The Effect of the Number of Neural Networks on Deep Learning Schemes for Solving High Dimensional Nonlinear Backward Stochastic Differential Equations

Lorenc Kapllani

Abstract We consider the deep learning based scheme proposed in [W. E and J. Han and A. Jentzen, Commun. Math. Stat., 5 (2017), pp. 349–380] and study the effect of the number of neural networks on the gradient of the solution. We demonstrate that using one neural network improves its numerical stability for the whole path and also reduces the computational time. This is illustrated with several 100-dimensional nonlinear backward stochastic differential equations including nonlinear pricing problems in finance.

1 Introduction

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

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

where $X_t, \mu \in \mathbb{R}^n$, σ is a $n \times d$ matrix, $W_t = \left(W_t^1, \ldots, W_t^d\right)^\top$ is a d-dimensional Brownian motion, $f(t, X_t, Y_t, Z_t) : [0, T] \times \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^{m \times d} \to \mathbb{R}^m$ is the driver function and ξ is the terminal condition. The existence and uniqueness of the solution of (1) are proven in [10]. In the sequel of this work, we investigate the effect of the number of neural networks in [4] that solve (1).

In the recent years, many numerical methods have been proposed for solving BSDEs, e.g., [1, 11, 15], which are not suitable for high-dimensional problems

L. Kapllani (✉)
Bergische Universität Wuppertal, Wuppertal, Germany
e-mail: kapllani@math.uni-wuppertal.de

M. Ehrhardt, M. Günther (eds.), *Progress in Industrial Mathematics at ECMI 2021*,
The European Consortium for Mathematics in Industry 39,
https://doi.org/10.1007/978-3-031-11818-0_10

as in physics or finance [9] (also sparse-grids [14] or parallel computing [6, 8]) due to exponential increase of algorithm complexity. Recently machine learning schemes show that they can deal with high dimensions for reasonable computational time [4, 5, 12, 13]. We study the well known deep learning based algorithm in [4] (we refer to it as SDNN-approach in the rest of the paper, where SDNN stands for Stacked Deep Neural Networks) where the gradient of the solution (process Z) is approximated by fully-connected neural networks. The authors in [3] analyzed it using different architectures. However, no result was shown regarding the instability of Z for SDNN-approach due to the use of different deep neural networks at each time layer. To have a more numerically stable algorithm, we study the effect of reducing the number of neural networks. In the sequel, we refer this scheme as DNN-approach.

The outline of the paper is organized as follows. In the next section, we describe the SDNN- and DNN-approaches. In Sect. 3, we illustrate our findings with several numerical tests. Section 4 concludes this work.

2 The SDNN-Approach and DNN-Approach

The Feynman-Kac formula and forward discretization of FBSDE are needed to formulate the FBSDE as a learning problem. 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 [9]

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

where $u(t,x)$ is the solution of the following semi-linear parabolic PDE:

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

with $u(T,x) = g(x)$. This is the Feynman-Kac formula. Using

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

and the notation $X_i = X_{t_i}$, $W_i = W_{t_i}$, $\Delta W_i = W_{i+1} - W_i$ and the approximated process as $X_i^\Delta = X_{t_i}^\Delta$, the discretization of (1) using the well-known Euler scheme is

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

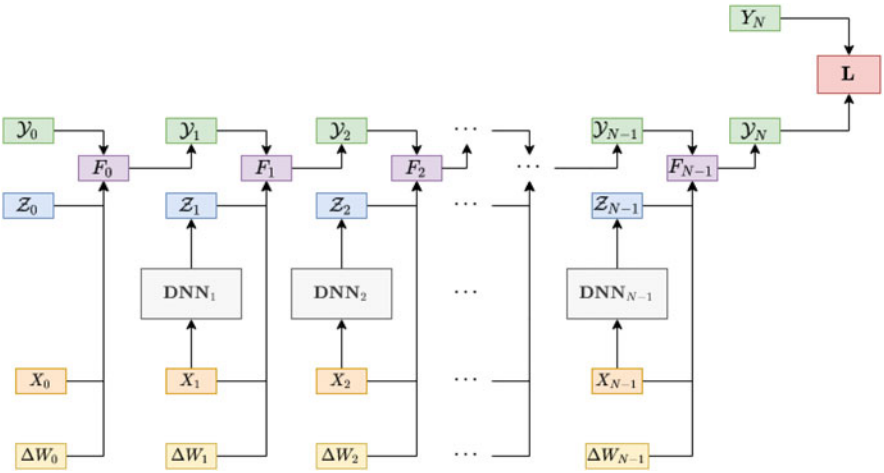

Fig. 1 Graph of the SDNN-approach

and

$$\begin{aligned} Y_{i+1}^{\Delta} &= Y_i^{\Delta} - f\left(t_i, X_i^{\Delta}, Y_i^{\Delta}, Z_i^{\Delta}\right)\Delta t + Z_i^{\Delta}\Delta W_i, \\ &:= F(t_i, X_i^{\Delta}, Y_i^{\Delta}, Z_i^{\Delta}, \Delta t, \Delta W_i). \end{aligned} \tag{3}$$

where $i = 0, 1, \ldots, N-1$ and $\Delta W_i \sim \mathcal{N}(0,\ \Delta t)$.

The numerical approximation of (Y^{Δ}, Z^{Δ}) in the SDNN-approach (Fig. 1) is designed as follows: starting from an estimation $(\mathcal{Y}_0(\theta), \mathcal{Z}_0(\theta))$ of $(Y_0^{\Delta}, Z_0^{\Delta})$, and then using at each time step t_i, $i = 1, 2, \ldots, N-1$ a different feedforward deep neural network $\psi_{i,k,L}^{\varrho}(x;\theta)\colon \mathbb{R}^d \to \mathbb{R}^{1\times d}$ to approximate Z_i^{Δ} as $\mathcal{Z}_i(\theta)$ and Y_i^{Δ}, $i = 1, 2, \ldots, N$ as $\mathcal{Y}_i(\theta)$ with (3), where the output $\mathcal{Y}_N(\theta)$ aims to match the terminal condition $g(X_T^{\Delta})$ of the BSDE (using Adam gradient descent-type optimizer with mini-batches):

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

Note that i represents the i-th network, k is the number of neurons, L are the number of hidden layers, ϱ is the activation function, the input x of the network is the Markovian process X_i^{Δ} and θ are network parameters. Specifically, the networks have 4 global layers, where hidden layers have d+10 neurons, the rectifier function $\mathbb{R} \ni x \to \max\{0, x\} \in [0, \infty)$ is used as the activation function, the weights are initialized using a normal or a uniform distribution and batch normalization is also used. For the DNN-approach, we consider $p < N-1$ networks, i.e. 1 network for consecutive subintervals, with input x having the time discretization t_i (to handle non-stationarities) and the Markovian process X_i^{Δ} (due to Feynman-Kac formula), with 6 global layers (2 hidden layers more than SDNN-approach for

Table 1 The dimension of the parameters

SDNN-approach	$d + 1 + (N-1)(2d(d+10) + (d+10)^2 + 4(d+10) + 2d)$
DNN-approach	$p\left(2d + 1 + (2d+5)(d+10) + 3(d+10)^2\right)$

a better accuracy). For sufficiently regular solutions, the gradient between two time points should be close. Therefore, the numerical stability of Z is affected from the number of DNNs. The dimension of the parameters $\rho \in \mathbb{N}$ for both approaches are given in Table 1. The complexity in the DNN-approach is lower for less number of networks, namely p.

3 Numerical Results

In this section we study the DNN-approach by comparing it to the SDNN-approach in several high dimensional examples. The results are presented using 10 independent runs with Tensorflow 1.15 from Google Colab. We start with an example with analytical solution where the driver function depends on Y and Z.

Example 1 Consider the Burgers type FBSDE [4]

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

where $W_t = (W_t^1, W_t^2, \cdots, W_t^d)^\top$, $X_t = (X_t^1, X_t^2, \cdots, X_t^d)^\top$, $Z_t = (Z_t^1, Z_t^2, \cdots, Z_t^d)$. The exact solution is $(Y_0, Z_0) \doteq (0.5, (0.1768, \cdots, 0.1768))$ with $d = 50$, $T = 0.2$ and $\sigma = \frac{d}{\sqrt{2}}$. We consider the same hyperparameters for both the SDNN- and DNN-approach, where the learning rate is $1e-2$, 8000 optimization iterations, 256 validation sample and a batch size of 64, which are used also for next examples if not specified. The authors in [4] used different hyperparameters and discretization values. The results are reported in Table 2 for $N = 40$. Note that $p = N - 1$ represents the SDNN-approach, $|\cdot|$ is the absolute value and $s(\cdot)$ represents the standard deviation. Moreover, $\epsilon_{Y_0} = |Y_0 - \mathcal{Y}_0|$, $\mathbf{Z}_0 = \frac{1}{d}\sum_{i=1}^{d} \mathcal{Z}_0^i$ and $\epsilon_{Z_0} = \frac{\sum_{i=1}^{d} |Z_0^i - \mathcal{Z}_0^i|}{d}$.

From Table 2 we observe that the DNN-approach with one network gives higher accuracy for both processes Y and Z, for less computation time. Increasing the number of networks worsens the performance. 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 $\bar{\mathcal{Z}}$ in Fig. 2, where the average over the dimension is also considered for the Z process, in order to have one value at each time point.

Table 2 The results for Example 1

p	$\mathcal{Y}_0$	ϵ_{Y_0}	$s(\epsilon_{Y_0})$	$\mathbf{Z}_0$	ϵ_{Z_0}	$s(\epsilon_{Z_0})$	Time
1	0.5196	0.0350	0.0198	0.1983	0.1250	0.0909	743.36
2	0.5409	0.1021	0.0983	0.3273	0.1814	0.1399	798.12
4	0.5312	0.1255	0.0651	0.5721	0.5092	0.2578	791.48
N − 1	1.2425	0.7425	0.0552	2.8510	2.6743	0.0290	1721.33

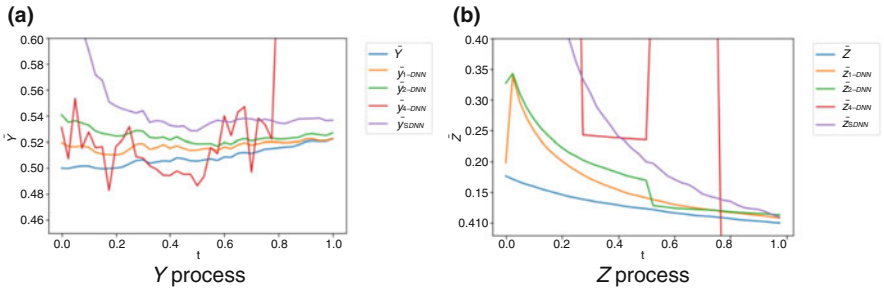

Fig. 2 The comparison of averages of the exact path $\bar{Y}$, $\bar{Z}$ and the approximated paths $\bar{\mathcal{Y}}$, $\bar{\mathcal{Z}}$ for Example 1

For a better view of the approximation of the whole path, we limit the axis for Y and Z, since some results are far from the exact solution. We see that the DNN-approach with one network approximates paths of both processes much better 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 [7]

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

where $\psi(t, W_t) = \sin\left(\left(T - t + \|W_t\|^2_{\mathbb{R}^d}\right)^{\alpha}\right)$. The exact solution is $(Y_0, Z_0) \doteq (0.8415, (0, \cdots, 0))$ with $d = 100$, $T = 1$ and $\alpha = 0.4$. Here we choose $d = 100$ to compare both the approaches in a higher dimension. We set the optimization iterations to $m = 4000$, and report the results in Table 3 with $N = 40$. We observe that the DNN-approach with one network gives again better results for both processes, even for the whole path displayed in Fig. 3.

Finally we consider an example without analytical solution in the case of $d = 100$, the problem of option pricing with different interest rates, also studied in [4, 5, 12].

Example 3 Consider the different interest rates option pricing FBSDE [2]

Table 3 The results for Example 2

p	$\mathcal{Y}_0$	ϵ_{Y_0}	$s(\epsilon_{Y_0})$	$\mathbf{Z}_0$	ϵ_{Z_0}	$s(\epsilon_{Z_0})$	Time
1	0.8583	0.0168	0.0077	0.0000	0.0087	0.0027	1197.47
2	0.8813	0.0398	0.0146	−0.0001	0.0120	0.0027	1161.98
4	0.9640	0.1226	0.0199	−0.0005	0.0188	0.0021	1304.86
N − 1	1.2541	0.4126	0.0247	−0.0004	0.0381	0.0037	1493.97

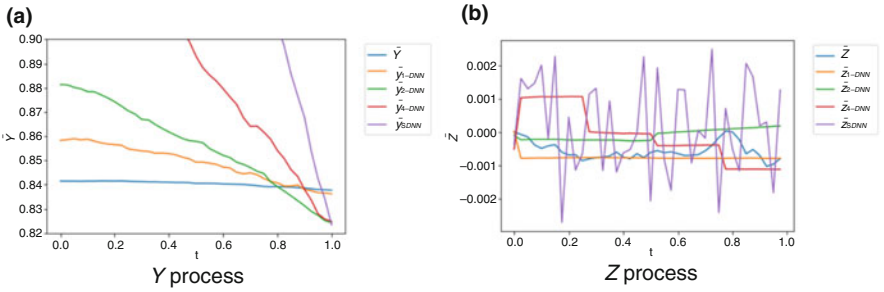

Fig. 3 The comparison of averages of the exact path $\bar{Y}$, $\bar{Z}$ and the approximated paths $\bar{\mathcal{Y}}$, $\bar{\mathcal{Z}}$ for Example 2

Table 4 The results for Example 3

p	$\mathcal{Y}_0$	$\lvert Y_0 - \mathcal{Y}_0 \rvert$	$s(\lvert Y_0 - \mathcal{Y}_0 \rvert)$	Time
1	21.0906	0.2082	0.0476	1137.93
2	21.0626	0.2362	0.0971	1144.90
4	21.0284	0.2704	0.1399	1206.71
N − 1	21.1140	0.1848	0.1017	1845.27

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

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 $Y_0 \doteq 21.2988$ [5]. Using a learning rate of $5e-2$ and 4000 optimization iterations, we present the results in Table 4 for $N = 40$, which shows comparable results for DNN-approach with one network and SDNN-approach.

4 Conclusions

In this work we have proposed the DNN-approach to improve the deep learning scheme [4]. With our numerical analysis we demonstrate that the DNN-approach with one neural network can give comparable approximation for Y and better approximation for Z on the whole time domain for lower computational cost.

References

1. Bender, C., Zhang, J.: Time discretization and Markovian iteration for coupled FBSDEs. Ann. Appl. Probab. **18**(1), 143–177 (2008).
2. Bergman, Y.Z.: Option pricing with differential interest rates, Rev. Finan. Stud. **8**(2), 475–500 (1995).
3. Chan-Wai-Nam, Q., Mikael, J., Warin, X.: Machine learning for semi linear PDEs, J. Sci. Comput. **79**(3), 1667–1712 (2019).
4. E, W., Han, J., Jentzen, A.: Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, Commun. Math. Stat. **5**(4), 349–380 (2017).
5. E, W., Hutzenthaler, M., Jentzen, A., Kruse, T.: 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 (2019).
6. Gobet, E., López-Salas, J.G., Turkedjiev, P., Vázquez, C.: Stratified regression Monte-Carlo scheme for semilinear PDEs and BSDEs with large scale parallelization on GPUs, SIAM J. Sci. Comput. **38**(6), C652–C677 (2016).
7. Gobet, E., Turkedjiev, P.: Linear regression MDP scheme for discrete backward stochastic differential equations under general conditions, Math. Comp. **85**(299), 1359–1391 (2015).
8. Kapllani, L., Teng, L.: Multistep schemes for solving backward stochastic differential equations on GPU. arXiv preprint arXiv:1909.13560 (2019).
9. Karoui, N.E., Peng, S., Quenez, M.C.: Backward stochastic differential equations in finance. Math. Finan. **7**(1), 1–71 (1997).
10. Pardoux, E., Peng, S.: Adapted solution of a backward stochastic differential equation, Syst. Control. Lett. **14**(1), 55–61 (1990).
11. Ruijter, M.J., Oosterlee, C.W.: A Fourier cosine method for an efficient computation of solutions to BSDEs, SIAM J. Sci. Comput. **37**(2), A859–A889 (2015).
12. Teng, L.: A review of tree-based approaches to solve forward-backward stochastic differential equations, arXiv preprint arXiv:1809.00325v4 (2019).
13. Teng, L.: Gradient boosting-based numerical methods for high-dimensional backward stochastic differential equations, arXiv preprint arXiv:2107.06673 (2021).
14. Zhang, G.: A sparse-grid method for multi-dimensional backward stochastic differential equations, J. Comput. Math. **31**(3), 221–248 (2013).
15. Zhao, W., Zhang, G., Ju, L.: A stable multistep scheme for solving backward stochastic differential equations, SIAM J. Numer. Anal. **48**(4), 1369–1394 (2010).