Bergische Universität Wuppertal

Fakultät für Mathematik und Naturwissenschaften

Institute of Mathematical Modelling, Analysis and Computational Mathematics (IMACM)

Preprint BUW-IMACM 20/42

Lorenc Kapllani and Long Teng

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

This version: Feburary 2021

http://www.math.uni-wuppertal.de

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 by using a single neural network architecture instead of the stacked deep neural networks. Furthermore, those schemes can be stuck in poor local minima or diverges, 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 nonlinear pricing problems in finance.

Keywords *high dimension, backward stochastic differential equations, deep learning, deep 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

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

where $X_t, \mu \in \mathbb{R}^n$, σ is a $n \times d$ matrix, $W_t = \left(W_t^1, \cdots, W_t^d\right)^\top$ is a d-dimensional Brownian motion, $f\left(t, X_t, Y_t, Z_t\right) : [0,T] \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^{m \times d} \to \mathbb{R}^m$ 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

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

where $Y_t \in \mathbb{R}^m$ and $f\left(t, Y_t, Z_t\right) : [0,T] \times \mathbb{R}^m \times \mathbb{R}^{m \times d} \to \mathbb{R}^m$. In the sequel of this work, we investigate to solve (1) in the high 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 [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, 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, , 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, 2013]. A novel algorithm is designed based on stratied 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 [E et al., 2017] (we refer as SDNN-approach in the rest of the paper, where SDNN stands for Stacked Deep Neural Networks) in which the gradient of the solution is approximated by fully-connected neural networks; the multilevel Monte Carlo method based on Picard iteration [E 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 [E et al., 2017] may be stuck in poor local minima or diverge 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 poor local minima or divergence can be overcomed, 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 approximations of Z [E et al., 2017] are not really 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 stacked neural networks we employ one deep neural network for a better numerical convergence and substantial reduction of computational time. In the sequel, we refer this scheme as DNN-approach. For the poor local minima or divergence 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 formular). 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 neural networks. Section 3 is devoted to the forward time discretization of the

FBSDE. The DNN- 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 $L^2 = L^2_{\mathcal{F}}\left(0,T;\mathbb{R}^d\right)$ the set of all $\mathcal{F}_t$-adapted and square integrable processes valued in $\mathbb{R}^d$. A pair of processes $(Y_t, Z_t) : [0,T]\times\Omega \to \mathbb{R}^m \times \mathbb{R}^{m\times d}$ is the solution of FBSDE (1) if it is $\mathcal{F}_t$-adapted, square integrable, and satisfies (1) in the sense of

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

where $f\left(t, X_t, Y_t, Z_t\right) : [0,T]\times\mathbb{R}^n\times\mathbb{R}^m\times\mathbb{R}^{m\times d}\to\mathbb{R}^m$ is $\mathcal{F}_t$-adapted, the third term on the right-hand side is an Itô-type integral and $\xi = g\left(X_T\right) : \mathbb{R}^n\to\mathbb{R}^m$. 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 [Karoui et al., 1997]

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

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

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

with the terminal condition $u\left(T,x\right) = g(x)$. 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 $k_l\in\mathbb{N}, l = 0,1,\cdots,L+1$ 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 $k_l = k\in\mathbb{N}, l = 1,2,\cdots,L-1$. A feedforward neural network is a function $\psi^{\varrho}_{d_0,d_1,L}(x;\theta) : \mathbb{R}^{d_0}\to\mathbb{R}^{d_1}$ as the composition

$$x\in\mathbb{R}^{d_0} \longmapsto T_0(x)\circ\varrho\circ T_1\circ\varrho\circ\cdots\circ\varrho\circ T_L\in\mathbb{R}^{d_1}. \tag{3}$$

Here, $\theta \in \mathbb{R}^\rho$ is the number of network parameters, $x \in \mathbb{R}^{d_0}$ is the input vector and $T_l, l = 0, 1, \cdots, L$ are affine transformations: $T_0 : \mathbb{R}^{d_0} \to \mathbb{R}^k$, $T_l, l = 1, \cdots, L-1 : \mathbb{R}^k \to \mathbb{R}^k$ and $T_L : \mathbb{R}^k \to \mathbb{R}^{d_1}$, 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, $\varrho : \mathbb{R} \to \mathbb{R}$ 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., $\varrho(x_1, x_2, \cdots, x_n) = (\varrho(x_1), \varrho(x_2), \cdots, \varrho(x_n))$. 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 k to increase. We denote by $\Theta_k = \mathbb{R}^\rho$ the set of possible parameters and $\Psi^\varrho_{d_0,d_1,L}(x; \Theta_k)$ the set of all neural networks $\psi^\varrho_{d_0,d_1,L}(x; \theta)$ and define

$$\Psi^\varrho_{d_0,d_1,L} = \bigcup_{k \in \mathbb{N}} \Psi^\varrho_{d_0,d_1,L}(x; \Theta_n)$$

as the class of all neural networks for the fixed structure given from d_0, d_1, L and ϱ.
In Sec. 2.3, we present the LSTMs, which will be used for the LSTM-approach.

2.3 Learning long term dependencies with LSTM

LSTM networks are type of RNNs 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 $x_1, x_2, \cdots, x_N$, each in $\mathbb{R}^{d_0}$, the network computes a sequence of hidden states $h_1, h_2, \cdots, h_N$, each in $\mathbb{R}^k$, and a sequence of predictions $\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_N$, 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

$$\begin{aligned} f_i &= \varrho(A_{fh}h_{i-1} + A_{fx}x_i + b_f), \\ i_i &= \varrho(A_{ih}h_{i-1} + A_{ix}x_i + b_i), \\ o_i &= \varrho(A_{oh}h_{i-1} + A_{ox}x_i + b_o), \\ \tilde{c}_i &= \varrho(A_{ch}h_{i-1} + A_{cx}x_i + b_c), \\ c_i &= f_i \odot c_{i-1} + i_i \odot \tilde{c}_i, \\ h_i &= o_i \odot \varrho(c_i), \\ y_i &= A_y h_i + b_y, \end{aligned} \tag{4}$$

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 SDNN-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. To represent the LSTM, we define as a function $\psi^{\varrho}_{d_0,d_1,L}(x,h;\theta) : \mathbb{R}^{d_0} \to \mathbb{R}^{d_1}$ which performs the calculations as in (4) at time t_i, and $\Psi^{\varrho}_{d_0,d_1,L}$ as a class of LSTMs for the fixed structure given by d_0, d_1, L and ϱ.

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

$$X_t = X_0 + \int_0^t \mu\left(s, X_s\right) ds + \int_0^t \sigma\left(s, X_s\right) dW_s, \quad t \in [0,T]\,. \tag{5}$$

The drift $\mu(\cdot)$ and diffusion $\sigma(\cdot)$ are assumed to be sufficiently smooth.
Lets consider a time discretization for the time interval $[0,T]$

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

For notational convenience we write $X_i = X_{t_i}$, $W_i = W_{t_i}$, $\Delta W_i = W_{i+1} - W_i$ and the approximated process as $X^\Delta_i = X^\Delta_{t_i}$. We start with $X^\Delta_0 = X_0$ and one of the following forward schemes is used to determine the other values up to time t_N, namely X^Δ_{i+1}, for $i = 0,1,\cdots,N-1$. 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, 1992] (Chapter 9.6 and 9.7)

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

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

$$X^\Delta_{i+1} = X^\Delta_i + \mu\left(t_i, X^\Delta_i\right)\Delta t + \sigma\left(t_i, X^\Delta_i\right)\Delta W_i,$$

where $\Delta W_i \sim \mathcal{N}(0,\, \Delta t)$. The scheme has $\gamma_s = \frac{1}{2}$ and $\gamma_w = 1$. 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

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

and the forward integral form is given as

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

Applying the Euler scheme for the latter equation one obtains

$$\begin{aligned} Y_{i+1}^{\Delta} &= Y_i^{\Delta} - f\left(t_i, X_i^{\Delta}, Y_i^{\Delta}, Z_i^{\Delta}\right)\Delta t + Z_i^{\Delta}\Delta W_i, \\ &:= F(t_i, X_i^{\Delta}, Y_i^{\Delta}, Z_i^{\Delta}, \Delta t, \Delta W_i). \end{aligned} \tag{6}$$

4 The SDNN-approach and DNN-approach

The numerical approximation of $Y_i^{\Delta}, i = 0, 1, \cdots, N$ in the SDNN-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 $t_i, i = 1, 2, \cdots, N-1$ a different feedforward multilayer neural network $\psi^{\varrho}_{i,d_0,d_1,L}(x;\theta) : \mathbb{R}^{d_0} \to \mathbb{R}^{d_1}$ 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 $Y_i^{\Delta}, i = 1, 2, \cdots, N$, namely $\mathcal{Y}_i(\theta)$, is calculated using the Euler discretization (6). 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 $(X_i^{\Delta})_{i=0,1,\cdots,N}$ and $(W_i)_{i=0,1,\cdots,N}$, 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:

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

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 [E et al., 2017] for a more general framework.

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

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

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

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

for every $m \in \mathbb{N}_0$ and let $\phi^m : \mathbb{R}^{\rho} \to \mathbb{R}$ 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 $\Phi^m : \mathbb{R}^{\rho} \to \mathbb{R}^{\rho}$ be a function which satisfies for all $\omega \in \Omega$, $\theta \in \{v \in \mathbb{R}^{\rho} : (\mathbb{R}^{\rho} \ni w \to \phi_s^m(w, \omega) \in \mathbb{R}$ is differentiable at $v \in \mathbb{R}^{\rho})\}$ that

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

and let $\Theta : \mathbb{N}_0 \times \Omega \to \mathbb{R}^\rho$ *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), $m \in \mathbb{N}$ (gradient descent iteration) 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 $(X_i^\Delta, \mathcal{Y}_i(\theta), \mathcal{Z}_i(\theta))_{i=1,2,\cdots,N-1}$ for times $t_1, \cdots, t_{N-1}$ and $(X_N^\Delta, \mathcal{Y}_N(\theta))$ at terminal time t_N are the approximated solution of the FBSDE (1).
The SDNN-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 $\mathcal{Z}_i(\theta), i = 1, 2, \cdots, N-1, \theta \in \mathbb{R}^\rho$. 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 $\mathbb{R} \ni x \to \max\{0, x\} \in [0, \infty)$ 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 SDNN-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*

$$\begin{aligned}\rho = &\underbrace{(1+d)}_{\text{item 1.}} + \underbrace{(N-1)(d(d+10) + (d+10)^2 + d(d+10))}_{\text{items 2.-4.}} \\ &+ \underbrace{(N-1)(2d(d+10) + 2d(d+10) + 2d)}_{\text{item 5.}} \\ &= d + 1 + (N-1)(2d(d+10) + (d+10)^2 + 4(d+10) + 2d).\end{aligned} \tag{7}$$

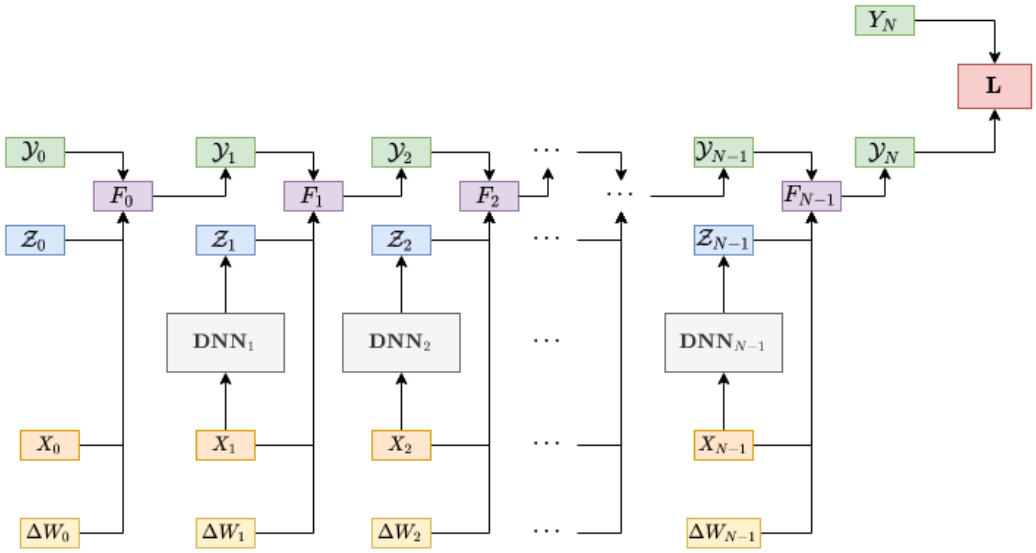


Figure 1: Graph of the SDNN-approach.

For a better view of the SDNN-approach, we give a graphical in Figure 1. The framework for the DNN-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. The numerical approximation in our DNN-approach for $Y_i^\Delta, i = 0, 1, \cdots, N$ 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 $t_i, i = 1, 2, \cdots, N-1$ the same neural network $\psi^{\varrho}_{d_0,d_1,L}(x;\theta) : \mathbb{R}^{d_0} \to \mathbb{R}^{d_1}$ to approximate Z_i^Δ as $\mathcal{Z}_i(\theta)$, where the input x of the network is now 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 $Y_i^\Delta, i = 1, 2, \cdots, N$, namely $\mathcal{Y}_i(\theta)$, is calculated using the Euler discretization (6). 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:

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

In the DNN-approach we also use the Adam optimizer with mini-batches. We consider 6 global layers (the input layer (d+1-dimensional), 4 hidden layers (d+10-dimensional) and the output layer (d-dimensional). Compared to the SDNN-approach we take 2 hidden layers more for a better accuracy, since only one (same) network at each time step is used. Note that the approximations can not be further improved by taking > 4 hidden layers. The rectifier function $\mathbb{R} \ni x \to \max(0, x) \in [0, \infty)$ 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 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. *$(d+1)(d+10) + (d+10)$ 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;* $(d+10)^2+(d+10)$ *components of* $\theta \in \mathbb{R}^\rho$ *are used to uniquely describe the linear transformation form (d+10)-dimensional third hidden layer to (d+10)-dimensional fourth 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 fourth hidden layer to d-dimensional output layer.*

Therefore, ρ *is given as*

$$\begin{aligned}\rho &= \underbrace{(1+d)+(d+1)(d+10)+4(d+10)+3(d+10)^2}_{\text{items 1.-2.}} \\ &\quad + \underbrace{d(d+10)+d}_{\text{item 3.}} \\ &= 2d+1+(2d+5)(d+10)+3(d+10)^2.\end{aligned}$$

Compared to (7), the complexity in the DNN-approach is substantially reduced. Having only one network for each time step reduces the computation time, and one deep neural network offers more stable approximation of Z process at each time step. The graph of the DNN-approach is similar to Figure 1, with the exception of having the same network at each time step, and the input has also the time discretization t_i.

5 The LSTM-approach

The idea of the LSTM-approach is to reformulate the learning of the BSDEs to overcome the poor local minima or divergence problem, which could occur in both the SDNN- and DNN-approaches. 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. It has been indicated by [Huré et al., 2020] that using the LSTM networks only with a global loss function can not solve that problem. Our idea is to include some local information in the loss function and then 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 these informations. 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 Euler method. 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, we develop the following scheme:

- At time t_0, use an estimation $\mathcal{Y}_0(\theta)$ for Y_0^Δ and $\mathcal{Z}_0(\theta)$ for Z_0^Δ.
- At each time t_i, $i = 1, 2, \cdots, N$: use the LSTM network $\psi^\varrho_{d_0,d_1,L}(x,h;\theta) : \mathbb{R}^{d_0} \to \mathbb{R}^{d_1}$ to approximate Y_i^Δ as $\mathcal{Y}_i(\theta)$, where the input x of the network is the Markovian process X_i^Δ, $d_0 = d, d_1 = 1$ and h is the activation at a previous time step, and

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

  Then, calculate the local loss

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

- Again at time $t_N = T$, using the terminal condition of the BSDE we have the second term of the loss (global loss)
$$\mathbf{L}^2(\theta) = |g(X_T^\Delta) - \mathcal{Y}_N(\theta)|^2.$$
- The final loss is given as
$$\mathbf{L}(\theta) = \mathbb{E}\big[\sum_{j=1}^{N} \mathbf{L}_j^1 + \mathbf{L}^2\big],$$
which will be minimized by optimizing the network over the parameters θ.

Note that 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 a 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 not get stuck to poor local minima or 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 $(\Omega, \mathcal{F}, \mathbb{P}, \{\mathcal{F}_t\}_{0\le t\le T})$ be a complete, filtered probability space, $f(t, X_t, Y_t, Z_t) : [0,T] \times \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^{1\times d} \to \mathbb{R}$ and $g(X_T) : \mathbb{R}^d \to \mathbb{R}$ are $\mathcal{F}_t$-adapted, let $W_i : [0,T] \times \Omega \to \mathbb{R}^d$, $i \in \mathbb{N}_0$, be independent d-dimensional standard Brownian motions on $(\Omega, \mathcal{F}, \mathbb{P}, \{\mathcal{F}_t\}_{0\le t\le T})$, let $t_0, t_1, \cdots, t_N \in [0,T]$ be real numbers with*

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

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 $i \in \{0, 1, 2, \cdots, N\}$, let $X_i^\Delta : [0,T] \times \Omega \to \mathbb{R}^d$ be a Markovian process, let $d_0 = d, d_1 = 1$ and $\psi^{\varrho}_{d_0,d_1,L}(x, h; \theta) : \mathbb{R}^{d_0} \to \mathbb{R}^{d_1}$ a function (LSTM) and the output given as $\mathcal{Y}_i(\theta)$, let $\psi^{\varrho}_{x,d_0,d_1,L}(x, h; \theta) : \mathbb{R}^{d_1} \to \mathbb{R}^{d_0}$ be the derivative function of the network with respect to the input x, let $\sigma \in \mathbb{R}^{d\times d}$ and $\mathcal{Z}_i(\theta) = \sigma(x_i)\, \psi^{\varrho}_{x,d_0,d_1,L}(x; \theta)\Big|_{x=x_i}$, for every $m \in \mathbb{N}_0$, $i \in \{1, 2, \cdots, N\}$, let $\phi_{i,1}^m : \mathbb{R}^\rho \to \mathbb{R}$ be the function which satisfies for all $\theta \in \mathbb{R}^\rho$, $\omega \in \Omega$ that

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

and let $\phi_2^m : \mathbb{R}^\rho \to \mathbb{R}$ 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 $\phi^m : \mathbb{R}^\rho \to \mathbb{R}$ 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 $\Phi^m : \mathbb{R}^\rho \to \mathbb{R}^\rho$ be a function which satisfies for all $\omega \in \Omega$, $\theta \in \{v \in \mathbb{R}^\rho : (\mathbb{R}^\rho \ni w \to \phi_s^m(w, \omega) \in \mathbb{R}$ is differentiable at $v \in \mathbb{R}^\rho)\}$ that

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

and let $\Theta : \mathbb{N}_0 \times \Omega \to \mathbb{R}^\rho$ 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 $(X_i^\Delta, \mathcal{Y}_i(\theta), \mathcal{Z}_i(\theta))_{i=0,1,\cdots,N}$ for times $t_0, t_1, \cdots, t_N$ are the approximated solution of the FBSDEs. In the LSTM-approach, we use the Adam optimizer with mini-batches. We consider only one network (LSTM) to approximate $\mathcal{Y}_i(\theta), i = 1, \cdots, N, \theta \in \mathbb{R}^\rho$. It has the structure as in (4). We use as activation function for LSTM layer $\mathbb{R} \ni x \to \tanh(x) \in [-1, 1]$. The initial value of activation h_0 and cell c_0 are considered as weights to be learned during the optimization process. All the network weights are initialized following the way proposed in [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 5.1.

Remark 5.1. *Let $\rho \in \mathbb{N}$ be the dimension of the parameters in the LSTM-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. *$2(d+10)$ components of $\theta \in \mathbb{R}^\rho$ are employed for initializing the activation h_0 and the cell c_0.*
3. *$4d(d+10) + 4(d+10)^2 + 4(d+10)$ components of $\theta \in \mathbb{R}^\rho$ are used to uniquely describe the linear transformation form d-dimensional input layer to (d+10)-dimensional output of the LSTM.*
4. *$(d+10)+1$ components of $\theta \in \mathbb{R}^\rho$ are used to uniquely describe the linear transformation form (d+10)-dimensional LSTM output layer to 1-dimensional output layer.*

Therefore, ρ is given as

$$\begin{aligned}\rho &= \underbrace{1 + d + 2(d+10) + 4d(d+10)^2 + 4(d+10)}_{\text{items 1.-3.}} \\ &\quad + \underbrace{(d+10)+1}_{\text{item 4.}} \\ &= 8d + 72 + 4d(d+10)^2.\end{aligned}$$

Since the complexity of the algorithm for LSTM-approach does not depend on N as in the SDNN-approach (Remarks 4.1 and 5.1), it is thus lower for a high N and higher for a low N compared to SDNN-approach. However, it insures that the algorithm will converge for a long learning process and very complex structures. The graph of LSTM-approach is presented in Figure 2.

6 Numerical results

In this section we illustrate the improved performances in the DNN- and LSTM-approach compared to the SDNN-approach in several high dimensional examples, which are taken in the related references for the purpose of comparison. The settings of each approach and used hyperparameters will be mentioned below in each example. The results are presented using 10 independent runs with Tensorflow 1.15 from Google Colab.
We start with the Burgers type FBSDE, where the driver function depends on the processes Y and Z, and has an explicit solution.

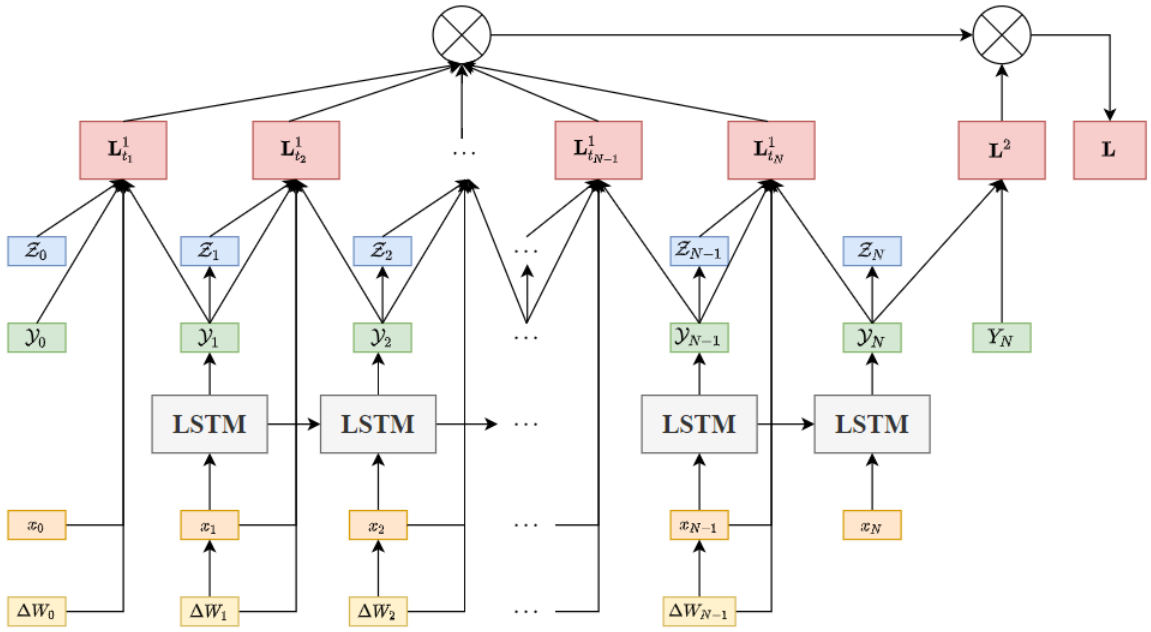


Figure 2: Graph of the LSTM-approach.

Example 1. Consider the Burgers type FBSDE [E et al., 2017]

$$\begin{cases} dX_t &= \sigma\, dW_t, \quad X_0 = 0, \\ -dY_t &= \left(Y_t - \frac{2+d}{2d}\right)\left(\sum_{i=1}^d Z_t^i\right) dt - Z_t\, dW_t, \\ Y_T &= 1 - \frac{1}{1+\exp\left(T+\frac{1}{d}\sum_{i=1}^d X_T^i\right)}, \end{cases}$$

where, $W_t = (W_t^1, W_t^2, \cdots, W_t^d)^\top$, $X_t = (X_t^1, X_t^2, \cdots, X_t^d)^\top$ and $Z_t = (Z_t^1, Z_t^2, \cdots, Z_t^d)$. The analytic solution is given by

$$\begin{cases} Y_t &= 1 - \frac{1}{1+\exp\left(t+\frac{1}{d}\sum_{i=1}^d X_t^i\right)}, \\ Z_t &= \frac{\sigma}{d}\frac{\exp\left(t+\frac{1}{d}\sum_{i=1}^d X_t^i\right)}{\left(1+\exp\left(t+\frac{1}{d}\sum_{i=1}^d X_t^i\right)\right)^2}\mathbb{1}_{\mathbb{R}^d}. \end{cases}$$

The exact solution with $d = 50$, $T = 0.2$ and $\sigma = \frac{d}{\sqrt{2}}$ is $(Y_0, Z_0) \doteq (0.5, (0.1768, \cdots, 0.1768))$. We consider the same hyperparameters for both the SDNN- and DNN-approach. We choose a learning rate of $5e-3$, set number of iterations for the optimizer to $m = 6000$ and batch size to be 64. The results are reported in Table 1 and 2 for the SDNN-approach and the DNN-approach, respectively, for an increasing time discretization N. Note that the approximations are calculated as the average of 10 independent runs, $|\cdot|$ is the absolute value and $s(\cdot)$ represents the standard deviation. Moreover, $\epsilon_{Y_0} = |Y_0 - \mathcal{Y}_0|$, $\boldsymbol{\mathcal{Z}}_0 = \frac{1}{d}\sum_{i=1}^d \mathcal{Z}_0^i$ and $\epsilon_{Z_0} = \frac{\sum_{i=1}^d |Z_0^i - \mathcal{Z}_0^i|}{d}$. The speedup in Table 2 is calculated as the ratio of computation time (in seconds) of the SDNN-approach and those of the DNN-approach.

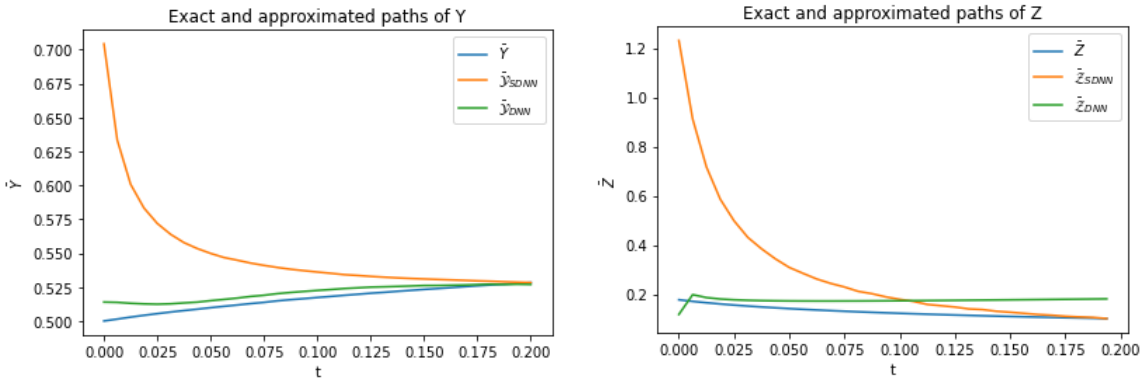
From Table 1 and 2 we observe that the DNN-approach gives higher accuracy for both processes Y and Z, for less computation time. To illustrate how good paths of each process are approximated, we display the averages of paths for Y as $\bar{Y}$ and Z as $\bar{Z}$, and the averages of approximated paths for $\mathcal{Y}$ as $\bar{\mathcal{Y}}$ and $\mathcal{Z}$ as $\bar{\mathcal{Z}}$ for both approaches in Figure 3, where $N = 32$, and the average over the dimension is also considered for the Z process, in order to have one value at each time point.

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

N	$\mathcal{Y}_0$	ϵ_{Y_0}	$s(\epsilon_{Y_0})$	$\boldsymbol{\mathcal{Z}}_0$	ϵ_{Z_0}	$s(\epsilon_{Z_0})$	Time
8	0.9795	0.4795	0.0506	0.7506	0.5738	0.0054	170.19
16	0.8172	0.3172	0.0250	1.1494	0.9727	0.0131	375.11
32	0.7042	0.2042	0.0085	1.2302	1.0535	0.0138	880.72

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

N	$\mathcal{Y}_0$	ϵ_{Y_0}	$s(\epsilon_{Y_0})$	$\boldsymbol{\mathcal{Z}}_0$	ϵ_{Z_0}	$s(\epsilon_{Z_0})$	Time	Speedup
8	0.5202	0.0486	0.0254	0.1777	0.1190	0.0454	97.18	1.87
16	0.5820	0.0820	0.0932	0.2402	0.1322	0.0754	191.27	1.96
32	0.5139	0.0852	0.0773	0.1173	0.1198	0.0610	418.47	2.11



(a) Y process. (b) Z process.

Figure 3: The comparison of averages of the exact path $\bar{Y}, \bar{Z}$ and the approximated paths $\bar{\mathcal{Y}}, \bar{\mathcal{Z}}$ using the SDNN- and DNN-approach for Example 1.

We see that the DNN-approach approximates paths of both the processes better than the SDNN-approach in this example when $d = 50$.
Next, we consider an example with a driver function where the Z process grows quadratically.

Example 2. Consider the nonlinear BSDE [Gobet and Turkedjiev, 2015]

$$\begin{cases} -dY_t &= \left(\|Z_t\|^2_{\mathbb{R}^{1\times d}} - \|\nabla\psi(t,W_t)\|^2_{\mathbb{R}^d} - \left(\partial_t + \frac{1}{2}\Delta\right)\psi(t,W_t)\right)\,dt - Z_t\,dW_t, \\ Y_T &= \sin\left(\|W_T\|^{2\alpha}_{\mathbb{R}^d}\right), \end{cases}$$

where $\psi(t,W_t) = \sin\left(\left(T-t+\|W_t\|^2_{\mathbb{R}^d}\right)^\alpha\right)$. The analytic solution is

$$\begin{cases} Y_t &= \sin\left(\left(T-t+\|W_t\|^2_{\mathbb{R}^d}\right)^\alpha\right), \\ Z_t &= 2\alpha W_t^\top \cos\left(\left(T-t+\|W_t\|^2_{\mathbb{R}^d}\right)^\alpha\right)\left(T-t+\|W_t\|^2_{\mathbb{R}^d}\right)^{\alpha-1}. \end{cases}$$

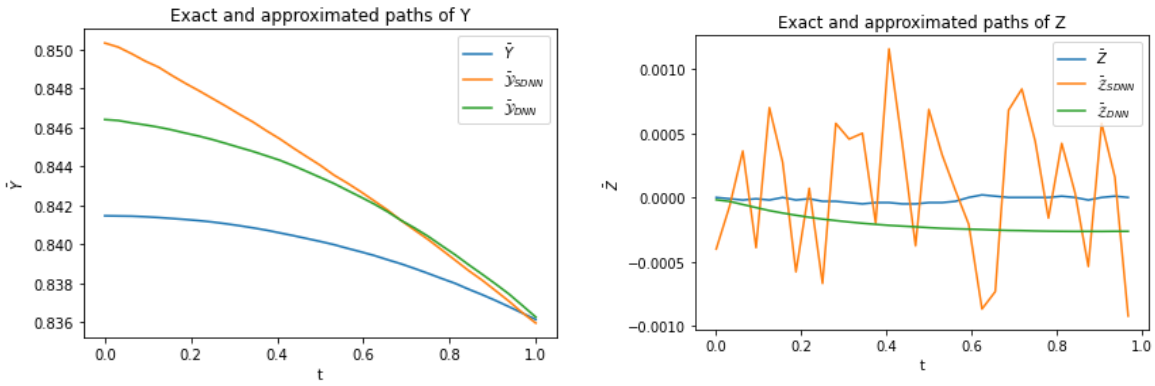
The exact solution with $d = 50$, $T = 1$ and $\alpha = 0.4$ is $(Y_0, Z_0) \doteq (0.8415, (0, \cdots, 0))$. We choose a learning rate of $5e-3$, set number of iterations for the optimizer to $m = 4000$ and batch size to be 64 for both approaches. We report the results in Table 3 and 4 for the SDNN-approach and the DNN-approach respectively. We observe that the DNN-approach (less computational time) gives comparable results for Y as the SDNN-approach, and slightly better approximation for Z.

Table 3: The results by the SDNN-approach for Example 2 with $d = 50$.

N	$\mathcal{Y}_0$	ϵ_{Y_0}	$s(\epsilon_{Y_0})$	$\boldsymbol{\mathcal{Z}}_0$	ϵ_{Z_0}	$s(\epsilon_{Z_0})$	Time
8	0.8407	0.0012	0.0006	0.0001	0.0036	0.0003	140.57
16	0.8431	0.0018	0.0011	-0.0000	0.0052	0.0005	283.38
32	0.8503	0.0089	0.0025	-0.0004	0.0090	0.0016	634.56

Table 4: The results by the DNN-approach results for Example 2 with $d = 50$.

N	$\mathcal{Y}_0$	ϵ_{Y_0}	$s(\epsilon_{Y_0})$	$\boldsymbol{\mathcal{Z}}_0$	ϵ_{Z_0}	$s(\epsilon_{Z_0})$	Time	Speedup
8	0.8452	0.0037	0.0016	-0.0000	0.0030	0.0004	67.96	2.07
16	0.8461	0.0046	0.0018	-0.0001	0.0036	0.0005	132.92	2.13
32	0.8464	0.0049	0.0017	-0.0000	0.0040	0.0006	296.28	2.14



(a) Y process.

(b) Z process.

Figure 4: The comparison of averages of the exact path $\bar{Y}, \bar{Z}$ and the approximated paths $\bar{\mathcal{Y}}, \bar{\mathcal{Z}}$ using the SDNN- and DNN-approach for Example 2 with $d = 50$.

To show this, as in the previous example, we display the averages of paths for both processes of each approach in Figure 4, where $N = 32$.
In order to compare both the approaches in a higher dimension, we set $d = 100$. The results are reported in Table 5 and 6 for the SDNN- and DNN-approach, respectively. Furthermore, the averages of paths are also displayed in Figure 5[1], where $N = 32$. We observe that the approximations of Z is still slightly better in the DNN-approach. However, the approximations of Y is better in the SDNN-approach.
In order to further confirm whether the SDNN-approach performs always better than the DNN-approach to approximate Y in the case of that $d = 100$, we consider the problem of option pricing with different interest rates, which have been considered in e.g., [E et al., 2017, E et al., 2019, Teng, 2019], and option pricing with default risk.

Example 3. Consider the option pricing FBSDE with different interest rates [Bergman, 1995]

$$\begin{cases} dS_t &= \mu S_t\, dt + \sigma S_t\, dW_t, \quad S_0 = S_0, \\ -dY_t &= -R^l Y_t - \frac{\mu - R^l}{\sigma} \sum_{i=1}^d Z_t^i + \left(R^b - R^l\right) \max\left(\frac{1}{\sigma}\sum_{i=1}^d Z_t^i - Y_t, 0\right) dt - Z_t\, dW_t, \\ Y_T &= \max\left(\max_{d=1,\cdots,D}(S_{T,d} - K_1, 0) - 2\max\left(\max_{d=1,\cdots,D}(S_{T,d} - K_2, 0\right), \end{cases}$$

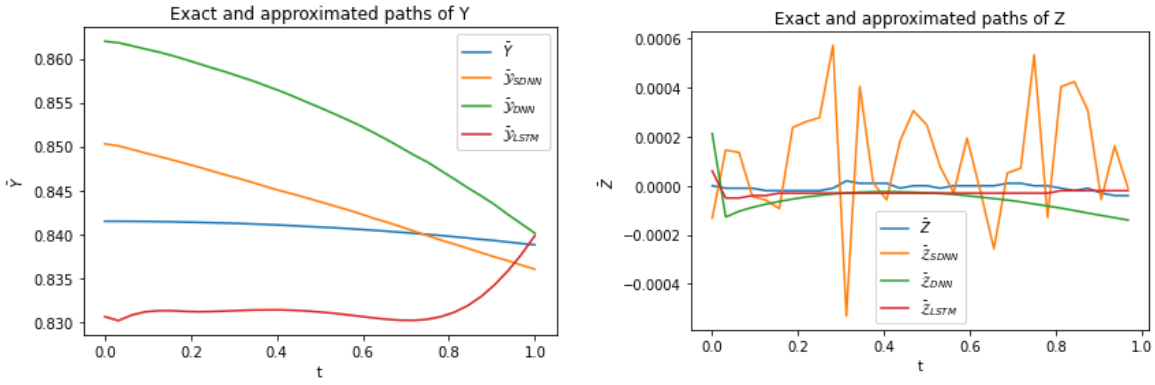
[1]The results with the LSTM-approach will be analyzed in the following sections.

Table 5: The results by the SDNN-approach for Example 2 with $d = 100$.

N	$\mathcal{Y}_0$	ϵ_{Y_0}	$s(\epsilon_{Y_0})$	$\boldsymbol{\mathcal{Z}}_0$	ϵ_{Z_0}	$s(\epsilon_{Z_0})$	Time
8	0.8418	0.0005	0.0004	-0.0001	0.0022	0.0002	388.04
16	0.8434	0.0019	0.0009	0.0003	0.0032	0.0002	739.02
32	0.8503	0.0088	0.0009	-0.0001	0.0068	0.0006	1180.95

Table 6: The results by the DNN-approach results for Example 2 with $d = 100$.

N	$\mathcal{Y}_0$	ϵ_{Y_0}	$s(\epsilon_{Y_0})$	$\boldsymbol{\mathcal{Z}}_0$	ϵ_{Z_0}	$s(\epsilon_{Z_0})$	Time	Speedup
8	0.8621	0.0206	0.0056	-0.0001	0.0040	0.0004	203.93	1.90
16	0.8610	0.0195	0.0041	0.0000	0.0044	0.0004	434.52	1.70
32	0.8620	0.0205	0.0051	0.0002	0.0055	0.0007	743.81	1.59



(a) Y process.

(b) Z process.

Figure 5: The comparison of averages of the exact path $\bar{Y}, \bar{Z}$ and the approximated paths $\bar{\mathcal{Y}}, \bar{\mathcal{Z}}$ using SDNN-, DNN- and LSTM-approach for Example 2 with $d = 100$.

where $S_t = (S_t^1, S_t^2, \cdots, S_t^d)^\top$. 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 [E et al., 2019]. In this example, we use a learning rate of $5e-2$, set number of iterations for the optimizer to $m = 4000$ and batch size to be 64. We present the results in Table 7 and 8 for the SDNN-approach and the DNN-approach, respectively.

Table 7: The results by the SDNN-approach for Example 3.

N	$\mathcal{Y}_0$	$\lvert Y_0 - \mathcal{Y}_0 \rvert$	$s(\lvert Y_0 - \mathcal{Y}_0 \rvert)$	Time
8	20.9536	0.3452	0.1047	387.58
16	21.0679	0.2309	0.1008	748.90
32	21.0961	0.2027	0.0725	1433.66

Table 8: The results by the DNN-approach for Example 3.

N	$\mathcal{Y}_0$	$\lvert Y_0 - \mathcal{Y}_0\rvert$	$s(\lvert Y_0 - \mathcal{Y}_0\rvert)$	Time	Speedup
8	20.9283	0.3705	0.0620	193.92	2.00
16	21.0183	0.2805	0.0652	450.65	1.66
32	21.0583	0.2405	0.0335	822.40	1.74

Example 4. Consider the option pricing FBSDE with default risk [Han et al., 2018]

$$\begin{cases} dS_t &= \mu S_t\, dt + \sigma S_t\, dW_t, \quad S_0 = S_0, \\ -dY_t &= -Y_t \left((1-\delta)\left(\max\left(\max\left(Y_t - v^h, 0\right)\frac{\gamma^h - \gamma^l}{v^h - v^l} + \gamma^h - \gamma^l, 0\right) + \gamma^l\right) + R\right) dt - Z_t\, dW_t, \\ Y_T &= \min_{i=1,\cdots,d}\left(S_T^i\right), \end{cases}$$

The benchmark value with $T = 1$, $\delta = 1.5$, $R = 0.02$, $\mu = 0.02$, $\sigma = 0.2$, $v^h = 50$, $v^l = 70$, $\gamma^h = 0.2$, $\gamma^l = 0.02$ and $S_0 = 100$ is $Y_0 \doteq 57.300$, which is computed using the multilevel Picard approach ([E et al., 2019]). We use the same hyperparameters as those in Example 3 and report the results in Table 9 and 10 for the SDNN-approach and the DNN-approach, respectively.

Fortunately, in Example 3 and 4, we see the comparable approximations for Y using both

Table 9: The results by the SDNN-approach for Example 4.

N	$\mathcal{Y}_0$	$\lvert Y_0 - \mathcal{Y}_0\rvert$	$s(\lvert Y_0 - \mathcal{Y}_0\rvert)$	Time
8	56.2321	1.0679	0.0610	281.08
16	56.8045	0.4955	0.0738	582.82
32	57.1297	0.1703	0.0676	1078.46

Table 10: The results by the DNN-approach for Example 4.

N	$\mathcal{Y}_0$	$\lvert Y_0 - \mathcal{Y}_0\rvert$	$s(\lvert Y_0 - \mathcal{Y}_0\rvert)$	Time	Speedup
8	56.1079	1.1921	0.1224	186.60	1.51
16	56.6225	0.6775	0.1180	436.80	1.33
32	56.8515	0.4485	0.2002	902.64	1.20

the approaches for $d = 100$, and the less computational cost in the DNN-approach is again highlighted. With our numerical analysis above we conclude that the DNN-approach, i.e., using only one neural network, can give comparable approximation for Y, and better approximation for Z (especially for the whole time domain) than the SDNN-approach for less computational cost.

As introduced before, when the solution has extremely complex structure, both the SDNN- and DNN-approach can be stuck in poor local minima or divergence. To tackle this problem we have proposed the LSTM-approach. We consider Example 5 where the SDNN- and DNN-approach can not converge.

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

$$\begin{cases} dX_t &= \mu\, dt + \sigma\, dW_t, \quad X_0 = x_0, \\ -dY_t &= \Big(\cos\left(\bar{X}\right)\Big(\exp\left(\frac{T-t}{2}\right) + \frac{\sigma^2}{2}\Big) + \mu\sin\left(\bar{X}\right)\Big)\exp\left(\frac{T-t}{2}\right) \\ & \quad -\frac{1}{2}\left(\sin\left(\bar{X}\right)\cos\left(\bar{X}\right)\exp\left(T-t\right)\right)^2 + \frac{1}{2}\left(Y_t\bar{Z}\right)^2\Big)\, dt - Z_t\, dW_t, \\ Y_T &= \cos\left(\bar{X}\right), \end{cases}$$

where $\bar{X} = \sum_{i=1}^{d} X_t^i$ and $\bar{Z} = \sum_{i=1}^{d} Z_t^i$. And the analytic solution is given by

$$\begin{cases} Y_t &= \exp\left(\frac{T-t}{2}\right)\cos\left(\bar{X}\right), \\ Z_t &= -\sigma\exp\left(\frac{T-t}{2}\right)\sin\left(\bar{X}\right)\mathbb{1}_{\mathbb{R}^d}. \end{cases}$$

We start with $d = 1$, the exact solution with $T = 2$, $\mu = 0.2$, $\sigma = 1$ and $x_0 = 1$ is $(Y_0, Z_0) \doteq (1.4687, -2.2874)$. We consider use the same hyperparameters for the SDNN-and LSTM-approach: learning rate of $1e-2$, $m = 1000$ and batch size of 64. In our tests, the LSTM architecture with automatic differentiation fits well the new loss formulation. Using another network architecture, e.g. as those in the SDNN- and DNN-approach, or using the LSTM without automatic differentiation, doesn't improve the results. It is important to note that the SDNN-approach still gets stuck in poor local minima or diverge by increasing the number of epochs. The results are given in Table 11 and 12 for the SDNN- and LSTM-approach, respectively.

Table 11: The results by the SDNN-approach for Example 5 with $d = 1$.

N	$\mathcal{Y}_0$	ϵ_{Y_0}	$s(\epsilon_{Y_0})$	$\boldsymbol{\mathcal{Z}}_0$	ϵ_{Z_0}	$s(\epsilon_{Z_0})$	Time
8	4.5356	3.0669	0.5538	-0.1582	2.1292	0.0473	18.87
16	nan	nan	nan	nan	nan	nan	nan
32	nan	nan	nan	nan	nan	nan	nan
64	nan	nan	nan	nan	nan	nan	nan

Table 12: The results by the LTSM-approach for Example 5 with $d = 1$.

N	$\mathcal{Y}_0$	ϵ_{Y_0}	$s(\epsilon_{Y_0})$	$\boldsymbol{\mathcal{Z}}_0$	ϵ_{Z_0}	$s(\epsilon_{Z_0})$	Time
8	1.2044	0.2783	0.1502	-1.5655	0.7219	0.0222	19.28
16	1.3604	0.1552	0.0893	-1.7355	0.5518	0.0192	37.40
32	1.4019	0.1705	0.1639	-1.8397	0.4476	0.0742	72.09
64	1.3062	0.3090	0.2508	-1.8786	0.4088	0.1627	146.82

Clearly, while the SDNN-approach diverges, the LSTM-approach gives the stable results.
We also test this example when $d = 100$, the exact solution with $T = 1$, $\mu = \frac{0.2}{d}$, $\sigma = \frac{1}{\sqrt{d}}$ and $x_0 = 1$ is $(Y_0, Z_0) \doteq (1.4217, (0.0835, \cdots, 0.0835))$. We use the same hyperparameters as those in the case of that $d = 1$. The results are reported in Table 13 and 14. We conclude that the proposed LSTM-approach can solve the local minimum problem.
Finally, to see how well does the LSTM-approach work for a general nonlinear high dimensional problem (not so complex structure), we run the LSTM-algorithm for Example 2 for $d = 100$ and Example 4. We just use the hyperparameters as those used in these exmaples above, i.e., without the hyperparameter optimization. The results are presented in Table 15 and 16, respectively, which are quite promising. We observe that the approximation for Y is comparable to that by using the SDNN- and DNN-approach. And the approximation of Z is more stable in all the time steps than both the SDNN- and DNN-approach, this has been shown in Figure 5.

Table 13: The results by the SDNN-approach for Example 5 with $d = 100$.

N	$\mathcal{Y}_0$	ϵ_{Y_0}	$s(\epsilon_{Y_0})$	$\boldsymbol{\mathcal{Z}}_0$	ϵ_{Z_0}	$s(\epsilon_{Z_0})$	Time
8	2.8730	1.4512	0.0943	0.0933	0.0297	0.0028	102.41
16	2.8537	1.4320	0.0803	0.0962	0.0327	0.0020	197.07
32	2.9854	1.5636	0.1866	0.0838	0.0353	0.0025	391.88
64	3.4819	2.0601	0.2943	0.0414	0.0663	0.0088	750.17

Table 14: The results by the LTSM-approach for Example 5 with $d = 100$.

N	$\mathcal{Y}_0$	ϵ_{Y_0}	$s(\epsilon_{Y_0})$	$\boldsymbol{\mathcal{Z}}_0$	ϵ_{Z_0}	$s(\epsilon_{Z_0})$	Time
8	2.0506	0.6289	0.0156	0.0001	0.0834	0.0004	120.40
16	1.8866	0.4649	0.0289	-0.0001	0.0836	0.0004	228.88
32	1.5747	0.2205	0.0603	0.0002	0.0833	0.0004	438.06
64	1.3145	0.1120	0.0805	-0.0000	0.0835	0.0010	831.47

Table 15: The results by the LSTM-approach for Example 2 with $d = 100$.

N	$\mathcal{Y}_0$	ϵ_{Y_0}	$s(\epsilon_{Y_0})$	$\boldsymbol{\mathcal{Z}}_0$	ϵ_{Z_0}	$s(\epsilon_{Z_0})$	Time
8	0.8392	0.0023	0.0007	0.0000	0.0024	0.0003	441.11
16	0.8356	0.0059	0.0010	-0.0001	0.0027	0.0004	824.62
32	0.8306	0.0109	0.0023	0.0001	0.0002	0.0002	1527.73

Table 16: The results by the LSTM-approach for Example 4.

N	$\mathcal{Y}_0$	$\lvert Y_0 - \mathcal{Y}_0\rvert$	$s(\lvert Y_0 - \mathcal{Y}_0\rvert)$	Time
8	56.1480	1.1520	0.0309	497.99
16	56.4575	0.8425	0.0648	838.68
32	56.9438	0.4049	0.2474	1248.72

7 Conclusions

In this work we have proposed the DNN-approach to improve the performances of the SDNN-approach proposed in [E et al., 2017] in terms of computational time, and the LSTM-approach to overcome the poor local minima or divergence problem which could appear in both the SDNN- and DNN-approach, especially when the solution has extremely complex structure. With our numerical results we find that the LSTM-approach works generally well, i.e., without local minima problem, and the DNN-approach is the best choice due to its efficiency when the solution structure is not so complex. Furthermore, our both approaches give more stable approximation for Z in each time step than the SDNN-approach. 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.

[Crisan and Manolarakis,] Crisan, D. and Manolarakis, K. Solving backward stochastic differential equations using the cubature method: Application to nonlinear pricing. *SIAM J. Financial Math.*, 3(1):534–571.

[E et al., 2017] E, W., 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.

[E et al., 2019] E, W., 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.

[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., and Warin, X. (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., Zhou, T., and and (2017). Efficient spectral sparse grid approximations for solving multi-dimensional forward backward SDEs. *Discrete Continuous Dyn. Syst. Ser. B*, 22(9):3439–3458.

[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., and Warin, X. (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 GPUs. *SIAM J. Sci. Comput.*, 38(6):C652–C677.

[Gobet and Turkedjiev, 2015] Gobet, E. and Turkedjiev, P. (2015). Linear regression MDP scheme for discrete backward stochastic differential equations under general conditions. *Math. Comp.*, 85(299):1359–1391.

[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*.

[Han et al., 2018] Han, J., Jentzen, A., and E, W. (2018). Solving high-dimensional partial differential equations using deep learning. *Proc. Natl. Acad. Sci. U.S.A.*, 115(34):8505–8510.

[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*.

[Karoui et al., 1997] Karoui, N. E., Peng, S., and Quenez, M. C. (1997). Backward stochastic differential equations in finance. *Math. Financ.*, 7(1):1–71.

[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, 1992] Kloeden, P. E. and Platen, E. (1992). *Numerical Solution of Stochastic Differential Equations*. Springer Berlin Heidelberg.

[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., and Warin, X. (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, 2016] Ruijter, M. and Oosterlee, C. (2016). Numerical fourier method and second-order taylor scheme for backward SDEs in finance. *Appl. Numer. Math.*, 103:1–26.

[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.

[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 Gnther, M. (2020). A multi-step scheme based on cubic spline for solving backward stochastic differential equations. *Appl. Numer. Math.*, 150:117–138.

[Zhang, 2013] Zhang, G. (2013). A sparse-grid method for multi-dimensional backward stochastic differential equations. *J. Comput. Math.*, 31(3):221–248.

[Zhang, 2004] Zhang, J. (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.