"Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations"

島内研究室 2025年1月14日 輪読会

発表者: 多田 瑛貴
(公立はこだて未来大学 複雑系知能学科 複雑系コース)

書誌情報

Physics Informed Deep Learning (Part I):
Data-driven Solutions of Nonlinear Partial Differential Equations

著者: Maziar Raissi, Paris Perdikaris, George Em Karniadakis
arXiv:1711.10561 (https://arxiv.org/abs/1711.10561)

2017年発表の論文、Part IとPart IIの2部構成

- Part I: data-driven solution (深層学習による非線形偏微分方程式の数値解法)
- Part II: data-driven discovery (パラメータ不定の方程式に対するPINNsの適用)

著者自身によるサンプル実装が存在

TensorFlow v1によるもので、メンテナンスは停止している

ただし、単一で動く実装のコレクションのため、スクラッチ実装の参考として十分参考になる

https://github.com/maziarraissi/PINNs

各機械学習フレームワーク向けの抽象化された実装も存在

実際に利用する場合はこちらが強く推奨されている

PyTorch: https://github.com/rezaakb/pinns-torch

JAX: https://github.com/rezaakb/pinns-jax

TensorFlow v2: https://github.com/rezaakb/pinns-tf2

本研究の概要

"physics-informed neural networks" (PINNs) の提案

- 非線形偏微分方程式を解くデータ駆動のアプローチ
- ある物理現象の支配方程式を考慮した深層学習を導入
- 既知の物理法則に対して、少量の教師データから有用な近似解を提供
- パラメータ不定の偏微分方程式に対しても適用可能 (Part IIの範囲)

背景

- データ駆動での数値解法のアプローチとして、深層学習を用いたい
  物理法則を深層学習を用いて解く意義は、研究結果を踏まえ述べていく
- 物理法則を深層学習を用いて解くとき
  現実の多くの場面では、教師データを十分な量取得できない
  データが少ない場合、ロバスト性を失い、かつ学習を通したモデルの収束を保証できなくなる
- 既知の支配方程式を活用できないだろうか？

関連手法

- 古典的な数値計算手法
  - ルンゲ＝クッタ法、有限要素法、スペクトル法など
- データ駆動のアプローチ
  - ガウス過程回帰 (Gaussian Process Regression)

補足: いくつかの手法、特にガウス過程回帰についてはキャッチアップが間に合わず
これらに対するPINNsの優位性については省略します、輪読会内で共有できればと思います
(非線形方程式の局所的な線形化による離散化の必要性と精度の悪化/ベイジアン的性質によるモデルの表現能力の制限)

提案手法

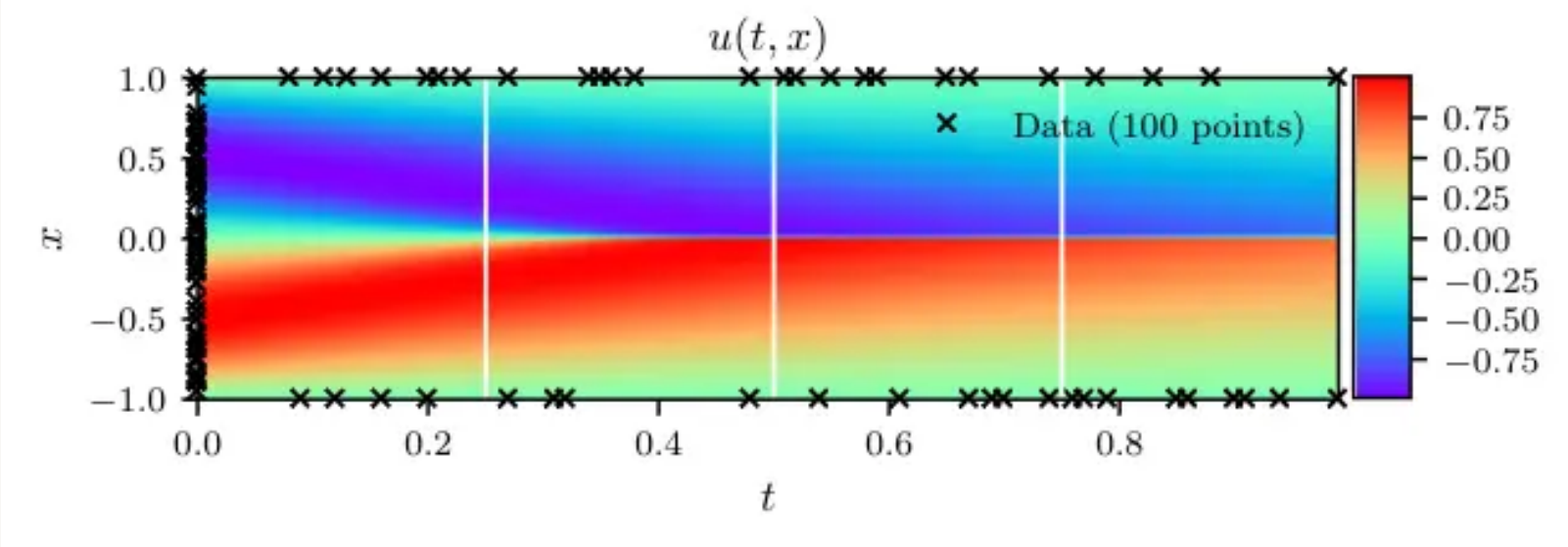
物理法則の支配方程式を損失関数に取り入れたニューラルネットワークを構築

以下の2つの適用方法を示す

- **Continuous Time Models**: 時空間領域上で連続する解を求める
- **Discrete Time Models**: ある時間からある時間への解の時間発展を求める

Continuous Time Models

時空間領域上で連続する解を求める



導入対象: Burgers方程式

一般に、動的粘性率νから

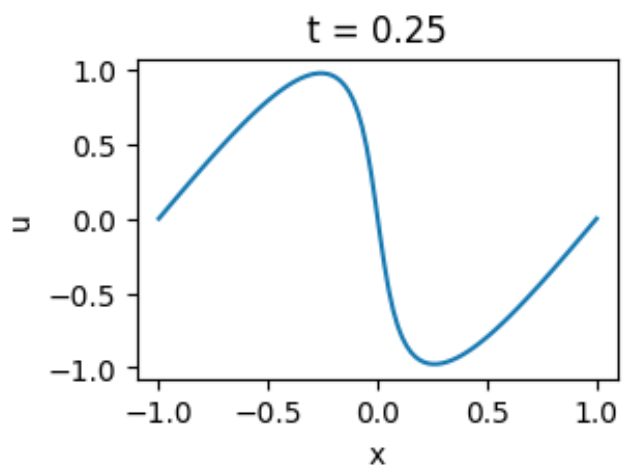
$$u_t + uu_x - \nu u_{xx} = 0$$

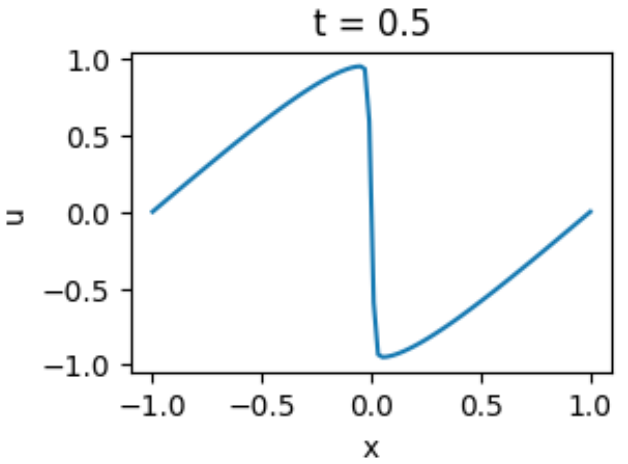
ナビエ-ストークス方程式から、圧力項を無視したもの

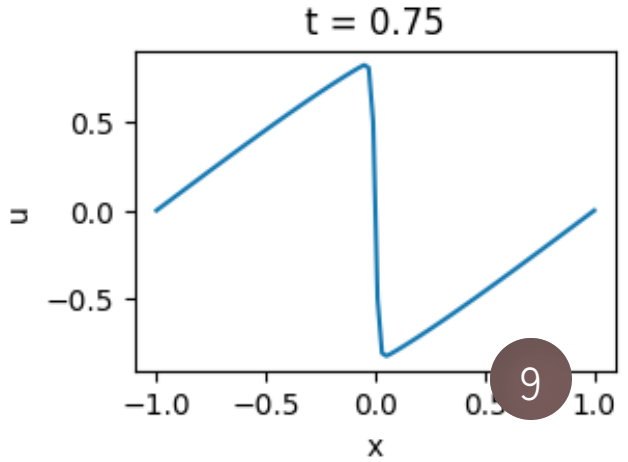
ナビエ-ストークス方程式: 流体の運動を記述する偏微分方程式
(非圧縮性粘性流体の運動方程式)

ここでは、パラメータおよび初期・境界条件を
次のように定め扱うこととする

$$\begin{cases} u_t + uu_x - (0.01/\pi)u_{xx} = 0 \\ u(0,x) = -\sin(\pi x) \\ u(t,-1) = u(t,1) = 0 \end{cases}$$







学習の目標

支配方程式を次のように表現 ($u(t,x)$: 未知関数, λ: パラメータ)

$$u_t + \mathcal{N}[u;\lambda] = 0$$

現実の物理法則の支配方程式の多くは、この形式で表現可能

Burgers方程式においては... $u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0$ より $\mathcal{N}[u;\lambda] = \lambda_1 u u_x - \lambda_2 u_{xx}$

本論文 Part I ではパラメータλの不定性を考慮せず、次のように表現

$$u_t + \mathcal{N}[u] = 0$$

目標: 未知関数 $u(t, x)$ を近似するニューラルネットワークを作る

ただし、$x \in \Omega, t \in [0, T]$ とし、Ωは$\mathbb{R}$の部分集合

```
def u(t, x):
  u = neural_net(tf.concat([t,x],1), weights, biases)
  return u
```

損失関数の概要

- 教師データに対する損失関数 MSE_u
- 支配方程式に対する損失関数 MSE_f

この和である

$$MSE = MSE_u + MSE_f$$

を全体の損失関数として最小化する

教師データの設定

ディレクレ境界条件から

$$\begin{cases} u(0,x) = -\sin(\pi x) \\ u(t,-1) = u(t,1) = 0 \end{cases}$$

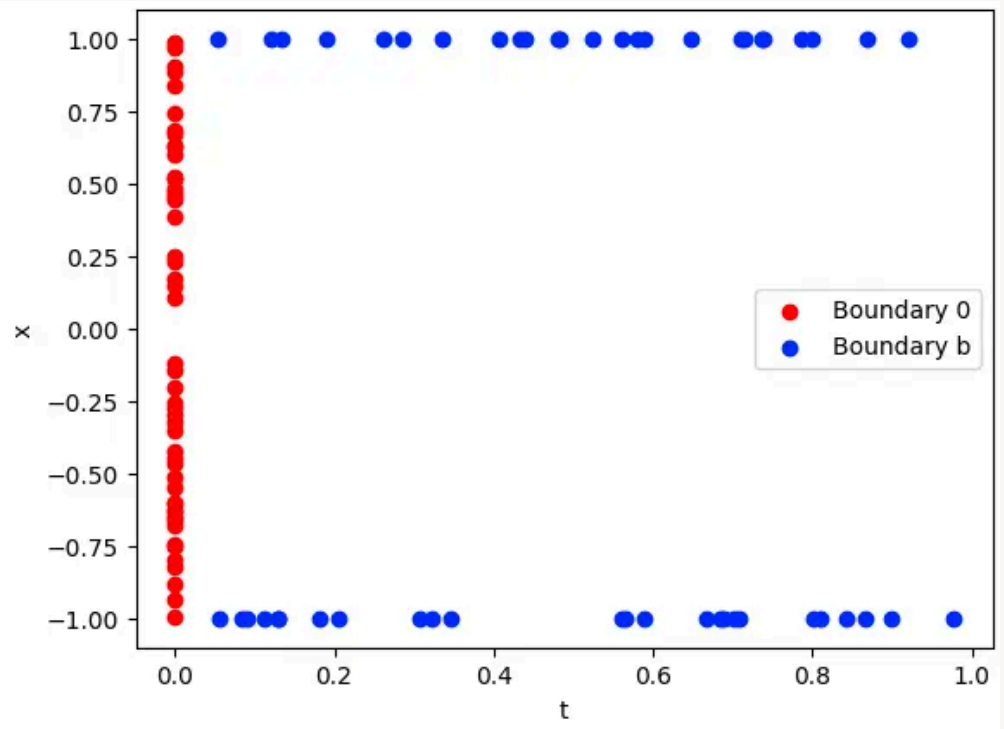
この条件を満たす時空間領域上の点を設け
教師データ(t_u^i, x_u^i, u^i)とする

(t_u^i, x_u^i): 時空間上の点、u^i: その点におけるuの値

損失関数は、この平均二乗誤差MSE_u

N_u: 教師データの数

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2$$



支配方程式の残差計算

支配方程式

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0$$

時空間領域からランダムにおいた選点
(t_f^i, x_f^i) をおき、残差を計算

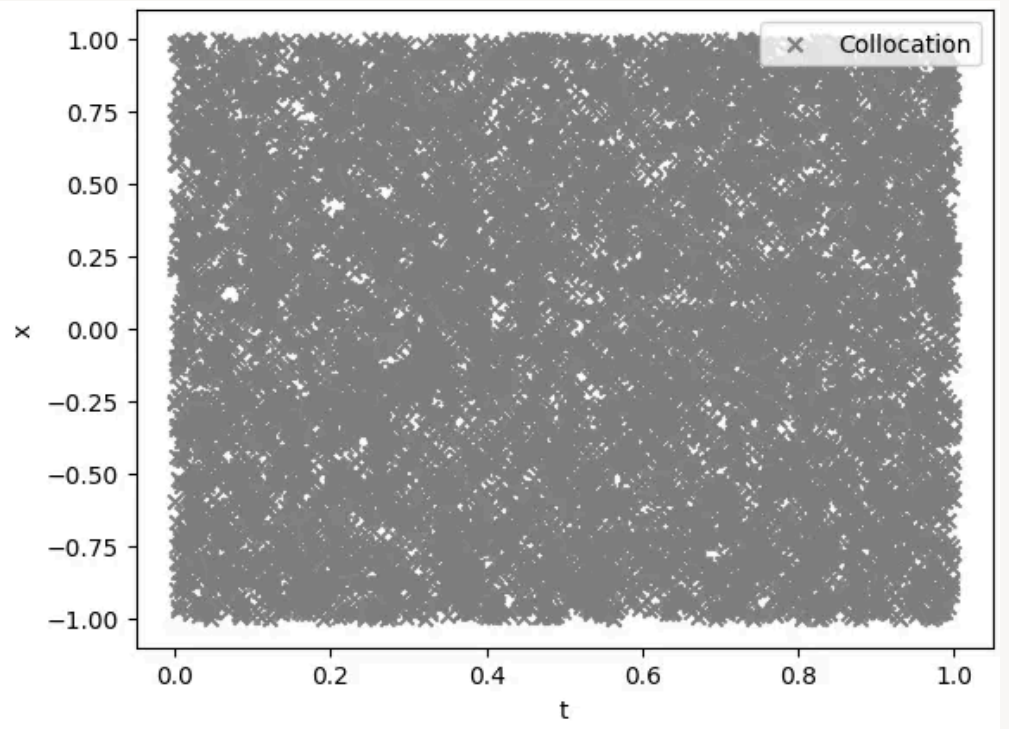
$$f(t_f^i, x_f^i) = u_t + uu_x - (0.01/\pi)u_{xx}$$

fを0に近づける =方程式を満たす よう学習

具体的なuの値は求めない

損失関数は、平均二乗誤差MSE_f

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2$$



参考: MSE_fの計算

$$f(t_f^i, x_f^i) = u_t + uu_x - (0.01/\pi)u_{xx}$$

ニューラルネットワークに対する自動微分から得る

```
def f(t, x):
  u = u(t, x)
  u_t = tf.gradients(u, t)[0]
  u_x = tf.gradients(u, x)[0]
  u_xx = tf.gradients(u_x, x)[0]
  f = u_t + u*u_x - (0.01/tf.pi)*u_xx
  return f
```

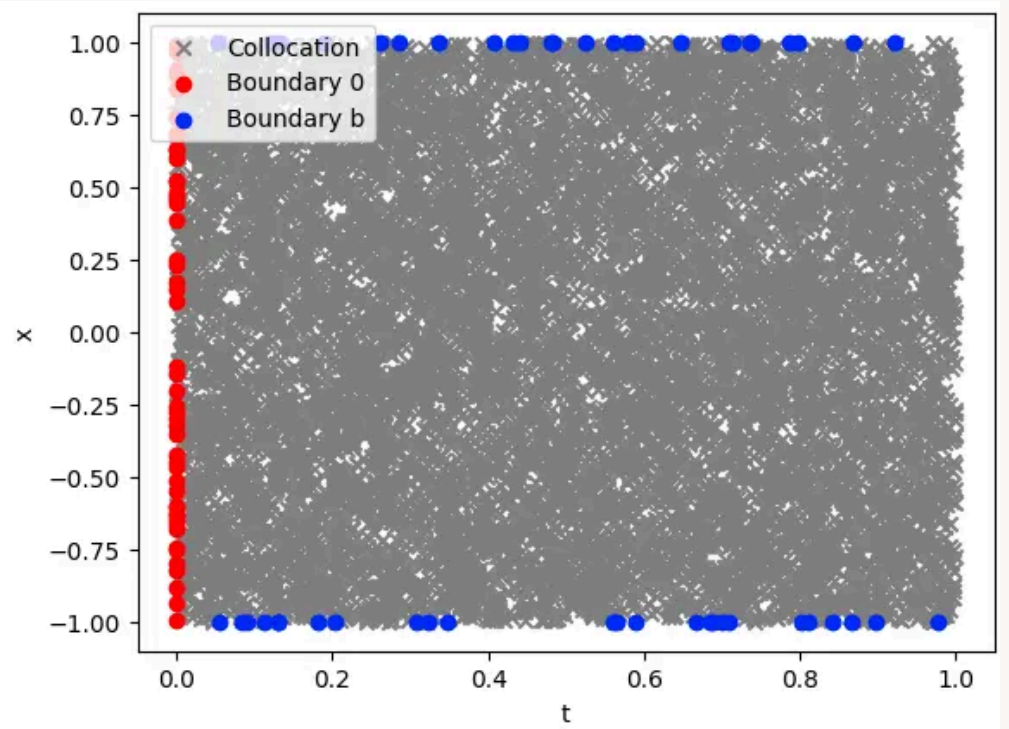
損失関数の設定

全体の損失関数は

$$MSE = MSE_u + MSE_f$$

右図は、$N_u = 100, N_f = 10000$

参考: $N_f = 0$であれば単に教師データのみに基づく
深層学習本来のアプローチと同じとなる



期待される利点

- 教師データが少ない場合でも
  支配方程式との残差を損失関数に取り入れることで、有用な近似解を提供

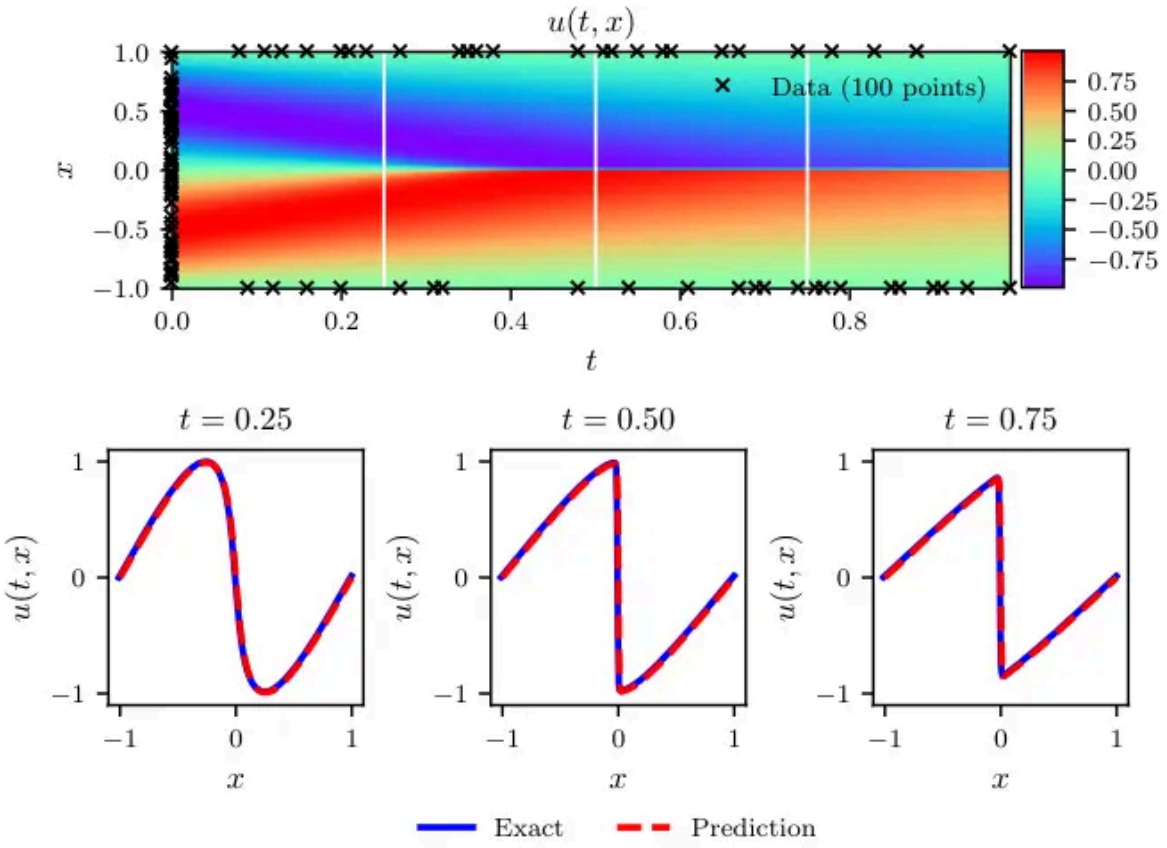
補足: 直感的には、ある時空間領域上の点に対する数値解を直接求めることにより
累積誤差や計算コストを抑えることができる利点があるのではないかと考えられますが
それに関する明確な言及は見られませんでした、ここについても議論したいです

実験では、$N_u = 100, N_f = 10000$での適用を試みる

実験

- $N_u = 100, N_f = 10000$
- 9層(中間7層)のニューラルネットワーク
- 中間層ニューロン数20
- 活性化関数: tanh
- 最適化手法: L-BFGS
- 選点の生成手法: Latin hypercube sampling

支配方程式・境界条件へのあてはまり



相対二乗誤差(RSE): 6.7×10^{-4}

上図: tx 平面上の予測値uの値, 下図: $t = 0.25, 0.50, 0.75$におけるxに対するuの値

N_u・N_fの値と性能の関係

相対二乗誤差による比較、9層(中間7層)、中間層ニューロン数20

N_u \ N_f	2000	4000	6000	7000	8000	10000
20	2.9e-01	4.4e-01	8.9e-01	1.2e+00	9.9e-02	4.2e-02
40	6.5e-02	1.1e-02	5.0e-01	9.6e-03	4.6e-01	7.5e-02
60	3.6e-01	1.2e-02	1.7e-01	5.9e-03	1.9e-03	8.2e-03
80	5.5e-03	1.0e-03	3.2e-03	7.8e-03	4.9e-02	4.5e-03
100	6.6e-02	2.7e-01	7.2e-03	6.8e-04	2.2e-03	6.7e-04
200	1.5e-01	2.3e-03	8.2e-04	8.9e-04	6.1e-04	4.9e-04

選点N_fを与えることによって精度が向上していることがわかる

レイヤ数・中間層ニューロン数と性能の関係

相対二乗誤差による比較、$N_u = 100, N_f = 10000$

Layers \ Neurons	10	20	40
2	7.4e-02	5.3e-02	1.0e-01
4	3.0e-03	9.4e-04	6.4e-04
6	9.6e-03	1.3e-03	6.1e-04
8	2.5e-03	9.6e-04	5.6e-04

概ね、レイヤ・ニューロン数が多いほど精度が高まる

複素関数への適用

Burgers方程式を近似するPINNでは、2入力1出力の実数関数をNNで表現していた
-> 複素関数の場合、実部と虚部を分けた2出力を用いる

導入例: Schrödinger方程式

$$\begin{cases} ih_t + 0.5h_{xx} + |h|^2 h = 0 \\ h(0,x) = 2\mathrm{sech}(x) \\ h(t,-5) = h(t,5) = 0 \\ h_x(t,-5) = h_x(t,5) = 0 \end{cases}$$

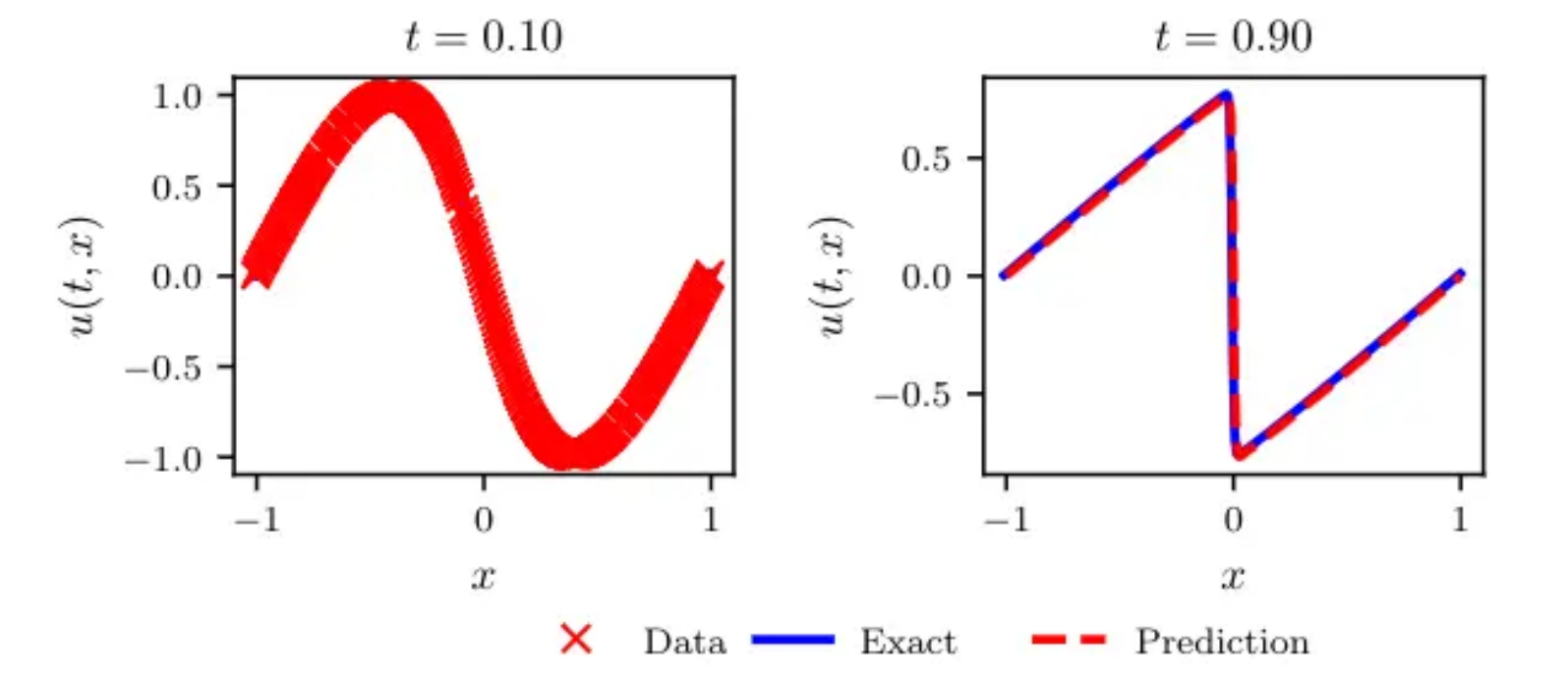
$h = u + iv$とすると、出力は$h(t,x) = [u(t,x), v(t,x)]$となる
この場合のPINNの損失関数は

$$MSE = MSE_0 + MSE_b + MSE_f$$

$$\begin{cases} MSE_0 = \frac{1}{N_0} \sum_{i=1}^{N_0} |h(t_0^i, x_0^i) - h^i|^2 \\ MSE_b = \frac{1}{N_b} \sum_{i=1}^{N_b} (|h^i(t_b^i, -5) - h^i(t_b^i, 5)|^2 + |h_x^i(t_b^i, -5) - h_x^i(t_b^i, 5)|^2) \\ MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |h(t_f^i, x_f^i)|^2 \end{cases}$$

Discrete Time Models

ある時間の解からある時間への時間発展を求める



目的・学習の目標

初期条件のような、特定の時間での値u^nを教師データとして与え
他の時間での解u^{n+1}を求めることを考えたい

例えば、$t = 0.1$での値から$t = 0.9$での解を求める

- 本来は、ルンゲ＝クッタ法などを用いて数値的に解く
- PINNsでは、ルンゲ＝クッタ法に基づき
  その計算過程の値を含めて損失を計算し最小化する

前提: ルンゲ＝クッタ法の一般形

段数qのルンゲ＝クッタ法は、u^nからu^{n+1}の時間刻みをΔtとし、次のように表される

$$u^{n+c_j(x)} = u(t_n + c_j \Delta t, x)$$

$$\begin{cases} u^{n+c_i} = u^n - \Delta t \sum_{j=1}^{s} a_{ij} \mathcal{N}[u^{n+c_j}] \\ u^{n+1} = u^n - \Delta t \sum_{j=1}^{s} b_j \mathcal{N}[u^{n+c_j}] \end{cases}$$

a_{ij}, b_i, c_iはルンゲ＝クッタ法の係数であり、精度が高くなるように選ばれる

古典的なルンゲ＝クッタ法は、$q = 4$ であり、係数は次の通り

$$a = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad b = [1/6 \quad 1/3 \quad 1/3 \quad 1/6], \quad c = [0 \quad 0.5 \quad 0.5 \quad 1]$$

これは、つぎのように変形する

$$
\begin{aligned}
u^n &= u_i^n, \quad i = 1 \dots q \\
u^n &= u_{q+1}^n
\end{aligned}
$$

つまり、$u_1^n, u_2^n, \dots, u_q^n$をルンゲ＝クッタ法の計算過程の値、$u_{q+1}^n$を解の値として
$[u_1^n, u_2^n, \dots, u_q^n, u_{q+1}^n]$が存在
これらをu_i^nおよびu_{q+1}^nを右辺においた式として表すと

$$
\begin{cases}
u_i^n = u^{n+c_i} - \Delta t \sum_{j=1}^s a_{ij} \mathcal{N}[u^{n+c_j}] \\
u_{q+1}^n = u^{n+1} - \Delta t \sum_{j=1}^s b_j \mathcal{N}[u^{n+c_j}]
\end{cases}
$$

この $[u_1^n, \dots, u_{q+1}^n]$ をニューラルネットワークの出力とし、損失を計算する

損失関数の概要

損失関数は残差二乗和 (SSE) を用いる

- ルンゲ＝クッタ法の計算結果に対する誤差関数 SSE_n
- 境界条件への当てはまりに対する損失関数 SSE_b

この和である

$$SSE = SSE_n + SSE_b$$

を全体の損失関数として最小化する

Burgers方程式への適用

ルンゲ＝クッタ法の一般形 $\cdots$ $\begin{cases} u_i^n = u^{n+c_i} - \Delta t \sum_{j=1}^{s} a_{ij} \mathcal{N}[u^{n+c_j}] \\ u_{q+1}^n = u^{n+1} - \Delta t \sum_{j=1}^{s} b_j \mathcal{N}[u^{n+c_j}] \end{cases}$

Burgers方程式は次のように表現される

$$\mathcal{N}[u^{n+c_i}] = u_t^{n+c_i} + u u_x^{n+c_i} - (0.01/\pi) u_{xx}^{n+c_i}$$

ルンゲ＝クッタ法の計算結果に対する誤差関数 SSE_n は

$$SSE_n = \sum_{j=1}^{q+1} \sum_{i=1}^{N_n} |u_j^n(x^{n,i}) - u^{n,i}|^2$$

ここで、初期条件に対するデータの数は N_n

境界条件への当てはまりに対する損失関数 SSE_b は

$$SSE_b = \sum_{i=1}^{q} \left(|u^{n+c_i}(-1)|^2 + |u^{n+c_i}(1)|^2 \right) + |u^{n+1}(-1)|^2 + |u^{n+1}(1)|^2$$

境界条件の具体的な値は与えておらず、N_b にあたる値がないことに注意

期待される利点

大きな段数qや時間ステップΔtに対しても
有用な解を一度の計算で、かつより高い精度で求めることができる

本来のルンゲ＝クッタ法では、精度のために段数を増やすと計算量が増大

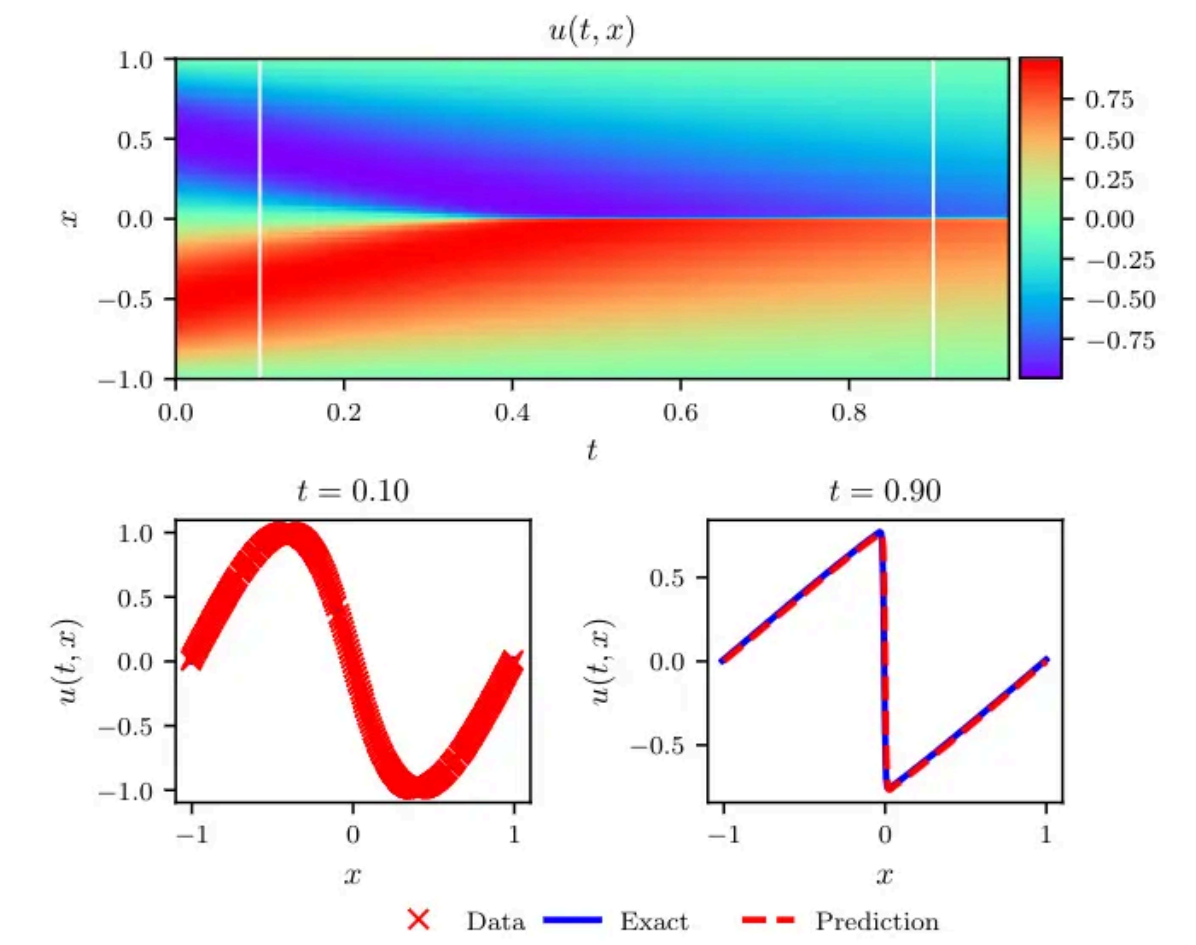
実験では、$q = 500$、$\Delta t = 0.8$での適用を試みる

実験

- $t = 0.1$におけるuの値から、$t = 0.9$の解を計算 (時間ステップ$\Delta t = 0.8$)
- $N_n = 250$
- ルンゲ＝クッタ法の段数$q = 500$

qとΔtの組み合わせ、およびレイヤ数・ニューロン数と性能の関係は論文を参照

支配方程式・境界条件へのあてはまり



相対二乗誤差(RSE): 8.2×10^{-4}

まとめ

PINNs: 物理法則を考慮した深層学習のアプローチ

2つの適用方法を示し、その有用性を示唆

- **Continuous Time Models**: 時空間領域上で連続する解を求める
  少ない教師データで良い結果を得られた
- **Discrete Time Models**: ある時間からある時間への解の時間発展を求める
  少ない計算量で高精度な時間発展の結果を得られた

議論

- 偏微分方程式を解くための古典的な手法を完全に置き換えるものではない
  古典的な手法も、多くの場合、それだけで妥当なロバスト性や計算効率を実現する
  - 古典的な手法との調和により、より優れた予測モデルの設計を実現しうる
    例えば、Discrete Time Modelsはルンゲ＝クッタ法を取り入れている
  - 実装のシンプルさにより、新しいアイデアの創出に寄与しうる
    Part IIにて、データ駆動のパラメータの発見という形で更に強調される
- 論文発表時点では、ニューラルネットワークによる
  予測の不確実性を定量的に把握できない → future work
  従来手法の一つであるガウス過程回帰では、自然に考慮された

ご清聴ありがとうございました

参考: PyTorchを用いた実装例 (Burgers, Continuous Time Models)

https://colab.research.google.com/drive/1yxV3gqjij-LULqGvU5NaYd3X3F9XkbuP?usp＝sharing