Mean Field Games: Numerical Methods and Applications in Machine Learning

Part 6: Deep Learning for MFG PDEs

Mathieu LAURIÈRE

https://mlauriere.github.io/teaching/MFG-PKU-6.pdf

Peking University
Summer School on Applied Mathematics
July 26 – August 6, 2021

RECAP

Outline

1. Deep Galerkin Method for MFG PDEs
   - Warm-up: ODE
   - Solving MFG PDE system
   - Link with Generative Adversarial Networks

Outline

1. Deep Galerkin Method for MFG PDEs
    - Warm-up: ODE
    - Solving MFG PDE system
    - Link with Generative Adversarial Networks

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

$$F(x, \varphi(x), \varphi'(x), \dots) = 0, \qquad x \in [a, b]$$

Solving ODEs with Neural Networks

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

$$F(x, \varphi(x), \varphi'(x), \dots) = 0, \qquad x \in [a, b]$$

- Look among NN φ_θ

$$F(x, \varphi_\theta(x), \varphi'_\theta(x), \dots) = 0, \qquad x \in [a, b]$$

Solving ODEs with Neural Networks

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

$$F(x, \varphi(x), \varphi'(x), \dots) = 0, \qquad x \in [a, b]$$

- Look among NN φ_θ

$$F(x, \varphi_\theta(x), \varphi_\theta'(x), \dots) = 0, \qquad x \in [a, b]$$

- Rephrase as minimization problem: minimizer over θ

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

Solving ODEs with Neural Networks

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

$$F(x, \varphi(x), \varphi'(x), \dots) = 0, \qquad x \in [a, b]$$

- Look among NN φ_θ

$$F(x, \varphi_\theta(x), \varphi'_\theta(x), \dots) = 0, \qquad x \in [a, b]$$

- Rephrase as minimization problem: minimizer over θ

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

- Use SGD

Numerical Illustration

Application to the ODE:

$$F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x))$$

Numerical Illustration

Application to the ODE:

$$F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x))$$

Solution:

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

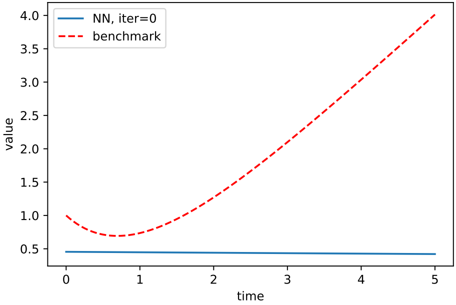
Numerical Illustration

Application to the ODE:

$$F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x))$$

Solution:

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



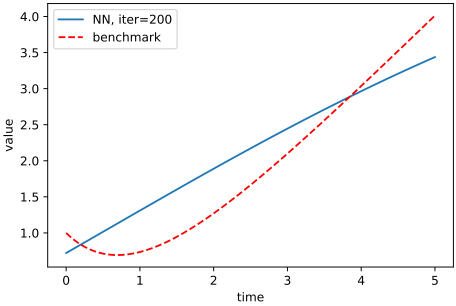
Numerical Illustration

Application to the ODE:

$$F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x))$$

Solution:

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



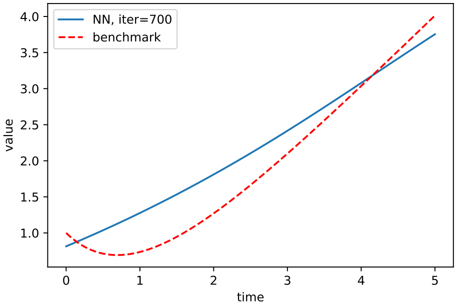
Numerical Illustration

Application to the ODE:

$$F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x))$$

Solution:

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



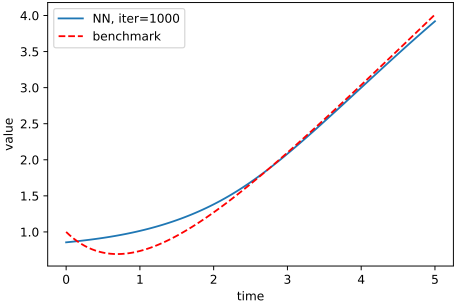
Numerical Illustration

Application to the ODE:

$$F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x))$$

Solution:

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



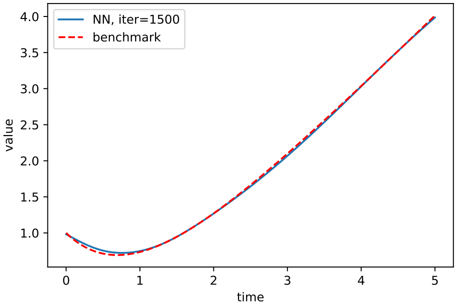
Numerical Illustration

Application to the ODE:

$$F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x))$$

Solution:

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



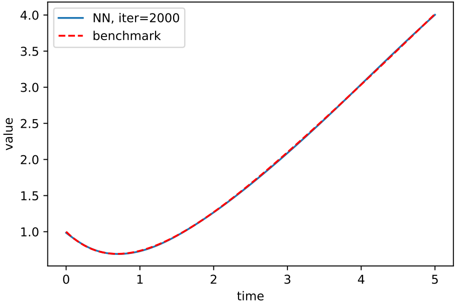
Numerical Illustration

Application to the ODE:

$$F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x))$$

Solution:

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



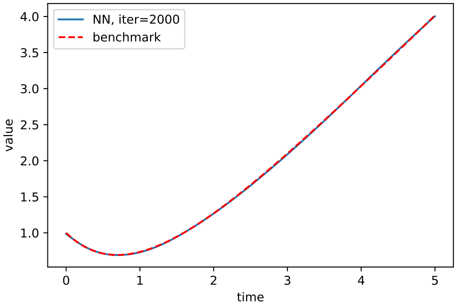
Numerical Illustration

Application to the ODE:

$$F(x, \varphi(x), \varphi'(x)) = \varphi'(x) - (x - \varphi(x))$$

Solution:

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



https://colab.research.google.com/drive/1LHuV1oE6eyO6AQgw3joQjow_uozQWSTw?usp=sharing

Outline

1. Deep Galerkin Method for MFG PDEs
   - Warm-up: ODE
   - Solving MFG PDE system
   - Link with Generative Adversarial Networks

Deep Galerkin Method (DGM), proposed by [Sirignano, Spiliopoulos][1]

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

$$F(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, \qquad x \in Q$$

[1] Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375, 1339-1364.

Solving PDEs with Neural Networks

Deep Galerkin Method (DGM), proposed by [Sirignano, Spiliopoulos][1]

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

$$F(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, \qquad x \in Q$$

- Look among NN φ_θ

$$F(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, \qquad x \in Q$$

[1] Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375, 1339-1364.

Solving PDEs with Neural Networks

Deep Galerkin Method (DGM), proposed by [Sirignano, Spiliopoulos][1]

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

$$F(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, \qquad x \in Q$$

- Look among NN φ_θ

$$F(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, \qquad x \in Q$$

- Rephrase as minimization problem: minimizer over θ

$$\mathbb{E}_{X \sim \mathcal{U}(Q)} \left[\left| F(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) \right|^2 \right]$$

[1] Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375, 1339-1364.

Solving PDEs with Neural Networks

Deep Galerkin Method (DGM), proposed by [Sirignano, Spiliopoulos][1]

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

$$F(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, \qquad x \in Q$$

- Look among NN φ_θ

$$F(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, \qquad x \in Q$$

- Rephrase as minimization problem: minimizer over θ

$$\mathbb{E}_{X \sim \mathcal{U}(Q)} \left[\left| F(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) \right|^2 \right]$$

- Use SGD

[1] Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375, 1339-1364.

Solving PDEs with Neural Networks

Deep Galerkin Method (DGM), proposed by [Sirignano, Spiliopoulos][1]

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

$$F(x, \varphi(x), D\varphi(x), D^2\varphi(x), \dots) = 0, \qquad x \in Q$$

- Look among NN φ_θ

$$F(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) = 0, \qquad x \in Q$$

- Rephrase as minimization problem: minimizer over θ

$$\mathbb{E}_{X \sim \mathcal{U}(Q)} \left[\left| F(x, \varphi_\theta(x), D\varphi_\theta(x), D^2\varphi_\theta(x), \dots) \right|^2 \right]$$

- Use SGD
- Remarks on the implementation:
  - Choice of distribution
  - Boundary conditions
  - Higher order derivatives computation

[1] Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375, 1339-1364.

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 $[0, T] \times \mathbb{T}^d$,

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

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, Lions'07; BFY'13, Chapter 7]

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, Lions'07; BFY'13, Chapter 7]

Analogous PDE systems for MFC problems

Example (of MFC) with explicit solution on $\mathbb{T}^d$ ($d = 10$)
Following [Almulla *et al.*'17], take

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

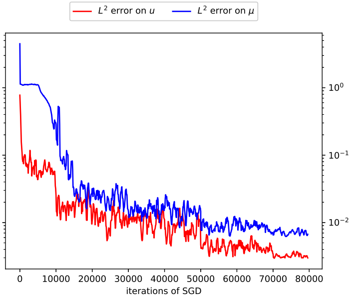
with $\tilde{f}(x) = 2\pi^2 \left[- \sum_{i=1}^d c \sin(2\pi x_i) + \sum_{i=1}^d |c \cos(2\pi x_i)|^2\right] - 2 \sum_{i=1}^d c \sin(2\pi x_i)$,
then the solution is given by $u(x) = c \sum_{i=1}^d \sin(2\pi x_i)$ and $m(x) = e^{2u(x)} / \int e^{2u}$

Example (of MFC) with explicit solution on $\mathbb{T}^d$ ($d = 10$)
Following [Almulla *et al.*'17], take

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

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

Error vs SGD iterations (see [Carmona, L.'21]):



Relative L^2 error on u and m

Numerical Illustration 2: Ergodic Example without Explicit Solution

Example (of MFG) without explicit solution on $\mathbb{T}^d$ ($d = 30$)
Inspired by [Achdou, Capuzzo-Dolcetta'11], take

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

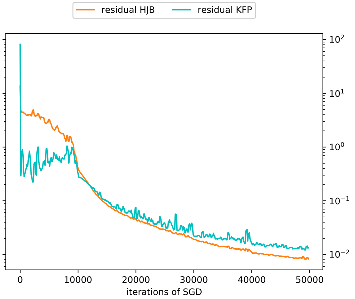
with $\tilde{f}(x) = 2\pi^2 c \sum_{i=1}^{d} [\sin(2\pi x_i) + \cos(2\pi x_i)]$

Example (of MFG) without explicit solution on $\mathbb{T}^d$ ($d = 30$)
Inspired by [Achdou, Capuzzo-Dolcetta'11], take

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

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

PDE residuals vs SGD iterations (see [Carmona, L.'21]):



L^2 norm of residuals for HJB and KFP

Numerical Illustration 3: Crowd Trading

Model of crowd trading [Cardaliaguet, Lehalle]:

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

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

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

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

Numerical Illustration 3: Crowd Trading

Model of crowd trading [Cardaliaguet, Lehalle]:

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

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

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

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

Ansatz [Cartea, Jaimungal]: $V(t,x,s,q) = x + qsu(t,q)$, $\qquad \hat{v}_t(q) = \frac{\partial_q u(t,q)}{2\kappa}$

where $u(\cdot)$ solves

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

Numerical Illustration 3: Crowd Trading

Model of crowd trading [Cardaliaguet, Lehalle]:

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

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

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

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

Ansatz [Cartea, Jaimungal]: $V(t,x,s,q) = x + qsu(t,q)$, $\quad \hat{v}_t(q) = \frac{\partial_q u(t,q)}{2\kappa}$

where $u(\cdot)$ solves

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

Mean field term: at equilibrium

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

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

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

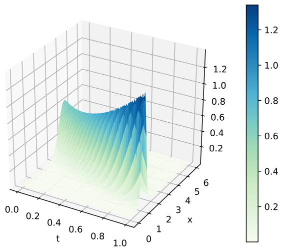
Reduced forward-backward PDE system:

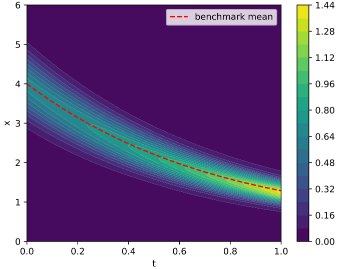
$$\begin{cases} 0 = -\partial_t u(t,q) + \phi q^2 - \dfrac{|\partial_q u(t,q)|^2}{4\kappa} = \gamma \bar{\nu}_t q \\ 0 = \partial_t m(t,q) + \partial_q \left(m(t,q) \dfrac{\partial_q u(t,q)}{2\kappa} \right) \\ \bar{\nu}_t = \displaystyle\int \frac{\partial_q u(t,q)}{2\kappa} m(t,q)\,dq \\ m(0,\cdot)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

Numerical Illustration 3: Crowd Trading

Numerical results obtained with DGM & comparison with ODE solution:

Evolution of m:

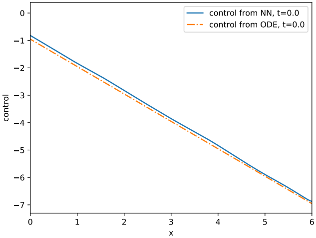


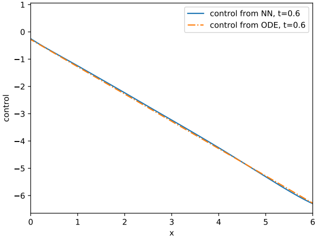


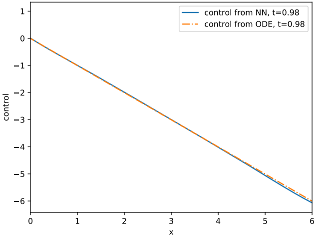
Numerical Illustration 3: Crowd Trading

Numerical results obtained with DGM & comparison with ODE solution:

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







Examples

Examples

thispersondoesnotexist.com

thiscatdoesnotexist.com

Examples

`thispersondoesnotexist.com`

`thiscatdoesnotexist.com`

[Karras *et al.*'20]: Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., & Aila, T. (2020). Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 8110-8119).

GANs & MFGs

Generative Adversarial Nets [Goodfellow *et al.*'14]:

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.*'14]:

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 $p_z \circ G^{-1} \approx p_{data}$

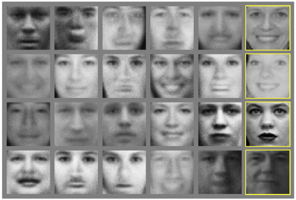
Generative Adversarial Nets [Goodfellow *et al.*'14]:

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.*'14]

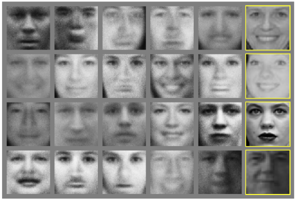
Generative Adversarial Nets [Goodfellow *et al.*'14]:

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.*'14]

NVIDIA'19

GANs & MFGs

Generative Adversarial Nets [Goodfellow *et al.*'14]:

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}

GANs & MFGs

Generative Adversarial Nets [Goodfellow *et al.*'14]:

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)

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

Generative Adversarial Nets [Goodfellow *et al.*'14]:

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: $\inf\limits_{u:[0,T]\times\mathbb{R}^d \to \mathbb{R}} \sup\limits_{m:[0,T]\times\mathbb{R}^d \to \mathbb{R}} \Phi(m, u)$, where

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

GANs & MFGs

Generative Adversarial Nets [Goodfellow *et al.*'14]:

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

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

$\to$ Conceptual connection GANs/MFGs [Cao, Guo, L.'20]

Generative Adversarial Nets [Goodfellow *et al.*'14]:

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: $\inf_{u:[0,T]\times\mathbb{R}^d \to \mathbb{R}} \sup_{m:[0,T]\times\mathbb{R}^d \to \mathbb{R}} \Phi(m, u)$, 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, Guo, L.'20]

Related work: [Domingo-Enrich *et al.*, NeurIPS'20; Onken *et al.*'20]