

Deep Learning algorithms for solving high dimensional nonlinear Backward Stochastic Differential Equations

LORENC KAPLLANI, LONG TENG

Lehrstuhl für Angewandte Mathematik und Numerische Analysis,
Fakultät für Mathematik und Naturwissenschaften,
Bergische Universität Wuppertal, Gaußstr. 20,
42119 Wuppertal, Germany

Abstract

We study deep learning-based schemes for solving high dimensional nonlinear backward stochastic differential equations (BSDEs). First we show how to improve the performances of the proposed scheme in [W. E and J. Han and A. Jentzen, Commun. Math. Stat., 5 (2017), pp.349-380] regarding computational time and stability of numerical convergence by using the advanced neural network architecture instead of the stacked deep neural networks. Furthermore, the proposed scheme in that work can be stuck in local minima, especially for a complex solution structure and longer terminal time. To solve this problem, we investigate to reformulate the problem by including local losses and exploit the Long Short Term Memory (LSTM) networks which are a type of recurrent neural networks (RNN). Finally, in order to study numerical convergence and thus illustrate the improved performances with the proposed methods, we provide numerical results for several 100-dimensional nonlinear BSDEs including a nonlinear pricing problem in finance.

Keywords *high dimension, backward stochastic differential equations, deep learning, residual neural network, recurrent neural network, nonlinear option pricing*

1 Introduction

In this work we consider the high dimensional *forward backward stochastic differential equation (FBSDE)* of the form

$$\begin{cases} dX_t &= \mu(t, X_t) dt + \sigma(t, X_t) dW_t, & X_0 = x_0, \\ -dY_t &= f(t, X_t, Y_t, Z_t) dt - Z_t dW_t, \\ Y_T &= \xi = g(X_T), \end{cases} \quad (1)$$

where $X_t, \mu \in \mathbb{R}^n$, σ is a $n \times d$ matrix, $W_t = (W_t^1, \dots, W_t^d)^\top$ is a d -dimensional Brownian motion, $f(t, X_t, Y_t, Z_t) : [0, T] \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^{m \times d} \rightarrow \mathbb{R}^m$ is the driver function and ξ is the terminal condition which depends on the final value of the forward stochastic differential equation (SDE), X_t . For $\mu = 0$ and $\sigma = 1$, namely $X_t = W_t$, one obtains a *backward stochastic differential equation (BSDE)* of the form

$$\begin{cases} -dY_t &= f(t, Y_t, Z_t) dt - Z_t dW_t, \\ Y_T &= \xi = g(W_T), \end{cases}$$

where $Y_t \in \mathbb{R}^m$ and $f(t, Y_t, Z_t) : [0, T] \times \mathbb{R}^m \times \mathbb{R}^{m \times d} \rightarrow \mathbb{R}^m$. In the sequel of this work, we investigate to solve (1) in the 100-dimensional case using the deep learning-based methods. The existence and uniqueness of the solution of (1) are proven in [Pardoux and Peng, 1990].

Developing efficient numerical algorithms for high dimensional nonlinear BSDEs has always been a big challenging task, e.g., as the dimensionality grows, the complexity of the algorithms grows exponentially. Solving high dimensional nonlinear BSDEs has a lot of practical importance in the field of physic and finance. For example, El Karoui et al. showed that the solution of a linear BSDE is in fact the pricing and hedging strategy of an option derivative in [El Karoui et al., 1997] which is the first claim of the application of BSDEs in finance. Due to the fact that many market models can be more conveniently described by the BSDEs, such as local volatility models [Labart and Lelong, 2011], stochastic volatility models [Fahim et al., 2011], jump-diffusion models [Eyraud-Loisel, 2005], defaultable options [Ankirchner et al., 2010] etc., efficient and accurate numerical schemes for solving high dimensional nonlinear BSDEs become thus desired.

In the recent years, many numerical methods have been proposed for solving BSDEs, e.g., [Bouchard and Touzi, 2004, Zhang et al., 2004, Gobet et al., 2005, Lemor et al., 2006, Zhao et al., 2006, Bender and Zhang, 2008, Ma et al., 2008, Zhao et al., 2010, Gobet and Labart, 2010, Crisan and Manolarakis, 2012, Zhao et al., 2014, Ruijter and Oosterlee, 2015, Ruijter and Oosterlee, 2016, Fu et al., 2017], which are not suitable for high-dimensional problems. Therefore, some articles appeared which try to apply those methods for higher dimensional problem by using sparse-grids or parallel computations on graphics processing unit (GPU). For example, Zhang et al. proposed a sparse-grid method for solving BSDEs with the satisfactory results up to 8 dimensions, see [Zhang et al., 2013]. A novel algorithm is designed based on stratified least-squares Monte-Carlo in [Gobet et al., 2016] which shows the results up to 19 dimensions with GPU computing. In [Kapllani and Teng, 2019], the authors parallelized the multi-step scheme proposed in [Teng et al., 2020] on GPU and presented results in very low computation cost. As we can see that only the moderate dimensional BSDEs can be solved with the aid of sparse-grids or GPU parallel computing.

Recently, three new schemes have been proposed which can solve numerically 100-dimensional BSDEs for reasonable, even satisfactory computational time: the deep-learning based algorithm in [Weinan et al., 2017] (we refer as DNN-approach in the rest of the paper) in which the gradient of the solution is approximated by fully-connected neural networks; the multilevel Monte Carlo method based on Picard iteration [Weinan et al., 2019]; the regression tree-based method in [Teng, 2019] where the resulting conditional expectations (from the backward discretization) is represented by the trees. It has been pointed out in [Huré et al., 2020] that the deep learning based algorithms proposed in [Weinan et al., 2017] may be stuck in local minima during the global optimization, and they investigated to solve this problem by defining a local loss function, which is optimized at each time step, namely local optimization. In this way the local minima problem can be overcome, however, it is not the best choice for very high dimensional problems due to the increased computational expense, the satisfactory results are shown only up to 50 dimensions in [Huré et al., 2020]. Furthermore, in our investigation we find that the numerical results of Z [Weinan et al., 2017] are not well stable. The reason can be that different deep networks are taken at each time layers.

In this work, we propose novel ways to solve both the problems mentioned above, respectively. More precisely, instead of feedforward neural networks we employ the stable residual network (NAIS-NET) [Ciccone et al., 2018] for a better numerical convergence and substantial reduction of computational time. In the sequel, we refer this scheme as NAIS-approach. For the local minima problem our suggestion is to exploit the Long Short Term Memory (LSTM) networks

which are a type of Recurrent Neural Networks (RNN). The Y process is approximated at each time step only with one LSTM network, and the gradient of the solution Z is computed with the automatic differentiation (nonlinear Feynman-Kac formula). In order to achieve good approximation by using only one network at all the time layers we need to use a novel loss function. This scheme will be referred as LSTM-approach in the rest of this paper.

The outline of the paper is organized as follows. In the next Section, we introduce some preliminaries including advanced neural networks. Section 3 is devoted to the forward time discretization of the FBSDE. The NAIS- and LSTM-approach are presented in Section 4 and Section 5, respectively. In Section 6, we illustrate our findings with several numerical tests. Finally, Section 7 concludes this work.

2 Preliminaries

2.1 The Feynman-Kac formula

Let $(\Omega, \mathcal{F}, \mathbb{P}, \{\mathcal{F}_t\}_{0 \leq t \leq T})$ be a complete, filtered probability space. In this space a standard d -dimensional Brownian motion W_t is defined, such that the filtration $\{\mathcal{F}_t\}_{0 \leq t \leq T}$ is the natural filtration of W_t . We define $|\cdot|$ as the standard Euclidean norm in the Euclidean space \mathbb{R}^m or $\mathbb{R}^{m \times d}$ and $L^2 = L^2_{\mathcal{F}}(0, T; \mathbb{R}^d)$ the set of all \mathcal{F}_t -adapted and square integrable processes valued in \mathbb{R}^d . A pair of processes $(Y_t, Z_t) : [0, T] \times \Omega \rightarrow \mathbb{R}^m \times \mathbb{R}^{m \times d}$ is the solution of FBSDE (1) if it is \mathcal{F}_t -adapted, square integrable, and satisfies (1) in the sense of

$$Y_t = \xi + \int_t^T f(s, X_s, Y_s, Z_s) ds - \int_t^T Z_s dW_s, \quad t \in [0, T],$$

where $f(t, X_t, Y_t, Z_t) : [0, T] \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^{m \times d} \rightarrow \mathbb{R}^m$ is \mathcal{F}_t -adapted, the third term on the right-hand side is an Itô-type integral and $\xi = g(X_T) : \mathbb{R}^n \rightarrow \mathbb{R}^m$. This solution exist uniquely under Lipschitz conditions [Pardoux and Peng, 1990].

Let us consider that the terminal value Y_T is of the form $g(X_T^{t,x})$, where $X_T^{t,x}$ denotes the solution of forward SDE in (1) starting from x at time t . Then, the solution $(Y_t^{t,x}, Z_t^{t,x})$ of (1) can be presented as [El Karoui et al., 1997]

$$Y_t^{t,x} = u(t, x), \quad Z_t^{t,x} = (\nabla u(t, x)) \sigma(t, x) \quad \forall t \in [0, T], \quad (2)$$

where ∇u denotes the derivative of $u(t, x)$ with respect to the spatial variable x and $u(t, x)$ is the solution of the following semi-linear parabolic PDE:

$$\frac{\partial u}{\partial t} + \sum_{i=1}^n \mu_i(t, x) \frac{\partial u}{\partial x_i} + \frac{1}{2} \sum_{i,j=1}^n (\sigma \sigma^\top)_{i,j}(t, x) \frac{\partial^2 u}{\partial x_i \partial x_j} + f(t, x, u, (\nabla u) \sigma) = 0,$$

with the terminal condition $u(T, x) = g(x)$. This is the Feynman-Kac formula, which plays an important role to formulate the FBSDE as a learning problem.

2.2 Neural Networks as function approximators

We give a brief introduction to neural networks as function approximators. Multilayer or deep neural networks are designed to approximate a large class of functions. They rely on the composition of simple functions, and appear to provide an efficient way to handle high-dimensional

approximation problems. Here, we consider the feedforward neural networks as basic type of deep neural networks for the introduction.

Let $d_0 \in \mathbb{N}$ be the input dimension of the problem and $d_1 \in \mathbb{N}$ the output one. Let $L \in \mathbb{N}$ and $L + 2$ be the global number of layers with $k_l \in \mathbb{N}, l = 0, 1, \dots, L + 1$ the number of neurons on each layer: the first layer is the input layer with $k_0 = d_0$, the last layer is the output layer with $k_{L+1} = d_1$, and the L layers between are called hidden layers, where we choose for simplicity the same dimension $k_l = k \in \mathbb{N}, l = 1, 2, \dots, L - 1$. A feedforward neural network is a function $\psi_{d_0, d_1, L}^\varrho(x; \theta) : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_1}$ as the composition

$$x \in \mathbb{R}^{d_0} \rightarrow T_0(x) \circ \varrho \circ T_1 \circ \varrho \circ \dots \circ \varrho \circ T_L \in \mathbb{R}^{d_1}. \quad (3)$$

Here, $\theta \in \mathbb{R}^\rho$ is the number of network parameters, $x \in \mathbb{R}^{d_0}$ is the input vector and $T_l, l = 0, 1, \dots, L$ are affine transformations: $T_0 : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^k$, $T_l, l = 1, \dots, L - 1 : \mathbb{R}^k \rightarrow \mathbb{R}^k$ and $T_L : \mathbb{R}^k \rightarrow \mathbb{R}^{d_1}$, represented by

$$T_l(x) = A_l x + b_l,$$

where $A_l \in \mathbb{R}^{k_l \times k_{l+1}}$ is the weight matrix and $b_l \in \mathbb{R}^{k_l}$ is the bias vector, $\varrho : \mathbb{R} \rightarrow \mathbb{R}$ is a nonlinear function (e.g. $\tanh(x), \sin(x), \max\{0, x\}$ etc.) called the activation function, and applied componentwise on the outputs of T_l , i.e., $\varrho(x_1, x_2, \dots, x_n) = (\varrho(x_1), \varrho(x_2), \dots, \varrho(x_n))$.

All the weight matrices and bias vectors are the parameters of the neural network, and can be identified as mentioned above with $\theta \in \mathbb{R}^\rho$, where $\rho = \sum_{l=0}^L k_l(k_l + k_{l+1}) = d_0(1 + k) + k(1 + k)(L - 1) + k(1 + d_1)$ is the total number of parameters for the neural network defined in (3), with fixed d_0, d_1, L and allow n to increase. We denote by $\Theta_k = \mathbb{R}^\rho$ the set of possible parameters and $\Psi_{d_0, d_1, L}^\varrho(x; \Theta_k)$ the set of all neural networks $\psi_{d_0, d_1, L}^\varrho(x; \theta)$ and define

$$\Psi_{d_0, d_1, L}^\varrho = \bigcup_{n \in \mathbb{N}} \Psi_{d_0, d_1, L}^\varrho(x; \Theta_k)$$

as the class of all neural networks for the fixed structure given from d_0, d_1, L and ϱ . The fundamental result in [Hornik et al., 1989] justifies the use of neural networks as function approximators.

Theorem 2.1. (*Universal approximation theorem (I)*): $\Psi_{d_0, d_1, L}^\varrho$ is dense in $L^2(\nu)$ for any finite measure $\nu \in \mathbb{R}^d$, whenever ϱ is continuous and nonconstant.

Moreover, we have a universal approximation result for the derivatives in the case of a single hidden layer, i.e., $L = 1$, and when the activation function is a smooth function [Hornik et al., 1990, White, 1992].

Theorem 2.2. (*Universal approximation theorem (II)*): Assume that ϱ is a nonconstant C^k function. Then $\Psi_{d_0, d_1, L}^\varrho$ approximates any function and its derivatives up to order k arbitrary well on any compact set of \mathbb{R}^d .

In Sec. 2.3, we present the stable Residual Neural networks, the Non-Autonomous Input-Output Stable Network (NAIS-Net), which will be used for the NAIS-approach.

2.3 A stable Residual Network: NAIS-Net

The residual networks (ResNet) in [He et al., 2016] address the problem of vanishing or exploding gradient for very deep neural networks by performing identity mappings (shortcut connections) that provide shortcuts for the gradient to flow back through hundreds of layers. Unfortunately, training them still requires extensive hyperparameter tuning, and, even if there was a

principled way to determine the optimal number of layers or processing depth for a given task, it still would be fixed for all patterns. The ResNets are defined as follows,

$$h_{l+1} = h_l + \varrho(h_l, \theta), \quad (4)$$

where $h_l \in \mathbb{R}^{n_l}$ is the output of the l^{th} layer and $h_{l+1} \in \mathbb{R}^{n_{l+1}}$ of the $(l+1)^{th}$ layer and $\theta \in \mathbb{R}^\rho$ the parameters of the l^{th} layer. This relation can be seen as a forward Euler discretization of an Ordinary Differential Equation (ODE) with a step size $\Delta t = 1$ [Haber and Ruthotto, 2017]

$$\dot{h} = \varrho(h_l, \theta). \quad (5)$$

A fundamental problem with the dynamical systems view of deep learning underlying these architectures is that they are autonomous: the input pattern sets the initial condition, only directly affecting the first processing stage. This means that if the system converges, there is either exactly one fix point or one limit cycle. Neither case is desirable from a learning perspective because a dynamical system should have input-dependent convergence properties so that representations are useful for learning. One possible approach is to have a non-autonomous system where, at each iteration, the system is forced by an external input.

For a better numerical convergence and substantial reduction compared to the DNN-approach, in our NAIS-approach we use the non-autonomous network architecture (NAIS-Net) first proposed in [Ciccone et al., 2018] instead of the feedforward neural networks. The reason is that the resulting network is globally asymptotically stable for every initial condition. This is a crucial property for the DNN-approach, since it starts from some random initialization of Y and Z process. NAIS-Net, a non-autonomous input-output stable neural network, tackles this issue by constraining the network as follows:

$$h_{l+1} = h_l + \Delta t \varrho(A_l h_l + B_l x + b_l).$$

In this setting, $(A_l, B_l, b_l) \in \mathbb{R}^\rho$ are trainable parameters. Parameter A_l refers to the traditional weight matrix and b_l refers to the bias. The extra term $B_l x$ is made of a matrix B_l and the input of the network $x \in \mathbb{R}^{d_0}$. Involving the input x makes the system non-autonomous, and the output of the system input-dependent.

Moreover, the weight matrix is constrained to be symmetric definite negative:

$$A_l = -R^\top R - \epsilon I,$$

where ϵ is a hyper-parameter that ensures the eigenvalues are strictly negative, $0 < \epsilon$ ($\epsilon = 0.01$ in our case). An additional constraint is proposed on the Frobenius norm $\|R^\top R\|_F$ with the algorithm. The projection used in this setting forces the weights of the neural network to stay within the set of feasible solutions and makes it more robust. We refer the interested reader to [Ciccone et al., 2018] for a detailed description of NAIS-Net.

We give the same notation to represent the NAIS-Net, namely $\psi_{d_0, d_1, L}^\varrho(x; \theta) : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_1}$ as a function and $\Psi_{d_0, d_1, L}^\varrho$ as a class of NAIS-Nets for the fixed structure given by d_0, d_1, L and ϱ . We assume that the Theorems 2.1 and 2.2 hold here.

2.4 Learning long term dependencies with LSTM

In this section we give a brief introduction to the LSTM networks which are type of RNNs, and will be used for the LSTM-approach. The RNN is a neural network that operates in time. At each time step, it accepts an input vector, updates its (possibly high-dimensional) hidden state

via non-linear activation functions, and uses it to make a prediction of its output. RNNs form a rich model class because their hidden state can store information as high-dimensional distributed representations and their nonlinear dynamics can implement rich and powerful computations, allowing the RNN to perform modeling and prediction tasks for sequences with highly complex structure. A formal definition of the standard RNN [Rumelhart et al., 1986] is as follows: given a sequence of inputs x_1, x_2, \dots, x_N , each in \mathbb{R}^{d_0} , the network computes a sequence of hidden states h_1, h_2, \dots, h_N , each in \mathbb{R}^k , and a sequence of predictions $\hat{y}_1, \hat{y}_2, \dots, \hat{y}_N$, each in \mathbb{R}^{d_1} , by the equations

$$\begin{aligned} h_i &= \varrho(A_{hh}h_{i-1} + A_{hx}x_i + b_h), \\ y_i &= A_y h_i + b_y, \end{aligned}$$

where $(A_{hh}, A_{hx}, b_h, A_y, b_y) \in \mathbb{R}^\rho$ are trainable parameters and $\varrho(x) = \tanh(x)$.

However, the traditional RNNs suffer from the vanishing or exploding gradients problem, when the time step N increases. One way to deal with is to use the LSTM networks. An LSTM layer consists of a set of recurrently connected blocks, known as memory blocks. These blocks can be thought of a differentiable version of the memory chips in a digital computer. Each one contains one or more recurrently connected memory cells and three multiplicative units, the input, output and forget gates that provide continuous analogues of write, read and reset operations for the cells. More precisely, the input to the cells is multiplied by the activation of the input gate, the output to the network is multiplied by that of the output gate, and the previous cell values are multiplied by the forget gate. The network can only interact with the cells via the gates. The LSTM algorithm is presented by the following equations

$$\begin{aligned} f_i &= \varrho(A_{fh}h_{i-1} + A_{fx}x_i + b_f), \\ i_i &= \varrho(A_{ih}h_{i-1} + A_{ix}x_i + b_i), \\ o_i &= \varrho(A_{oh}h_{i-1} + A_{ox}x_i + b_o), \\ \tilde{c}_i &= \varrho(A_{ch}h_{i-1} + A_{cx}x_i + b_c), \\ c_i &= f_i \odot c_{i-1} + i_i \odot \tilde{c}_i, \\ h_i &= o_i \odot \varrho(c_i), \\ y_i &= A_y h_i + b_y, \end{aligned}$$

where $(A_{fh}, A_{fx}, b_f, A_{ih}, A_{ix}, b_i, A_{oh}, A_{ox}, b_o, A_{ch}, A_{cx}, b_c, A_y, b_y) \in \mathbb{R}^\rho$ are trainable parameters, $\varrho(x) = \tanh(x)$ and \odot is the element-wise product, $(A \odot B)_{ij} = (A)_{ij}(B)_{ij}$ for $A, B \in \mathbb{R}^{p \times q}$. The traditional LSTM (RNN) architecture predicts an output \hat{y}_i at time point t_i , and has a label y_i to create a local loss. In order to overcome the local minima problem in the DNN-approach we use a hybrid loss, i.e., local and global loss in our new formulation of the problem, for which the LSTM architecture fits better. We hold here the same notations as before to represent the LSTM, namely $\psi_{d_0, d_1, L}^\varrho(x; \theta) : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_1}$ as a function and $\Psi_{d_0, d_1, L}^\varrho$ as a class of LSTMs for the fixed structure given by d_0, d_1, L and ϱ . We assume that the Theorems 2.1 and 2.2 hold here as well.

3 Forward time discretization of FBSDEs

In this section we consider the forward time discretization of FBSDE, which is the key for presenting the FBSDE as a learning problem. For simplicity, we discuss the discretization with one-dimensional processes, namely $m = n = d = 1$. The extension to higher dimensions is

possible and straightforward. The integral form of the forward SDE in (1) reads

$$X_t = X_0 + \int_0^t \mu(s, X_s) ds + \int_0^t \sigma(s, X_s) dW_s, \quad t \in [0, T]. \quad (6)$$

The drift $\mu(\cdot)$ and diffusion $\sigma(\cdot)$ are assumed to be sufficiently smooth.

Let's consider a time discretization for the time interval $[0, T]$

$$\Delta = \{t_i | t_i \in [0, T], i = 0, 1, \dots, N, t_i < t_{i+1}, \Delta t = t_{i+1} - t_i, t_0 = 0, t_N = T\}.$$

For notational convenience we write $X_i = X_{t_i}$, $W_i = W_{t_i}$, $\Delta W_i = W_{t_{i+1}} - W_{t_i}$ and the approximated process as $X_i^\Delta = X_{t_i}^\Delta$. We start with $X_0^\Delta = X_0$ and one of the following forward schemes is used to determine the other values up to time t_N , namely X_{i+1}^Δ , for $i = 0, 1, \dots, N-1$. The convergence of the schemes is analyzed as the strong convergence rate γ_s and the weak one γ_w . Each one, for sufficiently small Δt , satisfies the following equations [Kloeden and Platen, 2013] (Chapter 9.6 and 9.7)

$$\mathbb{E}[X_T - X_T^\Delta] \leq C (\Delta t)^{\gamma_s}, \quad \mathbb{E}[p(X_T) - p(X_T^\Delta)] \leq C (\Delta t)^{\gamma_w},$$

with $C > 0$ a constant, which does not depend on Δt , and $p(\cdot)$ any $2(\gamma_w + 1)$ times continuously differentiable function of polynomial growth. The well-known Euler scheme reads

$$X_{i+1}^\Delta = X_i^\Delta + \mu(t_i, X_i^\Delta) \Delta t + \sigma(t_i, X_i^\Delta) \Delta W_i,$$

where $\Delta W_i \sim \mathcal{N}(0, \Delta t)$. The scheme has $\gamma_s = \frac{1}{2}$ and $\gamma_w = 1$. For example, the Milstein scheme is given by

$$X_{i+1}^\Delta = X_i^\Delta + \mu(t_i, X_i^\Delta) \Delta t + \sigma(t_i, X_i^\Delta) \Delta W_i + \sigma(t_i, X_i^\Delta) \sigma_x(t_i, X_i^\Delta) \frac{1}{2} (\Delta W_i^2 - \Delta t),$$

which has $\gamma_s = 1$ and $\gamma_w = 1$. The same forward discretizations can be applied for the BSDE. For the time interval $[t_i, t_{i+1}]$, the integral form of the BSDE reads

$$Y_{t_i} = Y_{t_{i+1}} + \int_{t_i}^{t_{i+1}} f(s, X_s, Y_s, Z_s) ds - \int_{t_i}^{t_{i+1}} Z_s dW_s,$$

and the forward integral form is given as

$$Y_{t_{i+1}} = Y_{t_i} - \int_{t_i}^{t_{i+1}} f(s, X_s, Y_s, Z_s) ds + \int_{t_i}^{t_{i+1}} Z_s dW_s.$$

Applying the Euler and Milstein schemes for the latter equation one obtains

$$\begin{aligned} Y_{i+1}^\Delta &= Y_i^\Delta - f(t_i, X_i^\Delta, Y_i^\Delta, Z_i^\Delta) \Delta t + Z_i^\Delta \Delta W_i, \\ &:= F_E(t_i, X_i^\Delta, Y_i^\Delta, Z_i^\Delta, \Delta t, \Delta W_i), \end{aligned} \quad (7)$$

and

$$\begin{aligned} Y_{i+1}^\Delta &= Y_i^\Delta - f(t_i, X_i^\Delta, Y_i^\Delta, Z_i^\Delta) \Delta t + Z_i^\Delta \Delta W_i + Z_i^\Delta Z_{ix}^\Delta \frac{1}{2} (\Delta W_i^2 - \Delta t), \\ &:= F_M(t_i, X_i^\Delta, Y_i^\Delta, Z_i^\Delta, Z_{ix}^\Delta, \Delta t, \Delta W_i), \end{aligned} \quad (8)$$

respectively.

4 The DNN-approach and NAIS-approach

The numerical approximation of $Y_i^\Delta, i = 0, 1, \dots, N$ in the DNN-approach is designed as follows: starting from an estimation $\mathcal{Y}_0(\theta)$ of Y_0^Δ and $\mathcal{Z}_0(\theta)$ of Z_0^Δ , and then using at each time step $t_i, i = 1, 2, \dots, N-1$ a different feedforward multilayer neural network $\psi_{i,d_0,d_1,L}^\theta(x; \theta) : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_1}$ to approximate Z_i^Δ as $\mathcal{Z}_i(\theta)$, where the input x of the network is the Markovian process X_i^Δ and $d_0 = d, d_1 = 1 \times d$. The approximation $Y_i^\Delta, i = 1, 2, \dots, N$, namely $\mathcal{Y}_i(\theta)$, is calculated using the Euler discretization (7). Note that this algorithm forms a global deep neural network composed of neural networks at each time step using as input data the paths of $(X_i^\Delta)_{i=0,1,\dots,N}$ and $(W_i)_{i=0,1,\dots,N}$, and gives as output $\mathcal{Y}_N(\theta)$. The output aims to match the terminal condition $g(X_T^\Delta)$ of the BSDE, and then optimizes over the parameters θ the expected square loss function:

$$\mathbb{E}[|g(X_T^\Delta) - \mathcal{Y}_N(\theta)|^2],$$

which can be done by using stochastic gradient descent-type (SGD) algorithms. The algorithm framework (without using batch normalization, mini-batches and Adam optimizer) for $m = 1$ and $n = d \in \mathbb{N}$ is formulated in Framework 4.1, we refer [Weinan et al., 2017] for a more general framework.

Framework 4.1. *Let $T, \gamma \in (0, \infty)$, $d, \rho, N \in \mathbb{N}$, let $(\Omega, \mathcal{F}, \mathbb{P}, \{\mathcal{F}_t\}_{0 \leq t \leq T})$ be a complete, filtered probability space, $f(t, X_t, Y_t, Z_t) : [0, T] \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^{1 \times d} \rightarrow \mathbb{R}$ and $g(X_T) : \mathbb{R}^d \rightarrow \mathbb{R}$ are \mathcal{F}_t -adapted, let $W_i : [0, T] \times \Omega \rightarrow \mathbb{R}^d, i \in \mathbb{N}_0$, be independent d -dimensional standard Brownian motions on $(\Omega, \mathcal{F}, \mathbb{P}, \{\mathcal{F}_t\}_{0 \leq t \leq T})$, let $t_0, t_1, \dots, t_N \in [0, T]$ be real numbers with*

$$0 = t_0 < t_1 < \dots < t_N = T,$$

for every $\theta \in \mathbb{R}^\rho$ let $\mathcal{Y}_0(\theta) \in \mathbb{R}$ and $\mathcal{Z}_0(\theta) \in \mathbb{R}^{1 \times d}$, for every $\theta \in \mathbb{R}^\rho, i \in \{1, 2, \dots, N-1\}$, let $X_i^\Delta : [0, T] \times \Omega \rightarrow \mathbb{R}^d$ be a Markovian process, let $d_0 = d, d_1 = 1 \times d$ and $\psi_{i,d_0,d_1,L}^\theta(x; \theta) : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_1}$ a function (Neural Network) satisfying Theorem 2.1 and the output given as $\mathcal{Z}_i(\theta)$, for every $\theta \in \mathbb{R}^\rho$, let $\mathcal{Y}(\theta) : 0, \dots, N \times \Omega \rightarrow \mathbb{R}$ be the stochastic process which satisfies for all $i \in \{0, 1, 2, \dots, N-1\}$ the initial value $\mathcal{Y}_0^m(\theta)$ and

$$\mathcal{Y}_{i+1}^m(\theta) = F_E(t_i, X_i^{\Delta,m}, \mathcal{Y}_i^m(\theta), \mathcal{Z}_i^m(\theta), \Delta t, W_i^m),$$

for every $m \in \mathbb{N}_0$ and let $\phi^m : \mathbb{R}^\rho \rightarrow \mathbb{R}$ be the function which satisfies for all $\theta \in \mathbb{R}^\rho, \omega \in \Omega$ that

$$\phi^m(\theta, \omega) = |g(X_N^{\Delta,m}(\omega)) - \mathcal{Y}_N^m(\theta, \omega)|^2,$$

and let $\Phi^m : \mathbb{R}^\rho \rightarrow \mathbb{R}^\rho$ be a function which satisfies for all $\omega \in \Omega, \theta \in \{v \in \mathbb{R}^\rho : (\mathbb{R}^\rho \ni w \rightarrow \phi_s^m(w, \omega) \in \mathbb{R} \text{ is differentiable at } v \in \mathbb{R}^\rho)\}$ that

$$\Phi^m(\theta, \omega) = (\nabla_\theta \phi^m)(\theta, \omega),$$

and let $\Theta : \mathbb{N}_0 \times \Omega \rightarrow \mathbb{R}^\rho$ be a stochastic process which satisfy for all $m \in \mathbb{N}$ that

$$\Theta^m = \Theta^{m-1} - \gamma \Phi^m(\Theta^{m-1}).$$

In the case of sufficiently large $\rho \in \mathbb{N}$ (dimension of network parameters), $N \in \mathbb{N}$ (number of time layers), $m \in \mathbb{N}$ (realisations of the Brownian motion) and sufficiently small $\gamma \in (0, \infty)$ (learning rate), the triple $(X_0, \mathcal{Y}_0(\theta), \mathcal{Z}_0(\theta))$ at t_0 and $(X_i^\Delta, \mathcal{Y}_i(\theta), \mathcal{Z}_i(\theta))_{i=1,2,\dots,N-1}$ for times t_1, \dots, t_{N-1} and $(X_N^\Delta, \mathcal{Y}_N(\theta))$ at terminal time t_N are the approximated solution of the FBSDE (1).

The DNN-approach uses the Adam optimizer [Kingma and Ba, 2014] as a SGD optimization method with mini-batches. In the implementations, $N - 1$ fully-connected feedforward neural networks are employed to approximate $\mathcal{Z}_i(\theta), i = 1, 2, \dots, N - 1, \theta \in \mathbb{R}^\rho$. Each of the neural networks consists of 4 global layers (the input layer (d-dimensional), 2 hidden layers (d+10-dimensional) and the output layer (d-dimensional)). The authors also adopt batch normalization (BN) [Ioffe and Szegedy, 2015] right after each matrix multiplication and before activation. The rectifier function $\mathbb{R} \ni x \rightarrow \max\{0, x\} \in [0, \infty)$ is used as the activation function for the hidden variables. All the weights are initialized using a normal or a uniform distribution without any pre-training. The choice of the dimension of the parameters is given in Remark 4.1.

Remark 4.1. *Let $\rho \in \mathbb{N}$ be the dimension of the parameters in the DNN-approach.*

1. $1 + d$ components of $\theta \in \mathbb{R}^\rho$ are employed for approximating $Y_0^\Delta \in \mathbb{R}$ and $Z_0^\Delta \in \mathbb{R}^{1 \times d}$ respectively.
2. In each of $N - 1$ neural networks, $d(d + 10)$ components of $\theta \in \mathbb{R}^\rho$ are used to uniquely describe the linear transformation form d-dimensional input layer to (d+10)-dimensional first hidden layer.
3. In each of $N - 1$ neural networks, $(d + 10)^2$ components of $\theta \in \mathbb{R}^\rho$ are used to uniquely describe the linear transformation form (d+10)-dimensional first hidden layer to (d+10)-dimensional second hidden layer.
4. In each of $N - 1$ neural networks, $d(d + 10)$ components of $\theta \in \mathbb{R}^\rho$ are used to uniquely describe the linear transformation form (d+10)-dimensional second hidden layer to d-dimensional output layer.
5. After each of above the linear transformation in items 2.-4., a componentwise affine linear transformation within the batch normalization procedure is applied, i.e., in each of the employed $N - 1$ neural networks, $2(d + 10)$ components of $\theta \in \mathbb{R}^\rho$ for the componentwise affine linear transformation between the first linear transformation and the first application of the activation function, and again $2(d + 10)$ components of $\theta \in \mathbb{R}^\rho$ between the second linear transformation and the second application of the activation function, and $2d$ components of $\theta \in \mathbb{R}^\rho$ after the third linear transformation.

Therefore, ρ is given as

$$\begin{aligned}
\rho &= \underbrace{(1 + d)}_{\text{item 1.}} + \underbrace{(N - 1)(d(d + 10) + (d + 10)^2 + d(d + 10))}_{\text{items 2.-4.}} \\
&\quad + \underbrace{(N - 1)(2d(d + 10) + 2d(d + 10) + 2d)}_{\text{item 5.}} \\
&= d + 1 + (N - 1)(2d(d + 10) + (d + 10)^2 + 4(d + 10) + 2d).
\end{aligned} \tag{9}$$

For a better view of the DNN-approach, we give a graphical in Figure 1. The framework for the NAIS-approach is similar to Framework 4.1. Instead of deep networks at each time layer for the approximation for approximating Z , we consider only one network. This means that we have have a very deep network, which grows with the increase of time layers N . As mentioned before, the NAIS-Net fits the new framework quite well. The numerical approximation in our NAIS-approach for $Y_i^\Delta, i = 0, 1, \dots, N$ is designed as follows: starting from an estimation $\mathcal{Y}_0(\theta)$ of Y_0^Δ and $\mathcal{Z}_0(\theta)$ of Z_0^Δ , and then using at each time step $t_i, i = 1, 2, \dots, N - 1$ the same neural

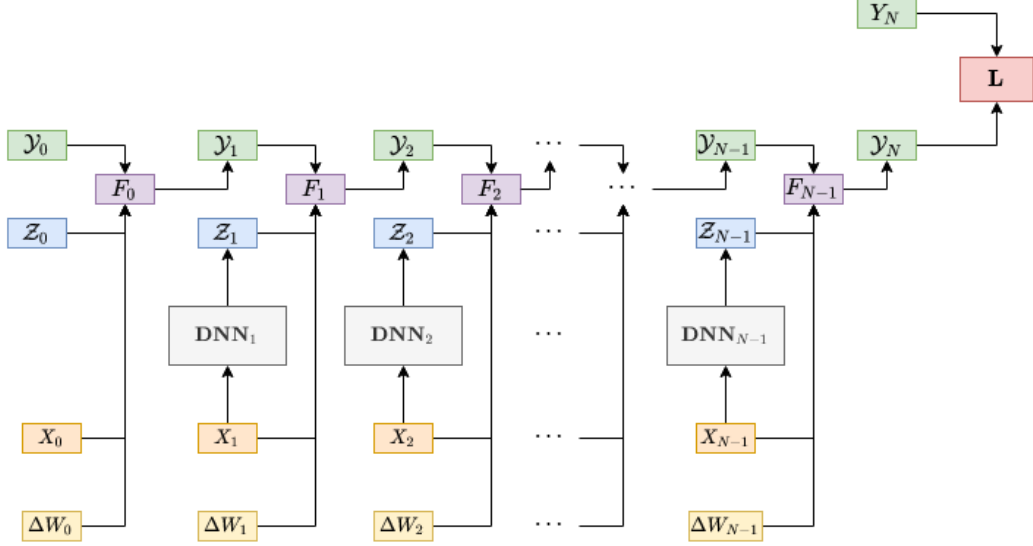


Figure 1: Graph of the DNN-approach.

network (NAIS-NET) $\psi_{d_0, d_1, L}^g(x; \theta) : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_1}$ to approximate Z_i^Δ as $Z_i(\theta)$, where the input x of the network is the time discretization t_i and the Markovian process X_i^Δ (since from the Feynman-Kac formula $Z_t^{t,x} = (\nabla u(t, x))\sigma(t, x)$) and $d_0 = d + 1, d_1 = 1 \times d$. The approximation $Y_i^\Delta, i = 1, 2, \dots, N$, namely $\mathcal{Y}_i(\theta)$, is calculated using the Euler discretization (7). The output aims again to match the terminal condition $g(X_T^\Delta)$ of the BSDE, and then optimizes over the parameters θ the expected square loss function:

$$\mathbb{E}[|g(X_T^\Delta) - \mathcal{Y}_N(\theta)|^2].$$

In the NAIS-approach we also use the Adam optimizer with mini-batches. We consider 5 global layers (the input layer ($d+1$ -dimensional), 3 hidden layers ($d+10$ -dimensional) and the output layer (d -dimensional)). The activation function is $\mathbb{R} \ni x \rightarrow \sin(x) \in [-1, 1]$ is used as the activation function for the hidden variables. All the weights of the network are initialized using [Glorot and Bengio, 2010] and the uniform distribution for the initialization of the solution of the BSDE. The choice of the dimension of the parameters is given in Remark 4.2.

Remark 4.2. Let $\rho \in \mathbb{N}$ be the dimension of the parameters of the NAIS-approach.

1. $1 + d$ components of $\theta \in \mathbb{R}^\rho$ are employed for approximating $Y_0^\Delta \in \mathbb{R}$ and $Z_0^\Delta \in \mathbb{R}^{1 \times d}$ respectively.
2. $(d + 1)(d + 10) + (d + 10)$ components of $\theta \in \mathbb{R}^\rho$ are used to uniquely describe the linear transformation form $d+1$ -dimensional input layer to $(d+10)$ -dimensional first hidden layer; $(d + 10)^2 + (d + 10)$ components of $\theta \in \mathbb{R}^\rho$ are used to uniquely describe the linear transformation form $(d+10)$ -dimensional first hidden layer to $(d+10)$ -dimensional second hidden layer; $(d+10)^2 + (d+10)$ components of $\theta \in \mathbb{R}^\rho$ are used to uniquely describe the linear transformation form $(d+10)$ -dimensional second hidden layer to $(d+10)$ -dimensional third hidden layer.
3. $d(d + 10) + d$ components of $\theta \in \mathbb{R}^\rho$ are used to uniquely describe the linear transformation form $(d+10)$ -dimensional third hidden layer to d -dimensional output layer.

Therefore, ρ is given as

$$\begin{aligned}
\rho &= \underbrace{(1+d)}_{\text{item 1.}} + \underbrace{(d+1)(d+10) + (d+10) + (d+10)^2 + (d+10) + (d+10)^2 + (d+10)}_{\text{item 2.}} \\
&\quad + \underbrace{d(d+10) + d}_{\text{item 3.}} \\
&= 6d + 41 + 2d(d+10) + 2(d+10)^2.
\end{aligned}$$

Compared to (9), the complexity in the NAIS-approach is substantially reduced. Having only one network for each time step reduces the computation time, and the NAIS-Net offers more stability in the approximation of Z process at each time step. The graph of the NAIS-approach is displayed in Figure 2.

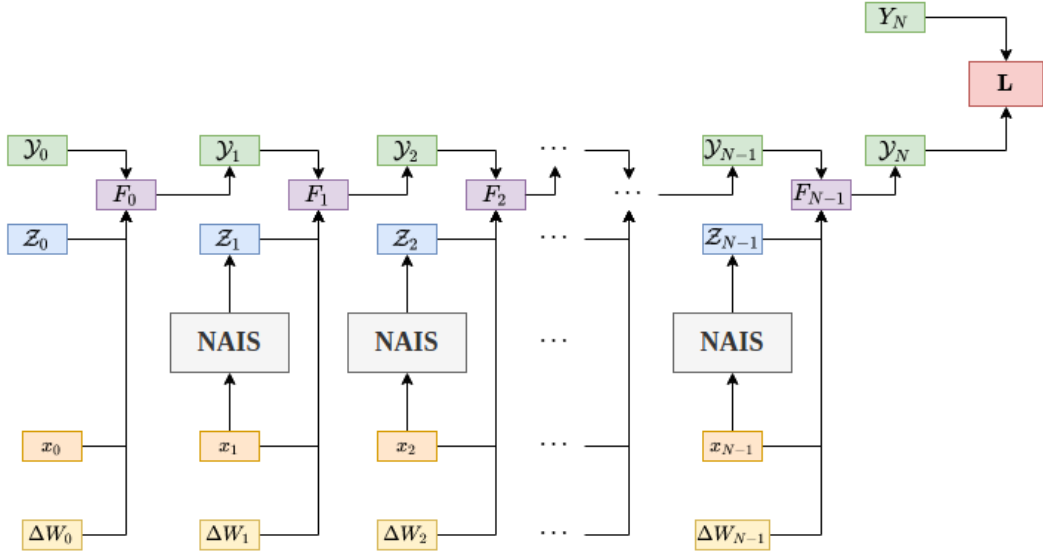


Figure 2: Graph of the NAIS-approach.

5 The LSTM-approach

The idea of the LSTM-approach is to reformulate the learning of the BSDEs to overcome the local minima problem, which could happen in the DNN- and NAIS-approach. The reason for the problem is that a loss function only based on terminal condition is not sufficient for long learning process high value of N . Therefore, we will include some local information in the loss function and apply the LSTM networks. More precisely, since we know both the terminal condition and the dynamics of the BSDE at each time step, we create thus a hybrid loss which includes those information. Next, we need such a neural network that both the dynamics of the BSDE at each time step and the terminal condition can be matched, the LSTM network can serve that purpose. Due to the included local information, i.e., the dynamics of the BSDE at each time step, we use the Milstein method instead of the Euler method for a higher order of accuracy. To the best of our knowledge, the only articles considering a local loss in learning BSDE problems are [Huré et al., 2020] and [Güler et al., 2019].

Using an LSTM connected with deep layers, we develop the following scheme:

- At each time $t_i, i = 0, 1, \dots, N-1$: use the network $\psi_{d_0, d_1, L}^g(x; \theta) : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_1}$ to approximate Y_i^Δ as $\mathcal{Y}_i(\theta)$, where the input x of the network is the time discretization t_i and the Markovian process X_i^Δ , $d_0 = d+1, d_1 = 1$ and

$$\mathcal{Z}_i(\theta) = \sigma(x_i) \frac{\partial \psi_{d_0, d_1, L}^g(x; \theta)}{\partial x} \Big|_{x=x_i}.$$

Then, calculate the local loss

$$\mathbf{L}_i^1(\theta) := |F_M(t_{i-1}, X_{i-1}^\Delta, \mathcal{Y}_{i-1}(\theta), \mathcal{Z}_{i-1}(\theta), \mathcal{Z}_{i-1x}(\theta), \Delta t, \Delta W_{i-1}) - \mathcal{Y}_i(\theta)|^2,$$

where $\mathcal{Z}_{i-1x}(\theta)$ is the derivative of $\mathcal{Z}_{i-1}(\theta)$ with respect to x .

- At time $t_N = T$: calculate $\mathcal{Y}_N(\theta)$ using the same network and

$$\mathcal{Z}_N(\theta) = \sigma(x_N) \frac{\partial \psi_{d_0, d_1, L}^g(x; \theta)}{\partial x} \Big|_{x=x_N}.$$

Calculate the global loss

$$\mathbf{L}_N^1(\theta) := |F_M(t_{N-1}, X_{N-1}^\Delta, \mathcal{Y}_{N-1}(\theta), \mathcal{Z}_{N-1}(\theta), \mathcal{Z}_{N-1x}(\theta), \Delta t, \Delta W_{N-1}) - \mathcal{Y}_N(\theta)|^2,$$

where $\mathcal{Z}_{N-1x}(\theta)$ is the derivative of $\mathcal{Z}_{N-1}(\theta)$ with respect to x . Moreover, using the terminal condition of the BSDE we have the second term of the loss

$$\mathbf{L}^2(\theta) = |g(X_T^\Delta) - \mathcal{Y}_N(\theta)|^2.$$

- The final loss is given as

$$\mathbf{L}(\theta) = \mathbb{E} \left[\sum_{j=1}^N \mathbf{L}_j^1 + \mathbf{L}^2 \right],$$

which will be minimized by optimizing the network over the parameters θ .

Note that all the derivatives above are calculated using automatic differentiation in tensorflow. As it is observed from the algorithm above, the idea of LSTM-approach is to create an network to approximate the solution of the BSDE such that it matches the terminal condition and follows the dynamics of the BSDE at each time step. This later part provides more information to the network, which insures that the algorithm will converge. Note that the parameters are shared at each time step. This reduces the computation time.

The algorithm framework for $m = 1$ and $n = d \in \mathbb{N}$ is formulated in Framework 5.1.

Framework 5.1. Let $T, \gamma \in (0, \infty)$, $d, \rho, N \in \mathbb{N}$, let $(\Omega, \mathcal{F}, \mathbb{P}, \{\mathcal{F}_t\}_{0 \leq t \leq T})$ be a complete, filtered probability space, $f(t, X_t, Y_t, Z_t) : [0, T] \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^{1 \times d} \rightarrow \mathbb{R}$ and $g(X_T) : \mathbb{R}^d \rightarrow \mathbb{R}$ are \mathcal{F}_t -adapted, let $W_i : [0, T] \times \Omega \rightarrow \mathbb{R}^d$, $i \in \mathbb{N}_0$, be independent d -dimensional standard Brownian motions on $(\Omega, \mathcal{F}, \mathbb{P}, \{\mathcal{F}_t\}_{0 \leq t \leq T})$, let $t_0, t_1, \dots, t_N \in [0, T]$ be real numbers with

$$0 = t_0 < t_1 < \dots < t_N = T,$$

for $i \in \{0, 1, 2, \dots, N\}$, let $X_i^\Delta : [0, T] \times \Omega \rightarrow \mathbb{R}^d$ be a Markovian process, let $d_0 = d+1, d_1 = 1$ and $\psi_{d_0, d_1, L}^g(x; \theta) : \mathbb{R}^{d_0} \rightarrow \mathbb{R}^{d_1}$ a function (LSTM connected with deep layers) satisfying Theorem 2.1 and 2.2 and the output given as $\mathcal{Y}_i(\theta)$, let $\psi_{x, d_0, d_1, L}^g(x; \theta) : \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_1}$ be the

derivative function of the network with respect to the input x , let $\sigma \in \mathbb{R}^{d \times d}$ and $\mathcal{Z}_i(\theta) = \sigma(x_i) \psi_{x, d_0, d_1, L}^{\theta}(x; \theta) \Big|_{x=x_i}$, let $\psi_{x, d_0, d_1, L}^{\theta}(x; \theta) : \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_1}$ be the derivative function of $\mathcal{Z}_i(\theta)$ with respect to the input x , for every $m \in \mathbb{N}_0$, $i \in \{1, 2, \dots, N\}$, let $\phi_{i,1}^m : \mathbb{R}^{\rho} \rightarrow \mathbb{R}$ be the function which satisfies for all $\theta \in \mathbb{R}^{\rho}$, $\omega \in \Omega$ that

$$\phi_{i,1}^m(\theta, \omega) = |F_M(t_{i-1}, X_{i-1}^{\Delta, m}(\omega), \mathcal{Y}_{i-1}^m(\theta, \omega), \mathcal{Z}_{i-1}^m(\theta, \omega), \mathcal{Z}_{i-1}^m(\theta, \omega), \Delta t, \Delta W_{i-1}^m(\omega)) - \mathcal{Y}_i(\theta, \omega)|^2,$$

and let $\phi_2^m : \mathbb{R}^{\rho} \rightarrow \mathbb{R}$ be the function which satisfies for all $\theta \in \mathbb{R}^{\rho}$, $\omega \in \Omega$ that

$$\phi_2^m(\theta, \omega) = |g(X_N^{\Delta}(\omega)) - \mathcal{Y}_N(\theta, \omega)|^2,$$

and let $\phi^m : \mathbb{R}^{\rho} \rightarrow \mathbb{R}$ be the function which satisfies for all $\theta \in \mathbb{R}^{\rho}$, $\omega \in \Omega$ that

$$\phi^m(\theta, \omega) = \sum_{j=1}^N \phi_{j,1}^m(\theta, \omega) + \phi_2^m(\theta, \omega),$$

and let $\Phi^m : \mathbb{R}^{\rho} \rightarrow \mathbb{R}^{\rho}$ be a function which satisfies for all $\omega \in \Omega$, $\theta \in \{v \in \mathbb{R}^{\rho} : (\mathbb{R}^{\rho} \ni w \rightarrow \phi_s^m(w, \omega) \in \mathbb{R} \text{ is differentiable at } v \in \mathbb{R}^{\rho})\}$ that

$$\Phi^m(\theta, \omega) = (\nabla_{\theta} \phi^m)(\theta, \omega),$$

and let $\Theta : \mathbb{N}_0 \times \Omega \rightarrow \mathbb{R}^{\rho}$ be a stochastic process which satisfy for all $m \in \mathbb{N}$ that

$$\Theta^m = \Theta^{m-1} - \gamma \Phi^m(\Theta^{m-1}).$$

For sufficiently large $\rho \in \mathbb{N}$, $N \in \mathbb{N}$, $m \in \mathbb{N}$ and sufficiently small $\gamma \in (0, \infty)$, the triple $(X_i^{\Delta}, \mathcal{Y}_i(\theta), \mathcal{Z}_i(\theta))_{i=0,1,\dots,N}$ for times t_0, t_1, \dots, t_N are the approximated solution of the FBSDEs.

In the LSTM-approach, we also use the Adam optimizer with mini-batches. We consider only one network (LSTM connected with deep layers) to approximate $\mathcal{Y}_i(\theta)$, $i = 0, 1, \dots, N$, $\theta \in \mathbb{R}^{\rho}$. It consists of 5 global layers (the input layer ($d+1$ -dimensional), 3 hidden layers ($d+50$ -dimensional) where the first is the LSTM layer and two other ones deep layers, as well as the output layer (1-dimensional)). We use as activation function for LSTM layer $\mathbb{R} \ni x \rightarrow \tanh(x) \in [-1, 1]$ and $\mathbb{R} \ni x \rightarrow \max\{0, x\} \in [0, \infty)$ for the deep layers. All the weights are initialized following the way proposed in [Glorot and Bengio, 2010]. The choice of the dimension of the parameters is given in Remark 5.1.

Remark 5.1. Let $\rho \in \mathbb{N}$ be the dimension of the parameters of the LSTM-approach.

1. $4(d+1)(d+50) + 4(d+50)^2 + 4(d+50)$ components of $\theta \in \mathbb{R}^{\rho}$ are used to uniquely describe the linear transformation form $d+1$ -dimensional input layer to $(d+50)$ -dimensional first hidden layer of the LSTM.
2. $(d+50)^2 + (d+50)$ components of $\theta \in \mathbb{R}^{\rho}$ are used to uniquely describe the linear transformation form $(d+50)$ -dimensional first hidden layer to $(d+50)$ -dimensional second hidden layer; $(d+50)^2 + (d+50)$ components of $\theta \in \mathbb{R}^{\rho}$ are used to uniquely describe the linear transformation form $(d+50)$ -dimensional second hidden layer to $(d+50)$ -dimensional third hidden layer.
3. $(d+10) + 1$ components of $\theta \in \mathbb{R}^{\rho}$ are used to uniquely describe the linear transformation form $(d+10)$ -dimensional third hidden layer to 1-dimensional output layer.

Therefore, ρ is given as

$$\begin{aligned}
\rho &= \underbrace{4(d+1)(d+50) + 4(d+50)^2 + 4(d+50)}_{\text{item 1.}} \\
&\quad + \underbrace{(d+50)^2 + (d+50) + (d+50)^2 + (d+50) + (d+50) + 1}_{\text{items 2.-3.}} \\
&= 11d + 551 + 4d(d+50) + 6(d+50)^2.
\end{aligned}$$

The complexity of the algorithm for LSTM-approach does not depend on N as in the DNN-approach (Remarks 4.1 and 5.1). The complexity is lower for a high N and higher for a low N compared to DNN-approach. However, it insures that the algorithm will converge for a long learning process and very complex structures. The graph of LSTM BSDE is presented in Figure 3.

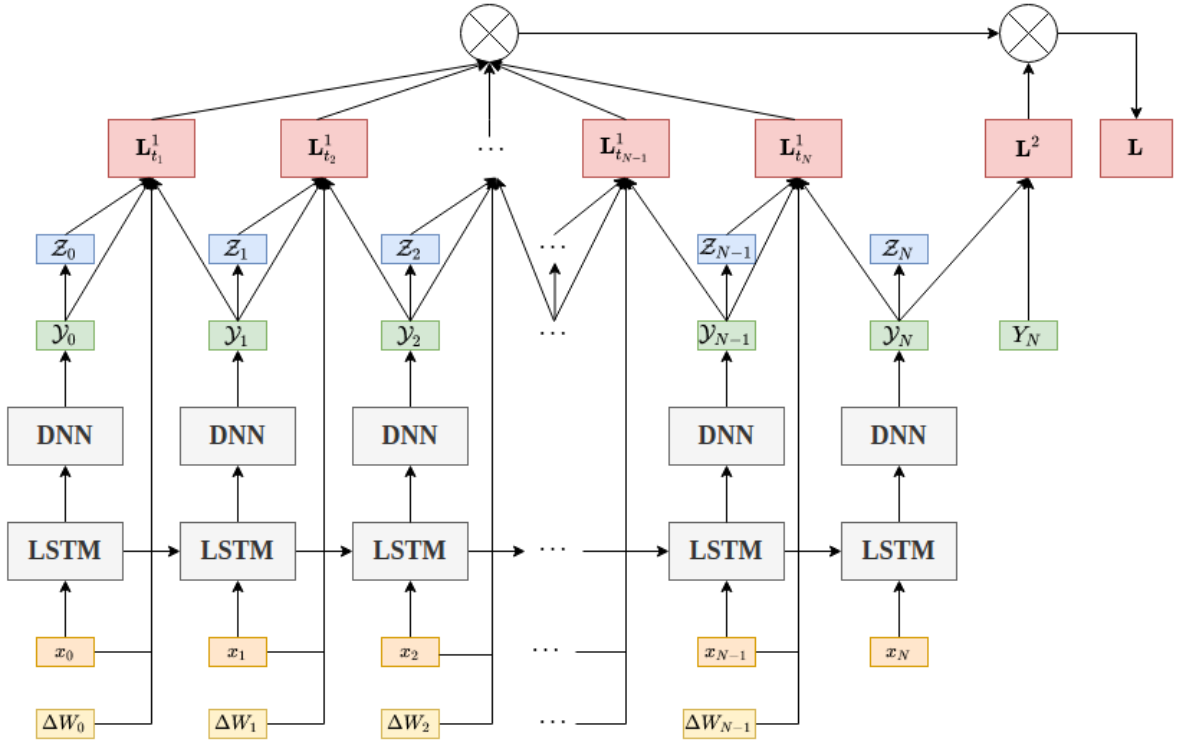


Figure 3: Graph of the LSTM-approach.

6 Numerical results

In this section we illustrate the improved performances in the NAIS- and LSTM-approach compared to the DNN-approach by several examples. The settings of each approach and used hyperparameters will be mentioned below in the corresponding examples. The results are presented using 10 independent runs with Tensorflow 1.15 from Google Colab.

Firstly, we start with the following linear example.

Example 1. Consider the linear BSDE (taken from [Zhao et al., 2006])

$$\begin{cases} -dY_t &= \left(\frac{Y_t}{2} - \frac{Z_t}{2}\right) dt - Z_t dW_t, \\ Y_T &= \sin\left(W_T + \frac{T}{2}\right). \end{cases}$$

The analytic solution is

$$\begin{cases} Y_t &= \sin\left(W_t + \frac{t}{2}\right), \\ Z_t &= \cos\left(W_t + \frac{t}{2}\right). \end{cases}$$

The exact solution $(Y_0, Z_0) = (0, 1)$. We consider maturity $T = 0.5$, and use the same hyperparameters for both the DNN- and NAIS-approach. We choose a learning rate of $1e-2$, and set epochs = 500, $M_{train} = 32000$, $M_{test} = 8000$ and batch size to be 64. M stands for the number of generated samples. We report the results in Tables 1 and 2 for the DNN-approach and the NAIS-approach, respectively, for an increasing time discretization N . Note that the approximations are calculated as the average of 10 independent runs, and $\sigma(\cdot)$ represents the standard deviation. The speedup in Table 2 is calculated as the ratio of computation time (in seconds) of the DNN-approach and these of the NAIS-approach. From Table 1 and 2 we observe that the

Table 1: The results by the DNN-approach for Example 1.

N	\mathcal{Y}_0	$ Y_0 - \mathcal{Y}_0 $	$\sigma(Y_0 - \mathcal{Y}_0)$	\mathcal{Z}_0	$ Z_0 - \mathcal{Z}_0 $	$\sigma(Z_0 - \mathcal{Z}_0)$	Time
8	-0.0007	0.0032	0.0023	0.9735	0.0265	0.0042	16.80
16	-0.0000	0.0016	0.0013	0.9884	0.0120	0.0075	35.36
32	0.0009	0.0018	0.0015	0.9930	0.0089	0.0050	72.94
64	0.0013	0.0024	0.0017	0.9945	0.0107	0.0075	150.10
128	0.0025	0.0028	0.0017	0.9875	0.0132	0.0079	300.62
256	0.0061	0.0062	0.0049	0.9632	0.0368	0.0115	619.65

Table 2: The results by the NAIS-approach results for Example 1.

N	\mathcal{Y}_0	$ Y_0 - \mathcal{Y}_0 $	$\sigma(Y_0 - \mathcal{Y}_0)$	\mathcal{Z}_0	$ Z_0 - \mathcal{Z}_0 $	$\sigma(Z_0 - \mathcal{Z}_0)$	Time	Speedup
8	-0.0007	0.0021	0.0012	0.9782	0.0218	0.0069	3.58	4.69
16	-0.0002	0.0011	0.0007	0.9842	0.0158	0.0056	6.11	5.79
32	-0.0004	0.0012	0.0011	0.9955	0.0060	0.0039	11.85	6.16
64	0.0000	0.0005	0.0003	0.9965	0.0049	0.0034	23.63	6.35
128	0.0004	0.0007	0.0005	0.9998	0.0046	0.0059	48.28	6.23
256	-0.0000	0.0008	0.0004	0.9980	0.0062	0.0058	100.53	6.16

NAIS-approach gives more stable results, especially for Z , for less computation time compared with the DNN-approach. To illustrate this more clearly, we display the averages of paths for Y as \bar{Y} and Z as \bar{Z} , and the averages of approximated paths for \mathcal{Y} as $\bar{\mathcal{Y}}$ and \mathcal{Z} as $\bar{\mathcal{Z}}$ in both the approaches in Figures 4 and 5, respectively, where $N = 128$. We see that the NAIS-approach is more numerically stable for approximating the Z process, which in turn also affects the Y process. As analysed before, the NAIS-approach is not only more numerically stable than the DNN-approach but also faster.

Since linear equations in Example 1 are over simplified and could mislead in terms of efficiency. Therefore, we consider next a nonlinear BSDE.

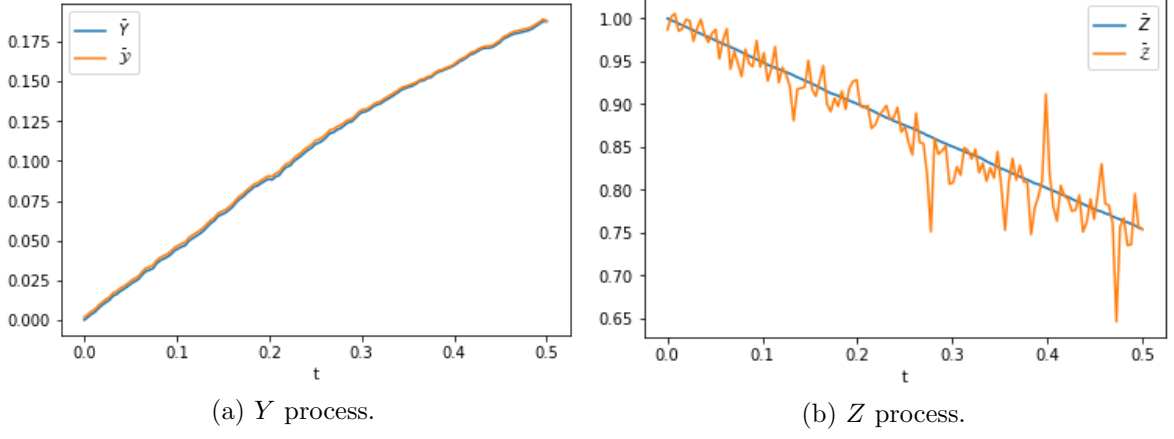


Figure 4: The comparison of averages of the exact path \bar{Y}, \bar{Z} and the approximated paths using the DNN-approach $\bar{\mathcal{Y}}, \bar{\mathcal{Z}}$ for Example 1.

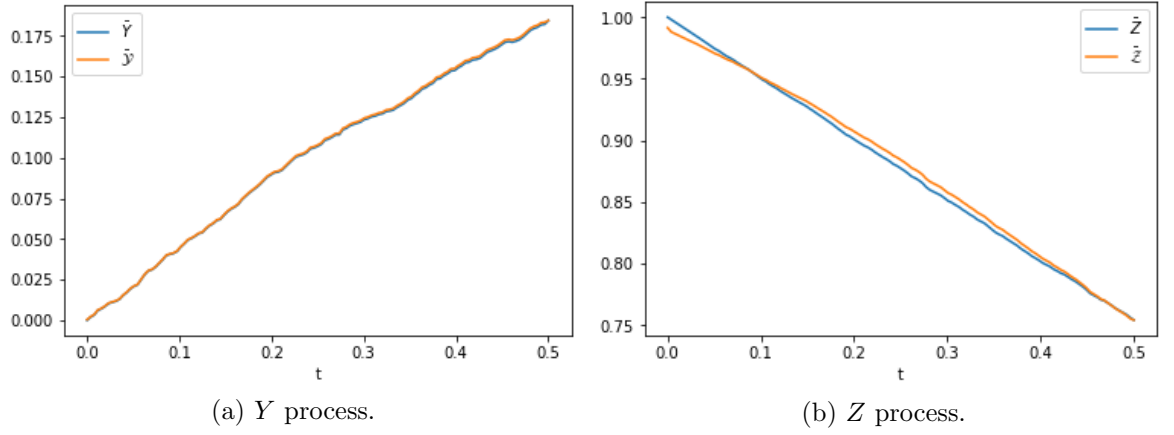


Figure 5: The comparison of averages of the exact path \bar{Y}, \bar{Z} and the approximated paths using the NAIS-approach $\bar{\mathcal{Y}}, \bar{\mathcal{Z}}$ for Example 1.

Example 2. Consider the non-linear BSDE (taken from [Zhao et al., 2010])

$$\begin{cases} -dY_t = \frac{1}{2} (\exp(t^2) - 4tY_t - 3 \exp(t^2 - Y_t \exp(-t^2)) + Z_t^2 \exp(-t^2)) dt - Z_t dW_t, \\ Y_T = \ln(\sin(W_T) + 3) \exp(T^2). \end{cases}$$

with the analytic solution as

$$\begin{cases} Y_t = \ln(\sin(W_t) + 3) \exp(t^2), \\ Z_t = \exp(t^2) \frac{\cos(W_t)}{\sin(W_t) + 3}. \end{cases}$$

The exact solution is $(Y_0, Z_0) = (\ln(3), \frac{1}{3})$. We set $T = 1$ and keep the same hyperparameters as those in the previous example. The results are reported in Table 3, 4 and Figure 6, 7. We see that both the approaches have almost same results (NAIS-approach slightly better) for approximating Y process, the approximations of Z process have been substantially improved by using the NAIS-approach. Furthermore, the computational cost has been reduced.

For the next example we consider the nonlinear pricing with different interest rates. The pricing problem in case of 100 dimension have been considered in [Weinan et al., 2017, Weinan et al., 2019, Teng, 2019]. We start with one dimensional case.

Table 3: The results by the DNN-approach for Example 2.

N	\mathcal{Y}_0	$ Y_0 - \mathcal{Y}_0 $	$\sigma(Y_0 - \mathcal{Y}_0)$	\mathcal{Z}_0	$ Z_0 - \mathcal{Z}_0 $	$\sigma(Z_0 - \mathcal{Z}_0)$	Time
8	1.4088	0.3102	0.0021	0.3356	0.0061	0.0069	14.21
16	1.2552	0.1566	0.0025	0.3388	0.0082	0.0053	30.61
32	1.1781	0.0795	0.0016	0.3377	0.0094	0.0069	63.55
64	1.1380	0.0394	0.0016	0.3385	0.0096	0.0059	129.95
128	1.1176	0.0189	0.0017	0.3408	0.0093	0.0082	272.78
256	1.1064	0.0078	0.0026	0.3253	0.0142	0.0092	571.46

Table 4: The results by the NAIS-approach for Example 2.

N	\mathcal{Y}_0	$ Y_0 - \mathcal{Y}_0 $	$\sigma(Y_0 - \mathcal{Y}_0)$	\mathcal{Z}_0	$ Z_0 - \mathcal{Z}_0 $	$\sigma(Z_0 - \mathcal{Z}_0)$	Time	Speedup
8	1.4100	0.3114	0.0022	0.3380	0.0047	0.0025	6.24	2.28
16	1.2550	0.1564	0.0017	0.3336	0.0069	0.0042	10.20	3.00
32	1.1778	0.0791	0.0019	0.3361	0.0048	0.0031	19.35	3.28
64	1.1375	0.0389	0.0009	0.3303	0.0049	0.0036	38.74	3.35
128	1.1188	0.0202	0.0012	0.3356	0.0075	0.0043	74.68	3.65
256	1.1084	0.0097	0.0011	0.3288	0.0090	0.0068	152.07	3.77

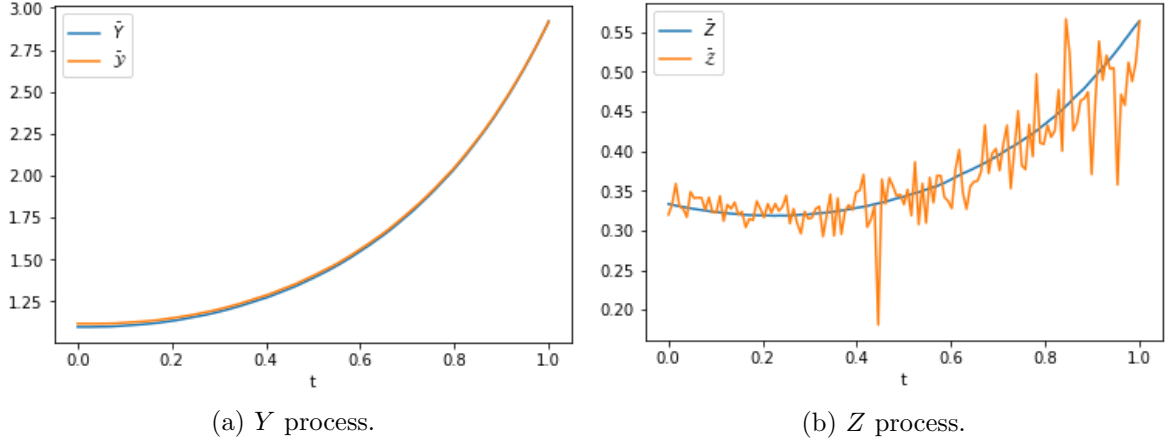


Figure 6: The comparison of averages of the exact path \bar{Y}, \bar{Z} and the approximated paths using the DNN-approach \tilde{Y}, \tilde{Z} for Example 2.

Example 3. The option pricing with different interest rates in one dimension: (taken from [Gobet et al., 2005])

$$\begin{cases} dS_t = \mu S_t dt + \sigma S_t dW_t, & S_0 = S_0, \\ -dY_t = -R^l y_t - \frac{\mu - R^l}{\sigma} Z_t + (R^b - R^l) \max\left(\frac{1}{\sigma} Z_t - Y_t, 0\right) dt - Z_t dW_t, \\ Y_T = \max(S_T - K, 0). \end{cases}$$

The benchmark value with $T = 0.5$, $\mu = 0.06$, $\sigma = 0.2$, $R^l = 0.04$, $R^b = 0.06$, $K = 100$ and $S_0 = 100$ is $Y_0 \doteq 7.156$, which is computed using the finite difference method [Gobet et al., 2005]. For this example, we use a learning rate of $1e-2$, and set epoch = 2000, $M_{train} = 128000$, $M_{test} = 32000$ and batch size to be 64. We report the results in Table 5 and 6 for the DNN- and NAIS-approach, respectively. In this example we obtain surprisingly better results even for approximating Y by the NAIS-approach than the DNN-approach with less computational time.

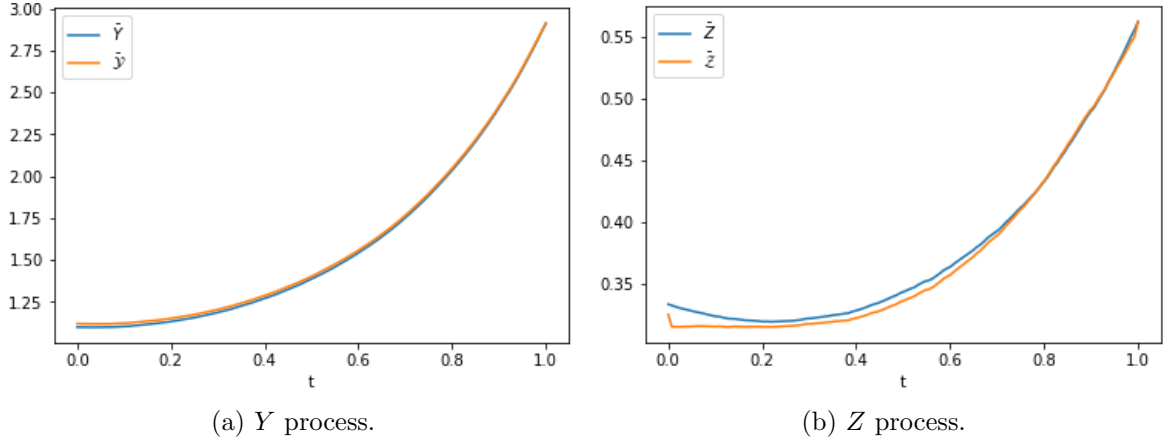


Figure 7: The comparison of averages of the exact path \bar{Y} , \bar{Z} and the approximated paths using the NAIS-approach $\bar{\mathcal{Y}}$, $\bar{\mathcal{Z}}$ for Example 2.

Table 5: The results by the DNN-approach for Example 3.

N	\mathcal{Y}_0	$ Y_0 - \mathcal{Y}_0 $	$\sigma(Y_0 - \mathcal{Y}_0)$	Time
8	7.1394	0.0171	0.0132	22.86
16	7.1505	0.0155	0.0075	47.84
32	7.1520	0.0155	0.0119	99.27
64	7.1399	0.0223	0.0150	205.31
128	7.1486	0.0260	0.0116	429.76
256	7.1585	0.0312	0.0206	1102.34

Table 6: The results by the NAIS-approach for Example 3.

N	\mathcal{Y}_0	$ Y_0 - \mathcal{Y}_0 $	$\sigma(Y_0 - \mathcal{Y}_0)$	Time	Speedup
8	7.1393	0.0184	0.0063	15.30	1.49
16	7.1496	0.0107	0.0052	27.78	1.72
32	7.1496	0.0098	0.0065	49.51	2.01
64	7.1494	0.0070	0.0050	95.68	2.15
128	7.1530	0.0042	0.0033	183.18	2.35
256	7.1526	0.0037	0.0034	303.25	3.64

Note that the approximations of Y in Example 1 and 2 are similar in both the approaches, only the approximation of Z is improved by the NAIS-approach.

Finally, we test our NAIS-approach for that problem of option pricing in 100-dimensions, which is formulated in Example 4.

Example 4. Consider the option pricing FBSDE with different interest rates (taken from [Bergman, 1995])

$$\begin{cases} dS_{t,d} = \mu S_{t,d} dt + \sigma S_{t,d} dW_{t,d}, & d = 1, \dots, D, \quad S_0 = S_0, \\ -dY_t = -R^l Y_t - \frac{\mu - R^l}{\sigma} \sum_{d=1}^D Z_{t,d} + (R^b - R^l) \max\left(\frac{1}{\sigma} \sum_{d=1}^D Z_{t,d} - Y_t, 0\right) dt - (Z_{t,d}) (dW_{t,d})^\top, \\ Y_T = \max(\max_{d=1, \dots, D} (S_{T,d} - K_1, 0) - 2 \max(\max_{d=1, \dots, D} (S_{T,d} - K_2, 0)). \end{cases}$$

The benchmark value with $T = 0.5$, $\mu = 0.06$, $\sigma = 0.2$, $R^l = 0.04$, $R^b = 0.06$, $K_1 = 120$,

$K_2 = 150$ and $S_0 = 100$ is $Y_0 \doteq 21.2988$, which is computed using the multilevel Monte Carlo with 7 Picard iterations approach ([Weinan et al., 2019]). For the high dimensional problem, we use a learning rate of $1e-2$, and set epochs = 4000, $M_{train} = 256000$, $M_{test} = 64000$ and batch size to be 64. We present the results in Tables 7 and 8 for the DNN-approach and the NAIS-approach, respectively. We observe almost same results for Y (as in Example 1 and 2)

Table 7: The results by the DNN-approach for Example 4.

N	\mathcal{Y}_0	$ Y_0 - \mathcal{Y}_0 $	$\sigma(Y_0 - \mathcal{Y}_0)$	Time
8	20.7498	0.5490	0.0327	253.37
16	20.8287	0.4701	0.0253	556.02
32	20.8661	0.4327	0.0290	1088.69
64	20.8708	0.4280	0.0366	2097.23

Table 8: The results by the NAIS-approach for Example 4.

N	\mathcal{Y}_0	$ Y_0 - \mathcal{Y}_0 $	$\sigma(Y_0 - \mathcal{Y}_0)$	Time	Speedup
8	20.7162	0.5826	0.0335	217.95	1.16
16	20.8209	0.4779	0.0272	455.50	1.22
32	20.8504	0.4484	0.0354	857.52	1.27
64	20.8675	0.4313	0.0463	1640.78	1.28

in both the approaches, less computation time is required in the NAIS-approach. Note that the comparison for Z can not be made due to the missing benchmark, a more accurate approximation for Z is expected by the NAIS-approach as it in Example 1 and 2.

As analyzed before, when the solution has extremely complex problems, the DNN-approach can be stuck in local minima, and the same problem is even for the NAIS-approach. Our LSTM-approach is designed to overcome this problem, which is shown by numerical examples. We consider Example 5 where the algorithm in [Weinan et al., 2017] does not converge.

Example 5. Consider the non-linear FBSDE (Taken from [Huré et al., 2020])

$$\begin{cases} dX_{t,d} = \mu dt + \sigma dW_{t,d}, & d = 1, \dots, D, \quad X_0 = x_0, \\ -dY_t = \left(\cos(\bar{X}) \left(\exp\left(\frac{T-t}{2}\right) + \frac{\sigma^2}{2} \right) + \mu \sin(\bar{X}) \right) \exp\left(\frac{T-t}{2}\right) dt \\ \quad + \left(-\frac{1}{2} (\sin(\bar{X}) \cos(\bar{X}) \exp(T-t))^2 + \frac{1}{2} (Y_t \bar{Z})^2 \right) dt - (Z_{t,d}) (dW_{t,d})^\top, \\ Y_T = \cos(\bar{X}), \end{cases}$$

where $\bar{X} = \sum_{d=1}^D X_{t,d}$ and $\bar{Z} = \sum_{d=1}^D Z_{t,d}$. And the analytic solution is given by

$$\begin{cases} Y_t = \exp\left(\frac{T-t}{2}\right) \cos(\bar{X}), \\ Z_t = -\sigma \exp\left(\frac{T-t}{2}\right) \sin(\bar{X}). \end{cases}$$

We start with $d = 1$, the exact solution with $T = 2$, $\mu = 0.2$, $\sigma = 1$ and $x_0 = 1$ is $Y_0 = 1.4687$. The hyperparameters for the DNN-approach are: learning rate of $1e-2$, epochs = 500, $M_{train} = 32000$, $M_{test} = 8000$ and batch size of 64. For the LSTM-approach, we increase the epochs with the increasing N to balance roughly learning and discretization error. The reason for this is that the loss function depends on the time discretization, which holds for training

of the neural networks as well. More precisely, we increasing epochs, training and test samples by the same factor as it for N . For instance, the hyperparameters for $N = 8$ are: epochs = 1000, $M_{train} = 64000$ and $M_{test} = 16000$. Then, for $N = 16$, we increase the aforementioned hyperparameters by a factor of 2. The learning rate and batch size are kept the same, which are $1e-3$ and 64. It is important to note that the DNN-approach can still not converge by increasing the number of epochs. The results are given in Table 9 and 10 for the DNN-approach and the LSTM-approach, respectively. One sees that the DNN-approach fails to compute this example,

Table 9: The results by the DNN-approach for Example 5.

N	\mathcal{Y}_0	$ Y_0 - \mathcal{Y}_0 $	$\sigma(Y_0 - \mathcal{Y}_0)$	Time
8	2.4859	1.0172	0.4773	13.69
16	nan	nan	nan	nan
32	nan	nan	nan	nan
64	nan	nan	nan	nan

Table 10: The results by the LTSM-approach for Example 5.

N	\mathcal{Y}_0	$ Y_0 - \mathcal{Y}_0 $	$\sigma(Y_0 - \mathcal{Y}_0)$	Time
8	0.5301	0.9386	0.0388	53.32
16	0.9451	0.5236	0.0439	144.09
32	1.2435	0.2252	0.0373	368.84
64	1.5335	0.0793	0.0292	969.75

i.e., it diverges. In contrast, our LSTM-approach gives stable results and shows surprisingly good convergence with the increase of N .

Finally, it is of course interesting to see how does the LSTM-approach work for a general nonlinear high dimensional problem. To make it clear we run the LSTM-algorithm for Example 3 and 4, namely the option pricing with different rates in both the 1 and 100 dimensions, where the same hyperparameters as those in Example 5. The results for the LSTM-approach for Example 3 are presented in Table 11, and for Example 4 in Table 12. By comparing Table 11 to Table

Table 11: The results by the LSTM-approach for Example 3.

N	\mathcal{Y}_0	$ Y_0 - \mathcal{Y}_0 $	$\sigma(Y_0 - \mathcal{Y}_0)$	Time
8	7.2137	0.3975	0.2853	21.29
16	7.3325	0.1968	0.1343	60.67
32	7.1659	0.1642	0.1228	157.87

Table 12: The results by the LSTM-approach for Example 4.

N	\mathcal{Y}_0	$ Y_0 - \mathcal{Y}_0 $	$\sigma(Y_0 - \mathcal{Y}_0)$	Time
8	21.4538	0.3918	0.3879	169.63
16	21.5916	0.6013	0.2920	497.56
32	21.5426	0.3161	0.3266	1452.51

5 and 6, and Table 12 to Table 7 and 8 we find that the LSTM-approach does not show any improvements in terms of efficiency and computation time than the DNN- and NAIS-approach, although the results are satisfactory. The LSTM-approach is suitable for the problems whose solution has so complex structure that the local minima problem is thus caused.

7 Conclusions

In this work we first have developed the NAIS-approach which improves the performances of the DNN-approach in [Weinan et al., 2017] regarding computational time and stability of numerical convergences. Secondly, to overcome the local minima problem which could appear in both the DNN-approach and NAIS-approach, especially when the solution has extremely complex structure, we developed the LTSM-approach by including local losses and exploiting the LSTM networks. Several numerical results have been provided which can clearly illustrate our findings. A rigorous convergence analysis for the proposed approaches is the task of our ongoing work.

References

- [Ankirchner et al., 2010] Ankirchner, S., Blanchet-Scalliet, C., and Eyraud-Loisel, A. (2010). Credit risk premia and quadratic bsdes with a single jump. *Int. J. Theor. Appl. Finance*, 13(07):1103–1129.
- [Bender and Zhang, 2008] Bender, C. and Zhang, J. (2008). Time discretization and markovian iteration for coupled fbsdes. *Ann. Appl. Probab.*, 18(1):143–177.
- [Bergman, 1995] Bergman, Y. Z. (1995). Option pricing with differential interest rates. *Rev. Financ. Stud.*, 8(2):475–500.
- [Bouchard and Touzi, 2004] Bouchard, B. and Touzi, N. (2004). Discrete-time approximation and monte-carlo simulation of backward stochastic differential equations. *Stoch. Process Their Appl.*, 111(2):175–206.
- [Ciccone et al., 2018] Ciccone, M., Gallieri, M., Masci, J., Osendorfer, C., and Gomez, F. (2018). Nais-net: Stable deep networks from non-autonomous differential equations. In *Advances in Neural Information Processing Systems*, pages 3025–3035.
- [Crisan and Manolarakis, 2012] Crisan, D. and Manolarakis, K. (2012). Solving backward stochastic differential equations using the cubature method: application to nonlinear pricing. *SIAM J. Financial Math.*, 3(1):534–571.
- [El Karoui et al., 1997] El Karoui, N., Peng, S., and Quenez, M. C. (1997). Backward stochastic differential equations in finance. *Math. Financ.*, 7(1):1–71.
- [Eyraud-Loisel, 2005] Eyraud-Loisel, A. (2005). Backward stochastic differential equations with enlarged filtration: Option hedging of an insider trader in a financial market with jumps. *Stoch. Process Their Appl.*, 115(11):1745–1763.
- [Fahim et al., 2011] Fahim, A., Touzi, N., Warin, X., et al. (2011). A probabilistic numerical method for fully nonlinear parabolic pdes. *Ann. Appl. Probab.*, 21(4):1322–1364.
- [Fu et al., 2017] Fu, Y., Zhao, W., and Zhou, T. (2017). Efficient spectral sparse grid approximations for solving multi-dimensional forward backward sdes. *Discrete Continuous Dyn. Syst. Ser. B*, 22(9):3439.

- [Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.
- [Gobet and Labart, 2010] Gobet, E. and Labart, C. (2010). Solving bsde with adaptive control variate. *SIAM J. Numer. Anal.*, 48(1):257–277.
- [Gobet et al., 2005] Gobet, E., Lemor, J.-P., Warin, X., et al. (2005). A regression-based monte carlo method to solve backward stochastic differential equations. *Ann. Appl. Probab.*, 15(3):2172–2202.
- [Gobet et al., 2016] Gobet, E., López-Salas, J. G., Turkedjiev, P., and Vázquez, C. (2016). Stratified regression monte-carlo scheme for semilinear pdes and bsdes with large scale parallelization on gpu. *SIAM J. Sci. Comput.*, 38(6):C652–C677.
- [Güler et al., 2019] Güler, B., Laignelet, A., and Parpas, P. (2019). Towards robust and stable deep learning algorithms for forward backward stochastic differential equations. *arXiv preprint arXiv:1910.11623*.
- [Haber and Ruthotto, 2017] Haber, E. and Ruthotto, L. (2017). Stable architectures for deep neural networks. *Inverse Probl.*, 34(1):014004.
- [He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- [Hornik et al., 1990] Hornik, K., Stinchcombe, M., and White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5):551–560.
- [Hornik et al., 1989] Hornik, K., Stinchcombe, M., White, H., et al. (1989). Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366.
- [Huré et al., 2020] Huré, C., Pham, H., and Warin, X. (2020). Deep backward schemes for high-dimensional nonlinear pdes. *Math. Comput.*, 89(324):1547–1579.
- [Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.
- [Kapllani and Teng, 2019] Kapllani, L. and Teng, L. (2019). Multistep schemes for solving backward stochastic differential equations on gpu. *arXiv preprint arXiv:1909.13560*.
- [Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- [Kloeden and Platen, 2013] Kloeden, P. E. and Platen, E. (2013). *Numerical solution of stochastic differential equations*, volume 23. Springer Science & Business Media.
- [Labart and Lelong, 2011] Labart, C. and Lelong, J. (2011). A parallel algorithm for solving bsdes-application to the pricing and hedging of american options. *arXiv preprint arXiv:1102.4666*.
- [Lemor et al., 2006] Lemor, J.-P., Gobet, E., Warin, X., et al. (2006). Rate of convergence of an empirical regression method for solving generalized backward stochastic differential equations. *Bernoulli*, 12(5):889–916.

- [Ma et al., 2008] Ma, J., Shen, J., and Zhao, Y. (2008). On numerical approximations of forward-backward stochastic differential equations. *SIAM J. Numer. Anal.*, 46(5):2636–2661.
- [Pardoux and Peng, 1990] Pardoux, E. and Peng, S. (1990). Adapted solution of a backward stochastic differential equation. *Syst. Control. Lett.*, 14(1):55–61.
- [Ruijter and Oosterlee, 2015] Ruijter, M. J. and Oosterlee, C. W. (2015). A fourier cosine method for an efficient computation of solutions to bsdes. *SIAM J. Sci. Comput.*, 37(2):A859–A889.
- [Ruijter and Oosterlee, 2016] Ruijter, M. J. and Oosterlee, C. W. (2016). Numerical fourier method and second-order taylor scheme for backward sdes in finance. *Appl. Numer. Math.*, 103:1–26.
- [Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.
- [Teng, 2019] Teng, L. (2019). A review of tree-based approaches to solve forward-backward stochastic differential equations. *arXiv preprint arXiv:1809.00325v4*.
- [Teng et al., 2020] Teng, L., Lapitckii, A., and Günther, M. (2020). A multi-step scheme based on cubic spline for solving backward stochastic differential equations. *Appl. Numer. Math.*, 150:117–138.
- [Weinan et al., 2017] Weinan, E., Han, J., and Jentzen, A. (2017). Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Commun. Math. Stat.*, 5(4):349–380.
- [Weinan et al., 2019] Weinan, E., Hutzenthaler, M., Jentzen, A., and Kruse, T. (2019). On multilevel picard numerical approximations for high-dimensional nonlinear parabolic partial differential equations and high-dimensional nonlinear backward stochastic differential equations. *J. Sci. Comput.*, 79(3):1534–1571.
- [White, 1992] White, H. (1992). *Artificial neural networks: approximation and learning theory*. Blackwell Publishers, Inc.
- [Zhang et al., 2013] Zhang, G., Gunzburger, M., and Zhao, W. (2013). A sparse-grid method for multi-dimensional backward stochastic differential equations. *J. Comput. Math.*, pages 221–248.
- [Zhang et al., 2004] Zhang, J. et al. (2004). A numerical scheme for bsdes. *Ann. Appl. Probab.*, 14(1):459–488.
- [Zhao et al., 2006] Zhao, W., Chen, L., and Peng, S. (2006). A new kind of accurate numerical method for backward stochastic differential equations. *SIAM J. Sci. Comput.*, 28(4):1563–1581.
- [Zhao et al., 2014] Zhao, W., Fu, Y., and Zhou, T. (2014). New kinds of high-order multistep schemes for coupled forward backward stochastic differential equations. *SIAM J. Sci. Comput.*, 36(4):A1731–A1751.
- [Zhao et al., 2010] Zhao, W., Zhang, G., and Ju, L. (2010). A stable multistep scheme for solving backward stochastic differential equations. *SIAM J. Numer. Anal.*, 48(4):1369–1394.