J. Plasma Fusion Res. Vol.102, No.3 (2026) 118-124

講座 ニューラルネットワークを用いた物理モデリング・シミュレーション

3. ニューラル作用素学習

3. Neural Operator Learning

田中 佑典[1,2)]
TANAKA Yusuke
[1)]NTT 株式会社 コミュニケーション科学基礎研究所，[2)]理化学研究所革新知能統合研究センター

（原稿受付：2025 年 7 月 10 日）

物理シミュレーションの超高速化に向けて「作用素学習」と呼ばれる新たなアプローチが注目を集めている．作用素学習では，ニューラルネットワークを用いて「関数から関数への写像」を学習することを目的とする．例えば，ある偏微分方程式に対して，初期条件から解を出力するニューラルネットワークを構築することで，初期値問題の高速解法として用いることができる．本章では，作用素学習の問題設定を紹介し，代表手法について解説する．また，作用素学習の枠組みに物理法則を組み込んだ学習法についても述べる．

Keywords:
physics simulation, partial differential equation, operator learning, deep operator network, neural operator

3.1 はじめに

作用素学習 (operator learning) は，2020 年頃に登場した新しい深層学習フレームワークである [1–3]．標準的なニューラルネットワークでは，有限次元のベクトル空間の間の写像を学習する．これに対し，作用素学習では，無限次元のベクトル空間（正確には無限次元の関数空間）の間の写像を学習する．言い換えると，**ある関数が入力として与えられたとき，それに対応する別の関数を出力する**ようなニューラルネットワークを構築するというのが，作用素学習の目的である．例えば，ある偏微分方程式に対して，入力関数を初期条件，出力関数を方程式の解に設定すれば，作用素学習は初期値問題の解を得るための**解作用素**をモデル化しているとみなせる．方程式を解くためには有限要素法などの数値解法を用いることが一般的であるが，これらは膨大な計算時間を要する．作用素学習では，解作用素を一度学習してしまえば，新たに与えられた初期条件に対する解をニューラルネットワークのフォワードパスを通じて瞬時に得ることができる．したがって，作用素学習は，**物理シミュレーションを劇的に高速化しうる手段**（〜数 100 倍程度）として大きなインパクトを秘めている．実際に，ナビエ・ストークス方程式などを対象にした計算機実験により作用素学習の有効性が示されており，プラズマ物理 [4,5] や気象予測 [6]，材料科学 [7] など様々な分野への応用が進んでいる所である．

一方で，複雑な物理系に対して，解作用素の高精度な学習は簡単でないことも多い．そこで近年，機械学習分野では，解作用素をモデル化するためのニューラルネットワークアーキテクチャ，および，その学習法についての研究が盛んに行われている [1,2,8–10]．本章では，最もよく使われる手法である深層作用素ネットワーク (Deep Operator Network: DeepONet) [1], および, ニューラル作用素 (Neural Operator: NO) [2] を中心に解説する．また，第 2 章における Physics-informed Neural Network (PINN) [11] の考え方を取り入れ，データだけでなく方程式も活用して作用素を学習する方法についても触れる．

3.2 作用素学習

本節では，作用素学習の全体像を把握することを目標とする．作用素学習では，入出力関数のペアがデータとして与えられたとき，それぞれが含まれる関数空間の間の写像を推定することを目的とする．作用素学習は一般的な枠組みであり，関数から関数への写像であれば，理論上はどんな問題にも適用が可能である．ここで，標準的な作用素学習の枠組みは，完全なデータ駆動型アプローチであり，物理法則等を組み込んだものではないことに注意されたい．

3.2.1 一般的な問題設定

まずは問題を物理シミュレーションに限定せず，一般的な作用素学習の問題設定について述べる．**図 1** に，2 つの関数空間をつなぐ作用素を視覚的に表す．入力関数空間を $\mathcal{A}$ とし，出力関数空間を $\mathcal{U}$ とする．これらの関数空間は必要に応じてバナッハ空間やヒルベルト空間などが採用される．2 つの関数空間の間の写像は「作用素」と呼ばれ，$\mathcal{S}:\mathcal{A}\to\mathcal{U}$ と表す．つまり入力関数を $a\in\mathcal{A}$ とし，出力関数を $u\in\mathcal{U}$ とすると，作用素 $\mathcal{S}$ は，

$$u=\mathcal{S}[a] \tag{1}$$

1) NTT Communication Science Laboratories, 2) RIKEN Center for Advanced Inteligence Project
author's e-mail: ysk.tanaka@ntt.com

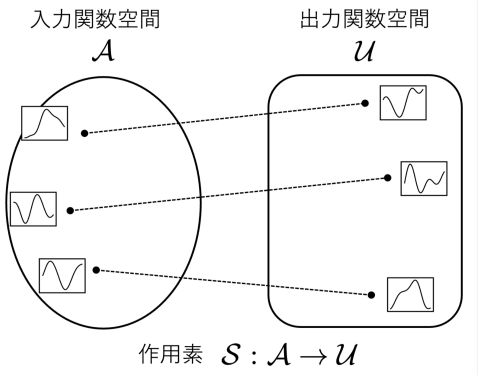

図 1　二つの関数空間をつなぐ作用素．

という関係を満たす．ここで，$[\cdot]$ は入力として関数をとることを意味する．学習時には，N 個の入出力関数のペア

$$\{(a_i, u_i) \mid i = 1, \cdots, N\} \tag{2}$$

が与えられるものとし，これを学習データとして $\mathcal{S}_\theta$ を推定する[*1]．ここで，$\mathcal{S}_\theta$ は真の作用素 $\mathcal{S}$ の近似であり，パラメータ θ を持つニューラルネットワークでパラメタライズされていることを表す．$\mathcal{S}_\theta$ の具体的なモデルは第 3.3 節で述べる．テスト時には，学習データに含まれていない入力関数 a^* が与えられたとき，学習済みの作用素を用いて出力関数を $u^* = \mathcal{S}_\theta[a^*]$ のように求めることができる．

3.2.2　解作用素の学習

・対象とする系

以下では，作用素学習の枠組みに基づき，偏微分方程式に関する問題を解くことを考えよう．時空間ドメイン $\mathcal{T} \times \mathcal{X}$ 上で定義される偏微分方程式は，

$$\frac{\partial u}{\partial t} = F_{\theta_{\mathrm{PDE}}}(t, \boldsymbol{x}, u, u_{\boldsymbol{x}}, u_{\boldsymbol{xx}}, \cdots) \quad (t, \boldsymbol{x}) \in \mathcal{T} \times \mathcal{X} \tag{3}$$

$$u(t_0, \boldsymbol{x}) = u^{(0)}(\boldsymbol{x}) \quad \boldsymbol{x} \in \mathcal{X} \tag{4}$$

$$\mathcal{B}[u](t, \boldsymbol{x}) = 0 \quad (t, \boldsymbol{x}) \in \mathcal{T} \times \partial\mathcal{X} \tag{5}$$

と表される．ここで，$u : \mathcal{T} \times \mathcal{X} \to \mathbb{R}$ は解を表し，$u^{(0)} : \mathcal{X} \to \mathbb{R}$ は初期条件を，$\mathcal{B}[u](t, \boldsymbol{x})$ は空間ドメイン $\mathcal{X}$ の境界 $\partial\mathcal{X}$ 上の境界条件を表す[*2]．また，θ_{PDE} は方程式の右辺に含まれる物理パラメータを表し，$u_{\boldsymbol{x}}$ および $u_{\boldsymbol{xx}}$ は，u の $\boldsymbol{x}$ に対する 1 階および 2 階の偏微分を表す．

・データの準備

以下では，作用素学習に基づいて**初期値問題**を解く場合について述べる．初期値問題では，入力関数を $a = u^{(0)}(\boldsymbol{x})$（初期条件）とし，出力関数を $u = u(t, \boldsymbol{x})$（解）とする．学習データは，入力関数を確率的に複数生成し，それに対応する解を適切な数値解法（ルンゲ・クッタ法や有限要素法など）により求めることで作成する[*3]．入力関数の生成方法には様々なものが考えられる．例えば，種々の多項式（三角多項式やチェビシェフ多項式など）を用いて，多項式の係数を確率的にサンプリングすることで複数の初期条件を生成する方法や，ガウス過程からのサンプルを初期条件として用いる方法がある．初期値問題を考える際には，方程式 $F_{\theta_{\mathrm{PDE}}}$ や境界条件 $\mathcal{B}$ は固定されていることに注意されたい．

このようにして得られた入出力関数のペアを学習データとするが，実際に計算機上で扱う際には，関数を離散化した表現を考えるのが自然である．入力関数（初期条件）に対する離散化を $\bar{\mathcal{X}}^{(a)} \subset \mathcal{X}$ とし，離散点における入力関数値の集合を $\bar{a}$ とすると，N 個の入力関数の集合は，

$$\left\{\left(\bar{\mathcal{X}}_i^{(a)}, \bar{a}_i\right) \mid i = 1, \cdots, N\right\} \tag{6}$$

と表される．同様にして，各入力関数に対する出力関数の離散表現を考える．出力関数（解）に対する離散化を $\overline{\mathcal{T} \times \mathcal{X}} \subset \mathcal{T} \times \mathcal{X}$ とし，離散点における出力関数値の集合を $\bar{u}$ とすると，N 個の出力関数の集合は，

$$\left\{\left((\overline{\mathcal{T} \times \mathcal{X}})_i, \bar{u}_i\right) \mid i = 1, \cdots, N\right\} \tag{7}$$

と表される．ここで重要なことは，**サンプルの引数 i によって離散化（点の数や位置）が異なっていても良い**という点である．多くの論文では，それぞれの離散化は異なっていても良いと断った上で，入出力関数のペア集合

$$\{(\bar{a}_i, \bar{u}_i) \mid i = 1, \cdots, N\} \tag{8}$$

が与えられると簡潔に書かれていることも少なくない．

・解作用素のモデル化

初期値問題に対する作用素学習の目的は，入力関数 $a = u^{(0)}(\boldsymbol{x})$（初期条件）から出力関数 $u = u(t, \boldsymbol{x})$（解）への写像 $u = \mathcal{S}[a]$ を満たす解作用素 $\mathcal{S}$ をニューラルネットワークを用いて推定することであった．しかし，ニューラルネットワークは有限次元のベクトル空間の間の写像をモデル化するため，関数から関数への写像（つまり無限次元ベクトル空間の間の写像）を直接扱うことはできない．そこで，解作用素を

$$u(t, \boldsymbol{x}) = \mathcal{S}_\theta[\bar{a}](t, \boldsymbol{x}) \tag{9}$$

のようにモデル化することを考える．典型的には，$\mathcal{S}_\theta$ は 4 つの性質を満たすことが望ましいとされる．

性質 1 任意の離散化に対する入力関数 $\bar{a}$ を入力することができる．

性質 2 任意の点 $(t, \boldsymbol{x})$ を入力として，その点に対応する出力関数の値 $u(t, \boldsymbol{x})$ を出力できる．

性質 3 離散化が異なる同じ入力関数に対して出力関数が同じになる．このような性質を**離散化不変性** (discretization invariance) という．

性質 4 適切な条件の下で[*4]，任意の作用素を任意の精度で

---

*1　計算機上で実装する際には，離散点における関数値の集合が与えられることを想定する．詳細は次節で述べる．

*2　簡単のため，u および $u^{(0)}$ のコドメインは $\mathbb{R}$ とするが，ベクトル値関数の場合にも拡張可能である．

*3　ここではシミュレーションデータを用いて学習することを想定するが，実データを用いることも可能である．

*4　入力関数空間がコンパクト集合であることや，作用素の連続性，作用素を構成するニューラルネットワークの幅や深さが十分に大きいこと等が仮定される．本稿では作用素学習の理論には踏み込まない．

近似できる．このような性質を**普遍近似性** (universal approximation) という．

$\mathcal{S}_\theta$ の具体的なモデルは第 3.3 節で述べる．

・パラメータ学習

次に，作用素モデル $\mathcal{S}_\theta$ のパラメータ θ を学習する方法について述べる．作用素学習の損失 $L(\theta)$ は，

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell\left(\bar{u}_i, \mathcal{S}_\theta[\bar{a}_i]\right) \tag{10}$$

と表され，個々のサンプルに対する損失 $\ell(\cdot,\cdot)$ は平均二乗誤差を用いて，

$$\ell\left(\bar{u}_i, \mathcal{S}_\theta[\bar{a}_i]\right) = \frac{1}{|(\mathcal{T}\times\mathcal{X})_i|}\sum_{(t,\boldsymbol{x})\in(\mathcal{T}\times\mathcal{X})_i} \left(\bar{u}_i(t,\boldsymbol{x}) - \mathcal{S}_\theta[\bar{a}_i](t,\boldsymbol{x})\right)^2 \tag{11}$$

と表される．ここで，$|\cdot|$ は集合の要素数を表す．Adam [12] などの確率的勾配降下法を用いて，式 (10) を最小化することによりパラメータの最適化が可能である．

・テスト

テスト時には，学習データに含まれていない入力関数 a^* に対して，出力関数 u^* が正確に予測可能かを評価する．このような未知の入力に対する予測精度のことを汎化性能 (generalization performance) という．また，学習時に用いた出力関数の離散点以外での予測性能を評価するために，u^* の離散化を，より高解像度なメッシュに設定することがある．このように学習データには含まれないような高解像度データを予測するタスクのことを**ゼロショット超解像** (zero-shot super-resolution) [8] と呼ぶことがある．

3.2.3 その他の問題設定

第 3.2.1 節で述べたように，作用素学習は一般的な枠組みであり，初期値問題以外にも，入出力関数を変更することによって様々な問題を扱うことができる．例えば，入力関数を境界条件とし，出力関数を解として作用素学習を行えば，境界条件を変更したときの物理シミュレーションの結果を効率的に出力する作用素を得ることができる．その他にも，入力関数を解とし，出力関数を初期条件や物理パラメータ（例えば $\theta_{\mathrm{PDE}}: \mathcal{X}\to\mathbb{R}$）として作用素学習を行えば，逆問題を解くための作用素を構成することも可能である [13,14]．

3.3 作用素のニューラルネットワークモデル

本節では，作用素モデル $\mathcal{S}_\theta$ の具体的な設計について述べる．2 つの代表的なモデルとして，深層作用素ネットワーク (Deep Operator Network: DeepONet) [1] とニューラル作用素 (Neural Operator: NO) [2,3] がある．これらのモデルは同時期に提案されたが，それぞれが異なる視点に基づいて設計されている．本稿では，これらを独立な手法として解説するが，二つのモデルの関係性についても議論されており，DeepONet は NO の特殊な場合とみなせることが指摘されている [3,15]．**表 1** に，標準的なニューラルネットワークと作用素モデルとの比較を示す．

表 1 標準的なニューラルネットワーク (NN) と作用素モデル (DeepONet, NO) との比較．

	NN	DeepONet	NO
性質 1（任意の入力離散化）	✗	✗	✓
性質 2（関数出力）	✗	✓	✓
性質 3（離散化普遍性）	✗	✗	✓
性質 4（作用素の普遍近似性）	✗	✓	✓

3.3.1 深層作用素ネットワーク (DeepONet)

表 1 に示すように，DeepONet [1] は性質 1 と性質 3 を満たさない．それでもなお，ニューラルネットワークを用いて作用素をモデル化しようとした初めての試みとしての価値は大きく，実装が容易であるという利点もあり，現在でも広く使われている[*5]．また，本節の最後に触れるが，性質 1 と性質 3 を満たすような DeepONet の改良版も提案されている．

DeepONet は以下に示すような作用素に対する普遍近似定理 [16] に基づき設計されている．

定理 1 任意の $\epsilon > 0$ に対して，以下の不等式を満たすような正の整数 P, Q, R と定数 $\beta_p^{(q)}, \xi_{pr}^{(q)}, b_p^{(q)}, \boldsymbol{w}_p, c_p$ が存在する．ここで，$\langle\cdot,\cdot\rangle$ はベクトルの内積を表し，$\sigma(\cdot)$ は非線形関数を表す[*6]．

$$\left|\mathcal{S}[a](t,\boldsymbol{x}) - \overbrace{\sum_{p=1}^{P}\sum_{q=1}^{Q}\left\{\beta_p^{(q)}\,\sigma\left(\sum_{r=1}^{R}\xi_{pr}^{(q)}\,\bar{a}(\boldsymbol{x}_r) + b_p^{(q)}\right)\right.}^{\text{Branch}} \left.\times \underbrace{\sigma\Big(\langle \boldsymbol{w}_p, (t,\boldsymbol{x})\rangle + c_p\Big)}_{\text{Trunk}}\right\}\right| < \epsilon \tag{12}$$

定理 1 は，R 個の離散点における入力関数値 $\{\bar{a}(\boldsymbol{x}_r)\}$ に対して，任意の点 $(t,\boldsymbol{x})$ における出力関数値を求める作用素 $\mathcal{S}$ を近似することを考えている．$\sigma(\cdot)$ が非線形な活性化関数 (activation function) を表していると考えれば，式 (12) における Branch と書かれた因子は $\{\bar{a}(\boldsymbol{x}_r)\}$ を入力とする全結合型ニューラルネットワーク (fully connected neural network) の一種とみなすことができる．また，Trunk と書かれた因子は $(t,\boldsymbol{x})$ を入力とする同様のニューラルネットワークの一種とみなすことができる．

このような観察に基づき設計されたのが DeepONet である．**図 2** に DeepONet のアーキテクチャを示す[*7]．DeepONet は，Branch net および Trunk net と呼ばれる二つのニューラルネットワークを用いて構成される．これらは既存のニューラルネットワークモデルを用いて実装することができ，最もよく使われるのは多層パーセプトロン (Multi-layer perceptron: MLP) である．Branch net は離散

*5 実用上，入力関数が固定された離散化に基づく場面も少なくないと考えられる．

*6 定理が成り立つ条件については [16] を参照されたい．

*7 文献 [1] における Unstacked DeepONet の形態を示す．

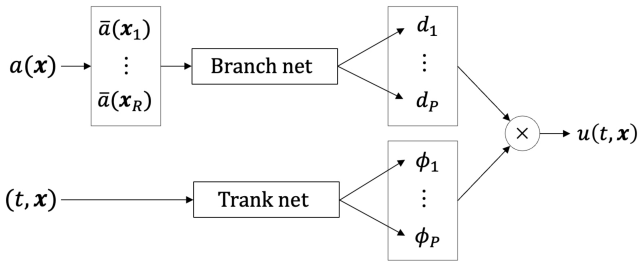

図 2　DeepONet のアーキテクチャ．

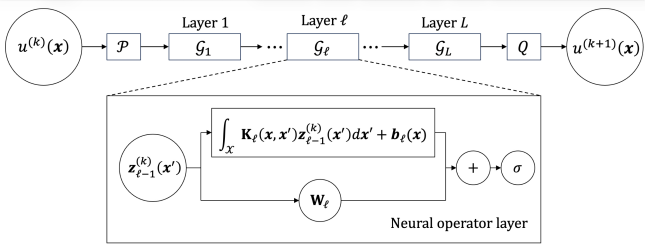

図 3　ニューラル作用素のアーキテクチャ．

化された入力関数 $\bar{a}$ を P 次元ベクトル $(d_1, \ldots, d_P) \in \mathbb{R}^P$ に変換し，Trank net は任意の点 $(t, \boldsymbol{x})$ を別の P 次元ベクトル $(\phi_1, \ldots, \phi_P) \in \mathbb{R}^P$ に変換する．得られたベクトルを用いて，作用素モデル $\mathcal{S}_\theta$ は，

$$\mathcal{S}_\theta[\bar{a}](t, \boldsymbol{x}) = \sum_{p=1}^{P} d_p(\bar{a}) \phi_p(t, \boldsymbol{x}) + b_0 \tag{13}$$

と表される．ここで，$b_0 \in \mathbb{R}$ はバイアスパラメータである．また，d_p は $\bar{a}$ に依存し，ϕ_p は $(t, \boldsymbol{x})$ に依存することを明確にするため，$d_p(\bar{a})$ および $\phi_p(t, \boldsymbol{x})$ と表した．式 (13) からわかるように，Trank net は基底関数 $\phi_p(t, \boldsymbol{x})$ をモデル化し，Branch net は重み $d_p(\bar{a})$ をモデル化していると解釈できる．つまり，DeepONet は基底関数表現をニューラルネットワークにより推定し，その重みつき和で出力関数 $u(t, \boldsymbol{x})$ を表現するモデルであると言える．

学習時は，式 (10) を最小化することで，Branch net や Trank net を構成するニューラルネットワークのパラメータを含む全ての未知パラメータを推定する．ただし，基底関数の個数 P はハイパーパラメータであり，評価用データを用いた汎化性能の推定等を通じて決定する必要がある．

本節の冒頭で述べたように，DeepONet は作用素モデルが満たすべき性質 1 と性質 3 を満たしていないが，既存の深層学習技術を組み合わせることで解決可能である．**図 2** において，Branch net の入力が R 次元に固定されていることが主な問題である．一つの解決策としては，Branch net を，可変サイズの集合を入力として扱うことが可能なアーキテクチャでモデル化することである．例えば，Deep Sets を用いた手法 [17] や Transformer を用いた手法 [18] などが提案されている．

3.3.2　ニューラル作用素 (NO)

本節では，作用素モデルを構築するために広く用いられている NO [2,3] について解説する．NO では，第 3.2.1 節で述べたような初期値問題を扱うことも可能であるが，固定幅の離散時刻における解（状態）の遷移をモデル化することが多い．$\bar{\mathcal{T}} = \{t_0, t_1, \cdots, t_K\} \subset \mathcal{T}$ を離散時刻の集合とし，ある時刻 $t_k \in \bar{\mathcal{T}}$ における解を $u^{(k)}(\boldsymbol{x}) = u(t_k, \boldsymbol{x})$ としたとき，解の遷移は作用素 $\mathcal{S}$ を用いて，

$$u^{(k)}(\boldsymbol{x}) = \mathcal{S}[u^{(k-1)}](\boldsymbol{x}) \qquad k = 1, \cdots, K \tag{14}$$

と表される．観測データは，

$$\left\{ \left(\bar{\mathcal{X}}_i^{(k)}, \bar{u}_i^{(k)} \right) \mid i = 1, \cdots, N;\ k = 0, \cdots, K \right\} \tag{15}$$

であり，空間離散化 $\bar{\mathcal{X}}$ はサンプルの引数 i および時刻の引数 k によって異なっていても良い．このとき，式 (10) における損失 $\ell(\cdot, \cdot)$ は，

$$\ell\left(\bar{u}_i, \mathcal{S}_\theta[\bar{u}_i]\right) = \frac{1}{K} \sum_{k=1}^{K} \left(\frac{1}{|\bar{\mathcal{X}}_i^{(k)}|} \sum_{\boldsymbol{x} \in \bar{\mathcal{X}}_i^{(k)}} \left(\bar{u}_i^{(k)}(\boldsymbol{x}) - \mathcal{S}_\theta[\bar{u}_i^{(k-1)}](\boldsymbol{x}) \right)^2 \right) \tag{16}$$

と表され，式 (10) の損失を最小化することによって，式 (14) における真の作用素 $\mathcal{S}$ に対する作用素モデル $\mathcal{S}_\theta$ を学習する．

以下では，NO によって作用素モデル $\mathcal{S}_\theta$ がどのように定式化されるかを述べる．**図 3** に NO のアーキテクチャを示す．MLP 等の順伝搬型ニューラルネットワークと同様に，NO は多層構造を持ち，作用素 $\mathcal{S}$ を

$$\mathcal{S}[u^{(k)}](\boldsymbol{x}) \approx (\mathcal{Q} \circ \mathcal{G}_L \circ \cdots \circ \mathcal{G}_1 \circ \mathcal{P})\, [u^{(k)}](\boldsymbol{x}) \tag{17}$$

のように近似する．ここで，$\circ$ は関数あるいは作用素の合成を表す．式 (17) は以下の 3 つの構成要素からなる．

- **Lifting**: 関数 $\mathcal{P} : \mathbb{R} \to \mathbb{R}^{M_0}$ によって構成され，入力関数 $u^{(k)} : \mathcal{X} \to \mathbb{R}$ を潜在表現 $\boldsymbol{z}_0^{(k)} : \mathcal{X} \to \mathbb{R}^{M_0}$ に変換する．関数 $\mathcal{P}$ は MLP 等を用いて実装され，$\boldsymbol{z}_0^{(k)}$ は入力関数よりも高次元なベクトル値関数に設定される．ここで，$\mathcal{P}$ は局所的な変換であり，入力関数の点毎 (pointwise) に作用する．
- **Iterative Kernel Integration**: Neural operator layer $\mathcal{G}_\ell$ を用いた多層構造により構成される（**図 3** 参照）．$\mathcal{G}_\ell$ は，$\ell = 1 \cdots, L$ に対して，$\boldsymbol{z}_{\ell-1}^{(k)} : \mathcal{X} \to \mathbb{R}^{M_{\ell-1}}$ を $\boldsymbol{z}_\ell^{(k)} : \mathcal{X} \to \mathbb{R}^{M_\ell}$ に変換する．$\mathcal{G}_\ell$ により，関数の大域的なダイナミクスを捉えることが可能となる．詳細は次の段落で述べる．
- **Projection**: 関数 $\mathcal{Q} : \mathbb{R}^{M_L} \to \mathbb{R}$ によって構成され，潜在表現 $\boldsymbol{z}_L^{(k)} : \mathcal{X} \to \mathbb{R}^{M_L}$ を出力関数 $u^{(k+1)} : \mathcal{X} \to \mathbb{R}$ に変換する．関数 $\mathcal{Q}$ は MLP 等を用いて実装される．$\mathcal{Q}$ は局所的な変換であり，潜在表現の点毎に作用する．

次に，$\mathcal{G}_\ell$ について述べる．$\mathcal{G}_\ell$ は，潜在表現である二つの関数の間の写像を表し，

$$\boldsymbol{z}_\ell^{(k)}(\boldsymbol{x}) = \mathcal{G}_\ell \left[\boldsymbol{z}_{\ell-1}^{(k)} \right] (\boldsymbol{x}) \tag{18}$$

$$= \sigma \left(\mathbf{W}_\ell \boldsymbol{z}_{\ell-1}^{(k)}(\boldsymbol{x}) + \underbrace{\int_{\mathcal{X}} \kappa_\ell(\boldsymbol{x}, \boldsymbol{x}') \boldsymbol{z}_{\ell-1}^{(k)}(\boldsymbol{x}')\, d\boldsymbol{x}'}_{\text{カーネル積分作用素}} + b_\ell(\boldsymbol{x}) \right) \tag{19}$$

と表される．ここで，$\mathbf{W}_\ell \in \mathbb{R}^{M_\ell \times M_{\ell-1}}$ は $\boldsymbol{z}_{\ell-1}^{(k)}(\boldsymbol{x})$ に局所的に作用する線形変換を表し，$\sigma(\cdot)$ は各点毎に対する非線形変換を表す活性化関数である．また，$b_\ell(\boldsymbol{x}) : \mathcal{X} \to \mathbb{R}$ はバイアスを表す関数であり，MLP 等を用いてモデル化される．式 (19) の括弧内における第 2 項は**カーネル積分作用素** (kernel integral operator) と呼ばれる．カーネル積分作用素は，入力として関数 $\boldsymbol{z}_{\ell-1}^{(k)}(\boldsymbol{x})$ をとり，局所的ではなく，大域的な関数のダイナミクスを捉えることを可能とする．カーネル積分作用素を構成する $\kappa_\ell : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^{M_\ell \times M_{\ell-1}}$ はカーネル関数と呼ばれる．カーネル積分作用素の設計については後の段落で紹介する．

カーネル積分作用素は，有限次元ベクトル空間の間の写像を表すニューラルネットワークを，無限次元ベクトル空間の間の写像を表すように拡張するための橋渡しの役割を担っている．関数 $\boldsymbol{z}_{\ell-1}^{(k)}(\boldsymbol{x})$ は，$\mathcal{X}$ 上の無限個の点における関数値の集合であると言えるので，直感的には，カーネル積分作用素は，無限次元の行列 κ_ℓ を用いた線形変換であると考えることができる．

最後に，式 (19) におけるカーネル積分作用素の設計について述べる．NO の研究は今まさに様々な角度から進展中であるが，多くの研究がカーネル積分作用素のモデル化や実装方法に焦点を当てている．以下では，カーネル積分作用素の実装例をいくつか紹介する．NO においても，入力関数 $u^{(k)}(\boldsymbol{x})$ は離散化されていることを想定するため，潜在表現 $\boldsymbol{z}_\ell^{(k)}(\boldsymbol{x})$ も同様に離散化される．したがって，カーネル積分作用素が，入力関数の離散化によらず（つまり性質 1 や性質 3 を満たす）ように設計するのが望ましい．

- **グラフニューラル作用素** (Graph Neural Operator: GNO) [2]: ある点 $\boldsymbol{x} \in \mathcal{X}_i$ に対して，近傍集合 $\mathcal{N}_i(\boldsymbol{x}) = \{\boldsymbol{y} \in \mathcal{X}_i \,;\, \| \boldsymbol{y} - \boldsymbol{x} \| < r\}$ を定義する．ここで，$r > 0$ は定数を表す．GNO では，離散点 $\boldsymbol{x} \in \mathcal{X}_i$ をノードとし，近傍集合 $\mathcal{N}_i(\boldsymbol{x})$ をエッジとするグラフを考えることにより，式 (19) のカーネル積分作用素を

$$\frac{1}{|\mathcal{N}_i(\boldsymbol{x})|} \sum_{\boldsymbol{x}' \in \mathcal{N}_i(\boldsymbol{x})} \kappa_\ell(\boldsymbol{x}, \boldsymbol{x}') \boldsymbol{z}_{\ell-1}^{(k)}(\boldsymbol{x}') \tag{20}$$

と計算する．式 (20) は，グラフニューラルネットワークにおけるメッセージ伝播の仕組み [19] と同様である．近傍からの情報を集約するという処理によって，入力関数の離散化に依存しない計算を可能としている．ここで，カーネル κ_ℓ は MLP 等を用いてモデル化され，r はハイパーパラメータである．
- **フーリエニューラル作用素** (Fourier Neural Operator: FNO) [8]: 式 (19) から，カーネル積分作用素は畳み込み積分で表されることがわかる．カーネルが平行移動不変[*8]であることを仮定すると，フーリエ解析における畳み込み定理より，カーネル積分作用素は，

$$\mathcal{F}^{-1}\left[\mathcal{F}[\kappa_\ell] \cdot \mathcal{F}[\boldsymbol{z}_{\ell-1}^{(k)}] \right] \tag{21}$$

と表される．ここで，$\mathcal{F}$ および $\mathcal{F}^{-1}$ は，フーリエ変換およびフーリエ逆変換を表す．実装時には，入力関数は離散化されているので高速フーリエ変換が用いられる[*9]．

FNO では，フーリエ解析で行われるのと同様に，高周波成分をカットし，S 個のフーリエモード $\mathcal{F}[\boldsymbol{z}_{\ell-1}^{(k)}] \in \mathbb{C}^{S \times M_{\ell-1}}$ を用いる．ここで，S はハイパーパラメータとして扱う．高速フーリエ変換を用いて，固定数のフーリエモードを抽出するという処理によって，FNO は入力関数の離散化に依存しない計算が可能となる．また，式 (21) の $\mathcal{F}[\kappa_\ell] \in \mathbb{C}^{S \times M_\ell \times M_{\ell-1}}$ は，周波数領域における線形変換を表している．FNO では，この線形変換を表す行列 $\mathbf{H}_\ell$ を直接パラメタライズし，

$$\mathcal{F}^{-1}\left[\mathbf{H}_\ell \cdot \mathcal{F}[\boldsymbol{z}_{\ell-1}^{(k)}] \right] \tag{22}$$

と表す．ここで，$\mathbf{H}_\ell$ は単純にパラメータとして導入するか，周波数の関数としてモデル化される．FNO の利点の一つは，フーリエ変換に基づくことで，関数の大域的なダイナミクスを考慮できることである．
- **その他の手法**: NO の亜種はここでは紹介しきれないほど様々なものが提案されている．例えば，カーネル積分作用素におけるカーネルを Sinc 関数とし，エイリアシングを防ぐようにモデル化された畳み込みニューラル作用素 (Convolutional Neural Operator: CNO) [9] や，フーリエ変換の代わりにウェーブレット変換を用いて局所的なダイナミクスを考慮した手法 [20] がある．他にも，FNO を不規則なグリッドでも使えるように拡張した Geometry-aware FNO (Geo-FNO) [21] や，球面調和解析 (Spherical Harmonic Transform) を活用し，球面上の物理現象に適用可能とした Spherical FNO (SFNO) [22] などが提案されている．

3.4 物理法則を組み込んだ作用素学習

作用素学習は，物理シミュレーションを劇的に高速化しうる手段であるが，正しいシミュレーション結果を得るためには，ニューラルネットワークによって作用素を高精度に近似する必要がある．良質な学習データを膨大に使用すれば，作用素の近似誤差は一般に限りなく小さくなるが，データ作成や学習にかかるコストが大きくなり，あまり現実的ではない．また，得られた作用素が，基本的な物理法則を満たさないことも十分に起こりうるため，信頼性に欠けるという問題もある．そこで，近年のひとつの研究の潮

*8 $\kappa_\ell(\boldsymbol{x}, \boldsymbol{x}') = \kappa_\ell(\boldsymbol{x} - \boldsymbol{x}')$ を満たす．

*9 ただし，高速フーリエ変換を用いる場合は $\{\tilde{\mathcal{X}}_i^{(k)}\}_{k=1}^K$ が，ある等間隔のグリッドであることが要求される．一方，引数 i に対しては異なる解像度のグリッドを使用できる．

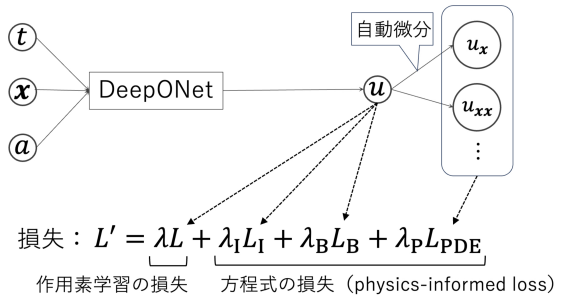

図 4　Physics-informed DeepONet のアーキテクチャ．

流[*10]として，データだけではなく，物理法則を組み込んだ作用素学習法が提案されている．

最も代表的な研究として，第 2 章における Physics-informed Neural Network (PINN) の考え方を用いて，DeepONet をデータと方程式から学習する手法が提案されており，Physics-informed DeepONet (PI-DeepONet) [24] と呼ばれる．**図 4** に，PI-DeepONet のアーキテクチャを示す．アイデアは非常にシンプルであり，作用素学習の損失 $L(\theta)$（式 (10) 参照）に，PINN と同様に，初期条件，境界条件，および，方程式に関する損失（Physics-informed loss と呼ばれる）を追加する．PI-DeepONet の損失 $L'(\theta)$ は，

$$L'(\theta) = \lambda L(\theta) + \lambda_{\mathrm{I}} L_{\mathrm{I}}(\theta) + \lambda_{\mathrm{B}} L_{\mathrm{B}}(\theta) + \lambda_{\mathrm{P}} L_{\mathrm{PDE}}(\theta) \quad (23)$$

と表される．ここで，$L_{\mathrm{I}}(\theta)$ および $L_{\mathrm{B}}(\theta)$ は，式 (4) および式 (5) を DeepONet の出力である $u(t, \boldsymbol{x})$ が満たすように平均二乗誤差を用いて導入される[*11]．また，$L_{\mathrm{PDE}}(\theta)$ は式 (3) を満たすように平均二乗誤差を用いて導入され，方程式を評価するために必要な t や $\boldsymbol{x}$ についての偏微分は，DeepONet の出力 $u(t, \boldsymbol{x})$ を自動微分することによって得られる．式 (23) における $\lambda, \lambda_{\mathrm{I}}, \lambda_{\mathrm{B}}, \lambda_{\mathrm{P}}$ は PINN と同様に何らかの手段で決定すべきハイパーパラメータである．PINN では，入力関数 a に相当する初期条件等を変更すると，解を求めるための再学習が必要であったが，PI-DeepONet では再学習をすることなく，学習された作用素を用いて直ちに解を出力可能であるという利点がある．

同様のアイデアに基づき FNO に Physics-informed loss を導入する方法も提案されている [25]．この研究では，テスト時に新たな条件や方程式が与えられた際に，ファインチューニングを通じて，さらなる精度向上が達成されることが示されている．また，第 3 章で扱われたハミルトニアンニューラルネットワーク [26] のアイデアを活用し，エネルギー関数であるハミルトニアンを推定しつつ，エネルギーの保存・散逸則を満たすような作用素学習の方法も提案されている [27]．

*10　作用素学習に限らず，物理法則を組み込んだ機械学習手法に関する研究分野は Physics-informed machine learning [23] と呼ばれる．

*11　データ損失である $L(\theta)$ に初期条件や境界条件における損失が含まれている場合は取り除くことができる．

3.5　おわりに

本稿では，深層学習研究の新たなパラダイムのひとつである「作用素学習」について述べた．作用素学習は，物理シミュレーションを劇的に高速化しうる手段として大きな期待がある一方で，高精度な学習のためにはさらなる技術開発が必要である．そのためのアプローチのひとつとして，物理法則を考慮した学習方法が検討されていることを述べた．その意味でも，本分野の研究は，機械学習の専門家だけでなく，物理や応用数学などの関連分野との連携が非常に重要である．最近では，作用素学習の Python パッケージ（DeepXDE など）も複数公開されているので，ぜひプログラムを動かしてみられることをおすすめする．また，本稿では作用素学習の理論には踏み込まなかったが，DeepONet や FNO の普遍近似定理（**表 1** の性質 4 に関連する）に興味のある方は，文献 [1,3] を参照いただきたい．今後，分野横断的な研究がより活発化し，本分野の深化や拡がりが加速されることを期待する．

参　考　文　献

[1] L. Lu *et al.*, Nat. Mach. Intell. **3**(3), 218–229 (2021).
[2] Z. Li *et al.*, arXiv:2003.03485 (2020).
[3] N. Kovachiki *et al.*, JMLR 23, 1-97 (2022).
[4] V. Gopakumar *et al.*, NeurIPS AI for Science Workshop (2021).
[5] M. Bonotto *et al.*, Fusion Eng. Des. **200**, 114193 (2024).
[6] J. Pathak *et al.*, arXiv, 2202.11214 (2022).
[7] W. Li *et al.*, Comput. Methods Appl. Mech. Eng. 416:116299 (2023).
[8] Z. Li *et al.*, ICLR (2021).
[9] B. Raonic *et al.*, NeurIPS (2023).
[10] M. Liu-Schiaffini *et al.*, ICML (2024).
[11] M. Raissi et al., J. Comput. Phys. **378**, 686–707 (2019).
[12] D.P. Kingma and J. Ba, ICLR (2015).
[13] R. Molinaro *et al.*, ICML (2023).
[14] T. Wang and C. Wang, NeurIPS (2024).
[15] L. Lu *et al.*, Comput. Methods Appl. Mech. Eng. 393(1):114778 (2022).
[16] T. Chen and H. Chen, IEEE Trans. Neural Netw. Learn. Syst. 6(4):911–917 (1995).
[17] S. Tretiakov *et al.*, arXiv, 2505.04738 (2025).
[18] M. Prasthofer *et al.*, arXiv, 2205.11404 (2022).
[19] 佐藤竜馬：グラフニューラルネットワーク (講談社, 2024).
[20] G. Gupta *et al.*, NeurIPS (2021).
[21] Z. Li *et al.*, JMLR (2023).
[22] B. Bonev *et al.*, ICML (2023).
[23] G. Karniadakis *et al.*, Nat. Rev. Phys. **3**(6), 422–400 (2021).
[24] S. Wang *et al.*, Sci. Adv. **7**(40), eabi8605 (2021).
[25] Z. Li *et al.*, arXiv, 2111.03794 (2021).
[26] S. Greydanus et al., NeurIPS (2019).
[27] Y. Tanaka *et al.*, AISTATS (2025).

田中（たなか） 佑典（ゆうすけ）

NTT 株式会社 コミュニケーション科学基礎研究所 研究主任．2011 年 神戸大学工学部電気電子工学科卒業．2013 年 京都大学大学院情報学研究科システム科学専攻博士前期課程修了．同年，NTT 株式会社入社．以来，機械学習に関する研究に従事．2020 年 京都大学大学院情報学研究科システム科学専攻博士後期課程修了．博士（情報学）．ペットのデグーを飼っています．