Imperial College
London

Towards Robust and Stable Deep Learning Algorithms for Forward-Backward Stochastic Differential Equations

Alexis Laignelet

Meetup - Analytics Club - ETH

15/06/20

About me

- I am a PhD student in the Computing Department of **Imperial College London**.
- I am supervised by **Dr Panos Parpas** from the **Computational Optimisation Group**. I also work with Dr Nikolas Kantas and Pr Grigorios A. Pavliotis from the Department of Mathematics.
- I am working on **smoothing techniques** and **implicit gradient methods**.
- I am also doing an internship within **JPMorgan AI & Machine Learning**.

Dr Panos Parpas
Department of Computing
www.doc.ic.ac.uk/~pp500

Pr Grigorios A. Pavliotis
Department of Mathematics
www.ma.ic.ac.uk/~pavl

Dr Nikolas Kantas
Department of Mathematics
www.ma.ic.ac.uk/~nkantas

About this work

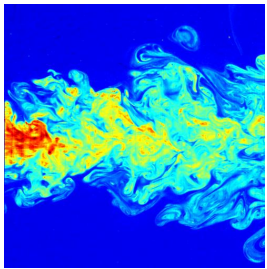
- This is joint work with **Batuhan Güler** and **Dr Panos Parpas** based on our master thesis.
- We have presented the project at **NeurIPS 2019** (in Vancouver) during the **Robust AI in Financial Services** workshop.
- Paper on arXiv: `https://arxiv.org/abs/1910.11623`
- Poster on my personal website: `https://alaignelet.github.io/pdfs/neurips2019_poster.pdf`

Vancouver Convention Centre, December 2019.

Table of contents

Partial Differential Equations

- **Partial Differential Equations** are everywhere:
  - quantum mechanics (Schrödinger),
  - classical mechanics (Hamilton-Jacobi),
  - fluid mechanics (Navier-Stokes),
  - electromagnetism (Maxwell),
  - financial mathematics (Black-Scholes),
  - ...
- There is usually no closed-form solution: numerical techniques are required.
- Numerical methods do not scale well with the dimension: **curse of dimensionnality**.



Turbulent jet due to non-linear terms in Navier-Stokes equation.

Stochastic Differential Equations

Stochastic process

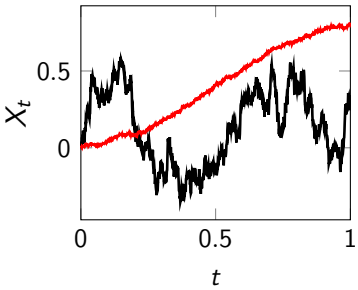
Defined by a **stochastic differential equation**:

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dW_t$$

where the **drift** $\boldsymbol{\mu(t, X_t)}$ and the **volatility** $\boldsymbol{\sigma(t, X_t)}$ are continuous functions, and the **Brownian motion** $\boldsymbol{W_t}$ is the limit of a random walk when $\delta_t \to 0$.

Example of stochastic processes:

- Geometric BM: $dX_t = \mu X_t dt + \sigma X_t dW_t$
- Ornstein-Uhlenbeck: $dX_t = -\alpha X_t dt + \sqrt{2D}dW_t$
- Cox-Ingersoll-Ross: $dX_t = \alpha(b - X_t)dt + \sigma\sqrt{X_t}dW_t$



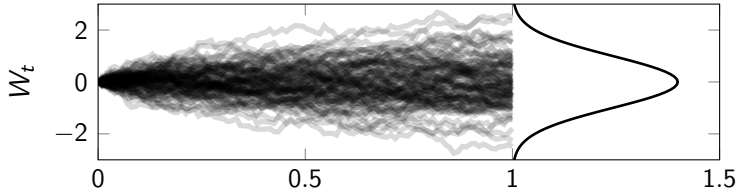
Stochastic processes with constant drift and volatility. In black, $\mu = 0.1$ and $\sigma = 1$ whereas for the red path $\mu = 1$ and $\sigma = 0.1$.

Example: Brownian motion

For the **stochastic differential equation** $dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dW_t$, the evolution of the **transition probabilities** $\boldsymbol{p(s, y; t, x)}$ is ruled by the **Fokker-Planck** equation:

$$\frac{\partial}{\partial t}p(s, y; t, x) = -\frac{\partial}{\partial x}\left(\mu(t, x)p(s, y; t, x)\right) + \frac{1}{2}\frac{\partial^2}{\partial x^2}\left(\sigma^2(t, x)p(s, y; t, x)\right)$$

In the simple case of the Brownian motion $dX_t = dW_t$, we obtain the **Heat equation** where the solution is the **Gaussian distribution**.



Plot of 100 different paths from Brownian motion and corresponding distribution at time $t = 1$.

Non-linear PDEs

Let **parabolic non-linear** PDEs of the form:

$$u_t = \varphi(t, x, u, \nabla u) - \mu(t, x, u, \nabla u)^T \nabla u - \frac{1}{2} \operatorname{Tr}\left[\sigma(t, x, u)\sigma(t, x, u)^T \nabla^2 u\right]$$

Forward-Backward SDEs

The following **forward-backward** SDEs:

$$\begin{aligned} dX_t &= \mu(t, X_t, Y_t, Z_t)dt + \sigma(t, X_t, Y_t)dW_t, & X_0 &= \xi \\ dY_t &= \varphi(t, X_t, Y_t, Z_t)dt + Z_t^T \sigma(t, X_t, Y_t)dW_t, & Y_T &= g(X_T) \end{aligned}$$

Equivalent to the above PDE with $Y_t = u(t, X_t)$ and $Z_t = \nabla u(t, X_t)$.

- $X_0 = \xi$ is the initial condition of the **forward** equation,
- $Y_T = g(X_T)$ is the terminal condition of the **backward** equation.

Designing a neural network

Ideas:

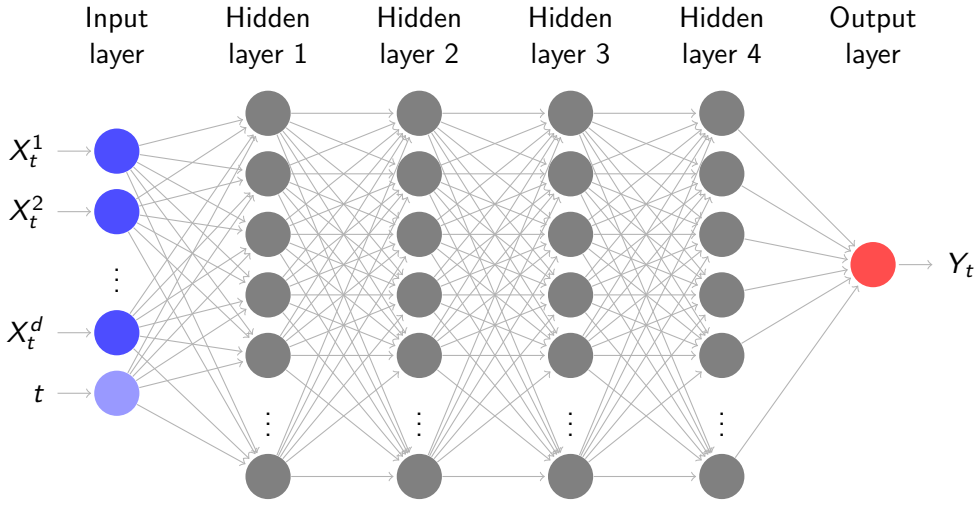
- train a **neural network** to map the function $Y_t = u(t, X_t)$,
- compute $Z_t = \nabla u(t, X_t)$ using automatic differentiation from deep learning libraries.

Euler-Maruyana scheme

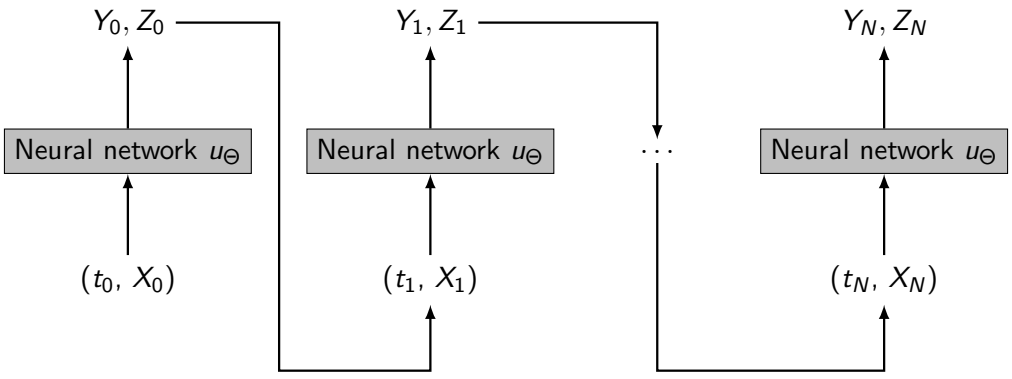
The discretisation in time leads to:

$$
\begin{aligned}
X_{n+1} &\approx X_n + \mu(t_n, X_n, Y_n, Z_n)\Delta t_n + \sigma(t_n, X_n, Y_n)\Delta W_n \\
Y_{n+1} &\approx Y_n + \varphi(t_n, X_n, Y_n, Z_n)\Delta t_n + Z_n^T \sigma(t_n, X_n, Y_n)\Delta W_n \\
\Delta W_n &\sim \mathcal{N}(0, \Delta t_n)
\end{aligned}
$$

Neural network: one time step



Neural network: N time steps



Θ: shared parameters through time.

Minimize the approximation error

Loss function

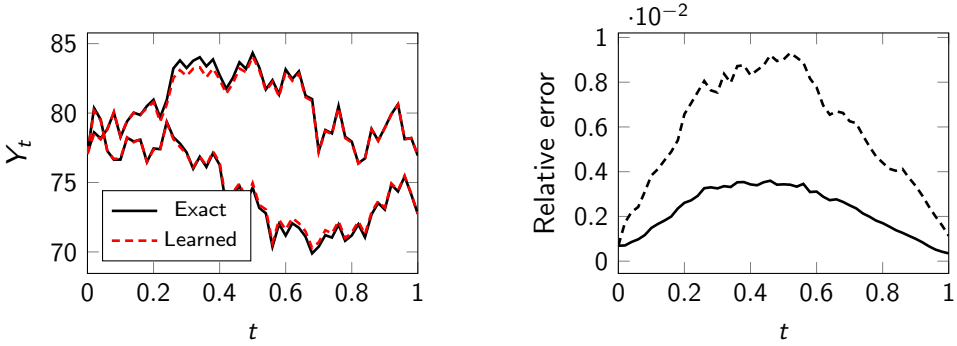
The **loss function** is defined as:

$$\min_{\Theta} \sum_{m=1}^{M} \sum_{n=0}^{N-1} |Y_{n+1}^m(\Theta) - Y_n^m(\Theta) - \varphi(t_n, X_n^m, Y_n^m(\Theta), Z_n^m(\Theta))\Delta t_n - (Z_n^m(\Theta))^T \sigma(t_n, X_n^m, Y_n^m(\Theta))\Delta W_n^m|^2 + \sum_{m=1}^{M} |Y_N^m(\Theta) - g(X_N^m)|^2$$

- M: number of trajectories (batch size),
- N: number of time steps.

Example: Black-Scholes equation

Eg: $g(x) = \|x\|^2$ leading to the closed-form solution $u(t,x) = e^{(r+\sigma^2)(T-t)}\|x\|^2$.



Experiment with $M = 100$, $N = 50$, $D = 100$, 4 hidden layers of 256 neurons, Adam optimiser. On the left, plot of two different trajectories (over $M = 100$). The red ones are the result of the neural network, whereas the black ones represent the exact solution. On the right, mean relative error (plain black), and mean + 2 standard deviations (dashed line) over $M = 100$ trajectories.

ResNet and stability

In a **feed forward** neural network the next layer is defined by:

$$x(t+1) = f(x(t), \theta(t))$$

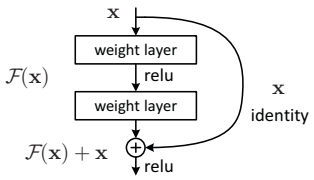
where f is a non-linear transformation and $\theta(t)$ a set of parameters. In a **ResNet**, the **skip connection** leads to:

$$x(t+1) = x(t) + f(x(t), \theta(t))$$

In the limit when step size tends to zero (infinite layers):

$$\frac{dx(t)}{dt} = f(x(t), \theta, t)$$

This can be studied as an ODE, and stability properties can be derived.



Skip connection as presented in the original paper.

NAIS-Net: a more sofisticated version

NAIS-Net is a non-autonomous input-output stable neural network:

$$x(t+1) = x(t) + h\sigma(Ax(t) + Bu + C)$$

where

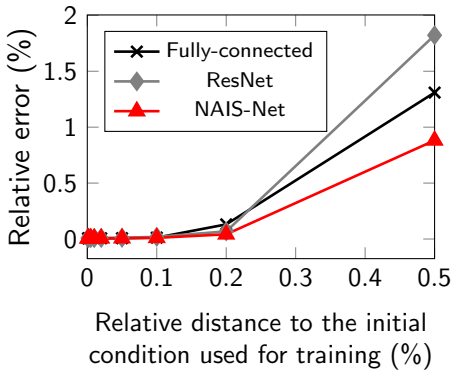
- A, B, C are trainable parameters,
- u makes the system non-autonomous and the output input-dependent.

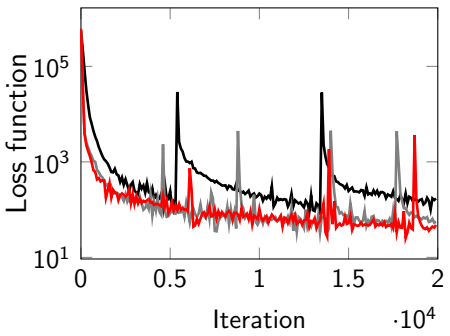
To ensure stability, the weight matrix is constrained to be symmetric definite negative:

$$A = -R^T R - \epsilon I$$

where $\epsilon > 0$ is an hyper-parameter to make the eigenvalues strictly negative.
Note that when $B = 0$ and $h = 1$ we obtain the original **ResNet** formulation.

Loss functions and generalisation





On the left, the generalisation error measured by slightly moving the initial condition, and check the error afterward. NAIS-Net appears to provide the lowest relative error. On the right, the loss function for the different architectures. ResNet and NAIS-Net provide a smoother loss function than the fully connected neural network.

Multilevel

Discretization

The discretization scheme, at level l, is:

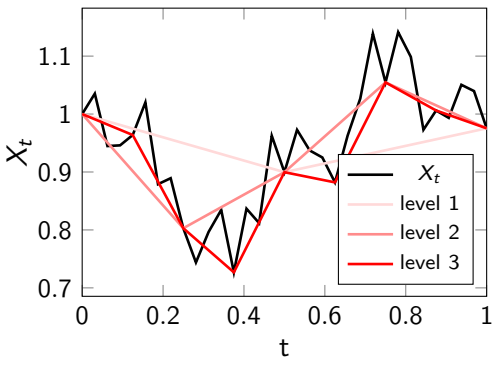
$$X_{n+1} = X_n + \mu(t_n, X_n)h + \sigma(t_n, X_n)\Delta W_n$$

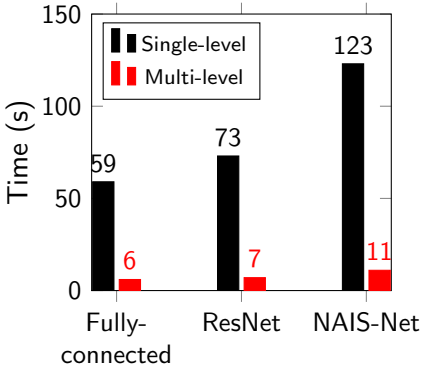
where $h_l = h_0 M^{-l}$ with h_0 the initial step size and M the ratio between two levels.

Warm up procedure:

1. Start with large initial temporal step size $\Delta t = h_0$ and train the model until the loss does not decrease substantially.
2. Decrease the step size according to $h_l = h_0 M^{-l}$.
3. Retrain the model until the loss does not decrease substantially.
4. Repeat steps 2. and 3.

Speed-up





On the left, different levels of the same stochastic path. The construction of more discretised paths is based on Brownian bridge. On the right, the average time to train the neural network (4 layers of 256 neurons) for the three studied architectures and using the multilevel technique. The convergence is roughly 10 times faster.

Achievements and future work

Achievements:

- reviewed the recent proposals for the solution of high-dimensional PDEs using deep learning,
- understood better stability, robustness and generalisation,
- adopted the dynamical view of the problem and showed NAIS-Net well perform,
- improved time computation by order of magnitude using multilevel technique.

Future work:

- use more advance discretisation techniques,
- improve sampling methods,
- try other terminal conditions, other differential equations.

References

- Gobet, E. (2016).
  Monte-Carlo methods and stochastic processes: from linear to non-linear.
  CRC Press.
- Henry-Labordere, P. (2017).
  Deep primal-dual algorithm for bsdes: Applications of machine learning to cva and im.
  Available at SSRN 3071506.
- Raissi, M. (2018).
  Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations.
  arXiv preprint arXiv:1804.07010.