DeepBSDE: Discussion of *Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations* by E, Han, Jentzen

Rachel Childers

2025-10-02

Outline

- Motivation
- Forward-Backward Stochastic Differential Equations (FBSDEs)
- Equivalence to PDEs by Noninear Feynman Kac
- Numerical Methods: DeepBSDE
- Comparison: optimal control by HJB, FBSDE, and Pontryagin

Motivation

- Econ & finance optimization problems have special structure
  - Option pricing: terminal instead of initial conditions
  - Investment: terminal and initial conditions
- HJB equation is one representation of optimality conditions
  - "Closed loop" control: find *feedback rule*
  - Amenable to generic PDE solvers like (Deep Galerkin) PINNs
- Equivalent representation: (Stochastic) Pontryagin Maximum Principle
  - "Open loop" control: find *time path*
  - Needs methods that find whole sequence
- Special case of **Nonlinear Feynman-Kac** lemma
  - Certain nonlinear PDEs have representation as FBSDEs
  - Coupled system of SDEs with states solved forward and values and costates solved backwards
- Deep BSDE: find neural network that represents time path
- Benefits: no need for Hessians, and stochastic solution means some integrals can be solved by Monte Carlo

Nonlinear Feynman Kac

- Shows equivalence between PDE and FBSDE representations
- Class of PDEs we can handle: nonlinear parabolic

$$\frac{\partial u}{\partial t} + \mathcal{A}[u](x,t) + h(x, u(x,t), \Sigma^\top (\nabla_x u)(x,t), t) = 0$$

  - Terminal condition $u(x,T) = g(x)$
- $\mathcal{A}[u] := (\nabla_x u)^\top (x,t)\mu(x,t) + \frac{1}{2} Tr(\Delta_x[u](x,t)\Sigma\Sigma^\top)$
- FBSDE connects deterministic PDE to stochastic processes
- $\mathcal{A}[u]$ is infinitesimal generator of a *forward* process
  - $dx(t) = \mu(x,t)dt + \Sigma(x,t)dw(t)$
  - Initial condition: $x(0) = \xi$
- FBSDE representation adds additional *backward* processes
  - Value: $y(t) = u(x(t), t)$
  - Co-state: $z(t) = \Sigma^\top (x(t), t)(\nabla_x u)(x(t), t)$
- Accompany $x(t)$ by a backward process, solved from T to t

$$dy(t) = -h(x(t), y(t), z(t), t)dt + z(t)^\top dw(t)$$

- Terminal condition $y(T) = g(x(T))$

Derivation (from Pardoux and Răşcanu (2014))

- Will show a solution $y(t)$ of the FBSDE is a realization of a solution of the PDE
- Rewrite backward process in integral form

$$y(t) = g(x(T)) + \int_t^T h(x(s), y(s), z(s), s)ds - \int_t^T z^\top(s)dw(s)$$

- Plug in PDE $-h(...) = \frac{\partial u}{\partial t} + \mathcal{A}[u](x,t)$

$$y(t) = g(x(T)) - \int_t^T [\frac{\partial u}{\partial s}(x(s), s) + \mathcal{A}[u](x,s)]ds - \int_t^T z^\top(s)dw(s)$$

- Replacing $z(t) = \Sigma^\top(\nabla_x u)(x(t), t)$, $y(t) = g(x(T)) -$

$$\int_t^T [\frac{\partial u}{\partial s}(x(s), s) + \mathcal{A}[u](x,s)]ds - \int_t^T \Sigma^\top(\nabla_x u)(x(t), t)dw(s)$$

Derivation, ctd

- Recall Ito's lemma: if $dx(t) = \mu(x,t)dt + \Sigma(x,t)dw(t)$, then

$$du(x(t),t) = \frac{\partial u}{\partial t}(x(t),t)dt + \mathcal{A}[u](x,t)dt + \Sigma^\top(\nabla_x u)(x(t),t)dw(t)$$

- In integral form $u(x(T),T) - u(x(t),t) =$

$$\int_t^T [\frac{\partial u}{\partial s}(x,s) + \mathcal{A}[u](x,s)]ds + \int_t^T \Sigma^\top(\nabla_x u)(x(s),s)dw(s)$$

- Replace this term in BSDE $y(t) = g(x(T)) -$

$$\int_t^T [\frac{\partial u}{\partial s}(x(s),s) + \mathcal{A}[u](x,s)]ds - \int_t^T \Sigma^\top(\nabla_x u)(x(t),t)dw(s)$$

- Becomes $y(t) = g(x(T)) - u(x(T),T) + u(x(t),t)$
- Impose terminal condition $g(x(T)) = u(x(T),T)$
- Obtain equivalence: $y(t) = u(x(t),t)$

Computation

- Feynman Kac typically is presented in terms of expectations
- $u(x,t) = E[u(x(t),t)|x(t) = x] = E[y(t)|x(t) = x]$ is deterministic
  - To solve, generate paths starting at x and take average
- Forward part: Simulate using standard SDE solver: $dx(t) = \mu(x(t),t)dt + \Sigma(x(t),t)^\top dw(t)$
- Euler-Maruyama: $x(t+\Delta t) =$
  - $x(t) + \mu(x(t),t)\Delta t + \Sigma(x(t),t)(w(t+\Delta t) - w(t))$
  - Just a left Riemann sum, as in definition of Ito integral
- Backward part: challenging! Where does $z(t)$ come from?
  - $y(t) = u(x(t),t)$, $z(t) = \Sigma^\top(x(t),t)\nabla_x u(x(t),t)$, so z should be a function of the path $x(t)$ just like y.
  - If we *knew* that function, then we could solve for $y(t)$ by solving diffusion backwards from known terminal condition
  - Could use, e.g., reverse Euler-Maruyama
- The problem is, we don't generally know $z(t)$, so we can't solve sequentially

Aside: classical Feynman Kac

- If we can solve out for relationships in closed form, we eliminate need for neural networks
- Special-case: semilinear parabolic PDEs
  - $h(x(t), u(x,t), \nabla_x u(x,t)) = -V(x,t)u(x,t) + f(x,t)$
- Can show
  - $u(x,t) = E[\exp(\int_t^T V(x(s),s))dsg(x(T)) + \int_t^T \exp(\int_t^\tau V(x(s),s))dsf(x(\tau),\tau)d\tau | x(t) = x]$
- In this setting, *only* need to simulate $x(t)$ forward, no backward part
  - Very easy, works for pricing European options!

DeepBSDE: Intuition

- If we don't know $z(t)$ as a function of $x(t)$, just conjecture some function $z(t) = \mathcal{V}(x(t), \theta)$
- Likewise guess an initial condition $\mathcal{U}(\theta)$ for $y(0)$
- Given a guess θ, we can simulate $x(t)$, obtain $z(t; \theta)$, and plug this into SDE to get $y(t; \theta)$ and solve *forwards*
- What can go wrong?
  - You might not hit the terminal condition.
- How to fix it
  - Adjust θ until you do hit terminal condition
  - Set up loss function as $(y(T; \theta) - g(x(T))^2$
- To ensure correct function can be found, use function class with universal approximation property to parameterize $\mathcal{U}(\theta), \mathcal{V}(x(t), \theta)$: *neural networks*!
- Minimize loss by SGD or your favorite gradient-based optimizer

DeepBSDE: algorithm

- Inputs: Learning rate λ, Neural networks $\mathcal{V}(x, \theta^0), \mathcal{U}(\theta^0)$
- Outputs: trained neural networks $\mathcal{V}(x, \theta^M)$ $\mathcal{U}(\theta^M)$
- For $m = 0 \dots M-1$
  - $x(0) = \xi$, $w_0^m = 0$ $y(0) = \mathcal{U}(\theta^m)$
  - For $n = 0 \dots N-1$
    - $w_{n+1}^m = w_n^m + z_n^m$, $z_n^m \overset{iid}{\sim} N(0, \Delta t)$
    - $x_{n+1}^m = x_n^m + \mu(x_n^m, t_n)\Delta t + \Sigma(x_n^m, t_n)^\top (w^m(t_{n+1}) - w^m(t_n))$
    - $y_{n+1}^m = y_n^m - h(x_n^m, y_n^m, \mathcal{V}(x_n^m, \theta), t_n)\Delta t + \mathcal{V}(x_n^m, \theta)(w^m(t_{n+1}) - w^m(t_n))$
  - $\phi^m(\theta) = \|y_N^m - g(x_n^m)\|^2$
  - $\theta^{m+1} = \theta^m - \lambda \nabla_\theta \phi^m(\theta^m)$

DeepBSDE in Pictures

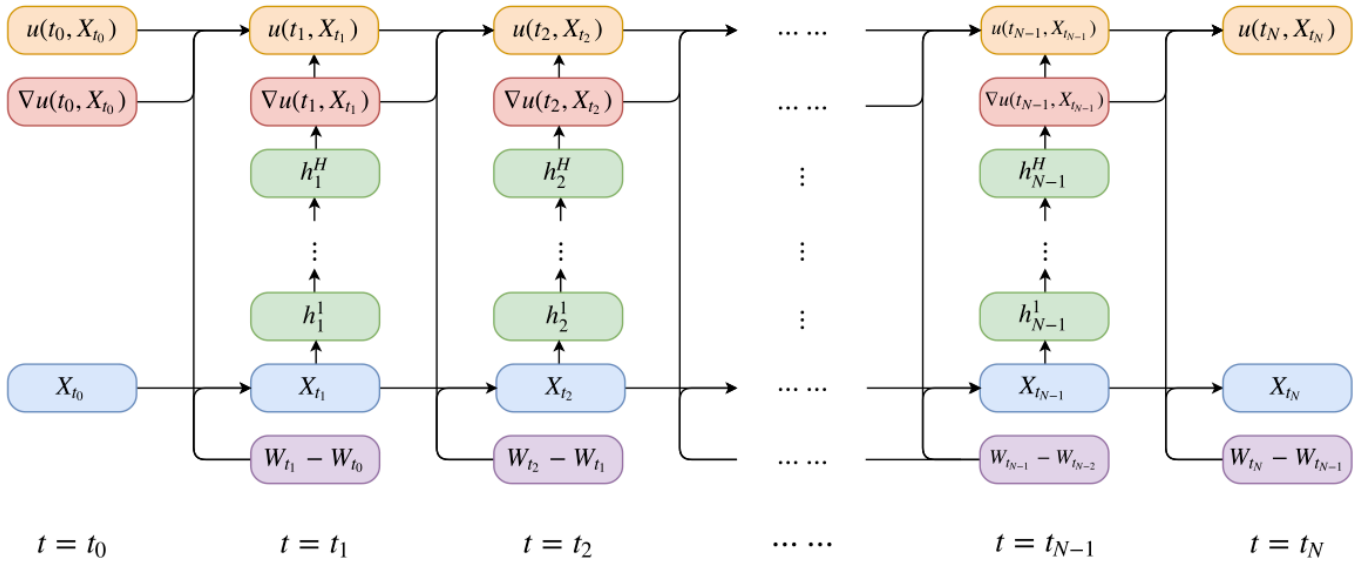


Figure 1: Rough sketch of the architecture of the deep BSDE solver.

Figure 1: Architecture Diagram (E, Han, and Jentzen (2017))

Dimensionality and costs

- Time-dependence of $z(t)$ function adds an input variable to state dimension, but only 1
- Big savings relative to HJB approaches: no second derivatives!
  - Hessians can be computed by Autodiff with a forward and backward pass, but may need mixed mode, which can challenge some AD systems, and in general has quadratic costs in parameter dimension, instead of linear for just gradients (cf "cheap gradient principle" of Griewank and Walther (2008))
- Cost is Monte Carlo integration has $\sqrt{samples}$ convergence rate, so is slow for high accuracy, esp in low dimensions
- Variance of samples can be high depending on process
  - Consider range of variance-reducing tricks common in Monte Carlo including control variates, importance sampling, etc (see Wang et al. (2019))

Performance

- Test case: 100-d HJB
  $\frac{\partial u}{\partial t}(x,t) + (\Delta_x u)(x,t) = \|(\nabla_x u)(x,t)\|^2_{\mathbf{R}^d}$
- Moderate accuracy, moderate speed
- But works in high dimension

Number of iteration steps m	Mean of $\mathcal{U}^{\Theta_m}$	Standard deviation of $\mathcal{U}^{\Theta_m}$	Relative L^1-appr. error	Standard deviation of the relative L^1-appr. error	Mean of the loss function	Standard deviation of the loss function	Runtime in sec. for one realization of $\mathcal{U}^{\Theta_m}$
0	0.3167	0.3059	0.9310	0.0666	18.4052	2.5090	
500	2.2785	0.3521	0.5036	0.0767	2.1789	0.3848	116
1000	3.9229	0.3183	0.1454	0.0693	0.5226	0.2859	182
1500	4.5921	0.0063	0.0013	0.006	0.0239	0.0024	248
2000	4.5977	0.0019	0.0017	0.0004	0.0231	0.0026	330

Table 2: Numerical simulations for the deep BSDE solver in Subsection 3.2 in the case of the PDE (36).

Figure 2: E, Han, and Jentzen (2017) HJB Performance

Optimal Control Application

- For us, PDE will usually be an HJB equation
  - Unknown $u(x,t)$ is a value *function*, dependent on state
- (F)BSDE gives realization of a stochastic process satisfying optimality
  - Unknown is a time path $(x(t), y(t), z(t))$
- Some problems may be easier to set up in one formulation or the other
- Full equivalence easiest to show with some restrictions
  - Control affects drift but not volatility
  - Some separability in drift term
  - Probably these can be relaxed?
- I will show life-cycle consumption-savings problem for concreteness

Optimal Control Setup

- Optimize $\max_{c \in \mathcal{C}} \mathbb{E}_t[\int_t^T e^{-\rho(s-t)} u(c(s))ds + g(a(T))]$
  - s.t. $da(t) = [f(a) + w - c]dt + \Sigma(a,t)dw(t)$, $a(0) = a_0$
- Results in HJB equation
  $\rho V(a,t) = \frac{\partial V}{\partial t}(a,t) + \max_{c \in \mathcal{C}} \mathcal{H}[V](a,t)$

$$\mathcal{H}[V] := \left\{ u(c) + V_a(a,t)[f(a) + w - c] + \frac{1}{2} Tr(V_{aa} \Sigma \Sigma^\top) \right\}$$

- Subject to $V(.,T) = g(a(T))$ and $a(0) = a_0$
- Solve Hamiltonian $\mathcal{H}[V](a,t)$ for c to get optimality
  - $c^*(a,t) = (u')^{-1}(V_a(a,t))$, eg $V_a(a,t)$ for log
- New HJB $\rho V(a,t) = \frac{\partial V}{\partial t}(a,t) + \{u((u')^{-1}(V_a(a,t))) +$

$$V_a(a,t)[f(a) + w - (u')^{-1}(V_a(a,t))] + \frac{1}{2} Tr(V_{aa} \Sigma \Sigma^\top) \Big\}$$

Converting to FBSDE

- You have some freedom to choose what goes into $\mathcal{A}$ vs h
  - State and drift in process need not be (x, μ) in FBSDE
  - Typically control c^* will depend on value, so you'll want to shove that part of drift into h
  - In option-pricing problems, can use process measure because there's no control
- For above problem, fits as

$$\frac{\partial V}{\partial t}(a,t) + \mathcal{A}[V](a,t) + h(a, V, \Sigma^\top V_a)$$

  - $da(t) = [f(a) + w]dt + \Sigma(a,t)dw(t)$
  - $\mathcal{A}[V] = V_a(a,t)[f(a) + w] + \frac{1}{2}Tr(V_{aa}\Sigma\Sigma^\top)$
  - $h(.) = u((u')^{-1}((\Sigma^\top)^{-1}\Sigma^\top V_a(a,t)))$
    $-(\Sigma^\top)^{-1}\Sigma^\top V_a(a,t)(u')^{-1}((\Sigma^\top)^{-1}\Sigma^\top V_a(a,t)) - \rho V(a,t)$

Final FBSDE representation

- Forward law
  - $da(t) = [f(a) + w]dt + \Sigma(a, t)dw(t)$
  - Initial condition $a(0) = a_0$
- Backward law
  - $dy(t) = -h(a(t), y(t), z(t), t)dt + z(t)^\top dw(t)$
  - $h(.) = u((u')^{-1}((\Sigma^\top)^{-1}z(t)))$
    $-(\Sigma^\top)^{-1}z(t)(u')^{-1}((\Sigma^\top)^{-1}z(t)) - \rho y(t)$
  - Terminal condition $y(T) = g(a(T))$

Relation to Pontryagin Maximum Principle

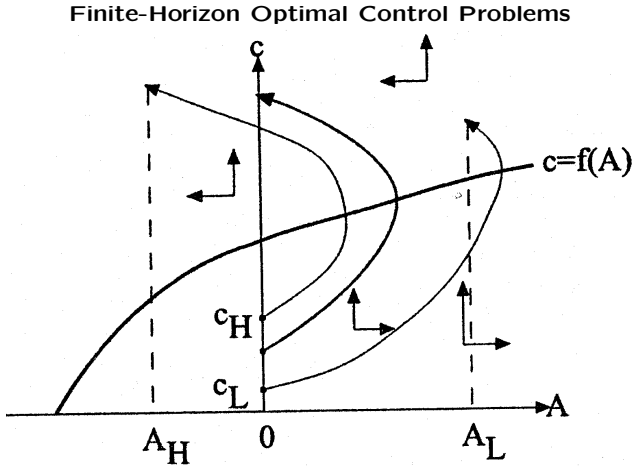
- Classical non-stochastic optimal control represents solution as coupled system of forward and backward ODEs
  - State a evolves forward
  - Co-state λ is V_a, marginal value of state, solved backwards
- Corresponds to FBSDE, except derivative of backward equation taken with respect to a to get dynamics in V_a instead of V
  - Need both when stochastic term added
- Pontryagin maximum principle: optimize Hamiltonian

$$\mathcal{H}(\lambda, a, t) := \{u(c(t)) + \lambda(t)[f(a(t)) + w - c(t)]\}$$

1. Optimality: $c^*(t) = \arg\max_{c \in \mathcal{C}} \mathcal{H}(\lambda, a, t)$
2. Law of motion $\dot{a} = \frac{\partial \mathcal{H}}{\partial \lambda} = f(a) + w - c$
3. Costate evolution: $\dot{\lambda} = \rho\lambda - \frac{\partial \mathcal{H}}{\partial a} = \rho\lambda - \lambda^\top f'(A)$
4. Terminal condition $\lambda(T) = g'(a(T)) = 0$

Optimal Control Numerics

- In non-stochastic setting, solve by shooting
  - Guess initial condition
  - Solve coupled system of ODEs to get terminal state
  - Iterate until terminal condition holds
- Simple special case of DeepBSDE (with no z to find)
  - Low-enough dimensional to only need classical optimizer
- Notoriously sensitive for large T/stiff dynamics
  - Needs high precision solvers
  - Worrisome for deep learning, which is always low precision



Shooting in a life-cycle consumption problem. Source: Judd, K. (1998), Figure 10.2.

Extensions/Applications

- Since E, Han, and Jentzen (2017), method has been widely applied and extended
- Huang (2025) explains method with economic applications
  - Notes you can get value fun by regressing $y(t)$ on $x(t)$
- Applications in control and games
  - Hu and Lauriere (2024)
- Applications to heterogeneous agents
  - Bonnmann and Proehl (2025), Huang (2024)
- Extensions to higher order BSDEs, and methods mixing PDE and SDE approaches
  - Beck, E, and Jentzen (2019), Beck et al. (2021)
- Etc

References I

Beck, Christian, Sebastian Becker, Patrick Cheridito, Arnulf Jentzen, and Ariel Neufeld. 2021. "Deep Splitting Method for Parabolic PDEs." *SIAM Journal on Scientific Computing* 43 (5): A3135–54.

Beck, Christian, Weinan E, and Arnulf Jentzen. 2019. "Machine Learning Approximation Algorithms for High-Dimensional Fully Nonlinear Partial Differential Equations and Second-Order Backward Stochastic Differential Equations." *Journal of Nonlinear Science* 29 (4): 1563–1619.

Bonnmann, Niklas, and Elisabeth Proehl. 2025. "A Global Solution Method for HACT Models with Aggregate Risk."

References II

E, Weinan, Jiequn Han, and Arnulf Jentzen. 2017. "Deep Learning-Based Numerical Methods for High-Dimensional Parabolic Partial Differential Equations and Backward Stochastic Differential Equations." *Communications in Mathematics and Statistics* 5 (4): 349–80.

Griewank, Andreas, and Andrea Walther. 2008. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM.

Hu, Ruimeng, and Mathieu Lauriere. 2024. "Recent Developments in Machine Learning Methods for Stochastic Control and Games." *Numerical Algebra, Control and Optimization* 14 (3): 435–525. https://doi.org/10.3934/naco.2024031.

Huang, Ji. 2024. "Breaking the Curse of Dimensionality in Heterogeneous-Agent Models: A Deep Learning-Based Probabilistic Approach."

References III

———. 2025. “A Probabilistic Solution to High-Dimensional Continuous-Time Macro and Finance Models.”

Pardoux, Étienne, and Aurel Răşcanu. 2014. “Backward Stochastic Differential Equations.” In *Stochastic Differential Equations, Backward SDEs, Partial Differential Equations*, 353–515. Springer.

Wang, Ziyi, Marcus Pereira, Ioannis Exarchos, and Evangelos Theodorou. 2019. “Learning Deep Stochastic Optimal Control Policies Using Forward-Backward SDEs.” In *Robotics: Science and Systems XV*. RSS2019. Robotics: Science; Systems Foundation. https://doi.org/10.15607/rss.2019.xv.070.