Numerical Methods for Mean Field Games

Lecture 5
Deep Learning Methods: Part II
FBPDEs and Master equations

Mathieu LAURIÈRE

New York University Shanghai

UM6P Vanguard Center, Université Cadi AYYAD,
University Côte d'Azur, & GE2MI
Open Doctoral Lectures
July 5 – 7, 2023

Outline

Recap of Lecture 4

- Background on deep learning (DL)
- DL for MFC using a direct approach
- DL for MKV FBSDEs using a “shooting method”
- Extensions
- What about DL for the PDE approach to MFG/MFC?

Outline

Outline

- Look for $\varphi : \mathbb{R} \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), \varphi'(x), \dots) = 0, & x \in [a, b] \\ G(a, \varphi(a), \varphi'(a), \dots) = 0 \end{cases}$$

Solving ODEs with Neural Networks

- Look for $\varphi : \mathbb{R} \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), \varphi'(x), \dots) = 0, & x \in [a, b] \\ G(a, \varphi(a), \varphi'(a), \dots) = 0 \end{cases}$$

- Look among NN φ_θ

$$\begin{cases} F(x, \varphi_\theta(x), \varphi_\theta'(x), \dots) = 0, & x \in [a, b] \\ G(a, \varphi_\theta(a), \varphi_\theta'(a), \dots) = 0 \end{cases}$$

Solving ODEs with Neural Networks

- Look for $\varphi : \mathbb{R} \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), \varphi'(x), \dots) = 0, & x \in [a, b] \\ G(a, \varphi(a), \varphi'(a), \dots) = 0 \end{cases}$$

- Look among NN φ_θ

$$\begin{cases} F(x, \varphi_\theta(x), \varphi_\theta'(x), \dots) = 0, & x \in [a, b] \\ G(a, \varphi_\theta(a), \varphi_\theta'(a), \dots) = 0 \end{cases}$$

- Rephrase as minimization problem: minimizer over θ

$$L(\theta) = \mathbb{E}_{X \sim \mathcal{U}([a,b])} \left[|F(X, \varphi_\theta(X), \varphi_\theta'(X), \dots)|^2 \right] + |G(a, \varphi_\theta(a), \varphi_\theta'(a), \dots)|^2$$

Solving ODEs with Neural Networks

- Look for $\varphi : \mathbb{R} \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), \varphi'(x), \dots) = 0, & x \in [a, b] \\ G(a, \varphi(a), \varphi'(a), \dots) = 0 \end{cases}$$

- Look among NN φ_θ

$$\begin{cases} F(x, \varphi_\theta(x), \varphi_\theta'(x), \dots) = 0, & x \in [a, b] \\ G(a, \varphi_\theta(a), \varphi_\theta'(a), \dots) = 0 \end{cases}$$

- Rephrase as minimization problem: minimizer over θ

$$L(\theta) = \mathbb{E}_{X \sim \mathcal{U}([a,b])} \left[|F(X, \varphi_\theta(X), \varphi_\theta'(X), \dots)|^2 \right] + |G(a, \varphi_\theta(a), \varphi_\theta'(a), \dots)|^2$$

- Use SGD

Solving ODEs with Neural Networks

- Look for $\varphi : \mathbb{R} \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), \varphi'(x), \dots) = 0, & x \in [a,b] \\ G(a, \varphi(a), \varphi'(a), \dots) = 0 \end{cases}$$

- Look among NN φ_θ

$$\begin{cases} F(x, \varphi_\theta(x), \varphi_\theta'(x), \dots) = 0, & x \in [a,b] \\ G(a, \varphi_\theta(a), \varphi_\theta'(a), \dots) = 0 \end{cases}$$

- Rephrase as minimization problem: minimizer over θ

$$L(\theta) = \mathbb{E}_{X \sim \mathcal{U}([a,b])} \left[\left| F(X, \varphi_\theta(X), \varphi_\theta'(X), \dots) \right|^2 \right] + \left| G(a, \varphi_\theta(a), \varphi_\theta'(a), \dots) \right|^2$$

- Use SGD
- Note: we solve and ODE without discretizing time!

Numerical Illustration

Application to the following ODE:

$$\begin{cases} F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x)), & x \in [0, 5] \\ \varphi(0) = 1 \end{cases}$$

Numerical Illustration

Application to the following ODE:

$$\begin{cases} F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x)), & x \in [0, 5] \\ \varphi(0) = 1 \end{cases}$$

Explicit solution:

$$\varphi(x) = x - 1 + 2e^{-x}$$

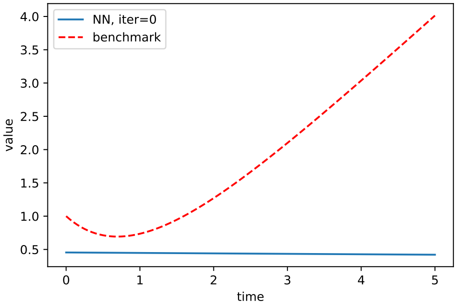
Numerical Illustration

Application to the following ODE:

$$\begin{cases} F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x)), & x \in [0, 5] \\ \varphi(0) = 1 \end{cases}$$

Explicit solution:

$$\varphi(x) = x - 1 + 2e^{-x}$$



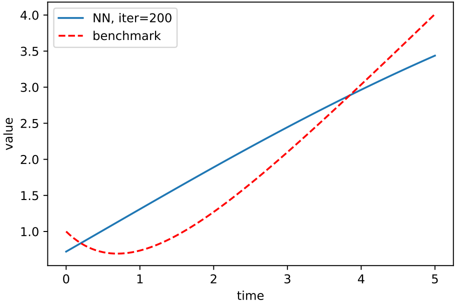
Numerical Illustration

Application to the following ODE:

$$\begin{cases} F(x,\varphi(x),\varphi'(x)) = \varphi'(x) - (x - \varphi(x)), & x \in [0,5] \\ \varphi(0) = 1 \end{cases}$$

Explicit solution:

$$\varphi(x) = x - 1 + 2e^{-x}$$



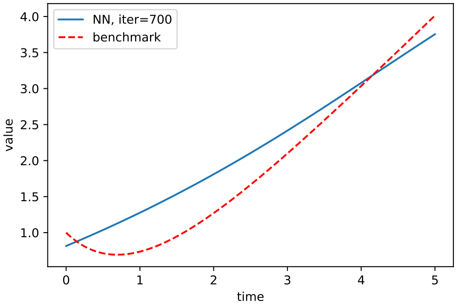
Numerical Illustration

Application to the following ODE:

$$\begin{cases} F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x)), & x \in [0, 5] \\ \varphi(0) = 1 \end{cases}$$

Explicit solution:

$$\varphi(x) = x - 1 + 2e^{-x}$$



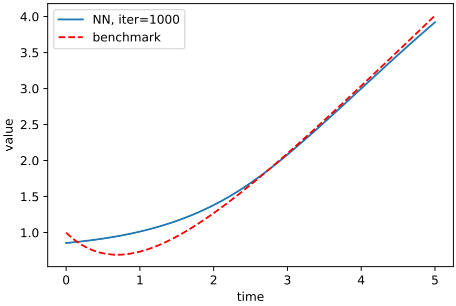
Numerical Illustration

Application to the following ODE:

$$\begin{cases} F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x)), & x \in [0, 5] \\ \varphi(0) = 1 \end{cases}$$

Explicit solution:

$$\varphi(x) = x - 1 + 2e^{-x}$$

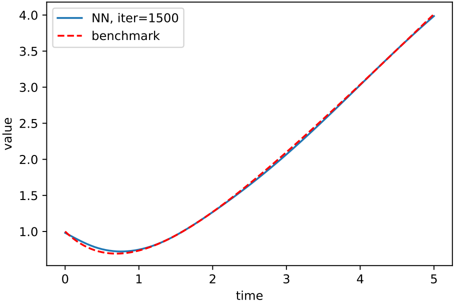


Application to the following ODE:

$$\begin{cases} F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x)), & x \in [0, 5] \\ \varphi(0) = 1 \end{cases}$$

Explicit solution:

$$\varphi(x) = x - 1 + 2e^{-x}$$



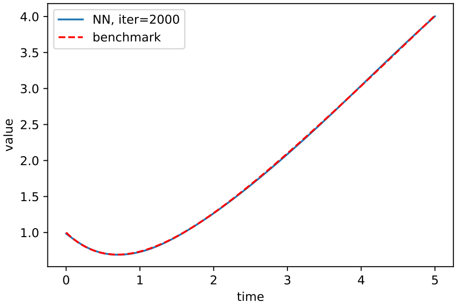
Numerical Illustration

Application to the following ODE:

$$\begin{cases} F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x)), & x \in [0, 5] \\ \varphi(0) = 1 \end{cases}$$

Explicit solution:

$$\varphi(x) = x - 1 + 2e^{-x}$$



Sample code

Code

Sample code to illustrate: IPython notebook

`https://colab.research.google.com/drive/1pHAKlcRwpeMwzTFI7CcEi3NI5uo0cVqE?usp=sharing`

- ODE
- Solved by DGM

Outline

Deep Galerkin Method (DGM), proposed by [Sirignano and Spiliopoulos, 2018]

- Look for $\varphi : \mathbb{R}^d \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

Deep Galerkin Method (DGM), proposed by [Sirignano and Spiliopoulos, 2018]

- Look for $\varphi : \mathbb{R}^d \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

- Look among NN φ_θ

$$\begin{cases} F(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

Deep Galerkin Method (DGM), proposed by [Sirignano and Spiliopoulos, 2018]

- Look for $\varphi : \mathbb{R}^d \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

- Look among NN φ_θ

$$\begin{cases} F(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

- Rephrase as minimization problem: minimizer over θ

$$\begin{aligned} L(\theta) = {} & \mathbb{E}_{X \sim \mathcal{U}(\Omega)} \left[\left| F(X, \varphi_\theta(X), D\varphi_\theta(X), D^2\varphi_\theta(X), \dots) \right|^2 \right] \\ & + \mathbb{E}_{Y \sim \mathcal{U}(\partial\Omega)} \left[\left| G(Y, \varphi_\theta(Y), D\varphi_\theta(Y), D^2\varphi_\theta(Y), \dots) \right|^2 \right] \end{aligned}$$

Deep Galerkin Method (DGM), proposed by [Sirignano and Spiliopoulos, 2018]

- Look for $\varphi : \mathbb{R}^d \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

- Look among NN φ_θ

$$\begin{cases} F(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

- Rephrase as minimization problem: minimizer over θ

$$\begin{aligned} L(\theta) = & \mathbb{E}_{X \sim \mathcal{U}(\Omega)} \left[|F(X, \varphi_\theta(X), D\varphi_\theta(X), D^2\varphi_\theta(X), \dots)|^2 \right] \\ & + \mathbb{E}_{Y \sim \mathcal{U}(\partial\Omega)} \left[|G(Y, \varphi_\theta(Y), D\varphi_\theta(Y), D^2\varphi_\theta(Y), \dots)|^2 \right] \end{aligned}$$

- Use SGD

Deep Galerkin Method (DGM), proposed by [Sirignano and Spiliopoulos, 2018]

- Look for $\varphi : \mathbb{R}^d \ni x \mapsto \varphi(x) \in \mathbb{R}$ s.t.

$$\begin{cases} F(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

- Look among NN φ_θ

$$\begin{cases} F(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \Omega \\ G(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, & x \in \partial\Omega \end{cases}$$

- Rephrase as minimization problem: minimizer over θ

$$\begin{aligned} L(\theta) = & \mathbb{E}_{X \sim \mathcal{U}(\Omega)} \left[|F(X, \varphi_\theta(X), D\varphi_\theta(X), D^2\varphi_\theta(X), \dots)|^2 \right] \\ & + \mathbb{E}_{Y \sim \mathcal{U}(\partial\Omega)} \left[|G(Y, \varphi_\theta(Y), D\varphi_\theta(Y), D^2\varphi_\theta(Y), \dots)|^2 \right] \end{aligned}$$

- Use SGD

DGM: Comments

Remarks on the implementation:

- Choice of distribution:
  - influences training and generalization
  - may depend on the problem (e.g., some regions are more important than others)
- Boundary conditions:
  - need to balance their importance with the PDE residual; can be challenging
  - can sometimes imposed by construction
- Higher order derivatives computation:
  - in principle, can be computed automatically but costly in high dimension
  - approximations are possible, see [Sirignano and Spiliopoulos, 2018] for an approximation of second order derivatives
- Choice of architecture:
  - in low dimension, feedforward fully connected networks work
  - in high dimension, they seem inefficient; [Sirignano and Spiliopoulos, 2018] proposed a specific architecture
- Other DL methods for PDEs e.g. [Raissi et al., 2019]

DGM Architecture

- Let $\vec{x} = (t, x)$ be the input
- Architecture: $L + 1$ hidden layers ($\odot$ denotes element-wise multiplication):

$$
\begin{aligned}
S^1 &= \sigma(W^1 \vec{x} + b^1), \\
Z^\ell &= \sigma(U^{z,\ell} \vec{x} + W^{z,\ell} S^\ell + b^{z,\ell}), \quad \ell = 1, \ldots, L, \\
G^\ell &= \sigma(U^{g,\ell} \vec{x} + W^{g,\ell} S^1 + b^{g,\ell}), \quad \ell = 1, \ldots, L, \\
R^\ell &= \sigma(U^{r,\ell} \vec{x} + W^{r,\ell} S^\ell + b^{r,\ell}), \quad \ell = 1, \ldots, L, \\
H^\ell &= \sigma(U^{h,\ell} \vec{x} + W^{h,\ell} (S^\ell \odot R^\ell) + b^{h,\ell}), \quad \ell = 1, \ldots, L, \\
S^{\ell+1} &= (1 - G^\ell) \odot H^\ell + Z^\ell \odot S^\ell, \quad \ell = 1, \ldots, L, \\
f(t, x; \theta) &= W S^{L+1} + b,
\end{aligned}
$$

- The parameters are

$$
\theta = \left\{ W^1, b^1, \left(U^{\alpha,\ell}, W^{\alpha,\ell}, b^{\alpha,\ell} \right)_{\ell=1,\ldots,L, \alpha \in \{z,g,r,h\}}, W, b \right\}.
$$

- The number of units in each layer is M and $\sigma : \mathbb{R}^M \to \mathbb{R}^M$ is an element-wise nonlinearity:

$$
\sigma(z) = \Big(\phi(z_1), \phi(z_2), \ldots, \phi(z_M) \Big),
$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is a nonlinear activation function.

MFG PDE system

Reminder: (m,u) solving, on $[0,T]\times\mathbb{T}^d$,

$$\begin{cases} 0 = -\dfrac{\partial u}{\partial t}(t,x) - \nu\Delta u(t,x) + H(x, m(t,\cdot), \nabla\, u(t,x)) \\ 0 = \dfrac{\partial m}{\partial t}(t,x) - \nu\Delta m(t,x) - \mathrm{div}\left(m(t,\cdot)\partial_p H(\cdot, m(t), \nabla\, u(t,\cdot))\right)(x) \\ \ u(T,x) = g(x, m(T,\cdot)), \qquad m(0,x) = m_0(x) \end{cases}$$

MFG PDE system

Reminder: (m, u) solving, on $[0, T] \times \mathbb{T}^d$,

$$
\begin{cases}
0 = -\dfrac{\partial u}{\partial t}(t,x) - \nu \Delta u(t,x) + H(x, m(t,\cdot), \nabla u(t,x)) \\
0 = \dfrac{\partial m}{\partial t}(t,x) - \nu \Delta m(t,x) - \operatorname{div}\left(m(t,\cdot)\partial_p H(\cdot, m(t), \nabla u(t,\cdot))\right)(x) \\
\quad u(T,x) = g(x, m(T,\cdot)), \qquad m(0,x) = m_0(x)
\end{cases}
$$

Or ergodic version: (m, u, λ) on $\mathbb{T}^d$

$$
\begin{cases}
0 = -\nu \Delta u(x) + H(x, m(\cdot), \nabla u(x)) + \lambda \\
0 = -\nu \Delta m(x) - \operatorname{div}\left(m(\cdot)\partial_p H(\cdot, m, \nabla u(\cdot))\right)(x) \\
\quad \displaystyle\int u(x)dx = 0, \qquad \int m(x)dx = 1, m > 0
\end{cases}
$$

See [Lasry and Lions, 2007], Chapter 7 in [Bensoussan et al., 2013]

MFG PDE system

Reminder: (m, u) solving, on $[0, T] \times \mathbb{T}^d$,

$$\begin{cases} 0 = -\dfrac{\partial u}{\partial t}(t,x) - \nu \Delta u(t,x) + H(x, m(t,\cdot), \nabla u(t,x)) \\ 0 = \dfrac{\partial m}{\partial t}(t,x) - \nu \Delta m(t,x) - \operatorname{div}\left(m(t,\cdot)\partial_p H(\cdot, m(t), \nabla u(t,\cdot))\right)(x) \\ \ u(T,x) = g(x, m(T,\cdot)), \qquad m(0,x) = m_0(x) \end{cases}$$

Or ergodic version: (m, u, λ) on $\mathbb{T}^d$

$$\begin{cases} 0 = -\nu \Delta u(x) + H(x, m(\cdot), \nabla u(x)) + \lambda \\ 0 = -\nu \Delta m(x) - \operatorname{div}\left(m(\cdot)\partial_p H(\cdot, m, \nabla u(\cdot))\right)(x) \\ \displaystyle\int u(x)dx = 0, \qquad \int m(x)dx = 1, m > 0 \end{cases}$$

See [Lasry and Lions, 2007], Chapter 7 in [Bensoussan et al., 2013]

There are analogous PDE systems for MFC problems

Inspired by [Almulla et al., 2017]

Example

Ergodic MFC with explicit solution on $\mathbb{T}^d$. Take:

$$f(x, m, \alpha) = \frac{1}{2}|\alpha|^2 + \tilde{f}(x) + \ln(m(x)),$$

with

$$\tilde{f}(x) = 2\pi^2 \left[-\sum_{i=1}^{d} c \sin(2\pi x_i) + \sum_{i=1}^{d} |c \cos(2\pi x_i)|^2 \right] - 2 \sum_{i=1}^{d} c \sin(2\pi x_i),$$

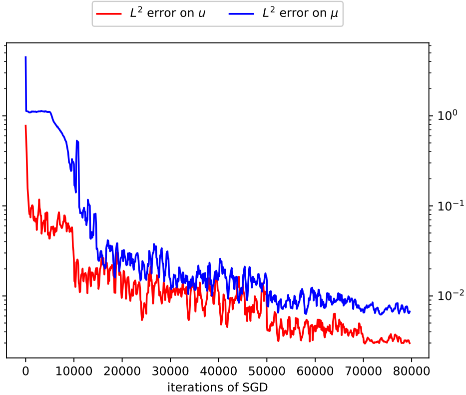
then the solution is given by

$$u(x) = c \sum_{i=1}^{d} \sin(2\pi x_i), \qquad m(x) = \frac{e^{2u(x)}}{\int e^{2u}}$$

Numerical Illustration 1: Ergodic Example with Explicit Solution

Numerical experiments in dimension $d = 10$

Error vs SGD iterations:



Relative L^2 error on u and m

More details in [Carmona and Laurière, 2021a]

Example of MFG without explicit solution on $\mathbb{T}^d$ inspired by [Achdou and Capuzzo-Dolcetta, 2010]

Example

Take:

$$f(x, m, \alpha) = \frac{1}{2}|\alpha|^2 + \tilde{f}(x) + |m(x)|^2,$$

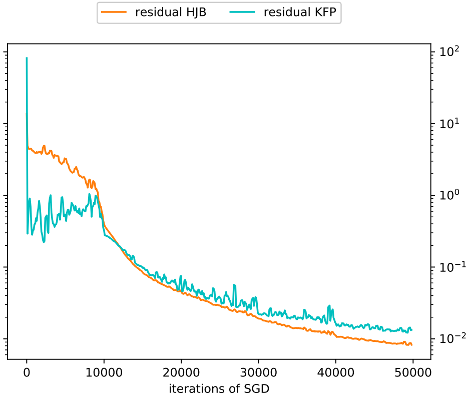
with

$$\tilde{f}(x) = 2\pi^2 c \sum_{i=1}^{d} [\sin(2\pi x_i) + \cos(2\pi x_i)]$$

Numerical Illustration 2: Ergodic Example without Explicit Solution

Numerical experiments in dimension $d = 30$

PDE residuals (training loss) vs SGD iterations:



L^2 norm of residuals for HJB and KFP

More details in [Carmona and Laurière, 2021a]

Sample code

Code

Sample code to illustrate: IPython notebook

`https://colab.research.google.com/drive/1xqamOTOCw7LRVxCMo1TECGM7st6XeB0H?usp=sharing`

- Ergodic mean field PDE system
- Solved by DGM

Numerical Illustration 3: Crowd Trading

Example

Model of crowd trading by [Cardaliaguet and Lehalle, 2018]:

$$\begin{cases} dS_t^{\bar{\nu}} = \gamma \bar{\nu}_t dt + \sigma dW_t & \text{(price)} \\ dQ_t^{\alpha} = \alpha_t dt & \text{(player's inventory)} \\ dX_t^{\alpha,\bar{\nu}} = -\alpha_t (S_t^{\bar{\nu}} + \kappa \alpha_t) dt & \text{(player's wealth)} \end{cases}$$

Objective: given $(\bar{\nu}_t)_t$, maximize

$$\mathbb{E}\left[X_T^{\alpha,\bar{\nu}} + Q_T^{\alpha} S_T^{\bar{\nu}} - A|Q_T^{\alpha}|^2 - \phi \int_0^T |Q_t^{\alpha}|^2 dt \right]$$

where: $\phi, A > 0 \Rightarrow$ penalty for holding inventory

Numerical Illustration 3: Crowd Trading

Ansatz (see [Cartea and Jaimungal, 2016]):

$$V(t,x,s,q) = x + qsu(t,q), \quad \hat{\alpha}_t(q) = \frac{\partial_q u(t,q)}{2\kappa}$$

where $u(\cdot)$ solves

$$-\gamma\bar{\nu}q = \partial_t u - \phi q^2 + \sup_{\alpha}\{\alpha\partial_q u - \kappa\alpha^2\}, \qquad u(T,q) = -Aq^2$$

Numerical Illustration 3: Crowd Trading

Ansatz (see [Cartea and Jaimungal, 2016]):

$$V(t,x,s,q) = x + qs u(t,q), \quad \hat{\alpha}_t(q) = \frac{\partial_q u(t,q)}{2\kappa}$$

where $u(\cdot)$ solves

$$-\gamma \bar{\nu} q = \partial_t u - \phi q^2 + \sup_{\alpha}\{\alpha \partial_q u - \kappa \alpha^2\}, \qquad u(T,q) = -A q^2$$

Mean field term: at equilibrium

$$\bar{\nu}_t = \int \hat{\alpha}_t(q) \hat{m}(t,dq) = \int \frac{\partial_q \hat{u}(t,q)}{2\kappa} \hat{m}(t,dq),$$

where $\hat{m}$ solves the KFP equation:

$$m(0,\cdot) = m_0, \qquad \partial_t m + \partial_q \left(m \frac{\partial_q \hat{u}(t,q)}{2\kappa} \right) = 0$$

Numerical Illustration 3: Crowd Trading

Reduced forward-backward PDE system:

$$\begin{cases} 0 = -\partial_t u(t,q) + \phi q^2 - \dfrac{|\partial_q u(t,q)|^2}{4\kappa} = \gamma \bar{\nu}_t q \\ 0 = \partial_t m(t,q) + \partial_q \left(m(t,q) \dfrac{\partial_q u(t,q)}{2\kappa} \right) \\ \bar{\nu}_t = \displaystyle\int \frac{\partial_q u(t,q)}{2\kappa} m(t,q) dq \\ m(0,\cdot) = m_0, u(T,q) = -Aq^2. \end{cases}$$

Note: the interactions are through the action distribution
$\Rightarrow$ yields a non-local term involving both u and m

It can be estimated e.g. by Monte Carlo samples (for a fixed t, sample various q)

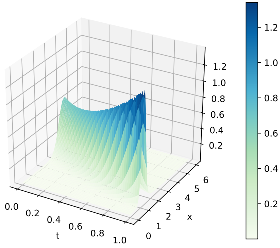
[Al-Aradi et al., 2019] applied DGM to this model.

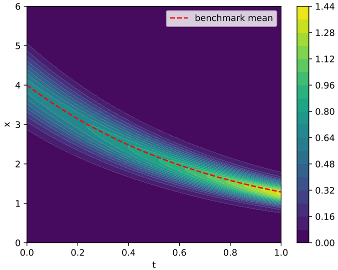
The results presented below are from [Carmona and Laurière, 2021b]

Numerical Illustration 3: Crowd Trading

Numerical results by DGM versus ODE solution

Evolution of m:



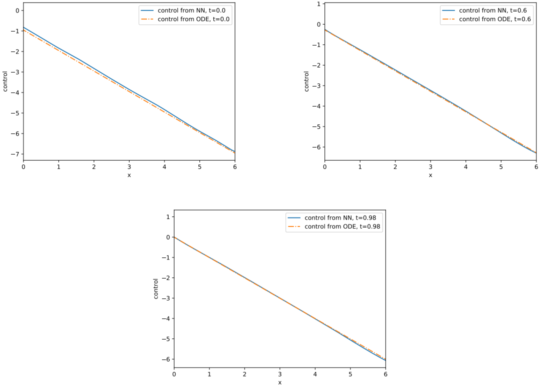


More details in [Carmona and Laurière, 2021b]

Numerical Illustration 3: Crowd Trading

Numerical results by DGM versus ODE solution

Evolution of equilibrium control $\hat{\alpha}$:



More details in [Carmona and Laurière, 2021b]

Numerical Illustration 3: Crowd Trading

- Convergence of DGM discussed in [Sirignano and Spiliopoulos, 2018]
  - By density, there exists a sequence of NN which approximates the solution and minimizes the DGM loss
  - Conversely, if the DGM loss is small, then the NN is close to the solution
- Similar analysis is possible for MFGs, see e.g. [Luo and Zheng, 2022]
- Variations and improvements, see e.g. [Reisinger et al., 2021]
- Obtaining (good) rates of convergence is challenging, even just for the approximation error
- Understanding the full generalization error remains challenging
- Application to other settings, e.g. mean field optimal transport [Baudelet et al., 2023], and the finite-state master equation (next section)

Outline

Master Equation

- Reminder: equilibrium: $(u, \mu) =$ sol. starting with m_0 at $t = 0$
- Idea: express the **value function** of a typical player as $u(t, x) = \mathcal{U}(t, x, \mu_t)$

Master Equation

- Reminder: equilibrium: $(u, \mu) =$ sol. starting with m_0 at $t = 0$
- Idea: express the **value function** of a typical player as $u(t,x) = \mathcal{U}(t, x, \mu_t)$
- Value function $\mathcal{U}$: PDE on the Wasserstein space

Master Equation

- Reminder: equilibrium: $(u, \mu) =$ sol. starting with m_0 at $t = 0$
- Idea: express the **value function** of a typical player as $u(t, x) = \mathcal{U}(t, x, \mu_t)$
- Value function $\mathcal{U}$: PDE on the Wasserstein space
- Motivations:
  - Convergence of N-player games, large deviation principles, ...
  - Unknown initial distribution μ_0
  - Macroscopic shocks, common noise
- From a practical viewpoint, if we know $\mathcal{U}$, then we know the optimal behavior of a representative player for **any** current distribution

Master Equation

- Reminder: equilibrium: $(u, \mu) =$ sol. starting with m_0 at $t = 0$
- Idea: express the **value function** of a typical player as $u(t, x) = \mathcal{U}(t, x, \mu_t)$
- Value function $\mathcal{U}$: PDE on the Wasserstein space
- Motivations:
  - Convergence of N-player games, large deviation principles, ...
  - Unknown initial distribution μ_0
  - Macroscopic shocks, common noise
- From a practical viewpoint, if we know $\mathcal{U}$, then we know the optimal behavior of a representative player for **any** current distribution
- How can we compute $\mathcal{U}$?

Outline

Finite state MFG:

- Finite state space $\mathcal{X}$
- $\mu \in \Delta^{|\mathcal{X}|}$
- $\dot{\mu}_t = \mu_t Q(\mu_t)$, $Q =$ transition rate matrix

Finite State MFGs

Finite state MFG:

- Finite state space $\mathcal{X}$
- $\mu \in \Delta^{|\mathcal{X}|}$
- $\dot{\mu}_t = \mu_t Q(\mu_t)$, $Q =$ transition rate matrix

Master PDE for $\mathcal{U}$:

$$\begin{cases} \mathcal{U}(T, x, \mu) = g(x, \mu) \\ -\partial_t \mathcal{U}(t, x, \mu) = \underbrace{H^*(t, x, \mu, \mathcal{U}(t, \cdot, \mu))}_{\text{Hamiltonian}} + \sum_{x' \in \mathcal{X}} \underbrace{\bar{Q}^*(t, \mu, \mathcal{U}(t, \cdot, \mu))(x')}_{\text{avg transition}} \underbrace{\frac{\partial \mathcal{U}(t, \cdot, \mu)}{\partial \mu(x')}}_{\text{classical deriv.}} \end{cases}$$

for $(t, x, \mu) \in [0, T] \times \mathcal{X} \times \Delta^{|\mathcal{X}|}$

Finite State MFGs

Finite state MFG:

- Finite state space $\mathcal{X}$
- $\mu \in \Delta^{|\mathcal{X}|}$
- $\dot{\mu}_t = \mu_t Q(\mu_t)$, $Q =$ transition rate matrix

Master PDE for $\mathcal{U}$:

$$\begin{cases} \mathcal{U}(T,x,\mu) = g(x,\mu) \\ -\partial_t \mathcal{U}(t,x,\mu) = \underbrace{H^*(t,x,\mu,\mathcal{U}(t,\cdot,\mu))}_{\text{Hamiltonian}} + \sum_{x' \in \mathcal{X}} \underbrace{\bar{Q}^*(t,\mu,\mathcal{U}(t,\cdot,\mu))(x')}_{\text{avg transition}} \underbrace{\frac{\partial \mathcal{U}(t,\cdot,\mu)}{\partial \mu(x')}}_{\text{classical deriv.}} \end{cases}$$

for $(t,x,\mu) \in [0,T] \times \mathcal{X} \times \Delta^{|\mathcal{X}|}$

Numerical solution using the DGM described above

Example 1: Cyber-Security Model

Example (Cyber-security model [Kolokoltsov and Bensoussan, 2016])

- **State space:** $\mathcal{X} = \{DI, DS, UI, US\}$
  - defended/unprotected
  - infected/susceptible
- **Actions:** want to switch level of protection; event happens at rate $\alpha\lambda$
  - $\alpha = 1$ (want to switch level of protection)
  - or 0 (happy)
- **Time:** continuous time, finite time horizon T

Example 1: Cyber-Security Model

Example (Cyber-security model [Kolokoltsov and Bensoussan, 2016])

- **State space:** $\mathcal{X} = \{DI, DS, UI, US\}$
  - defended/unprotected
  - infected/susceptible
- **Actions:** want to switch level of protection; event happens at rate $\alpha\lambda$
  - $\alpha = 1$ (want to switch level of protection)
  - or 0 (happy)
- **Time:** continuous time, finite time horizon T
- **Mean field interactions:** more infected units $\Rightarrow$ higher infection rate

Example 1: Cyber-Security Model

Example (Cyber-security model [Kolokoltsov and Bensoussan, 2016])

- **State space:** $\mathcal{X} = \{DI, DS, UI, US\}$
  - defended/unprotected
  - infected/susceptible
- **Actions:** want to switch level of protection; event happens at rate $\alpha\lambda$
  - $\alpha = 1$ (want to switch level of protection)
  - or 0 (happy)
- **Time:** continuous time, finite time horizon T
- **Mean field interactions:** more infected units $\Rightarrow$ higher infection rate

$$\dot{\mu}(t) = \mu(t) \underbrace{\begin{pmatrix} \dots & q_{\text{rec}}^{D} & \alpha\lambda & 0 \\ q_{\text{inf}}^{D} + \beta_{\text{D}}(\mu_{DI}(t) + \mu_{UI}(t)) & \dots & 0 & \alpha\lambda \\ \alpha\lambda & 0 & \dots & q_{\text{rec}}^{U} \\ 0 & \alpha\lambda & q_{\text{inf}}^{U} + \beta_{\text{U}}(\mu_{UI}(t) + \mu_{DI}(t)) & \dots \end{pmatrix}}_{\text{transition rates}}$$

Example 1: Cyber-Security Model

Example (Cyber-security model [Kolokoltsov and Bensoussan, 2016])

- **State space:** $\mathcal{X} = \{DI, DS, UI, US\}$
  - defended/unprotected
  - infected/susceptible
- **Actions:** want to switch level of protection; event happens at rate $\alpha\lambda$
  - $\alpha = 1$ (want to switch level of protection)
  - or 0 (happy)
- **Time:** continuous time, finite time horizon T
- **Mean field interactions:** more infected units $\Rightarrow$ higher infection rate

$$\dot{\mu}(t) = \mu(t) \underbrace{\begin{pmatrix} \dots & q^D_{\text{rec}} & \alpha\lambda & 0 \\ q^D_{\text{inf}} + \beta_{\text{D}}(\mu_{DI}(t) + \mu_{UI}(t)) & \dots & 0 & \alpha\lambda \\ \alpha\lambda & 0 & \dots & q^U_{\text{rec}} \\ 0 & \alpha\lambda & q^U_{\text{inf}} + \beta_{\text{U}}(\mu_{UI}(t) + \mu_{DI}(t)) & \dots \end{pmatrix}}_{\text{transition rates}}$$

- **Running cost:**

$$k_D 1_{\{DI,DS\}} + k_I 1_{\{DI,UI\}} = \text{cost of defense} + \text{penalty for being infected}$$

- **Terminal cost:** 0

Example 1: Cyber-Security Model

We apply the DGM. See [Laurière, 2021] for more details.

- Neural network: $\mathcal{U}_\theta$ to approximate $\mathcal{U}$
- Samples: Pick points $(t, x, \mu) \in [0, T] \times \mathcal{X} \times \Delta^{|\mathcal{X}|}$
- Loss: PDE residual + terminal condition

Comparison:

- $\mathcal{U}_\theta(t, x, \mu(t, \cdot))$
- $\mu(t, x)$, $u(t, x)$: finite state space $\rightarrow$ forward-backward ODE system

Example 1: Cyber-Security Model

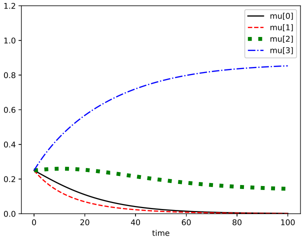
We apply the DGM. See [Laurière, 2021] for more details.

- Neural network: $\mathcal{U}_\theta$ to approximate $\mathcal{U}$
- Samples: Pick points $(t, x, \mu) \in [0, T] \times \mathcal{X} \times \Delta^{|\mathcal{X}|}$
- Loss: PDE residual + terminal condition

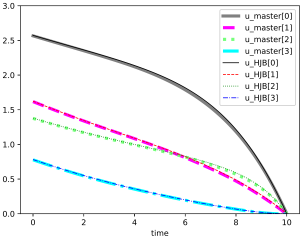
Comparison:

- $\mathcal{U}_\theta(t, x, \mu(t, \cdot))$
- $\mu(t, x)$, $u(t, x)$: finite state space $\to$ forward-backward ODE system

Test 1: $m_0 = (1/4, 1/4, 1/4, 1/4)$



Evolution of μ



Evolution of $u, \mathcal{U}$

Example 1: Cyber-Security Model

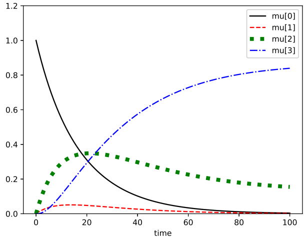
We apply the DGM. See [Laurière, 2021] for more details.

- Neural network: $\mathcal{U}_\theta$ to approximate $\mathcal{U}$
- Samples: Pick points $(t, x, \mu) \in [0, T] \times \mathcal{X} \times \Delta^{|\mathcal{X}|}$
- Loss: PDE residual + terminal condition

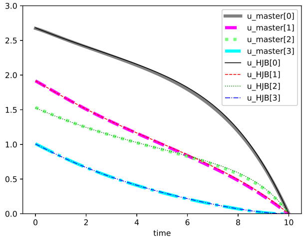
Comparison:

- $\mathcal{U}_\theta(t, x, \mu(t, \cdot))$
- $\mu(t, x)$, $u(t, x)$: finite state space $\rightarrow$ forward-backward ODE system

Test 2: $m_0 = (1, 0, 0, 0)$



Evolution of μ



Evolution of $u, \mathcal{U}$

Example 1: Cyber-Security Model

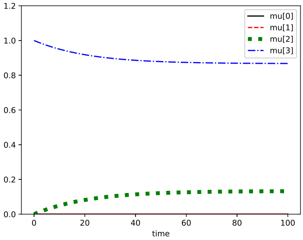
We apply the DGM. See [Laurière, 2021] for more details.

- Neural network: $\mathcal{U}_\theta$ to approximate $\mathcal{U}$
- Samples: Pick points $(t, x, \mu) \in [0, T] \times \mathcal{X} \times \Delta^{|\mathcal{X}|}$
- Loss: PDE residual + terminal condition

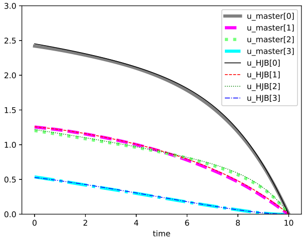
Comparison:

- $\mathcal{U}_\theta(t, x, \mu(t, \cdot))$
- $\mu(t, x)$, $u(t, x)$: finite state space $\rightarrow$ forward-backward ODE system

Test 3: $m_0 = (0, 0, 0, 1)$



Evolution of μ

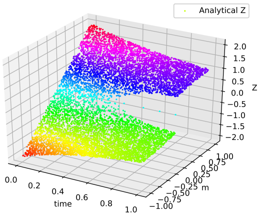


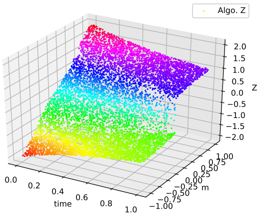
Evolution of $u, \mathcal{U}$

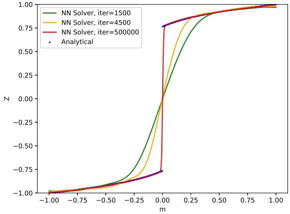
Example 2: Entropic solution

Example of a 2-state MFG [Cecchin et al., 2019] with

- multiple solutions to the master equation
- a unique one is an entropic solution







More details in [Laurière, 2021], section 7.2

Extensions

Some ongoing works:

- Analysis of the DGM convergence for finite-state master equation (ongoing work with Asaf Cohen and Ethan Zell)

- Application to (continuous space) macroeconomic models, joint work with Jonathan Payne and Sebastian Merkel. Working draft on Jonathan's webpage.

Outline

Master Bellman Equation for MFC

- **MFC problem** in **continuous space** with common noise:

$$J^{MFC}(\alpha) = \mathbb{E}\Big[\int_0^T f(X_t, \mathbb{P}^0_{X_t}, \alpha_t)dt + g(X_T, \mathbb{P}^0_{X_T})\Big].$$

subj. to: $dX_t = b(X_t, \mathbb{P}^0_{X_t}, \alpha_t)dt + \sigma dW_t + \sigma_0 dW^0_t$,
where $\mathbb{P}^0_{X_t}$ = conditional law of X_t given the common noise W^0

Master Bellman Equation for MFC

- **MFC problem** in **continuous space** with common noise:

$$J^{MFC}(\alpha) = \mathbb{E}\Big[\int_0^T f(X_t, \mathbb{P}^0_{X_t}, \alpha_t)dt + g(X_T, \mathbb{P}^0_{X_T})\Big].$$

subj. to: $dX_t = b(X_t, \mathbb{P}^0_{X_t}, \alpha_t)dt + \sigma dW_t + \sigma_0 dW^0_t$,
where $\mathbb{P}^0_{X_t}$ = conditional law of X_t given the common noise W^0

- **Master Bellman equation** in the Wasserstein space $\mathcal{P}_2(\mathbb{R}^d)$:

$$\begin{cases} \partial_t V + \mathcal{F}(\mu, V, \partial_\mu V, \partial_x \partial_\mu V, \partial^2_\mu V) & = & 0, & (t,\mu) \in [0,T) \in \mathcal{P}_2(\mathbb{R}^d) \\ V(T,\mu) & = & \mathcal{G}(\mu), & \mu \in \mathcal{P}_2(\mathbb{R}^d), \end{cases}$$

where:

Master Bellman Equation for MFC

- **MFC problem** in **continuous space** with common noise:

$$J^{MFC}(\alpha) = \mathbb{E}\Big[\int_0^T f(X_t, \mathbb{P}^0_{X_t}, \alpha_t)dt + g(X_T, \mathbb{P}^0_{X_T})\Big].$$

subj. to: $dX_t = b(X_t, \mathbb{P}^0_{X_t}, \alpha_t)dt + \sigma dW_t + \sigma_0 dW^0_t$,
where $\mathbb{P}^0_{X_t}$ = conditional law of X_t given the common noise W^0

- **Master Bellman equation** in the Wasserstein space $\mathcal{P}_2(\mathbb{R}^d)$:

$$\begin{cases} \partial_t V + \mathcal{F}(\mu, V, \partial_\mu V, \partial_x\partial_\mu V, \partial^2_\mu V) & = & 0, & (t,\mu) \in [0,T) \in \mathcal{P}_2(\mathbb{R}^d) \\ V(T,\mu) & = & \mathcal{G}(\mu), & \mu \in \mathcal{P}_2(\mathbb{R}^d), \end{cases}$$

where:

- $\partial_\mu V(\mu)(.) : \mathbb{R}^d \to \mathbb{R}^d$, $\partial_x\partial_\mu V(\mu)(.) : \mathbb{R}^d \to \mathbb{S}^d$, $\partial^2_\mu V(\mu)(.,.) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{S}^d$, are the L-derivatives of V on $\mathcal{P}_2(\mathbb{R}^d)$ (see [Carmona and Delarue, 2018], Chapter 5)
- and

$$\mathcal{F}(\mu, y, Z(.), \Gamma(.), \Gamma_0(.,.)) = \int_{\mathbb{R}^d} h(x,\mu,Z(x),\Gamma(x))\mu(dx) + \int_{\mathbb{R}^d\times\mathbb{R}^d} \tfrac{1}{2}\mathrm{tr}\Big(\sigma_0\sigma_0^\intercal \Gamma_0(x,x')\Big)\mu(dx)\mu(dx'),$$

$$\mathcal{G}(\mu) = \int_{\mathbb{R}^d} g(x,\mu)\mu(dx),$$

$$h(x,\mu,z,\gamma) = \inf_{a\in A}\Big[b(x,\mu,a).z + \frac{1}{2}\mathrm{tr}\Big(\sigma\sigma^\intercal\gamma\Big) + f(x,\mu,a)\Big].$$

Symmetric Neural Networks

- How can we solve the Bellman PDE and compute V?
- Idea: approximate V by a NN and use backward induction
- Challenge: How can we represent μ and input it to the neural network?

Symmetric Neural Networks

- How can we solve the Bellman PDE and compute V?
- Idea: approximate V by a NN and use backward induction
- Challenge: How can we represent μ and input it to the neural network?
- Connection between N-agents problem and MFC: $\mu^N = \frac{1}{N}\sum_{i=1}^N \delta_{x^i}$

$$v^N(t, x, x^1, \dots, x^N) = V^N(t, x, \mu^N) \xrightarrow[N \to +\infty]{} V(t, x, \mu^N)$$

- Approximate $V(t, x, \cdot)$ by a **symmetric** function of N inputs (N large)

Symmetric Neural Networks

- How can we solve the Bellman PDE and compute V?
- Idea: approximate V by a NN and use backward induction
- Challenge: How can we represent μ and input it to the neural network?
- Connection between N-agents problem and MFC: $\mu^N = \frac{1}{N}\sum_{i=1}^N \delta_{x^i}$

$$v^N(t,x,x^1,\dots,x^N) = V^N(t,x,\mu^N) \xrightarrow[N\to+\infty]{} V(t,x,\mu^N)$$

- Approximate $V(t,x,\cdot)$ by a **symmetric** function of N inputs (N large)
- **Symmetric Neural Networks:**
  - Symmetry by construction; e.g. with a sum:

$$(x^i)_{i=1,\dots,N} \mapsto \sum_{i=1}^N \psi_\omega(x^i) \mapsto \varphi_\theta\left(\sum_{i=1}^N \psi_\omega(x^i)\right)$$

  - DeepSets [Zaheer et al., 2017], PointNet [Qi et al., 2017], ...

Deep Backward Dynamic Programming for MFC

Deep Learning for MFC with DPP and Symmetric NN [Germain et al., 2021a]

- **Symmetric NN:** $\mathcal{V}(t, x^1, \dots, x^N)$
- **D-Symmetric NN:** sym. except in one space variable:

$$\mathcal{Z}(x^1, \dots, x^N, x^i) \leftrightarrow \partial_{x^i} \mathcal{V}(x^1, \dots, x^N) = \frac{1}{N} \partial_\mu \mathcal{V} \left(\frac{1}{N} \sum_j x^j \right) (x^i)$$

Output: $(\widehat{\mathcal{V}}_n, \widehat{\mathcal{Z}}_n)_{n=0,\dots,N_T}$ s.t. $\widehat{\mathcal{V}}_n(\underline{x}) \approx V(t_n, \mu^N_{\underline{x}})$,
$\widehat{\mathcal{Z}}_n(\underline{x}, x^i) \approx \frac{1}{N} \partial_\mu V(t_n, \mu^N_{\underline{x}})(x^i)$

1. Set $\widehat{\mathcal{V}}_{N_T}(\cdot) = G(\cdot)$
2. **for** $n = N_T - 1, N_T - 2, \dots, 1, 0$ **do**
3. Compute $(\widehat{\mathcal{V}}_n, \widehat{\mathcal{Z}}_n)$ as a minimizer of:

$$(\mathcal{V}_n, \mathcal{Z}_n) \mapsto \mathbb{E}\Big| \widehat{\mathcal{V}}_{n+1}(\mathbf{X}_{n+1}) - \mathcal{V}_n(\mathbf{X}_n) + H\big(t_n, \mathbf{X}_n, \mathcal{V}_n(\mathbf{X}_n), \mathbf{Z}_n(\mathbf{X}_n)\big)\Delta t - \sum_{i=1}^N \sum_{j=0}^N \big(\mathcal{Z}_n(\mathbf{X}_n, X_n^i)\big)^\intercal \sigma_{ij} \Delta W_n^j \Big|^2,$$

where $\widehat{\mathcal{V}}_n$ is a sym. NN, $\widehat{\mathcal{Z}}_n$ is a D-sym. NN, $H =$ sym. version of h

4. **return** $(\widehat{\mathcal{V}}_n, \widehat{\mathcal{Z}}_n)_{n=0,\dots,N_T}$

Deep Backward Dynamic Programming for MFC

Deep Learning for MFC with DPP and Symmetric NN [Germain et al., 2021a]

- **Symmetric NN:** $\mathcal{V}(t, x^1, \dots, x^N)$
- **D-Symmetric NN:** sym. except in one space variable:

$$\mathcal{Z}(x^1, \dots, x^N, x^i) \leftrightarrow \partial_{x^i} \mathcal{V}(x^1, \dots, x^N) = \frac{1}{N} \partial_\mu \mathcal{V} \left(\frac{1}{N} \sum_j x^j \right) (x^i)$$

Output: $(\widehat{\mathcal{V}}_n, \widehat{\mathcal{Z}}_n)_{n=0,\dots,N_T}$ s.t. $\widehat{\mathcal{V}}_n(\underline{x}) \approx V(t_n, \mu^N_{\underline{x}})$,
$\widehat{\mathcal{Z}}_n(\underline{x}, x^i) \approx \frac{1}{N} \partial_\mu V(t_n, \mu^N_{\underline{x}})(x^i)$

Set $\widehat{\mathcal{V}}_{N_T}(\cdot) = G(\cdot)$

for $n = N_T - 1, N_T - 2, \dots, 1, 0$ **do**

Compute $(\widehat{\mathcal{V}}_n, \widehat{\mathcal{Z}}_n)$ as a minimizer of:

$$(\mathcal{V}_n, \mathcal{Z}_n) \mapsto \mathbb{E} \Big| \widehat{\mathcal{V}}_{n+1}(\mathbf{X}_{n+1}) - \mathcal{V}_n(\mathbf{X}_n) + H\big(t_n, \mathbf{X}_n, \mathcal{V}_n(\mathbf{X}_n), \mathbf{Z}_n(\mathbf{X}_n)\big)\Delta t - \sum_{i=1}^{N} \sum_{j=0}^{N} \big(\mathcal{Z}_n(\mathbf{X}_n, X^i_n)\big)^\intercal \sigma_{ij} \Delta W^j_n \Big|^2,$$

where $\widehat{\mathcal{V}}_n$ is a sym. NN, $\widehat{\mathcal{Z}}_n$ is a D-sym. NN, $H =$ sym. version of h

return $(\widehat{\mathcal{V}}_n, \widehat{\mathcal{Z}}_n)_{n=0,\dots,N_T}$

See [Germain et al., 2021a] for numerical results and more details about the implementation, and [Germain et al., 2022] for the analysis

Outline

1. Introduction

2. Deep Galerkin Method for MFG PDEs

3. Master Equation

4. Conclusion

Summary

- Deep Galerkin Method principle
  - Application to solve FB PDE system
  - Application to solve finite-state Master equations
- Deep Backward Dynamic Programming & symmetric NN
  - Application to compute the value function of MFC
- Many open questions for mathematicians (proofs of approximation, rates of convergence, . . .)

More references

The presentation in this lecture and the previous one is not exhaustive. Other works, such as: [Ruthotto et al., 2020], and works on the connection between (variational) MFGs and Generative Adversarial Networks (GANs) [Cao et al., 2020], [Lin et al., 2020].

Surveys on deep learning for:

- PDEs [Beck et al., 2020]
- Stochastic control and PDEs in finance: [Germain et al., 2021b]
- Stochastic control and games: [Hu and Laurière, 2023]

Thank you for your attention

Questions?

Feel free to reach out: mathieu.lauriere@nyu.edu

References I

[Achdou and Capuzzo-Dolcetta, 2010] Achdou, Y. and Capuzzo-Dolcetta, I. (2010).
Mean field games: numerical methods.
SIAM J. Numer. Anal., 48(3):1136–1162.

[Al-Aradi et al., 2019] Al-Aradi, A., Correia, A., Naiff, D. d. F., Jardim, G., and Saporito, Y. (2019).
Applications of the deep galerkin method to solving partial integro-differential and hamilton-jacobi-bellman equations.
arXiv preprint arXiv:1912.01455.

[Almulla et al., 2017] Almulla, N., Ferreira, R., and Gomes, D. (2017).
Two numerical approaches to stationary mean-field games.
Dyn. Games Appl., 7(4):657–682.

[Baudelet et al., 2023] Baudelet, S., Frénais, B., Laurière, M., Machtalay, A., and Zhu, Y. (2023).
Deep learning for mean field optimal transport.
arXiv preprint arXiv:2302.14739.

[Beck et al., 2020] Beck, C., Hutzenthaler, M., Jentzen, A., and Kuckuck, B. (2020).
An overview on deep learning-based approximation methods for partial differential equations.
arXiv preprint arXiv:2012.12348.

[Bensoussan et al., 2013] Bensoussan, A., Frehse, J., and Yam, S. C. P. (2013).
Mean field games and mean field type control theory.
Springer Briefs in Mathematics. Springer, New York.

[Cao et al., 2020] Cao, H., Guo, X., and Laurière, M. (2020).
Connecting GANs, MFGs, and OT.
arXiv preprint arXiv:2002.04112.

[Cardaliaguet and Lehalle, 2018] Cardaliaguet, P. and Lehalle, C.-A. (2018).
Mean field game of controls and an application to trade crowding.
Mathematics and Financial Economics, 12(3):335–363.

[Carmona and Delarue, 2018] Carmona, R. and Delarue, F. (2018).
Probabilistic theory of mean field games with applications. I, volume 83 of *Probability Theory and Stochastic Modelling*.
Springer, Cham.
Mean field FBSDEs, control, and games.

[Carmona and Laurière, 2021a] Carmona, R. and Laurière, M. (2021a).
Convergence analysis of machine learning algorithms for the numerical solution of mean field control and games i: The ergodic case.
SIAM Journal on Numerical Analysis, 59(3):1455–1485.

[Carmona and Laurière, 2021b] Carmona, R. and Laurière, M. (2021b).
Deep learning for mean field games and mean field control with applications to finance.
arXiv preprint arXiv:2107.04568.

References III

[Cartea and Jaimungal, 2016] Cartea, Á. and Jaimungal, S. (2016).
Incorporating order-flow into optimal execution.
Mathematics and Financial Economics, 10(3):339–364.

[Cecchin et al., 2019] Cecchin, A., Pra, P. D., Fischer, M., and Pelino, G. (2019).
On the convergence problem in mean field games: a two state model without uniqueness.
SIAM Journal on Control and Optimization, 57(4):2443–2466.

[Domingo-Enrich et al., 2020] Domingo-Enrich, C., Jelassi, S., Mensch, A., Rotskoff, G., and Bruna, J. (2020).
A mean-field analysis of two-player zero-sum games.
arXiv preprint arXiv:2002.06277.

[Germain et al., 2021a] Germain, M., Laurière, M., Pham, H., and Warin, X. (2021a).
Deepsets and their derivative networks for solving symmetric pdes.
arXiv preprint arXiv:2103.00838.

[Germain et al., 2021b] Germain, M., Pham, H., and Warin, X. (2021b).
Neural networks-based algorithms for stochastic control and pdes in finance.
arXiv preprint arXiv:2101.08068.

[Germain et al., 2022] Germain, M., Pham, H., and Warin, X. (2022).
Rate of convergence for particle approximation of pdes in wasserstein space.
Journal of Applied Probability, 59(4):992–1008.

References IV

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014).
Generative adversarial nets.
Advances in neural information processing systems, 27.

[Hu and Laurière, 2023] Hu, R. and Laurière, M. (2023).
Recent developments in machine learning methods for stochastic control and games.
arXiv preprint arXiv:2303.10257.

[Karras et al., 2020] Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., and Aila, T. (2020).
Analyzing and improving the image quality of stylegan.
In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119.

[Kolokoltsov and Bensoussan, 2016] Kolokoltsov, V. N. and Bensoussan, A. (2016).
Mean-field-game model for botnet defense in cyber-security.
Appl. Math. Optim., 74(3):669–692.

[Lasry and Lions, 2007] Lasry, J.-M. and Lions, P.-L. (2007).
Mean field games.
Jpn. J. Math., 2(1):229–260.

[Laurière, 2021] Laurière, M. (2021).
Numerical methods for mean field games and mean field type control.
arXiv preprint arXiv:2106.06231.

[Lin et al., 2020] Lin, A. T., Fung, S. W., Li, W., Nurbekyan, L., and Osher, S. J. (2020).
Apac-net: Alternating the population and agent control via two neural networks to solve high-dimensional stochastic mean field games.
arXiv preprint arXiv:2002.10113.

[Luo and Zheng, 2022] Luo, J. and Zheng, H. (2022).
Deep neural network solution for finite state mean field game with error estimation.
Dynamic Games and Applications, pages 1–38.

[Qi et al., 2017] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017).
Pointnet: Deep learning on point sets for 3d classification and segmentation.
In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660.

[Raissi et al., 2019] Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019).
Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations.
Journal of Computational physics, 378:686–707.

References VI

[Reisinger et al., 2021] Reisinger, C., Stockinger, W., and Zhang, Y. (2021).
A fast iterative pde-based algorithm for feedback controls of nonsmooth mean-field control problems.
arXiv preprint arXiv:2108.06740.

[Ruthotto et al., 2020] Ruthotto, L., Osher, S. J., Li, W., Nurbekyan, L., and Fung, S. W. (2020).
A machine learning framework for solving high-dimensional mean field game and mean field control problems.
Proceedings of the National Academy of Sciences, 117(17):9183–9193.

[Sirignano and Spiliopoulos, 2018] Sirignano, J. and Spiliopoulos, K. (2018).
DGM: a deep learning algorithm for solving partial differential equations.
J. Comput. Phys., 375:1339–1364.

[Zaheer et al., 2017] Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R., and Smola, A. (2017).
Deep sets.
arXiv preprint arXiv:1703.06114.

5. Link with Generative Adversarial Networks

Examples

Examples

thispersondoesnotexist.com

thiscatdoesnotexist.com

Examples

`thispersondoesnotexist.com`

`thiscatdoesnotexist.com`

[Karras et al., 2020]

GANs & MFGs

Generative Adversarial Nets [Goodfellow et al., 2014]:

Setup: data space $\mathcal{X}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Generative Adversarial Nets [Goodfellow et al., 2014]:

Setup: data space $\mathcal{X}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{X}$ such that $p_z \circ G^{-1} \approx p_{data}$

GANs & MFGs

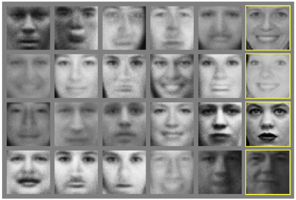
Generative Adversarial Nets [Goodfellow et al., 2014]:

Setup: data space $\mathcal{X}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{X}$ such that $p_z \circ G^{-1} \approx p_{data}$



[Goodfellow et al., 2014]

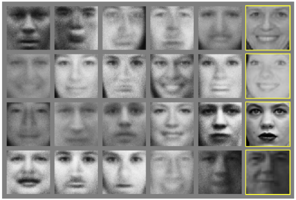
Generative Adversarial Nets [Goodfellow et al., 2014]:

Setup: data space $\mathcal{X}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{X}$ such that $p_z \circ G^{-1} \approx p_{data}$



[Goodfellow et al., 2014]

NVIDIA'19

Generative Adversarial Nets [Goodfellow et al., 2014]:

Setup: data space $\mathcal{X}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{X}$ such that $p_z \circ G^{-1} \approx p_{data}$

Idea++: also learn $D : \mathcal{X} \to \mathbb{R}$ to distinguish between samples from $p_z \circ G^{-1}$ and p_{data}

Generative Adversarial Nets [Goodfellow et al., 2014]:

Setup: data space $\mathcal{X}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{X}$ such that $p_z \circ G^{-1} \approx p_{data}$

Idea++: also learn $D : \mathcal{X} \to \mathbb{R}$ to distinguish between samples from $p_z \circ G^{-1}$ and p_{data}

Mathematically: min-max game between two neural networks D_δ, G_γ (params: δ, γ)

$$\min_\gamma \max_\delta \left\{ \mathbb{E}_{x \sim \mathbb{P}_r}[\log D_\delta(x)] + \mathbb{E}_{z \sim \mathbb{P}_z}[\log(1 - D_\delta(G_\gamma(z)))] \right\}.$$

Generative Adversarial Nets [Goodfellow et al., 2014]:

Setup: data space $\mathcal{X}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{X}$ such that $p_z \circ G^{-1} \approx p_{data}$

Idea++: also learn $D : \mathcal{X} \to \mathbb{R}$ to distinguish between samples from $p_z \circ G^{-1}$ and p_{data}

Mathematically: min-max game between two neural networks D_δ, G_γ (params: δ, γ)

$$\min_{\gamma} \max_{\delta} \Big\{ \mathbb{E}_{x \sim \mathbb{P}_r} [\log D_\delta(x)] + \mathbb{E}_{z \sim \mathbb{P}_z} [\log(1 - D_\delta(G_\gamma(z)))] \Big\}.$$

Variational MFG: $\inf_{u:[0,T]\times\mathbb{R}^d \to \mathbb{R}} \sup_{m:[0,T]\times\mathbb{R}^d \to \mathbb{R}} \Phi(m,u)$, where

$$\Phi(m,u) = \int_0^T \int_{\mathbb{T}^d} [m(-\partial_t u - \epsilon \Delta_x u) + m H(x, \nabla_x u, m)]\, dx dt + \int_{\mathbb{T}^d} [m(T) u(T) - m_0 u(0)]\, dx$$

GANs & MFGs

Generative Adversarial Nets [Goodfellow et al., 2014]:

Setup: data space $\mathcal{X}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{X}$ such that $p_z \circ G^{-1} \approx p_{data}$

Idea++: also learn $D : \mathcal{X} \to \mathbb{R}$ to distinguish between samples from $p_z \circ G^{-1}$ and p_{data}

Mathematically: min-max game between two neural networks D_δ, G_γ (params: δ, γ)

$$\min_\gamma \max_\delta \left\{ \mathbb{E}_{x\sim\mathbb{P}_r}[\log D_\delta(x)] + \mathbb{E}_{z\sim\mathbb{P}_z}[\log(1 - D_\delta(G_\gamma(z)))] \right\}.$$

Variational MFG: $\inf_{u:[0,T]\times\mathbb{R}^d\to\mathbb{R}} \quad \sup_{m:[0,T]\times\mathbb{R}^d\to\mathbb{R}} \Phi(m,u)$, where

$$\Phi(m,u) = \int_0^T \int_{\mathbb{T}^d} [m(-\partial_t u - \epsilon\Delta_x u) + mH(x, \nabla_x u, m)]\, dx dt + \int_{\mathbb{T}^d} [m(T)u(T) - m_0 u(0)]\, dx$$

$\to$ Conceptual connection GANs/MFGs: [Cao et al., 2020]

GANs & MFGs

Generative Adversarial Nets [Goodfellow et al., 2014]:

Setup: data space $\mathcal{X}$ (e.g. images of fixed size); *unknown* data distribution p_{data}

Goal: be able to generate samples according p_{data}

Given: samples from data, and random noise generator p_z over some space $\mathcal{Z}$

Idea: learn $G : \mathcal{Z} \to \mathcal{X}$ such that $p_z \circ G^{-1} \approx p_{data}$

Idea++: also learn $D : \mathcal{X} \to \mathbb{R}$ to distinguish between samples from $p_z \circ G^{-1}$ and p_{data}

Mathematically: min-max game between two neural networks D_δ, G_γ (params: δ, γ)

$$\min_{\gamma} \max_{\delta} \Big\{ \mathbb{E}_{x \sim \mathbb{P}_r} [\log D_\delta(x)] + \mathbb{E}_{z \sim \mathbb{P}_z} [\log(1 - D_\delta(G_\gamma(z)))] \Big\}.$$

Variational MFG: $\inf_{u:[0,T]\times\mathbb{R}^d \to \mathbb{R}} \sup_{m:[0,T]\times\mathbb{R}^d \to \mathbb{R}} \Phi(m, u)$, where

$$\Phi(m,u) = \int_0^T \int_{\mathbb{T}^d} [m(-\partial_t u - \epsilon \Delta_x u) + m H(x, \nabla_x u, m)] \, dx dt + \int_{\mathbb{T}^d} [m(T)u(T) - m_0 u(0)] \, dx$$

$\to$ Conceptual connection GANs/MFGs: [Cao et al., 2020]

Related work: [Domingo-Enrich et al., 2020], [Lin et al., 2020], ...