# Neural variance reduction for stochastic differential equations

P.D. Hinds \* M.V. Tretyakov<sup>†</sup>
May 23, 2023

#### Abstract

Variance reduction techniques are of crucial importance for the efficiency of Monte Carlo simulations in finance applications. We propose the use of neural SDEs, with control variates parameterized by neural networks, in order to learn approximately optimal control variates and hence reduce variance as trajectories of the SDEs are being simulated. We consider SDEs driven by Brownian motion and, more generally, by Lévy processes including those with infinite activity. For the latter case, we prove optimality conditions for the variance reduction. Several numerical examples from option pricing are presented.

**Keywords:** stochastic differential equations, control variates, deep learning, option pricing.

## 1 Introduction

Stochastic differential equations (SDEs) driven by Lévy processes arise as a modelling tool in several fields including finance, insurance, physics, and chemistry [9, 32, 28]. In finance, SDEs are employed for the pricing and hedging of financial derivatives (see e.g. [8, 9, 21, 46] and references therein). In particular, SDEs are used to find current prices and hedges of options which are not traded (or not regularly traded on exchanges) and to evaluate future option prices for risk management purposes.

In many cases including option pricing, one wants to compute the value

$$u(t,x) = \mathbb{E}\Big[f\big(X_{t,x}(T)\big)\Big],\tag{1.1}$$

where  $X_{t,x}(s)$ ,  $s \ge t$ , is the solution to a system of SDEs with the initial condition  $X_{t,x}(t) = x$ , T > 0 is a terminal (maturity) time, and f is a real-valued function (payoff in the case of option pricing). Weak-sense numerical integration of SDEs together with the Monte Carlo (MC) technique can be used to find u(t,x) [21, 37, 40]. When employing these probabilistic methods, there are two sources of the error present in the numerical approximations: the numerical integration error and the MC error. Methods of variance reduction play an important role in minimizing the MC error, which is the focus of this paper.

As it is well known (see e.g. [14, 15]), the function u(t, x) solves the related, via the Feynman-Kac formula, partial differential equation (PDE) problem in the diffusion case or partial integro-differential equation (PIDE) problem in the case of general Lévy-driven SDEs. When SDEs are driven purely by Brownian motion, there is an optimality condition (see [39, 34, 37] or Theorem 1 here) relating the optimal variance reduction with the solution of the PDE problem, u(t, x), and

<sup>\*</sup>School of Mathematical Sciences, University of Nottingham, UK; pmxph7@nottingham.ac.uk

 $<sup>^\</sup>dagger S chool \ of \ Mathematical \ S ciences, \ University \ of \ Nottingham, \ UK; \ Michael. Tretyakov@nottingham.ac.uk$ 

its spatial derivatives,  $\partial u(t,x)/\partial x^i$ . Of course, if the PDE solution u(t,x) is known, there would be no need to resort to probabilistic methods in the first place. The importance of this theoretical result lies in the fact that it demonstrates that perfect variance reduction is possible and, if an approximation of the solution u(t,x) can be learned, it can induce efficient variance reduction. In addition, this variance reduction method does not bias the MC approximation (see e.g. [39, 36, 37]), indicating that the approximation of the required u(t,x) and its derivatives need not have a high accuracy in order to be useful.

To make variance reduction practical, a suitable method of constructing u(t,x) and its spatial derivatives should be inexpensive. Hence, a trade-off between accuracy and computational costs in finding u(t,x) and  $\partial u(t,x)/\partial x^i$  is needed. To address this problem, it was suggested in [36] (see also [37] and [5]) to exploit conditional probabilistic representations of u(t,x) in conjunction with linear regression, which allows one to evaluate u(t,x) and  $\partial u(t,x)/\partial x^i$  using the single auxiliary set of approximate trajectories starting from an initial position. This leads to obtaining sufficiently inexpensive, but useful for variance reduction, estimates of u(t,x) and  $\partial u(t,x)/\partial x^i$ . The drawback of this approach is the reliance on linear regression. To make it computationally feasible, a careful selection of basis functions for linear regression is needed. This selection is heavily problem dependent and limits applicability of this variance reduction approach.

In this direction, Vidales, Šiška and Szpruch [48] propose several algorithms with the aim of constructing a control variate via deep learning. An analogue of the linear regression algorithm from [36] is proposed, this time using a deep neural network to approximate the PDE solution. Again, the rough solution will be biased, but the bias is removed from the actual approximation by using the MC technique. In addition to this, a method is put forward to learn the control variates directly, without first constructing a solution to the PDE. In order to do this, the empirical variance is used as a loss function and minimized using stochastic gradient descent. In all cases, the training of neural networks is carried out offline over a parametric family of SDEs (PDEs). In the context of financial option pricing, this means that training one network is sufficient for a single model-payoff combination.

The key objective of our paper is to show that deep learning (i.e. nonlinear regression) can successfully replace linear regression in variance reduction for SDEs in a universal manner, i.e. without need to do tuning for each new system of SDEs which is important for applications, in particular for those arising in financial engineering. The use of neural networks instead of linear regression as in [36] eliminates the need of finding a specific set of basis functions, thus making the variance reduction truly practical.

Furthermore, in contrast to [48] and many other deep learning works related to computational finance and more generally to efficient SDEs and PDEs simulations (see e.g. [43, 24, 18, 44] and references therein), here we do not rely on an (often costly) offline training of neural networks but rather we do network training online together with actual simulation of the required expectation u(t,x) with reduced variance. In other words, we propose a black-box fashion practical variance reduction tool which does not require any lengthy pre-training and tuning, analogously how the variance reduction method of [36] intended to work but without the limitations due to linear regression. Note that this approach differs to [48], where training is done offline across a parametric family of PDEs.

For general Lévy-driven SDEs, variance reduction has been considerably less studied than in the Brownian case. To the best of our knowledge, a result demonstrating that there exists an optimal choice of control variates, analogous to the results (in the diffusion case) in [39, 34] (see also [37]), does not exist. We note that in the context of financial applications, there have been several effective approaches towards variance reduction under Lévy-driven models (e.g. [42, 13, 41] and references therein).

We also remark that there are other techniques than variance reduction aimed at reducing computational complexity of MC simulations: multi-level MC method [19] and quasi-Monte Carlo method [12]. They have their own areas of applicability and can potentially be combined

with deep learning and neural SDEs considered in this paper. Here, we restrict ourselves with considering the use of deep learning for improving the plain-vanilla MC technique via variance reduction.

This paper is most closely related to [34, 36, 48]. We show that for Lévy-driven SDEs, there exist conditions ensuring the optimal variance reduction, which is an analogous result to the one in the Brownian SDEs case [34]. We also provide details of a novel algorithm which produces low variance simulations of a given SDEs system. Given a system of SDEs, we introduce control variates and parameterize them with neural networks. This can be interpreted in two ways. First, we can treat the problem of fitting the neural network as a nonlinear regression problem, for which we need to solve an optimization problem. As such, the proposed method can be seen as a more general framework than that proposed in [36]. Second, the use of a neural network as a coefficient in SDEs gives rise to neural SDEs (see e.g. [47, 26]), which is essentially a generative model (see [29]). Regardless of the perspective, the parameters of the neural SDEs can be learned from numerical simulations of the system, in such a way that the empirical variance is minimized. SDEs' trajectories with high accuracy can then be simulated using the approximate control variates. We leverage the fact that that no prescribed accuracy is required for the approximated control variates, since they do not introduce bias into the MC approximation. This algorithm is not limited to the case that SDEs are driven only by Brownian motion, as in [36, 48], and can be applied in the general setting of Lévy noise, even when we have infinite activity of jumps.

The paper is structured as follows. In Section 2, we consider the case where SDEs are driven purely by Brownian motion and the corresponding PDE problem. We recall the optimality conditions for variance reduction and present a numerical algorithm to find an approximation of the optimal control variate using deep learning. In Section 3, we consider the more general case where SDEs are driven by Lévy noise. We derive optimality conditions for variance reduction and present a corresponding numerical algorithm. Section 4 contains several numerical examples of computing option prices with SDE models of underliers in both the Brownian motion and Lévy noise cases. In the Lévy noise case, especially for SDEs driven by infinite activity Lévy processes, efficient approximation of the SDEs requires the use of jump-adapted numerical schemes (see e.g. [11] and references therein) which pose a unique challenge (addressed in Section 4) of how to simulate independent trajectories in parallel since each trajectory may have a different number of time steps. By our numerical tests, we show that the proposed neural variance reduction technique can considerably reduce variance and hence increase computational efficiency (up to 40 times) of option pricing.

# 2 SDEs driven by Brownian motion

In the case of SDEs driven only by Brownian motion, the two variance reduction methods of importance sampling and control variates are well-known and have been studied in [39, 34, 37] and references therein.

Let  $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t_0 \leq t \leq T}, P)$  be a filtered probability space on which a d-dimensional Brownian motion W(t) is defined. For  $s \in [t_0, T]$  and  $x \in \mathbb{R}^d$ , consider the stochastic processes  $X_{s,x}(t), Y_{s,x}(t), Z_{s,x}(t), t \geq s$ , defined as the solution to the system:

$$dX(t) = b(t, X)dt + \sigma(t, X)dW(t), X(s) = x, (2.1)$$

$$dY(t) = c(t, X)Y(t)dt, Y(s) = 1, (2.2)$$

$$dZ(t) = g(t, X)Y(t)dt, Z(s) = 0, (2.3)$$

where b(t, x) is a d-dimensional vector,  $\sigma(t, x)$  is a  $d \times d$  matrix, and c(t, x) and g(t, x) are scalar functions, all with appropriate regularity properties [15, 16]. We are interested in calculating the quantity

$$u(s,x) = \mathbb{E}[f(X_{s,x}(T))Y_{s,x}(T) + Z_{s,x}(T)]. \tag{2.4}$$

Let us denote the random variable of interest as

$$\Gamma := \Gamma_{s,x} := f(X_{s,x}(T))Y_{s,x}(T) + Z_{s,x}(T), \tag{2.5}$$

and, more generally, let us define

$$\Gamma(t) := \Gamma_{s,x}(t) := u(t, X_{s,x}(t)) Y_{s,x}(t) + Z_{s,x}(t), \tag{2.6}$$

noting that  $\Gamma_{s,x}(T) = \Gamma_{s,x}$ . The function  $u: [t_0,T] \times \mathbb{R}^d \to \mathbb{R}$  is known [14, 15, 16] to satisfy the related Cauchy problem for the parabolic PDE:

$$\frac{\partial u}{\partial t} + Lu + c(t, x)u + g(t, x) = 0, \qquad (t, x) \in [t_0, T) \times \mathbb{R}^d$$
 (2.7)

$$u(T,x) = f(x), x \in \mathbb{R}^d, (2.8)$$

where L is a differential operator of the form

$$Lu(t,x) := \frac{1}{2} \operatorname{tr}[a(t,x)\nabla^2 u(t,x)] + \langle b(t,x), \nabla u(t,x) \rangle. \tag{2.9}$$

Here  $a(t,x) = \sigma(t,x)\sigma^{\top}(t,x)$  is a symmetric positive semi-definite  $d \times d$ -matrix. Instead of the system (2.1)-(2.3), consider now the system

$$dX = b(t, X)dt - \sigma(t, X)\mu(t, X)dt + \sigma(t, X)dW(t), \qquad X(s) = x, \qquad (2.10)$$

$$dY = c(t, X)Ydt + \mu^{\top}(t, X)YdW(t), Y(s) = 1, (2.11)$$

$$dZ = g(t, X)Ydt + G^{\mathsf{T}}(t, X)YdW(t), \qquad Z(s) = 0, \qquad (2.12)$$

where  $\mu$  and G are d-dimensional vector functions with good analytical properties. Note that the system (2.1)-(2.3) is a special case of this system with  $\mu = G = 0$ . It is straightforward to show that while  $\mathbb{E}\Gamma(T)$  does not depend on  $\mu$  or G, the variance of  $\Gamma(T)$  does indeed depend on them (see e.g. [37]). The case when  $\mu = 0$  corresponds to the method of control variates (first considered in [39, 38]). The case when G = 0 corresponds to the method of importance sampling (first considered in [20, 49]). The combining method, i.e. using both importance sampling and control variates simultaneously, was introduced in [34] (see also [37]). Those methods can be made optimal, in the sense that  $\Gamma(T)$  becomes deterministic through an optimal choice of  $\mu$  or/and G. This optimal choice (see the theorem below), however, depends on (and consequently requires knowledge of) the full solution of the related PDE problem and its spatial derivatives.

**Theorem 1** (Milstein & Schoenmakers [34]). If  $\mu$  and G are such that

$$u(t,x)\mu(t,x) + G(t,x) = -\sigma^{\top}(t,x)\nabla u(t,x)$$
(2.13)

for all  $(t,x) \in [s,T] \times \mathbb{R}^d$ , then  $\operatorname{Var}\Gamma_{s,x}(T) = 0$ .

This theorem demonstrates a general possibility of perfect variance reduction and serves as a guidance towards choosing (some suboptimal but practical)  $\mu$  or/and G.

#### 2.1 Numerical algorithm

We consider the neural SDEs

$$dX = b(t, X)dt + \sigma(t, X)dW(t), X(s) = x, (2.14)$$

$$dY = c(t, X)Ydt, Y(s) = 1, (2.15)$$

$$dZ = q(t, X)Ydt + G_{\theta}^{\top}(t, X)YdW(t), \qquad Z(s) = 0, \qquad (2.16)$$

where  $G_{\theta}: [t_0, T] \times \mathbb{R}^d \to \mathbb{R}^d$  is a neural network parameterization of G (see Appendix A).

Provided that the system (2.1)-(2.3) has a unique strong solution, it is sufficient that  $G_{\theta}$  be Lipschitz in x for the existence and uniqueness of a solution to (2.14)-(2.16). In the case of feed-forward architectures, which we will make use of, this amounts to  $G_{\theta}$  having a Lipschitz activation function (see Appendix A).

Comparably to before, define

$$\Gamma_{\theta} := f(X_{s,x}(T))Y_{s,x}(T) + Z_{s,x}(T).$$
 (2.17)

The subscript  $\theta$  in  $\Gamma_{\theta}$  denotes the implicit dependence of  $\Gamma_{\theta}$  on  $G_{\theta}$ .

For any choice of  $\theta$ , the expectation,  $\mathbb{E}\Gamma_{\theta}$ , remains unchanged. Moreover, since Theorem 1 implies that there exists an optimal choice of G such that we have zero variance of  $\Gamma$ , our objective is to find parameters  $\theta^*$  such that  $G_{\theta^*}$  approximately satisfies (2.13) and hence  $\operatorname{Var}\Gamma_{\theta^*} \approx 0$ . We will denote the true optimal value of G by  $G^*$ . In other words,  $G^*$  satisfies

$$G^*(t,x) = -\sigma^{\top}(t,x)\nabla u(t,x). \tag{2.18}$$

Let us briefly justify the choice of feed-forward neural networks as approximators. There exists many theoretical results on the expressivity of neural networks [10, 27, 3]. In particular, the universal approximation theorem given by Hornik et al. [27] states that for any finite measure on  $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ , the class of feed-forward neural networks, mapping from  $\mathbb{R}^d$  to  $\mathbb{R}$  with continuous and non-constant activation is dense in  $L^p$ . The function we would like to approximate,  $G^*$ , maps into  $\mathbb{R}^d$  but the above approximation theorem can be applied componentwise. Furthermore, there is a growing body of work on the numerical solution to PDEs using neural networks [43, 24, 18], in particular focused on the effectiveness of neural networks in solving high-dimensional problems, see for example [6, 22] and the references mentioned therein. It is precisely in these high-dimensional situations where one may resort to Monte Carlo methods, and the above cited research in deep learning provides a justification for making use of neural SDEs within this context.

The problem of finding optimal parameters  $\theta^*$  amounts to solving the optimization problem

$$\theta^* \in \arg\min_{\theta} \text{Var}\Gamma_{\theta}. \tag{2.19}$$

Of course, in any non-trivial situation  $\text{Var}\Gamma_{\theta}$  can not be evaluated analytically, nor can the random variable  $\Gamma_{\theta}$  be simulated exactly. Instead we must use the empirical (sample) variance of independent realizations of an approximate random variable,  $\bar{\Gamma}_{\theta}$ , which is close to  $\Gamma_{\theta}$  in the weak sense. Fixing a large  $M_r \in \mathbb{N}$ , the problem becomes

$$\theta^* \in \arg\min_{\theta} \operatorname{Var}_M \bar{\Gamma}_{\theta},$$
 (2.20)

where  $\operatorname{Var}_{M_r}(\cdot)$  denotes the empirical variance over  $M_r$  realizations. The random variables  $(\bar{\Gamma}_{\theta,m})_{m=1}^{M_r}$  are obtained by numerical integration of the neural SDE system (2.14)-(2.16) together with (2.17). This optimization problem can be solved using a stochastic optimisation algorithm, noting that the loss function

$$\mathcal{L}(\theta) := \operatorname{Var}_{M_r} \bar{\Gamma}_{\theta} \tag{2.21}$$

is differentiable.

Once the parameters  $\theta^*$  have been found, realizations of  $\bar{\Gamma}_{\theta^*}$  can be simulated using a (potentially different) numerical integration scheme. The new MC estimator is then given by

$$\bar{u}(s,x) = M^{-1} \sum_{m=1}^{M} \bar{\Gamma}_{\theta^*,m},$$
 (2.22)

where  $(\bar{\Gamma}_{\theta^*,m})_{m=1}^M$  are independent copies of  $\Gamma_{\theta^*}$ .

This motivates a two-pass algorithm, where the first pass finds optimal parameters  $\theta^*$  and the second pass uses these parameters to simulate low-variance random variables for the MC estimator (cf. the two-run algorithm in [36]). This is described in Algorithm 1. In the first pass, we simulate (using a numerical integration scheme) trajectories of the solution to the system (2.14)-(2.16) with  $G_{\theta} = 0$  and store them in memory along with the random variables used in the scheme. These trajectories can then be used to simulate the  $\theta$ -dependent term,  $\int_{s}^{T} G_{\theta}(t, X)Y(t)dW(t)$ , at each iteration of the stochastic optimisation algorithm. This eliminates the need to simulate the entire system for each iteration of the stochastic optimisation algorithm.

In the second pass, we simply simulate solutions to the system (2.14)-(2.16) using  $G_{\theta^*}$ . Note that the numerical scheme used in the second pass does not need to have anything in common with the numerical scheme of the first pass. In particular, we find it to be effective to use a coarse grid (larger h) in the first pass, followed by a fine grid (smaller h) in the second pass (cf. a similar observation in [36] in the case of the linear regression-based algorithm).

Letting  $V(t) = (X(t)^{\top}, Y(t), Z(t))^{\top}$  and  $v = (x^{\top}, 1, 0)^{\top}$ , we consider numerical methods with a uniform step-size h > 0 of the form:

$$\bar{V}_{k+1} = \bar{V}_k + A(t_k, \bar{V}_k, h, \xi_k),$$
 (2.23)

$$\bar{V}_0 = V(s) = v,$$
 (2.24)

for some Borel measurable, vector-valued function A and random variables  $(\xi_n)_{n\geq 0}$ , where  $\xi_0$  is independent of  $\bar{V}_0$  and  $\xi_k$  is independent of  $(\xi_n)_{0\leq n\leq k}$  and  $(\bar{V}_n)_{0\leq n\leq k}$ . We denote the scheme by (A,h).

```
Algorithm 1: Neural control variate method for Brownian-driven SDEs
  Result: MC approximation of the solution to the PDE (2.7)-(2.8) at the point (s, x_0).
  Initialise: Number of trials for first-pass M_r, number for trials for second-pass M,
                  numerical scheme for first-pass (A_r, h_r), numerical scheme for second-pass
  for m \leftarrow 1 to M_r do
      Initialise: \bar{X}_0, \bar{Y}_0, \bar{Z}_0 \leftarrow x_0, 1, 0
      Compute: (t_k, \bar{X}_k, \bar{Y}_k, \bar{Z}_k)_{0 \le k \le N}^m and (\bar{\Gamma})^m according to (2.14)-(2.16) with G_\theta = 0
                       and the scheme (A_r, h_r)
      Store: (t_k, \bar{X}_k, \bar{Y}_k, \bar{Z}_k)_{0 \le k \le N}^m and (\bar{\Gamma})^m and random variables (\xi_k)_{0 \le k \le N}
  end
  Compute: \theta^* = \arg \min \operatorname{Var}_{M_r} \bar{\Gamma}_{\theta} by the stochastic optimisation algorithm using the
                  stored trajectories and random variables to compute \bar{\Gamma}_{\theta} with the scheme
                  (A_r,h_r).
  for m \leftarrow 1 to M do
      Initialise: \bar{X}_0, \, \bar{Y}_0, \, \bar{Z}_0 \leftarrow x_0, \, 1, \, 0
      Compute: \Gamma_{\theta^*,m} using (2.14)-(2.16) with G_{\theta^*} and the numerical scheme (A,h)
      Store: Updated sample statistics of \bar{\Gamma}_{\theta^*}
  end
  Return: \bar{u}(s, x_0) = M^{-1} \sum_{m=1}^{M} \bar{\Gamma}_{\theta^*, m}
```

**Remark 1.** For the numerical methods, Algorithm 1 and later Algorithm 2, we only consider the case of control variates ( $\mu=0$ ). The principal reason for this is computational efficiency. What makes Algorithm 1 effective is the fact that a large batch of trajectories of the SDEs can be simulated once, in parallel, before training. However, in the importance sampling case, trajectories of X,Y and Z depend on the neural network parameters,  $\theta$ . Thus, each time  $\theta$  is updated new trajectories have to be sampled, which makes the approach computationally intractable.

In practice the learned  $G_{\theta^*}$  is not equal to the optimal  $G^*$  satisfying (2.18) due to errors of deep learning (finite size of the network, limited training set and accuracy of the stochastic optimisation algorithm) and due to the numerical integration error. Recall [37, p. 151]:

$$Var\Gamma_{\theta^*} = E \int_s^T Y_{s,x_0}^2(t) \sum_{j=1}^d \left( \sum_{i=1}^d \sigma^{ij} \frac{\partial u}{\partial x^i} + G_{\theta^*}^j \right)^2 dt.$$

Then, using (2.18), we get that the standard deviation of  $\Gamma_{\theta^*}$  gives the error of  $\Gamma_{\theta^*}$  in a weighted norm:

$$\sqrt{Var\Gamma_{\theta^*}} = \left(E \int_s^T Y_{s,x_0}^2(t) |G_{\theta^*} - G^*|^2 dt\right)^{1/2}.$$

Consequently, ignoring the error of numerical integration, we can view

$$\operatorname{Err}_{G_{\theta^*}} = \frac{\sqrt{\operatorname{Var}\Gamma_{\theta^*}}}{\mathbb{E}\Gamma_{\theta^*}}$$
 (2.25)

as the appropriated relative error of the trained  $G_{\theta^*}$ .

## 3 SDEs driven by Lévy processes

We now turn our attention to the case of SDEs driven by a more general Lévy noise, i.e., SDEs driven by both Wiener and Poisson processes. In this case, we have the corresponding Cauchy problem for the PIDE:

$$\frac{\partial u}{\partial t} + Lu + c(t, x)u + g(t, x) = 0, \qquad (t, x) \in [t_0, T) \times \mathbb{R}^d, \qquad (3.1)$$

$$u(T,x) = f(x), x \in \mathbb{R}^d, (3.2)$$

where L is a partial integro-differential operator of the form

$$Lu(t,x) := \frac{1}{2} \text{tr}[a(t,x)\nabla^2 u(t,x)] + \langle b(t,x), \nabla u(t,x) \rangle$$

$$+ \int_{\mathbb{R}^q} \left[ u(t,x + F(t,x)z) - u(t,x) - \langle F(t,x)z, \nabla u(t,x) \rangle \mathbb{1}_{|z| \le 1} \right] \nu(\mathrm{d}z).$$
(3.3)

Here a(t,x) is a symmetric positive semidefinite  $d \times d$ -matrix, b(t,x) is a d-dimensional vector, c(t,x) and g(t,x) are scalar functions, F(t,x) is a  $d \times q$ -matrix, and  $\nu$  is a Lévy measure such that  $\int_{\mathbb{R}^q} (|z|^2 \wedge 1) \nu(\mathrm{d}z) < \infty$ . We allow for the possibility that  $\nu$  is of infinite intensity, i.e. we may have  $\nu(B(0,r)) = \infty$  for some r > 0, where as usual for  $x \in \mathbb{R}^d$  and r > 0 we write B(x,r) for the open ball of radius r centred at x. Conditions for the existence and uniqueness of a solution to the problem (3.1)-(3.2) can be found in [17].

Let  $(\Omega, \mathcal{F}, \{\mathcal{F}_t\}_{t_0 \leq t \leq T}, P)$  be a filtered probability space on which a d-dimensional standard Wiener process w(t) and a Poisson random measure N on  $[0, \infty) \times \mathbb{R}^q$  with intensity  $ds \times \nu(\mathrm{d}z)$  are defined. Let  $\hat{N}$  denote the corresponding Poisson random measure with compensated small jumps. That is, for all  $B \in \mathcal{B}(\mathbb{R}^q)$ ,  $t \geq 0$ ,

$$\hat{N}([0,t] \times B) = \int_{[0,t] \times B} \left( N(\mathrm{d}s, \mathrm{d}z) - \mathbb{1}_{|z| \le 1} \nu(\mathrm{d}z) \mathrm{d}s \right). \tag{3.4}$$

We assume that the problem (3.1)-(3.2) admits a classical solution,  $u \in C^{1,2}([t_0, T] \times \mathbb{R}^d)$ . It has the probabilistic representation (see e.g. [1, 9]) given by

$$u(s,x) = \mathbb{E}[f(X_{s,x}(T))Y_{s,x}(T) + Z_{s,x}(T)], \tag{3.5}$$

where  $X_{s,x}(t)$ ,  $Y_{s,x}(t)$ ,  $Z_{s,x}(t)$ ,  $s \le t \le T$ , is the solution of the system of SDEs:

$$dX = b(t, X)dt + \sigma(t, X)dw(t) + \int_{\mathbb{R}^q} F(t, X(t-))z\hat{N}(dt, dz), \qquad X(s) = x,$$
 (3.6)

$$dY = c(t, X)Y(t-)dt, Y(s) = 1, (3.7)$$

$$dZ = g(t, X)Y(t-)dt, Z(s) = 0. (3.8)$$

Here the matrix  $\sigma(t,x)$  is a solution of the equation  $a(t,x) = \sigma(t,x)\sigma^{\top}(t,x)$ .

In order to be able to efficiently simulate approximate realisations of the process X, we follow [2] (see also [31, 11]) and consider a modified process  $X^{\varepsilon}$ , where the small jumps of X are approximated with an additional diffusion component. The resulting  $X^{\varepsilon}$  is a jump-diffusion with only finitely many jumps on any finite time interval.

Let W(t) be a q-dimensional Brownian motion independent of w and N. Then the process  $X_{s,x}^{\varepsilon}(t)$  and the corresponding  $Y_{s,x}^{\varepsilon}(t)$ ,  $Z_{s,x}^{\varepsilon}(t)$  are defined as the solution to

$$dX^{\varepsilon} = b(t, X^{\varepsilon}(t-)) - F(t, X^{\varepsilon}(t-))\gamma_{\epsilon}dt + \sigma(t, X^{\varepsilon}(t-))dw(t) + F(t, X^{\varepsilon}(t-))\beta_{\varepsilon}dW(t) + \int_{|z|>\varepsilon} F(t, X^{\varepsilon}(t-))zN(dt, dz),$$
(3.9)

$$dY^{\varepsilon} = c(t, X^{\varepsilon})Y^{\varepsilon}(t-)dt, \tag{3.10}$$

$$dZ^{\varepsilon} = g(t, X^{\varepsilon})Y^{\varepsilon}(t-)dt, \tag{3.11}$$

with the same initial conditions as in (3.6)-(3.8). The vector  $\gamma_{\varepsilon}$  is defined component-wise as

$$\gamma_{\varepsilon}^{i} = \int_{\varepsilon \le |z| \le 1} z^{i} \nu(\mathrm{d}z), \tag{3.12}$$

and  $\beta_{\varepsilon}$  is defined by

$$B_{\varepsilon}^{ij} = \int_{|z| < \varepsilon} z^{i} z^{j} \nu(\mathrm{d}z), \ \beta_{\varepsilon} \beta_{\varepsilon}^{\top} = B_{\varepsilon}.$$
 (3.13)

As before, introduce the random variable of interest as

$$\Gamma^{\varepsilon} := \Gamma^{\varepsilon}_{s,x} := f(X^{\varepsilon}_{s,x}(T))Y^{\varepsilon}_{s,x}(T) + Z^{\varepsilon}_{s,x}(T), \tag{3.14}$$

and, more generally,

$$\Gamma^{\varepsilon}(t) := \Gamma^{\varepsilon}_{s,r}(t) := u(t, X^{\varepsilon}_{s,r}(t)) Y^{\varepsilon}_{s,r}(t) + Z^{\varepsilon}_{s,r}(t), \tag{3.15}$$

noting that  $\Gamma^{\varepsilon} = \Gamma^{\varepsilon}(T)$ .

We can approximate the solution, u(t,x), to the PIDE (3.1)-(3.2) by

$$u(s,x) \approx u^{\varepsilon}(s,x) := \mathbb{E}[\Gamma^{\varepsilon}(T)].$$
 (3.16)

It is shown in [11] (see also [31]) that  $u^{\varepsilon}(t,x)$  is a good approximation for u(t,x), whose accuracy is controlled by  $\varepsilon$ . The PIDE problem for  $u^{\varepsilon}$  is given by

$$\frac{\partial u}{\partial t} + L_{\varepsilon}u^{\varepsilon} + c(t, x)u^{\varepsilon} + g(t, x) = 0, \tag{3.17}$$

$$u^{\varepsilon}(T, x) = f(x), \tag{3.18}$$

where  $L_{\varepsilon}$  is the partial integro-differential operator

$$L_{\varepsilon}v(t,x) := \frac{1}{2} \operatorname{tr} \left[ \left( a(t,x) + F(t,x) B_{\varepsilon} F^{\top}(t,x) \right) \nabla^{2} v(t,x) \right] + \left\langle b(t,x) - F(t,x) \gamma_{\varepsilon}, \nabla v(t,x) \right\rangle$$

$$+ \int_{|z| > \varepsilon} \left[ v\left( t, x + F(t,x) z \right) - v(t,x) \right] \nu(\mathrm{d}z). \tag{3.19}$$

In the same spirit as in the Brownian case considered in Section 2, we modify the system (3.9)-(3.11) to allow for variance reduction of  $\Gamma^{\varepsilon}$  without changing the expectation of  $\Gamma^{\varepsilon}$ . In this case we have three sources of noise, namely w, W, and N. For the Brownian motions, we use importance sampling and control variate analogously to the results of Section 2. In addition, we need a control variate to deal with the Poisson random measure N. To this end, we introduce the five auxiliary functions,  $\mu_w: [0,\infty) \times \mathbb{R}^d \to \mathbb{R}^d$ ,  $\mu_W: [0,\infty) \times \mathbb{R}^d \to \mathbb{R}^q$ ,  $G_w: [0,\infty) \times \mathbb{R}^d \to \mathbb{R}^d$ , and  $G_N: [0,\infty) \times \mathbb{R}^{d+q} \to \mathbb{R}$ , which can be arbitrary except for some regularity conditions.

Consider now the system

$$dX^{\varepsilon} = \left[b(t, X^{\varepsilon}(t-)) - F(t, X^{\varepsilon}(t-))\gamma_{\epsilon} - \sigma(t, X^{\varepsilon}(t-))\mu_{w}(t, X^{\varepsilon}(t-))\right] - F(t, X^{\varepsilon}(t-))\beta_{\varepsilon}\mu_{W}(t, X^{\varepsilon}(t-))\right]dt + \sigma(t, X^{\varepsilon}(t-))dw(t) + F(t, X^{\varepsilon}(t-))\beta_{\varepsilon}dW(t) + \int_{|z| \ge \varepsilon} F(t, X^{\varepsilon}(t-))zN(dt, dz),$$

$$dY^{\varepsilon} = c(t, X^{\varepsilon}(t-))Y(t-)dt + Y(t-)\mu_{w}^{\top}(t, X^{\varepsilon}(t-))dw(t), \qquad (3.21) + Y(t-)\mu_{w}^{\top}(t, X^{\varepsilon}(t-))dW(t)$$

$$dZ^{\varepsilon} = g(t, X^{\varepsilon}(t-))Y^{\varepsilon}(t-)dt + G_{w}^{\top}(t, X^{\varepsilon}(t-))Y^{\varepsilon}(t-)dw(t) + G_{w}^{\top}(t, X^{\varepsilon}(t-))Y^{\varepsilon}(t-)dw(t) + G_{w}^{\top}(t, X^{\varepsilon}(t-))Y^{\varepsilon}(t-)dw(t) + G_{w}^{\top}(t, X^{\varepsilon}(t-), z)N(dt, dz) - Y^{\varepsilon}(t-)\int_{|z| \ge \varepsilon} G_{N}(t, X^{\varepsilon}(t-), z)\nu(dz)dt.$$

Note that the previous system (3.9)-(3.11) is a special case of the above system when  $\mu_w = \mu_W = G_w = G_N = 0$ .

We now show that these auxiliary functions can be chosen in such a way that the variance of  $\Gamma_{s,x}^{\varepsilon}(T)$  becomes zero.

**Theorem 2.** If the functions  $\mu_w$ ,  $\mu_W$ ,  $G_w$ ,  $G_W$ , and  $G_N$  satisfy

$$u^{\varepsilon}(t,x)\mu_{w}(t,x) + G_{w}(t,x) = -\sigma^{\top}(t,x)\nabla u^{\varepsilon}(t,x), \tag{3.23}$$

$$u^{\varepsilon}(t,x)\mu_{W}(t,x) + G_{W}(t,x) = -\beta_{\varepsilon}^{\top} F^{\top}(t,x)\nabla u^{\varepsilon}(t,x), \tag{3.24}$$

$$G_N(t, x, z) = u^{\varepsilon}(t, x) - u^{\varepsilon}(t, x + F(t, x)z), \tag{3.25}$$

for all  $(t, x, z) \in [s, T] \times \mathbb{R}^d \times \mathbb{R}^q$ , then

$$Var\left[\Gamma_{s,x}^{\varepsilon}(T)\right] = 0. \tag{3.26}$$

Moreover,

$$u^{\varepsilon}(s,x) = \Gamma_{s,x}^{\varepsilon}(T).$$
 (3.27)

*Proof.* Applying Ito's formula to  $\Gamma_{s,x}^{\varepsilon}(t)$ , we obtain

$$\begin{split} &\Gamma_{s,x}^{\varepsilon}(T) = u^{\varepsilon}(s,x) \\ &+ \int_{s}^{T} Y^{\varepsilon} \bigg[ \frac{\partial u^{\varepsilon}}{\partial t} + \frac{1}{2} \sum_{i,j=1}^{d} a^{ij} \frac{\partial^{2} u^{\varepsilon}}{\partial x^{i} \partial x^{j}} + \langle b, \nabla u^{\varepsilon} \rangle - \langle F \gamma_{\varepsilon}, \nabla u^{\varepsilon} \rangle + c u^{\varepsilon} + g \bigg] \mathrm{d}t \\ &+ \frac{1}{2} \int_{s}^{T} Y^{\varepsilon} \sum_{i,j=1}^{d} (F B_{\varepsilon} F^{\top})^{ij} \frac{\partial^{2} u^{\varepsilon}}{\partial x^{i} \partial x^{j}} \mathrm{d}t + \int_{s}^{T} Y^{\varepsilon} \Big( \nabla^{\top} u^{\varepsilon} \sigma + u^{\varepsilon} \mu_{w}^{\top} \Big) \mathrm{d}w(t) \\ &+ \int_{s}^{T} Y^{\varepsilon} \Big( \nabla u^{\varepsilon^{\top}} F \beta_{\varepsilon} + u^{\varepsilon} \mu_{w}^{\top} \Big) \mathrm{d}W(t) + \int_{s}^{T} Y^{\varepsilon} G_{w}^{\top} \mathrm{d}w(t) + \int_{s}^{T} Y^{\varepsilon} G_{W}^{\top} \mathrm{d}W(t) \\ &+ \int_{s}^{T} \int_{|z| \geq \varepsilon} Y^{\varepsilon} \Big[ u^{\varepsilon}(t, X^{\varepsilon}(t-) + Fz) - u^{\varepsilon} \Big] N(\mathrm{d}t, \mathrm{d}z) \\ &+ \int_{s}^{T} \int_{|z| \geq \varepsilon} Y^{\varepsilon} G_{N} N(\mathrm{d}t, \mathrm{d}z) - \int_{s}^{T} \int_{|z| \geq \varepsilon} Y^{\varepsilon} G_{N} \nu(\mathrm{d}z) \mathrm{d}t. \end{split}$$

Since  $u^{\varepsilon}$  satisfies the PIDE (3.17)-(3.18), we arrive at

$$\Gamma_{s,x}^{\varepsilon}(T) = u^{\varepsilon}(s,x)$$

$$+ \int_{s}^{T} \int_{|z| \geq \varepsilon} Y^{\varepsilon} \left[ u^{\varepsilon}(t, X^{\varepsilon}(t-) + Fz) - u^{\varepsilon} \right] \left( N(\mathrm{d}t, \mathrm{d}z) - \nu(\mathrm{d}z) \mathrm{d}t \right)$$

$$+ \int_{s}^{T} \int_{|z| \geq \varepsilon} Y^{\varepsilon} G_{N} \left( N(\mathrm{d}t, \mathrm{d}z) - \nu(\mathrm{d}z) \mathrm{d}t \right) + \int_{s}^{T} Y^{\varepsilon} \left( \nabla^{\top} u^{\varepsilon} \sigma + G_{w}^{\top} + u^{\varepsilon} \mu_{w}^{\top} \right) \mathrm{d}w(t)$$

$$+ \int_{s}^{T} Y^{\varepsilon} \left( \nabla u^{\varepsilon} F \beta_{\varepsilon} + G_{w}^{\top} + u^{\varepsilon} \mu_{w}^{\top} \right) \mathrm{d}w(t).$$

$$(3.28)$$

Then, it is not difficult to see that

$$\mathbb{E}\Gamma_{s,r}^{\varepsilon}(T) = u^{\varepsilon}(s,x),\tag{3.29}$$

regardless of the choice of the auxiliary functions. Moreover, if the conditions (3.23)-(3.25) are satisfied, the integrands in (3.28) become zero and

$$\operatorname{Var}\Gamma_{s,x}^{\varepsilon}(T) = 0. \tag{3.30}$$

#### 3.1 Numerical algorithm

Theorem 2 can be used in the same way as Theorem 1 was used to motivate Algorithm 1. Theorem 2 shows that via an optimal choice of  $G_w$ ,  $G_W$ , and  $G_N$  (here we take  $\mu_w = \mu_W = 0$ ; see Remark 1) we can achieve a system of SDEs such that the random variable of interest,  $\Gamma^{\varepsilon}$ , has zero variance. Moreover, no matter the choice of  $G_w$ ,  $G_W$ , or  $G_N$  we do not introduce any additional bias into the MC approximation which is clear from (3.29). We propose an algorithm whose objective is to find a good approximation of an optimal choice of the functions  $G_w$ ,  $G_W$ , and  $G_N$  and which then uses this approximation to simulate low-variance realisations of  $\Gamma^{\varepsilon}$ .

As before, we parameterize  $G_w$ ,  $G_W$ , and  $G_N$  with feed-forward neural networks  $G_{w,\theta}$ ,  $G_{W,\theta}$ ,

and  $G_{N,\theta}$ , respectively. We obtain the neural SDEs

$$dX^{\varepsilon} = b(t, X^{\varepsilon}(t-)) - F(t, X^{\varepsilon}(t-))\gamma_{\epsilon}dt + \sigma(t, X^{\varepsilon}(t-))dw(t) + F(t, X^{\varepsilon}(t-))\beta_{\varepsilon}dW(t)$$

$$+ \int_{|z| \ge \varepsilon} F(t, X^{\varepsilon}(t-))zN(dt, ds),$$
(3.31)

$$dY^{\varepsilon} = c(t, X^{\varepsilon}(t-))Y(t-)dt, \tag{3.32}$$

$$dZ^{\varepsilon} = g(t, X^{\varepsilon}(t-))Y^{\varepsilon}(t-)dt + G_{w,\theta}^{\top}(t, X^{\varepsilon}(t-))Y^{\varepsilon}(t-)dw(t)$$

$$+ G_{W,\theta}^{\top}(t, X^{\varepsilon}(t-))Y^{\varepsilon}(t-)dW(t) + Y^{\varepsilon}(t-) \int_{|z| \ge \varepsilon} G_{N,\theta}(t, X^{\varepsilon}(t-), z)N(dt, dz)$$

$$- Y^{\varepsilon}(t-) \int_{|z| \ge \varepsilon} G_{N,\theta}(t, X^{\varepsilon}(t-), z)\nu(dz)dt.$$
(3.33)

Define

$$\Gamma_{\theta}^{\varepsilon} = f(X^{\varepsilon}(T))Y^{\varepsilon}(T) + Z^{\varepsilon}(T). \tag{3.34}$$

Letting  $V(t) = (X(t)^\top, Y(t), Z(t))^\top$  and  $v = (x^\top, 1, 0)^\top$ , we consider a numerical scheme of the form

$$\bar{V}_{k+1} = \bar{V}_k + A(t_k, \bar{V}_k, h, \xi_k), \tag{3.35}$$

$$\bar{V}_0 = V(s) = v,$$
 (3.36)

where a deterministic h > 0 is a maximum step size, A is a Borel measurable function, and  $(\xi_n)_{n\geq 0}$  are appropriately chosen random variables taking into account randomness coming from the Wiener and Poisson processes. We allow for the case of a restricted jump-adapted numerical scheme as e.g. in [11] when, in particular, the number of time steps is random (see further details in Section 4.2).

Algorithm 2 for the Lévy noise case is analogous to the Brownian motion case (Algorithm 1), except we have more sources of noise.

We note that Remark 1 is also applicable in the case of Algorithm 2

**Remark 2.** When implementing Algorithm 2, it is computationally expensive to evaluate the double integrals in (3.33). We found that, in practice, replacing the function  $G_{N,\theta}: \mathbb{R}^{d+q+1} \to \mathbb{R}$  with a linear approximation of  $G_{N,\theta}$  in z,  $g_{N,\theta}: \mathbb{R}^{d+1} \to \mathbb{R}^q$  is more efficient. In other words,

$$G_{N,\theta}(t,x,z) \approx g_{N,\theta}(t,x)^{\top} z.$$
 (3.37)

This means that the inner integrals need not be numerically approximated at each time step since the values can be computed in advance (and in many cases will be known analytically). Explicitly, the integral in (3.33) becomes

$$\int_{|z| \ge \varepsilon} G_{N,\theta}(t,x,z) \nu(dz) \approx g_{N,\theta}(t,x)^{\top} \int_{|z| \ge \varepsilon} z \nu(dz), \tag{3.38}$$

and  $\int_{|z|\geq\varepsilon} z\nu(dz)$  does not have to be computed at each time step.

Analogously to the discussion on the relative error in the diffusion case at the end of Section 2, let us consider the error of the trained  $G_{w,\theta^*}$ ,  $G_{W,\theta^*}$ ,  $G_{N,\theta^*}$  in comparison with the optimal  $G_w^*$ ,  $G_W^*$ ,  $G_N^*$  satisfying the conditions (3.23)-(3.25). It is not difficult to obtain using (3.28) and Ito's formula that

$$Var\Gamma_{\theta^*} = E \int_s^T (Y_{s,x_0}^{\varepsilon}(t))^2 \Big[ \left( \nabla^\top u^{\varepsilon} \sigma + G_{w,\theta^*}^{\top} \right)^2 + \left( \nabla u^{\varepsilon}^{\top} F \beta_{\varepsilon} + G_{W,\theta^*}^{\top} \right)^2 + \int_{|z| \ge \varepsilon} \left( u^{\varepsilon}(t, X^{\varepsilon}(t-) + Fz) - u^{\varepsilon} + G_{N,\theta^*} \right)^2 \nu(\mathrm{d}z) \Big] dt.$$

```
Algorithm 2: Neural control variate method for Lévy-driven SDEs
  Result: MC approximation of the solution to the PIDE (3.1)-(3.2), \bar{u}(s,x_0), at the
                point (s, x_0)
  Initialise: Number of trials for first-pass M_r, number for trials for second-pass M,
                    numerical scheme for first-pass (A_r, h_r), numerical scheme for second-pass
                    (A,h)
  for m \leftarrow 1 to M_r do
       Initialise: \bar{X}_0, \bar{Y}_0, \bar{Z}_0 \leftarrow x_0, 1, 0
       Compute: (t_k, \bar{X}_k, \bar{Y}_k, \bar{Z}_k)_{0 < k < N}^m and (\bar{\Gamma})^m according to (3.31)-(3.33) with G_w = G_W = G_N = 0 and the scheme (A_r, h_r)
Store: (t_k, \bar{X}_k, \bar{Y}_k, \bar{Z}_k)_{0 < k < N}^m and (\bar{\Gamma})^m and random variables (\xi_k)_{0 \le k < N}
  Compute: \theta^* = \arg \min \operatorname{Var}_{M_r} \bar{\Gamma}_{\theta} by the stochastic optimisation algorithm using the
                     stored trajectories and random variables to compute \bar{\Gamma}_{\theta} with the scheme
                     (A_r,h_r).
  for m \leftarrow 1 to M do
       Initialise: \bar{X}_0, \bar{Y}_0, \bar{Z}_0 \leftarrow x_0, 1, 0
       Compute: \bar{\Gamma}_{\theta^*,m} using (3.31)-(3.33) with G_{w,\theta^*}, G_{W,\theta^*}, G_{N,\theta^*} and the numerical
                          scheme (A, h)
       Store: Updated sample statistics of \bar{\Gamma}_{\theta^*}
  Return: \bar{u}(s, x_0) = M^{-1} \sum_{m=1}^{M} \bar{\Gamma}_{\theta^*, m}
```

Then, using (3.23)-(3.25), we get that the standard deviation of  $\Gamma_{\theta^*}$  gives the weighted error of  $\Gamma_{\theta^*}$ :

$$\sqrt{Var\Gamma_{\theta^*}} = \left( E \int_s^T Y_{s,x_0}^2(t) \left[ |G_{w,\theta^*} - G_w^*|^2 + |G_{W,\theta^*} - G_W^*|^2 + \int_{|z| \ge \varepsilon} \left( G_{N,\theta^*} - G_N^* \right)^2 \nu(\mathrm{d}z) \right] dt \right)^{1/2}.$$

Consequently (analogously to (3.39)), ignoring the error of numerical integration, we can view

$$\operatorname{Err}_{G_{\theta^*}} = \frac{\sqrt{\operatorname{Var}\Gamma_{\theta^*}}}{\mathbb{E}\Gamma_{\theta^*}}$$
 (3.39)

as the appropriated relative error of the trained  $G_{w,\theta^*}$ ,  $G_{W,\theta^*}$  and  $G_{N,\theta^*}$ .

# 4 Numerical Experiments

In this section, we provide several numerical examples from computational finance to demonstrate efficiency of Algorithms 1 and 2. We consider the cases of the SDEs driven by Lévy processes with infinite activity, finite activity, and being driven only by Brownian motion.

In the paper's spirit of the 'on-the-fly' variance reduction algorithm, we propose to fix an ANN architecture prior to any knowledge of the particular problem. Of course, better results can be achieved if a particular architecture is chosen and/or tuned for each problem. We propose to use a fully-connected feed forward ANN with ReLU activation (see Appendix A). The number of hidden layers and the size of the hidden layers are chosen via a hyperparameter search detailed in Appendix B and the values selected are shown in Table 1. We also make use of batch normalization before the first layer of the network. For the optimization, we use the

Adam algorithm [30] with a fixed learning rate of  $\eta = 10^{-3}$ . All experiments are performed on an NVidia Tesla V100 GPU.

For each experiment, we compare the neural control variate algorithm with vanilla MC. By vanilla MC, we mean MC without the use of control variates or any other variance reduction techniques. This comparison serves as a convenient benchmark since, as a rule, the additional training run of Algorithm 1/2 must be less computationally expensive than simply increasing the number of simulations of vanilla MC. The implementation is carried out in PyTorch and the code is available at https://github.com/piers-hinds/sde\_mc.

Both the MC simulations and the training of the ANNs are done on a GPU. MC is particularly efficient to implement in a GPU-supported scientific computing library like PyTorch since each MC simulation can be carried out independently, i.e. in parallel. The corresponding implementation is at https://github.com/piers-hinds/sde\_mc (see the SdeSolver class which can solve multiple independent trajectories in parallel).

Jump-adapted schemes pose a unique challenge in order to simulate trajectories in parallel, since each trajectory may have a different number of time steps when a jump-adapted numerical scheme is used. Our solution involves storing the time points corresponding to each trajectory, as well as the trajectory itself. Full details of the implementation can be found at https://github.com/piers-hinds/sde\_mc (see the JumpDiffusionSolver class).

In addition, in Section 4.4, we compare the neural control variate algorithm with Multilevel Monte Carlo method [19] as well as a crude control variate method [21].

In each experiment, the network is trained on  $M_r = 3 \cdot 10^4$  trajectories for a maximum of 20 epochs before being used to generate trajectories. We propose and use a stopping rule for the training which is explained in Appendix B; the rule allows for the termination of the training early if it becomes too slow. The training trajectories are chosen to have 5 times larger time-step sizes than the trajectories used in the final MC estimation, i.e.  $h_r = 5h$ , we refer to this value as the step factor. We use a batch size of 2000 in all experiments. We summarize the network hyperparameters in Table 1. In each experiment, the reported time taken includes training time.

Table 1: Hyperparameters and their chosen values

Hyperparameter	Value
Number of hidden layers	3
Hidden layer size	50
Step factor	5
Training data size	$3 \cdot 10^{4}$

#### 4.1 Diffusion models

In the first two experiments (Sections 4.1.1 - 4.1.2), we use the explicit Euler scheme (see e.g. [37]), which for a system

$$dX(t) = b(t, X)dt + \sigma(t, X)dW(t), \qquad X(s) = x,$$
(4.1)

is defined by

$$X_0 = X(s) = x, (4.2)$$

$$X_{k+1} = X_k + b_k h + \sigma_k \Delta_k W, (4.3)$$

where h > 0 is a fixed step-size and  $\Delta_k W$  are independent Gaussian random variables (or, in the case of many Wiener processes, vectors consisting of independent Gaussian random variables) with zero mean and variance h. The step-size used for each experiment is  $h = \frac{T-s}{1000}$ . In the experiment of Section 4.1.3 we use a custom numerical scheme from [35].

#### 4.1.1 Geometric Brownian motion

Consider the one-dimensional Black-Scholes model, in which the price u(t, x) of a contingent claim satisfies the Black-Scholes equation

$$\frac{\partial u}{\partial t} + \frac{\sigma^2 x^2}{2} \frac{\partial^2 u}{\partial x^2} + rx \frac{\partial u}{\partial x} - ru = 0 \tag{4.4}$$

with terminal condition

$$u(T,x) = f(x), (4.5)$$

where the short rate  $r \in \mathbb{R}$  and the volatility  $\sigma > 0$ . We consider the case of a call option when  $f(x) = (x - K)_+$  for various strikes K > 0. The problem (4.4)-(4.5) has the following probabilistic representation:

$$u(s,x) = \mathbb{E}\Big[f\big(X_{s,x}(T)\big)e^{-r(T-s)} + Z_{s,x}(T)\Big],\tag{4.6}$$

where

$$dX(t) = rX(t)dt + \sigma X(t)dW(t), X(s) = x, (4.7)$$

$$dZ(t) = G(t, X(t))e^{-r(t-s)}dW(t), Z(s) = 0. (4.8)$$

Numerical results for various strikes, K, are given in Table 2 for both vanilla MC (i.e. without any control variates) and for our neural variance reduction algorithm. We observe substantial (up to 18 time) speed up of option valuation when the variance reduction is used in comparison with the vanilla MC. The MC tolerance level is set at  $10^{-4}$  in this and all the other experiments, aside from those in Section 4.3 where it is set to  $10^{-3}$ . The relative error given in the table (and in all the other tables of Section 4.1) corresponds to  $\text{Err}_{Ga^*}$  as defined in (2.25).

Figure 1 shows the true optimal control variate, as given by Theorem 1, as well as the learned approximate control variate in the case that K=1. It can be seen that the learned control variate is similar to the optimal control variate but not especially accurate, indicating that high accuracy is not required for the control variate method to be effective.

Remark 3. If the optimal control variate  $G^*$  is known (or approximated well) then this would give knowledge of  $\nabla u$  through Theorem 1, which can be used for computing deltas (e.g., analogously how conditional probabilistic representations accompanied by linear regression is used for computing sensitivities in [4]). However, in this paper we refrain from doing so since we have no guarantee of accuracy of the neural network approximation of  $G^*$ , so there is no theoretical guarantee that the learned approximation of  $\nabla u$  is accurate, and importantly no reliable way to quantify the error. This is in contrast to using  $G_{\theta}$  as a control variate, since in this case the bias of the neural network approximation is nullified by integrating against the Brownian motion. Moreover, the MC error can be quantified in the normal way by computing the sample variance.

#### 4.1.2 Multi-dimensional geometric Brownian motion

Consider now the multi-dimensional Black-Scholes model for dynamics of assets' prices. The price u(t,x) of a contingent claim satisfies the PDE:

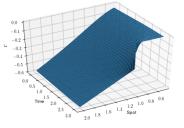
$$\frac{\partial u}{\partial t} + \frac{\sigma^2}{2} \sum_{i,j=1}^d \rho^{i,j} x^i x^j \frac{\partial^2 u}{\partial x^i \partial x^j} + r \sum_{i=1}^d x^i \frac{\partial u}{\partial x^i} - ru = 0$$
 (4.9)

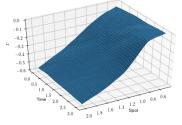
with terminal condition

$$u(T,x) = f(x), \tag{4.10}$$

Table 2: European Call option,  $f(x) = (x - K)_+$ , under GBM with r = 0.02,  $\sigma = 0.3$  and T=3: MC approximations (and a 95% confidence interval given after  $\pm$ ) with and without a control variate.

K	u(0,1)	Vanil	Control Variate MC			Relative		
11	a(0,1)	$\hat{u}(0,1)$	Time (s)	M	$\hat{u}(0,1)$	Time (s)	M	Error
0.7	0.39031	$0.39043 \pm 0.00010$	87.2	$1.04 \cdot 10^8$	$0.39032 \pm 0.00010$	5.73	$7.00 \cdot 10^4$	0.034
0.8	0.32826	$0.32833\pm0.00010$	80.3	$9.62 \cdot 10^{7}$	$0.32821\pm0.00010$	4.27	$7.00 \cdot 10^4$	0.040
0.9	0.27484	$0.27486\pm0.00010$	70.9	$8.49 \cdot 10^7$	$0.27478\pm0.00010$	4.31	$7.50 \cdot 10^4$	0.051
1	0.22943	$0.22943\pm0.00010$	63.1	$7.56 \cdot 10^7$	$0.22940\pm0.00010$	6.17	$1.15 \cdot 10^5$	0.075
1.1	0.19117	$0.19115\pm0.00011$	48.4	$5.79 \cdot 10^7$	$0.19115\pm0.00010$	6.49	$1.40 \cdot 10^5$	0.098
1.2	0.15914	$0.15914\pm0.00010$	46.2	$5.53 \cdot 10^7$	$0.15915\pm0.00010$	4.54	$9.00 \cdot 10^4$	0.096
1.3	0.13245	$0.13249\pm0.00010$	44.2	$5.29 \cdot 10^7$	$0.13247\pm0.00010$	6.48	$1.40 \cdot 10^5$	0.140





(a) Optimal control variate

(b) Learned control variate

Figure 1: The true optimal control variate and the learned control variate for a European call option with strike K=1 under the Black-Scholes model with parameters in Table 2.

where  $r \in \mathbb{R}$ ,  $\sigma > 0$  and  $(\rho)_{ij} = \rho^{i,j}$  is the correlation matrix of the Brownian motion in the following probabilistic representation:

$$u(s,x) = \mathbb{E}\Big[f(X_{s,x}(T))e^{-r(T-s)} + Z_{s,x}(T)\Big],$$
 (4.11)

and

$$dX^{i}(t) = rX^{i}(t)dt + \sigma X^{i}(t)dW_{i}(t), X^{i}(s) = x, i = 1,...,d, (4.12)$$
  
$$dZ(t) = G^{T}(t, X(t))e^{-r(t-s)}dW(t), Z(s) = 0, (4.13)$$

$$dZ(t) = G^{\top}(t, X(t))e^{-r(t-s)}dW(t), Z(s) = 0, (4.13)$$

 $W(t) = (W_1(t), \dots, W_d(t))^{\mathsf{T}}$ , and  $W_i(t)$ ,  $i = 1, \dots, d$ , are correlated Wiener processes. We consider the case of a call-on-max option, that is an option with the payoff

$$f(x) = (\max(x_1, \dots, x_d) - K)_{+}. \tag{4.14}$$

Table 3 displays the results for Algorithm 1 in the case d=3 with the parameters r=0.02,  $\sigma = 0.3, T = 3, x = 1$  and the correlation coefficients of the Wiener processes:

$$\rho^{1,2} = 0.7, \ \rho^{1,3} = 0.2, \ \rho^{2,3} = -0.3.$$

We again see the benefit of using the control variate method which gives speed up of up to 10 times.

Table 3: Call-on-max rainbow option under three-dimensional GBM with  $r=0.02, \sigma=0.3,$  T=3, x=1: Monte Carlo approximations (and a 95% confidence interval) with and without a control variate.

K	u(0,1)	Vanil	la MC		Control Variate MC			Relative
11	w(0,1)	$\hat{u}(0,1)$	Time (s)	$\overline{M}$	$\hat{u}(0,1)$	Time (s)	M	Error
0.7	-	$0.73571 \pm 0.00010$	187	$1.51 \cdot 10^{8}$	$0.73570 \pm 0.00010$	20.2	$9.15 \cdot 10^5$	0.066
0.8	-	$0.64800 \pm 0.00010$	188	$1.52 \cdot 10^8$	$0.64811 \pm 0.00010$	20.0	$9.20 \cdot 10^5$	0.076
0.9	-	$0.56563\pm0.00010$	185	$1.50 \cdot 10^8$	$0.56559\pm0.00010$	20.8	$9.65 \cdot 10^5$	0.088
1	-	$0.48988 \pm 0.00010$	177	$1.44 \cdot 10^8$	$0.48984 \pm 0.00010$	17.5	$7.60 \cdot 10^5$	0.091
1.1	-	$0.42146 \pm 0.00010$	155	$1.26 \cdot 10^8$	$0.42150 \pm 0.00010$	20.2	$9.20 \cdot 10^5$	0.120
1.2	-	$0.36085\pm0.00010$	153	$1.24 \cdot 10^8$	$0.36098\pm0.00010$	18.8	$8.30 \cdot 10^5$	0.130
1.3	-	$0.30781\pm0.00010$	134	$1.08 \cdot 10^8$	$0.30784\pm0.00010$	17.1	$7.35 \cdot 10^5$	0.140

#### 4.1.3 Heston Model

Consider the Heston stochastic volatility model [25], under which the price u(t, x, v) of a contingent claim satisfies the PDE:

$$\frac{\partial u}{\partial t} + \frac{1}{2}x^2v\frac{\partial^2 u}{\partial x^2} + \frac{1}{2}\sigma^2v\frac{\partial^2 u}{\partial v^2} + \sigma\rho vx\frac{\partial^2 u}{\partial x\partial v} + rx\frac{\partial u}{\partial x} + \kappa(\theta - v)\frac{\partial u}{\partial v} - ru = 0, \tag{4.15}$$

$$u(T, x, v) = f(x),$$
 (4.16)

where  $\kappa > 0$ ,  $\theta > 0$ ,  $\sigma > 0$ ,  $\rho \in (-1,1)$  and  $r \in \mathbb{R}$ , such that  $2\kappa\theta > \sigma^2$ . The terminal condition, f(x), corresponds to the payoff function. We consider the case of a European call option where

$$f(x) = (x - K)_{+}, (4.17)$$

for some strike price K > 0. The Heston PDE problem (4.15)-(4.16) has a probabilistic representation of the form:

$$u(s, x, v) = \mathbb{E} \Big[ f(X_{s,x,v}(T)) e^{-r(T-s)} + Z_{s,x,v}(T) \Big], \tag{4.18}$$

$$dX(t) = rX(t)dt + \sqrt{V(t)}X(t)dW_1(t), X(s) = x, (4.19)$$

$$dV(t) = \kappa \left(\theta - V(t)\right) + \sigma \sqrt{V(t)} \left(\rho dW_1(t) + \sqrt{1 - \rho^2} dW_2(t)\right), \qquad V(s) = v, \tag{4.20}$$

$$dZ(t) = G(t, X(t), V(t))e^{-r(t-s)}dW(t), Z(s) = 0, (4.21)$$

where  $W(t) = (W_1(t), W_2(t))^{\top}$  is a two-dimensional standard Wiener process.

To simulate trajectories of (4.19)-(4.20), we use the explicit Euler scheme for X and the fully implicit Euler scheme for V [35]. For a step-size h > 0,

$$X_{k+1} = X_k + rX_k h + \sqrt{V_k} X_k \Delta_k W, \tag{4.22}$$

$$V_{k+1} = V_k + \kappa(\theta - V_{k+1})h - \frac{\sigma^2}{2}h + \sigma\sqrt{V_{k+1}}\Delta_k W.$$
 (4.23)

The importance of using the fully implicit Euler scheme for V lies in the fact that the explicit Euler scheme does not necessarily preserve positivity of V, while the semi-implicit scheme (4.22)-(4.23) guarantees positivity of V under  $\kappa\theta \geq \sigma^2/2$  [35]. Note that the implicitness in (4.23) can be resolved analytically by solving the quadratic equation. The results are presented in Table 4, which demonstrates efficiency of the proposed control variate method, it is up to 42 time faster than vanilla MC.

Table 4: European Call option under the Heston model with  $v=0.15, r=0.02, \kappa=0.25, \theta=0.5, \sigma=0.3, \rho=-0.3, T=3$ : MC approximations (and a 95% confidence interval) with and without a control variate.

K	u(0,1)	Vanilla MC			Control Variate MC			Relative
11	u(0,1)	$\hat{u}(0,1)$	Time (s)	M	$\hat{u}(0,1)$	Time (s)	M	Error
0.7	0.47517	$0.47520 \pm 0.00010$	1360	$3.09 \cdot 10^8$	$0.47519 \pm 0.00008$	45.3	$2.58 \cdot 10^6$	0.14
0.8	0.42623	$0.42625\pm0.00011$	1120	$2.57 \cdot 10^{8}$	$0.42636\pm0.00010$	30.9	$1.65 \cdot 10^6$	0.16
0.9	0.38271	$0.38274\pm0.00010$	1180	$2.72 \cdot 10^{8}$	$0.38267\pm0.00011$	32.0	$1.73 \cdot 10^6$	0.19
1	0.34406	$0.34401\pm0.00010$	1070	$2.47 \cdot 10^8$	$0.34407\pm0.00009$	54.6	$3.17 \cdot 10^6$	0.24
1.1	0.30977	$0.30977\pm0.00010$	1170	$2.7 \cdot 10^{8}$	$0.30971\pm0.00010$	27.4	$1.43 \cdot 10^6$	0.20
1.2	0.27934	$0.27927\pm0.00009$	1250	$2.88 \cdot 10^{8}$	$0.27927\pm0.00011$	39.5	$2.20 \cdot 10^6$	0.28
1.3	0.25232	$0.25230\pm0.00010$	990	$2.28 \cdot 10^8$	$0.25232\pm0.00010$	46.3	$2.61 \cdot 10^6$	0.34

#### 4.2 Non-singular Lévy measure

In this section and the subsequent section (Section 4.3), we use the restricted jump-adapted numerical integration scheme from [11, Algorithm 1]. Here we give its brief description for completeness.

For the SDEs

$$dX = b(t, X) - F(t, X)\gamma_{\epsilon}dt + \sigma(t, X)dw(t) + F(t, X)\beta_{\epsilon}dW(t) + \int_{|z| \ge \epsilon} F(t, X)zN(dt, dz),$$
(4.24)

$$X(s) = x, (4.25)$$

we set  $X_0 = x$  and obtain the approximation  $X_{k+1}$  from  $X_k$  as follows. We find the next timestep  $\theta = \delta \wedge h$ , where h > 0 is the pre-defined maximum step-size and  $\delta$  is the time to the next jump sampled with the intensity  $\lambda_{\varepsilon} = \int_{|z| \geq \varepsilon} \nu(z)$ . Then, if  $\theta = h$ , we use the standard explicit Euler scheme with no jumps. If  $\theta < h$ , we use

$$X_{k+1} = X_k + (b_k - F_k)\gamma_{\varepsilon}\theta + \sigma_k \Delta_k w + F_k \beta_{\varepsilon} \Delta W_k + F_k J, \tag{4.26}$$

where  $\Delta_k w$  and  $\Delta_k W$  are the respective Brownian increments over the step  $\theta$  and J is the size of the jump, sampled (independently of other jumps,  $\delta$ ,  $\Delta_k w$  and  $\Delta_k W$ ) according to the density

$$\rho_{\varepsilon}(z) := \frac{\nu(z) \mathbb{1}_{|z| > \varepsilon}}{\lambda_{\varepsilon}}.$$
(4.27)

The approximations for Y and Z are simulated using the explicit Euler scheme, with the same random time step  $\theta$  as X. The (maximum) step-size used in all the experiments is  $h = \frac{T-s}{1000}$ .

#### 4.2.1 Merton model

Consider the one-dimensional Merton jump-diffusion model [33], under which the price u(t, x) of a contingent claim satisfies the PIDE:

$$\frac{\partial u}{\partial t}(t,x) + \frac{1}{2}\sigma^2 x^2 \frac{\partial^2 u}{\partial x^2}(t,x) + (r - \beta \lambda) x \frac{\partial u}{\partial x}(t,x) - ru(t,x) 
+ \frac{\lambda}{\sqrt{2\pi\gamma^2}} \int_{\mathbb{R}} \left[ u(t,xe^z) - u(t,x) \right] \exp\left\{ -\frac{(z-\alpha^2)}{2\sigma^2} \right\} dz = 0,$$
(4.28)

where  $r \in \mathbb{R}$ ,  $\sigma > 0$ ,  $\lambda > 0$ ,  $\alpha, \gamma \in \mathbb{R}$  and  $\beta = \exp(\alpha + \frac{1}{2}\gamma^2) - 1$ . The terminal condition is given by

$$u(T,x) = f(x). (4.29)$$

The probabilistic representation of (4.28)-(4.29) takes the form:

$$u(s,x) = \mathbb{E}\Big[f(X_{s,x}(T))e^{-r(T-s)} + Z_{s,x}(T)\Big]$$
(4.30)

with

$$dX(t) = X(t-)((r-\lambda\beta)dt + \sigma dW(t) + J(t)dN(t)), X(s) = x, (4.31)$$

$$dZ(t) = G_W(t, X(t))e^{-r(t-s)}dW(t) + G_N(t-, X(t-))e^{-r(t-s)}J(t)dN(t)$$
(4.32)

$$-\lambda e^{-r(t-s)} \int_{\mathbb{R}} G_N(t, X) z \exp\left\{-\frac{(z-\alpha^2)}{2\sigma^2}\right\} dz dt, \qquad Z(s) = 0.$$

where W(t) is a one-dimensional standard Wiener process and N(t) is a Poisson process with intensity  $\lambda$ . The jumps have a shifted log-normal distribution:

$$J_i = \exp(\eta_i) - 1, \qquad i = 1, 2, \dots$$

with  $\eta_i \sim \mathcal{N}(\alpha, \gamma^2)$ . The mean jump size is  $\mathbb{E}J_1 =: \beta = \exp(\alpha + \frac{1}{2}\gamma^2) - 1$ . The price of a European call option, u(t, x), has the terminal condition

$$f(x) = (x - K)_+,$$

for a strike price K > 0.

Table 5 shows the results of Algorithm 2 in the case of r = 0.02,  $\sigma = 0.2$ ,  $\lambda = 1$ ,  $\alpha = -0.05$ ,  $\gamma = 0.3$ . The proposed control variates method here is up to 30 times faster than vanilla MC. The relative error given in the table (and in all the other tables in the next subsections) corresponds to  $\text{Err}_{Ga^*}$  as defined in (3.39).

Table 5: European Call option under Merton model with r = 0.02,  $\sigma = 0.2$ ,  $\lambda = 1$ ,  $\alpha = -0.05$ ,  $\gamma = 0.3$  and T = 3: MC approximations (and a 95% confidence interval) with and without a control variate.

K	u(0,1)	Vanil	lla MC		Control Variate MC			Relative
	w(0, 1)	$\hat{u}(0,1)$	Time (s)	M	$\hat{u}(0,1)$	Time (s)	M	Error
0.7	0.41361	$0.41346 \pm 0.00010$	1860	$1.61 \cdot 10^8$	$0.41364 \pm 0.00010$	62.1	$1.40 \cdot 10^6$	0.15
0.8	0.35593	$0.35575 \pm 0.00009$	2010	$1.74 \cdot 10^8$	$0.35592 \pm 0.00011$	85.5	$2.05 \cdot 10^6$	0.21
0.9	0.30592	$0.30575 \pm 0.00011$	1480	$1.28 \cdot 10^8$	$0.30604 \pm 0.00010$	104	$2.49 \cdot 10^6$	0.26
1	0.26298	$0.26278 \pm 0.00010$	1460	$1.27 \cdot 10^8$	$0.26302 \pm 0.00010$	127	$3.09 \cdot 10^6$	0.34
1.1	0.22634	$0.22624 \pm 0.00011$	1200	$1.05 \cdot 10^{8}$	$0.22633 \pm 0.00010$	140	$3.47 \cdot 10^6$	0.42
1.2	0.19519	$0.19502 \pm 0.00011$	1090	$9.52 \cdot 10^7$	$0.19532 \pm 0.00010$	151	$3.79 \cdot 10^6$	0.51
1.3	0.16877	$0.16866 \pm 0.00011$	949	$8.28 \cdot 10^7$	$0.16881 \pm 0.00010$	167	$4.17 \cdot 10^6$	0.62

#### 4.3 Singular Lévy measure

In this section, we test Algorithm 2 on SDEs driven by Lévy processes with infinite activity of jumps. In this case the algorithm is of particular importance because variance of quantities of interest is typically very large and practical use of such models in financial engineering requires efficient variance reduction.

Consider the process to model an underlier:

$$S_i(t) = S_i(s) \exp\left\{rt + X_i(t)\right\},\tag{4.33}$$

where r > 0 and X is a d-dimensional process defined as

$$X(T) = \int_{s}^{T} b(t, X(t-)) dt + \int_{s}^{T} \sigma(t, X(t-)) dw(t) + \int_{s}^{T} \int_{\mathbb{R}} F(t, X(t-)) z \hat{N}(dt, dz), \quad (4.34)$$

with Lévy measure given by

$$\nu(\mathrm{d}z) = \begin{cases} C_{-}e^{-\mu(|z|-1)}\mathrm{d}z & \text{if } z < -1, \\ C_{-}|z|^{-(\alpha+1)}\mathrm{d}z & \text{if } -1 \le z < 0, \\ C_{+}|z|^{-(\alpha+1)}\mathrm{d}z & \text{if } 0 < z \le 1, \\ C_{+}e^{-\mu(|z|-1)}\mathrm{d}z & \text{if } 1 < z, \end{cases}$$

$$(4.35)$$

where  $C_-, C_+$  and  $\mu$  are positive constants and  $\alpha \in (0,2)$ . Throughout, we take  $\sigma(t,x)$  to be a constant matrix,  $\sigma(t,x) = \sigma \in \mathbb{R}^{d\times d}$  and  $F(t,x) = (f_1,\ldots,f_d) \in \mathbb{R}^d$ . Since the discounted price processes,  $\tilde{S}_i(t) = e^{-rt}S_i(t)$ , should be martingales, the drift component b(t,x) is chosen as (see [11] or [1, Sec 5.2])

$$b_i = -\frac{1}{2} \sum_{j=1}^d \sigma_{ij}^2 - \int_{\mathbb{R}} (e^{f_i z} - 1 - f_i z \mathbb{1}_{|z| < 1}) \nu(\mathrm{d}z).$$
 (4.36)

We consider this model in one and four dimensions. In all of the examples, we choose  $C_{-} = C_{+} = 1$ ,  $\alpha = 0.5$ ,  $\mu = 2$  and  $\varepsilon = 10^{-3}$ .

#### 4.3.1 One-dimensional European call

We consider the problem of pricing a European call option under model (4.33)-(4.36) with d = 1. Table 6 shows the results for Algorithm 2 in the case of r = 0.02,  $\sigma_{11} = 0.2$  and  $f_1 = 0.2$ . The computational speed up achieved here by the proposed control variate method is up to 8 times in comparison with vanilla MC.

Table 6: European Call option under exponential Lévy model (4.33): MC approximations (and a 95% confidence interval) with and without a control variate.

	u(0,1)	Vani	lla MC		Control	Relative		
	w(0,1)	$\hat{u}(0,1)$	Time (s)	M	$\hat{u}(0,1)$	Time (s)	M	Error
0.7	-	$0.46285 \pm 0.00100$	225	$3.00 \cdot 10^6$	$0.46298 \pm 0.00095$	28.1	$1.30000 \cdot 10^5$	0.38
0.8	-	$0.41284{\pm}0.00101$	208	$2.79 \cdot 10^6$	$0.41376 \pm 0.00103$	27.8	$1.25000 \cdot 10^5$	0.45
0.9	-	$0.36926 {\pm} 0.00099$	205	$2.76 \cdot 10^6$	$0.36934 {\pm} 0.00101$	25.2	$1.40000 \cdot 10^5$	0.52
1	-	$0.33181 {\pm} 0.00100$	193	$2.58 \cdot 10^6$	$0.33189 \pm 0.00100$	29.4	$1.35000 \cdot 10^5$	0.57
1.1	-	$0.29821 {\pm} 0.00102$	175	$2.34 \cdot 10^6$	$0.29721 {\pm} 0.00098$	25.7	$1.25000 \cdot 10^5$	0.59
1.2	-	$0.26838 {\pm} 0.00102$	166	$2.22 \cdot 10^6$	$0.26841 {\pm} 0.00101$	27.3	$1.40000 \cdot 10^5$	0.72
1.3	-	$0.24385 \pm 0.00105$	148	$1.99 \cdot 10^6$	$0.24272 \pm 0.00105$	25.4	$1.45000 \cdot 10^5$	0.84

#### 4.3.2 Four-dimensional call-on-max option

As before, consider the model (4.33)-(4.36) with d=4 and the terminal condition

$$f(x) = \left(\max(x_1, x_2, x_3, x_4) - K\right)_{+} \tag{4.37}$$

for some K>0. Table 7 shows the results of Algorithm 2 in the case of  $r=0.02,\,f=(0.2,0.15,0.1)^{\top}$  and

$$\sigma = 0.15L,\tag{4.38}$$

where L is the lower triangular matrix obtained via the Cholesky decomposition of the correlation matrix

$$LL^{\top} = \begin{bmatrix} 1 & 0.87 & 0.94 & 0.86 \\ 0.87 & 1 & 0.87 & 0.93 \\ 0.94 & 0.87 & 1 & 0.96 \\ 0.86 & 0.93 & 0.96 & 1 \end{bmatrix}. \tag{4.39}$$

Here the achieved speed up is up to 14 times

Table 7: Rainbow call-on-max option under four-dimensional exponential Lévy model (4.33): Monte Carlo approximations (and a 95% confidence interval) with and without a control variate.

	u(0,1)	Vanilla MC			Control	Relative		
	w(0,1)	$\hat{u}(0,1)$	Time (s)	M	$\hat{u}(0,1)$	Time (s)	M	Error
0.7	-	$0.61422 \pm 0.00100$	699	$2.28 \cdot 10^6$	$0.61501 \pm 0.00101$	47.9	$1.00 \cdot 10^5$	0.26
0.8	-	$0.53704 \pm 0.00099$	699	$2.28 \cdot 10^6$	$0.53739 \pm 0.00080$	72.9	$1.65 \cdot 10^5$	0.31
0.9	-	$0.46754 {\pm} 0.00100$	650	$2.12 \cdot 10^6$	$0.46939 \pm 0.00103$	51.1	$1.05 \cdot 10^5$	0.36
1	-	$0.40659 \pm 0.00100$	623	$2.03 \cdot 10^6$	$0.40621 {\pm} 0.00100$	50.3	$1.10 \cdot 10^5$	0.42
1.1	-	$0.35247{\pm}0.00101$	574	$1.87 \cdot 10^6$	$0.35399 \pm 0.00098$	60.2	$1.25 \cdot 10^5$	0.50
1.2	-	$0.30811 \pm 0.00102$	530	$1.72 \cdot 10^6$	$0.30763 \pm 0.00097$	56.9	$1.25{\cdot}10^5$	0.57
1.3	-	$0.26923 {\pm} 0.00099$	510	$1.68 \cdot 10^6$	$0.26925 {\pm} 0.00100$	61.6	$1.40 \cdot 10^5$	0.71

#### 4.4 Comparison with alternative complexity and variance reduction methods

So far, we have only compared our neural control variate method to vanilla MC. In this section, we compare our method to other standard complexity and variance reduction techniques. We discuss the Multilevel Monte Carlo (MLMC) method [19] and also a crude control variate method [21].

#### 4.4.1 Multilevel Monte Carlo method

Let us consider the problem of pricing a call-on-max option on the model (4.33)-(4.36) with d=2. That is,

$$f(x) = (\max(x_1, x_2) - K)_{+} \tag{4.40}$$

for K > 0.

In contrast to the previous experiments, here we do not compare our method with the vanilla MC method but instead the MLMC method. We fix h = T/1024, and then choose the levels of the MLMC method to be  $\{T/N: N=16, 64, 256, 1024\}$ , i.e. geometrically decreasing step size by a factor of 4. We use the jump-adapted Euler scheme, where between each jump the Brownian component is approximated by the Euler scheme while the jumps are simulated exactly [40]. In the context of MLMC, the jumps happen at the same time and with the same size across the coarse and fine paths [19]. Table 8 details the results for a range of strike prices. We observe a speed-up of up to 5.5 times achieved by our neural control variate method compared to MLMC.

Table 8: Rainbow call-on-max option under two-dimensional exponential Lévy model (4.33): Monte Carlo approximations (and a 95% confidence interval) with a control variate and MLMC approximation.

K	u(0,1)	MLMC			Control	Relative		
11	<i>a</i> (0, 1)	$\hat{u}(0,1)$	Time (s)	$\overline{M}$	$\hat{u}(0,1)$	Time (s)	M	Error
0.7	-	$0.71275 \pm 0.00049$	94.4	-	$0.71304 \pm 0.00050$	17.0	$9.50 \cdot 10^4$	0.11
0.8	-	$0.65320{\pm}0.00049$	91.8	-	$0.65331 {\pm} 0.00050$	20.0	$1.25 \cdot 10^5$	0.14
0.9	-	$0.59884 {\pm} 0.00049$	89.2	-	$0.59901 \pm 0.00049$	21.9	$1.30 \cdot 10^5$	0.15
1	-	$0.54941 {\pm} 0.00049$	86.3	-	$0.54955 {\pm} 0.00050$	21.8	$1.25 \cdot 10^5$	0.17
1.1	-	$0.50511 \pm 0.00049$	82.8	-	$0.50456 \pm 0.00049$	24.0	$1.65 \cdot 10^5$	0.20
1.2	-	$0.46439 \pm 0.00049$	80.0	-	$0.46466 \pm 0.00049$	24.3	$1.75 \cdot 10^5$	0.23
1.3	-	$0.42781 {\pm} 0.00049$	77.4	-	$0.42757 \pm 0.00050$	28.9	$2.05 \cdot 10^5$	0.27

**Remark 4.** We experimentally observed that MLMC outperforms our neural control variate method in the case of standard diffusions considered in Sections 4.1.1, 4.1.2 and 4.1.3 (usually

providing a 1.5-3 times speed-up). However, in the case of jump-diffusions the neural control variate method is competitive and performs better when the jump rate is high. In these cases, MLMC is less effective since the advantage of having a larger time-step is diminished when jumps are frequent. We also expect that the neural control variate method can outperform MLMC when stochastic models are more complex and hence cannot be simulated with larger time steps (due to some stability restrictions) as required for MLMC efficiency.

**Remark 5.** It is possible to integrate the multilevel method with the neural control variate method (Algorithm 1/2). For instance, the first-pass (the training of the control variates) can be done in the usual manner, while the second pass can utilize the MLMC method. Exploring this combination offers a potential direction for future research.

#### 4.4.2 Crude control variate

We now compare our method to using the well-known method of employing the terminal spot value as a control variate [21]. That is, for an asset price process X(t), using the random variable

$$\Gamma = e^{-rT} f(X(T)) + c(e^{-rT} X(T) - X(0)), \tag{4.41}$$

for some  $c \in \mathbb{R}$ . This has the same expectation as  $e^{-rT}f(X(T))$  given that the discounted process  $e^{-rt}X(t)$  is a martingale under the pricing measure.

We work in the same setting as Section 4.1.3, pricing a European call under the Heston model (4.15)-(4.16). The parameter  $c \in \mathbb{R}$  is chosen in the normal way to minimize the variance of  $\Gamma$ , see e.g. [21]. Table 9 compares the results of this method to Algorithm 1. We see a speedup across all strikes, but particularly for out-of-money options. It is well known that the terminal spot control variate performs worse for these options, see [21].

Table 9: European call option under the Heston model with  $v=0.15, r=0.02, \kappa=0.25, \theta=0.5, \sigma=0.3, \rho=-0.3, T=3$ : Monte Carlo approximations (and a 95% confidence interval) with our neural control variate and a crude control variate.

	u(0,1)	Crude Control Variate			Neural Cor	Relative		
11	ω(0,1)	$\hat{u}(0,1)$	Time (s)	$\overline{M}$	$\hat{u}(0,1)$	Time (s)	M	Error
0.7	0.47517	$0.47524\pm0.00010$	57.6	$9.60 \cdot 10^6$	$0.47519 \pm 0.00008$	45.3	$2.58 \cdot 10^6$	0.14
0.8	0.42623	$0.42633 \pm 0.00010$	77.2	$12.9 \cdot 10^7$	$0.42636 \pm 0.00010$	30.9	$1.65 \cdot 10^6$	0.16
0.9	0.38271	$0.38279 \pm 0.00010$	97.3	$16.4 \cdot 10^7$	$0.38267 \pm 0.00011$	32.0	$1.73 \cdot 10^6$	0.19
1	0.34406	$0.34415\pm0.00010$	118	$19.9 \cdot 10^7$	$0.34407 \pm 0.00009$	54.6	$3.17 \cdot 10^6$	0.24
1.1	0.30977	$0.30983 \pm 0.00010$	138	$23.2 \cdot 10^7$	$0.30971\pm0.00010$	27.4	$1.43 \cdot 10^6$	0.20
1.2	0.27934	$0.27934\pm0.00010$	157	$26.3 \cdot 10^7$	$0.27927\pm0.00011$	39.5	$2.20 \cdot 10^6$	0.28
1.3	0.25232	$0.25229\pm0.00010$	174	$29.2 \cdot 10^7$	$0.25232\pm0.00010$	46.3	$2.61 \cdot 10^6$	0.34

#### 4.5 Transfer learning

In the previous experiments, we initialise the weights of the ANN each time the parameters of the financial model change. However, if the change in parameters is small, e.g. when we vary the strike price while keeping the other parameters fixed, we can use the previous weights of the ANN as the initial weights of the next ANN. This approach is termed as transfer learning (see e.g. [23]). Such a procedure can reduce computational costs further by decreasing the training time. We demonstrate this with the following example. Consider the same experiment as in Section 4.1.2 (see Table 3). Table 10 shows the time taken for the control variate method from Table 3, where the weights of the ANN are initialised each time, compared to the method of transfer learning where the weights are only initialised once. Note that there is no difference in time for K=0.7, since the weights are initialised in the same way. We see that using transferred

weights can give up to 2 times of further speed up, giving overall speed-up up to 20 times in comparison with the plain vanilla MC. Hence, transfer learning can further accelerate variance reduction offered by Algorithms 1 and 2.

Table 10: Experiment from Section 4.1.2: comparison of times using weights re-initialised and weights transferred from previous simulations with different K.

K	Time using new weights	Time using transferred weights
0.7	20.2	20.2
0.8	20.0	11.5
0.9	20.8	10.2
1	17.5	8.6
1.1	20.2	8.3
1.2	18.8	11.0
1.3	17.1	8.1

## 5 Conclusions

In this paper we proposed novel Monte Carlo (MC) algorithms based on neural SDEs with control variates parameterized by neural networks, which effectively simulate SDEs with substantially reduced variance via efficient approximation of optimal control variates. We considered both SDEs driven by Wiener processes and by general Lévy processes including those with infinite activity. The use of deep learning allows us to find effective control variates on the fly (i.e., without prior training of a neural network) in a truly black-box fashion, in comparison with the use of linear regression for this purpose, where a careful selection of basis functions are needed separately for each model [36]. In numerical tests, we demonstrated that the proposed neural control variate MC can achieve a speed-up up to 40 times in comparison with the plain vanilla MC. We also showed that it can outperform the multi-level Monte Carlo method in some cases. Doing training of the network online is attractive from the practical perspective as it can be used immediately within the MC simulation and we showed in several numerical tests that it is efficient. An alternative to the exploited here online training is to train networks offline, which need to be done for any practical use in a parametric space, including parameters of the model for underliers and the payoff. Such a training procedure is typically computationally very costly and hence this approach requires knowing well in advance which model and its range of parameters are of future potential interest – in contrast to our approach where we train networks on the fly. The natural limitation of our algorithms is the dimension of the system of SDEs. They work well for relatively low dimensional SDEs as those typically used in financial engineering, when the cost of network training for a single set of parameters on a fly is relatively low. For larger systems, say of dimension 10 or more, the use of the neural control variates would normally require pre-training of networks (see e.g., [48]) in a parametric space. As usual (see, e.g., [37, 48]), the considered variance reduction does not introduce an additional bias, which is controlled in the standard fashion by a choice of a numerical method and an integration step [37] and which can be estimated in practice using the Talay-Tubaro expansion [45, 37]. The Monte Carlo error is also estimated in the usual way. In other words, the use of deep learning here (similarly to [48]) does not introduce errors which cannot be estimated theoretically or in practice. We considered here SDEs in the whole  $\mathbb{R}^d$  and we note that the proposed algorithms can be applied together with suitable SDEs' approximations [37] in the case of SDEs in bounded domains in  $\mathbb{R}^d$ , e.g. with an absorption condition on the boundary, which can be used for pricing barrier options.

## Acknowledgement

We are grateful for access to the University of Nottingham's Augusta HPC service.

#### Declarations of Interest

The authors report no conflicts of interest. The authors alone are responsible for the content and writing of the paper.

## References

- [1] Applebaum, D. Lévy Processes and Stochastic Calculus. 2nd ed. Cambridge: Cambridge University Press, 2009. DOI: 10.1017/CB09780511809781.
- [2] Asmussen, S. and Rosiński, J. "Approximations of small jumps of Lévy processes with a view towards simulation". J. Appl. Probab. 38(2) (2001), pp. 482–493. DOI: 10.1239/jap/996986757.
- [3] Barron, A. R. "Approximation and estimation bounds for artificial neural networks". Machine Learning 14(1) (1994), pp. 115–133. DOI: 10.1007/BF00993164.
- [4] Belomestny, D., Milstein, G. N., and Schoenmakers, J. G. M. "Sensitivities for Bermudan options by regression methods". *Decisions Econ. Finance* 33 (2010), pp. 117–138. DOI: 10.1007/s10203-009-0101-z.
- [5] Belomestny, D., Häfner, S., Nagapetyan, T., and Urusov, M. "Variance reduction for discretised diffusions via regression". J. Mathem. Anal. Applic. 458(1) (2018), pp. 393–418. DOI: 10.1016/j.jmaa.2017.09.002.
- [6] Berner, J., Grohs, P., and Jentzen, A. "Analysis of the generalization error: Empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations". SIAM J. Mathem. Data Science 2(3) (2020), pp. 631-657. DOI: 10.1137/19M125649X.
- [7] Berner, J., Grohs, P., Kutyniok, G., and Petersen, P. "The modern mathematics of deep learning". *Mathematical Aspects of Deep Learning*. Ed. by Grohs, P. and Kutyniok, G. Cambridge University Press, 2022, pp. 1–111. DOI: 10.1017/9781009025096.002.
- [8] Brigo, D. and Mercurio, F. Interest rate models theory and practice: with smile, inflation and credit. Berlin: Springer, 2006. DOI: 10.1007/978-3-540-34604-3.
- [9] Cont, R. and Tankov, P. Financial modelling with jump processes. Boca Raton: Chapman & Hall/CRC, 2004. DOI: 10.1201/9780203485217.
- [10] Cybenko, G. "Approximation by superpositions of a sigmoidal function". *Mathematics of Control, Signals, and Systems* 2(4) (1989), pp. 303–314. DOI: 10.1007/BF02551274.
- [11] Deligiannidis, G., Maurer, S., and Tretyakov, M. V. "Random walk algorithm for the Dirichlet problem for parabolic integro-differential equation". *BIT Numer. Math.* 61(4) (2021), pp. 1223–1269. DOI: 10.1007/s10543-021-00863-2.
- [12] Dick, J., Kuo, F., and Sloan, I. "High-dimensional integration: the quasi-Monte Carlo way". Acta Numerica 22 (2013), pp. 133–288. DOI: 10.1017/S0962492913000044.

- [13] Dingeç, K. D. and Hörmann, W. "A general control variate method for option pricing under Lévy processes". *European J. Oper. Research* 221(2) (2012), pp. 368–377. DOI: 10. 1016/j.ejor.2012.03.046.
- [14] Dynkin, E. B. Markov processes. Berlin: Springer, 1965. DOI: 10.1007/978-3-662-00031-1.
- [15] Freidlin, M. I. Functional integration and partial differential equations. Princeton: Princeton Univ. Press, 1985. DOI: 10.1515/9781400881598.
- [16] Friedman, A. Stochastic differential equations and applications. Berlin: Springer, 2011. DOI: 10.1007/978-3-642-11079-5\_2.
- [17] Garroni, M. G. and Menaldi, J. L. Second order elliptic integro-differential problems. New York: Chapman & Hall/CRC, 2002. DOI: 10.1201/9781420035797.
- [18] Geist, M., Petersen, P., Raslan, M., Schneider, R., and Kutyniok, G. "Numerical solution of the parametric diffusion equation by deep neural networks". J. Scien. Comp. 88 (2021). DOI: 10.1007/s10915-021-01532-w.
- [19] Giles, M. B. "Multilevel Monte Carlo methods". Acta Numerica 24 (2015), pp. 259–328. DOI: 10.1017/S096249291500001X.
- [20] Gladyshev, S. A. and Milstein, G. N. "The Runge-Kutta method for calculation of Wiener integrals of functionals of exponential type". Zh. Vychisl. Mat. i Mat. Fiz. 24 (1984), pp. 1136–1149.
- [21] Glasserman, P. Monte Carlo methods in financial engineering. Berlin: Springer, 2003. DOI: 10.1007/978-0-387-21617-1.
- [22] Gonon, L., Grohs, P., Jentzen, A., Kofler, D., and Šiška, D. "Uniform error estimates for artificial neural network approximations for heat equations". *IMA J. Numer. Anal.* 42(3) (2022), pp. 1991–2054. DOI: 10.1093/imanum/drab027.
- [23] Goodfellow, I., Bengio, Y., and Courville, A. Deep learning. MIT press, 2016.
- [24] Han, J., Jentzen, A., and E, W. "Solving high-dimensional partial differential equations using deep learning". *Proc. National Acad. Sci.* 115(34) (2018), pp. 8505–8510. DOI: 10.1073/pnas.171894211.
- [25] Heston, S. L. "A closed-form solution for options with stochastic volatility with applications to bond and currency options". *Review Finan. Studies* 6 (1993), pp. 327–343. DOI: 10.1093/rfs/6.2.327.
- [26] Hodgkinson, L., Heide, C. van der, Roosta, F., and Mahoney, M. W. "Stochastic continuous normalizing flows: training SDEs as ODEs". Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence. Ed. by Campos, C. de and Maathuis, M. H. Vol. 161. Proceedings of Machine Learning Research. PMLR, 2021, pp. 1130–1140.
- [27] Hornik, K., Stinchcombe, M., and White, H. "Multilayer feedforward networks are universal approximators". *Neural Networks* 2(5) (1989), pp. 359–366. DOI: 10.1016/0893-6080(89)90020-8.
- [28] Kampen, N. G. van. Stochastic processes in physics and chemistry. 3rd ed. Amsterdam: North Holland, 2007. DOI: 10.1016/B978-0-444-52965-7.X5000-4.
- [29] Kidger, P., Foster, J., Li, X., and Lyons, T. J. "Neural SDEs as infinite-dimensional GANs". Intern. Confer. Machine Learning. Vol. 139. PMLR, 2021, pp. 5453–5463.
- [30] Kingma, D. P. and Ba, J. "Adam: a method for stochastic optimization". *Intern. Confer. Learning Representations* (2015).
- [31] Kohatsu-Higa, A., Ortiz-Latorre, S., and Tankov, P. "Optimal simulation schemes for Lévy driven stochastic differential equations". Math. Comp. 83 (2014), pp. 2293–2324. DOI: 10.1090/S0025-5718-2013-02786-X.

- [32] Kyprianou, A. E. Introductory lectures on fluctuations of Lévy processes with applications. Springer, 2006. DOI: 10.1007/978-3-540-31343-4.
- [33] Merton, R. C. "Option pricing when underlying stock returns are discontinuous". *J. Finan. Econom.* 3(1) (1976), pp. 125–144. DOI: 10.1016/0304-405X(76)90022-2.
- [34] Milstein, G. N. and Schoenmakers, J. G. M. "Monte Carlo construction of hedging strategies against multi-asset European claims". Stoch. Stoch. Reports 73(1-2) (2002), pp. 125–157. DOI: 10.1080/10451120212868.
- [35] Milstein, G. N. and Tretyakov, M. V. "Numerical analysis of Monte Carlo evaluation of Greeks by finite differences". *J. Comp. Finance* 8 (2005), pp. 1–33. DOI: 10.21314/JCF. 2005.133.
- [36] Milstein, G. N. and Tretyakov, M. V. "Practical variance reduction via regression for simulating diffusions". SIAM J. Numer. Anal. 47(2) (2009), pp. 887–910. DOI: 10.1137/ 060674661.
- [37] Milstein, G. N. and Tretyakov, M. V. Stochastic numerics for mathematical physics. 2nd Edition. Berlin: Springer, 2021. DOI: 10.1007/978-3-030-82040-4.
- [38] Newton, N. J. "Continuous-time Monte Carlo methods and variance reduction". Numerical methods in finance. Ed. by Rogers, L. and Talay, D. Cambridge: Cambridge University Press, 1997, pp. 22–42. DOI: 10.1017/CB09781139173056.003.
- [39] Newton, N. J. "Variance reduction for simulated diffusions". SIAM J. Appl. Math. 54(6) (1994), pp. 1780–1805. DOI: 10.1137/S0036139992236220.
- [40] Platen, E. and Bruti-Liberati, N. Numerical solution of stochastic differential equations with jumps in finance. Berlin: Springer, 2010. DOI: 10.1007/978-3-642-13694-8.
- [41] Shiraya, K. and Takahashi, A. "A general control variate method for multi-dimensional SDEs: An application to multi-asset options under local stochastic volatility with jumps models in finance". European J. Oper. Res. 258(1) (2017), pp. 358–371. DOI: 10.1016/j.ejor.2016.08.060.
- [42] Shiraya, K., Uenishi, H., and Yamazaki, A. "A general control variate method for Lévy models in finance". European J. Oper. Res. 284(3) (2020), pp. 1190–1200. DOI: 10.1016/j.ejor.2020.01.043.
- [43] Sirignano, J. and Spiliopoulos, K. "DGM: A deep learning algorithm for solving partial differential equations". *J. Comput. Phys.* 375 (2018), pp. 1339–1364. DOI: 10.1016/j.jcp.2018.08.029.
- [44] Su, H., Tretyakov, M. V., and Newton, D. P. "Option valuation through deep learning of transition probability density". arxiv:2105.10467 (2021).
- [45] Talay, D. and Tubaro, L. "Expansion of the global error for numerical schemes solving stochastic differential equations". *Stoch. Anal. Appl.* 8 (1990), pp. 483–509. DOI: 10. 1080/07362999008809220.
- [46] Tretyakov, M. V. Introductory course on financial mathematics. London: ICP, 2013. DOI: 10.1142/p889.
- [47] Tzen, B. and Raginsky, M. "Neural stochastic differential equations: deep latent Gaussian models in the diffusion limit". arXiv:1905.09883 (2019).
- [48] Vidales, M. S., Siska, D., and Szpruch, L. "Unbiased deep solvers for linear parametric PDEs". *Applied Mathematical Finance* 28(4) (2021), pp. 299–329. DOI: 10.1080/1350486X.2022.2030773.
- [49] Wagner, W. "Monte Carlo evaluation of functionals of solutions of stochastic differential equations. Variance reduction and numerical examples". Stoch. Anal. Appl. 6 (1988), pp. 447–468. DOI: 10.1080/07362998808809161.

## Appendix A Artificial Neural Networks

We follow the notation in [7]. For  $L \in \mathbb{N}$ ,  $N = (N_0, \dots, N_L) \in \mathbb{N}^{L+1}$  and a Lipschitz function  $\rho : \mathbb{R} \to \mathbb{R}$ , a fully connected feed-forward neural network is defined by its architecture  $a = (N, \rho)$  and its realization  $\Phi_a : \mathbb{R}^{N_0} \times \mathbb{R}^{P(N)} \to \mathbb{R}^{N_L}$ , where P(N) is the number of parameters

$$P(N) := \sum_{l=1}^{L} N_l N_{l-1} + N_l. \tag{A.1}$$

The realization of the neural network architecture is of the form

$$\Phi_{a,\theta}(x) = W^{(L)} \left( \dots A^{(2)} \left( W^{(2)} \left( A^{(1)} (W^{(1)} x + b^{(1)}) \right) + b^{(2)} \right) \dots \right) + b^{(L)}, \tag{A.2}$$

where

$$W^{(l)} \in \mathbb{R}^{N_l \times N_{l-1}}, \ b^{(l)} \in \mathbb{R}^l, \ l = 1, \dots, L,$$
 (A.3)

and  $A^{(l)}: \mathbb{R}^{N_l} \to \mathbb{R}^{N_l}$  is defined by

$$A^{(l)}(x) = \left(\rho(x_1), \dots, \rho(x_{N_l})\right)^{\top}, \quad l = 1, \dots, L - 1,$$
(A.4)

and  $\theta = ((W^{(l)}, b^{(l)}))_{l=1}^L \in \mathbb{R}^{P(N)}$ . The neural network is said to have (L-1) hidden layers. As the activation function  $\rho$  we use ReLU. For further details, see e.g. [23].

Remark 6. Both the SELU and ELU activation functions were tested in addition to ReLU but neither performed better than ReLU. We experienced no issues during training with the ReLU activation, perhaps because we are working with quite shallow networks (3 layers) and we do not train for a long time (< 100 iterations). We have intentionally made the software straightforward to adapt such that if, for example, problems with ReLU become apparent in different examples, then the activation function can be easily changed.

# Appendix B Discussion of deep learning setup

In this appendix, we provide details concerning hyperparmeter estimation and a stopping rule to terminate the training procedure used in the experiments of Section 4.

Fix some tolerance level,  $\epsilon > 0$ . For the  $(1 - \alpha) \times 100\%$  confidence interval of the MC error, let  $\mathcal{C} = \mathcal{C}(\epsilon, \alpha)$  denote the total cost of the control variate method (i.e., Algorithm 1/2) required to reach this tolerance level. The total cost  $\mathcal{C}$  has two parts: the cost of training  $\mathcal{C}_{\text{train}}$  and the cost of running M independent trajectories to compute the quantity of interest  $\mathbb{E}\Gamma$ . The MC tolerance is equal to

$$\epsilon = \Phi^{-1} (1 - \alpha/2) \frac{\sqrt{\operatorname{Var}\bar{\Gamma}_{\theta^*}}}{\sqrt{M}},$$

and the cost to achieve the corresponding MC simulation is M multiplied by the cost to run a single trajectory. Then, the total cost  $\mathcal{C}$  can be conveniently expressed as

$$C = C_{\text{train}} + \Phi^{-1} (1 - \alpha/2)^2 C_{\text{batch}} \frac{\text{Var}\bar{\Gamma}_{\theta^*}}{\epsilon^2 S_{\text{batch}}},$$
(B.1)

where  $C_{\text{batch}}$  is the cost to sample one batch and  $S_{\text{batch}}$  is the size of the batch (note that the cost of simulating a single trajectory is equivalent to  $C_{\text{batch}}/S_{\text{batch}}$ ).

#### B.1 Hyperparameter estimation

In order to choose hyperparameters, i.e. the size of hidden layers, size of training sample, etc., we require a method of evaluating the effectiveness of the variance reduction. Naturally, we would like to choose hyperparameters to minimize the cost  $\mathcal{C}$ . Our approach is to perform the hyperparameter optimization on some of the simple examples and then use these hyperparameters on the remaining 'unseen' examples. We hope that this demonstrates that the algorithms proposed can be used in a black-box fashion. Of course, for optimal performance one could select hyperparameters for each individual example.

We identify the following hyperparameters: the number of hidden layers in each neural network and the size of these hidden layers (see Appendix A); the reduction factor of the number of steps in the training data compared to the number of steps in the final simulations, which we call the step factor; and the size of the training data. Note that for the hidden layer size, we add d neurons to the number stated, where d is the dimension of X; for example, when we choose a hidden layer size of 50 the neural network has 50 + d neurons in each layer. We list each hyperparameter and its corresponding search space in Table 11.

Table 11: Hyperparameters and their possible values.

In order to determine suitable values of the hyperparameters we minimize the cost  $\mathcal{C}$  (with  $\alpha = 0.05$ ,  $\epsilon = 10^{-3}$ ) averaged over four test examples from Section 4. Specifically, we use the experiments from Tables 2, 4, 5 and 6. We employ a grid search over the search space. The optimal values are those given in Table 1 and they are used in all of the experiments of Section 4.

#### B.2 Stopping rule for training

Rather than train for a fixed number of epochs for each problem, we propose a stopping rule which will determine when to stop training. On a heuristic level, we want to terminate the training procedure if the cost of training for a further epoch outweighs the variance reduction caused by this further training.

Denote the cost of training for one epoch (we assume that this is consistent across epochs) by  $C_{\text{train}}^{(1)}$ . Let  $\text{Var}\bar{\Gamma}_{\theta_i}$  denote the reduced variance after the *i*-th epoch with  $\text{Var}\bar{\Gamma}_{\theta_0} := \text{Var}\Gamma$ . Then, before the *i*-th epoch, we should decide to stop the training algorithm when

$$C_{\text{train}}^{(1)} + \Phi^{-1} (1 - \alpha/2)^2 C_{\text{batch}} \frac{\text{Var}\bar{\Gamma}_{\theta_i}}{\epsilon^2 S_{\text{batch}}} > \Phi^{-1} (1 - \alpha/2)^2 C_{\text{batch}} \frac{\text{Var}\bar{\Gamma}_{\theta_{i-1}}}{\epsilon^2 S_{\text{batch}}}, \tag{B.2}$$

which is equivalent to

$$\operatorname{Var}\bar{\Gamma}_{\theta_{i-1}} - \operatorname{Var}\bar{\Gamma}_{\theta_i} < \frac{\mathcal{C}_{\text{train}}^{(1)}}{K},$$
 (B.3)

where

$$K = \frac{\Phi^{-1}(1 - \alpha/2)^2 \mathcal{C}_{\text{batch}}}{\varepsilon^2 S_{\text{batch}}}.$$
 (B.4)

Of course, the variance after the *i*-th epoch is unknown until after the epoch, so we estimate the change  $\operatorname{Var}\bar{\Gamma}_{\theta_{i-1}} - \operatorname{Var}\bar{\Gamma}_{\theta_i}$  with the change from the previous epoch, that is  $\operatorname{Var}\bar{\Gamma}_{\theta_{i-2}} - \operatorname{Var}\bar{\Gamma}_{\theta_{i-1}}$ .

In practice, the cost to train for one epoch,  $\mathcal{C}_{\text{train}}^{(1)}$ , is estimated during training as the running average of the time taken for one epoch of training. The cost to sample one batch,  $\mathcal{C}_{\text{batch}}$ , is computed before training takes place.