Applied Mathematics and Computation 417 (2022) 126775

Contents lists available at ScienceDirect

Applied Mathematics and Computation

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

Explainable neural network for pricing and universal static hedging of contingent claims

Vikranth Lokeshwar, Vikram Bharadwaj, Shashi Jain*

Department of Management Studies,Indian Institute of Science, Bangalore, India

ARTICLE INFO

Article history:
Received 17 June 2021
Revised 26 October 2021
Accepted 3 November 2021
Available online 21 November 2021

Keywords:
Universal static hedging
Neural network
American Monte Carlo
Regress later

ABSTRACT

We present here a *regress-later* based Monte Carlo approach that uses neural networks for pricing multi-asset discretely-monitored contingent claims. The choice of specific architecture of the neural networks used in the proposed algorithm provides for interpretability of the model, a feature that is often desirable in the financial context. Specifically, the interpretation leads us to demonstrate that any discretely monitored contingent claim —possibly high-dimensional and path-dependent— under Markovian and no-arbitrage assumptions, can be semi-statically hedged using a portfolio of short maturity options. We also show, for Bermudan style derivatives, how the method can be used to obtain an upper and lower bound to the true price, where the lower bound is obtained by following a sub-optimal policy, while the upper bound is found by exploiting the dual formulation. Unlike other duality based upper bounds where one typically has to resort to nested simulation for constructing super-martingales, the martingales in the current approach come at no extra cost, without the need for any sub-simulations. We demonstrate through numerical examples the simplicity and efficiency of the method for both pricing and semi-static hedging of path-dependent options.

1. Introduction

In this paper, we consider the pricing and semi-static hedging[1] of derivative securities that include path-dependent options that are potentially based on several underlying assets. The efficient numerical evaluation of options written on several underlying assets can be challenging due to the so-called *curse of dimensionality*, which affects finite-difference or tree-based methods. Given these restrictions, Monte Carlo methods appear as an attractive approach for the valuation of such derivatives.

When it comes to pricing of path-dependent options, using the Monte Carlo approach, the main difficulty lies in the computation of the conditional expectation across different time intervals. The Monte Carlo approach obtains approximate solutions by combining simulation, regression, and a dynamic programming formulation of the problem. The Monte Carlo methods for pricing Bermudan style options can be broadly classified as either *regress-now* or *regress-later* based schemes.

* Corresponding author.
E-mail address: shashijain@iisc.ac.in (S. Jain).

[1] Static hedging involves taking positions at inception in a portfolio of financial instruments so that the future cash-flows of the target trade match the future cash-flows of the hedge's as well as possible. Semi-static hedging, on the other hand, implies that a finite number of trades are performed after inception. Semi-static replication is a hedging strategy in between static and dynamic hedging, where the latter requires trading along the time continuum from trade initiation.

https://doi.org/10.1016/j.amc.2021.126775
0096-3003/

In the *regress-later* approach, to compute the conditional expectation at t_m, one first approximates the value function at time t_{m+1} by regressing the option values at t_{m+1} against the basis functions constructed using the simulated data at t_{m+1}. The basis functions in the *regress-later* approach are chosen such that their conditional expectations can be computed exactly, which in turn results in the necessary conditional expectation at t_m. Therefore, in the regress later scheme, regression performed at t_{m+1} is used to make the early exercise decision at t_m. Examples of the *regress-later* approach include [3,14,19]. In [18], the *regress-later* approach is extended to compute path-wise forward Greeks. Pelsser and Schweizer [29] shows that as the approximation error from the regression in the *regress-later* approach vanishes, the coefficients obtained are the same regardless of the measure used for calibration. This property can be leveraged for problems where one has to work with mixed probability measures, examples of which include computing potential future exposures of Bermudan swaptions in [13], and computing the capital valuation adjustment, KVA, in [17], where one has to work simultaneously with both the risk-neutral and the real-world measures. A common problem faced by both, the *regress-now* and the *regress-later* approaches —when a linear model is used for the regression— is that the selection of the basis functions for the regression is arbitrary and varies between different payoffs.

Static replication is an attractive alternative to the dynamic hedging of portfolios, as dynamic hedging often breaks down when there are sharp movements in markets or when the market faces liquidity issues. Unfortunately, these are the precise moments where an effective hedge is highly desired. Breeden and Litzenberger [6] shows that path-independent securities can be hedged using a portfolio of standard options that mature along with the claim. For more general path-dependent securities, in a single factor setting, good examples of the static replication approach can be found in [10,12,28]. For high-dimensional path-independent derivatives, [27] found that a hedging portfolio composed of statically held simple univariate options resulted in a hedging performance comparable to dynamic hedging, although only when there were significant transaction costs.

There has been a recent boom in the financial applications of neural networks, mostly driven by the progress made in deep learning and the availability of specialized software and hardware. They have been used in [1,4,25,26] for calibration of stochastic volatility and rough stochastic volatility models. Application of neural networks for model-based pricing of early-exercise options includes the approach of policy iteration in [5], value function iteration in [15,21,22]. The work here falls under the category of value function iteration, although while others follow the *regress-now* framework for value function iteration, we here exploit the benefits of the *regress-later* approach.

In this paper we develop a regress-later based method for pricing multi-asset path-dependent discrete-time contingent claims. As the method uses a feed-forward neural network for the regression at each time step, we call it *Regress-Later with Neural Network* (RLNN). Specifically, RLNN uses a shallow network with a single hidden layers —with rectified linear units (ReLU) activation functions used for the hidden layer and linear activation functions used for the output layer— for the regression at each monitoring date. The choice of this basic architecture is deliberate, as it lends interpretability to the model, something valuable from a practitioner's point of view. Concisely, the first layer determines the payoff structure of a set of options, while the second layer determines the corresponding quantities of each of these options that need to be held in a portfolio, so that the aggregated payoff of the resultant portfolio at any time $t \leq T$ replicates the value function of the target claim at time t, where T is the next monitoring date of the target claim.

The semi-static replication using RLNN is not limited to single factor models but naturally applies to multi-asset contingent claims as well. RLNN therefore not only allows us to efficiently price a contingent claim, but also can be useful in determining an optimal static hedge, to avoid the practical difficulties and costs involved in dynamic hedging of the target claim. An advantage of RLNN over the traditional *regress-later* or *regress-now* approaches —that rely on linear regression— is that the method doesn't require a prior selection of basis functions depending upon the problem at hand. A challenge that RLNN inherits from the *regress-later* schemes is the requirement for closed form or fast numerical approximations for the valuation of the short maturity options that are part of the static hedge portfolio. We demonstrate through examples that this issue can often be addressed in a relatively straightforward manner.

There are three major contributions of this paper. First, we develop a method called regress-later with Neural Networks (RLNN) for pricing high-dimensional discretely monitored contingent claims. The second contribution is that we show, any discretely monitored contingent claim —written possibly on several underliers— under the Markovian and no-arbitrage assumptions, can be semi-statically hedged using a basket of short maturity options. The structure of the short maturity options, and their corresponding weights to be held in the hedge portfolio are the outcomes of the RLNN algorithm. Finally, we show that using RLNN, for the Bermudan style claims, one can obtain a tight lower and upper bound to the true price of the contingent claim at almost no extra costs. The lower bound is obtained by following the sub-optimal policy obtained by RLNN on a new set of paths. The upper bound is obtained using the dual formulation as prescribed by Haugh and Kogan [15] and Rogers [31]. The major computational advantage with obtaining the duality-based upper bound using RLNN comes from the fact that the required martingale can be obtained at no extra cost, which otherwise typically involves computationally heavy sub-simulations. While [14] points out the computational advantage of obtaining the duality-based upper bounds for general *regress-later* algorithms, the quality of the upper bounds is left for future investigations. We through numerical experiments show that the quality of the upper and lower bounds obtained using RLNN are comparable or better than those reported in the literature.

The paper is organized as follows: Section 2 explains the notations used in the paper and the problem formulation. In Section 3 we describe the RLNN algorithm, the underlying neural network architecture, and semi-static hedging of contingent claims using the RLNN. Section 4 provides the proof of convergence of the RLNN method. Discussion on obtaining an

upper and lower bound to the true price of the contingent claim using the RLNN method is presented in Section 4.1. In Section 5 through numerical examples we illustrate the accuracy of the method from both the pricing and static hedging perspective. Finally, we provide the conclusions in Section 6.

2. Problem formulation

This section defines the Bermudan option pricing problem and sets up the notations used in this paper. We assume a complete probability space $(\Omega, \mathcal{F}, \mathbb{P})$ and finite time horizon $[0, T]$. Ω is the set of all possible realizations of the stochastic economy between 0 and T. The information structure in this economy is represented by an augmented filtration $\mathcal{F}_t : t \in [0, T]$, with $\mathcal{F}_t$ the sigma field of distinguishable events at time t, and $\mathbb{P}$ is the risk-neutral probability measure on elements of $\mathcal{F}$. It is assumed that $\mathcal{F}_t$ is generated by W_t, a d-dimensional standard Brownian motion, and the state of economy is represented by an $\mathcal{F}_t$-adapted Markovian process, $\mathbf{S}_t = (S_t^1, \ldots, S_t^d) \in \mathbb{R}^d$, which has dependence on model parameters $\Theta = \{\theta_1, \ldots, \theta_{N_\theta}\}$. We only consider discretely monitored contingent claims, i.e. the payoff of the claim depends only on the value of the price paths on the monitoring dates, $t \in [t_0 = 0, \ldots, t_m, \ldots, t_M = T]$. Let $h_t := h(\mathbf{S}_t)$ be an adapted process representing the intrinsic value of the option, i.e. the holder of the option receives $\max(h_t, 0)$, if the option is exercised at time t. With the risk-free savings account process, $B_t = \exp(\int_0^t r_s\, ds)$, where r_t denotes the instantaneous risk-free rate of return, we define

$$D_{t_{m-1}} = \frac{B_{t_{m-1}}}{B_{t_m}}.$$

We consider the special case where r_t is constant. The problem is then to compute

$$\frac{V_{t_0}(\mathbf{S}_{t_0})}{B_{t_0}} = \max_{\tau} \mathbb{E}\left[\frac{h(\mathbf{S}_\tau)}{B_\tau}\right], \tag{1}$$

where $V_t(\cdot) : t \in [0, T]$ is the option value function, and τ is a stopping time, taking values in the finite set $\{0, t_1, \ldots, T\}$.

The dynamic programming formulation to solve this optimization problem is then as follows. The value of the option at the terminal time T is equal to the product's pay-off,

$$V_T(\mathbf{S}_T) = \max(h(\mathbf{S}_T), 0). \tag{2}$$

Recursively, moving backwards in time, the following iteration is then solved, given V_{t_m} has already been determined, the continuation or hold value $Q_{t_{m-1}}$ is given by:

$$Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) = B_{t_{m-1}} \mathbb{E}\left[\frac{V_{t_m}(\mathbf{S}_{t_m})}{B_{t_m}} \mid \mathbf{S}_{t_{m-1}}\right]. \tag{3}$$

The Bermudan option value at time t_{m-1} and state $\mathbf{S}_{t_{m-1}}$ is then given by

$$V_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) = \max(h(\mathbf{S}_{t_{m-1}}), Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}})). \tag{4}$$

In a Monte Carlo simulation, the time evolution of the process $\mathbf{S}$ is approximated using some discretization scheme. We define a general Markov discretization scheme as

$$\mathbf{S}_{t_m} = F_{m-1}(\mathbf{S}_{t_{m-1}}, \mathbf{Z}_{t_{m-1}}, \Theta), \tag{5}$$

where $\mathbf{Z}_{t_{m-1}}$ is a d-dimensional standard normal random vector, and F_{m-1} is a transformation from $\mathbb{R}^d$ to $\mathbb{R}^d$.

3. Static hedging with portfolio of short maturity European basket options

The principle of the static replication is to construct a portfolio of instruments that mirrors the value function of a target security in every possible state of the world. Unlike dynamic hedging the portfolio weights of a static replicating portfolio do not change with changes in the market conditions. By the no-arbitrage condition, if the value function of a target security is perfectly replicated at future time T, the replicating portfolio should match the security's value at an earlier time t, $(t < T)$ given no early-exercise decision can be made between t and T. In this paper, we propose that a portfolio of short-maturity European style basket options can be used to semi-statically hedge a discretely monitored contingent claim. The payoff of a portfolio of European basket options can be modelled as the output of a single layered feed-forward neural network, that uses ReLU activation functions in the hidden layer and a linear activation function in the output layer. The number of neurons in the hidden layer of this neural network would be equal to the number of options to be held in the portfolio. *Universal approximation theorem shows that any continuous function defined over a compact set can be approximated arbitrarily well using a sufficiently large number of neurons. For our static hedging problem, the analogy would then be, a continuous pay-off can be replicated arbitrarily well, using a sufficiently large number of options. From a practical point of view, the choice of number of options used would be such that the incremental improvement in the replication from the use of an additional option is not offset by the cost of managing the additional option.* The objective is then to find the composition of each European basket option held in the portfolio, and the number of units of that option to be held in the portfolio, so that the hedge portfolio

replicates the value function of the target security at time T. This can –based on the neural network interpretation– then be translated to the objective of finding the optimal parameters of the neural network. The above interpretation forms the basis of the RLNN algorithm described here.

The algorithm begins by generating N independent copies of the sample paths, $\{\mathbf{S}_{t_0}, \ldots, \mathbf{S}_{t_M}\}$, of the underlying process that are obtained using the recursion, $\mathbf{S}_{t_m}(n) = F_{m-1}(\mathbf{S}_{t_{m-1}}(n), \mathbf{Z}_{t_{m-1}}(n), \Theta)$, where $n = 1, \ldots, N$ is the index of the path.

The method then computes the value of the contingent claim at terminal time as $V_{t_M}(\mathbf{S}_{t_M}) = \max(h(\mathbf{S}_{t_M}), 0)$.

The following steps are then employed for each monitoring date, t_m, $m \leq M$, recursively moving backwards in time, starting from t_M:

Assume that $\tilde{V}_{t_m}(\mathbf{S}_{t_m}(n))$, $n = 1, \ldots, N$, our estimates for $V_{t_m}(\mathbf{S}_{t_m}(n))$, are known.

- ***Regress-later***

  In this step a parametrized value function $\tilde{G} : \mathbb{R}^d \times \mathbb{R}^{N_p} \mapsto \mathbb{R}$, which assigns values $\tilde{G}(\mathbf{S}_{t_m}, \beta_{t_m})$ to states $\mathbf{S}_{t_m}$, is computed. Here $\beta_{t_m} \in \mathbb{R}^{N_p}$ is a vector of free parameters. The objective is to choose for each t_m, a parameter vector β_{t_m}, so that

  $$\tilde{G}(\mathbf{S}_{t_m}, \beta_{t_m}) = \tilde{V}_{t_m}(\mathbf{S}_{t_m}).$$

  By the universal approximation theorem, for appropriate choice of the activation functions, a neural network with a single hidden layer and a sufficiently large number of neurons can approximate any continuous function over a compact set arbitrarily well (see [16]). We choose as our approximation architecture at t_m a feed-forward network $\tilde{G}^{\beta} : \mathbb{R}^d \to \mathbb{R}$ of the form

  $$\tilde{G}^{\beta_{t_m}} := \psi \circ A_2 \circ \varphi \circ A_1$$

  where $A_1 : \mathbb{R}^d \to \mathbb{R}^p$ and $A_2 : \mathbb{R}^p \to \mathbb{R}$ are affine functions of the form,

  $$A_1(\mathbf{x}) = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \quad \text{for } \mathbf{x} \in \mathbb{R}^d, \ \mathbf{W}_1 \in \mathbb{R}^{p\times d}, \mathbf{b}_1 \in \mathbb{R}^p,$$

  and

  $$A_2(\mathbf{x}) = \mathbf{W}_2\mathbf{x} + b_2 \quad \text{for } \mathbf{x} \in \mathbb{R}^p, \ \mathbf{W}_2 \in \mathbb{R}^{1\times p}, b_2 \in \mathbb{R}.$$

  $\varphi : \mathbb{R}^j \to \mathbb{R}^j$, $j \in \mathbb{N}$ is the component-wise ReLU activation function given by:

  $$\varphi(x_1, \ldots, x_j) := \big(\max(x_1, 0), \ldots, \max(x_j, 0)\big),$$

  while $\psi : \mathbb{R}^j \to \mathbb{R}^j$, $j \in \mathbb{N}$ is the component-wise linear activation function given by:

  $$\psi(x_1, \ldots, x_j) := \big(x_1, \ldots, x_j\big).$$

  The dimension of the parameter space is therefore,

  $$N_p = 1 + p + p + p \times d.$$

  Specifically, $\beta_{t_m} \in \mathbb{R}^{N_p}$ is chosen to minimize the following mean squared error.

  $$\beta_{t_m} = \arg\min_{\beta_{t_m}} \left(\frac{1}{N} \sum_{n=1}^{N} \left(\tilde{V}_{t_m}(\mathbf{S}_{t_m}(n)) - \tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m}(n)) \right)^2 \right), \tag{6}$$

  where $\mathbf{S}_{t_m}(n)$, $n = 1, \ldots, N$ now constitute the set of training points. Note that β_{t_m} is the vector of free parameters for the neural network model, i.e. the weights and biases from the two layers.

  Figure 1 presents a schematic diagram of the neural network at t_m.

- **Computing the continuation and option values** at t_{m-1}

  The continuation values for the sample points $\mathbf{S}_{t_{m-1}}(n)$, $n = 1, \ldots, N$ are then approximated by,

  $$\begin{aligned} \widehat{Q}_{t_{m-1}}\big(\mathbf{S}_{t_{m-1}}(n)\big) &= B_{t_{m-1}} \mathbb{E}\left[\frac{\tilde{V}_{t_m}(\mathbf{S}_{t_m})}{B_{t_m}} \mid \mathbf{S}_{t_{m-1}}(n) \right] \\ &\approx B_{t_{m-1}} \mathbb{E}\left[\frac{\tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m})}{B_{t_m}} \mid \mathbf{S}_{t_{m-1}}(n) \right] \end{aligned} \tag{7}$$

  Like the other *regress-later* based schemes, the method requires, either a closed-form solution, or a fast-numerical approximation for the expression $\mathbb{E}\left[\frac{\tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m})}{B_{t_m}} \mid \mathbf{S}_{t_{m-1}}(n) \right]$. For the specific choice of network architecture Eq. (7) boils down to evaluating the price of short maturity options that are written at t_{m-1} and expire at t_m. In Section 5, we show with the help of practical examples how the above expectation can usually be readily evaluated.

  Once the approximate continuation values $\widehat{Q}_{t_{m-1}}\big(\mathbf{S}_{t_{m-1}}(n)\big)$ for samples $\mathbf{S}_{t_{m-1}}(n)$, $n = 1, \ldots, N$ have been computed, the corresponding approximations for the option values are given by

  $$\tilde{V}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}(n)) = \max\left(h\big(\mathbf{S}_{t_{m-1}}(n)\big), \widehat{Q}_{t_{m-1}}\big(\mathbf{S}_{t_{m-1}}(n)\big) \right). \tag{8}$$

  The method discussed above is summarized as Algorithm 1 below.

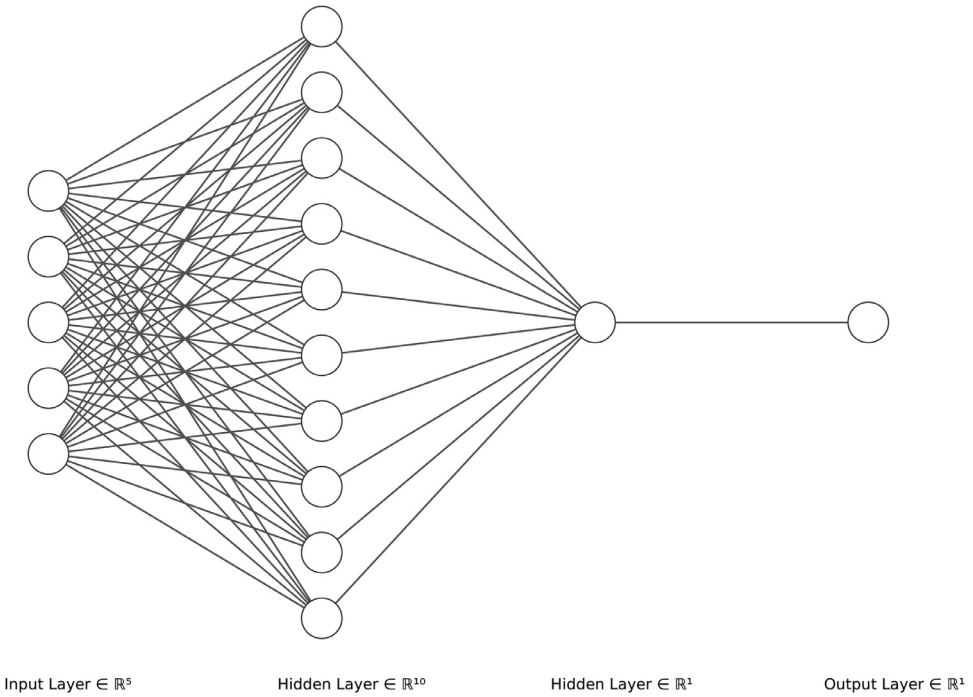

Fig. 1. The neural network architecture chosen for each monitoring date, where the input dimension $d = 5$, the number of neurons p in the hidden layer is 16 and the output layer is equal to 1.

3.1. Interpretation of the first hidden layer

The outcome of the first hidden layer with a choice of p neurons can be represented as

$$\mathbf{o} = \varphi(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \ \mathbf{x} \in \mathbb{R}^d, \ \mathbf{W}_1 \in \mathbb{R}^{p\times d}, \mathbf{b}_1 \in \mathbb{R}^p,$$

where $\varphi : \mathbb{R}^p \to \mathbb{R}^p$, $p \in \mathbb{N}$ is the component-wise ReLU activation function.

Each of the p elements of $\mathbf{o} := \{o_1, \ldots, o_p\}$ therefore has the form,

$$o_i = \max\left(\sum_{j=1}^{d} w_{ij}x_j + b_{i1}, 0\right),$$

where $w_{ij}, \ j = 1, \ldots, d$ is the ith row of $\mathbf{W}_1$,

$$\mathbf{W}_1 := \begin{bmatrix} w_{11} & w_{12} & \ldots & w_{1d} \\ w_{21} & w_{22} & \ldots & w_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{p1} & w_{p2} & \ldots & w_{pd} \end{bmatrix} := \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \vdots \\ \mathbf{w}_p^\top \end{bmatrix}$$

It is easy to see that the ith neuron, where $i = 1, \ldots, p$, has the form of the payoff of an arithmetic basket option with weights $w_{ij}, \ j = 1, \ldots, d$ and strike b_{i1}, written on the underlying $\mathbf{x} := \{x_1, \ldots, x_d\}$. Therefore, the outcome of the first hidden layer provides the structure, i.e. the composition of the weights and the strike, of a set of p arithmetic basket options.

If pricing is the only objective, one could ideally use a deeper network –a network with multiple hidden layers– as opposed to the single hidden layered network considered in RLNN, as deep networks can generally fit non-linear functions using fewer parameters. However, the algorithm also requires a closed form expression for the conditional expectation of the fitted network, as given by Eq. (7), which makes the use of deeper networks a non-trivial exercise.

3.2. Interpretation of the second hidden layer

The second hidden layer performs the following operation:

$$y = \left(\sum_{i=1}^{p} \omega_i o_i\right) + b_2, \ \text{where, } \mathbf{W}_2 := \{\omega_1, \ldots, \omega_p\}, \tag{9}$$

which can be seen as determining the weights of the p basket options that need to held in the portfolio. The amount to be invested in the risk-free asset is given by the bias of the second hidden layer, i.e. b_2.

4. Convergence of RLNN

Under the assumption that the asset price follows a Markovian process, the market is frictionless and there is no arbitrage, we next show that the portfolio of options constructed as above replicates the contingent claim, between monitoring dates. We additionally assume that $\mathbf{S}_{t_m} \in \mathcal{I}_d$, for $m = 0, \ldots, M$, almost surely, where $\mathcal{I}_d$ is some d-dimensional hypercube. As a consequence, we assume, $V_{t_m} \in C(\mathcal{I}_d)$, where $C(\mathcal{I}_d)$ denotes the space of real-valued continuous functions on $\mathcal{I}_d$. *Note that generally, $\mathbf{S}_{t_m} \in \mathbb{R}^d$, and therefore V_{t_m} might not have a compact support (especially for call options). To circumvent the issue we use a truncated domain defined by the sampled Monte Carlo paths for $\mathbf{S}_{t_m}$, and assume the value of V_{t_m} beyond the sampled domain to be equal to zero. As the number of sampled paths increases, the error from the truncation of the domain will become significantly smaller, since the part of the domain that is left out is associated with low probability events.* For ease of notation, we hereafter drop out the discounting terms.

Lemma 1. *Given $\epsilon > 0$ there exists $p > 0$ such that*

$$\tilde{G}^{\beta_{t_m}}(\mathbf{x}) = \sum_{i=1}^{p} \omega_i \varphi\left(\mathbf{w}_i^\top \mathbf{x} + b_i\right) + b_2, \tag{10}$$

satisfies

$$\sup_{\mathbf{x} \in \mathcal{I}_d} \left| V_{t_m}(\mathbf{x}) - \tilde{G}^{\beta_{t_m}}(\mathbf{x}) \right| < \epsilon \tag{11}$$

Proof. Lemma 1 is the direct consequence of the extension of the universal approximation theorem of [16] by Leshno et al. [24], who shows that an activation function will lead to a network with universal approximation capacity, if and only if, the function (in our case φ) is not a polynomial almost everywhere.

Lemma 2. *Under the Markovian and no arbitrage assumption, and for sufficiently large p, the value of the contingent claim at $t \in (t_{m-1}, t_m]$ holds the following relation*

$$\sup_{\mathbf{S}_t \in \mathcal{I}_d} \left| V_t(\mathbf{S}_t) - \mathbb{E}\left[\tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m}) \mid \mathbf{S}_t\right] \right| < \epsilon \tag{12}$$

where $\epsilon > 0$ is given.

□

Proof. Lemma 2 is the result of the Markovian and the no-arbitrage assumptions, together with the use of Lemma 1.

$$\begin{aligned}
\left| V_t(\mathbf{S}_t) - \mathbb{E}\left[\tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m}) \mid \mathbf{S}_t\right] \right| &= \left| \mathbb{E}[V_{t_m}(\mathbf{S}_{t_m}) \mid \mathbf{S}_t] - \mathbb{E}\left[\tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m}) \mid \mathbf{S}_t\right] \right|, \\
&= \left| \mathbb{E}\left[V_{t_m}(\mathbf{S}_{t_m}) - \tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m}) \mid \mathbf{S}_t\right] \right|, \\
&\leq \mathbb{E}\left[\left| V_{t_m}(\mathbf{S}_{t_m}) - \tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m}) \right| \mid \mathbf{S}_t \right], \\
&< \mathbb{E}[\epsilon \mid \mathbf{S}_t] = \epsilon,
\end{aligned}$$

□

where the first line is the consequence of the Markovian and the no-arbitrage assumption, the second line is from the no-arbitrage assumption and the use of the linearity of the conditional expectations. The third line uses the Jensen's inequality, while the last line is the outcome of Lemma 1.

We have shown that when the value of the target contingent claim V_{t_m}, $m = 1, \ldots, M$, is known, then between monitoring dates $(t_{m-1}, t_m]$ a static portfolio can be set up that replicates the value of contingent claim. We next show that RLNN would converge to the true price of the contingent claim at any time point, including t_0. We still hold the assumptions that the underlying follows a Markovian process, the market is frictionless and there is no arbitrage.

Theorem 1. *Under the assumptions stated above, for any $\epsilon > 0$,*

$$\sup_{\mathbf{S}_{t_{m-1}} \in \mathcal{I}_d} |V_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) - \tilde{V}_{t_{m-1}}(\mathbf{S}_{t_{m-1}})| < (M - (m-1))\epsilon, \;\; m = 1, \ldots, M+1, \tag{13}$$

where $\tilde{V}_{t_{m-1}}$ is obtained by following the iteration defined by Eqs. (7) and (8).

Proof. The proof for Theorem 1 is provided in Appendix A. □

A direct consequence of Theorem 1 is that for a Bermudan option with M monitoring dates, the following holds,

$$\sup_{\mathbf{S}_{t_0} \in \mathcal{I}_d} |V_{t_0}(\mathbf{S}_{t_0}) - \tilde{V}_{t_0}(\mathbf{S}_{t_0})| < M\epsilon. \tag{14}$$

It can be seen that the errors accumulate linearly with increasing number of monitoring dates.

4.1. Lower and upper bounds

Once the neural networks in the RLNN have been trained to approximate the value functions at each monitoring date, the lower and upper bounds to the true price can be obtained in a fairly straightforward manner as discussed below. An upper and lower bound estimate for the true price, rather than just relying on the direct price estimate from the RLNN method, i.e. $\tilde{V}_{t_0}$, is important as unlike linear regression models where the coefficients can be determined exactly, with neural networks typically it is difficult to directly infer whether the coefficients have converged after a certain number of iterations of the stochastic gradient descent algorithm. A tight lower and upper bound value would then be an indicator for the quality of the RLNN price estimate.

4.1.1. Lower bound

In order to obtain a lower bound to the true price, we first simulate a new set of N_L paths following Eq. (5). The early stopping times for each path, $n = 1, \ldots, N_L$, are then determined as:

$$\tau(n) := \inf\left\{t_m \in \{t_0, \ldots, t_M\} : h(\mathbf{S}_{t_m}(n)) > \widehat{Q}_{t_m}(\mathbf{S}_{t_m}(n))\right\}, \tag{15}$$

where $\widehat{Q}_{t_m}(\mathbf{S}_{t_m}(n))$ is computed as Eq. (7), using the already obtained vector of free parameters $\beta_{t_{m+1}}$ from RLNN. An unbiased estimate of the lower bound is then obtained as

$$\underline{\tilde{V}}_{t_0} = \frac{B_{t_0}}{N_L}\sum_{n=1}^{N_L}\frac{h\left(\mathbf{S}_{\tau(n)}\right)}{B_{\tau(n)}}.$$

4.1.2. Upper bound

Haugh and Kogan [15] and Rogers [31] proposed the dual formulation for pricing Bermudan options. For an arbitrary adapted super-martingale process $\mathcal{M}_t$, it follows that,

$$\begin{aligned}
V_{t_0}(\mathbf{S}_{t_0}) &= \sup_{\tau}\mathbb{E}\left[\frac{h_\tau}{B_\tau}\right] \\
&= \sup_{\tau}\mathbb{E}\left[\frac{h_\tau}{B_\tau} + \mathcal{M}_\tau - \mathcal{M}_\tau\right] \\
&\le \mathcal{M}_{t_0} + \sup_{\tau}\mathbb{E}\left[\frac{h_\tau}{B_\tau} - \mathcal{M}_\tau\right] \\
&\le \mathcal{M}_{t_0} + \mathbb{E}\left[\max_t\left(\frac{h_t}{B_t} - \mathcal{M}_t\right)\right],
\end{aligned}$$

which gives us the upper bound of the option price $V_{t_0}(\mathbf{S}_{t_0})$. Thus, the dual problem is to minimize the upper bound with respect to all adapted super-martingale processes, i.e.,

$$\overline{V}_{t_0}(\mathbf{S}_{t_0}) = \inf_{\mathcal{M}\in\Pi}\left(\mathcal{M}_{t_0} + \mathbb{E}\left[\max_t\left(\frac{h_t}{B_t} - \mathcal{M}_t\right)\right]\right), \tag{16}$$

where Π is the set of all adapted super-martingale processes.

With, $\mathcal{M}_{t_0} = 0$, we construct a martingale process as:

$$\mathcal{M}_{t_m}(n) = \sum_{i=0}^{m-1}\left[\frac{\tilde{G}^{\beta_{t_{i+1}}}\left(\mathbf{S}_{t_{i+1}}(n)\right)}{B_{t_{i+1}}} - \frac{\widehat{Q}_{t_i}(\mathbf{S}_{t_i}(n))}{B_{t_i}}\right] \tag{17}$$

The upper bound, $\overline{V}_{t_0}$, is then given by

$$\begin{aligned}
\overline{V}_{t_0}(\mathbf{S}_{t_0}) &= \mathbb{E}\left[\max_t\left(\frac{h_t}{B_t} - \mathcal{M}_t\right)\right] \\
&= \frac{1}{N_L}\sum_{n=1}^{N_L}\max_{t_m}\left(\frac{h(\mathbf{S}_{t_m}(n))}{B_{t_m}} - \mathcal{M}_{t_m}(n)\right),\ t_m \in [t_0, \ldots, t_M]
\end{aligned} \tag{18}$$

The important point here, as was also pointed out by Glasserman and Yu [14], is that the martingale terms given by Eq. (17) are available at almost no extra cost. This is possible because we assume that $\tilde{G}^{\beta_{t_{m+1}}}$ has a closed-form conditional expectation, which by construction is equal to $\widehat{Q}_{t_m}$. In contrast, for *regress-now* schemes, in order to obtain the optimal martingale, one has to resort to sub-simulations which makes them computationally quite expensive.

Remark 1. From a practical perspective the lower bound and upper bounds can be interpreted as follows. For the holder of an early-exercise option, the difference between the true price and the lower bound gives the risk-neutral expectation

of the loss she incurs when she exercises her option following the policy obtained using the RLNN. On the other hand, the difference between the upper bound and the true price is an indicator of the expected maximum loss the writer of the option has to bear for imperfect hedging. Equation (17) can be seen as the rolling over of the semi-static hedge portfolio, where at t_i an amount $\widehat{Q}_{t_i}(\mathbf{S}_{t_i}(n)$ is used to set up a portfolio of European options that results in a payoff of $\tilde{G}^{\beta_{t_{i+1}}}(\mathbf{S}_{t_{i+1}}(n))$ at t_{i+1}. If the holder of the target option does not decide to exercise her option at t_{i+1} the proceeds from the payoff of the static hedge portfolio would be greater than or just enough to set up the static hedge, a portfolio with options maturing at t_{i+2}, i.e.

$$\tilde{G}^{\beta_{t_{i+1}}}(\mathbf{S}_{t_{i+1}}(n)) \geq \widehat{Q}_{t_{i+1}}(\mathbf{S}_{t_{i+1}}(n).$$

In case the holder of the target option decides to exercise at t_{i+1}, $\tilde{G}^{\beta_{t_{i+1}}}(\mathbf{S}_{t_{i+1}}(n))$ should ideally be greater than or equal to $h(\mathbf{S}_{t_{i+1}})$, or otherwise the writer of the target option suffers a loss. Therefore, Eq. (18) gives the expectation of the worst-case hedging loss for the writer of the target option along each simulated risk-neutral scenario.

On the practical side we show through numerical examples that RLNN can be used to obtain tight upper and lower bounds without the need for any sub-simulations.

4.2. Computing the conditional expectation

In order to compute the continuation value $\widehat{Q}_{t_{m-1}}$, a closed-form expression for Eq. (7), i.e. we need closed form expressions for

$$\mathbb{E}\Big[\varphi\Big(\mathbf{w}_i^\top \mathbf{S}_{t_m} + b_i\Big) \mid \mathbf{S}_{t_{m-1}}\Big].$$

For a single asset case, depending upon the signs of w_i, and b_i, $\mathbb{E}\Big[\varphi(w_i S_{t_m} + b_i) \mid S_{t_{m-1}}\Big]$ is the value of a call option, put option, or a forward contract. These possible cases might then arise:

Case 1: w_i and b_i are both greater than 0. Then, we have

$$\mathbb{E}[\max(w_i S_{t_m} + b_i, 0)|S_t] = \mathbb{E}\Big[(w_i S_{t_m} + b_i)|S_{t_{m-1}}\Big],$$

which is the price of a forward contract.

Case 2: $w_i > 0$ and $b_i < 0$, then we encounter

$$\mathbb{E}[\max(w_i S_{t_m} + b_i, 0) \mid S_t] = w_i \mathbb{E}\left[\max\left(S_{t_m} + \frac{b_i}{w_i}, 0\right) \mid S_{t_{m-1}}\right],$$

which is the time t_{m-1} value of a European call option, with strike $-\frac{b_i}{w_i}$, that expires at t_m.

Case 3: $w_i < 0$ and $b_i > 0$, then we find,

$$\mathbb{E}[\max(w_i S_{t_m} + b_i, 0) \mid S_t] = -w_i \mathbb{E}\left[\max\left(-\frac{b_i}{w_i} - S_{t_m}, 0\right) \mid S_{t_{m-1}}\right],$$

which is the time t_{m-1} value of a European put option, with strike $-\frac{b_i}{w_i}$, that expires at t_m.

Case 4: $w_i < 0$ and $b_i < 0$, then

$$\mathbb{E}[\max(w_i S_{t_m} + b_i, 0) \mid S_t] = 0.$$

When pricing is the objective, and the asset follows a Geometric Brownian Motion (GBM) process, the implementation can be greatly simplified by taking the log of the asset price as the input to the neural network. We then want to evaluate the following expression,

$$\mathbb{E}\Big[\varphi\Big(\mathbf{w}_i^\top \log(\mathbf{S}_{t_m}) + b_i\Big) \mid \mathbf{S}_{t_{m-1}}\Big].$$

We begin by noting that conditioned on $\mathcal{F}_{t_{m-1}}$, the random vector $\log(\mathbf{S}_{t_m})$ —under the risk-neutral measure— has a multivariate normal distribution with the mean vector given by

$$\boldsymbol{\mu} := \begin{pmatrix} \mu_1 \\ \vdots \\ \mu_d \end{pmatrix} := \log\left(\mathbf{S}_{t_{m-1}}\right) + \begin{pmatrix} r - \frac{1}{2}\sigma_1^2 \\ \vdots \\ r - \frac{1}{2}\sigma_d^2 \end{pmatrix} \Delta t,$$

and the covariance matrix by,

$$\Sigma := \begin{pmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 & \dots & \rho_{1d}\sigma_1\sigma_d \\ \rho_{21}\sigma_2\sigma_1 & \sigma_2^2 & \dots & \rho_{2d}\sigma_2\sigma_d \\ \vdots & \vdots & \ddots & \vdots \\ \rho_{d1}\sigma_d\sigma_1 & \rho_{d2}\sigma_d\sigma_2 & \dots & \sigma_d^2 \end{pmatrix} \Delta t,$$

where $\Delta t = t_m - t_{m-1}$.

We denote the conditional weighted sum of the log asset prices by a random variable Y. Then Y has a normal distribution with mean, $\mu_Y := \mathbf{w}_i^\top \boldsymbol{\mu}$, and variance, $\sigma_Y^2 := \mathbf{w}_i^\top \Sigma \mathbf{w}_i$.

We need to then determine

$$\begin{aligned}\mathbb{E}\left[\varphi\left(\mathbf{w}_i^\top \log\left(\mathbf{S}_{t_m}\right) + b_i\right) \mid \mathbf{S}_{t_{m-1}}\right] &= \mathbb{E}[\max\left(Y + b_i, 0\right)] \\ &= \mathbb{E}\left[\max\left(\tilde{Y}, 0\right)\right],\end{aligned}$$

where $\tilde{Y}$ is a normal with mean $\mu_{\tilde{Y}} = \mu_Y + b_i$, and variance σ_Y.The expression is straightforward to evaluated and is given by

$$\mathbb{E}\left[\max\left(\tilde{Y}, 0\right)\right] = \frac{1}{\sqrt{2\pi\sigma_{\tilde{Y}}^2}} \int_0^\infty y e^{\frac{-\left(y-\mu_{\tilde{Y}}\right)^2}{2\sigma_{\tilde{Y}}^2}}\, dy \tag{19}$$

$$= \frac{1}{\sqrt{2\pi\sigma_{\tilde{Y}}^2}} \int_{-\mu_{\tilde{Y}}}^\infty \left(x + \mu_{\tilde{Y}}\right) e^{\frac{-x^2}{2\sigma_{\tilde{Y}}^2}}\, dx \tag{20}$$

$$= \frac{1}{\sqrt{2\pi\sigma_{\tilde{Y}}^2}} \int_{-\mu_{\tilde{Y}}}^\infty (x) e^{\frac{-x^2}{2\sigma_{\tilde{Y}}^2}}\, dx + \frac{1}{\sqrt{2\pi\sigma_{\tilde{Y}}^2}} \int_{-\mu_{\tilde{Y}}}^\infty \left(\mu_{\tilde{Y}}\right) e^{\frac{-x^2}{2\sigma_{\tilde{Y}}^2}}\, dx \tag{21}$$

$$= \frac{\sigma_Y}{\sqrt{2\pi}} e^{\frac{-\mu_{\tilde{Y}}^2}{2\sigma_{\tilde{Y}}^2}} + \mu_{\tilde{Y}}\left(1 - F_{\tilde{Y}}(-\mu_{\tilde{Y}})\right), \tag{22}$$

where $F_{\tilde{Y}}$ is the cdf of $\tilde{Y}$.

It should be noted that a more generic approach, that is applicable to wider sets of models, would be to use the Brownian increment $(W_{t_{m+1}} - W_{t_m})$ as the input, rather than the log asset prices.

5. Numerical examples

In this section we illustrate the performance of the RLNN by pricing path-dependent and path-independent contingent claims on single and several assets. We through experiments illustrate the two major themes covered in this paper. First, we look at the pricing of Bermudan options for different numbers of underliers and different payoff structures. Specifically, we consider the pricing of the following Bermudan options, put on a single asset, arithmetic basket put option on five assets, and max call option on two, three and five assets, respectively. We also construct the upper and lower bounds for these examples using the methodology discussed in Section 4.1. Second, through numerical examples, we compare the performance of dynamic hedging against the semi-static hedging technique discussed here, for a single asset European and barrier option case.

For the examples considered here, we assume that the d underlying assets follow a multivariate geometric Brownian motion, i.e.

$$\frac{dS_t^\delta}{S_t^\delta} = (r - q_\delta)dt + \sigma_\delta dW_t^\delta,\ \delta = 1, \ldots, d, \tag{23}$$

where r is the constant risk-free rate, q_δ is the continuous dividend rate for the δth asset. W_t^δ is the standard Brownian motion and the instantaneous correlation coefficient between W_t^i and W_t^j is ρ_{ij}.

All the experiments were carried out on a machine with Intel Xeon Gold 6130 processor with 128 GB RAM.

5.1. Choice of hyper-parameters and other considerations

The choice of appropriate hyper-parameters for training the neural network can affect the rate of convergence and therefore should be carefully picked. For the experiments we conducted we made the following choices.

- We use Adam [20] with initial leaning rate as 10^{-3}, as the optimizer for the weights update in the mini-batch gradient ascent algorithm. We choose the default parameters for Adam as suggested in [20].
- The batch size is chosen as one-tenth of the total training points. Varying batch sizes impacted the computational time, as large batches could potentially benefit from parallel processing, while smaller batch sizes allowed greater updates per epoch. The above batch size was selected as it converged relatively faster compared to a few different choices of batch sizes that we experimented with.
- For training the neural network corresponding to the monitoring date $t_M = T$, we initialize the weights and biases randomly with uniform random variables, default initializer in Keras.

- As $\tilde{V}_{t_{m-1}} \approx \tilde{V}_{t_m}$ we transfer the final weights obtained from training the network for monitoring date t_m as the initial weights for training the network at t_{m-1}.
- In order to avoid over-fitting, we divide the training points into a training set and a validation set, in the ratio 0.7 to 0.3 respectively and use the mean squared error of the validation set as the early-stopping criterion with a patience of 6 epochs.
- We normalize the initial asset price to 1 and appropriately adjust the strike. This restricts the domain in which the network needs to be trained, something we found especially beneficial while training the network for max options. Normalization is a standard practice while training neural networks, as we can then follow a uniform approach for initialization of the weights of the network.
- We use 50,000 training points generated using the GBM process under the risk-neutral measure.

The codes for our experiments have been implemented in Python with the Keras library (with Tensorflow backend) used for the implementation of the neural network[2]. For obtaining the upper and lower bound estimates we generate a fresh set of 200,000 scenarios for each of the example considered.

5.2. Down-and-out barrier call option

We first consider a discretely monitored down-and-out call options, i.e., $S_{t_0} > H$, where H is the barrier level, and the option becomes worthless if S crosses the barrier before the option expires at time T. We denote the first time the asset crosses the barrier as:

$$\tau = \inf\{t_m > 0 : S_{t_m} < H\}$$

The price of the discrete down-and-out call option is then given by:

$$V_{t_0} = \mathbb{E}\Big[(S_{t_M} - K)^+ | \tau > t_M\Big].$$

The backward iteration for RLNN to compute the barrier price would be as follows.We start with the expiry, t_M, where the value of the claims would be given by

$$V_{t_M} = (S_{t_M} - K)^+ \mathbb{1}_{S_{t_M} > H}$$

Given the approximations for the claims value for the paths at t_m, i.e. $\widehat{V}_{t_m}(S_{t_m})$ the neural network is trained to obtain the approximate value function $\tilde{G}^{\beta_{t_m}}(S_{t_m})$. The claims value at t_{m-1} is then computed as:

$$\widehat{V}_{t_{m-1}}(S_{t_{m-1}}) = \widehat{Q}_{t_{m-1}}(S_{t_{m-1}})\mathbb{1}_{S_{t_{m-1}} > H},$$

where,

$$\widehat{Q}_{t_{m-1}}(S_{t_{m-1}}) = \mathbb{E}\Big[\tilde{G}^{\beta_{t_m}}(S_{t_m}) | S_{t_{m-1}}\Big].$$

If the objective is to determine the composition of the static hedge the following form of $\tilde{G}$ is more suitable,

$$\tilde{G}^{\beta_{t_m}}(S_{t_m}) = \sum_{i=1}^{p} \omega_i \varphi(w_i S_{t_m} + b_i) + b_2,$$

while if pricing is the objective then we recommend to use the following:

$$\tilde{G}^{\beta_{t_m}}(S_{t_m}) = \sum_{i=1}^{p} \omega_i \varphi(w_i \log(S_{t_m}) + b_i) + b_2,$$

as a more convenient expression for computing $\widehat{Q}_{t_{m-1}}(S_{t_{m-1}})$ can then be obtained, as described in Section 4.2.

Table 2 compares the value of the down and out discrete barrier obtained using the RLNN against the reference value, determined from a trinomial procedure as described in [9], for different barrier levels. Also reported are the approximation of the discrete barrier obtained using the continuity correction (CC) as described in [8]. Continuity correction for pricing discretely monitored barrier options is attractive, especially when the barrier level is in close proximity to the initial asset price, as traditional Monte Carlo method would require significantly large number of paths to obtain a tight bound for the estimated price. For the Barrier case, regress-now (RN) scheme boils down to vanilla Monte Carlo estimation. We observe that the RLNN computes fairly accurate prices and has lower standard errors (and therefore tighter bounds) than the corresponding values for RN. For most cases the results from RLNN are comparable or better than ones obtained using the continuity correction. The reported run time for the RLNN scheme was 75s on an average. The run time is relatively higher compared to the Arithmetic Basket Option case as the pay-off has a discontinuity which requires significant iterations to fit. In general the run time would also be highly dependent upon the initialization of the network.

[2] The source code for the experiments are available at: http://github.com/Vikranth1508/FinML/tree/neural_hedge

Table 1
Parameter values used in the examples.

Set I
$S^1_{t_0} = 100,\ K = 100,\ r = 0.1,\ \sigma = 0.3,\ T = 0.2,\ M = 5.$
Set II
$S^\delta_{t_0} = 1,\ K = 1,\ r = 0.05,\ q_\delta = 0,\ T = 1,\ M = 10.$
$\sigma = [0.518, 0.648, 0.623, 0.570, 0.530]^\top$

$$\rho := \begin{pmatrix} 1.00 & 0.79 & 0.82 & 0.91 & 0.84 \\ 0.79 & 1.00 & 0.73 & 0.80 & 0.76 \\ 0.82 & 0.73 & 1.00 & 0.77 & 0.72 \\ 0.91 & 0.80 & 0.77 & 1.00 & 0.90 \\ 0.84 & 0.76 & 0.72 & 0.90 & 1.00 \end{pmatrix}$$

Set III
$S^\delta_{t_0} = 100,\ K = 100,\ r = 0.05,\ q_\delta = 0.1,\ \sigma_\delta = 0.2,\ \rho_{ij} = 0.0,\ T = 3,\ M = 9.$
Set IV
$S^1_{t_0} = 1,\ K = 1,\ r = 0.1,\ \sigma_1 = 0.3,\ T = 1.$
Set V
$S^1_{t_0} = 1,\ K = 1,\ r = 0.1,\ \sigma_1 = 0.3,\ T = 0.2,\ M = 5.$

Table 2
Price of a down and out barrier call on a single asset for different barrier levels, with the parameter values taken from Set I in Table 1. Both RLNN and RN use 50,000 independent antithetic paths, and RLNN uses 32 nodes in the hidden layer. The standard errors for both the methods are computed from 30 independent runs.

H	RLNN Direct est. (s.e.)	RN (s.e.)	CC	Reference Value
85	6.337 (0.0006)	6.339 (0.0281)	6.337	6.337
87	6.32 (0.0012)	6.314 (0.0274)	6.323	6.321
89	6.277 (0.0004)	6.280 (0.0331)	6.284	6.281
91	6.177 (0.0008)	6.187 (0.0219)	6.194	6.187
93	5.977 (0.0015)	5.997 (0.0278)	6.004	6.000
95	5.632 (0.0048)	5.671 (0.0298)	5.646	5.671
97	5.107 (0.0057)	5.158 (0.0317)	5.028	5.167
99	4.406 (0.0086)	4.487 (0.0343)	4.050	4.489

5.3. Arithmetic basket option

The weighted basket put option payoff reads,

$$\max\left(h(\mathbf{S}_T), 0\right) = \max\left(K - \sum_{i=1}^{d} w_i S^i_T, 0\right).$$

The model parameters are the same ones as used by Reisinger and Wittum [30] to approximate the DAX index, and correspond to Set 2 in Table 1. The weights for the basket are taken as

$$w = [0.381, 0.065, 0.057, 0.270, 0.227]^\top.$$

We consider a Bermudan put option that expires at $T = 1$ year and has an early-exercise feature on $M = 10$ equally spaced time points between $t = 0$ and $t = T$.

When the input to the neural network at t_m are the asset prices $\mathbf{S}_{t_m}$, the output of the network is a portfolio of European arithmetic basket options that expires at t_m. While fast numerical approximations and semi-analytical expressions for computing the price of European arithmetic basket options are available, the problem can be greatly simplified — when the underlying assets follow the GBM process— if the log asset prices are taken as the input to the neural network. With log asset prices as the input, the output of the network at t_m translates to the payoff of a portfolio of European geometric basket options that mature at t_m. In order to compute the continuation value, one needs to compute the following:

$$\widehat{Q}_{t_{m-1}}(S_{t_{m-1}}) = \mathbb{E}\left[\tilde{G}^{\beta_{t_m}}(\log(\mathbf{S}_{t_m})) \mid \mathbf{S}_{t_{m-1}}\right]$$

Table 3
The RLNN direct estimator, upper, and lower bound values of a Bermudan arithmetic basket put option on five assets when 64 hidden units are used in the RLNN method. For obtaining the lower and upper bound values we use 200,000 independent paths. The parameters for the model and the option are taken from Set II in Table 1. The reference values are obtained using the SGBM method [19].

S_0	RLNN Direct est. (s.e.)	RLNN Lower Bound. (s.e.)	RLNN Upper Bound (s.e.)	RLNN 95% CI	SGBM
0.9	0.2222 (0.0001)	0.2221 (0.0005)	0.2223 (0.00001)	[0.2219, 0.2223]	0.2220
1	0.1804 (0.00009)	0.1803 (0.0003)	0.1805 (0.00001)	[0.1802, 0.1805]	0.1803
1.1	0.1464 (0.0001)	0.1464 (0.0003)	0.1464 (0.00001)	[0.1463, 0.1464]	0.1463

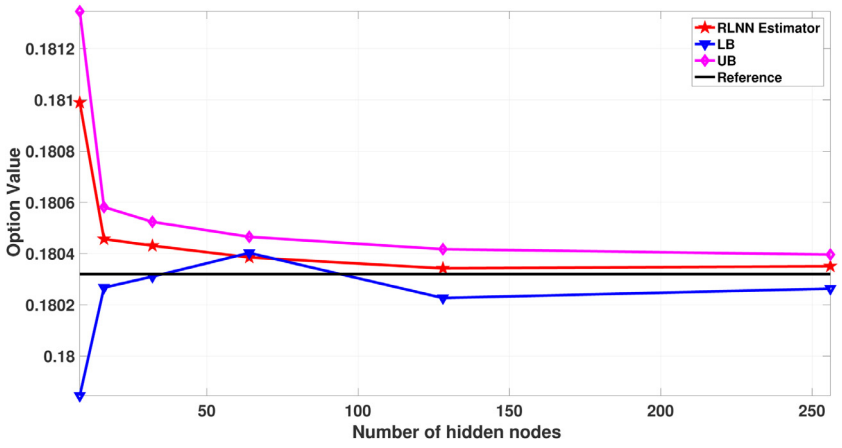

Fig. 2. The upper and lower bound values for a Bermudan arithmetic basket put option, when an increasing number of hidden units are used in the first layer of RLNN. The parameters for the model and the option are taken from Set II in Table 1. The reference value is obtained using SGBM method and is equal to 0.1803. Also plotted are the corresponding direct RLNN estimated values, i.e. $\tilde{V}_{t_0}$.

$$= \mathbb{E}\left[\sum_{i=1}^{p} \omega_i \varphi\left(\mathbf{w}_i^\top \log\left(\mathbf{S}_{t_m}\right) + b_i\right) + b_2 \mid \mathbf{S}_{t_{m-1}}\right]$$
$$= \sum_{i=1}^{p} \omega_i \mathbb{E}\left[\varphi\left(\mathbf{w}_i^\top \log\left(\mathbf{S}_{t_m}\right) + b_i\right) \mid \mathbf{S}_{t_{m-1}}\right] + b_2.$$

In Section 4.2 we provide the expression for $\mathbb{E}\left[\varphi\left(\mathbf{w}_i^\top \log\left(\mathbf{S}_{t_m}\right) + b_i\right) \mid \mathbf{S}_{t_{m-1}}\right]$, which is required for evaluation of $\widehat{Q}_{t_{m-1}}$.

Table 3 provides the lower and upper bound values for the Bermudan arithmetic basket put option obtained using the RLNN method. The reported standard errors are obtained using results from 30 independent runs. The reference values are obtained using the SGBM method [19]. We again notice that the standard errors for the duality-based upper bound values are orders of magnitude smaller than others. The runtime for the 64 node case was 5 s on an Intel Xeon Gold 6130 processor. Figure 2 illustrates the convergence of the results for the arithmetic basket case when an increasing number of hidden units is used.

We next consider a special case of arithmetic basket option, where the asset prices are perfectly correlated. For the co-monotonic case, the arithmetic basket can be reformulated as a basket of single asset European options with the appropriately computed strikes, as described in [11]. Once the arithmetic basket has been broken into a basket of single asset European options, a static hedge could be obtained using the Carr-Wu approach, and an accurate reference price can then be obtained. As often non-linear regression tends to be unstable when the input data has comonotonicity we test the performance of the RLNN for this case. Table 4 reports the value of the European arithmetic basket put option when the underlying assets are comonotonic. The results from RLNN and MC are close to each other with a lower standard error for RLNN. Both MC and RLNN are within few basis points of the true price.

5.4. Max option

We next consider a Bermudan max call option, whose payoff at t is given by:

$$\max\left(h(\mathbf{S}_t), 0\right) = \max\left(\max\left(S_t^1, \ldots, S_t^d\right) - K, 0\right).$$

Table 4
The price of an arithmetic basket put option when the underlying assets are comonotonic. The true price is obtained by evaluating the weighted price of the corresponding basket of single asset put options, as described in [11]. The parameter values from this case are taken from Set II of Table 1 with correlation matrix set to 1 for each element.

RLNN Direct est. (s.e.)	MC . (s.e.)	True
18.732 (0.0149)	18.731 (0.0912)	18.745

Table 5
Bermudan option values for a call on the maximum of 2, 3 and 5 assets, with parameter values taken from Set III, in Table 1. For the two asset case we use 256 hidden units, for three assets 512 hidden units, while for five assets we use 1024 hidden units. The reference confidence interval for the two and three asset cases are taken from Andersen and Broadie [2], and for the five asset case from Broadie and Cao [7].

S_0	RLNN Direct est. (s.e.)	RLNN Lower Bound. (s.e.)	RLNN Upper Bound (s.e.)	RLNN 95% CI	Literature 95% CI	Binomial
d=2 assets:						
90	8.078 (0.016)	8.071 (0.025)	8.086 (0.0003)	[8.062, 8.086]	[8.053, 8.082]	8.075
100	13.902 (0.017)	13.905 (0.028)	13.924 (0.0004)	[13.894, 13.924]	[13.892, 13.934]	13.902
110	21.346 (0.010)	21.352 (0.029)	21.353 (0.0002)	[21.341, 21.353]	[21.316, 21.359]	21.345
d=3 assets:						
90	11.282 (0.017)	11.295 (0.030)	11.303 (0.001)	[11.287, 11.303]	[11.265, 11.308]	11.29
100	18.702 (0.022)	18.688 (0.025)	18.715 (0.001)	[18.677, 18.715]	[18.661, 18.728]	18.69
110	27.572 (0.021)	27.554 (0.023)	27.592 (0.001)	[27.545, 27.592]	[27.512, 27.663]	27.58
d=5 assets:						
90	16.680 (0.063)	16.636 (0.044)	16.743 (0.003)	[16.624, 16.744]	[16.620, 16.653]	
100	26.177 (0.062)	26.141 (0.034)	26.268 (0.002)	[26.125, 26.270]	[26.115, 26.164]	
110	36.815 (0.042)	36.760 (0.045)	36.909 (0.003)	[36.744, 36.909]	[36.710, 36.798]	

The parameters for this case correspond to Set III in Table 1. Similar to the case of the Bermudan arithmetic basket option, one can either use the asset prices as inputs to the neural network, in which case the output would correspond to a portfolio of short maturity arithmetic basket options. When the underlying assets follow a GBM process, it might be more convenient to use the log asset prices as the input to the neural network, in which case the output of the network corresponds to a static hedge with a portfolio of geometric basket options.

The computation of the continuation value remains the same as that for the Bermudan arithmetic basket case. Unlike the linear regression-based models, with RLNN we do not have to make payoff specific choices for basis functions, which greatly simplifies the implementation and maintenance of the pricing library.

Table 5 provides the lower and upper bounds for the Bermudan max option on 2, 3 and 5 assets and compares the results with benchmark values reported in [7] and [2]. The runtime for the five asset case when 1024 nodes are used was 8 min on an Intel Xeon Gold 6130 processor. The convergence of the results to the reference value with an increasing number of hidden units is illustrated in Fig. 3. Notice that, as the accuracy of the approximated early exercise policy improves, the unbiased error from the finite sample size of the Monte Carlo estimator dominates the biased error due to the use of a sub-optimal exercise policy, and therefore we see instances where the estimated lower bound values are higher than the true price.

In the case of ordinary least squares based linear regression, for a given number of samples, there are finite number of operations to obtain the unknown coefficients of the parametric linear function. However, in the case of the neural network, the number of iterations required by the stochastic gradient descent algorithm to obtain the weights of the network, to achieve a given level of accuracy, cannot be determined apriori. In Table 6 we report the computational time for the Bermudan max call on 5 assets case where a varying level of accuracy was achieved using increasing number of neurons in the hidden layer.

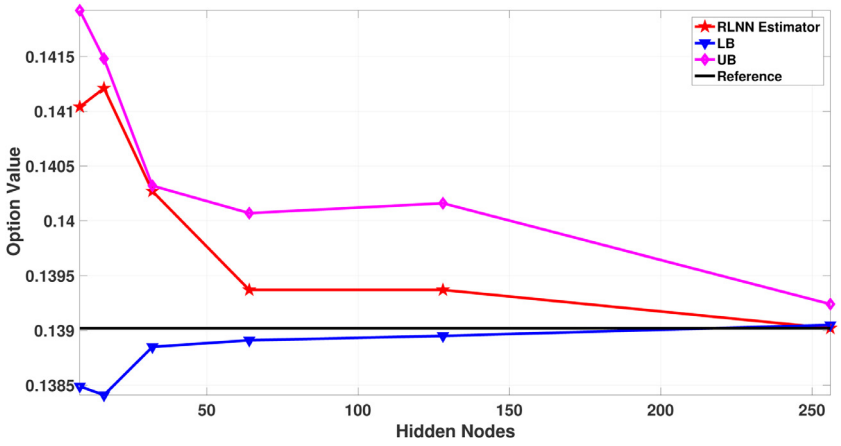

Fig. 3. The upper and lower bound values for a Bermudan call on the maximum of two assets when an increasing number of hidden units are used in the first layer of RLNN. The parameters for the model and the option are taken from Set III in Table 1. The reference value is obtained from a binomial tree and is equal to 13.902. Also plotted are the corresponding direct RLNN estimator values, i.e. $\tilde{V}_{t_0}$.

Table 6
The computational time for the ATM Bermudan call option on 5 asset (Set III Table 1) reported in seconds for decreasing error levels. Errors are controlled by increasing the number of neurons used in the hidden layer. Column RLNN reports the time for the direct estimator, LB+UB reports the time to compute the upper and lower bounds, and the final column reports the total time taken in seconds.

Neurons	Error (%)	RLNN (s)	LB+UB (s)	Total (s)
16	4.20	53	30	83
32	3.00	83	31	113
64	2.00	68	42	110
128	1.00	93	61	153
256	0.50	77	87	164
512	0.10	96	159	255

5.5. Performance of the static hedge

We here, first conduct numerical experiments to compare the static hedge portfolio obtained using the RLNN, with the one obtained using the Carr-Wu [10] approach, for a single asset case. The target option to be hedged is a European call with strike, $K = 100$, and a maturity, $t_M = 1$ year. The maturity of the short term options to be used to hedge the target option are taken to be, $t_1 = 0.25$ years. The annualized volatility, σ, of the asset price is 0.27, the constant risk free rate, $r = 0.06$, and the annualized dividend yield is $\delta = 0.02$. The current asset price, $S_{t_0} = 100$. We assume the underlying asset follows the geometric Brownian motion.

Case 1 Unconstrained weights : In this case we assume that the weights and biases in the neural network are unconstrained, i.e, the options used in the static hedge portfolio could be either call, put, forward contract, or dead option as discussed in Section 4.2. The weights of the inner layer are initialized randomly from a uniform distribution between $[-0.1, 0.1]$, and the biases as 0. Figure 4 compares the RLNN and the Carr Wu absolute static hedge errors at t_1 for four independent runs. Note that a separate independent set of paths are used for the training of the networks in RLNN, and for testing the performance of the static hedge. For the experiment we considered 15 nodes in the hidden layer and used 15 short maturity options for the Carr-Wu static hedge. It was observed that in the case of RLNN nearly half of the nodes become dead nodes (are never activated) due to our choice of initialization. The RMSE errors for the RLNN were slightly lower than the Carr Wu static hedge.

Case 2 Constrained weights : When S_t is directly used as input to the neural network, the usual random initialization method results in several dead nodes. This would not be a problem, either if we use log asset prices as input to the network, or we constrain the weights of the hidden layer to be positive. In this example we constrain the weights of the hidden layer to be positive, while the weights of the output layer are unconstrained. Figure 5 compares the RLNN and the Carr-Wu absolute static hedge errors at t_1. It can be seen that the RMSE values for the RLNN are much lower in this case when compared to the corresponding Carr-Wu hedge values. While Carr-Wu use a Gaussian quadrature to arrive at the strikes for the 15 short maturity options, in the case of the RLNN, the strikes and portfolio weights are calibrated to minimize the mean squared error between the target and the hedge value. We however, observe that the discounted expected portfolio

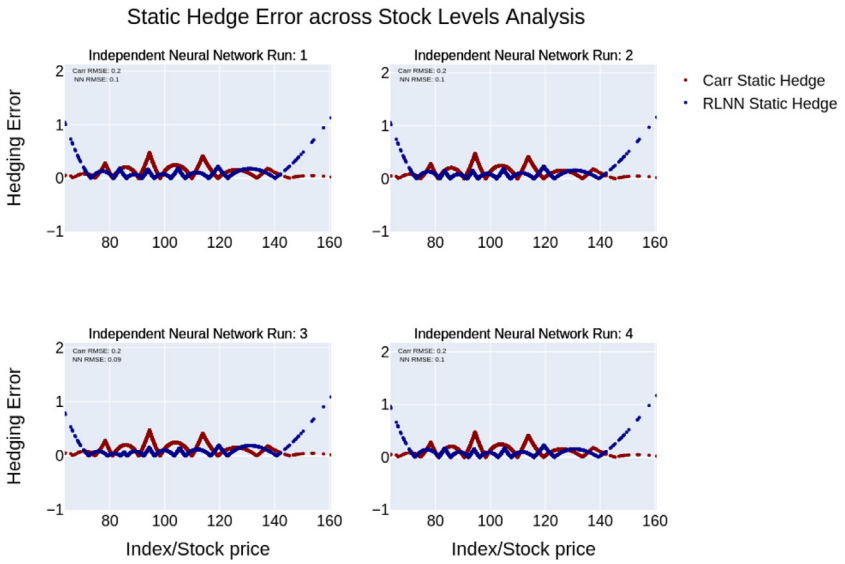

Fig. 4. Absolute hedge errors for the simulated S_{t_1} samples for four independent runs for Carr Wu static hedge and the RLNN static hedge, when weights for the hidden layer are unconstrained.

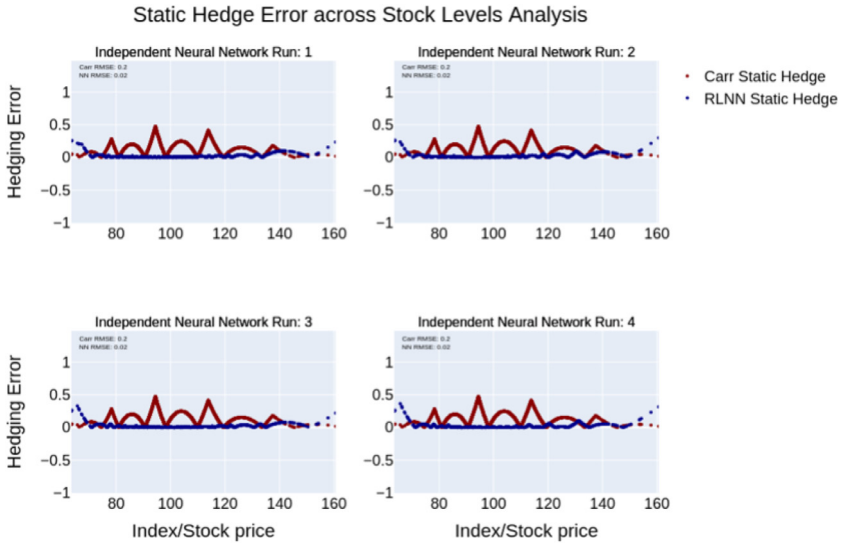

Fig. 5. Absolute hedge errors for the simulated S_{t_1} samples for four independent runs for Carr Wu static hedge and the RLNN static hedge, when weights of the hidden layers in RLNN are constrained to be positive.

value for the Carr Wu hedge portfolio converges to the true target option price with fewer options (as it is a deterministic method). Table 7 reports the t_0 discounted hedge portfolio values for Carr Wu, RLNN with unconstrained weights, and RLNN with constrained weights.

As the parameters in the neural network are searched using the stochastic gradient descent, there is an element of noise in the parameter values obtained. Figure 6 plots the histogram of the strikes obtained from different runs when the weights of the hidden layer are constrained to be positive. We see that fairly consistent results are arrived at for the different runs.

Comparison with dynamic hedging

Here we consider the problem faced by the writer of a European vanilla put option with a maturity of one year, and a down-and-out call barrier option with maturity of 0.2 years. In the first case the writer wants to hold a short position on the European put option for a month, after which the option position is closed. The writer has the option to hedge the risk using either a dynamic delta hedging strategy, or using the static hedge portfolio of short maturity vanilla options constructed using the RLNN method. The dynamic hedging strategy involves daily rebalancing, in our example we use 25 equally distributed points between 0 to 1 month (i.e. $\frac{1}{12}$ year fraction). The second case involves the writer of a discretely monitored down-and-out-call barrier option, with a maturity of 60 days and 5 equally spaced monitoring dates, who wants

Table 7
The estimated V_{t_0} value for the target option for Carr Wu, RLNN with constrained weights, and RLNN with unconstrained weights when increasing number of short maturity options are used in the hedge portfolio. The true price is 12.3538.

	Carr Wu	RLNN UnConstrained	RLNN Constrained
		5 options	
V_{t_0}	12.318	12.346	12.349
RMSE t_1	0.606	0.649	0.093
		9 options	
V_{t_0}	12.353	12.355	12.351
RMSE t_1	0.328	0.152	0.037
		15 options	
V_{t_0}	12.354	12.353	12.350
RMSE t_1	0.195	0.095	0.020

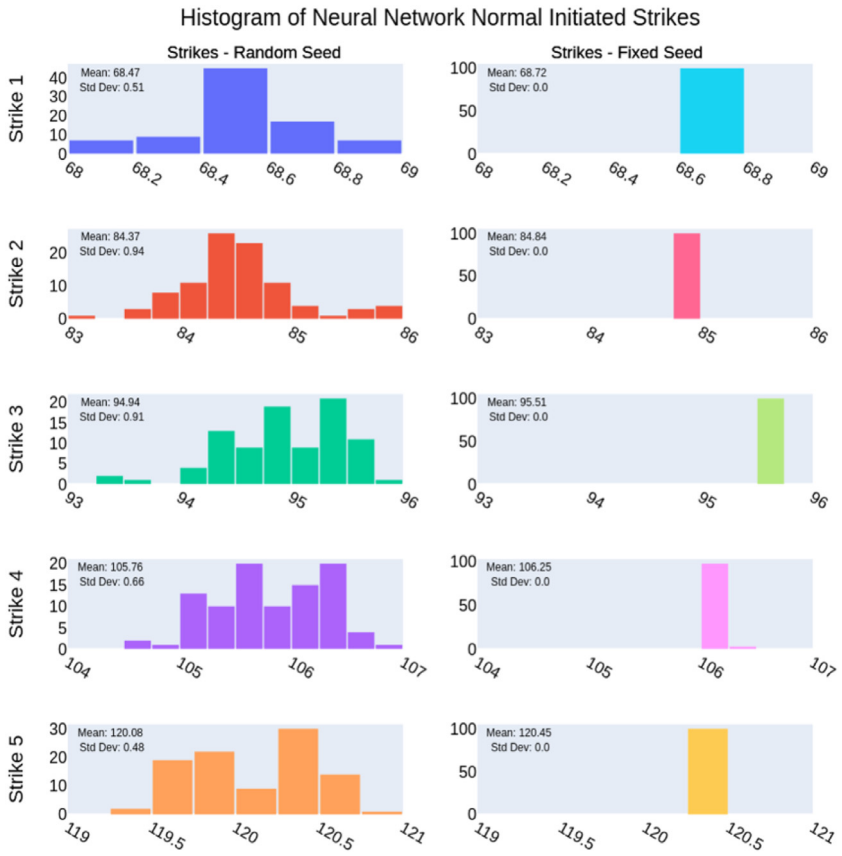

Fig. 6. Left: Histogram of the 5 strikes for 100 runs when different seeds are used for the random initialization of the weights of the network. Right is when the seed used for initialization is fixed. The corresponding Carr Wu strikes are: 48.41, 68.78, 94.43, 129.65, and 184.17 respectively.

to hold a short position for 12 days. She has a choice of hedging her risks using the static hedge portfolio constructed using RLNN or through dynamic delta hedging with daily rebalancing.

The comparison for the above two cases here is based on Monte Carlo simulations, with 50,000 paths, where the paths are generated following the Black-Scholes model under the risk-neutral measure. In the limiting case both dynamic and static hedging results should work perfectly. However, we here consider the performance when only a fixed number of short maturity options are used for the static hedge, and the portfolio is rebalanced on finite time points for the dynamic hedge. Table 8 outlines the results of the experiment for the European put option case. We see that the dynamic and static hedge strategies have similar performance, although the static hedge gives better results when it comes to VaR and conditional VaR. Also evident is that the performance of the static hedge improves when an increasing number of short

Algorithm 1 Regress-later with Neural Networks (RLNN).

Generate $\mathbf{S}_{t_m}(n)$ for paths $n = 1, \ldots, N$, $m = 0, \ldots, M$
$\tilde{V}_{t_M} \leftarrow h\left(\mathbf{S}_{t_M}\right)$ evaluate final time option value for each path.
Initialize β_{t_M} from the uniform distribution.
for $m = M \ldots, 1$ **do**
  $\beta_{t_m} \leftarrow_{\beta_{t_m}} \left(\frac{1}{N}\sum_{n=1}^{N}\left(\tilde{V}_{t_m}(\mathbf{S}_{t_m}(n)) - \tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m}(n))\right)^2\right)$ fitting the network
  for $n = 1, \ldots, N$ **do**
    $\widehat{Q}_{t_{m-1}}\left(\mathbf{S}_{t_{m-1}}(n)\right) \leftarrow B_{t_{m-1}}\mathbb{E}\left[\frac{\tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m})}{B_{t_m}} \mid \mathbf{S}_{t_{m-1}}(n)\right]$ the estimated continuation value
    if $h(\mathbf{S}_{t_{m-1}}(n)) > \widehat{Q}_{t_{m-1}}\left(\mathbf{S}_{t_{m-1}}(n)\right)$ **then**
      $\tilde{V}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}(n)) \leftarrow h(\mathbf{S}_{t_{m-1}}(n))$
    else
      $\tilde{V}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}(n)) \leftarrow \widehat{Q}_{t_{m-1}}\left(\mathbf{S}_{t_{m-1}}(n)\right)$
    end if
  end for
  $\beta_{t_{m-1}} \leftarrow \beta_{t_m}$ initialize the parameters for network to be trained at t_{m-1}
end for

Table 8
Hedge Performance of European Put Options: This table compares the hedging performance of the dynamic and static hedging (constructed using the RLNN) strategies for a European Vanilla put option for a duration of one month. The results are reported for varying levels of moneyness (K/S). The parameters for the model and the barrier are taken from Set IV in Table 1.

			Moneyness (K/S)		
Hedging Error	Hedge Type	Options Count	0.5	1	1.5
		10	-9e-06	-4e-06	-1e-05
	Static	25	-5e-07	1e-06	3e-05
Mean		50	1e-07	-2e-06	7e-06
	Dynamic	NA	5e-07	-7e-06	1e-05
		10	2e-05	0.0010	-0.0090
	Static	25	1e-05	0.0002	0.0025
Standard Deviation		50	1e-05	0.0002	0.0003
	Dynamic	NA	3e-05	0.0013	0.0011
		10	5e-05	0.0017	0.0013
	Static	25	1e-05	0.0004	0.0036
VaR(95%)		50	2e-05	0.0004	0.0004
	Dynamic	NA	6e-05	0.0026	0.0021
		10	7e-05	0.0031	0.0033
	Static	25	2e-05	0.0006	0.0079
CVaR (95%)		50	3e-05	0.0007	0.0010
	Dynamic	NA	0.0001	0.0033	0.0027

maturity options are held in the static hedge portfolio. Therefore, when the writer of the option is particularly concerned about avoiding large losses, he should prefer the static strategy.

Table 9 outlines the results of the experiment for the discretely monitored down-and-out call barrier case. We see that the dynamic and static hedge strategies have similar performance for lower barrier values, but with an increasing barrier level the static hedge seems to have slight advantage over dynamic hedging. The poor performance of dynamic hedging close to the barrier is due to the sharp change in the delta values as one gets closer to the barrier level. It should be noted that as the pay-off is not continuous, in the case of the barrier, the neural network is not guaranteed to fit the pay-off arbitrarily well. However, from a practical perspective, the fit is reasonably good as can be seen from the lower static hedge errors. The static hedge, in most cases, gives better results when it comes to VaR and conditional VaR.

6. Conclusion

In this paper we have developed a fully interpretable neural network based pricing method for path-dependent multi-asset contingent claims. The main theoretical contribution is that we show, under the Markovian and the no-arbitrage assumptions, any contingent claim —with continuous payoffs— can be semi-statically hedged using a finite number of shorter maturity options. The structure and the quantity of the options to be held at different monitoring dates in the static hedge portfolio are the outcomes of the RLNN method. We show that using RLNN one can obtain lower and upper bounds to the true value of the claim and the quality of the bounds is demonstrated through numerical examples. An advantage of the

Table 9
Hedge Performance of Down and Out Barrier Call Options: This table compares the performance of dynamic and static hedging strategies for a 60-days down-and-out call barrier options at varied levels of barrier for a duration of 12 days. The parameters for the model and the barrier are taken from Set V in Table 1.

			Barrier			
Hedging Error	Hedge Type	Options Count	0.91	0.93	0.95	0.97
		5	-0.0001	-4e-5	-1e-5	-2e-5
	Static	10	-0.0002	1.6e-5	-1e-5	5e-5
Mean		20	-0.0001	-6e-5	-8e-5	8e-5
	Dynamic	NA	-0.0001	-0.0002	-0.0001	0.0008
		5	0.0028	0.0037	0.0038	0.0042
	Static	10	0.0036	0.0046	0.0038	0.0042
Standard Deviation		20	0.0024	0.0026	0.0038	0.0042
	Dynamic	NA	0.0028	0.0043	0.0076	0.0132
		5	0.0062	0.0062	0.0077	0.0090
	Static	10	0.0076	0.0083	0.0077	0.0089
VaR(95%)		20	0.0049	0.0048	0.0076	0.0089
	Dynamic	NA	0.0048	0.0093	0.0168	0.0249
		5	0.0100	0.0114	0.0106	0.0129
	Static	10	0.0122	0.0128	0.0106	0.0127
CVaR(95%)		20	0.0088	0.0091	0.0106	0.0128
	Dynamic	NA	0.0085	0.0153	0.0263	0.0461

RLNN is that, using it a duality-based upper bound can be obtained without the need of sub-simulations. Through simulations we demonstrate that the static hedge constructed using the RLNN performs as effectively or better than delta hedging with daily rebalancing. We see the analysis of the performance of the static hedge under more realistic conditions as an important extension to this line of research. A drawback of neural network for regression is that the number of operations required to attain a certain level of accuracy is not deterministic and dependent upon the choice of initialization. A step ahead would be to exploit the possibility of massive parallelization, using GPUs, of gradient calculation within batches, or following [23], GPU based parallelization of the underlying regress later scheme.

Appendix A. Proof for Theorem 1

We prove Theorem 1 by induction.

At the maturity of the contingent claim the proof for $V_{t_M} = \tilde{V}_{t_M}$, is trivial, as $\widehat{Q}_{t_M}$ is equal to 0 and the payoff h is deterministic and known.

Assuming,

$$\sup_{\mathbf{S}_{t_m} \in \mathcal{I}_d} |V_{t_m}(\mathbf{S}_{t_m}) - \tilde{V}_{t_m}(\mathbf{S}_{t_m})| < (M-m)\epsilon,$$

and by the application of Lemma 1, i.e.

$$\sup_{\mathbf{S}_{t_m} \in \mathcal{I}_d} |\tilde{V}_{t_m}(\mathbf{S}_{t_m}) - \tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m})| < \epsilon,$$

we can then show with the use of the triangular inequality that,

$$\begin{aligned} |V_{t_m}(\mathbf{S}_{t_m}) - \tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m})| &\leq |V_{t_m}(\mathbf{S}_{t_m}) - \tilde{V}_{t_m}(\mathbf{S}_{t_m})| + |\tilde{V}_{t_m}(\mathbf{S}_{t_m}) - \tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m})| \\ &< (M-m)\epsilon + \epsilon = (M-(m-1))\epsilon. \end{aligned} \tag{A.1}$$

We next want to show that,

$$\sup_{\mathbf{S}_{t_{m-1}} \in \mathcal{I}_d} |Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) - \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}})| < (M-(m-1))\epsilon, \tag{A.2}$$

where $\widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) := \mathbb{E}\left[\tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m}) \mid \mathbf{S}_{t_{m-1}}\right]$, as defined in Eq. (7).

$$\begin{aligned} |Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) - \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}})| &= |\mathbb{E}\left[V_{t_m}(\mathbf{S}_{t_m})|\mathbf{S}_{t_{m-1}}\right] - \mathbb{E}\left[\tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m}) \mid \mathbf{S}_{t_{m-1}}\right]| \\ &= |\mathbb{E}\left[V_{t_m}(\mathbf{S}_{t_m}) - \tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m}) \mid \mathbf{S}_{t_{m-1}}\right]| \\ &\leq \mathbb{E}\left[|V_{t_m}(\mathbf{S}_{t_m}) - \tilde{G}^{\beta_{t_m}}(\mathbf{S}_{t_m})| \mid \mathbf{S}_{t_{m-1}}\right] \\ &< \mathbb{E}\left[\epsilon \mid \mathbf{S}_{t_{m-1}}\right] = (M-(m-1))\epsilon, \end{aligned}$$

where the first equation is restating the definition, the second is the outcome of no-arbitrage and the linearity of expectation operator, the third line uses Jensen's inequality, while the final line is the outcome of substitution of the result in Eq. (A.1).

The following cases then arise:

- Case 1:

$$h(\mathbf{S}_{t_{m-1}}) \le Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) \le \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}),$$

or

$$h(\mathbf{S}_{t_{m-1}}) \le \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) \le Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}})$$

For both these sub-cases, it follows that,

$$\begin{aligned}|V_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) - \tilde{V}_{t_{m-1}}(\mathbf{S}_{t_{m-1}})| &= |\max\left(h(\mathbf{S}_{t_{m-1}}), Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}})\right) - \max(h(\mathbf{S}_{t_{m-1}}), \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}))| \\ &= |h(\mathbf{S}_{t_{m-1}}) - h(\mathbf{S}_{t_{m-1}})| = 0 < (M-(m-1))\epsilon.\end{aligned}$$

- Case 2:

$$h(\mathbf{S}_{t_{m-1}}) < Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) < \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}),$$

or

$$h(\mathbf{S}_{t_{m-1}}) < \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) < Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}})$$

For both these sub-cases, we find

$$\begin{aligned}|V_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) - \tilde{V}_{t_{m-1}}(\mathbf{S}_{t_{m-1}})| &= |\max\left(h(\mathbf{S}_{t_{m-1}}), Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}})\right) - \max(h(\mathbf{S}_{t_{m-1}}), \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}))| \\ &= |Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) - \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}})| < (M-(m-1))\epsilon,\end{aligned}$$

where the inequality in the second line is the outcome of Eq. (A.2).

- Case 3:

$$Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) < h(\mathbf{S}_{t_{m-1}}) < \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}),$$

For this sub-case, we have,

$$|h(\mathbf{S}_{t_{m-1}}) - \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}})| < |Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) - \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}})|$$

Therefore,

$$\begin{aligned}|V_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) - \tilde{V}_{t_{m-1}}(\mathbf{S}_{t_{m-1}})| &= |\max\left(h(\mathbf{S}_{t_{m-1}}), Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}})\right) - \max(h(\mathbf{S}_{t_{m-1}}), \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}))| \\ &= |h(\mathbf{S}_{t_{m-1}}) - \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}})| < |Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) - \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}})| < (M-(m-1))\epsilon.\end{aligned}$$

- Case 4:

$$\widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) < h(\mathbf{S}_{t_{m-1}}) < Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}})$$

For this sub-case,

$$|Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) - h(\mathbf{S}_{t_{m-1}})| < |Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) - \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}})|$$

Therefore,

$$\begin{aligned}|V_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) - \tilde{V}_{t_{m-1}}(\mathbf{S}_{t_{m-1}})| &= |\max\left(h(\mathbf{S}_{t_{m-1}}), Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}})\right) - \max(h(\mathbf{S}_{t_{m-1}}), \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}}))| \\ &= |Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) - h(\mathbf{S}_{t_{m-1}})| < |Q_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) - \widehat{Q}_{t_{m-1}}(\mathbf{S}_{t_{m-1}})| < (M-(m-1))\epsilon.\end{aligned}$$

With this we have shown that for all possible cases that might arise, $|V_{t_{m-1}}(\mathbf{S}_{t_{m-1}}) - \tilde{V}_{t_{m-1}}(\mathbf{S}_{t_{m-1}})| < (M-(m-1))\epsilon$. This ends the proof.

References

[1] D. Ackerer, N. Tagasovska, T. Vatter, Deep smoothing of the implied volatility surface, arXiv preprint arXiv:1906.05065(2019).
[2] L. Andersen, M. Broadie, Primal-dual simulation algorithm for pricing multidimensional American options, Manage. Sci. 50 (9) (2004) 1222–1234.
[3] A. Balata, J. Palczewski, Regress-Later Monte Carlo for optimal control of Markov processes, arXiv preprint arXiv:1712.09705(2017).
[4] C. Bayer, B. Horvath, A. Muguruza, B. Stemper, M. Tomas, On deep calibration of (rough) stochastic volatility models, arXiv preprint arXiv:1908.08806(2019).
[5] S. Becker, P. Cheridito, A. Jentzen, Deep optimal stopping, J. Mach. Learn. Res. 20 (74) (2019) 1–25.
[6] D.T. Breeden, R.H. Litzenberger, Prices of state-contingent claims implicit in option prices, J. Bus. (1978) 621–651.
[7] M. Broadie, M. Cao, Improved lower and upper bound algorithms for pricing American options by simulation, Quant. Finance 8 (8) (2008) 845–861.
[8] M. Broadie, P. Glasserman, S. Kou, A continuity correction for discrete barrier options, Math. Finance 7 (4) (1997) 325–349.
[9] M. Broadie, P. Glasserman, S.-G. Kou, Connecting discrete and continuous path-dependent options, Finance Stochastics 3 (1) (1999) 55–82.
[10] P. Carr, L. Wu, Static hedging of standard options, J. Financ. Econom. 12 (1) (2013) 3–46.
[11] A. Chateauneuf, M. Mostoufi, D. Vyncke, Comonotonic monte carlo simulation and its applications in option pricing and quantification of risk, J. Derivatives 24 (1) (2016) 18–28.
[12] E. Derman, D. Ergener, I. Kani, Static options replication, J. Derivatives 2 (4) (1995) 78–95.
[13] Q. Feng, S. Jain, P. Karlsson, D. Kandhai, C.W. Oosterlee, Efficient computation of exposure profiles on real-world and risk-neutral scenarios for Bermudan swaptions, J. Comput. Finance 20 (1) (2016) 139–172.
[14] P. Glasserman, B. Yu, Simulation for American options: regression now or regression later? in: Monte Carlo and Quasi-Monte Carlo Methods 2002, Springer, 2004, pp. 213–226.

[15] M.B. Haugh, L. Kogan, Pricing American options: a duality approach, Oper. Res. 52 (2) (2004) 258–270.
[16] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366.
[17] S. Jain, P. Karlsson, D. Kandhai, KVA, Mind your P's and Q's!, Wilmott 2019 (102) (2019) 60–73.
[18] S. Jain, Á. Leitao, C.W. Oosterlee, Rolling adjoints: fast Greeks along Monte Carlo scenarios for early-exercise options, J. Comput. Sci. 33 (2019) 95–112.
[19] S. Jain, C.W. Oosterlee, The stochastic grid bundling method: efficient pricing of Bermudan options and their Greeks, Appl. Math. Comput. 269 (2015) 412–431.
[20] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv preprint arXiv:1412.6980(2014).
[21] M. Kohler, A. Krzyżak, N. Todorovic, Pricing of high-dimensional American options by neural networks, Math. Finance 20 (3) (2010) 383–410.
[22] B. Lapeyre, J. Lelong, Neural network regression for Bermudan option pricing, arXiv preprint arXiv:1907.06474 (2019).
[23] Á. Leitao, C.W. Oosterlee, GPU acceleration of the stochastic grid bundling method for early-exercise options, Int. J. Comput. Math. 92 (12) (2015) 2433–2454.
[24] M. Leshno, V.Y. Lin, A. Pinkus, S. Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, Neural Netw. 6 (6) (1993) 861–867.
[25] S. Liu, A. Borovykh, L.A. Grzelak, C.W. Oosterlee, A neural network-based framework for financial model calibration, arXiv preprint arXiv:1904.10523(2019a).
[26] S. Liu, C.W. Oosterlee, S.M. Bohte, Pricing options and computing implied volatilities using neural networks, Risks 7 (1) (2019) 16.
[27] P. Pellizzari, Static hedging of multivariate derivatives by simulation, Eur. J. Oper. Res. 166 (2) (2005) 507–519.
[28] A. Pelsser, Pricing and hedging guaranteed annuity options via static option replication, Insurance Math. Econ. 33 (2) (2003) 283–296.
[29] A. Pelsser, J. Schweizer, The difference between LSMC and replicating portfolio in insurance liability modeling, Eur. Actuarial J. 6 (2) (2016) 441–494.
[30] C. Reisinger, G. Wittum, Efficient hierarchical approximation of high-dimensional option pricing problems, SIAM J. Sci. Comput. 29 (1) (2007) 440–458.
[31] L.C. Rogers, Monte Carlo valuation of American options, Math. Finance 12 (3) (2002) 271–286.