PDGM: A Neural Network Approach to Solve Path-Dependent Partial Differential Equations

Yuri F. Saporito

FGV EMAp
ESCOLA DE MATEMÁTICA APLICADA

Joint work with Zhaoyu Zhang (USC)

Bachelier Finance Society One World Seminars
January 28, 2021

Literature Review

- Path-Dependent Partial Differential Equations (**PPDEs**)
  - **Functional Itô Calculus:** *B. Dupire [2009]*
  - Viscosity solution of PPDEs: *I. Ekren, N. Touzi and J. Zhang [2014, 2016ab]*.
  - Monotone schemes for PPDEs: *J. Zhang and J. Zhuo [2014], Z. Ren and X. Tan [2017]*
  - etc.
- **Machine Learning** on PDEs:
  - **Deep Galerkin Method (DGM)**: *J. Sirignano and K. Spiliopoulos [2018]*
  - Using BSDEs: *W. E, J. Han, and A. Jentzen [2017]*
  - Physics-informed neural networks: *M. Raissi and co-authors [2018, 2019]*
  - Deep PPDEs for rough vol: *A. Jacquier and M. Oumgari [2019]*
  - Many others!

Functional Itô Calculus and PPDEs

Functional Itô Calculus

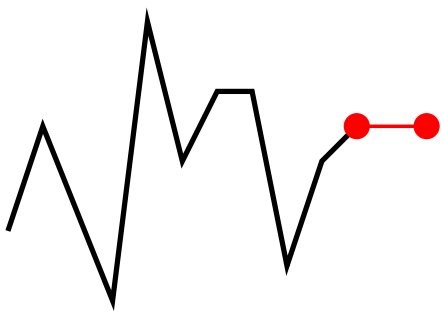
- Time horizon $T > 0$ fixed
- Denote Λ_t the space of càdlàg paths in $[0, t]$ taking values in $\mathbb{R}^n$ and define $\Lambda = \bigcup_{t \in [0,T]} \Lambda_t$
- Capital letters will denote elements of Λ (i.e. paths) and lower-case letters will denote spot value of paths
  - $Y_t \in \Lambda$ means $Y_t \in \Lambda_t$ and $y_s = Y_t(s)$, for $s \leq t$
- We consider here continuity of functionals as the usual continuity in metric spaces with respect to the metric:

$$d_\Lambda(Y_t, Z_s) = \|Y_{t,s-t} - Z_s\|_\infty + |s - t|$$

Functional Itô Calculus

Flat extension

$$Y_{t,\delta t}(u) = \begin{cases} y_u, & \text{if } 0 \le u \le t, \\ y_t, & \text{if } t \le u \le t + \delta t. \end{cases}$$



Bump

$$Y_t^h(u) = \begin{cases} y_u, & \text{if } 0 \le u < t, \\ y_t + h, & \text{if } u = t. \end{cases}$$

Functional Itô Calculus

- A functional is any function $f : \Lambda \longrightarrow \mathbb{R}$. For such objects, we define, when the limits exist, the **time** and **space** functional derivatives, respectively, as

$$\Delta_t f(Y_t) = \lim_{\delta t \to 0^+} \frac{f(Y_{t,\delta t}) - f(Y_t)}{\delta t},$$
$$\Delta_x f(Y_t) = \lim_{h \to 0} \frac{f(Y_t^h) - f(Y_t)}{h}.$$

- Our numerical method is based on the following approximation of the functional derivatives: for a smooth functional, we use

$$\Delta_t f(Y_t) = \frac{f(Y_{t,\delta t}) - f(Y_t)}{\delta t} + o(\delta t),$$
$$\Delta_x f(Y_t) = \frac{f(Y_t^h) - f(Y_t^{-h})}{2h} + o(h^2),$$
$$\Delta_{xx} f(Y_t) = \frac{f(Y_t^h) - 2f(Y_t) + f(Y_t^{-h})}{h^2} + o(h^2).$$

Conditioned Expectation

Consider a functional $g : \Lambda_T \longrightarrow \mathbb{R}$ and define the *conditioned expectation*

$$f(Y_t) = \mathbb{E}\left[g(X_T) \mid Y_t\right] = \mathbb{E}\left[g(X_T^{Y_t})\right],$$

where $X_T^{Y_t}$ is defined as the path from 0 to T of a process starting with the path Y_t and following the dynamics of x:

$$dx_s = \mu(X_s)ds + \sigma(X_s)dw_s.$$

Functional Feynman-Kac Formula; Dupire 2009

Consider functionals $g : \Lambda_T \longrightarrow \mathbb{R}$, $\lambda : \Lambda \longrightarrow \mathbb{R}$ and $k : \Lambda \longrightarrow \mathbb{R}$ and define the functional f as

$$f(Y_t) = \mathbb{E}\left[e^{-\int_t^T \lambda(X_u)du} g(X_T) + \int_t^T e^{-\int_t^s \lambda(X_u)du} k(X_s)ds \;\middle|\; Y_t \right],$$

for any path $Y_t \in \Lambda$, $t \in [0, T]$.

Thus, if f is smooth and k, λ, μ and σ are Λ-continuous, then f satisfies the (linear) PPDE:

$$\Delta_t f(Y_t) + \mu(Y_t)\Delta_x f(Y_t) + \frac{1}{2}\sigma^2(Y_t)\Delta_{xx} f(Y_t) - \lambda(Y_t)f(Y_t) + k(Y_t) = 0,$$

with $f(Y_T) = g(Y_T)$, for any Y_t in the topological support of the stochastic process process x.

PDGM Algorithm - Idea

- Consider the general class of final-value PPDE problem:

$$\begin{cases} \Delta_t f(Y_t) + \mathcal{L} f(Y_t) = 0, \\ f(Y_T) = g(Y_T). \end{cases}$$

- As an illustration, $\mathcal{L}$ could be given by the linear operator

$$\mathcal{L} f(Y_t) = \mu(Y_t)\Delta_x f(Y_t) + \frac{1}{2}\sigma^2(Y_t)\Delta_{xx} f(Y_t) - \lambda(Y_t) f(Y_t) + k(Y_t).$$

- The main idea of our algorithm is to generalize the DGM algorithm to our context.

- We will model the functional f as $u(\cdot;\theta)$ and minimize the loss function

$$L(\theta) = \|\Delta_t u(\cdot;\theta) + \mathcal{L} u(\cdot;\theta)\|_{\Lambda}^2 + \|g - u(\cdot;\theta)\|_{\Lambda_T}^2$$

Modeling functionals using Neural Networks

Feed-Forward Neural Networks

- A set of **layers** $\mathbb{M}^{\rho}_{d,k}$ with input $x \in \mathbb{R}^d$ in a feed-forward neural network can be defined as

$$\mathbb{M}^{\rho}_{d,k} := \{M : \mathbb{R}^d \to \mathbb{R}^k \ ; \ M(x) = \rho(Ax + b), A \in \mathbb{R}^{k \times d}, b \in \mathbb{R}^k\}.$$

- ρ is some activation function such as

$$\rho_{\tanh}(x) := \tanh(x) \text{ and } \rho_s(x) := \frac{1}{1 + e^{-x}}$$

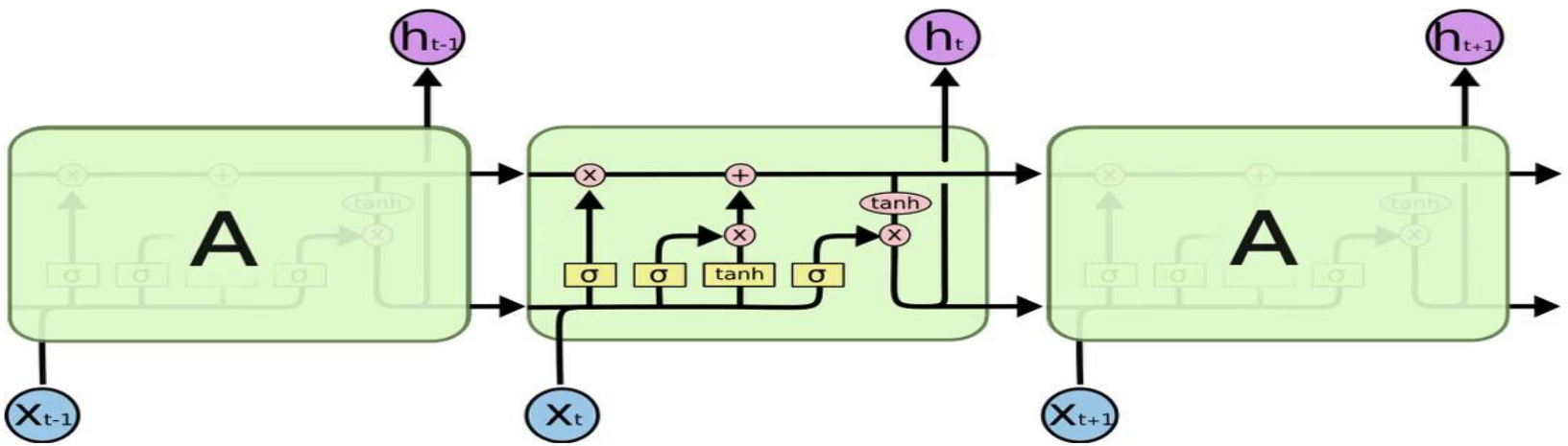
- The set of **feed-forward** neural networks with ℓ hidden layers is defined as a composition of layers:

$$\begin{aligned}\mathbb{NN}^{\ell}_{d_1,d_2} = \{\widetilde{M} : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2} \ ; \ & \widetilde{M} = M_\ell \circ \cdots \circ M_1 \circ M_0, \\ & M_0 \in \mathbb{M}^{\rho}_{d_1,k_1}, M_\ell \in \mathbb{M}^{\rho}_{k_l,d_2}, M_i \in \mathbb{M}^{\rho}_{k_i,k_{i+1}}, \\ & k_i \in \mathbb{Z}^+, i = 1, \ldots, \ell - 1\}.\end{aligned}$$

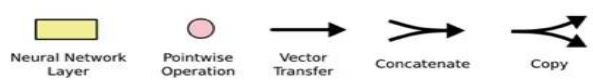
LSTM Network

- The LSTM network, proposed in Hochreiter & Schmidhuber(1997), is designed to solve the **shrinking gradient effects** which basic RNN often suffers from.
- The LSTM network is a **chain of cells**. Each LSTM cell composes of a cell state, which contains information, and three gates, which regulate the flow of information.

Long-Short Term Memory module: LSTM



long-short term memory modules used in an RNN



http://colah.github.io/posts/2015-08-Understanding-LSTMs/ Eugenio Culurciello © 2016

LSTM Network

- Mathematically, the rule inside ***i*th cell** follows,

$$
\begin{aligned}
\textbf{Forget gate}, \quad & \Gamma_{F_i}(x_i, a_{i-1}) = \rho_s(A_F x_i + U_F a_{i-1} + b_F), \\
\textbf{Input gate}, \quad & \Gamma_{I_i}(x_i, a_{i-1}) = \rho_s(A_I x_i + U_I a_{i-1} + b_I), \\
\textbf{Output gate}, \quad & \Gamma_{O_i}(x_i, a_{i-1}) = \rho_s(A_O x_i + U_O a_{i-1} + b_O), \\
\textbf{Cell state}, \quad & c_i = \Gamma_{F_i} \odot c_{i-1} + \Gamma_{I_i} \odot \rho_{\tanh}(A_C x_i + U_C a_{i-1} + b_C), \\
\textbf{Output vector}, \quad & a_i = \Gamma_{O_i} \odot \rho_{\tanh}(c_i).
\end{aligned}
$$

- The set of **LSTM network** up to time i as

$$
\begin{aligned}
\mathbb{LSTM}_{i,d,k} = \Big\{ & M : (\mathbb{R}^d)^i \times \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}^k \times \mathbb{R}^k \ ; \ M(x_{[0,i]}, a_{-1}, c_{-1}) = (a_i, c_i), \\
& c_i = \Gamma_{F_i} \odot c_{i-1} + \Gamma_{I_i} \odot \rho_{\tanh}(A_C x_i + U_C a_{i-1} + b_C), \\
& a_i = \Gamma_{O_i} \odot \rho_{\tanh}(c_i), a_{-1} = c_{-1} = 0 \Big\},
\end{aligned}
$$

where $x_{[0,i]} = [x_0, \ldots, x_i]$.

PDGM Architecture

- Time discretization $\{t_i\}_{i=1,\ldots,N}$, with $\delta t = t_i - t_{i-1}$.
- Approximate $f(Y_t)$ by

$$f(Y_t) \approx u(Y_{t_i}; \theta) = \varphi(t_i, y_{t_i}, a_{t_{i-1}}; \theta^f),$$

  where $t_i \leq t < t_{i+1}$.
- We are abusing the notation: $u(Y_{t_i}; \theta) = u(y_{t_0}, \ldots, y_{t_i}; \theta)$
- Here $\varphi \in \mathbb{NN}^{\ell}_{k+2,1}$, where a is an output vector from an LSTM network, i.e.

$$a_{t_{i-1}} = \psi(y_{t_0}, \ldots, y_{t_{i-1}}; \theta^r),$$

  for some $\psi \in \mathbb{LSTM}_{i-1,1,k}$.
- $\theta = [\theta^f, \theta^r]$ are the neural network's parameters.

PDGM Architecture

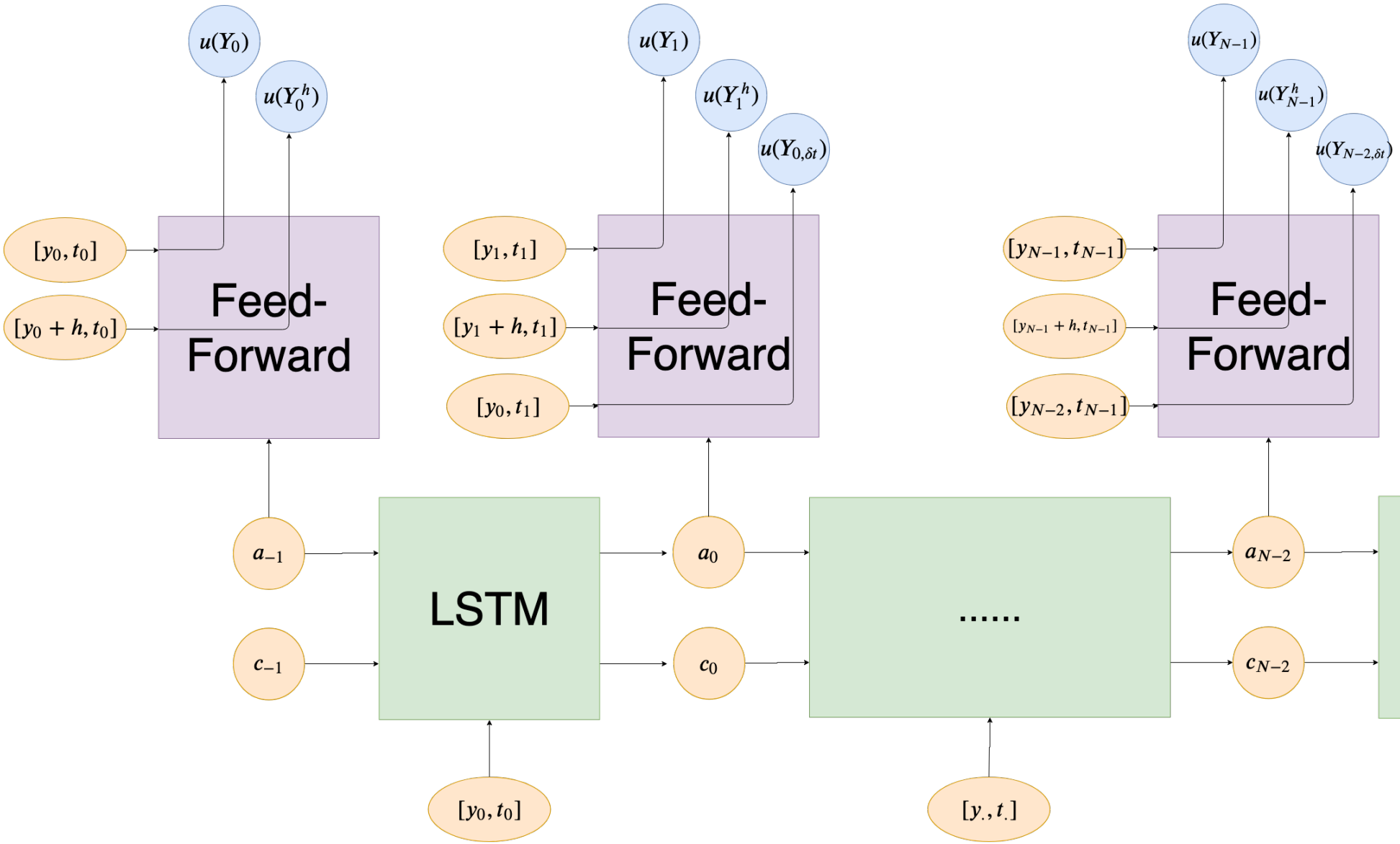
- Neural Network approximation of the **solution**.

$$
\begin{aligned}
u(Y_{t_i};\theta) &= \varphi(t_i, y_{t_i}, a_{t_{i-1}};\theta^f)\\
u(Y^h_{t_i};\theta) &= \varphi(t_i, y_{t_i}+h, a_{t_{i-1}};\theta^f),\\
u(Y_{t_i,\delta t};\theta) &= \varphi(t_{i+1}, y_{t_i}, a_{t_i};\theta^f).
\end{aligned}
$$

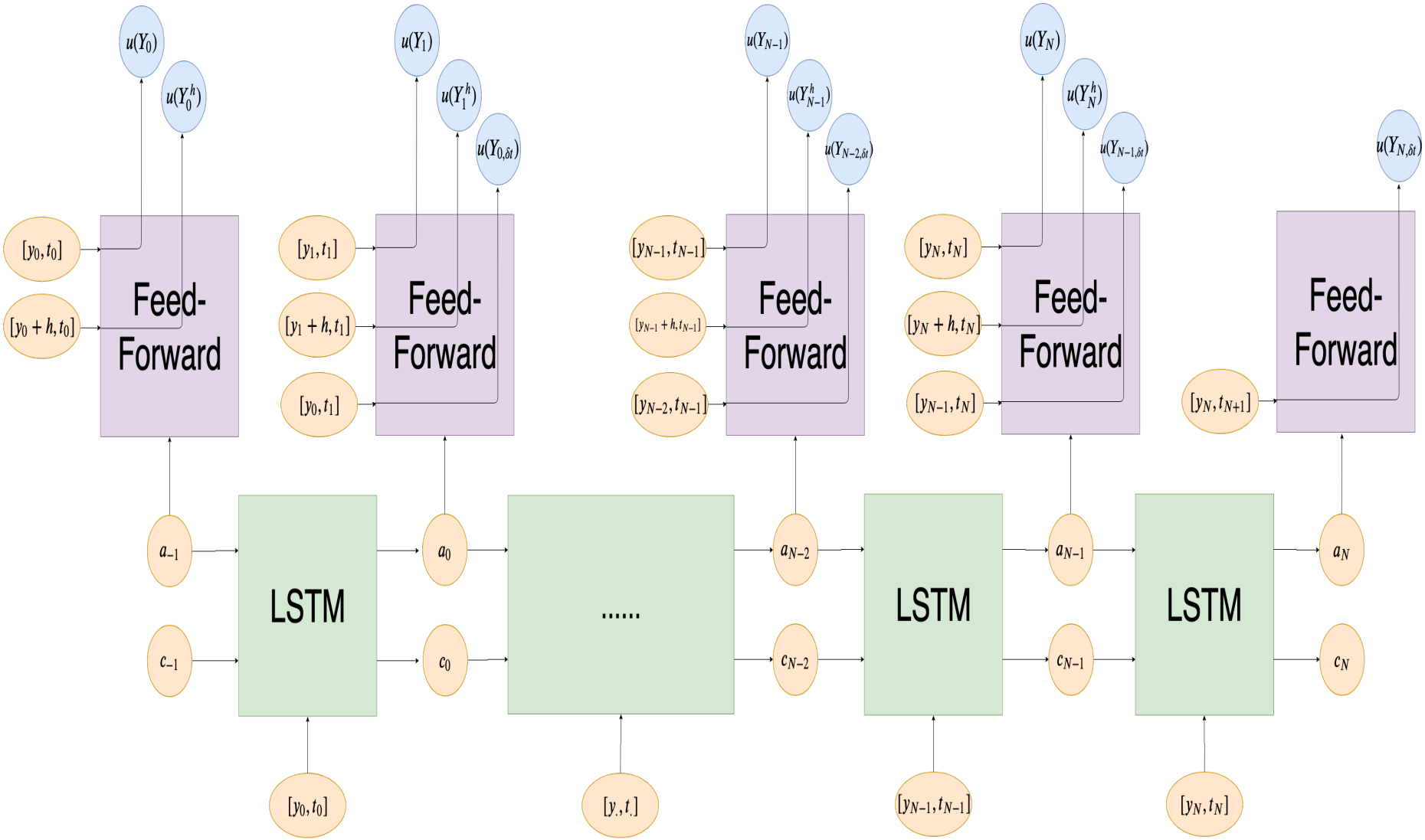
- Approximation of the **derivatives**.

$$
\begin{aligned}
\Delta_t^{[\delta t]} u(Y_{t_i};\theta) &= \frac{u(Y_{t_i,\delta t};\theta) - u(Y_{t_i};\theta)}{\delta t},\\
\Delta_x^{[h]} u(Y_{t_i};\theta) &= \frac{u(Y^h_{t_i};\theta) - u(Y_{t_i};\theta)}{h},\\
\Delta_{xx}^{[h]} u(Y_{t_i};\theta) &= \frac{u(Y^h_{t_i};\theta) - 2u(Y_{t_i};\theta) + u(Y^{-h}_{t_i};\theta)}{h^2}.
\end{aligned}
$$

PDGM Architecture



PDGM Architecture



PDGM Algorithm

PDGM Algorithm

- Consider the general class of final-value PPDE problem:

$$\begin{cases} \Delta_t f(Y_t) + \mathcal{L} f(Y_t) = 0, \\ f(Y_T) = g(Y_T). \end{cases}$$

- Given M simulated paths, time and space discretization parameters δt and h, the loss J will be approximated by

$$J_{N,M}(\theta) = \frac{1}{M}\frac{1}{N}\sum_{j=1}^{M}\sum_{i=0}^{N}\left(\Delta_t^{[\delta t]} u(Y_{t_i}^{(j)};\theta) + \mathcal{L}^{[h]} u(Y_{t_i}^{(j)};\theta)\right)^2 + \frac{1}{M}\sum_{j=1}^{M}\left(u(Y_{t_N}^{(j)};\theta) - g(Y_{t_N}^{(j)})\right)^2.$$

Algorithm 1: Path-Dependent DGM - PDGM

initialize discretization parameter δt, mini-batch M and threshold ϵ;

while $J_{N,M}(\theta) > \epsilon$ **do**

  generate a mini-batch of M paths $\{(Y_{t_i}^{(j)})_{i=0,\dots,N}\}_{j=1,\dots,M}$

  for $i \in \{1, \dots, N\}$ **do**

    calculate $u(Y_{t_i}^{(j)}; \theta)$, $\Delta_t^{[\delta t]} u(Y_{t_i}^{(j)}; \theta)$, $\Delta_x^{[h]} u(Y_{t_i}^{(j)}; \theta)$ and $\Delta_{xx}^{[h]} u(Y_{t_i}^{(j)}; \theta)$;

    put them all together to compute $\mathcal{L}^{[h]} u(Y_{t_i}^{(j)}; \theta)$;

  end

  calculate the approximated loss function, $J_{N,M}(\theta)$;

  minimize $J_{N,M}(\theta)$, update θ using stochastic gradient descent.

end

Examples

Sanity Check - Linear Running Integral

- Consider the heat PPDE

$$\begin{cases} \Delta_t f(Y_t) + \frac{1}{2}\Delta_{xx} f(Y_t) = 0, \\ f(Y_T) = g(Y_T). \end{cases}$$

- As an example, let $g(Y_T) = \int_0^T y_u du$. The explicit solution can be calculated as

$$f(Y_t) = \int_0^t y_u du + y_t(T - t).$$

- Training paths in this subsection are 12800 simulated standard Brownian motions paths with $T = 1$ and $N = 100$. We choose mini-batch size $M = 128$.

- We use a single layer LSTM network with 64 units connecting with a deep feed-forward neural network which consists of three hidden layers with 64, 128, 64 respectively.

Linear Running Integral

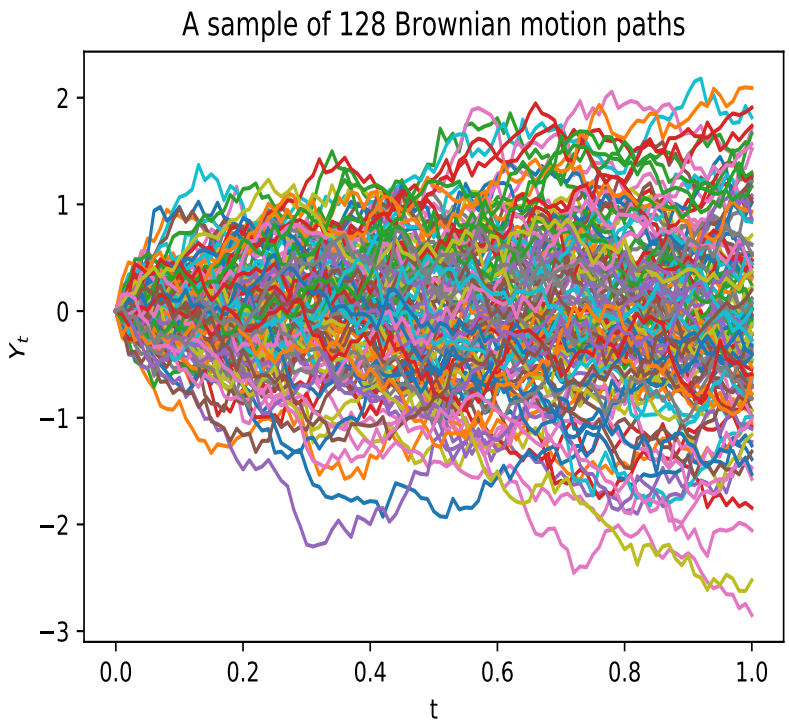


Figure: A sample of 128 Brownian motion paths.

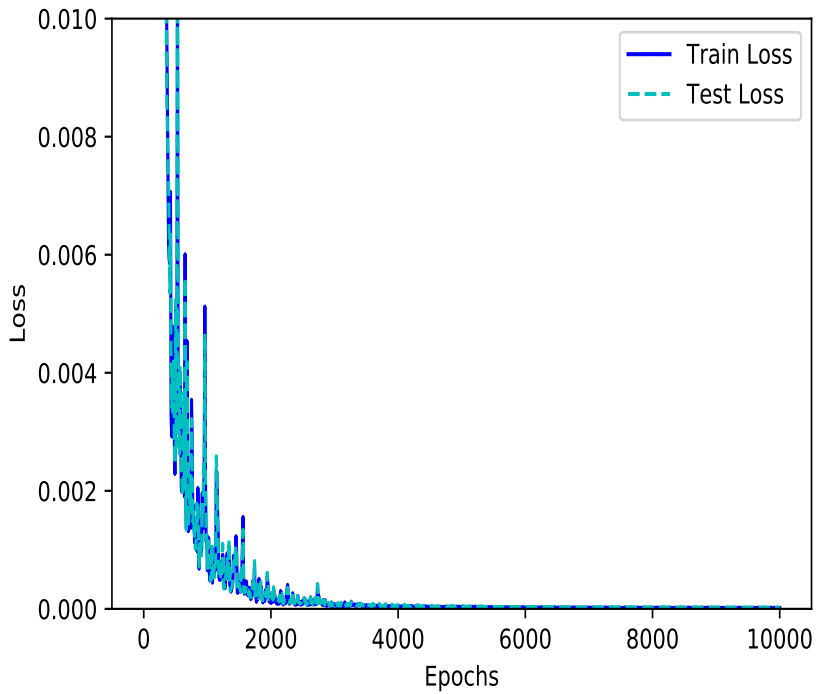
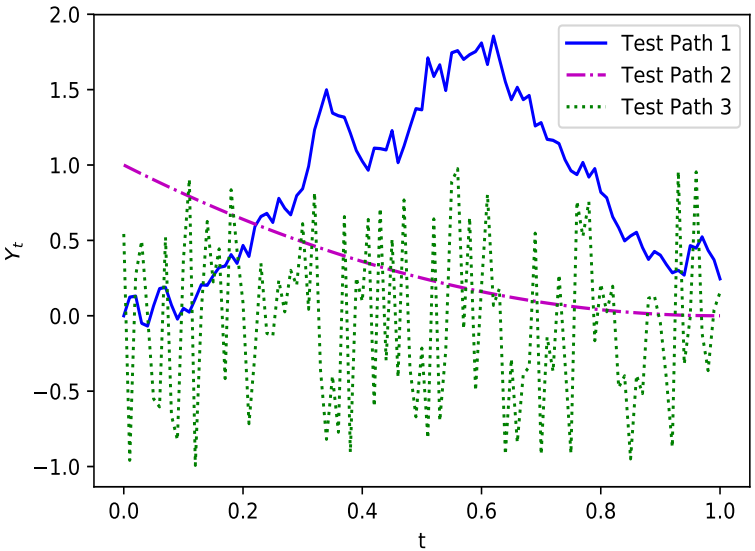
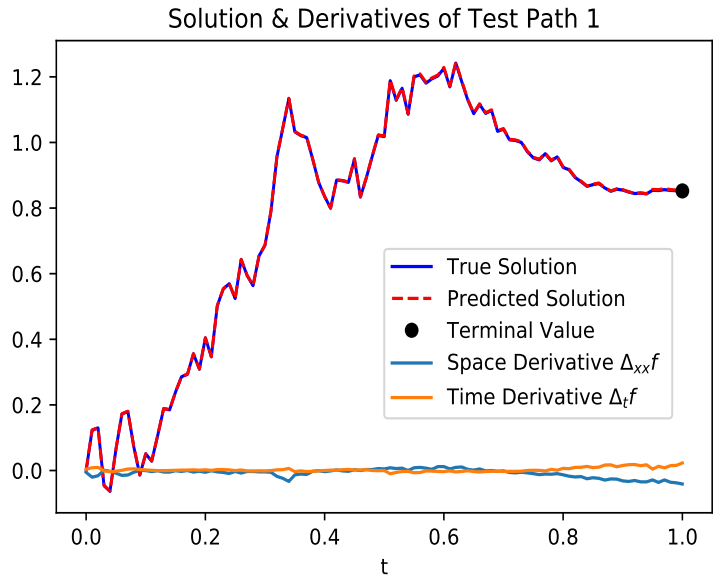
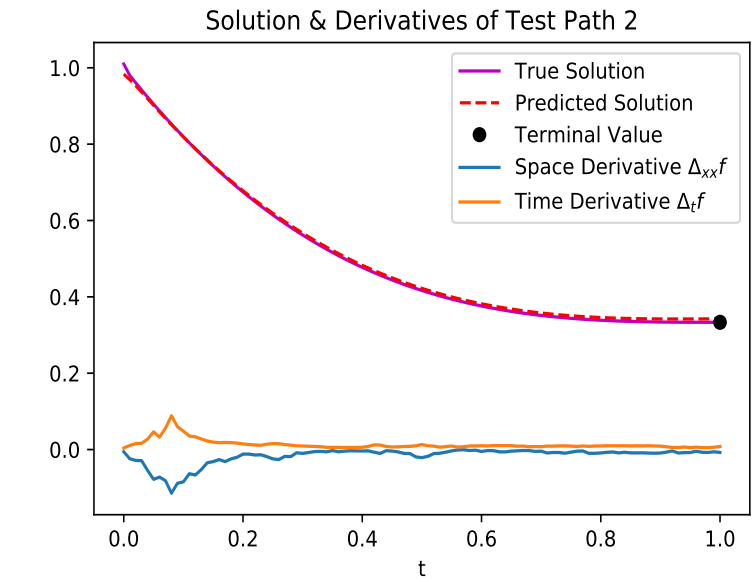


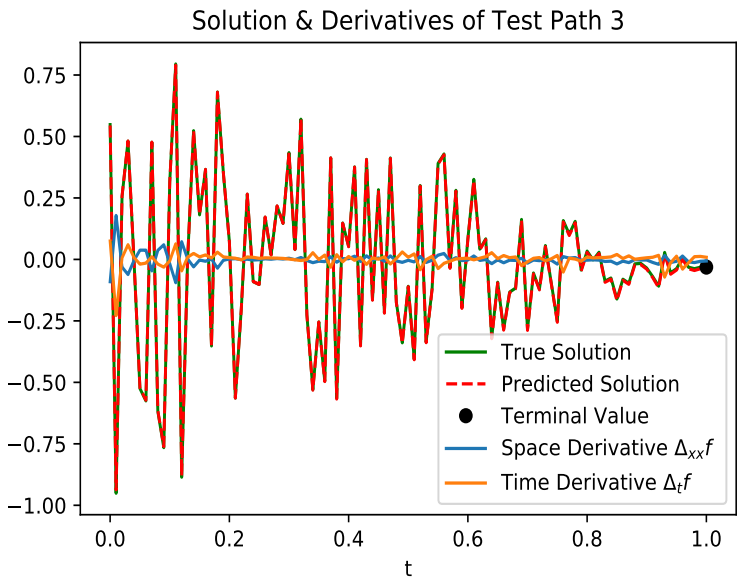
Figure: Train and test losses for the linear running integral example.

Linear Running Integral









Linear Running Integral

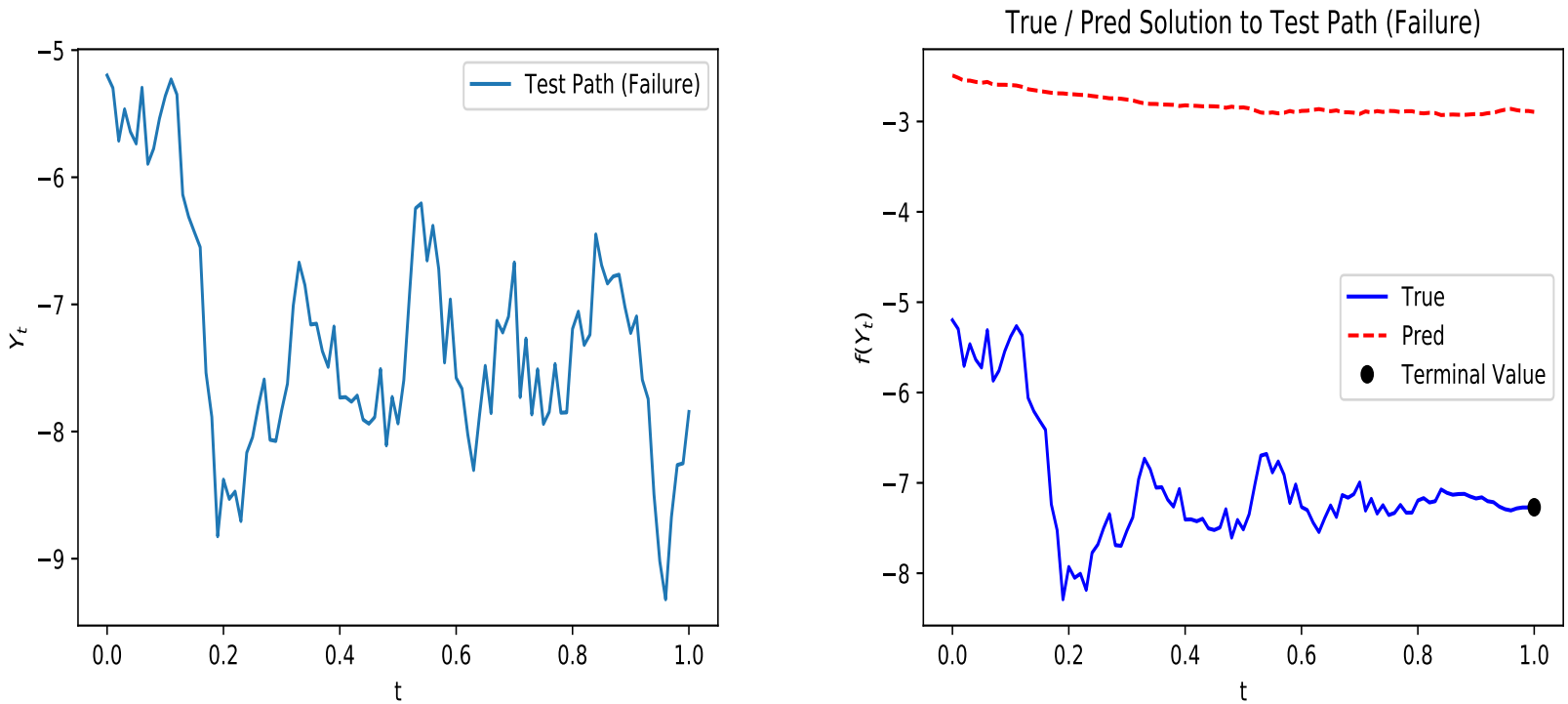


Figure: Prediction failure due to the limitation of training domain.

Linear Running Integral

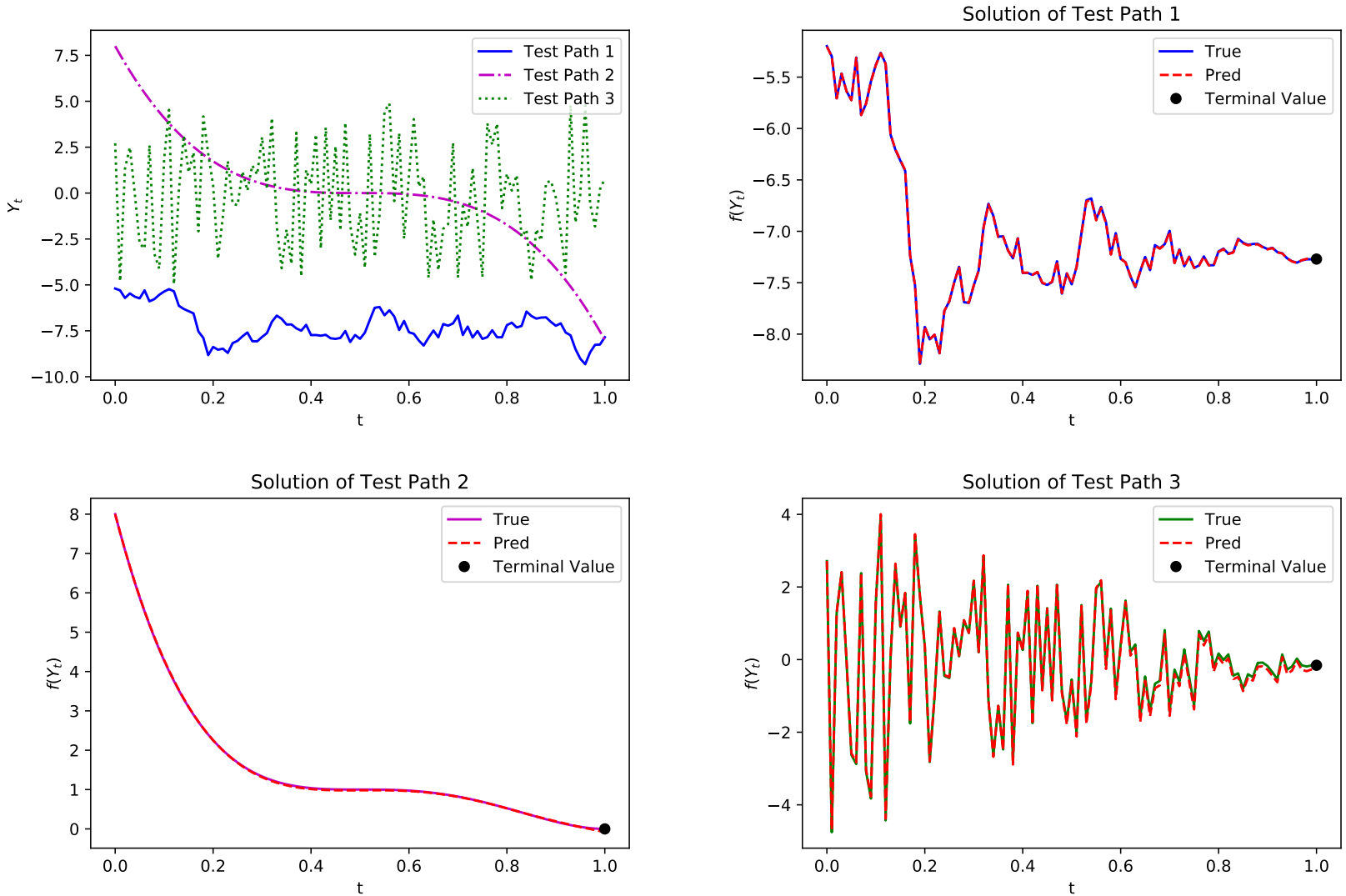


Figure: Training with Brownian paths with volatility $\sigma \in \{1, 2, 3, 4\}$ and initial value $y_0 \sim U(-10, 10)$

Non-Linear Example

- Consider PPDE is of the form

$$\begin{cases} \Delta_t f(Y_t) + (\underline{\mu} 1_{\{\Delta_x f(Y_t) > 0\}} + \overline{\mu} 1_{\{\Delta_x f(Y_t) < 0\}}) \Delta_x f(Y_t) \\ \\ + \frac{1}{2} (\underline{\sigma}^2 1_{\{\Delta_{xx} f(Y_t) < 0\}} + \overline{\sigma}^2 1_{\{\Delta_{xx} f(Y_t) > 0\}}) \Delta_{xx} f(Y_t) + \phi(Y_t) = 0, \\ \\ f(Y_T) = \cos(y_T + I_T). \end{cases}$$

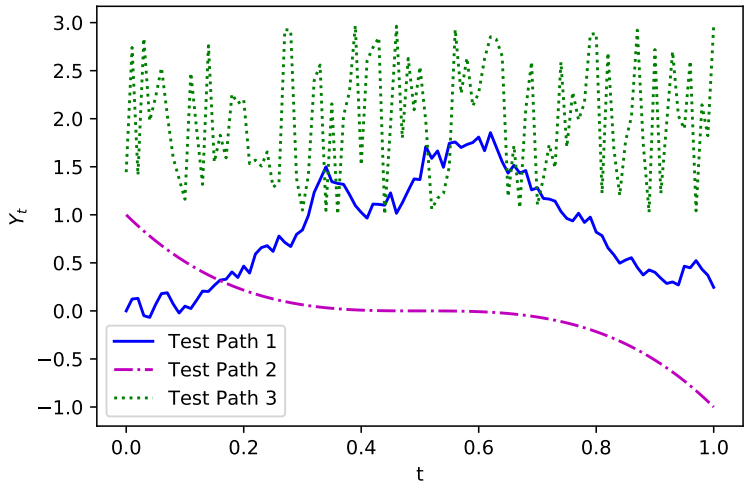
  with

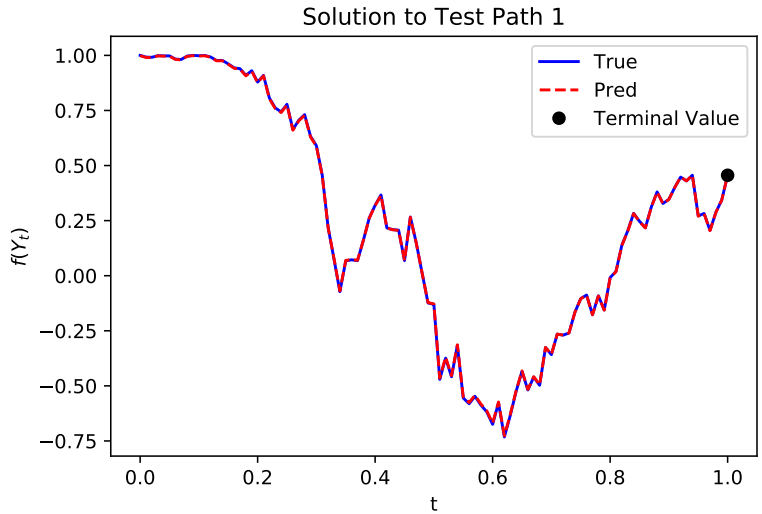
$$\begin{aligned} \phi(Y_t) = (y_t + \underline{\mu}) \min(\sin(y_t + I_t), 0) + (y_t + \overline{\mu}) \max(\sin(y_t + I_t), 0) \\ + \frac{\underline{\sigma}^2}{2} \max(\cos(y_t + I_t), 0) + \frac{\overline{\sigma}^2}{2} \min(\cos(y_t + I_t), 0), \end{aligned}$$

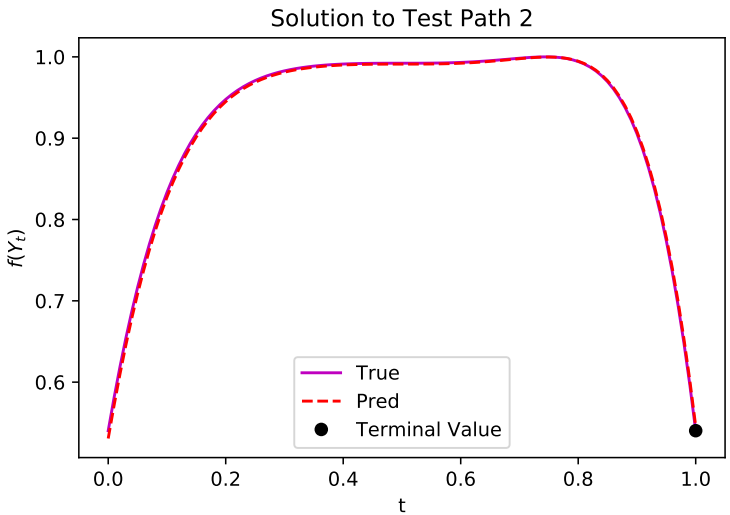
  where $I_t = \int_0^t y_u du$ is the running integral.
- The closed-formula solution is given by

$$f(Y_t) = \cos(y_t + I_t).$$

Non-Linear Example







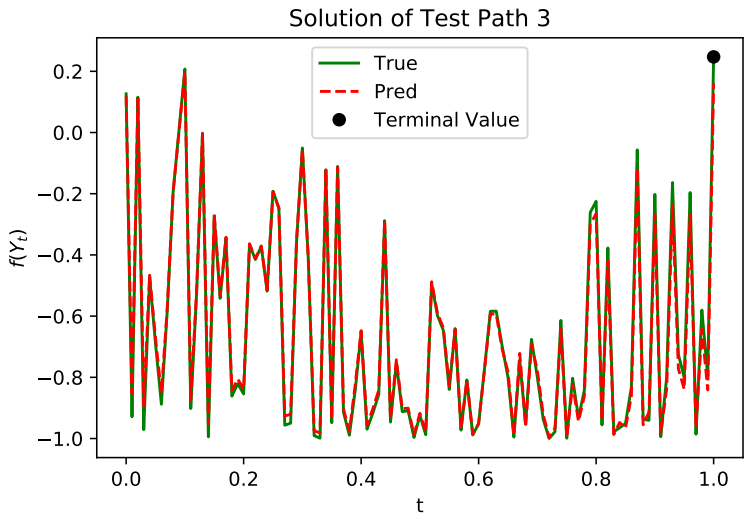


Figure: $\underline{\mu} = -0.2, \overline{\mu} = 0.2, \underline{\sigma} = 0.2,$ and $\overline{\sigma} = 0.3$. Loss reaches around 4×10^{-6} after 15000 epochs.

Applications in Mathematical Finance

- For simplicity, we consider the Black–Scholes model:

$$dx_t = (r - q)x_t dt + \sigma x_t dw_t.$$

- More complex models, as we will see, could easily be considered
- Pricing PPDE:

$$\begin{cases} \Delta_t f(Y_t) + (r-q)y_t \Delta_x f(Y_t) + \frac{1}{2}\sigma^2 y_t^2 \Delta_{xx} f(Y_t) - rf(Y_t) = 0, \\ f(Y_T) = g(Y_T). \end{cases}$$

  - Asian Option: e.g. $g(Y_T) = \left(\frac{1}{T}\int_0^T y_t dt - K\right)^+$.
  - Lookback Option: e.g. $g(Y_T) = y_T - \inf_{0\le t\le T} y_t$.
  - Barrier Option: e.g. $g(Y_T) = (y_T - K)^+ 1_{\left\{\inf\limits_{0\le t\le T} y_t > B\right\}}$.

Down-and-Out Call

- Down-and-out call: option becomes worthless if the spot value crosses a down barrier $B < S_0$. Otherwise, the payoff is a call with strike $K \geq B$.
- The payoff functional can then be written as

$$g(Y_T) = (y_T - K)^+ 1_{\left\{ \inf_{0 \leq t \leq T} y_t > B \right\}}.$$

- A closed-form solution is available:

$$f(Y_t) = \begin{cases} 0, & \text{if } \inf_{0 \leq u \leq t} y_u \leq B, \\ C_{BS}(y_t, T-t) - \left(\frac{y_t}{B}\right)^{1-\lambda} C_{BS}\left(\frac{B^2}{y_t}, T-t\right), & \text{if } \inf_{0 \leq u \leq t} y_u > B, \end{cases}$$

where $C_{BS}(y_t, T-t)$ is the price of a call option with strike K and maturity T at (t, y_t) and $\lambda = \frac{2(r-q)}{\sigma^2}$.

Down-and-Out Call

- Because of the barrier, we modify the loss function
- The loss for a given sample path j at time t_i is

$$J_{t_i}^{(j)}(\theta) = \begin{cases} |u(Y_{t_i}^{(j)};\theta) - 0| & \text{if } \inf_{0\le i'\le i} Y_{t_{i'}}^{(j)} < B, \\ \left(\Delta_t u(Y_{t_i}^{(j)};\theta) + \mathcal{L}u(Y_{t_i}^{(j)};\theta)\right)^2 & \text{otherwise.} \end{cases}$$

The total loss is calculated as

$$\begin{aligned} J_{N,M}(\theta) = \frac{1}{M}\frac{1}{N}\sum_{j=1}^{M}\sum_{i=0}^{N} J_{t_i}^{(j)}(\theta) \\ + \frac{1}{M}\sum_{j=1}^{M}\Bigg[\left(u(Y_{t_N}^{(j)};\theta) - g(Y_{t_N}^{(j)})1_{\left\{\inf_{0\le i\le N} y_{t_i} > B\right\}}\right)^2 \\ + |u(Y_{t_N}^{(j)};\theta) - 0|1_{\left\{\inf_{0\le i\le N} y_{t_i} < B\right\}}\Bigg]. \end{aligned}$$

Down-and-Out Call

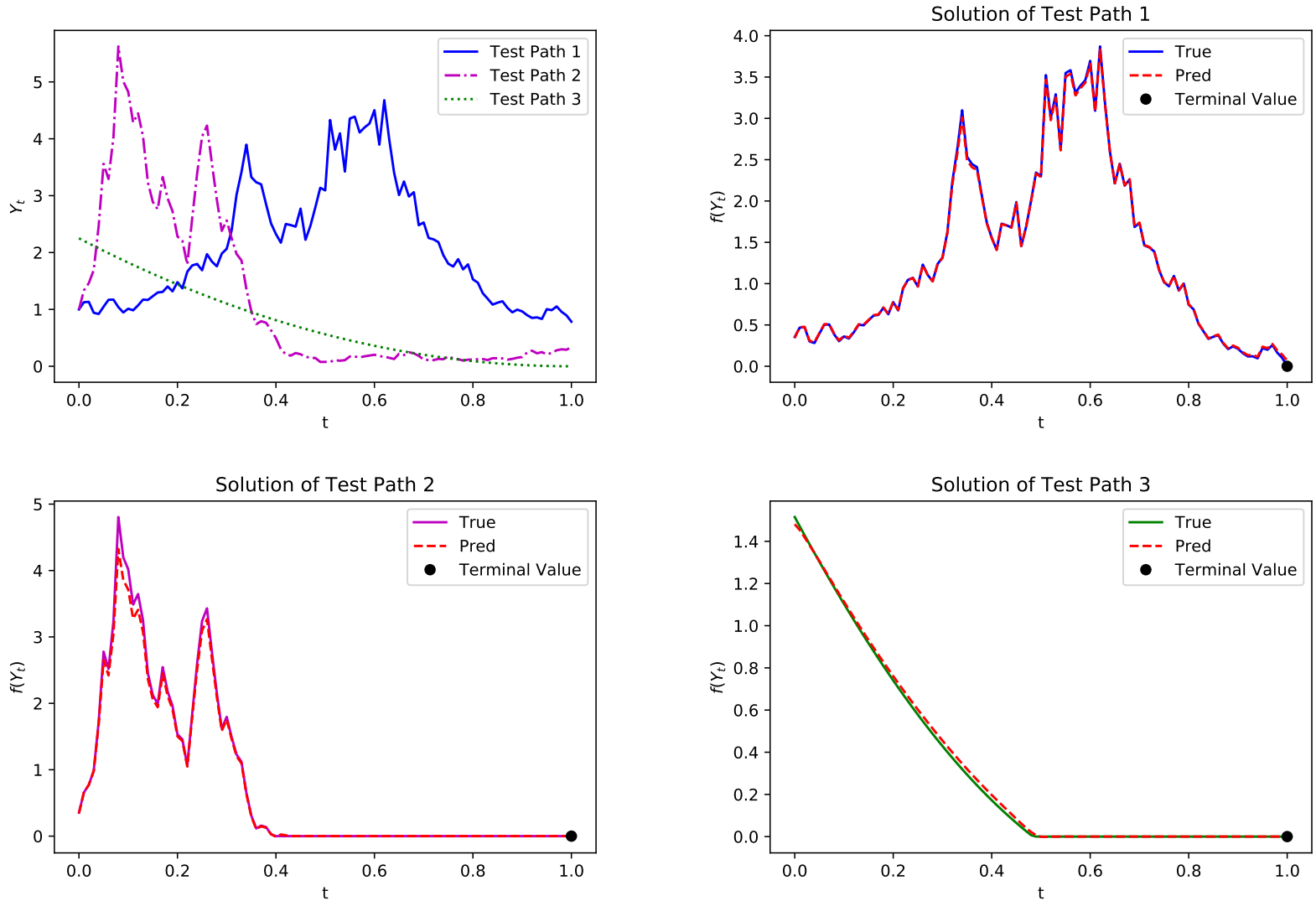


Figure: $B = 0.6$ and $K = 0.8$

Heston Model

- Consider is the well-known Heston model:

$$\begin{cases} dx_t = (r-q)x_t dt + \sqrt{v_t} x_t dw_t, \\ dv_t = \kappa(m - v_t)dt + \xi\sqrt{v_t} dw_t^*, \\ dw_t dw_t^* = \rho dt. \end{cases}$$

- Pricing PPDE

$$\begin{cases} \Delta_t f(Y_t, v) + (r-q)y_t \Delta_x f(Y_t, v) + \frac{1}{2} v y_t^2 \Delta_{xx} f(Y_t, v) - r f(Y_t, v) \\ +\kappa(m-v)\partial_v f(Y_t, v) + \frac{1}{2}\xi^2 v \partial_{vv} f(Y_t, v) + \rho\xi v y_t \Delta_x \partial_v f(Y_t, v) = 0 \\ f(Y_T, v) = g(Y_T). \end{cases}$$

Heston Model

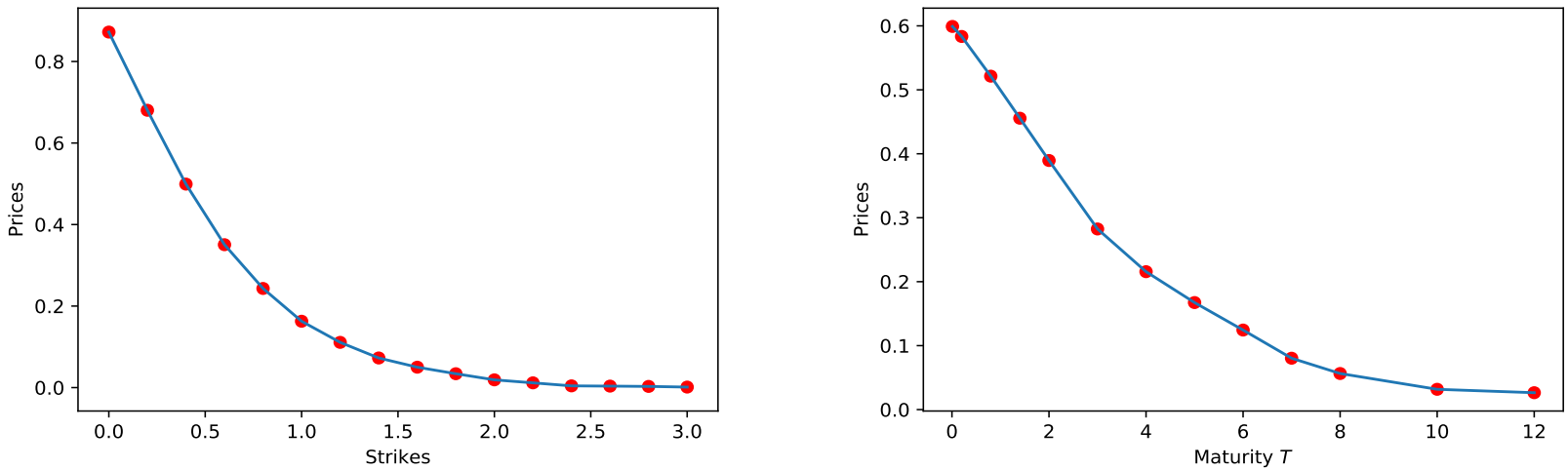


Figure: Parameters: $r = 0.03, q = 0.01, \kappa = 3, m = 1, \xi = 1, \rho = 0.6, x_0 = v_0 = 1$.

On the left ($T = 1$): prices vs strike prices K. On the right ($K = 0.4$): prices vs maturity times T.

Conclusion

- We have presented a deep learning method to solve PPDEs
- Many other examples can be considered (and it was considered in the paper): high-dimensional, stronger path-dependence, etc.
- Future work: "proof" of convergence

Thank you!