On Hamilton-Jacobi partial differential equation and architectures of neural networks

Jérôme Darbon

Division of Applied Mathematics, Brown University

ICODE Workshop on numerical solution of HJB equation
January 9, 2020

Joint work with Tingwei Meng and Gabriel Provencher Langlois
Work supported by NSF DMS 1820821

Context and motivation

- Consider the initial value problem

$$\begin{cases} \dfrac{\partial S}{\partial t}(x,t) + H(\nabla_x S(x,t), x, t) \;=\; \varepsilon \triangle S(x,t), & \text{in } \mathbb{R}^n \times (0,+\infty) \\ S(x,0) \;=\; J(x) & \forall x \in \mathbb{R}^n. \end{cases}$$

- Goals: compute the viscosity solution for a given $(x,t) \in \mathbb{R}^n \times [0,+\infty)$
  - evaluate $S(x,t)$ and $\nabla_x S(x,t)$
  - very high dimension
  - fast to allow applications requiring real-time computations
  - low memory and low energy for embedded system
- Pros and cons for computations with grid-based approaches:
  - many advanced and sophisticated numerical schemes (e.g., ENO, WENO, DG) with excellent theoretical properties
  - the number of grid-points is exponential in n

  $\rightarrow$ using numerical approximations is not doable for $n \geq 4$

Overcoming/mitigating the curse of dimensionality

- Several approaches to mitigate/overcome the curse of differentiability
  - Max-plus methods
    [Akian, Dower, McEneaney, Flening, Gaubert, Qu . . .]
  - Tensor decomposition methods
    [Doglov, Horowitz, Kalise, Kunisch, Todorov, . . .]
  - Sparse grids
    [Bokanovski, Garcke, Grieble, Kang, Klompmaker, Kröner, Wilcox]
  - Model order reduction
    [Alla, Kunisch, Falcone, Wolkwein, . . .]
  - Optimization techniques via representation formulas
    [D., Dower, Osher, Yegerov, . . .]
  - . . .
- More recently, there is a significant trend in using Machine Learning and Neural Network techniques for solving PDEs
  $\rightarrow$ A key idea is to leverage universal approximation theorems

Neural Network: a computational point of view

- Pros and cons of Neural Networks for evaluating solutions
  - It seems to be hard to find Neural Networks that are interpretable, generalizable which yield reproducible results
  - Huge computational advantage
    - dedicated hardware for NN is now available: e.g., Xilinx AI (FPGA + silicon design), Intel AI (FPGA + new CPU assembly instructions), and many other (startup) companies
    - high throughput / low latency (more precise meaning of "fast")
    - low energy requirement (e.g., a few Watts)

    $\rightarrow$ suitable for embedded computing and data centers
- Can we leverage these computational resources for high-dimensional H-J PDEs?
- How can we mathematically certify that Neural Networks (NNs) actually computes a viscosity solution of a H-J PDE?

$\Rightarrow$ Establish new connections between NN architectures and representation formulas of H-J PDE solutions

$\rightarrow$ the physics of some H-J PDEs can be encoded by NN architecture

$\rightarrow$ the parameters of the NN define Hamiltonians and initial data

$\rightarrow$ no approximation: exact evaluation of $S(x,t)$ and $\nabla_x S(x,t)$

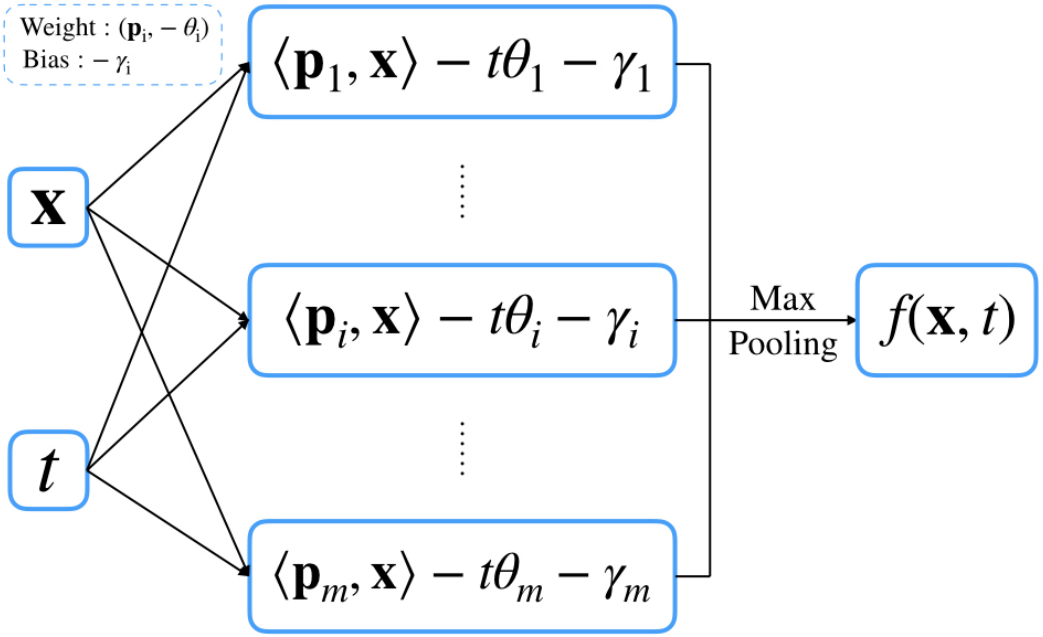
$\rightarrow$ suggests an interpretation of some NN architectures in terms of H-J PDEs

$\rightarrow$ does not rely on NN universal approximation theorems

Outline

1. Shallow NN architectures and representation of solution of H-J PDEs
    1. A class of first-order H-J
    2. Associated conservation law (1D)
    3. A class of second order H-J
2. Numerical experiments for solving some inverse problems involving H-J using data and learning/optimization algorithms
3. Suggestions of other NN architectures for other H-J PDEs
4. Some conclusions

A first shallow network architecture



- Architecture: fully connected layer followed by the activation function "max-pooling"
- This network defines a function $f : \mathbb{R}^n \times [0, +\infty) \to \mathbb{R}$

$$f(\boldsymbol{x}, t; \{\boldsymbol{p}_i, \theta_i, \gamma_i\}_{i=1}^m) = \max_{i \in \{1,\dots,m\}} \{\langle \boldsymbol{p}_i, \boldsymbol{x} \rangle - t\theta_i - \gamma_i\}.$$

Goal: Find conditions on the parameters such that f satisfies a PDE, and find the PDE

Assumptions on the parameters

- Recall: the network $f(\boldsymbol{x}, t; \{\boldsymbol{p}_i, \theta_i, \gamma_i\}_{i=1}^m) = \max_{i\in\{1,\dots,m\}}\{\langle \boldsymbol{p}_i, \boldsymbol{x}\rangle - t\theta_i - \gamma_i\}$
- We adopt the following assumptions on the parameters:

  (A1) The parameters $\{\boldsymbol{p}_i\}_{i=1}^m$ are pairwise distinct, i.e., $\boldsymbol{p}_i \neq \boldsymbol{p}_j$ if $i \neq j$.
  (A2) There exists a convex function $g\colon\ \mathbb{R}^n \to \mathbb{R}$ such that $g(\boldsymbol{p}_i) = \gamma_i$.
  (A3) For any $j \in \{1, \dots, m\}$ and any $(\alpha_1, \dots, \alpha_m) \in \mathbb{R}^m$ that satisfy

$$\begin{cases} (\alpha_1, \dots, \alpha_m) \in \Delta_m \text{ with } \alpha_j = 0, \\ \sum_{i\neq j} \alpha_i \boldsymbol{p}_i = \boldsymbol{p}_j, \\ \sum_{i\neq j} \alpha_i \gamma_i = \gamma_j, \end{cases}$$

  there holds $\sum_{i\neq j} \alpha_i \theta_i > \theta_j$.

  where Δ_m denotes the unit simplex of dimension m

- ($A1$) and ($A3$) are NOT strong assumptions.
  - ($A1$) simplifies the mathematical analysis - ($A3$) simply states the each "neuron" should contribute to the definition of f.
  - If ($A3$) is not satisfied, then it means that some neurons can be removed and the NN still defines the same function f

Define initial data and Hamiltonians from parameters

- Recall: the network f

$$f(\boldsymbol{x}, t; \{\boldsymbol{p}_i, \theta_i, \gamma_i\}_{i=1}^m) = \max_{i \in \{1,\dots,m\}} \{\langle \boldsymbol{p}_i, \boldsymbol{x} \rangle - t\theta_i - \gamma_i\} \tag{1}$$

- Define the initial data J using the NN parameters $\{\boldsymbol{p}_i, \gamma_i\}_{i=1}^m$

$$f(\boldsymbol{x}, 0) = J(\boldsymbol{x}) := \max_{i \in \{1,\dots,m\}} \{\langle \boldsymbol{p}_i, \boldsymbol{x} \rangle - \gamma_i\} \tag{2}$$

  Then, $J : \mathbb{R}^n \to \mathbb{R}$ is convex, and its Legendre transform J^* reads

$$J^*(\boldsymbol{p}) = \begin{cases} \min_{\substack{(\alpha_1,\dots,\alpha_m) \in \Delta_m \\ \sum_{i=1}^m \alpha_i \boldsymbol{p}_i = \boldsymbol{p}}} \left\{\sum_{i=1}^m \alpha_i \gamma_i\right\}, & \text{if } \boldsymbol{p} \in \operatorname{conv}\left(\{\boldsymbol{p}_i\}_{i=1}^m\right), \\ +\infty, & \text{otherwise.} \end{cases}$$

  Denote by $\mathcal{A}(\boldsymbol{p})$ the set of minimizers in the above optimization problem.

- Define the Hamiltonian H: $\mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ by

$$H(\boldsymbol{p}) := \begin{cases} \inf_{\alpha \in \mathcal{A}(\boldsymbol{p})} \left\{\sum_{i=1}^m \alpha_i \theta_i\right\}, & \text{if } \boldsymbol{p} \in \operatorname{dom} J^*, \\ +\infty, & \text{otherwise.} \end{cases} \tag{3}$$

NN computes viscosity solutions

Theorem

Assume (A1)-(A3) hold. Let f be the neural network defined by Eq. (1) with parameters $\{(\boldsymbol{p}_i, \theta_i, \gamma_i)\}_{i=1}^m$. Let J and H be the functions defined in Eqs. (2) and (3), respectively, and let $\tilde{H}\colon\ \mathbb{R}^n \to \mathbb{R}$ be a continuous function. Then the following two statements hold.

(i) *The neural network f is the unique uniformly continuous viscosity solution to the Hamilton–Jacobi equation*

$$\begin{cases} \frac{\partial f}{\partial t}(\boldsymbol{x}, t) + H(\nabla_{\boldsymbol{x}} f(\boldsymbol{x}, t)) = 0, & \boldsymbol{x} \in \mathbb{R}^n, t > 0, \\ f(\boldsymbol{x}, 0) = J(\boldsymbol{x}), & \boldsymbol{x} \in \mathbb{R}^n. \end{cases} \tag{4}$$

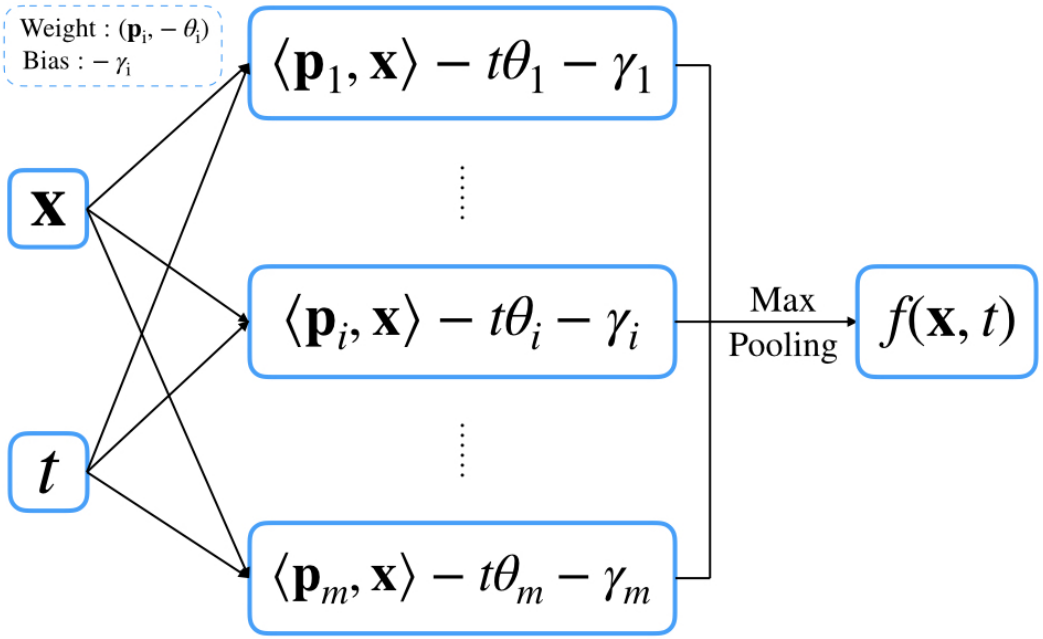
Moreover, f is jointly convex in ($\boldsymbol{x}$,t).

(ii) *The neural network f is the unique uniformly continuous viscosity solution to the Hamilton–Jacobi equation*

$$\begin{cases} \frac{\partial f}{\partial t}(\boldsymbol{x}, t) + \tilde{H}(\nabla_{\boldsymbol{x}} f(\boldsymbol{x}, t)) = 0, & \boldsymbol{x} \in \mathbb{R}^n, t > 0, \\ f(\boldsymbol{x}, 0) = J(\boldsymbol{x}), & \boldsymbol{x} \in \mathbb{R}^n. \end{cases} \tag{5}$$

if and only if $\tilde{H}(\boldsymbol{p}_i) = H(\boldsymbol{p}_i)$ for every $i = 1, \ldots, m$ and $\tilde{H}(\boldsymbol{p}) \geqslant H(\boldsymbol{p})$ for every $\boldsymbol{p} \in \operatorname{dom} J^*$.

NN computes viscosity solutions



- The network computes viscosity solution for H and J given by parameters
- Hamiltonians are not unique. However, among all possible Hamiltonians, H is the smallest one.
- In addition, $\nabla_x S(x, t)$ (when it exists) is given by the element that realizes the maximum is the "max-pooling"

NN computes viscosity solutions: An example

- Consider $J_{orig}(\boldsymbol{x}) = \|\boldsymbol{x}\|_\infty$ and $H_{orig}(\boldsymbol{p}) = -\frac{\|\boldsymbol{p}\|_2^2}{2}$ for every $\boldsymbol{x}, \boldsymbol{p} \in \mathbb{R}^n$.
- Denote by e_i the i^{th} standard unit vector in $\mathbb{R}^n$. Let $m = 2n$, $\{\boldsymbol{p}_i\}_{i=1}^m = \{\pm e_i\}_{i=1}^n$, $\theta_i = -\frac{n}{2}$, and $\gamma_i = 0$ for every $i \in \{1, \dots, m\}$.
- The viscosity solution S is given by

$$S(\boldsymbol{x}, t) = \|\boldsymbol{x}\|_\infty + \frac{nt}{2} = \max_{i \in \{1,\dots,m\}} \{\langle \boldsymbol{p}_i, \boldsymbol{x} \rangle - t\theta_i - \gamma_i\}, \text{ for every } \boldsymbol{x} \in \mathbb{R}^n \text{ and } t \geqslant 0.$$

- Hence, S can be represented using the proposed neural network with parameters $\{(\boldsymbol{p}_i, -\frac{n}{2}, 0)\}_{i=1}^m$.
- The parameters $\{(\boldsymbol{p}_i, -\frac{n}{2}, 0)\}_{i=1}^m$ define the following initial data and smallest Hamiltonian

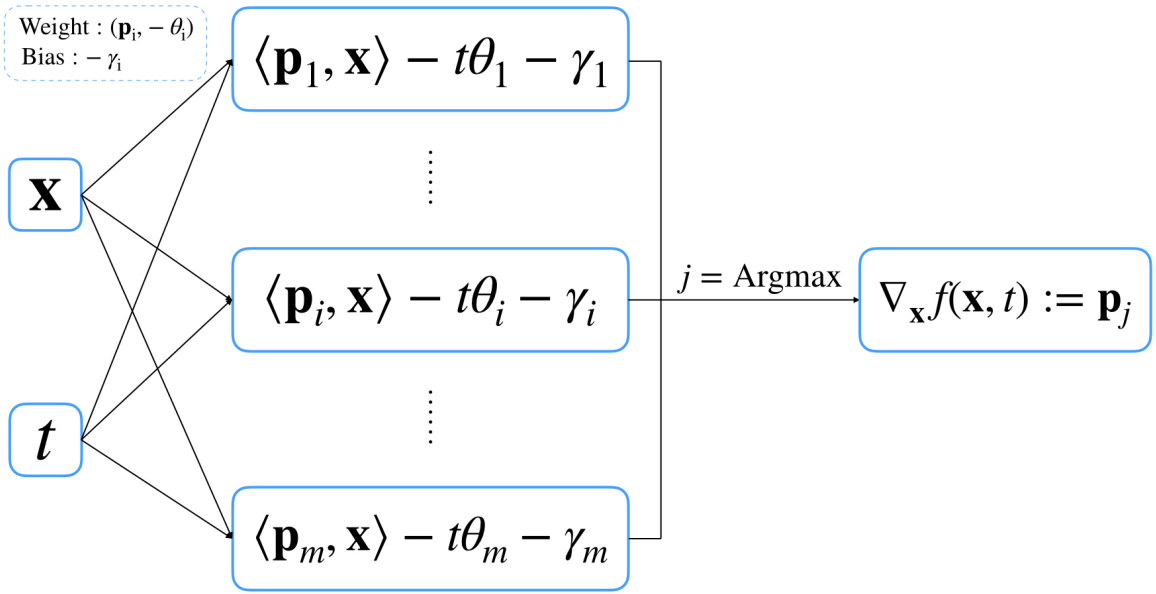
$$J(\boldsymbol{x}) = \|\boldsymbol{x}\|_\infty, \text{ for every } \boldsymbol{x} \in \mathbb{R}^n;$$

$$H(\boldsymbol{p}) = \begin{cases} -\frac{n}{2}, & \boldsymbol{p} \in B_n; \\ +\infty, & \text{otherwise}, \end{cases}$$

where B_n is the unit ball with respect to the l^1 norm in $\mathbb{R}^n$, i.e.,
$B_n = \operatorname{conv}\{\pm e_i : \ i \in \{1, \dots, n\}\}$

- By Thm. 1, S is a viscosity solution to the HJ PDE (5) if and only if $\tilde{H}(\boldsymbol{p}_i) = -\frac{n}{2}$ for every $i \in \{1, \dots, m\}$ and $\tilde{H}(\boldsymbol{p}) \geq -\frac{n}{2}$ for every $\boldsymbol{p} \in B_n \setminus \{\boldsymbol{p}_i\}_{i=1}^m$.
- Therefore, the Hamiltonian H_{orig} is one candidate satisfying these constraints.

Architecture for the gradient map



- This NN architecture computes the spatial gradient of the solution (i.e., the momentum)
- Consider $u : \mathbb{R}^n \times [0, +\infty) \to \mathbb{R}^n$ defined by

$$u(x,t) = \nabla_x f(x,t) = p_j, \text{ where } j \in \underset{i \in \{1,\ldots,m\}}{\arg\max} \{\langle p_i, x \rangle - t\theta_i - \gamma_i\}. \quad (6)$$

NN architecture and 1D conservation laws

Theorem

Consider the one-dimensional case, i.e., $n = 1$. Suppose assumptions (A1)-(A3) hold. Let $u := \nabla_x f$ be the function from $\mathbb{R} \times [0, +\infty)$ to $\mathbb{R}$ defined in Eq. (6). Let J and H be the functions defined in Eqs. (2) and (3), respectively, and let $\tilde{H} \colon \mathbb{R} \to \mathbb{R}$ be a locally Lipschitz continuous function. Then the following two statements hold.

(i) *The neural network u is the entropy solution to the conservation law*

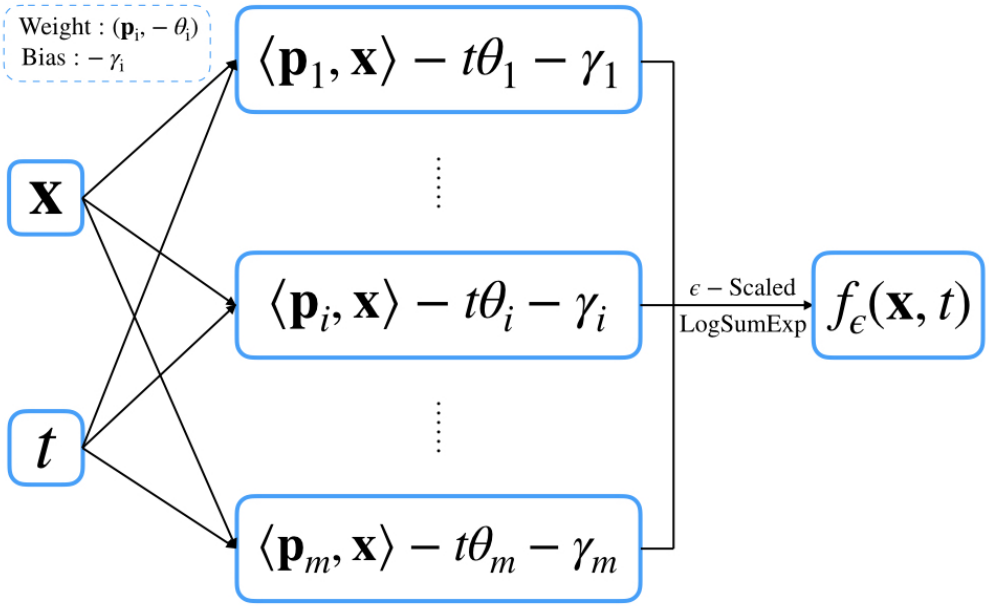
$$\begin{cases} \frac{\partial u}{\partial t}(x,t) + \nabla_x H(u(x,t)) = 0, & x \in \mathbb{R}, t > 0, \\ u(x,0) = \nabla J(x), & x \in \mathbb{R}. \end{cases} \tag{7}$$

(ii) *The neural network u is the entropy solution to the conservation law*

$$\begin{cases} \frac{\partial u}{\partial t}(x,t) + \nabla_x \tilde{H}(u(x,t)) = 0, & x \in \mathbb{R}, t > 0, \\ u(x,0) = \nabla J(x), & x \in \mathbb{R}, \end{cases} \tag{8}$$

if and only if there exists a constant $C \in \mathbb{R}$ such that $\tilde{H}(p_i) = H(p_i) + C$ for every $i \in \{1, \dots, m\}$ and $\tilde{H}(p) \geqslant H(p) + C$ for any $p \in \operatorname{conv} \{p_i\}_{i=1}^m$.

Another shallow architecture

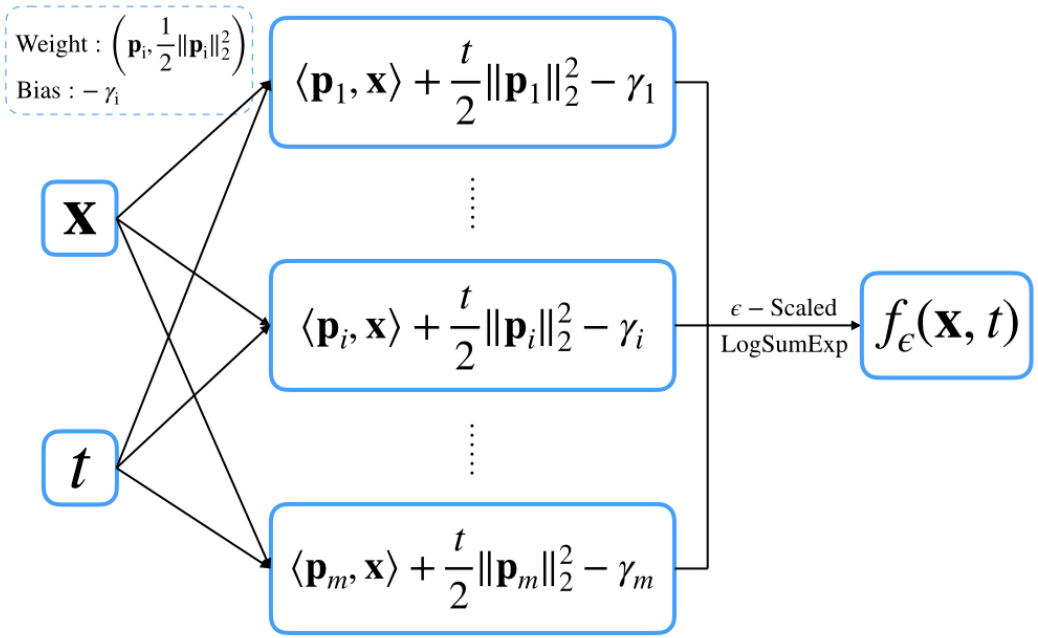


- Replace “max-pooling” by “smooth max pooling”.
- This network defines a function $f : \mathbb{R}^n \times [0, +\infty) \to \mathbb{R}$

$$f_\epsilon(\boldsymbol{x}, t) := \epsilon \log \left(\sum_{i=1}^{m} e^{(\langle \boldsymbol{p}_i, \boldsymbol{x} \rangle - t\theta_i - \gamma_i)/\epsilon} \right).$$

Specialize this architecture

Specialize the parameters: $\theta_i = -\frac{1}{2}\|p_i\|_2^2 \quad$ for $i = 1, \ldots, m$



- This network defines the function $f : \mathbb{R}^n \times [0, +\infty) \to \mathbb{R}$

$$f_\epsilon(\boldsymbol{x}, t) := \epsilon \log \left(\sum_{i=1}^{m} e^{\left(\langle \boldsymbol{p}_i, \boldsymbol{x} \rangle + \frac{t}{2}\|\boldsymbol{p}_i\|_2^2 - \gamma_i \right)/\epsilon} \right). \tag{9}$$

NN computes viscosity solutions of some second order H-J PDEs

Theorem

Let f_ϵ defined by (9) with parameters $\{\boldsymbol{p}_i, \theta_i, \gamma_i\}_{i=1}^m$ and let $\theta_i = -\frac{1}{2}\|\boldsymbol{p}_i\|_2^2$ for $i \in \{1, \dots, m\}$. For every $\epsilon > 0$, the neural network $f_\epsilon \equiv \epsilon \log(w_\epsilon)$ is the unique smooth solution to the second-order Hamilton–Jacobi equation

$$\begin{cases} \frac{\partial f_\epsilon(\boldsymbol{x},t)}{\partial t} - \frac{1}{2}\|\nabla_{\boldsymbol{x}} f_\epsilon(\boldsymbol{x},t)\|_2^2 = \frac{\epsilon}{2}\triangle_{\boldsymbol{x}} f_\epsilon(\boldsymbol{x},t) & \text{in } \mathbb{R}^n \times (0,+\infty), \\ f_\epsilon(\boldsymbol{x},0) = \epsilon \log\left(\sum_{i=1}^m e^{(\langle \boldsymbol{p}_i, \boldsymbol{x}\rangle - \gamma_i)/\epsilon}\right) & \forall \boldsymbol{x} \in \mathbb{R}^n. \end{cases} \tag{10}$$

Moreover, f_ϵ is jointly convex in $(\boldsymbol{x}, t)$ and the following holds

$$\lim_{\substack{\epsilon \to 0 \\ \epsilon > 0}} f_\epsilon(\boldsymbol{x},t) = \max_{i \in \{1,\dots,m\}} \{\langle \boldsymbol{p}_i, \boldsymbol{x}\rangle + \frac{t}{2}\|\boldsymbol{p}_i\|_2^2 - \gamma_i\} \tag{11}$$

for every $\boldsymbol{x} \in \mathbb{R}^n$ and $t \geqslant 0$. Finally, if assumptions (A1)-(A3) hold, then the right hand side of (11) solves the first-order Hamilton–Jacobi equation (5) with $\tilde{H} := -\frac{1}{2}\|\cdot\|_2^2$.

Partial summary

- We have exhibited classes of network architecture that represent viscosity solution of some H-J PDEs
- Initial data and Hamiltonians are induced by the parameters of the network
  - Initial datum is completely determined by the parameters of the NN
  - Hamiltonians are not unique, but there exists a smallest Hamiltonian that is completely determined by the parameters of the NN.

Outline

1. Shallow NN architectures and representation of solution of H-J PDEs
    1. A class of first-order H-J
    2. Associated conservation law (1D)
    3. A class of second order H-J
2. Numerical experiments for solving some inverse problems involving H-J using data and learning/optimization algorithms
3. Suggestions of other NN architectures for other PDEs
4. Some conclusions

Trying to solve some simple inverse problems

- We are given data samples $\{(\boldsymbol{x}_j, t_j, S(\boldsymbol{x}_j, t_j))\}_{j=1}^N$, where $\{(\boldsymbol{x}_j, t_j)\}_{j=1}^N \subset \mathbb{R}^n \times [0, +\infty)$, we train the neural network f with structure linear + max-pooling using the mean square loss function defined by

$$I(\{(\boldsymbol{p}_i, \theta_i, \gamma_i)\}_{i=1}^m) = \frac{1}{N} \sum_{j=1}^{N} |f(\boldsymbol{x}_j, t_j; \{(\boldsymbol{p}_i, \theta_i, \gamma_i)\}_{i=1}^m) - S(\boldsymbol{x}_j, t_j)|^2.$$

- The training problem is formulated as

$$\underset{\{(\boldsymbol{p}_i, \theta_i, \gamma_i)\}_{i=1}^m \subset \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}}{\arg\min} I(\{(\boldsymbol{p}_i, \theta_i, \gamma_i)\}_{i=1}^m).$$

- After training, we approximate the initial condition in the HJ equation, denoted by $\tilde{J}$, by evaluating the trained neural network at $t = 0$, i.e., we approximate the initial condition by

$$\tilde{J} := f(\cdot, 0).$$

- We obtain partial information of the Hamiltonian H using the parameters in the trained neural network via the following procedure: (i) detect the effective neurons of the network, which we define to be the affine functions $\{\langle \boldsymbol{p}_i, \boldsymbol{x} \rangle - t\theta_i - \gamma_i\}$ that contribute to the pointwise maximum in the neural network f. Denote by L the set of indices that correspond to the parameters of the effective neurons, i.e.,

$$L := \bigcup_{\boldsymbol{x} \in \mathbb{R}^n,\, t \geq 0} \underset{i \in \{1, \ldots, m\}}{\arg\max} \{\langle \boldsymbol{p}_i, \boldsymbol{x} \rangle - t\theta_i - \gamma_i\},$$

  and we finally use each effective parameter $(\boldsymbol{p}_l, \theta_l)$ for $l \in L$ to approximate the point $(\boldsymbol{p}_l, H(\boldsymbol{p}_l))$ on the graph of the Hamiltonian.

Randomly piecewise affine Hamiltonian and Initial data

- We randomly select m parameters $\boldsymbol{p}_i^{true}$ in $[-1,1)^n$ for $i \in \{1,\dots,m\}$, and define θ_i^{true} and γ_i^{true} as follows

Case 1. $\theta_i^{true} = -\|\boldsymbol{p}_i^{true}\|_2$ and $\gamma_i^{true} = 0$, for $i \in \{1,\dots,m\}$.
Case 2. $\theta_i^{true} = -\|\boldsymbol{p}_i^{true}\|_2$ and $\gamma_i^{true} = \frac{1}{2}\|\boldsymbol{p}_i^{true}\|_2^2$, for $i \in \{1,\dots,m\}$.
Case 3. $\theta_i^{true} = -\frac{1}{2}\|\boldsymbol{p}_i^{true}\|_2^2$ and $\gamma_i^{true} = 0$, for $i \in \{1,\dots,m\}$.
Case 4. $\theta_i^{true} = -\frac{1}{2}\|\boldsymbol{p}_i^{true}\|_2^2$ and $\gamma_i^{true} = \frac{1}{2}\|\boldsymbol{p}_i^{true}\|_2^2$, for $i \in \{1,\dots,m\}$.

- Define the function S as

$$S(\boldsymbol{x},t) := \max_{i\in\{1,\dots,m\}} \{\langle \boldsymbol{p}_i^{true}, \boldsymbol{x}\rangle - t\theta_i^{true} - \gamma_i^{true}\}.$$

  By our Thm. this function S is a viscosity solution to the HJ equations whose Hamiltonian and initial function are the piecewise affine functions.

- We train the neural network f with training data $\{(\boldsymbol{x}_j, t_j, S(\boldsymbol{x}_j,t_j))\}_{j=1}^N$, where the points $\{(\boldsymbol{x}_j,t_j)\}_{j=1}^N$ are randomly sampled in $\mathbb{R}^n \times [0,+\infty)$ with respect to the standard normal distribution for each $j \in \{1,\dots,N\}$
  - The number of training data points is $N = 20{,}000$. We run 60,000 descent steps using the Adam optimizer to train the neural network f.
  - The parameters for the Adam algorithm are chosen to be $\beta_1 = 0.5$, $\beta_2 = 0.9$, the learning rate is 10^{-4} and the batch size is 500.

Measure of performance

- We compute the relative mean square errors of the sorted parameters in the trained neural network, denoted by $\{(\boldsymbol{p}_i, \theta_i, \gamma_i)\}_{i=1}^m$, and the sorted underlying true parameters $\{(\boldsymbol{p}_i^{true}, \theta_i^{true}, \gamma_i^{true})\}_{i=1}^m$. To be specific, the errors are computed as follows

$$\text{relative mean square error of } \{\boldsymbol{p}_i\} = \frac{\sum_{i=1}^m \|\boldsymbol{p}_i - \boldsymbol{p}_i^{true}\|_2^2}{\sum_{i=1}^m \|\boldsymbol{p}_i^{true}\|_2^2},$$

$$\text{relative mean square error of } \{\theta_i\} = \frac{\sum_{i=1}^m |\theta_i - \theta_i^{true}|^2}{\sum_{i=1}^m |\theta_i^{true}|^2},$$

$$\text{relative mean square error of } \{\gamma_i\} = \frac{\sum_{i=1}^m |\gamma_i - \gamma_i^{true}|^2}{\sum_{i=1}^m |\gamma_i^{true}|^2}.$$

- We test Cases 1–4 on the neural networks with 2 and 4 neurons, and repeat the experiments 100 times. We then compute the relative mean square errors in each experiments and take the average.
- Note that all relative errors should be 0 provided the optimizer is able to find a global minimizer.

Numerical results: 2 neurons

# Case		Case 1	Case 2	Case 3	Case 4
Averaged Relative Errors of $\{p_i\}$	2D	4.10E-03	2.10E-03	3.84E-03	2.82E-03
	4D	1.41E-09	1.20E-09	1.38E-09	1.29E-09
	8D	1.14E-09	1.03E-09	1.09E-09	1.20E-09
	16D	1.14E-09	6.68E-03	1.23E-09	7.74E-03
	32D	1.49E-09	3.73E-01	1.46E-03	4.00E-01
Averaged Relative Errors of $\{\theta_i\}$	2D	4.82E-02	7.31E-02	1.17E-01	1.79E-01
	4D	3.47E-10	2.82E-10	1.15E-09	1.15E-09
	8D	1.47E-10	1.08E-10	2.10E-10	2.25E-10
	16D	5.44E-11	1.69E-03	4.75E-11	4.12E-03
	32D	3.61E-11	3.27E-01	6.42E-03	2.39E-01
Averaged Relative Errors of $\{\gamma_i\}$	2D	1.35E-02	1.01E-01	1.33E-02	9.24E-02
	4D	3.71E-10	1.24E-09	3.67E-10	1.10E-09
	8D	2.91E-10	1.74E-10	2.82E-10	2.01E-10
	16D	2.80E-10	2.08E-04	3.10E-10	3.20E-04
	32D	3.56E-10	1.88E-02	1.56E-01	3.62E-02

Numerical results: 4 neurons

# Case		Case 1	Case 2	Case 3	Case 4
Averaged Relative Errors of $\{p_i\}$	2D	3.12E-01	2.21E-01	2.85E-01	2.14E-01
	4D	7.82E-02	6.12E-02	7.92E-02	4.30E-02
	8D	2.62E-02	4.31E-03	4.02E-02	7.82E-03
	16D	2.88E-02	3.64E-02	4.35E-02	1.73E-02
	32D	1.42E-02	3.72E-01	1.42E-01	5.04E-01
Averaged Relative Errors of $\{\theta_i\}$	2D	2.59E-01	3.68E-01	4.82E-01	1.34E+00
	4D	6.07E-02	8.37E-02	9.47E-02	1.23E-01
	8D	1.04E-02	8.48E-03	1.41E-02	1.31E-02
	16D	2.66E-03	2.53E-02	7.80E-03	1.90E-02
	32D	8.09E-04	4.41E-01	1.81E-02	3.66E-01
Averaged Relative Errors of $\{\gamma_i\}$	2D	1.01E-02	3.19E-01	1.51E-02	2.65E-01
	4D	6.72E-03	1.79E-02	1.03E-02	1.30E-02
	8D	3.22E-03	2.34E-03	3.93E-03	2.65E-03
	16D	9.48E-03	3.70E-03	1.92E-02	1.94E-03
	32D	1.33E-02	5.35E-02	4.73E-01	1.17E-01

Partial summary

- Off the shelf standard stochastic optimization optimizer used in machine learning may not yield “good minimizer” for these non-convex optimization problems
- Other numerical experiments have been conducted and yield the same mixed results.

Outline

1. Shallow NN architectures and representation of solution of H-J PDEs
    1. A class of first-order H-J
    2. Associated conservation law (1D)
    3. A class of second order H-J
2. Numerical experiments for solving some inverse problems involving H-J using data and learning/optimization algorithms
3. Suggestions of other NN architectures for other PDEs
4. Some conclusions

Possible extensions

Draw architectures based on Lax-Oleinik formulas on balckboard

Conclusion + ongoing work

- Some NN architectures represent viscosity solutions of certain H-J PDEs in very high-dimensions
- Hamiltonian and initial data are given by the parameters. Specifically,
  - Initial data completely determined by the NN parameters
  - Hamiltonians may not be unique, however, the smallest Hamiltonian is well defined and determined by the parameters.
- Preliminary results on solving some simple inverse problems involving these NN architecture do not yield satisfactory results.
  - Standard stochastic gradient algorithm may not be able to find a "good" minimizer

  $\rightarrow$ Use properties of HJ solutions to design taylored numerical optimization algorithms.
- We have obtained similar theorems for Lax-Oleinik-based formulas NN architecture

  $\rightarrow$ it paves the way to combine the NN architectures with Max-Plus methods
- Extension to optimal control of linear ODEs using a generalized Hopf-based formula
- FPGA implementation is underway