas at $t = 0$ computed by our proposed method, along with the ones computed using the DRL method. The DRL method produces percent errors of the deltas ranged from 1.01% to as high as 37.71%; this is due to the limited sampling available under the run time constraint. The limited sampling leads to a poorly trained network and a degradation of the exercise boundary as shown in figure 6. We see better performance from our proposed model under our time constraints, as we consistently see errors below 2.519%. Furthermore, we compare the exercise boundaries computed by our approach with the ones computed using the DRL method.

† Only a single value of delta is reported for the geometric average, since our experiment was run for $S_1 = \cdots = S_d$ all deltas are equal as shown in Chen and Wan (2021).

Table 4. 30-dimensional geometric average call option: computed prices and deltas at $t = 0$.

	(a) Finite difference method				
s_0	Price	Percent error (%)	Delta	Percent error (%)	Run time (s)
90	5.7740	–	1.2028	–	–
100	9.9660	–	1.6593	–	–
110	15.3390	–	2.0697	–	–
	(b) Proposed method				
s_0	Price	Percent error (%)	Delta	Percent error (%)	Run time (s)
90	5.7200	0.93701	1.1944	0.69837	52.49
100	9.8960	0.70136	1.6307	1.72362	52.37
110	15.4680	0.8449	2.0405	1.41083	53.26
	(c) DRL method				
s_0	Price	Percent error (%)	Delta	Percent error (%)	Run time (s)
90	5.6557	2.04375	1.1907	1.00599	52.44
100	8.3430	16.2839	1.9863	19.707	52.44
110	14.8900	2.92532	2.3296	12.557	53.17

Table 5. 100-dimensional geometric average call option: computed prices and delta at $t = 0$.

	(a) Finite difference method				
s_0	Price	Percent error (%)	Delta	Percent error (%)	Run time (s)
90	5.6352	–	0.3548	–	–
100	10.0658	–	0.4967	–	–
110	15.2953	–	0.6272	–	–
	(b) Proposed method				
s_0	Price	Percent error (%)	Delta	Percent error (%)	Run time (s)
90	5.6960	0.93701	0.3566	0.5073	212.74
100	9.9118	1.52993	0.4903	1.2885	205.08
110	15.3499	0.3570	0.6114	2.51913	206.21
	(c) DRL method				
s_0	Price	Percent error (%)	Delta	Percent error (%)	Run time (s)
90	5.9140	4.9404	0.4046	14.036	212.46
100	8.9989	10.5993	0.6195	24.723	205.59
110	16.2755	6.4085	0.6789	8.243	206.43

In table 6 we evaluate the F1 score of the exercise boundary classification using (33). For the proposed method, the F1 score ranges from 0.8945 to 0.9545 . For the DRL method, the F1 score ranges from 0.6193 to 0.6651 . This illustrates the more consistent performance of our proposed approach under time constraints.

In figure 6, we visualize the exercise boundaries computed by the proposed method and the DRL method. The results of our computation shows more points because the efficiency of our method allows us to use more timesteps for our simulation. Sample points are generated over the entire spacetime, i.e. $\{(S_n^m, t_n) \mid n = 0, \ldots, N; m = 1, \ldots, M\}$. Each sample point is classified using the proposed method or the DRL method. We use bold dark blue crosses to symbolize the sample points that are misclassified as continued (false positives), and bold dark red crosses to symbolize the points that are misclassified as exercised (false negatives). The plots show that under runtime constraints, the exercise boundary of our proposed method are more precise than the DRL method.

Experiment 2: Delta hedging of American style geometric average call option.

In Experiment 2, we perform delta hedging simulations over the period $[0, T]$ using the proposed method for the geometric call option. We evaluate the quality visually using the density plotted of the relative profit and loss (P&L) as described in Forsyth and Vetzal (2002) and He *et al.* (2007). Relative P&L is given by

$$\text{Relative } P\&L \equiv \frac{e^{-rT}\Pi_T}{V_0},$$

where Π_T is the replicating portfolio at expiry T. The relative P&L for perfectly hedged portfolios should be a Dirac delta function (Forsyth and Vetzal 2002).

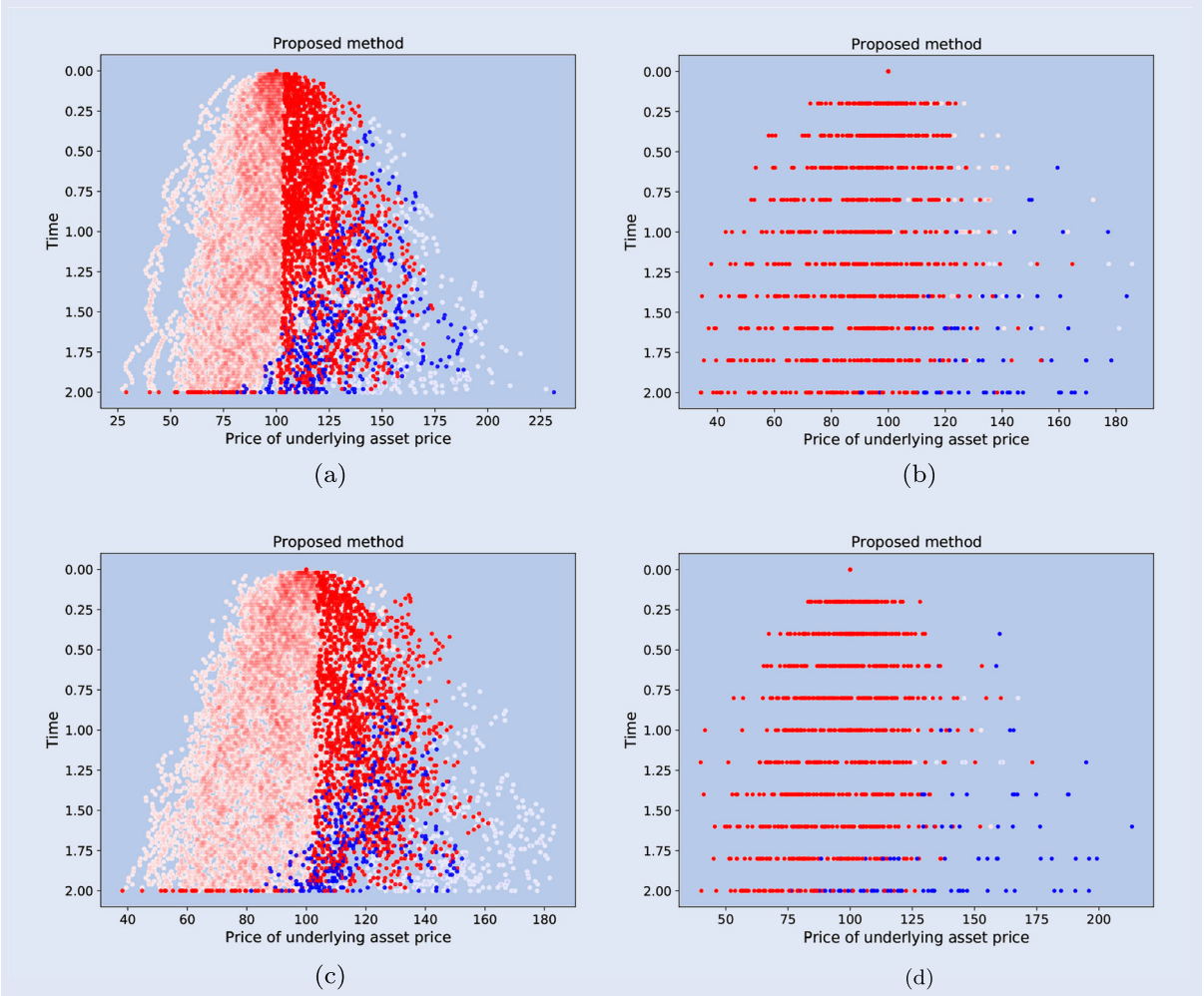

Figure 6. Multi-dimensional geometric call options. Comparison of the exercised sample points between the proposed method (a and c) and the DRL method (b and d). Points exercised (blue) and points continued (red); points misclassified as continued (bold dark blue) and points misclassified as exercised (bold dark red).

Table 6. F1-score calculated using (33) for multi-dimensional American style geometric average call option under time constraint.

	Proposed method				DRL method			
s_0	$d = 2$	$d = 5$	$d = 30$	$d = 100$	$d = 2$	$d = 5$	$d = 30$	$d = 100$
90	0.9465	0.9486	0.9394	0.9545	0.6563	0.6550	0.6504	0.6565
100	0.9241	0.9193	0.9390	0.9213	0.6644	0.6651	0.6453	0.6550
110	0.8945	0.8953	0.9004	0.8956	0.6248	0.6403	0.6324	0.6193

Due to the discretization of time, the relative P&L in numerical computation is a normal distribution with mean zero and small standard deviation proportional to Δt (Chen and Wan 2021). We did not compare the hedging results with the DRL method in this experiment as the hedging intervals were different. The computation of relative P&L requires both prices and deltas for the entire spacetime.

In table 7, we show the calculated mean and standard deviation of the relative P&Ls for all 200, 000 simulation paths, using the proposed method. We computed values that are close to zero and standard deviations are also close to zero.

In figure 7, we can see the densities of the relative P&Ls. The computed densities are normally distributed and approach the Dirac function with zero means. This further illustrates the capabilities of our approach in determining prices and deltas across spacetime. These results confirm that our methods provide a good hedge of assets across the spacetime.

6.2. *American style max call option*

We consider a d-dimensional American style 'max' call option, where $\rho_{ij} = \rho$ for $i \neq j$, $\sigma_i = \sigma$ for all i's, and the payoff function is given by $f(\vec{s}) = \max\{\max_{i=1,\ldots,d}\{s_i\} - K, 0\}$. Multi-dimensional max options are common in practical applications.

In the following experiments 3, 4, and 5, we evaluate the American style max call option. Like experiments 1 and 2, the methods presented in experiment 3 is run such that the runtime between methods match. This is done to show that

Table 7. Multi-dimensional geometric average call options. Computed (a) portfolio mean and (b) portfolio standard deviation of the relative P&Ls, subject to 250 hedging intervals.

	(a) Portfolio mean			
s_0	$d = 2$	$d = 5$	$d = 30$	$d = 100$
90	−8.928e−8	−1.064e−8	−2.544e−9	7.430e−11
100	−9.805e−8	−5.697e−9	−1.347e−8	−1.931e−9
110	−1.774e−7	−4.014e−8	−1.426e−8	−4.312e−10
	(b) Portfolio std dev			
s_0	$d = 2$	$d = 5$	$d = 30$	$d = 100$
90	1.718e−4	5.711e−5	1.107e−5	3.329e−6
100	2.789e−4	9.691e−5	1.809e−5	5.446e−6
110	3.9398e−4	1.458e−4	2.656e−5	8.035e−6

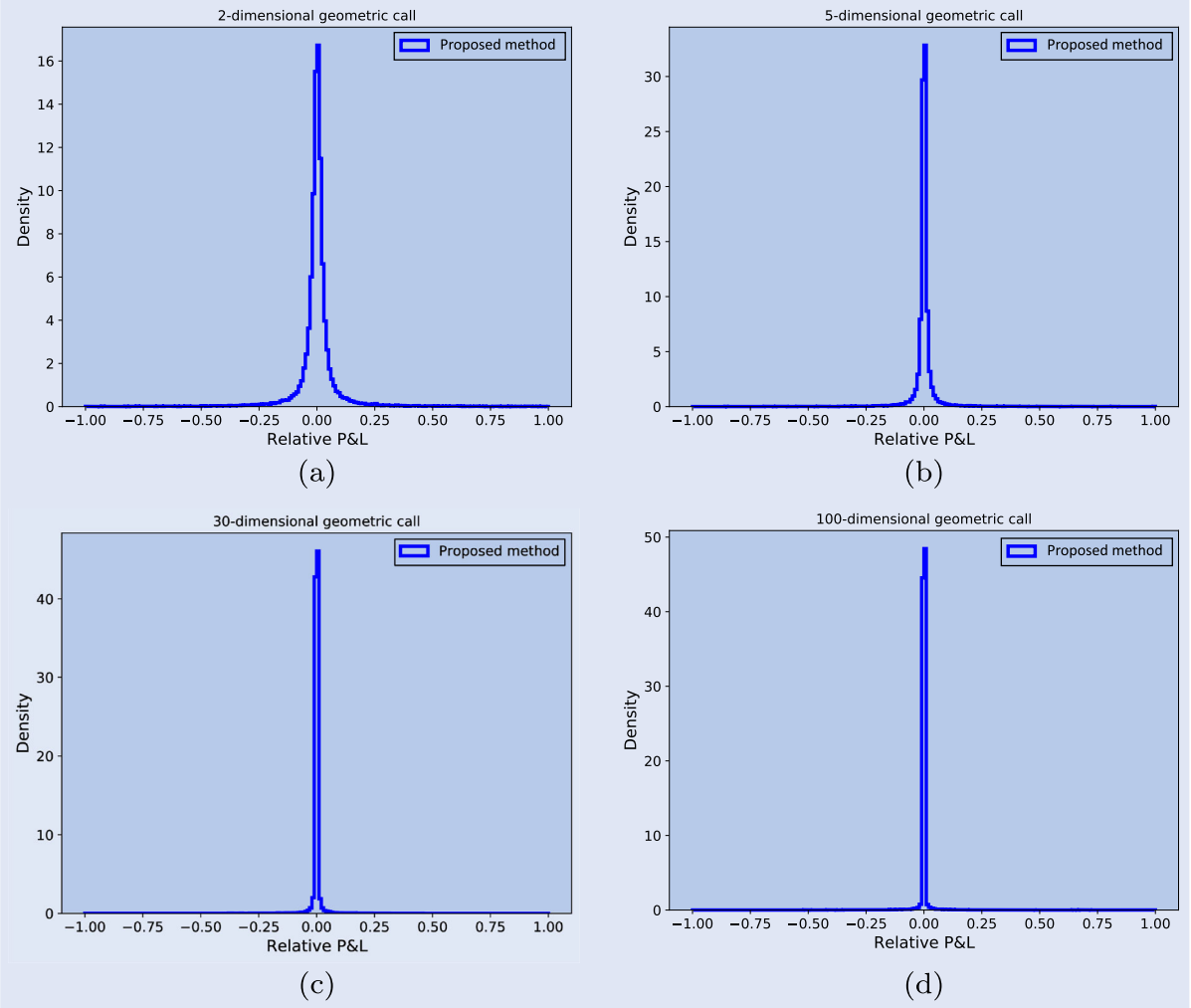

Figure 7. Multidimensional geometric call options. Density of the relative P&Ls computed by the proposed approach, subject to 250 hedging intervals at $s_0 = 100$ for dimensions: (a) $d = 2$, (b) $d = 5$, (c) $d = 30$, and (d) $d = 100$.

our method performs more efficiently with other payoff structures. In this experiment, we also compare our method to the unsupervised method and investigate the performance of our method far ITM and far OTM, i.e. $s_0 = 80, 120$. Experiment 4 and 5 are run until the accuracy of our proposed method matches the accuracy of the DRL method. In experiment 5, we also look at the timing results of the Longstaff–Schwartz method that matches the accuracy of the DRL method. For $d = 2$, we computed price of the American style max call option as described in Broadie and Glasserman (1996), we set $\{T = 1, r = 0.05, \sigma = 0.2, \rho = 0.3\}$. For $d = 5, 30$ we computed the price of the American max call option as described in Firth (2005). We set $\{T = 3, r = 0.05, \sigma = 0.2, \rho = 0\}$.

Experiment 3: Two-dimensional American style max call option.

The purpose of this example is to show that our method can be used to get the price and delta of different payoff

functions. More specifically, we compare our method with the DRL method, using the the finite difference method solution as a ground truth. We approximate the exact prices and deltas by the Crank–Nicolson finite difference method with 1000 timesteps and 2049×2049 space grid points as in Chen and Wan (2021). The DRL method for this experiment was run with $J = 8$, with sample size $200,000$, for $N = 10$ timesteps. Our experiments compare the runtime of our proposed method and the DRL method given the same runtime. For $s_0 = 90, 100, 110$, we do not compare our method to the Longstaff–Schwartz method as this comparison for these s_0, since that has been done in Chen and Wan (2021). However, we present additional results for $s_0 = 80, 120$, where we make the comparison to the Longstaff–Schwartz method with a fourth-degree polynomial.

Comparison with DRL method

Using our proposed method, the percent errors of the computed prices at $t = 0$ are less than 2.062% (table 8), the percent errors of the computed deltas at $t = 0$ are less than 2.3665% (table 8). For the same runtime, our method produces smaller errors than the DRL method, which had errors between $4.2139\% - 27.6173\%$ for price and $4.3196\% - 66.1151\%$ for delta (figure 8).

Delta hedging of American style max call option

In addition, we compute the relative P&Ls of the delta hedging portfolio using the finite difference method and compare them with the values computed by our approach. Table 9 shows the F1 score of the two methods. The F1 score of our method ranges from 0.8011 to 0.9564. The F1 score of the DRL method ranges from 0.6667 to 0.7965.

Table 10 and figure 9 show the means, standard deviations, and distributions of the relative P&Ls computed by the proposed approach versus by finite difference methods. The variation computed by our proposed approach decreases as more timesteps are used in the training of the model, however, we can observe that the proposed method produces a skewed normal P&L distribution. The right skewness of the resulting P&L distribution shows that the replicating portfolio of our methodology is biased toward overestimating the risk of the portfolio.

American style max call option price and delta Deep OTM and ITM

Next we looked at testing the performance of our proposed method in the case where the max call option was far ITM, $s_0 = 80$, and far OTM, $s_0 = 120$. We compared this with the DRL method and the Longstaff–Schwartz method and summarize the results in table 11. Using our proposed method, the percent errors of the computed prices at $t = 0$ are less than 2.7671% (table 8), the percent errors of the computed deltas at $t = 0$ are less than 3.8461% (table 11). For the same runtime, our method produces smaller errors than the DRL method, which had errors 13.5132% and 43.3926% for prices and 44.32096% and 6.3112% for deltas. However, this is not the case for the Longstaff–Schwartz method. We see that for small dimensions the price approximated by the Longstaff–Schwartz method has errors of 1.6020% and 0.6170%, which outperforms both our proposed method and the DRL method under time constraint. However, our proposed method still outperforms the Longstaff–Schwartz method when it comes to computing deltas. The Longstaf-f-Schwartz method had relative errors of 8.6042% and 3.1955%.

For pricing under limited runtime, Longstaff–Schwartz method performs better than both neural network approaches when $d = 2$, however, we see that the delta performance is not as good as our proposed method.

Price comparison with unsupervised method

We also compare the output of the proposed method to the unsupervised method using the L-BFGS optimizer and the loss function outlined in Salvador *et al.* (2019). We implemented the unsupervised method as in Salvador *et al.* (2019), where an $101 \times 101 \times 75$ spacetime grid was used to run the unsupervised method. As in the other comparisons, we will limit the total run time to be the same as those given in table 8. Then for the initial prices of $s_0 = 90, 100, 110$, the resulting values output by the unsupervised method are $V_{unsupervised} = 25.72, 25.00, 24.59$, respectively.

Pricing errors are calculated based on FDM prices in table 8 for each s_0. The relative errors of the unsupervised method for each s_0 are $[513.481\%, 167.160\%, 48.527\%]$, respectively. The inaccurate prices of the unsupervised method can be

Table 8. Two-dimensional Max call: computed option prices and deltas at $t = 0$.

	(a) Finite difference method					
s_0	Price	Std	% Error	Delta	% Error	Run time (s)
90	4.1941	–	–	[20.2762, 20.2353]	–	–
100	9.6309	–	–	[32.8168, 32.8905]	–	–
110	17.3235	–	–	[42.4883, 42.4207]	–	–
	(b) Proposed method					
s_0	Price	Std	% Error	Delta	% Error	Run Time (s)
90	4.2806	0.00162	2.062421	[20.7641, 20.7061]	2.3665	150.3
100	9.7198	0.00177	0.923071	[33.3599, 33.3998]	1.6016	156.39
110	17.4483	0.00168	0.72041	[43.3701, 43.1945]	1.9499	150.61
	(c) DRL method					
s_0	Price	Std	% Error	Delta	% Error	Run Time (s)
90	3.0358	0.3946	27.61737	[27.4312, 27.3825]	35.3041	150.4
100	8.0745	0.3856	16.16048	[39.4164, 39.3535]	66.1151	156.43
110	16.5994	0.3761	4.2139	[40.5587, 40.6825]	4.3197	150.85

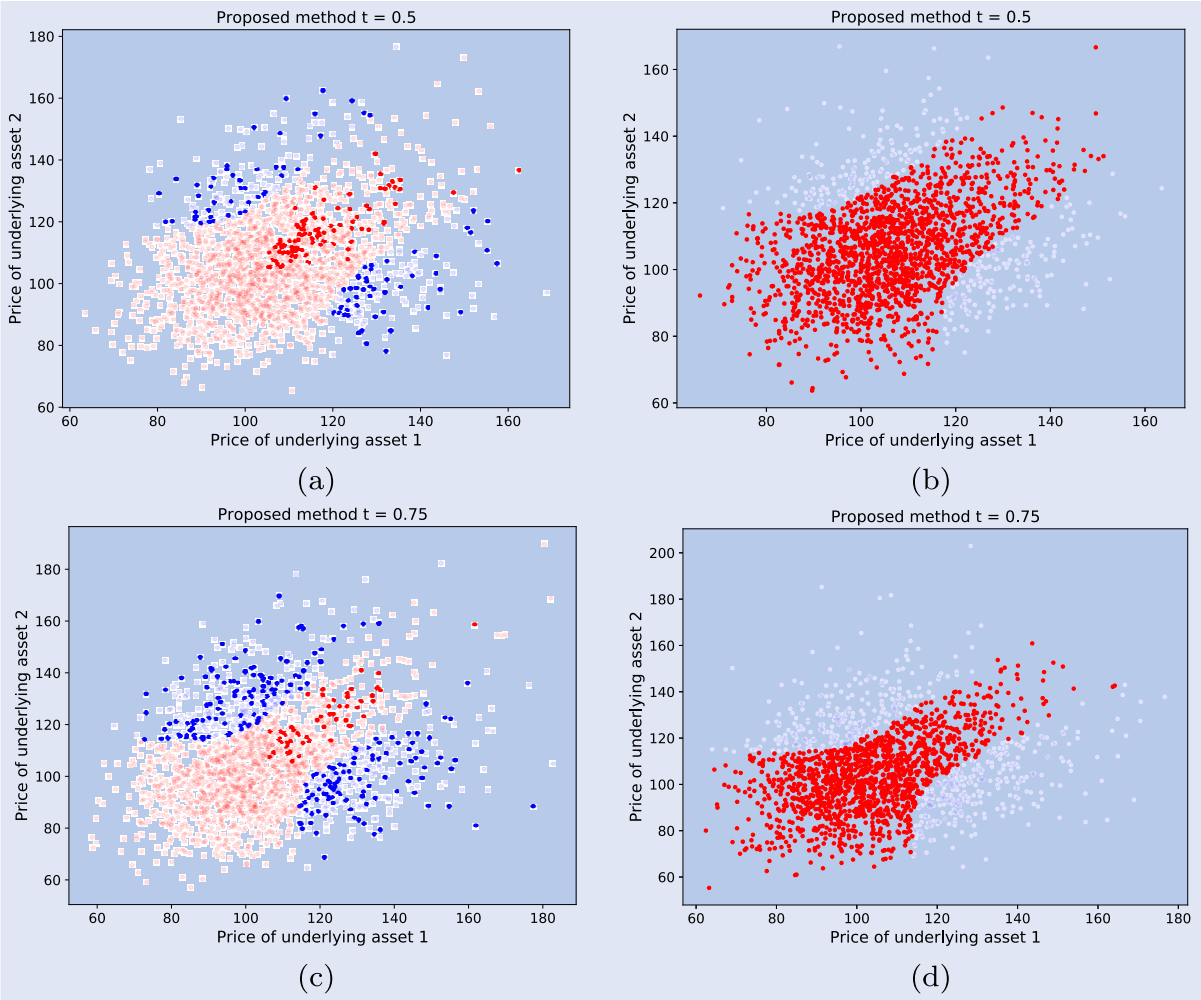

Figure 8. Two-dimensional max call options. Comparison of exercised sample points between the proposed method ((a) and (c)) and the DRL method ((b) and (d)) computed at times $t = 0.5$ ((a) and (b)) and $t = 0.74$ ((c) and (d)) and initial price $s_0 = 110$. Exercised points(blue) and continued points (red); points misclassified as continued(bold dark blue) and points misclassified as exercised (bold dark red).

Table 9. F1-score calculated using (33) for two-dimensional American style max call option under time constraint.

	Proposed method		DRL method	
s_0	$t = 0.5$	$t = 0.75$	$t = 0.5$	$t = 0.75$
90	0.9564	0.9405	0.7187	0.7163
100	0.8983	0.8780	0.7965	0.7471
110	0.8362	0.8011	0.6667	0.6667

Table 10. 2-dimensional Max call option: Means and standard deviations of the relative P&Ls by finite difference versus the proposed method, subject to 100 hedging intervals.

(a) 100 hedging intervals				
	Proposed method		Finite difference method	
s_0	Mean	std	Mean	std
90	−0.0018	0.0899	0.022	0.1932
100	−0.0022	0.0434	0.0016	0.0990
110	−0.0023	0.0072	0.0016	0.0614
(b) 500 hedging intervals				
	Proposed method		Finite difference method	
s_0	Mean	std dev	Mean	std dev
90	−0.0018	0.0011	0.022	0.1932
100	−0.0028	0.0017	0.0016	0.0990
110	−0.0029	0.0021	0.0016	0.0614

attributed to the prescribed limited training time. Note that if the method is allowed to run until convergence, the method is accurate and produces good results. In other words, given the same amount of time, our method is able to produce more accurate results than the unsupervised method. We also note that the unsupervised method is not designed to compute deltas so no delta comparison has been made.

Experiment 4: Five-dimensional American style max call option.

We note that unlike experiment 3, exact solutions are not available. Therefore, in this section, we run our proposed method until it reaches a level of accuracy similar to the results produced by the DRL method. We do not compare with the Longstaff–Schwartz method as this has been done in Chen

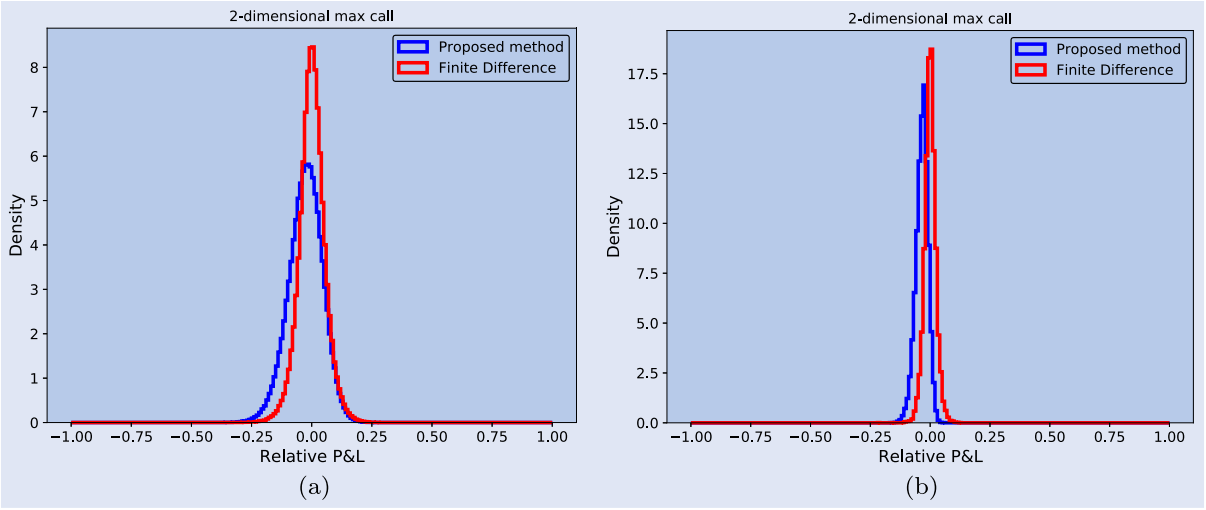

Figure 9. Two-dimensional Max call option. Comparison of the densities of the relative P&Ls computed by the proposed approach (blue) versus the relative P&Ls computed by the finite difference method (red) at $s_0 = 1.1$ K: (a) 100 hedging intervals (b) 500 hedging intervals.

Table 11. Two-dimensional Max call: Deep OTM and ITM computed option prices and deltas at $t = 0$.

	(a) Finite difference method					
s_0	Price	Std	% Error	Delta	% Error	Run time (s)
80	1.3046	–	–	[8.9656, 8.9176]	–	–
120	26.5793	–	–	[47.9348, 47.7876]	–	–
	(b) Proposed method					
s_0	Price	Std	% Error	Delta	% Error	Run time (s)
80	1.3407	0.00163	2.7671	[9.3101, 9,2609]	3.8461	151.82
120	26.826	0.00167	0.9281	[49.3198, 49.1329]	2.8523	153.67
	(c) DRL method					
s_0	Price	Std	% Error	Delta	% Error	Run time (s)
80	0.7385	0.3891	43.3926	[12.9289, 12.8604]	44.2096	151.78
120	22.9876	0.3742	13.5132	[50.8435, 50.9202]	6.3112	153.56
	(d) Lonstaff–Schwartz					
s_0	Price	Std	% Error	Delta	% Error	Run time (s)
80	1.2837	0.01369	1.6020	[9.5721, 9.8498]	8.60416	151.14
120	26.4153	0.01334	0.6170	[49.0682, 49.713]	3.19549	153.13

and Wan (2021) for this experiment. The DRL method for this experiment was run with $J = 2$, with sample size 720000, for $N = 100$ timesteps.

Table 12 reports the runtime, option prices, and deltas at $t = 0$ computed by the proposed method and the DRL method. We observe in this experiment that, for our method, prices that have an absolute difference of order 10^{-2} to the price results from the DRL method following the setup in Chen and Wan (2021). For $s_0 = 90, 100, 110$, the DRL method requires 10026.46 seconds, 10592.07 seconds and 10171.40 respectively. Our method requires 939.86 seconds, 953.41 seconds, and 944.85 seconds respectively. Our method shows about a 10 times faster run time to achieve similar accuracies.

Experiment 5: 30-dimensional American style max call option

In this experiment, we extend Experiment 4 to 30-dimensions, with a simulation size of 270, 000. We report the runtime, the mean option price, the option price standard deviation, the mean delta and the delta standard deviation that the methods took to achieve prices within the order of 10^{-2} absolute difference from the DRL method. In this example we also make comparison with the Longstaff–Schwartz method, in addition we also report standard deviation for price and delta, and report mean price and delta. The experimental results are summarized in table 13.

Table 13 reports the runtime, option prices and deltas at $t = 0$ computed by the proposed method, the DRL method and the Longstaff–Schwartz method. This experiment shows that, for our method, prices have an absolute difference of order 10^{-2} to the price computed from the DRL method. For $s_0 = 90, 100, 110$ the DRL method requires 15763.37 seconds, 15783.14 seconds, and 15743.56 seconds respectively. Our method required 1384.35 seconds, 1388.06 seconds, and 1389.65 seconds, respectively. The Longstaff–Schwartz method required 47034.76 seconds, 47094.21 seconds, and

Table 12. Five-dimensional max call option: computed prices and deltas at $t = 0$.

	(a) Proposed method			
s_0	Price	Std	Delta	Run time (s)
90	16.9152	0.000851	[18.196,18.060,18.142,18.173,18.206]	939.86
100	26.5210	0.000999	[19.100,19.117,19.087,19.511,19.072]	953.41
110	37.1389	0.000813	[20.475,20.646,20.678,20.443,20.601]	944.85
	(b) DRL method			
s_0	Price	Std	Delta	Run time (s)
90	16.8896	0.000709	[17.280, 17.320, 17.540, 17.380, 17.470]	10026.46
100	26.4876	0.000728	[20.170, 20.040, 19.980, 20.710, 20.410]	10592.07
110	37.0996	0.000721	[21.570, 21.980, 21.900, 21.490, 22.020]	10171.40

Table 13. 30-dimensional max call option: computed prices and deltas at $t = 0$.

	(a) Proposed method				
s_0	Price	Std	Delta	Std	Run time (s)
90	41.6933	0.00175	4.9568	0.0230	1384.35
100	57.382	0.00195	4.9354	0.021	1388.06
110	72.3044	0.00146	4.9315	0.0232	1389.65
	(b) DRL method				
s_0	Price	Std	Delta	Std	Run time (s)
90	41.7125	0.00152	5.0139	0.0117	15763.37
100	57.4205	0.00148	5.0121	0.0108	15775.14
110	72.3504	0.00186	5.0133	0.0137	15743.56
	(c) Longstaff-Schwartz				
s_0	Price	Std	Delta	Std	Run time (s)
90	41.6844	0.00259	4.8842	0.03386	47034.76
100	57.3705	0.00283	4.8221	0.03459	47094.21
110	72.5003	0.00309	4.9585	0.03614	47099.81

47099.81 seconds, respectively. Our method shows about a 10 times faster run time to achieve similar accuracy over the DRL method and over 35 times faster than the Longstaff–Schwartz method. We can also see that the delta of the option computed for the Longstaff–Schwartz method has a larger variation over the deltas computed using the DRL method and the Proposed method.

7. Conclusion

We propose a deep Recurrent network framework for pricing and hedging high-dimensional American option. Our proposed method achieves a better runtime efficiency than the DRL method. We reformulate the BSDE as a least-squares problem that is solved using a neural network. We present a loss function (12), which allows us to train our deep RNN framework in all spacetime for the continuation price and delta with theoretical convergence results as shown in (Hure *et al.* 2020). The theoretical time complexity of our proposed method is linear in the number of timesteps N which is better than the quadratic results of Chen and Wan (2021). Interestingly, we found that as N decreases there is an constant increase of the quadratic complexity term. Our algorithms 3 and 4 yield prices and deltas at all points in space and time and it is more accurate than the DRL method under time constraints. The main drawback of our method is that, when there are no time constraints, our method is not as accurate. This can be explained by the difference in accuracy between different loss functions used to solve (7) as shown in Hure *et al.* (2020). A potential future work is to improve the accuracy of the method by improving the Recurrent network architecture, or an extension to transformer networks. Another avenue of work can also explore the use of neural network sample generators to generate better sample data for training.

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the Natural Sciences and Engineering Research Council of Canada.

ORCID

Andrew S. Na http://orcid.org/0000-0002-6162-8171
Justin W. L. Wan http://orcid.org/0000-0001-8367-6337

References

Achdou, Y. and Pironneau, O., Computational methods for option pricing. *Frontiers Appl. Math.*, 2005, **30**, 57–99.

Beck, C., Becker, S., Grohs, P., Jaafari, N. and Jentzen, A., Solving stochastic differential equations and Kolmogorov equations by means of deep learning, 2018.

Bouchard, B. and Warin, X., Monte-Carlo valuation of American options: Facts and new algorithms to improve existing methods. *Numer. Methods Finance*, 2012, **12**, 212–255.

Broadie, M. and Glasserman, P., Estimating security price derivatives using simulation. *Manage. Sci.*, 1996, **42**, 269–285.

Broadie, M. and Glasserman, P., A stochastic mesh method for pricing high-dimensional American options. *J. Comput. Finance*, 2004, **7**, 35–72.

Chen, Y. and Wan, J.W.L., Deep neural network framework based on backward stochastic differential equations for pricing and hedging American options in high dimensions. *Quant. Finance*, 2021, **21**, 45–67.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y., Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, 2014 (Association for Computational Linguistics: Doha).

Duffy, D.J., *Finite Difference Methods in Financial Engineering*, 2006 (John Wily & Sons, Ltd.: Chichester).

El Karoui, N., Peng, S. and Quenez, M.C., Backward stochastic differential equations in finance. *Math. Finance*, 1997, **7**, 1–71.

Elman, J.L., Finding structure in time. *Cogn. Sci.*, 1990, **14**(2), 179–211.

Firth, N.P., High dimensional American options. PhD Thesis, University of Oxford, 2005.

Forsyth, P.A. and Vetzal, K.R., Quadratic convergence for valuing American options using a penalty method. *SIAM J. Sci. Comput.*, 2002, **23**, 2095–2122.

Fujii, M., Takahashi, A. and Takahashi, M., Asymptotic expansion as prior knowledge in deep learning method for high dimensional BSDEs. *Asia-Pacific Financial Markets*, 2017, **26**(3), 391–408.

Glasserman, P. and Yu, B., Simulation for American options: Regression now or regression later? In *Proceedings of the Monte Carlo and Quasi-Monte Carlo Methods 2002*, pp. 213–226, 2004 (Springer: Berlin, Heidelberg).

Goodfellow, I., Bengio, Y. and Courville, A., Deep learning. In *Adaptive Computation and Machine Learning*, 2016 (MIT Press: Cambridge, MA).

Guler, B. and Laignelet, P.P., Towards robust and stable deep learning algorithms for forward backward stochastic differential equations, 2019.

Han, J. and Jentzen, A., Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Commun. Math. Statist.*, 2017, **5**, 349–380.

Han, J., Deep learning approximation for stochastic control problems, 2016.

Haugh, M. and Kogan, L., Pricing American options: A duality approach. *Oper. Res.*, 2004, **52**, 258–270.

He, C., Kennedy, J.S., Coleman, T.F., Forsyth, P.A., Li, Y. and Vetzal, K.R., Calibration and hedging under jump diffusion. *Rev. Derivatives Res.*, 2007, **9**, 1–35.

He, K., Zhang, X., Ren, S. and Sun, J., Identity mappings in deep residual networks. In *Proceedings of the Computer Vision – ECCV 2016*, edited by B. Leibe, J. Matas, N. Sebe and M. Welling, pp. 630–645, 2016 (Springer International Publishing: Cham).

Herrera, C., Krach, F., Ruyssen, P. and Teichmann, J., Optimal stopping via randomized neural networks, 2021.

Hochreiter, S. and Schmidhuber, J., Long short-term memory. *Neural Comput.*, 1997, **9**, 1735–1780.

Hull, J., *Options, Futures and Other Derivatives*, 2003 (Prentice Hall: Upper Saddle River, NJ).

Hure, C., Pham, H. and Warin, X., Deep backward schemes for high-dimensional nonlinear PDEs. *Math. Comput.*, 2020, **89**, 1547–1579.

Jordan, M.I., Attractor dynamics and parallelism in a connectionist sequential machine. In *Artificial Neural Netowrks: Concept Learning*, pp. 112–127, 1990 (IEEE Press: New York City).

Kholer, M., Kryzyzak, A. and Todorovic, N., Pricing of high-dimensional American options by neural networks. *Math. Finance*, 2001, **20**, 383–410.

Kingma, D. and Ba, J., Adam: A method for stochastic optimization, 2017.

Longstaff, F. and Schwartz, E., Valuing American options by simulation: A simple least-squares approach. *Rev. Financ. Stud.*, 2001, **14**, 113–147.

Ramachandran, P., Zoph, B. and Le, Q., Swish: A self-gate activation function, 2017.

Salvador, B., Oosterlee, C. and van der Meer, R., Financial option valuation by unsupervised learning with artificial neural networks, 2019.

Schäfer, A.M. and Zimmermann, H.G., Recurrent neural networks are universal approximators. In *Proceedings of the Artificial Neural Networks – ICANN 2006*, edited by S.D. Kollias, A. Stafylopatis, W. Duch and E. Oja, pp. 632–640, 2006 (Springer Berlin Heidelberg: Berlin, Heidelberg).

Sirignano, J. and Spiliopoulos, K., DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 2018, **375**, 1339–1364.

Thom, H., Longstaff Schwartz pricing of Bermudan options and their greeks. PhD Thesis, University of Oxford, 2009.

Tsitsiklis, J.N. and Van Roy, B., Optimal stopping of Markov processes: Hilbert space theory, approximation algorithms and an application to pricing high-dimensional financial derivatives. *IEEE Trans. Automat. Control.*, 1999, **44**, 1840–1851.