

Multilevel Monte Carlo learning

Thomas Gerstner
Bastian Harrach
Daniel Roth

Department of Mathematics
Goethe University Frankfurt
60325 Frankfurt am Main, Germany

GERSTNER@MATH.UNI-FRANKFURT.DE
 HARRACH@MATH.UNI-FRANKFURT.DE
 ROTH@MATH.UNI-FRANKFURT.DE

Martin Simon

Deka Investment GmbH, Germany

Abstract

In this work, we study the approximation of expected values of functional quantities on the solution of a stochastic differential equation (SDE), where we replace the Monte Carlo estimation with the evaluation of a deep neural network. Once the neural network training is done, the evaluation of the resulting approximating function is computationally highly efficient so that using deep neural networks to replace costly Monte Carlo integration is appealing, e.g., for near real-time computations in quantitative finance. However, the drawback of these nowadays widespread ideas lies in the fact that training a suitable neural network is likely to be prohibitive in terms of computational cost. We address this drawback here by introducing a multilevel approach to the training of deep neural networks. More precisely, we combine the deep learning algorithm introduced by Beck et al. Beck et al. (2018) with the idea of multilevel Monte Carlo path simulation of Giles Giles (2008a). The idea is to train several neural networks, each having a certain approximation quality and computational complexity, with training data computed from so-called *level estimators*, introduced by Giles Giles (2008a). We show that under certain assumptions, the variance in the training process can be reduced by shifting most of the computational workload to training neural nets at coarse levels where producing the training data sets is comparably cheap, whereas training the neural nets corresponding to the fine levels requires only a limited number of training data sets. We formulate a complexity theorem showing that the multilevel idea can indeed reduce computational complexity.

Keywords: deep learning, multilevel, Monte Carlo, computational complexity

1. Introduction

Consider a multi-dimensional SDE

$$dS(t) = \mu(S, t) dt + \sigma(S, t) dW(t), \quad 0 < t \leq T, \quad (1.1)$$

with initial value S_0 , drift $\mu(S, t)$ and volatility $\sigma(S, t)$, which under certain conditions has a pathwise unique strong solution $S(t)$, see, e.g., Kloeden et al. (2012). We are interested in the expected value $V(S(T))$, where $V(S)$ is a scalar function (payoff) and S the solution

. The opinions expressed in this article are those of the authors and do not necessarily reflect views of Deka Investment GmbH.

of the above SDE. The Milstein discretization of the SDE with step-width h is of the form

$$\begin{aligned}\hat{S}_{n+1} &= \hat{S}_n + \mu(\hat{S}_n, t_n)h + \sigma(\hat{S}_n, t_n)\sqrt{h}\Delta Z_n \\ &\quad + \frac{1}{2}\sigma(\hat{S}_n, t_n)\sigma'(\hat{S}_n, t_n)\left((\sqrt{h}\Delta Z_n)^2 - h\right),\end{aligned}\tag{1.2}$$

with ΔZ_n i.i.d. standard normal for $n = 0, \dots, T/h - 1$, $t_n = nh$, and $\hat{S}_0 = S_0$. We denote the approximation of $V(S(T))$ using step-width h as follows:

$$P_h := V(\hat{S}_{T/h}).\tag{1.3}$$

The mean of the sampled payoff values given by $N^{-1} \sum_{i=1}^N P_h^{(i)}$ from N independent path simulations, is the simplest estimate of $\mathbb{E}[V(S(T))]$. Assuming certain conditions on the drift, volatility and payoff, see e.g. Kloeden et al. (2012), the estimators' mean squared error (MSE) is asymptotically of the form

$$\text{MSE} \approx c_1 N^{-1} + c_2 h^2,$$

with positive constants c_1, c_2 . Hence, we can achieve an error bound of $\mathcal{O}(\epsilon^2)$, for any $\epsilon > 0$, at a computational complexity of order $\mathcal{O}(\epsilon^{-3})$ for the MSE, i.e., it would require that $N = \mathcal{O}(\epsilon^{-2})$ and $h = \mathcal{O}(\epsilon^{-1})$. Giles (2008a) introduced the idea of multilevel Monte Carlo simulation, which achieves a complexity reduction. Under certain conditions and assuming, e.g., a Lipschitz-continuous payoff and the Milstein scheme, the complexity can be reduced to $\mathcal{O}(\epsilon^{-2})$, see Giles (2008b).

Let P_l denote the approximations, defined by (1.3), using the discretizations $h_l = 2^{-l}T$ for $l = 0, 1, \dots, L$. We have

$$\mathbb{E}[P_{h_L}] = \mathbb{E}[P_{h_0}] + \sum_{l=1}^L \mathbb{E}[P_{h_l} - P_{h_{l-1}}]$$

and the multilevel Monte Carlo idea is to independently estimate each of the expectations on the right-hand side. Therefore, consider the following so-called *level estimators* for $\mathbb{E}[P_{h_0}]$ and $\mathbb{E}[P_{h_l} - P_{h_{l-1}}]$ defined by

$$\hat{Y}_l = \begin{cases} N_0^{-1} \sum_{i=1}^{N_0} P_{h_0}^{(i)} & \text{for } l = 0, \\ N_l^{-1} \sum_{i=1}^{N_l} \left(P_{h_l}^{(i)} - P_{h_{l-1}}^{(i)} \right), & \text{for } l > 0, \end{cases}\tag{1.4}$$

using N_l paths and where the two discrete approximations $P_{h_l}^{(i)}$ and $P_{h_{l-1}}^{(i)}$ come from the same Brownian path, such that the difference $P_{h_l}^{(i)} - P_{h_{l-1}}^{(i)}$ is often small due to the strong convergence properties of the Milstein scheme, see, e.g., Giles (2008a) for further explanations. The final multilevel estimator \hat{Y} is given by the sum of the *level estimators*:

$$\hat{Y} = \sum_{l=0}^L \hat{Y}_l.\tag{1.5}$$

The complexity reduction of the multilevel approach depends on the *level estimators* (1.4). Giles presented a quite general complexity theorem that can be applied to a variety of financial models and payoffs without using a specific numerical approximation scheme. Further studies are e.g. made in Giles (2008b).

In practice, the volatility term $\sigma(\cdot, \cdot)$ is persistently re-calibrated as it depends on the market-implied volatility. Instead of contributing new price computations arising from an updated volatility term, this work's key motivation is to replace these by estimating an appropriately trained neural network \mathcal{N} . We refer to Higham and Higham (2019) for an in-depth introduction to neural network training. As a first focus in this work, we will study different approaches to generate the necessary training data and correctly choose the input parameters.

The neural network maps the parameter vector (inputs) to the expected value of a payoff function (output). Thus, we reformulate the issue as a suitable stochastic optimization problem and solve it through an artificial neural network approximation. We extend the above considerations to a set of stochastic differential equations and to a family of payoff functions. Consider the multi-dimensional stochastic differential equation

$$dS(t) = \mu(S, t, a) dt + \sigma(S, t, b) dW(t), \quad 0 < t \leq T, \quad (1.6)$$

with initial value $S(0) = s_0 \in \mathbb{R}_+$, time of maturity $T \in \mathbb{R}_+$, drift $\mu(S, t, a)$ and volatility $\sigma(S, t, b)$, with $a = (a_1, \dots, a_m) \in \mathbb{R}^m$ and $b = (b_1, \dots, b_s) \in \mathbb{R}^s$.

Let the stochastic process $S_{a,b,s_0,T}$ be the solution of the SDE (1.6) defined by the parameters a, b, s_0 and T . Consider a family of payoff functions $V(S, v)$, with $v = (v_1, \dots, v_r) \in \mathbb{R}^r$. Then, we will be interested in the expected value of

$$P : y \mapsto V(S_{a,b,s_0,T}(T), v), \quad (1.7)$$

for fixed

$$y := (a, b, v, s_0, T) \in Y \subset \mathbb{R}^V \times \mathbb{R}_+^2, \quad (1.8)$$

with $V = m + s + r$. We call Y the training set, i.e. it contains all parameter vectors of interest. Finally, we will be interested in

$$\bar{P} : y \mapsto \mathbb{E}[P(y)], \quad (1.9)$$

the input (model and payoff parameters) to price (expected value of the payoff) map and we aim to find an appropriate neuronal network for its approximation, i.e. a network $\mathcal{N} : Y \rightarrow \mathbb{R}$ minimizing

$$\|\bar{P}(y) - \mathcal{N}(y)\|_{L^p}, \quad (1.10)$$

for $1 \leq p \leq \infty$.

One intuitive approach to efficiently generate training data (outputs) for learning such a network is to use, e.g., a multilevel Monte Carlo estimator for reasonably chosen or randomly selected input values (for a high-dimensional case). One chooses a proper set of inputs of the form (1.8) and estimates the outputs (1.9) at the required accuracy.

However, Beck et al. (2018) presented an alternative approach: rather than using estimated prices for chosen input values, they use the individual sampled paths of the single level estimator (1.3) for randomly selected input values. In other words, one uses randomly sampled inputs and estimates single paths of (1.7), instead of (1.9), as outputs. Within this work, we will compare both approaches with respect to computational complexity.

This work's main idea is to use the advantages of the multilevel Monte Carlo path simulation and combine it with the procedure of using only single paths as training data. I.e., we will present an approach, which trains several neural networks $\mathcal{N}_l : Y \rightarrow \mathbb{R}$, for $l = 0, 1, \dots, L$, where each network uses paths of the multilevel *level estimators* (1.5) as training data. We will call this approach multilevel Monte Carlo learning.

Therefore, let's shortly extend the ideas of the Milstein payoff discretization (1.3) and the multilevel *level estimators* (1.5) to the reformulated optimization problem (1.10) on a training set Y .

Consider using the Milstein scheme (1.2), with step-width h , as an approximation of the solution of SDE (1.6), leading to

$$P_h : y \mapsto V \left(\left(\hat{S}_{a,b,s_0,T} \right)_{T/h}, v \right), \quad (1.11)$$

as an approximation of (1.7).

Similar ideas lead to paths of the *level estimators* on the training set given by

$$\hat{Y}_l : y \mapsto \begin{cases} P_{h_0}(y) & \text{for } l = 0, \\ P_{h_l}(y) - P_{h_{l-1}}(y), & \text{for } l > 0. \end{cases} \quad (1.12)$$

Hence, instead of using (1.11) for the training data, the training data for each network \mathcal{N}_l will be simulated by using the paths (1.12). All things considered, the multilevel Monte Carlo training approach searches the approximation

$$\hat{\mathcal{N}} := \sum_{l=0}^L \mathcal{N}_l, \quad (1.13)$$

which minimize (1.10).

The rest of this work's structure is as follows: First of all, we will present an introductory example to show a possible strength and an easy implementation for which no theoretical knowledge is needed.

Section 2 will briefly review and compare both of the approaches mentioned above, using one neural network with respect to the computational complexity. We will call these single-level approaches. The derived complexities will be the references for this work's primary goal, further complexity reductions. Then, the main complexity theorem and an extension of numerical examples will be presented. Sections 3 and 4 will give some possible extensions and a conclusion. In the appendix, we will discuss some mathematical background and present the proof. Furthermore, we will give some numerical results, studying the convergence with respect to the batch-size and some variance reduction effects.

1.1 Introductory example

We start by presenting an easy and short procedure to implement the multilevel approach, to which no further theoretical background knowledge is necessary for its application.

Consider the following procedure to train a neural network to achieve the required accuracy. The practitioner chooses a promising network structure, fixes a limit for the number of training steps, chooses a batch-size (training data used in each training step), and uses the training algorithm as stated in Beck et al. (2018). The approach computes the needed training data in each training step by using single paths, as in (1.11), on randomly selected inputs. If the resulting network does not achieve the required accuracy, the training process is repeated from scratch using increased batch-size, increased training steps, or adjusted network structure.

The examples in this section will use the following variation: Though, using a fixed amount of training steps and fixed network structure for all training processes, we gradually increase the batch-size until the required accuracy is achieved. We will use a five-dimensional training set from a geometric Brownian motion's initial parameters since a closed solution is available for this example.

1.1.1 SINGLE-LEVEL LEARNING EXAMPLE

First, we train a single neural network, by successively increasing the batch-size, until the required accuracy is achieved. Consider a geometric Brownian motion

$$dS(t) = \mu S(t) dt + \sigma S(t) dW(t), \quad 0 < t \leq T, \quad (1.14)$$

with initial value $S(0) = s_0$, constant drift μ and constant volatility σ .

Furthermore, consider a European call option given by

$$V(S, K) := \max\{S - K, 0\}, \quad (1.15)$$

with fixed strike price K . Hence, we want to approximate

$$P : y \mapsto V(S_{\mu, \sigma, s_0, T}(T), K)$$

with $y = (\mu, \sigma, s_0, T, K) \in Y$. Technically, the final price is denoted as the discounted expected payoff, which, for simplicity, we omit here.

Consider the training set $Y \subset \mathbb{R}^5$ to be of the specific form:

$$Y = [0.02, 0.05] \times [0.1, 0.2] \times [80, 120] \times [0.9, 1.0] \times [109, 110].$$

It is well known that the GBM (1.14) together with payoff (1.15) lead to a closed solution for the option price for each $y \in Y$, see e.g. Hull (2008).

The idea is to train a neural network to evaluate the expected value up to a required error ϵ for each input vector y within the training set Y . E.g., requiring accuracy of $\epsilon = 0.01$ (L^∞ -error) of the trained network we use the feasible step-width $h = 1/128$ for the Milstein scheme and the paths discretization (1.11).

As introduced above, we will use the algorithm of Beck et al., to which the PYTHON source code can be found in section 4 of Beck et al. (2018). The code uses the open-source

Parameter	Value
neurons	(50, 50, 1)
decay rate	0.1
initial learning rate	0.01
step rate	40.000
training steps	150.000

Table 1.1: Structure and learning rate parameters for each neural network.

batch-size	mean of L^∞	time (hours)
125.000	0.0273	2.32h
500.000	0.0180	7h
2.000.000	0.0119	26.66h

Table 1.2: Mean of the L^∞ -error for the single-level algorithms using the Milstein discretization (1.2), whereat the simulations were repeated 10 times. The last column shows the computation time using a Nvidia K80 GPU.

software library TensorFlow. For our tests, we slightly modify the code: instead of using fixed learning-rate boundaries, we use exponential decay, which, however, would deliver similar results for specific decay parameters.

We obtained feasible results using network structure and training parameters for the introduced training set, as stated in table 1.1, see Higham and Higham (2019) for further parameter explanations. The results of the training processes for increasing batch-sizes can be found in table 1.2. The computations were made on an Nvidia K80 GPU and were repeated 10 times for each batch-size. The L^∞ error was estimated by comparing the approximation with the closed solution on 2.000.000 randomly selected initial parameter vectors.

We see that a batch-size of 2.000.000 achieves a feasible accuracy, to which the training process took 26.66 hours.

1.1.2 MULTILEVEL LEARNING EXAMPLE

Now, we explain the procedure for the multilevel approach. To present easy comparable results, we use the same network structure and amount of training steps used for the single-level approach for each of the networks we will train. Furthermore, we will use the same model setting and training set, as in the example above. However, we will use individualized batch-sizes. We will increase these likewise until the required accuracy is achieved.

We use the paths of the *level estimators*, as described in (1.12), to compute the training data in each training step. We present the modification of the code of Beck et al. used for the simulation of the *level estimators'* paths, e.g., for $\hat{P}_1 - \hat{P}_0$, in listing 1 in the appendix.

The used batch-sizes M_l for each net and the amount of levels/nets L are calculated with the multilevel sample estimator of Giles suggested in Giles (2008a) (for which MATLAB codes can be found in Giles). The code originally estimates the Monte Carlo sample-sizes N_l and amount of levels L needed for the Multilevel approach to achieve a required accuracy ϵ .

level l	0	1	2	3	4	5	6	7
N_l	3.000.000	72695	27756	10550	3691	1308	476	182

Table 1.3: Estimated needed Monte Carlo samples N_l for the Multilevel Monte Carlo approach for $y = (0.05, 0.2, 100, 1, 110)$.

multilevel id	level l	0	1	2	3	4	5	6	7
1	M_l	75.000	1817	690	264	93	33	12	5
2	M_l	300.000	7268	2760	1056	372	132	48	20
3	M_l	1.200.000	29072	11040	4224	1488	528	192	80

Table 1.4: Estimated needed batch-size for the training of the specific level nets for $l = 0, \dots, 7$.

Let us present an example of its application. For e.g. $\epsilon = 0.01$ and $y = (0.05, 0.2, 100, 1, 110)$ the code delivers parameters as stated in table 1.3.

For the multilevel approach, we use these sample-sizes in the following way: We use the multilevel sample-ratios N_l/N_0 , for $l = 1, \dots, L$, as the multilevel batch-size-ratios. Furthermore, we use the amount of multilevel levels L as the amount of multilevel networks.

I.e. we will multiply the above ratios with the initial batch-size M_0 used for network \mathcal{N}_0 . Hence, we obtain $M_l = M_0 \cdot N_l/N_0$ for $l = 1, \dots, L$ as the batch-sizes of the networks \mathcal{N}_l .

If we, e.g., study the ratios in table 1.3, we see that the ratio between level one and level zero is given by $N_1/N_0 = 72.695/3.000.000 \approx 0.024$. Now, by multiplying this ratio to an initial batch-size of e.g. $M_0 = 75.000$ delivers the batch-size $M_1 = 1817$, for the net \mathcal{N}_1 on level $l = 1$. With increasing initial batch-sizes M_0 , we obtain the batch-sizes for the remaining networks as presented in table 1.4, at which, for a certain initial batch-size, we summarized the set of batch-sizes to a so-called multilevel id.

The multilevel training results using the estimated amount of levels and batch-sizes and its comparison to the single-level approach are given in table 1.5.

single-level		multilevel	
batch-size	mean error (time)	id	mean error (time)
125.000	0.0273 (2.32h)	1	0.0290 (4.15h)
500.000	0.0182 (7h)	2	0.0184 (5.29h)
2.000.000	0.0119 (26.66h)	3	0.0103 (11.18h)

Table 1.5: Mean of the L^∞ -error for the single-level algorithm using the Milstein discretization (second column). The fourth column shows the mean of the L^∞ -error for the multilevel algorithm using respective batch-sizes as presented in table 1.4. All simulations were repeated 10 times. The brackets show needed computation time using a Nvidia K80 GPU.

Using the network structure, learning rates, and training steps of the single-level approach, we see that the multilevel approach leads to significant time-saving. For the lowest chosen batch-size (125.000), we see that single-level learning is slightly faster than multilevel learning. A reason for this could be that learning 8 neural networks results in higher basic costs. However, if we require a higher error bound, e.g., as achieved for this low batch-size, the above-introduced procedure would suggest fewer neural networks.

2. Neural network training

In this section, we will study the error sources and computational costs. Therefore, we start with a more profound introduction to neural network training. Then, we compare the first introduced approach and the approach used in section 1.1.1 with respect to computational complexity. The second subsection will introduce the multilevel approach and present the computational complexity theorem.

2.1 Single-level: error sources and complexity

First, we want to discuss some neural networks' properties and their training processes, such as error sources and computational cost. For a more general introduction to neural networks, we refer to Higham and Higham (2019). This work aims to train a neural network in such a way that it is capable of evaluating the expected value up to a required error ϵ for each input vector within the training set Y .

We study the training process of a generic artificial neural network (see e.g. (A) for the definition) which is given by the following series of functions

$$\mathcal{N}_{\nu, \theta_i, P_h, \mathbb{L}_M} : Y \rightarrow \mathbb{R}, \quad (2.1)$$

for $i = 1, \dots, K$, with initial weights θ_0 . I.e. each of the K training steps modifies the weights $\theta_i \in \mathbb{R}^\nu$ and hence defines a new element of the series. For each training process, we fix the network structure ν (layers and neurons), the approach P_h to generate the training data and the amount of samples M (batch-size) used to evaluate the loss function \mathbb{L}_M .

I.e., we will neither modify the estimation P_h nor the batch-size M during each of the training processes.

Demanding a specific error bound ϵ , the challenge consists of choosing the most efficient parameters ν, M, h , and K .

Since, using a stochastic gradient descent algorithm, we will study the expectation of (1.10). Therefore, we will be interested in finding certain weights after K trainings steps satisfying

$$\mathbb{E} \left[\left\| \bar{P}(y) - \mathcal{N}_{\nu, \theta_K, P_h, \mathbb{L}_M}(y) \right\|_{L^1}^2 \right] < \epsilon^2, \quad (2.2)$$

for a required error $\epsilon > 0$. Using the standard variance expansion, the left hand side can be expanded to

$$\mathbb{V} \left[\left\| \bar{P}(y) - \mathcal{N}_{\nu, \theta_K, P_h, \mathbb{L}_M}(y) \right\|_{L^1} \right] + \mathbb{E} \left[\left\| \bar{P}(y) - \mathcal{N}_{\nu, \theta_K, P_h, \mathbb{L}_M}(y) \right\|_{L^1} \right]^2. \quad (2.3)$$

We specify the error sources by further decomposing the inner term of (2.2) to obtain

$$\|\bar{P}(y) - \mathcal{N}_{\nu, \theta_K, P_h, \mathbb{L}_M}(y)\|_{L^1} \leq \|\bar{P}(y) - \mathbb{E}[P_h(y)]\|_{L^1} \tag{e1}$$

$$+ \|\mathbb{E}[P_h(y)] - \mathcal{N}_{\nu, \Theta, \mathbb{E}[P_h], \mathbb{L}}(y)\|_{L^1} \tag{e2}$$

$$+ \|\mathcal{N}_{\nu, \Theta, \mathbb{E}[P_h], \mathbb{L}}(y) - \mathcal{N}_{\nu, \Theta, P_h, \mathbb{L}}(y)\|_{L^1} \tag{e3}$$

$$+ \|\mathcal{N}_{\nu, \Theta, P_h, \mathbb{L}}(y) - \mathcal{N}_{\nu, \theta_K, P_h, \mathbb{L}}(y)\|_{L^1} \tag{e4}$$

$$+ \|\mathcal{N}_{\nu, \theta_K, P_h, \mathbb{L}}(y) - \mathcal{N}_{\nu, \theta_K, P_h, \mathbb{L}_M}(y)\|_{L^1}. \tag{e5}$$

Let us shortly explain the above individual error sources, whereat we will have further studies on the mathematical background of the decomposition in appendix A. The first line (e1) describes the error made through the discretization of the stochastic process, using a discretization scheme with step-width h . Error (e2) describes the error made through the approximation of this function with a neural network with fixed structure ν , whereat here we assume to know the best respective weights Θ (best fit). The error arising in (e3) results from the statistical error of using an approximation of the expectation. Finally, the optimization algorithm using K iteration steps leads to (e4) and the approximation of the loss function by M samples per training step results in (e5).

Since, we assume a fixed neural network structure ν , the computational complexity $C_{\mathcal{N}}$ of the training process for a neural network can be bounded by

$$C_{\mathcal{N}} \leq c(K \cdot \text{cost}_{\text{training step}})$$

with a positive constant c , where each of the K training steps consist of the computation of the training data, evaluating of the loss function and modifying the weights to minimize the loss function.

Now, we want to compare both methods mentioned in the introduction based on this error decomposition. The first approach uses an estimation of the expectation (e.g., computed through multilevel Monte Carlo) as P_h , computed typically on deterministically selected input parameters (for a low dimensional case). The second approach, introduced by Beck et al. (2018), uses only single paths instead of an expectation estimation, as P_h , computed on randomly selected input parameter vectors.

At a first glance, if choosing a proper h , both approaches should have similar properties for the error sources (e1), (e2) and (e4), but differ in (e3), (e5) and obviously in the computation cost. However, as we will see in appendix A.3, error (e3) is negligible. Now, by interpreting (e5) as an integral approximation problem using M samples for the estimation, a possible description of the differences could be as follows: While we expect the first approach to have higher convergence order with respect to batch-size, it may suffer under the curse of dimensionality for high dimensional training sets. On the other hand, we expect the approach of Beck et al. (2018) to have a lower convergence order with respect to batch-size, but it could be computed without suffering from the curse of dimensionality. Furthermore, as explained in the introduction, the cost for the training data computed with the first approach is of order $\mathcal{O}(\epsilon^{-2})$, where the cost for a path is of order $\mathcal{O}(h^{-1})$.

Both approaches, using somewhat simplified assumptions, could lead to similar complexities presented in the following two lemmata for a low-dimensional case. For the first approach, studied in lemma 2.1, we assume the convergence order of the loss function with

respect to M (batch-size) to be one. For the second approach, studied in lemma 2.2, we assume the loss function's convergence order with respect to M to be $1/2$. For both approaches, we will assume a convergence of $1/2$ with respect to training steps K , and we consider a network structure ν such that (e2) can be neglected.

Lemma 2.1 *Consider the training of a neural network, as described in (2.1), for the approximation of (1.9) on a training set $Y \subset \mathbb{R}$. If there exist positive constants c_1, c_2, c_3 and c_4 such that the decomposition (e1)-(e5) of (2.2) can be bounded by*

$$\begin{aligned} \text{(e1)} &\leq \frac{1}{4}\epsilon, & \mathbb{E}[\text{(e4)}] &\leq c_3 K^{-1/2}, & \mathbb{E}[\text{(e5)}] &\leq c_2 M^{-1}, \\ \text{(e2)} &\leq \frac{1}{4}\epsilon, & \mathbb{V}[\text{(e4)}] &\leq c_3^2 K^{-1}, & \mathbb{V}[\text{(e5)}] &\leq c_2^2 M^{-2}, \\ \text{(e3)} &= 0, \end{aligned}$$

such that the computational complexity of the training is bounded by

$$C_{\mathcal{N}} \leq c_4 K M \epsilon^{-2}.$$

Then there exist positive constants c_5, M and K such that the overall error (2.2) can be bounded by ϵ^2 with a computational complexity bound of

$$C_{\mathcal{N}} \leq c_5 \epsilon^{-5}.$$

Proof See appendix B. ■

Lemma 2.2 *Consider the training of a neural network, as described in (2.1), for the approximation of (1.9) on a training set $Y \subset \mathbb{R}$. If there exist positive constants c_1, c_2, c_3 and c_4 such that the decomposition (e1)-(e5) of (2.2) can be bounded by*

$$\begin{aligned} \text{(e1)} &\leq c_1 h, & \mathbb{E}[\text{(e4)}] &\leq c_3 K^{-1/2}, & \mathbb{E}[\text{(e5)}] &\leq c_2 M^{-1/2}, \\ \text{(e2)} &\leq \frac{1}{4}\epsilon, & \mathbb{V}[\text{(e4)}] &\leq c_3^2 K^{-1}, & \mathbb{V}[\text{(e5)}] &\leq c_2^2 M^{-1}, \\ \text{(e3)} &= 0, \end{aligned}$$

such that the computational complexity of the training is bounded by

$$C_{\mathcal{N}} \leq c_4 K M h^{-1}.$$

Then there exist positive constants c_5, h, M and K such that the overall error (2.2) can be bounded by ϵ^2 with a computational complexity bound of

$$C_{\mathcal{N}} \leq c_5 \epsilon^{-5}.$$

Proof See appendix B. ■

If the assumptions are reasonable, improved numerical integration techniques such as sparse grid integration for the first approach, see, e.g., Gerstner and Griebel (1998), or variance reduction, for the second approach, see, e.g., Glasserman (2003), should reduce the overall complexity. Further studies on the implications of variance reduction techniques for the second approach will follow in section 2.2.3.

2.2 Multilevel Monte Carlo training

The following section will contain the error decomposition, the main complexity theorem, and a numerical experiment for the multilevel Monte Carlo training approach.

2.2.1 ERROR DECOMPOSITION

In this section, we will study the error sources of the multilevel training approach.

Since, the approach consists in training of individual neural networks for each level, we aim to find suitable weights on each level, such that the networks are approximations of the expectations of (1.12). We aim to bound

$$\left\| \mathbb{E}[\hat{Y}_l(y)] - \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_l, \mathbb{L}_{M_l}}(y) \right\|_{L^p},$$

for each level $l = 0, \dots, L$ using K_l training steps and loss functions discretized by using batch-size M_l . Again, we assume all networks to have the same fixed structure ν .

Then, the final multilevel approximation $\hat{\mathcal{N}}$ will be given by

$$\hat{\mathcal{N}} : y \mapsto \mathcal{N}_{\nu, \theta_{K_0}^0, \hat{Y}_0, \mathbb{L}_{M_0}}(y) + \sum_{l=1}^L \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_l, \mathbb{L}_{M_l}}(y), \quad (2.4)$$

whereat it should satisfy

$$\mathbb{E} \left[\left\| \bar{P}(y) - \hat{\mathcal{N}}(y) \right\|_{L^1}^2 \right] < \epsilon^2, \quad (2.5)$$

for a required error $\epsilon > 0$. Using the standard variance expansion, the left hand side can be expanded to

$$\mathbb{V} \left[\left\| \bar{P}(y) - \hat{\mathcal{N}}(y) \right\|_{L^1} \right] + \mathbb{E} \left[\left\| \bar{P}(y) - \hat{\mathcal{N}}(y) \right\|_{L^1} \right]^2. \quad (2.6)$$

Again, we specify the error sources by decomposing the inner term of (2.5) by

$$\left\| \bar{P}(y) - \hat{\mathcal{N}}(y) \right\|_{L^1} \leq \left\| \bar{P}(y) - \mathbb{E}[\hat{P}_{h_L}(y)] \right\|_{L^1} \quad (E1)$$

$$+ \sum_{l=0}^L \left\| \mathbb{E}[\hat{Y}_l(y)] - \mathcal{N}_{\nu, \Theta^l, \mathbb{E}[\hat{Y}_l], \mathbb{L}}(y) \right\|_{L^1} \quad (E2)$$

$$+ \sum_{l=0}^L \left\| \mathcal{N}_{\nu, \Theta^l, \mathbb{E}[\hat{Y}_l], \mathbb{L}}(y) - \mathcal{N}_{\nu, \Theta^l, \hat{Y}_l, \mathbb{L}}(y) \right\|_{L^1} \quad (E3)$$

$$+ \sum_{l=0}^L \left\| \mathcal{N}_{\nu, \Theta^l, \hat{Y}_l, \mathbb{L}}(y) - \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_l, \mathbb{L}}(y) \right\|_{L^1} \quad (E4)$$

$$+ \sum_{l=0}^L \left\| \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_l, \mathbb{L}}(y) - \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_l, \mathbb{L}_{M_l}}(y) \right\|_{L^1}. \quad (E5)$$

The explanations of the error sources are analogous to those of (e1) to (e5), whereas from (E2) to (E5) we use the sum of the $L + 1$ errors.

Again by fixing the net structures, the computational complexity $C_{\mathcal{N}_l}$ of the training for each network $\mathcal{N}_l := \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_l, \mathbb{L}_{M_l}}$ is bounded by

$$C_{\mathcal{N}_l} \leq c(K_l \cdot \text{cost}_{\text{training step on level}}), \quad (2.7)$$

with a positive constant c . Each of the K_l training steps consist of the computation of the M_l training data of type \hat{Y}_l , the calculation of the loss function $\hat{\mathbb{L}}_l$ and the modification of the weights to minimize the loss function.

2.2.2 COMPLEXITY THEOREM

In this subsection, we present the main complexity theorem. Since further studies on the error sources are required, we will word some assumptions quite generally. The idea of theoretical complexity reduction can be described as follows. The multilevel approach uses different time-steps for the discretization to obtain a complexity reduction. Under certain circumstances, this induces different amounts of random samples on each level, especially fewer samples on the finer discretizations. The idea can be transformed into multilevel training, which - when using finer path discretizations - we expect to result in lower needed batch-sizes. As introduced by Giles (2008a) the variance of the *level estimators* needs to be bounded with an order of $\mathcal{O}(h_l^\beta)$, for $\beta > 0$. For e.g. a European call option Giles, Debrabant and Rössler prove $\beta = 2$ in Giles et al. (2013).

Theorem 2.1 *Consider a multilevel training process as described in (2.4), for the approximation of (1.9) on a specific training set Y . If there exist positive constants $\alpha \geq 1/2, \beta, \gamma, \eta, c_1, c_2, c_3$ and c_4 such that the decomposition (E1)-(E5) of (2.5) can be bounded by*

$$\left\| \bar{P}(y) - \mathbb{E}[\hat{P}_{h_l}(y)] \right\|_{L^1} \leq c_1 h_l, \quad (\text{A1})$$

$$\left\| \mathbb{E}[\hat{Y}_l(y)] - \mathcal{N}_{\nu, \Theta^l, \mathbb{E}[\hat{Y}_l], \mathbb{L}}(y) \right\|_{L^1} \leq \frac{1}{\sqrt{32}} \epsilon, \quad (\text{A2})$$

$$\left\| \mathcal{N}_{\nu, \Theta^l, \mathbb{E}[\hat{Y}_l], \mathbb{L}}(y) - \mathcal{N}_{\nu, \Theta^l, \hat{Y}_l, \mathbb{L}}(y) \right\|_{L^1} = 0, \quad (\text{A3})$$

$$\mathbb{E} \left[\left\| \mathcal{N}_{\nu, \Theta^l, \hat{Y}_l, \mathbb{L}}(y) - \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_l, \mathbb{L}}(y) \right\|_{L^1} \right] \leq c_2 h_l^{\alpha\gamma} K_l^{-1/2}, \quad (\text{A4 i})$$

$$\mathbb{V} \left[\left\| \mathcal{N}_{\nu, \Theta^l, \hat{Y}_l, \mathbb{L}}(y) - \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_l, \mathbb{L}}(y) \right\|_{L^1} \right] \leq c_2^2 h_l^{2\alpha\gamma} K_l^{-1}, \quad (\text{A4 ii})$$

$$\mathbb{E} \left[\left\| \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_l, \mathbb{L}}(y) - \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_l, \mathbb{L}_{M_l}}(y) \right\|_{L^1} \right] \leq c_3 h_l^{\beta/2} \rho_l^\eta M_l^{-1/2}, \quad (\text{A5 i})$$

$$\mathbb{V} \left[\left\| \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_l, \mathbb{L}}(y) - \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_l, \mathbb{L}_{M_l}}(y) \right\|_{L^1} \right] \leq c_3^2 h_l^\beta \rho_l^{2\eta} M_l^{-1}, \quad (\text{A5 ii})$$

with $\rho_l = K_l^{-1/2}$, such that the computational complexity of the training for each net is bounded by

$$C_l \leq c_4 h_l^{-1} M_l K_l. \quad (2.8)$$

level l	0	1	2	3	4	5	6	7
M_l	1200000	64000	32000	16000	8000	4000	2000	1000
K_l	150000	20000	19000	18000	15000	14000	13000	11000

Table 2.1: Batch-sizes M_l and training steps K_l for the training of the specific level nets \mathcal{N}_l for $l = 1, \dots, 8$.

Then, there exists a positive constants c_5 such that for any $\epsilon < e^{-1}$, there are values L, M_l and K_l for which

$$\mathbb{E} \left[\left\| \bar{P}(y) - \hat{\mathcal{N}}(y) \right\|_{L^1}^2 \right] \tag{2.9}$$

can be bounded by ϵ^2 , with a computational complexity $C_{\mathcal{N}}$ with bound

$$C_{\mathcal{N}} \leq c_5 \begin{cases} \epsilon^{-3.0}, & \text{for } \eta = 0.5, \gamma = 2, \beta = 2, \alpha = 1, \\ \epsilon^{-3.0} |\log \epsilon|^4, & \text{for } \eta = 0.5, \gamma = 1, \beta = 1, \alpha = 1, \\ \epsilon^{-3.5} |\log \epsilon|^4, & \text{for } \eta = 0.25, \gamma = 1, \beta = 1, \alpha = 1, \\ \epsilon^{-4} |\log \epsilon|^5, & \text{for } \eta = 0, \gamma = 0, \beta = 1, \alpha = 1, \\ \epsilon^{-5} |\log \epsilon|^4, & \text{for } \eta = 0, \gamma = 0, \beta = 0, \alpha = 1, \\ \epsilon^{-6} |\log \epsilon|^4, & \text{for } \eta = 0, \gamma = 0, \beta = 0, \alpha = 1/2. \end{cases}$$

Proof See appendix B ■

Remark 2.2 By assuming $\eta = \gamma = 0$ and $\beta = \alpha = 1$, we focus on the level effect with respect to the batch-size, as, e.g., studied in the introductory example.

2.2.3 NUMERICAL RESULTS

We will now present an extension of the example of section 1.1 by considering the level effect with respect to training steps.

We extend the example of section 1.1 by including the level effect with respect to training steps, as introduced in theorem 2.1.

Using the identical model and network parameters, we will use the number of training steps and batch-sizes, as stated in table 2.1.

The mean and standard deviation of the L^∞ for this modified training are presented in table 2.2.

The multilevel training results using the stated amount of levels, batch-sizes, and training steps are given in table 2.2.

Compared to table 1.5, we see a further computational reduction while satisfying the required error-bound.

3. Extensions

In this subsection, we want to discuss some promising possible extensions beyond this work's scope except the first one.

single net: mean error (time)	multilevel: mean error (time)
0.0119 (26.66h)	0.0111 (9.66h)

Table 2.2: Mean of the L^∞ -error for the single-level algorithm using the Milstein discretization and the multilevel algorithm using respective batch-sizes as presented in table 2.1. All simulations were repeated 10 times. The brackets show needed computation time using a Nvidia K80 GPU.

3.1 Optimal K_l

As explained in the introduction, we could use a second level effect by using a possible connection between the variance of the *level estimators*' variance and training steps needed for the respective network. In the example above, we explicitly used the same parameters of the single-level for the multilevel approach. This method implied a relatively straightforward procedure to implement the multilevel approach and still had an advantage in computation time. However, we observed that for the finest level, much fewer training steps would be sufficient. Therefore, it could lead to further savings using individual decay rates, initial learning rates, or training steps. Further studies are made in section 2.2.2.

3.2 Optimal N_l

We only used single paths ($N_l = 1$) to compute the training data for the computations above. We discussed the advantages and disadvantages of explicitly computed prices in lemma 2.2 and lemma 2.1. Furthermore, we will see in appendix A.3 that this approach does not lead to a bias for the final trained neural network under certain assumptions. Nevertheless, if we study (A4 i) to (A5 ii) we see that the estimator's variance could affect the needed batch-size M_l and the needed training steps K_l . Therefore, we believe that further studies on N_l could lead to efficiency improvements. For example, for levels with a high estimators' variance, increasing N_l could be reasonable.

3.3 Optimal H

In our work so far, we have not specified H , which is the factor by which the time-step is refined. Again, we refer to Giles (2008a) for some further explanations on H for the multilevel Monte Carlo estimator. For the multilevel Monte Carlo simulation, further studies on H could lead to further efficiency improvements for the multilevel training.

3.4 Individual structure ν_l

In this work, we fixed the net structure ν for each neural net, and for simplicity, we assumed the approximation error to be negligible. Nevertheless, we see the potential for further studies on individual structures ν_l , which could further improve efficiency.

4. Conclusion

In this work, we combined the ideas of a single-paths deep learning approximation with the multilevel Monte Carlo path simulation concept. We showed that the resulting multilevel Monte Carlo training approach could reduce the complexity of the training process.

Deep learning algorithms have become very popular in recent years. However, there are not many rigorous mathematical convergence results for the different error sources. The decomposition into three parts, the approximation error, the generalization error, and the optimization error, and the first overall error analysis was made by Beck et al. (2019). However, their convergence speed analysis is far from optimal and suffers from the curse of dimensionality. Hence, we worded the main theorem quite generally. Nevertheless, several challenging areas for further research arise in this work.

Therefore, the first is a more in-depth theoretical analysis of the convergence speed with respect to each error source.

The second is the individual network structure. In this work, we used a fixed network structure for each net. Using distinct network structures could lead to further numerical savings. Furthermore, we ignore a possible third level-effect arising if the network structures could be linked to the level.

A further research area concerns the heuristic of the multilevel algorithm. Like the heuristic for the batch-sizes, it would be desirable to have an algorithm to obtain an optimal amount of training steps for each net.

Finally, it may be possible to further reduce the complexity by using more than just one path to simulate the training data. To achieve that, we must ensure a better understanding of the introduced level-effect parameter.

References

- Francis Bach and Eric Moulines. Non-strongly-convex smooth stochastic approximation with convergence rate $o(1/n)$. *arXiv preprint arXiv:1306.2119*, 2013.
- Andrew R Barron. Approximation and estimation bounds for artificial neural networks. *Machine learning*, 14(1):115–133, 1994.
- Christan Beck, Arnulf Jentzen, and Benno Kuckuck. Full error analysis for the training of deep neural networks. *arXiv preprint arXiv:1910.00121*, 2019.
- Christian Beck, Sebastian Becker, Philipp Grohs, Nor Jaafari, and Arnulf Jentzen. Solving stochastic differential equations and kolmogorov equations by means of deep learning. *arXiv preprint arXiv:1806.00421*, 2018.
- Bernard Bercu and Jean-Claude Fort. Generic stochastic gradient methods. *Wiley Encyclopedia of Operations Research and Management Science*, pages 1–8, 2011.
- Julius Berner, Philipp Grohs, and Arnulf Jentzen. 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 Journal on Mathematics of Data Science*, 2(3):631–657, 2020.

- Ngoc Huy Chau, Éric Moulines, Miklos Rásonyi, Sotirios Sabanis, and Ying Zhang. On stochastic gradient langevin dynamics with dependent data streams: the fully non-convex case. *arXiv preprint arXiv:1905.13142*, 2019.
- Benjamin Fehrman, Benjamin Gess, and Arnulf Jentzen. Convergence rates for the stochastic gradient descent method for non-convex objective functions. *Journal of Machine Learning Research*, 21, 2020.
- Sara A Geer and Sara van de Geer. *Empirical Processes in M-estimation*, volume 6. Cambridge university press, 2000.
- Thomas Gerstner and Michael Griebel. Numerical integration using sparse grids. *Numerical algorithms*, 18(3-4):209, 1998.
- M. Giles. <http://people.maths.ox.ac.uk/gilesm/acta/>.
- Michael Giles, Kristian Debrabant, and Andreas Rößler. Numerical analysis of multilevel monte carlo path simulation using the milstein discretisation. *arXiv preprint arXiv:1302.4676*, 2013.
- Michael B Giles. Multilevel monte carlo path simulation. *Operations Research*, 56(3):607–617, 2008a.
- Mike Giles. Improved multilevel monte carlo convergence using the milstein scheme. In *Monte Carlo and quasi-Monte Carlo methods 2006*, pages 343–358. Springer, 2008b.
- P. Glasserman. *Monte Carlo Methods in financial Engeineering*. Springer, 2003. vol. 53 of Stochastic Modelling and Applied Probability.
- László Györfi, Michael Kohler, Adam Krzyzak, and Harro Walk. *A distribution-free theory of nonparametric regression*. Springer Science & Business Media, 2006.
- Catherine F Higham and Desmond J Higham. Deep learning: An introduction for applied mathematicians. *SIAM Review*, 61(4):860–891, 2019.
- Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.
- Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5):551–560, 1990.
- J.C. Hull. *Options, Futures and other Derivatives*. Prentice Hall, 2008. 7th edition.
- Arnulf Jentzen, Benno Kuckuck, Ariel Neufeld, and Philippe von Wurstemberger. Strong error analysis for stochastic gradient descent optimization algorithms. *IMA Journal of Numerical Analysis*, 41(1):455–492, 2021.

Peter Eris Kloeden, Eckhard Platen, and Henri Schurz. *Numerical solution of SDE through computer experiments*. Springer Science & Business Media, 2012.

Tomaso Poggio and Christian R Shelton. On the mathematical foundations of learning. *American Mathematical Society*, 39(1):1–49, 2002.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

The appendix's proceeding will be as follows: First, we give a definition of a neural network and have a closer look at the mathematical background of the error sources (E1) to (E5). Then, we present some numerical results aiming to support some of the discussed assumptions. Finally, we will provide the missing proof.

Appendix A. Mathematical background

In this subsection we will give some mathematical background and references. For this, we first give a short definition of neural networks used in this work. Then, we discuss the error sources (E1) to (E5). Finally, we present an example of an *level estimator*' code.

A.1 Neural network definition and optimization

Let $\mathcal{L}_d : \mathbb{R}^d \rightarrow \mathbb{R}^d$ be the function which satisfies for every $y = (y_1, y_2, \dots, y_d) \in \mathbb{R}^d$ that

$$\mathcal{L}_d(y) = \left(\frac{\exp(y_1)}{\exp(y_1) + 1}, \frac{\exp(y_2)}{\exp(y_2) + 1}, \dots, \frac{\exp(y_d)}{\exp(y_d) + 1} \right),$$

for every $k, l \in \mathbb{N}, v \in \mathbb{N}_0, \theta = (\theta_1, \dots, \theta_\nu) \in \mathbb{R}^\nu$ with $v + l(k + 1) \leq \nu$, let $A_{k,l}^{\theta,v} : \mathbb{R}^k \rightarrow \mathbb{R}^l$ be the function which satisfies for every $x = (x_1, \dots, x_k) \in \mathbb{R}^k$ that

$$A_{k,l}^{\theta,v}(x) = \begin{pmatrix} \theta_{v+1} & \theta_{v+2} & \dots & \theta_{v+k} \\ \theta_{v+k+1} & \theta_{v+k+2} & \dots & \theta_{v+2k} \\ \theta_{v+2k+1} & \theta_{v+2k+2} & \dots & \theta_{v+3k} \\ \vdots & \vdots & \vdots & \vdots \\ \theta_{v+(l-1)k+1} & \theta_{v+(l-1)k+2} & \dots & \theta_{v+lk} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_k \end{pmatrix} + \begin{pmatrix} \theta_{v+kl+1} \\ \theta_{v+kl+2} \\ \theta_{v+kl+3} \\ \vdots \\ \theta_{v+kl+l} \end{pmatrix},$$

let $s \in \{3, 4, 5, 6, \dots\}$, assume that $(s-1)d(d+1) + d + 1 \leq \nu$ and let $\mathcal{N}_{\nu,\theta} : \mathbb{R}^d \rightarrow \mathbb{R}$ be the function which satisfies for every $y \in \mathbb{R}^d$ that

$$\mathcal{N}_{\nu,\theta}(y) = \left(A_{d,1}^{\theta,(s-1)d(d+1)} \circ \mathcal{L}_d \circ A_{d,d}^{\theta,(s-2)d(d+1)} \circ \dots \circ \mathcal{L}_d \circ A_{d,1}^{\theta,d(d+1)} \circ \mathcal{L}_d A_{d,d}^{\theta,0} \right)(y).$$

The function $\mathcal{N}_{\nu,\theta}$ describes an artificial neural network with $s + 1$ layers and standard logistic functions as activation functions. Fixing the network structure ν and assuming to know the best fit weights $\Theta \in \mathbb{R}^\nu$ leads to the approximation error (E2):

$$\|\bar{P}(y) - \mathcal{N}_{\nu,\Theta}(y)\|_{L^p}. \quad (\text{A.1})$$

For an overview over the approximation error, we refer to Higham and Higham (2019) and to Barron (1994); Hornik (1991); Hornik et al. (1989, 1990) for further studies.

Now, consider a neural net $\mathcal{N}_{\nu,\theta}$, as defined in (A.1), but without knowing the best-fit-weights Θ . Aiming to train the network such that it approximates \bar{P} , we are interested in minimizing the loss function

$$\mathbb{L} : \theta \mapsto \|\bar{P}(y) - \mathcal{N}_{\nu,\theta}(y)\|_{L^p}. \quad (\text{A.2})$$

The Taylor series expansion gives

$$\mathbb{L}(\theta + \Delta\theta) = \mathbb{L}(\theta) + \sum_{r=1}^{\nu} \frac{\partial \mathbb{L}(\theta)}{\theta_r} \Delta\theta,$$

where $\partial \mathbb{L}(\theta)/\theta_r$ denotes the partial derivative of the loss function with respect to the r -th parameter. Generally, only being an approximation for a small step $\Delta\theta$, we limit the step in that direction by η , leading to series of weights

$$\theta_{i+1} := \theta_i - \eta \nabla \mathbb{L}(\theta_i), \tag{A.3}$$

for $i = 0, \dots, K - 1$ and with the so-called learning rate η . We call an iteration of this form a training step. All in all this leads to the optimization errors (e4) or (E4).

We complete the neural network training definition by defining the following series of functions

$$\mathcal{N}_{\nu, \theta_i, P, \mathbb{L}} : \mathbb{R}^d \rightarrow \mathbb{R}, \tag{A.4}$$

with $i = 1, \dots, K$, initial weights θ_0 and loss function \mathbb{L} , which is evaluated with training data P .

In this work, we use stochastic gradient descent optimization and backpropagation, see e.g., Higham and Higham (2019), for an overview. For studies on the optimization error, see, e.g., Bach and Moulines (2013); Bercu and Fort (2011); Chau et al. (2019); Fehrman et al. (2020); Jentzen et al. (2021).

A.2 Discretization error

We shortly introduce the parameters α and β . The (weak) convergence, for a fixed parameter vector $y \in Y$, is defined by

$$\mathbb{E}[P - P_h] \leq ch^\alpha, \tag{A.5}$$

with a positive constant c . It is well known, provided certain assumptions are satisfied that both the Euler scheme and the Milstein scheme (1.2) converge with $\alpha = 1$ for Lipschitz-continuous payoff P only depending on the time of maturity, see, e.g., the monograph Kloeden et al. (2012). The second parameter β is used for bounding the estimators' variance, as explained in theorem 2.1. This parameter is, e.g., the focus of the article Giles et al. (2013). The authors prove different constants for β for a different type of option under certain SDE conditions.

A.3 Generalization error

This section will give some background for a more in-depth understanding of the loss functions' discretization error. As explained above, we discretize the loss function (A.2) in each training step by a finite set $y_i \in Y$, with $i = 1, \dots, M$.

The usage of the approximated loss function leads to the generalization error (e5). For the multilevel approach, we discretize the loss function at randomly selected inputs $y_i \in Y$.

Consider the loss function (A.2). First, we transform both arising integrals to the unit cube. For simplicity, we assume the training set Y to be of the form $[a, b]^d$, without loss of generality. For the transformation of the norm-part, we will use the linear transformation $\mathcal{U} : [a, b]^d \rightarrow [0, 1]^d$. For the expectation-integral, we will use the inversion method, see e.g. Glasserman (2003), e.g. for the cumulative standard normal distribution function $\Phi(x) : \mathbb{R}^H \rightarrow [0, 1]^H$, leading to the following lemma.

Lemma A.1 *Let $Y = [a, b]^d$ and consider P_h to be multivariate standard normal distributed for each $y \in Y$. Then, by using the linear transformation $\mathcal{U} : Y \rightarrow [0, 1]^d =: U$ and the cumulative standard normal distribution function $\Phi(x) : \mathbb{R}^H \rightarrow [0, 1]^H$, we have*

$$\begin{aligned} \left\| \mathbb{E}[P_h(y)] - \mathcal{N}_{\nu, \theta, \mathbb{E}[P_h], \mathbb{L}_Y}(y) \right\|_{L^p(Y)} = \\ (b-a)^{-d/p} \left\| \mathbb{E}^*[\hat{P}_h(u)] - \hat{\mathcal{N}}_{\nu, \theta, \mathbb{E}[\hat{P}_h], \mathbb{L}_U}(u) \right\|_{L^p(U)}, \end{aligned}$$

with $\hat{P}_h(u) : u \mapsto P_h(\mathcal{U}^{-1}(u))$, $\hat{\mathcal{N}}_{\nu, \theta, \mathbb{E}[\hat{P}_h], \mathbb{L}_U} : u \mapsto \mathcal{N}_{\nu, \theta, \mathbb{E}^*[P^h], \mathbb{L}_Y}(\mathcal{U}^{-1}(u))$, and the expectation \mathbb{E}^* defined on the unit cube.

Proof

$$\begin{aligned} & \left\| \mathbb{E}[P_h(y)] - \mathcal{N}_{\nu, \theta, \mathbb{E}[P_h], \mathbb{L}_Y}(y) \right\|_{L^p} \\ &= \left(\int_{[a, b]^d} \left| \mathbb{E}[P_h(y)] - \mathcal{N}_{\nu, \theta, \mathbb{E}[P^h], \mathbb{L}_Y}(y) \right|^p dy \right)^{1/p} \\ &= \left((b-a)^d \int_{[0, 1]^d} \left| \mathbb{E}[P_h(\mathcal{U}^{-1}(u))] - \mathcal{N}_{\nu, \theta, \mathbb{E}[P_h], \mathbb{L}_Y}(\mathcal{U}^{-1}(u)) \right|^p du \right)^{1/p} \\ &= (b-a)^{d/p} \left\| \mathbb{E}^*[\hat{P}_h(u)] - \hat{\mathcal{N}}_{\nu, \theta, \mathbb{E}[\hat{P}_h], \mathbb{L}_U}(u) \right\|_{L^p}. \end{aligned}$$

■

I.e. we can study the neural net training on the unit cube.

Corollary A.1 *Consider $Y = [0, 1]^d =: U$ and P_h to be uniformly distributed. We have the following inequality:*

$$\left\| \mathbb{E}[P_h(u)] - \mathcal{N}(u) \right\|_{L^1}^2 \leq \left\| \mathbb{E}[P_h(u)] - \mathcal{N}(u) \right\|_{L^2}^2 \quad (\text{A.6})$$

and the difference is given by

$$\int_U \mathbb{V} \left[\mathbb{E}[P_h(u)] - \mathcal{N}(u) \right] du. \quad (\text{A.7})$$

Proof (A.6) is given through Hölders' inequality. (A.7) is given by

$$\begin{aligned} & \int_U \mathbb{V}^* \left[\mathbb{E}[\hat{P}_h(u)] - \mathcal{N}(u) \right] du \\ &= \int_U \left(\mathbb{E}[P_h(u)] - \hat{\mathcal{N}}(u) \right)^2 du - \left(\int_U \mathbb{E}[P^h(u)] - \mathcal{N}(u) du \right)^2. \end{aligned}$$

■

In practice, we will be interested in approximating the right-hand side of (A.6). The standard Monte Carlo approximation leads to the following corollary.

Corollary A.2 *For $p = 2$, the right-hand side of (A.6) can be estimated by the Monte Carlo estimator*

$$\frac{1}{M} \sum_{i=1}^M \left(\left(\frac{1}{N} \sum_{j=1}^N P(u_{i,j}^{(2)}, u_i^{(1)}) \right) - \mathcal{N}(u_i^{(1)}) \right)^2,$$

with $u_i^{(1)} \in [0, 1]^d$ for $i = 1, \dots, M$ and $u_{i,j}^{(2)} \in [0, 1]^H$ for $j = 1, \dots, N$ and $i = 1, \dots, M$ both i.i.d. uniformly distributed. This estimator leads to the generalization error.

For, e.g., $d = H = 1$, this estimator needs $M + MN$ random samples for the training data for each training step. The introduced Monte Carlo estimator relies on the choice of both M and N . The attempt to avoid this choice leads to the following considerations.

Lemma A.2 *For the right-hand side of (A.6) the following inequality holds:*

$$\|\mathbb{E}[P_h(u)] - \mathcal{N}(u)\|_{L^2}^2 \leq \int_{[0,1]^{d+H}} (P_h(u) - \mathcal{N}(u))^2 du \quad (\text{A.8})$$

and the difference is given by

$$\|\mathbb{V}[P_h(u)]\|_{L^1}. \quad (\text{A.9})$$

Proof Let $u^{(1)} \in [0, 1]^d$ and $u^{(2)} \in [0, 1]^H$. Then,

$$\begin{aligned} & \|\mathbb{E}[P_h(u)] - \mathcal{N}(u)\|_{L^2}^2 \\ &= \int_{[0,1]^d} \left(\int_{[0,1]^H} P^h(u^{(2)}, u^{(1)}) du^{(2)} - \mathcal{N}(u^{(1)}) \right)^2 du^{(1)} \\ &\leq \int_{[0,1]^d} \int_{[0,1]^H} \left(P^h(u^{(2)}, u^{(1)}) - \mathcal{N}(u^{(1)}) \right)^2 du^{(2)} du^{(1)} \end{aligned}$$

The difference follows from above with

$$\begin{aligned} &= \int_{[0,1]^d} \int_{[0,1]^H} \left(P^h(u^{(2)}, u^{(1)}) - \mathcal{N}(u^{(1)}) \right)^2 du^{(2)} du^{(1)} \\ &- \int_{[0,1]^d} \left(\int_{[0,1]^H} P^h(u^{(2)}, u^{(1)}) du^{(2)} - \int_{[0,1]^H} \mathcal{N}(u^{(1)}) du^{(2)} \right)^2 du^{(1)} \\ &= \int_{[0,1]^d} \mathbb{V} \left[P^h(u) - \mathcal{N}(u) \right] du. \end{aligned}$$

■

The Monte Carlo approximation of the right-hand side of (A.8) leads to the following corollary.

Corollary A.3 *The loss function $\mathbb{L}_{2^*}^2$ defined on the right-hand side of (A.8) can be estimated by the Monte Carlo estimator*

$$\widehat{L}_M = \frac{1}{M} \sum_{i=1}^M \left([\widehat{P}(u_{2,i}, u_{1,i})] - \mathcal{N}_{\nu, \theta, \widehat{P}}(u_{1,i}) \right)^2,$$

with $u_{1,i} \in [0, 1]^d$ and $u_{2,i} \in [0, 1]^H$, for $i = 1, \dots, M$ independent uniformly distributed.

For, e.g., $d = H = 1$, this leads to $2M$ needed random numbers per iteration step.

Furthermore, from, e.g., proposition 2.2 of Beck et al. (2019), we know that under certain assumptions, there exists a neural network $N : [0, 1]^d \rightarrow \mathbb{R}$ and a unique continuous function f such that

$$\inf_{f \in \mathcal{C}([0, 1]^d, \mathbb{R})} \int_{[0, 1]^{d+H}} \left(P^h(u) - f(u) \right)^2 du = \int_{[0, 1]^{d+H}} \left(P^h(u) - N(u) \right)^2 du \quad (\text{A.10})$$

and it holds for every $u \in [0, 1]^d$ that

$$N(u) = \int_{[0, 1]^H} P^h(u) du = \mathbb{E} \left[P^h(u) \right]. \quad (\text{A.11})$$

Remark A.4 *In other words, the function minimizing the right-hand side of (A.8) could as well minimize the right-hand side of (A.6). I.e., for the training process of a neural network, both Monte Carlo estimators of theorem A.3 and theorem A.3 lead to unbiased results. With respect to the error assumptions, this justifies (A3). However, one should keep in mind that the Monte Carlo estimator of theorem A.2 could be more efficient, even though, depending on the choice of N and M . The apparent reason for this is that even though the norm choice does not apply a bias, e.g., in (A3), its respective Monte Carlo estimators' quality will affect the loss functions' error, e.g., in (A5 i).*

For further studies on the generalization error, see, e.g., Berner et al. (2020); Poggio and Shelton (2002); Györfi et al. (2006); Shalev-Shwartz and Ben-David (2014); Geer and van de Geer (2000).

We will study a single-level training process for a European option using a geometric Brownian motion in the second subsection. While neglecting the training step-size, we will focus on the numerical properties of the loss functions' error.

batch-size	mean error	standard deviation	mean reduction
1k	0.0263	0.0102	
4k	0.0139	0.0042	0.5285
16k	0.0071	0.0022	0.5096
64k	0.00321	0.0010	0.4500

Table A.1: Mean and standard deviation of the L^∞ -error with respect to increasing batch-size. Training set is $[100, 104]$ and the training is repeated 10 times. The last column describes the error reduction.

A.4 Numerical results: batch-size convergence and variance reduction

Even though we know that the standard Monte Carlo simulation converges with order $1/2$, it is not entirely clear whether this property applies to the neural network training, as assumed, e.g., in (A5 i). Hence, we present some results for increasing batch-sizes for the GBM and a European call option for a small fixed training set, e.g., $s_0 \in [100, 104]$. Using this small training set, the closed solution as training data, and a vast amount of training steps, we suppose the batch-size to be the crucial factor for the error. A first result can be found in table A.1. If not further specified, we use the same network and training parameters as in the examples above.

We observe a convergence order of approximately $1/2$ for an increasing batch-size with respect to the mean of the sampled L^∞ error.

As well known, our specific choice of the payoff leads to a particular standard deviation of the standard Monte Carlo estimator. Furthermore, it is well known that this standard deviation is responsible for the Monte Carlo error. It could be reduced by, e.g., variance reduction techniques, see, e.g., Glasserman (2003). For our example, we will use some importance sampling by only sampling paths that stay above the strike price. Since we know that the variance is increasing monotonously for this specific example, we evaluated the variance reduction factor to be included in the interval $[0.34179, 0.39067]$ for our training set's initial values. Now, by training a neural network with training data computed with the variance reduced Monte Carlo estimator, we obtain results as can be seen in table A.2.

batch-size	mean error standard net	mean error OSS net	mean reduction
1k	0.0263	0.0106	0.3869

Table A.2: Mean of the L^∞ -error for both the neural networks. The calculated mean reduction is given in the last column. The training was computed on the interval $[100, 104]$ and repeated 10 times.

Comparing the results' mean L^∞ -error, we see that the L^∞ -error reduction factor is included in the above interval of variance reduction ratios. This property may justify a relation between the needed batch-size and the estimators' variance as, e.g., used in (A5 i).

A.5 Python code example

Code of the *level estimator* $P_1 - P_0$.

Listing 1: Training data for level estimator

```

1 def sde_body_p1_p0(idx, s_coarse, s_fine, samples):
2     z1 = tf.random_normal(shape=(samples, batch_size_p1_p0, d),
3                           stddev=1., dtype=dtype)
4     z2 = tf.random_normal(shape=(samples, batch_size_p1_p0, d),
5                           stddev=1., dtype=dtype)
6     z=(z1+z2)/np.sqrt(2.)
7     h_fine=1
8     h_coarse=1/2
9     s_fine=s_fine + mu *s_fine * h_fine +sigma * s_fine ...
10    *np.sqrt(h_fine)*z1 + 0.5 *sigma *s_fine *sigma * ...
11    ((np.sqrt(h_fine)*z1)**2-h_fine)
12    s_fine=s_fine + mu *s_fine * h_fine +sigma * s_fine ...
13    *np.sqrt(h_fine)*z2 + 0.5 *sigma *s_fine *sigma * ...
14    ((np.sqrt(h_fine)*z2)**2-h_fine)
15    s_coarse=s_coarse + mu *s_coarse * h_coarse +sigma * s_coarse ...
16    *np.sqrt(h_coarse)*z + 0.5 *sigma *s_coarse *sigma * ...
17    ((np.sqrt(h_coarse)*z)**2-h_coarse)
18    return tf.add(idx, 1), s_coarse, s_fine

```

Appendix B. Proof

This subsection provides the missing proof.

Proof [Proof of lemma 2.2]

We derive an upper bound of $\frac{1}{2}\epsilon^2$ on the square of the expectation of the bias and an upper bound of $\frac{1}{2}\epsilon^2$ on the variance, which together with (2.3) give an ϵ^2 upper bound on (2.2).

Setting $K = 32c_3^2\epsilon^{-2}$ and $M = \sqrt{32}c_2\epsilon^{-1}$ together with (e1) $\leq \frac{1}{\sqrt{32}}\epsilon$, (e2) $\leq \frac{1}{\sqrt{32}}\epsilon$ and (e3) = 0 leads to

$$\left(\mathbb{E} \left[\left\| \mathbb{E}[P(y)] - \mathcal{N}_{\nu, \theta, \hat{P}, \hat{L}}(y) \right\|_{L^1} \right]\right)^2 \leq \left(\frac{1}{\sqrt{32}}\epsilon + \frac{1}{\sqrt{32}}\epsilon + 0 + \frac{1}{\sqrt{32}}\epsilon + \frac{1}{\sqrt{32}}\epsilon \right)^2 = \frac{1}{2}\epsilon^2,$$

Furthermore, the above choices for K and M together with the assumptions $\mathbb{V}[(e5)] \leq c_2^2M^{-2}$, and $\mathbb{V}[(e4)] \leq c_3^2K^{-1}$ lead to

$$\mathbb{V} \left[\left\| \mathbb{E}[P(y)] - \mathcal{N}_{\nu, \theta, \hat{P}, \hat{L}}(y) \right\|_{L^1} \right] \leq \frac{1}{32}\epsilon^2 + \frac{1}{32}\epsilon^2 \leq \frac{1}{2}\epsilon^2.$$

The assumption on the complexity is bounded together with the choices for K and M lead to $C_N \leq c_432c_3^2\sqrt{32}c_2\epsilon^{-5}$. Hence, setting $c_5 = c_432c_3^2\sqrt{32}c_2$ completes the proof. \blacksquare

Proof [Proof of lemma 2.1]

The proof is analogue to the proof of lemma 2.2, but we choose $K = 32c_3^2\epsilon^{-2}$, $M = 32c_2^2\epsilon^{-2}$ and

$$h = \frac{\epsilon}{c_1\sqrt{32}},$$

which again leads to the ϵ^2 bound on (2.2). The computational complexity is bounded by

$$C_{\mathcal{N}} \leq c_4 K M h^{-1} \leq c_4 32^2 c_3^2 c_2^2 c_1 \sqrt{32} \epsilon^{-5}.$$

Hence, setting $c_5 = c_4 32^2 c_3^2 c_2^2 c_1 \sqrt{32}$ completes the proof. \blacksquare

To shorten the proof of theorem 2.1 and simplify its presentation, we will use generalized K_l and M_l from the second case onwards, which we apply for all the remaining cases. Nevertheless, this leads to bad estimates for the $\log \epsilon$ terms. Hence, if interested in a particular case, we recommend a more in-depth examination of the first case's proof.

Proof [Proof of theorem 2.1]

For each case, we choose specific K_l and M_l such that these bound (2.9). We use the decomposition (2.6) and show that both satisfy an $\frac{1}{2}\epsilon^2$ bound. Finally, using the assumptions (2.8), we show that the chosen parameters can bound the overall complexity

$$C_{\mathcal{N}} \leq \sum_{l=0}^L c_4 h_l^{-1} M_l K_l, \quad (\text{B.1})$$

with the stated complexity for the specific case. Let $h_l = H^{-l}T$, for $l = 0, \dots, L$ be different step-widths with $H > 1$. We start by choosing L to be

$$L = \left\lceil \frac{\log(\sqrt{32}c_1 T^\alpha \epsilon^{-1})}{\alpha \log H} \right\rceil,$$

so that

$$\frac{1}{\sqrt{32}} H^{-\alpha} \epsilon < c_1 h_L^\alpha \leq \frac{1}{\sqrt{32}} \epsilon. \quad (\text{B.2})$$

Let θ be positive, then

$$\sum_{l=0}^L h_l^\theta = \sum_{l=0}^L (H^{-l}T)^\theta = T^\theta \sum_{l=0}^L (H^{-\theta})^l < \frac{T^\theta}{1 - H^{-\theta}}.$$

On the contrary, for negative exponents we have

$$\sum_{l=0}^L h_l^{-\theta} = h_L^{-\theta} \sum_{l=0}^L (H^\theta)^{-l} < \frac{H^\theta}{H^\theta - 1} h_L^{-\theta},$$

which holds due to (B.2) and with

$$h_L^{-\theta} < H^\theta \left(\frac{\epsilon}{\sqrt{32}c_1} \right)^{-\theta/\alpha},$$

we obtain

$$\sum_{l=0}^L h_l^{-\theta} < \frac{H^{2\theta}}{H^\theta - 1} \left(\sqrt{32}c_1 \right)^{\theta/\alpha} \epsilon^{-\theta/\alpha}.$$

For a simplified presentation, we denote

$$g : \theta \mapsto \begin{cases} \frac{T^\theta}{1-H^{-\theta}} & \text{for } \theta > 0, \\ \frac{H^{2\theta}}{H^\theta - 1} \left(\sqrt{32}c_1 \right)^{\theta/\alpha}, & \text{for } \theta < 0. \end{cases} \quad (\text{B.3})$$

I.e., we will have the following inequalities

$$\sum_{l=0}^L h_l^\theta \leq \begin{cases} g(\theta) & \text{for } \theta > 0, \\ L + 1 & \text{for } \theta = 0, \\ g(\theta)\epsilon^{-\theta/\alpha}, & \text{for } \theta < 0. \end{cases} \quad (\text{B.4})$$

Now, let us consider the different parameter values.

(a) If $\eta = 0.5, \gamma = 2, \beta = 2$ and $\alpha = 1$ we set

$$K_l = \lceil 32\epsilon^{-2}c_2^2h_l^{1.5}g(1.25)^2 \rceil \quad (\text{B.5})$$

and

$$M_l = \lceil 32\epsilon^{-1}c_3^2c_6^2h_l^{3/4} \rceil, \quad (\text{B.6})$$

with

$$c_6 = \max \left\{ g(2/8)(32c_2^2g(1.25)^2)^{-1/4}, T(1 - H^{-1/2})^{-2} \left(\sqrt{32c_2^2g(1.25)^2} \right)^{-2} \right\},$$

Using (A4 i), (B.3) and (B.5), we obtain

$$\begin{aligned} \mathbb{E} \left[\sum_{l=0}^L \left\| \mathcal{N}_{\nu, \Theta^l, \hat{Y}_l, \mathbb{L}}(y) - \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_l, \mathbb{L}}(y) \right\|_{L^1} \right] &\leq \sum_{l=0}^L c_2 h_l^2 K_l^{-1/2} \\ &\leq c_2 \frac{1}{\sqrt{32c_2^2g(1.25)^2}} \epsilon \sum_{l=0}^L h_l^{1.25} \\ &\leq \frac{1}{\sqrt{32}} \epsilon. \end{aligned}$$

Using (A5 i), (B.3) and (B.6), we obtain

$$\begin{aligned}
 \mathbb{E} \left[\sum_{l=0}^L \left\| \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_{l, \mathbb{L}}}(y) - \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_{l, \mathbb{L}_{M_l}}}(y) \right\|_{L^1} \right] &\leq \sum_{l=0}^L c_3 h_l \rho_l^{1/2} M_l^{-1/2} \\
 &\leq \sum_{l=0}^L c_3 h_l K_l^{-1/4} \frac{1}{\sqrt{32 \epsilon^{-1} c_3^2 c_6^2 h_l^{3/4}}} \\
 &\leq \frac{1}{\sqrt{32 c_6^2}} \epsilon \frac{1}{(32 c_2^2 (g(1.25))^2)^{1/4}} \sum_{l=0}^L h_l^{2/8} \\
 &\leq \frac{1}{\sqrt{32 c_6^2}} \epsilon \frac{g(2/8)}{c_6 (32 c_2^2 (g(1.25))^2)^{1/4}} \\
 &\leq \frac{1}{\sqrt{32}} \epsilon.
 \end{aligned}$$

Hence, using these results together with assumptions (A1)-(A3) leads to

$$\left(\mathbb{E} \left[\left\| \mathbb{E}[P(y)] - \hat{\mathcal{N}}(y) \right\|_{L^1} \right] \right)^2 \leq \left(\frac{1}{\sqrt{32}} \epsilon + \frac{1}{\sqrt{32}} \epsilon + \frac{1}{\sqrt{32}} \epsilon + \frac{1}{\sqrt{32}} \epsilon \right)^2 = \frac{1}{2} \epsilon^2. \quad (\text{B.7})$$

I.e., we obtain the searched $\frac{1}{2} \epsilon^2$ error bound on the square of the bias. Now, using (A4 ii), (B.5) and (B.3), we obtain

$$\begin{aligned}
 \mathbb{V} \left[\sum_{l=0}^L \left\| \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_{l, \mathbb{L}}}(y) - \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_{l, \mathbb{L}}}(y) \right\|_{L^1} \right] &\leq \sum_{l=0}^L c_2^2 h_l^4 K_l^{-1} \\
 &\leq \sum_{l=0}^L c_2^2 h_l^4 \frac{1}{32 \epsilon^{-2} c_2^2 h_l^{1.5} (g(1.25))^2} \\
 &= \frac{1}{32} \epsilon^2 \sum_{l=0}^L h_l^4 \frac{1}{h_l^{1.5} (g(1.25))^2} \\
 &\leq \frac{1}{32} \epsilon^2 \frac{g(2.5)}{g(1.25)^2} \\
 &\leq \frac{1}{4} \epsilon^2,
 \end{aligned}$$

which holds, since we have $g(2.5) < g(1.25)^2$. Using (A5 ii), (B.6) and (B.3), we obtain

$$\begin{aligned}
 \mathbb{V} \left[\sum_{l=0}^L \left\| \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_{l, \mathbb{L}}}(y) - \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_{l, \mathbb{L}_{M_l}}}(y) \right\|_{L^1} \right] &\leq c_3^2 h_l^2 \rho_l^1 M_l^{-1} \\
 &\leq \epsilon^2 \frac{1}{\sqrt{32 c_2^2 (g(1.25))^2}} \frac{1}{32 c_6^2} \sum_{l=0}^L h_l^{1/2} \\
 &\leq \frac{1}{32 c_6^2} \epsilon^2 \frac{g(1/2)}{\sqrt{32 c_2^2 (g(1.25))^2}} \\
 &\leq \frac{1}{4} \epsilon^2.
 \end{aligned}$$

Hence, we obtain an $\frac{1}{2}\epsilon^2$ upper bound on the variance and the together with the bound on the bias the required ϵ^2 bound on (2.9). Finally, we study the computational cost for the chosen parameters. Using (B.1), (B.3), (B.4), (B.5), (B.6) and the following upper bounds on K_l and M_l

$$K_l < 32\epsilon^{-2}c_2^2h_l^{1.5}(g(1.25))^2 + 1, \quad 32\epsilon^{-1}c_3^2c_6^2h_l^{3/4} + 1,$$

we obtain

$$\begin{aligned} C_{\mathcal{N}} &\leq \sum_{l=0}^L c_4 h_l^{-1} M_l K_l \\ &\leq \sum_{l=0}^L c_4 h_l^{-1} (32\epsilon^{-2}c_2^2h_l^{1.5}(g(1.25))^2 + 1) (32\epsilon^{-1}c_3^2c_6^2h_l^{3/4} + 1) \\ &= c_4 \sum_{l=0}^L h_l^{-1} (c_7\epsilon^{-2}(h_l)^{1.5} + 1) (c_8\epsilon^{-1}h_l^{0.75} + 1) \\ &\leq c_4c_7c_8\epsilon^{-3}g(1.25) + c_4c_7\epsilon^{-2}g(0.5) + c_4c_8\epsilon^{-1}g(-0.25)\epsilon^{-0.25/\alpha} + c_4g(-1)\epsilon^{-1/\alpha}, \end{aligned}$$

with $c_7 = 32c_2^2(g(1.25))^2$ and $c_8 = 32c_3^2c_6^2$. Hence, for this case we obtain the required complexity bound

$$C_{\mathcal{N}} \leq c_5\epsilon^{-3},$$

with $c_5 = c_4c_7c_8g(1.25) + c_4c_7g(0.5) + c_4c_8g(-0.25) + c_4g(-1)$, which completes the proof. Now, as mentioned above, instead of individual choices for K_l and M_l for the remaining cases, we will choose them in a more generic way, i.e. let

$$K_l = \left\lceil 32\epsilon^{-2}c_2^2h_l^{2\gamma\alpha}(L+1)^2 \right\rceil \tag{B.8}$$

and

$$M_l = \left\lceil 32\epsilon^{-2+2\eta}c_3^2h_l^{\bar{\beta}}(L+1)^2 \right\rceil. \tag{B.9}$$

We will use (B.8) and (B.9) for each of the remaining cases. As we will see in the following, these choices satisfy the required error bound on the MSE for each case. Hence, for each case, we will only have to study the individual computational cost. Using (A4 i) and (B.8), we obtain

$$\begin{aligned} \mathbb{E} \left[\sum_{l=0}^L \left\| \mathcal{N}_{\nu, \Theta^l, \hat{Y}_l, \mathbb{L}}(y) - \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_l, \mathbb{L}}(y) \right\|_{L^1} \right] &\leq \sum_{l=0}^L c_2 h_l^{\alpha\gamma} K_l^{-1/2} \\ &\leq \frac{1}{\sqrt{32}}\epsilon. \end{aligned} \tag{B.10}$$

Using (A5 i) and (B.9), we obtain

$$\begin{aligned}
 \mathbb{E} \left[\sum_{l=0}^L \left\| \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_{l, \mathbb{L}}}(y) - \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_{l, \mathbb{L}_{M_l}}}(y) \right\|_{L^1} \right] &\leq \sum_{l=0}^L c_3 h_l^{\beta/2} \rho_l^\eta M_l^{-1/2} \\
 &\leq \sum_{l=0}^L c_3 h_l^{\beta/2} K^{-\eta/2} M_l^{-1/2} \\
 &\leq \frac{1}{\sqrt{32}(L+1)} \epsilon^{\eta+1-\eta} \sum_{l=0}^L h_l^{\beta/2} \frac{1}{h_l^{\gamma\alpha\eta} h_l^{\beta/2}} \\
 &\leq \frac{1}{\sqrt{32}} \epsilon, \tag{B.11}
 \end{aligned}$$

where the last inequality holds by choosing $\bar{\beta} = \beta - 2\alpha\gamma\eta$. For the variance analogue formulas hold, as we will see in the following calculations. Using (A4 ii) and (B.8), we obtain

$$\begin{aligned}
 \mathbb{V} \left[\sum_{l=0}^L \left\| \mathcal{N}_{\nu, \Theta^l, \hat{Y}_{l, \mathbb{L}}}(y) - \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_{l, \mathbb{L}}}(y) \right\|_{L^1} \right] &\leq \sum_{l=0}^L c_2^2 h_l^{2\alpha\gamma} K_l^{-1} \\
 &\leq \sum_{l=0}^L \frac{1}{32\epsilon^{-2}(L+1)^2} \\
 &\leq \frac{1}{4} \epsilon^2. \tag{B.12}
 \end{aligned}$$

Using (A5 ii) and (B.9), we obtain

$$\begin{aligned}
 \mathbb{V} \left[\sum_{l=0}^L \left\| \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_{l, \mathbb{L}}}(y) - \mathcal{N}_{\nu, \theta_{K_l}^l, \hat{Y}_{l, \mathbb{L}_{M_l}}}(y) \right\|_{L^1} \right] &\leq \sum_{l=0}^L c_3^2 h_l^\beta \rho_l^{2\eta} M_l^{-1} \\
 &\leq \frac{1}{32(L+1)^2} \epsilon^{2\eta+2-2\eta} \sum_{l=0}^L h_l^\beta \frac{1}{h_l^{2\gamma\alpha\eta} h_l^{\bar{\beta}}} \\
 &\leq \frac{1}{4} \epsilon^2, \tag{B.13}
 \end{aligned}$$

again the last inequality holds by choosing $\bar{\beta} = \beta - 2\alpha\gamma\eta$.

Hence, we obtain an $\frac{1}{2}\epsilon^2$ upper bound on the variance and with (B.10) and (B.11) an $\frac{1}{2}\epsilon^2$ upper bound on the bias. Together, the required we obtain a ϵ^2 bound on the MSE (2.9).

Now, we will study the computational cost for each case. We have the following upper bounds

$$K_l < 32\epsilon^{-2} c_2^2 h_l^{2\gamma\alpha} (L+1)^2 + 1$$

and

$$M_l < 32\epsilon^{-2+2\eta} c_3^2 h_l^{\bar{\beta}} (L+1)^2 + 1.$$

Together with (B.1) this leads to

$$\begin{aligned}
 C_{\mathcal{N}} &\leq \sum_{l=0}^L c_4 h_l^{-1} \left(32\epsilon^{-2} c_2^2 h_l^{2\gamma\alpha} (L+1)^2 + 1 \right) \left(32\epsilon^{-2+2\eta} c_3^2 h_l^{\bar{\beta}} (L+1)^2 + 1 \right) \\
 &\leq c_4 c_7 c_8 (L+1)^4 \epsilon^{-4+2\eta} \sum_{l=0}^L h_l^{-1+2\gamma\alpha+\bar{\beta}} + c_4 c_7 (L+1)^2 \epsilon^{-2} \sum_{l=0}^L h_l^{-1+2\gamma\alpha} \\
 &\quad + c_4 c_8 (L+1)^2 \epsilon^{-2+2\eta} \sum_{l=0}^L h_l^{-1+\bar{\beta}} + c_4 g(-1) \epsilon^{-1/\alpha},
 \end{aligned} \tag{B.14}$$

with the constants $c_7 = 32c_2^2$ and $c_8 = 32c_3^2$. For L of (B.14) we have

$$L \leq \frac{\log \epsilon^{-1}}{\alpha \log H} + \frac{\log(\sqrt{2}c_1 T^\alpha)}{\alpha \log H} + 1$$

and since $1 < \log \epsilon^{-1}$ for $\epsilon < \exp(-1)$ it follows that

$$L+1 \leq c_9 \log \epsilon^{-1}, \tag{B.15}$$

where

$$c_9 = \frac{1}{\alpha \log H} + \max \left(0, \frac{\log(\sqrt{2}c_1 T^\alpha)}{\alpha \log H} \right) + 2.$$

Let us consider the different parameter values. Again, we will use $\bar{\beta} = \beta - 2\alpha\gamma\eta$ for each case.

(b) Let $\eta = 0, \gamma = 0, \beta = 0$ and $\alpha = 1/2$. Then, $\bar{\beta} = 0$ and with (B.14) we have

$$C_{\mathcal{N}} \leq c_6 \epsilon^{-4} (L+1)^4 \sum_{l=0}^L h_l^{-1},$$

with $c_6 = c_4(c_7 c_8 + c_7 + c_8 + 1)$. Using (B.3), (B.4) and (B.15) we obtain

$$C_{\mathcal{N}} \leq c_6 \epsilon^{-6} |\log \epsilon|^4,$$

with $c_5 = c_6 c_9^4 g(-1)$. (c) Let $\eta = 0, \gamma = 0, \beta = 0$ and $\alpha = 1$. Then, $\bar{\beta} = 0$ and with (B.14) we have

$$C_{\mathcal{N}} \leq c_5 \epsilon^{-4} (L+1)^4 \sum_{l=0}^L h_l^{-1},$$

with $c_6 = c_4(c_7 c_8 + c_7 + c_8 + 1)$. Using (B.3), (B.4) and (B.15) we obtain

$$C_{\mathcal{N}} \leq c_6 \epsilon^{-5} |\log \epsilon|^4,$$

with $c_5 = c_6 c_9^4 g(-1)$.

(d) Let $\eta = 0, \gamma = 0, \beta = 1$ and $\alpha = 1$. Then, $\bar{\beta} = 1$ and with (B.14) we have

$$C_{\mathcal{N}} \leq c_6 \epsilon^{-4} (L+1)^4 \sum_{l=0}^L h_l^0,$$

with $c_6 = c_4(c_7 c_8 + c_7 + c_8 + 1)$. Using (B.3), (B.4) and (B.15) we obtain

$$C_{\mathcal{N}} \leq c_5 \epsilon^{-4} |\log \epsilon|^5,$$

with $c_5 = c_6 c_9^5$.

(e) Let $\eta = 0.25, \gamma = 1, \beta = 1$ and $\alpha = 1$. Then, $\bar{\beta} = 0.5$ and with (B.14) we have

$$C_{\mathcal{N}} \leq c_6 \epsilon^{-3.5} (L+1)^4 \sum_{l=0}^L h_l^{1.5},$$

with $c_6 = c_4(c_7 c_8 + c_7 + c_8 + 1)$. Using (B.3), (B.4) and (B.15) we obtain

$$C_{\mathcal{N}} \leq c_5 \epsilon^{-3.5} |\log \epsilon|^4,$$

with $c_5 = c_6 c_9^4 g(1.5)$. (f) Let $\eta = 0.5, \gamma = 1, \beta = 1$ and $\alpha = 1$. Then, $\bar{\beta} = 0$ and with (B.14) we have

$$C_{\mathcal{N}} \leq c_6 \epsilon^{-3} (L+1)^4 \sum_{l=0}^L h_l^1,$$

with $c_6 = c_4(c_7 c_8 + c_7 + c_8 + 1)$. Using (B.3), (B.4) and (B.15) we obtain

$$C_{\mathcal{N}} \leq c_5 \epsilon^{-3} |\log \epsilon|^4,$$

with $c_5 = c_6 c_9^4 g(1)$. ■