New Ways to Design Deep Neural Networks*

Xue-Cheng Tai[1], Hao Liu[2,*], Raymond H. Chan[3] and Lingfeng Li[4]

[1]Norwegian Research Centre, Nygårdsgaten 112, 5008 Bergen, Norway
[2]Department of Mathematics, Hong Kong Baptist University, Kowloon Tong, Kowloon, Hong Kong SAR
[3]Lingnan University, Tuen Mun, Hong Kong SAR
[4]Hong Kong Centre for Cerebro-Cardiovascular Health Engineering, Hong Kong SAR

Abstract

In this work, we propose a general framework for designing neural network architectures inspired by dynamic differential equations, utilizing the operator-splitting technique. The central idea is to treat neural network design as a discretizations of a continuous-time optimal control problem, where the underlying dynamics are governed by differential equations serving as constraints which is then unrolled as our network. These dynamics are discretized through operator-splitting schemes, which allow complex evolution equations to be decomposed into simpler substeps. Each step in the splitting scheme is then unrolled and interpreted as a layer in a neural network, with certain control variables modeled as learnable parameters. This formulation provides a principled way to incorporate prior knowledge about dynamics and structure into the network design. Using our theory, we give a rigorous mathematical explanation of the well-known UNet and show that it is a discretizations of a simple differential equation. By adding regularization to UNet, we can derive the PottsMGNet also through our proposed framework.

Keywords

deep neural network, control problem, operator splitting, UNet, image segmentation

1. Introduction

Deep neural networks have achieved remarkable success across diverse tasks, including image segmentation [1, 2, 3], image denoising [4, 5], and natural language processing [6]. Extensive research has sought to explain the empirical success of deep neural networks and establish connections with mathematical models. One line of research focuses on representation and generalization theory. These studies demonstrate that, with properly designed architectures, deep networks can approximate target functions [7], functionals [8], or operators [9] to arbitrary accuracy while achieving generalization errors dependent on sample size.

Another research direction leverages mathematical models to interpret network behaviors and inspire new architectures. For instance, works such as [10] reinterpret neural networks as discretized dynamical systems, while [11] explores links to control theory. Unrolling is used in [12, 13] to incorporate networks with mathematical algorithms. Inspired by some control problems, some stable architectures have been proposed in [14, 15]. The weak formulation of PDEs motivated Zang et al. [16] to design weak adversarial networks, which was later extended to constrained optimization in [17]. Recent works like [18, 19, 20] unify the Potts model, operator-splitting methods, and control theory to derive interpretable networks. For example, Tai et al. [18] developed PottsMGNet, which achieved robust

NAIS 2025: Symposium of the Norwegian AI Society, June 17–18, 2025, Tromsø, Norway

*The work of Xue-Cheng Tai is partially supported by NORCE Kompetanseoppbygging program. The work of Hao Liu is partially supported by NSFC 12201530 and HKRGC ECS 22302123. The work of Raymond H. Chan is partially supported by HKRGC GRF grants CityU1101120, CityU11309922, CRF grant C1013-21GF, and HKRGC-NSFC Grant N-CityU214/19. The work of Lingfeng Li is supported by the InnoHK project at Hong Kong Centre for Cerebro-Cardiovascular Health Engineering (COCHE)

*Corresponding author.

xtai@norceresearch.no (X. Tai); haoliu@hkbu.edu.hk (H. Liu); raymond.chan@ln.edu.hk (R. H. Chan); lfli@hkcoche.org (L. Li)

https://www.norceresearch.no/personer/xue-cheng-tai (X. Tai); https://www.math.hkbu.edu.hk/~haoliu/ (H. Liu); https://raymondhonfu.github.io/ (R. H. Chan)

0000-0003-3359-9104 (X. Tai); 0000-0002-7504-9859 (H. Liu); 0000-0003-0910-4685 (R. H. Chan)

image segmentation across noise levels via a unified framework grounded in multigrid and control perspectives. Notably, their analysis revealed that many encoder-decoder networks implicitly implement operator-splitting algorithms for control problems. Further advancing this interplay, Liu et al. [20] proposed a double-well network that learns region force terms in the Potts model through neural representations. We also want to mention that the work [21] has provided another way to interpret neural networks through multigrid techniques.

In this work, we aim to present a general framework for designing neural networks based on dynamic differential equations using operator-splitting technique. The basic idea is discretizing a continuous control problem using the dynamic system as a constraint which is then approximated by some operator-splitting schemes. We unroll this splitting scheme as a neural network with some control variables parameterized as learnable modules. We also illustrate this approach through examples of the UNet and the PottsMGNet.

2. The Central Ideas

A central insight of our work is that deep neural networks (DNN) can be naturally derived as discretizations of continuous control problems governed by partial differential equations. This perspective provides a unifying framework to design and interpret network architectures through dynamical systems and optimal control.

Consider a continuous control problem where the state evolution is described by a PDE of the form:

$$\partial_t u = \mathscr{A}(u, \theta) \text{ for } (\mathbf{x}, t) \in \Omega \times [0, T], \qquad u(0) = u_0 \text{ in } \Omega, \tag{1}$$

where Ω is the spatial computational domain, $\mathscr{A}$ is a differential operator parameterized by the control variable θ which could depends on both $\mathbf{x}, t$ or one of them. The function u_0 is the initial condition. By properly discretizing this PDE in time by operator-splitting methods and unrolling the dynamics over discrete steps, we can derive a numerical scheme for solving (1). It is shown that this scheme has the same architecture of a network, for which the control variables plays the role of network learnable parameters. Based on this scheme, one optimizes control variables to minimize a given cost functional so that $u(T)$ matches the target state. The optimization procedure is the same as network training.

This is a general framework that applies to all networks. From another perspective, it also implies that for any given network, we can find a corresponding control problem so that this network is a numerical scheme (with proper discretization and unrolling) solving this problem. This framework offers several advantages:

- **Theoretical Analysis**: Tools from PDE theory (e.g., stability, convergence, and regularity analysis) can be applied to study the behavior of DNNs.
- **Architecture Design**: New network architectures can be derived by choosing appropriate PDE models and discretization schemes. It enables to equip the designed neural networks with different kinds of physical properties which are often desired, but have not been able to achieve with existing networks.
- **Scalability, Interoperability and Explainability**: The continuous viewpoint provides insights into the role of depth, initialization, and parameterization in DNNs.

In the following sections, we give details of this idea and demonstrate how to use operator-splitting methods and unrolling to discretize the control problems. Applications of this framework on UNet [1] and PottsMGNet [18] will be discussed.

3. Discretization by Operator-Splitting Methods

Assume that we have decomposed the differential operator $\mathscr{A}$ in the following form:

$$\partial_t u = \sum_{i=1}^{K} \mathscr{A}_i(u, \theta), \quad u(0) = u_0,$$

where each $\mathscr{A}_i(u,\theta)$ is possibly nonlinear and acts as a distinct operator. Operator-splitting methods approximate the full solution by evolving u through each operator sequentially or in parallel over a time step τ. For example, the sequential splitting [22, 23] evolves the solution by applying the sub-solvers sequentially:

$$u^{n+1} = S_K^{\tau,t^n} \circ \cdots \circ S_1^{\tau,t^n}(u^n),$$

where $S_i^{\tau,t^n}(\hat{u}) = u(t^n+\tau)$ denotes the solution operator of

$$\partial_t u = \mathscr{A}_i(u,t), \quad u(t^n) = \hat{u}.$$

The solution operator $S_i^{\tau,t^n}(\hat{u})$ can either be solved explicitly or approximated numerically. The parallel splitting scheme [24] solves each sub-problem in parallel and combines them together

$$u^{n+1} = \frac{1}{K}\sum_{i=1}^{K} S_i^{K\tau,t^n}(u^n).$$

We may also apply the sequential and parallel schemes together to form hybrid splitting schemes as explained in Appendix A, see also [18, Appendix D] for some more details.

After splitting, unrolling methods are used to construct neural networks. Unrolling maps the splitting scheme to a neural network architecture, where each operator S_i becomes a network layer. Specifically, we have the following construction:

1. Layer structure: Each time step $u^n \to u^{n+1}$ is unrolled into K sub-layers, one per operator S_i. If $\mathscr{A}_i$ is known (e.g., a Laplacian), we implement S_i as a fixed layer. If $\mathscr{A}_i$ contains unknown learnable parameters, we represent S_i as a trainable neural network block.
2. Parameterization: Replace unknown $\mathscr{A}_i$ with learnable modules $\mathscr{A}_i^{\theta_i}(u,t)$ with parameters $\theta_i(t)$. The operator S_i becomes a learnable layer S_i^{τ,t^n,θ_i}.
3. Unrolled network: By replacing S_i with S_i^{τ,t^n,θ_i} in the splitting scheme, we get a neural network model N_Θ with learnable parameters $\Theta = \{\theta_i(t^n)\}_{i=1,n=1}^{K,N}$.

This unrolled network can be trained by minimizing the discrepancy between predictions from a given initial condition u_0^j and the corresponding ground-truth solutions y^j (e.g., MSE):

$$\mathscr{L}(\Theta) = \sum_{j=1}^{\hat{N}} \left\| N_\Theta(u_0^j) - y^j \right\|^2.$$

where $\{u_0^j, y^j\}_{j=1}^{\hat{N}}$ is a set of training samples. Let us emphasis that we can use other loss functions instead of MSE, such as the cross-netropy loss. Different deep neural networks in the literature are just different approximations of some specially chosen PDEs or ODEs. In the following section, we will show that the well-known UNet [1] is just an unrolling of a numerical approximation of a special PDE, see (2).

4. UNet as Multigrid Operator Splitting

The well-known deep neural network UNet [1] is designed to segment images into foreground and background represented by a binary image. By applying the framework in Section 2-3, we show that it is just a numerical approximation of a PDE.

4.1. The control problem

Given an input image f defined on the image domain Ω, we consider the following initial value problem

$$\begin{cases} \frac{\partial u(\mathbf{x},t)}{\partial t} = W(\mathbf{x},t) * u(\mathbf{x},t) + d(t) - \epsilon \ln \frac{u(\mathbf{x},t)}{1-u(\mathbf{x},t)}, \ (\mathbf{x},t) \in \Omega \times (0,T], \\ u(\mathbf{x},0) = H(f(\mathbf{x})), \ \mathbf{x} \in \Omega, \end{cases} \tag{2}$$

where $*$ denotes convolution, $W(\mathbf{x},t), d(t)$ are control variables which will be learned in the training process. The convolution kernel $W(\mathbf{x},t) : D\times(0,T]\to\mathbb{R}$ is supported on some spatial domain $D\subset\mathbb{R}^2$. In practice, D can be different from Ω, and is usually a small region. In this paper, for simplicity, we take $D=\Omega=[-1,1]^2$. When computing convolution, we extend functions to $\mathbb{R}^2$ by padding convolution kernels $W(\mathbf{x},t)$ by 0 and solution function $u(\mathbf{x},t)$ periodically.

Equation (2) evolves any input image f into a probability in $u(x,t)\in[0,1]$, see [18, Appendix A] and [25] for some more details with the derivation of this equation. Above, $H(f)$ is some operation to generate initial condition from f. Due to the appearance of the term $\ln\frac{u}{1-u}$, the solution of the above equation is forced to be in $(0,1)$. For numerical consideration and to make the connection between operator-splitting methods and neural networks clearer, we introduce a constraint $u(x,t)\geq 0$ to the control problem. Due to the property of the term $\ln\frac{u}{1-u}$, the introduced constraint does not change the solution. Next, we incorporate the constraint into the equation by introducing an indicator function

$$\begin{cases}\frac{\partial u}{\partial t}-W(\mathbf{x},t)*u-d(t)+\epsilon\ln\frac{u}{1-u}+\partial\mathcal{I}_\Sigma(u)\ni 0,\ (\mathbf{x},t)\in\Omega\times(0,T],\\ u(\mathbf{x},0)=H(f(\mathbf{x})),\ \mathbf{x}\in\Omega,\end{cases}\tag{3}$$

where

$$\Sigma=\{u\,:\,u(\mathbf{x},t)\geq 0 \text{ for } (\mathbf{x},t)\in\Omega\times(0,T]\},$$

$\mathcal{I}_\Sigma$ is the indicator function of Σ and $\partial\mathcal{I}_\Sigma$ denotes the subdifferential of $\mathcal{I}_\Sigma$. By solving (3) for any input image f, the initial value $u(\mathbf{x},0)$ will evolve to $u(\mathbf{x},T)$, which is a probability function. By choosing ϵ close to 1, we can force this probability function to be close to a binary function. For simplify of explanation, we will take $\epsilon=1$ and this is also the value the original UNet is using.

To solve (3) numerically, [26] decomposed control variables (learnable variables) $\{W(\mathbf{x},t),d(t)\}$ in (3) using the multigrid idea. Here, we introduce a simplified version of their algorithm that can recover a simple UNet-like structure. The discussion can be generalized to recover the original UNet structure from (3).

4.2. Decomposition of control variables

In traditional multigrid methods, a popular framework is the "fine grid → coarse grid → fine grid" strategy [27, 28]. Such a form of V-cycle multigrid method can be interpreted as space decomposition and subspace correction [29, 30].

Motivated by the fact that images can be viewed as piecewise constant functions in practice, in this paper, we consider piecewise constant approximation of different resolutions. Let $s>0$ be some integer. For the finest resolution, we consider discretizing D into $(2^s)^2$ small squares, each of which is of size $(2^{-s+1})\times(2^{-s+1})$. They are called pixels in image processing and rectangular elements in finite elements methods. In real applications, the finest mesh is the grid the input image is given. Denote $h_1=2^{-s+1}$ and this set of pixles by $\mathscr{T}^1$. Starting with $\mathscr{T}^1$, we can define a sequence of coarse pixels $\{\mathscr{T}^j\}_{j=1}^{s+1}$. Specifically, let $h_j=2^{j-1}h_1=2^{j-s}$. The set $\mathscr{T}^j$ consists of coarse pixels with size $h_j\times h_j$ that form a partition of D. For each $\mathscr{T}^j$, we can define a set of piecewise constant basis functions. For the k-th pixel in $\mathscr{T}^j$, denoted by ω_k, we define $\phi_k^j(\mathbf{x})$ such that it equals to 1 if $\mathbf{x}\in\omega_k$ and equals to 0 otherwise. We denote the space spanned by this set of functions by $\mathscr{V}^j$. We have that

$$\mathscr{V}^{s+1}\subset\mathscr{V}^s\subset\cdots\subset\mathscr{V}^1.\tag{4}$$

Note that each function in $\mathscr{V}^j$ has $2^{2(s+1-j)}$ coefficients and $\dim(\mathscr{V}^j)=2^{2(s+1-j)}$. Traditional multigrid methods solve the decomposed subproblems by simple Gauss-Seidel or Jacobi iterations. Here, the multigrids problems are used in a different way.

We will decompose $\{W(\mathbf{x},t),d(t)\}$ into a sum of variables with different scales over the multigrids. Then, we use a hybrid splitting method to solve (3) so that all decomposed variables are distributed into several subproblems, which are solved sequentially or in parallel. Within one iteration of the splitting method, all decomposed variables are gone through. The general splitting idea is to split the operators

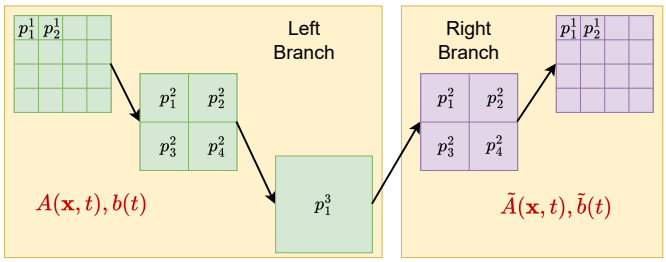

Figure 1: An illustration of the V-cycle and the decomposition (5) with $J = 3$. The p_k^j's illustrate patches for (8) with $m = 1$.

based on a V-cycle according to the grid level, cf. Figure 1 for an illustration. We decompose all terms in the right-hand side of (3) via the following steps:

1. According to the idea of a V-cycle, we decompose $W(\mathbf{x},t)$ and $d(t)$ as

$$W(\mathbf{x},t) = A(\mathbf{x},t) + \tilde{A}(\mathbf{x},t),\ d(t) = b(t) + \tilde{b}(t). \tag{5}$$

   These variables will be further decomposed next. Above, A, b are sums of control variables in the left branch of the V-cycle, and $\tilde{A}, \tilde{b}$ are sums of the control variables in the right branch, see an illustration in Figure 1. We also decompose the nonlinear operator as follows:

$$-\ln\frac{u}{1-u} - \partial\mathcal{J}_\Sigma(u) = S(u) + \tilde{S}(u). \tag{6}$$

   Here, $S(u)$ contains nonlinear operations in the left branch and $\tilde{S}(u)$ contains nonlinear operations in the right branch. In particular, we put $-\ln\frac{u}{1-u}$ in $\tilde{S}(u)$ only, i.e., $S(u) = \partial\mathcal{J}_\Sigma(u)$ and $\tilde{S}(u) = \partial\mathcal{J}_\Sigma(u) - \ln\frac{u}{1-u}$. Later, we will show that our operator splitting method recovers a simplified UNet, in which the operation $-\ln\frac{u}{1-u}$ corresponds to the sigmoid layer at the end of UNet.
2. We further decompose the operators into components at different grids as:

$$\begin{aligned} A(\mathbf{x},t) &= \sum_{j=1}^{J} A^j(\mathbf{x},t), \quad b(t) = \sum_{j=1}^{J} b^j(t), \quad S(u) = \sum_{j=1}^{J} S^j(u), \\ \tilde{A}(\mathbf{x},t) &= \sum_{j=1}^{J-1} \tilde{A}^j(\mathbf{x},t) + A^*(\mathbf{x},t), \quad \tilde{b}(t) = \sum_{j=1}^{J-1} \tilde{b}^j(t) + b^*(t), \quad \tilde{S}(u) = \sum_{j=1}^{J-1} \tilde{S}^j(u) + S^*(u), \end{aligned} \tag{7}$$

   where $A^j, b^j, \tilde{A}^j, \tilde{b}^j$ contain control variables at grid level j, A^*, b^* are control variables that are applied to the output of the V-cycle at the finest mesh. Operators $S^j(u) = \tilde{S}^j(u) = \partial\mathcal{J}_\Sigma(u)$ are applied to the intermediate solution on grid level j. Operator $S^*(u) = -\ln\frac{u}{1-u}$ is applied to the output of the V-cycle at the finest mesh.
3. At grid level j in the left branch, A^j is defined on D, which contains $2^{2(s+1-j)}$ coefficients to be learned. This leads to a high complexity when j is small. In order to decrease the complexity, we decompose A^j into a sum of A_k^j which is nonzero only on small patch p_k^j of size $(mh_j)\times(mh_j)$, denoted by $\{A_k^j\}_{k=1}^{c_j}$, where c_j is the total number of patches p_k^j. Normally, we take $m = 3$ or 5 and all p_k^j shall cover D. Patches with $J = 3, m = 1$ is illustrated in Figure 1. Each A_k^j equals to A^j over p_k^j if the support sets p_k^j do not overlap. We also decompose b^j and S^j into a sum of c_j components. Thus we have

$$A^j(\mathbf{x},t) = \sum_{k=1}^{c_j} A_k^j(\mathbf{x},t),\ b^j = \sum_{j=1}^{c_j} b_k^j(t),\ S^j = \sum_{k=1}^{c_j} S_k^j(u). \tag{8}$$

Note that A_k^j's have different supports, making specifying the support for each function complicated. In order to simplify the settings, we shift p_k^j so that they are centered at $(0,0)$. Specifically, for each A_k^j, let χ_k^j be a shifting kernel so that $A_k^j = \chi_k^j * \hat{A}_k^j$ where $\hat{A}_k^j$ supported on a square centered at $(0,0)$ is a shifted version of A_k^j. According to the shift-invariant property of convolution, we have

$$A^j(\mathbf{x},t) * u(\mathbf{x},t) = \sum_{k=1}^{c_j} A_k^j(\mathbf{x},t) * u(\mathbf{x},t) = \sum_{k=1}^{c_j} (\chi_k^j(\mathbf{x},t) * \hat{A}_k^j(\mathbf{x},t)) * u(\mathbf{x},t)$$
$$= \sum_{k=1}^{c_j} \hat{A}_k^j(\mathbf{x},t) * (\chi_k^j(\mathbf{x},t) * u(\mathbf{x},t)). \tag{9}$$

Thus learning a kernel of with $2^{2(s+1-j)}$ coefficients is converted to learning c_j kernels $\hat{A}_k^j(\mathbf{x},t)$ with $m \times m$ coefficients arond the origin. In implementation, c_j corresponds to the number of channels in a CNN and $\{A_k^j\}_{k=1}^{c_j}$ corresponds to CNN convolution kernels of size $m \times m$. In our experiments, for simplicity, we omit the shifting operator χ_k^j and replace $(\chi_k^j(\mathbf{x},t) * u(\mathbf{x},t))$ by $u(\mathbf{x},t))$ and it gives good results. This is also what is done in the original UNet. The same decomposition is done for $\tilde{A}^j, \tilde{b}^j$ and $\tilde{S}^j$.

4. For each grid level j and each channel k, we further decompose

$$\hat{A}_k^j(\mathbf{x},t) = \sum_{s=1}^{c_{j-1}} A_{k,s}^j(\mathbf{x},t), \tag{10}$$

and $A_{k,s}^j(\mathbf{x},t)$ has support around the origin with $m \times m$ coefficients. The purpose of this decomposition is to increase the number of parameters for the training variables. In our algorithm, each channel will compute an intermediate result. In the computation, $A_{k,s}^j$ is used to convolve with the intermediate solution in the s-th channel of grid level $j-1$. The same decomposition is conducted for $\tilde{A}_k^j$.

Before the decomposition, the control variables are $W(\mathbf{x},t), b(t)$. After these decompositions, we see that the control variables are decomposed as in the following and the control variables are now $A_{k,s}^j(\mathbf{x},t), b_k^j(t), \tilde{A}_{k,s}^j(\mathbf{x},t), \tilde{b}_k^j(t), A_{k,s}^*(\mathbf{x},t), b_k^*(t),$:

$$A(\mathbf{x},t) = \sum_{j=1}^{J}\sum_{k=1}^{c_j}\sum_{s=1}^{c_{j-1}} A_{k,s}^j(\mathbf{x},t), \qquad \tilde{A}(\mathbf{x},t) = \sum_{j=1}^{J-1}\sum_{k=1}^{c_j}\sum_{s=1}^{c_{j-1}} \tilde{A}_{k,s}^j(\mathbf{x},t) + \sum_{s=1}^{c_1} A_s^*(\mathbf{x},t), \tag{11}$$

$$b(\mathbf{x},t) = \sum_{j=1}^{J}\sum_{k=1}^{c_j} b_k^j(\mathbf{x},t), \qquad \tilde{b}(\mathbf{x},t) = \sum_{j=1}^{J-1}\sum_{k=1}^{c_j} \tilde{b}_k^j(\mathbf{x},t) + \tilde{b}^*(\mathbf{x},t), \tag{12}$$

and the operators $S(u), \tilde{S}(u)$ are decomposed as:

$$S(u) = \sum_{j=1}^{J}\sum_{k=1}^{c_j} S_k^j(u),\ \tilde{S}(u) = \sum_{j=1}^{J-1}\sum_{k=1}^{c_j} \tilde{S}_k^j(u) + S^*(u) \tag{13}$$

$$\text{with } S_k^j(u) = \tilde{S}_k^j(u) = \partial\mathcal{I}_\Sigma(u),\ S^*(u) = -\ln\frac{u}{1-u}. \tag{14}$$

The original PDE (2) is turned to

$$\begin{cases} \frac{\partial u}{\partial t} = A * u + \tilde{A} * u + b + \tilde{b} + S(u) + \tilde{S}(u),\ (\mathbf{x},t) \in \Omega \times [0,T], \\ u(\mathbf{x},0) = H(f),\ \mathbf{x} \in \Omega. \end{cases} \tag{15}$$

To solve (15), we use a hybrid splitting method shown in Appendix A. Divide the time interval $[0,T]$ into N subintervals with time step $\tau = T/N$. Denote the computed solution at time $t^n = n\tau$ by U^n. The resulting algorithm that updates U^n to U^{n+1} is summarized in Algorithm 1, where $\mathscr{D}$ and $\mathscr{U}$ represent the downsampling and upsampling operator respectively. For simplicity, variable dependencies on $\mathbf{x}$ are omitted.

Algorithm 1: A hybrid splitting method to approximate the PDE (15) for UNet

Data: The initial condition solution U^n.
Result: The computed solution U^{n+1}.
Set $c_0 = 1, v_1^0 = \bar{v}^0 = U^n$.
for $j = 1, \cdots, J$ **do**
  for $k = 1, \cdots, c_j$ **do**
    Compute $v_k^j \in \mathscr{V}^j$ by solving

$$\frac{v_k^j - \mathscr{D}(v^{j-1})}{2^{j-1}c_j\tau} - \sum_{s=1}^{c_{j-1}} A_{k,s}^j(t^n) * \mathscr{D}(v_s^{j-1}) - b_k^j(t^n) - S_k^j(v_k^j) \ni 0. \tag{16}$$

  end for
  Compute $v^j = \frac{1}{c_j}\sum_{k=1}^{c_j} v_k^j$.
end for
Set $u^J = v^J$, and $u_k^J = v_k^J$ for $k = 1, ..., c_J$.
for $j = J-1, \cdots, 1$ **do**
  Compute for $k = 1, ..., c_j$

$$u_k^j = \frac{1}{2}v_k^j + \frac{1}{2}\mathscr{U}(u^{j+1}) \tag{17}$$

  for $k = 1, \cdots, c_j$ **do**
    Compute $u_k^j \in \mathscr{V}^j$ by solving

$$\frac{u_k^j - \mathscr{U}(u^{j+1})}{2^{j-1}c_j\tau} - \sum_{s=1}^{c_{j+1}} \tilde{A}_{k,s}^j(t^n) * \mathscr{U}(u_s^{j+1}) - \tilde{b}_k^j(t^n) - \tilde{S}_k^j(u_k^j) \ni 0. \tag{18}$$

  end for
  Compute $u^j = \frac{1}{c_j}\sum_{k=1}^{c_j} u_k^j$.
end for
Compute U^{n+1} by solving

$$\frac{U^{n+1} - u^1}{\tau} - A^*(t^n) * u^1 - b^*(t^n) - S^*(U^{n+1}) \ni 0. \tag{19}$$

4.3. On the solution to (16), (18) and (19)

Observe that (16) and (18) are in the form of

$$\frac{u - u^*}{\gamma\tau} - \sum_{s=1}^{c} \hat{A}_s * u_s^* - \hat{b} + \partial\mathscr{I}_\Sigma(u) \ni 0, \tag{20}$$

where γ is some constant, $u^* = \frac{1}{c}\sum_{s=1}^{c} u_s^*$ is known, $\hat{A}_s$ is a convolution operator, c is the number of input channel, $\hat{b}$ is a bias function. The solution to (20) is computed by a two-sub-step splitting method:

$$\begin{cases} \bar{u} = u^* + \gamma\tau\left(\sum_{s=1}^{c} \hat{A}_s * u_s^* + \hat{b}\right), \\ \frac{u-\bar{u}}{\gamma\tau} + \partial\mathscr{I}_\Sigma(u) \ni 0. \end{cases} \tag{21}$$

In (21), there is no difficulty in solving for $\bar{u}$ in the first sub-step as it is an explicit step. Let us emphasis that the term $\sum_{s=1}^{c} \hat{A}_s * u_s^* + \hat{b}$ in this step is exactly the Pytorch function Conv2d. For u in the second sub-step, it is, in fact, a projection. Its closed-form solution is given as

$$u = \max\{\bar{u}, 0\} = \mathrm{ReLU}(\bar{u}), \tag{22}$$

where $\mathrm{ReLU}(u) = \max\{\bar{u}, 0\}$ is the rectified linear unit.

Problem (19) can be written as

$$\frac{u - u^*}{\gamma\tau} = \hat{A} * u^* + \hat{b} - \ln\frac{u}{1-u}. \tag{23}$$

Following the steps for solving (16) and (18) above, we solve (23) as

$$\begin{cases} \bar{u} = u^* + \gamma\tau\left(\sum_{s=1}^{c} \hat{A}_s * u_s^* + \hat{b}\right), \\ \frac{u-\bar{u}}{\tau} = -\ln\frac{u}{1-u}. \end{cases} \tag{24}$$

The first sub-step is an explicit step using Pytorch function Conv2d. We solve the second sub-step approximately by a fixed point iteration. Initialize $p^0 = \bar{u}$. Given p^k, we update p^{k+1} by solving

$$\frac{p^k - \bar{u}}{\tau} = -\ln\frac{p^{k+1}}{1-p^{k+1}}, \tag{25}$$

for which we have the closed-form solution

$$p^{k+1} = \mathrm{Sig}\left(-\frac{p^k - \bar{u}}{\tau}\right), \tag{26}$$

where $\mathrm{Sig}(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. By repeating (26) so that p^{k+1} converges to some function p^*, we set $u = p^*$. In particular, since $p^0 = \bar{u}$, the updating formula (26) always gives $p^1 = 0.5$. If we only consider a two-step fixed point iteration, we get

$$u = \mathrm{Sig}\left(-\frac{0.5 - \bar{u}}{\tau}\right) = \mathrm{Sig}\left(\frac{\bar{u} - 0.5}{\tau}\right). \tag{27}$$

4.4. Algorithm 1 recover the UNet

We first show that a building block of Algorithm 1 is equivalent to a layer of a simplified UNet. Each layer of UNet is a convolution layer activated by ReLU:

$$\begin{cases} \bar{v} = \sum_{s=1}^{c} K_s * v_s^* + b, \\ v = \mathrm{ReLU}(\bar{v}), \end{cases} \tag{28}$$

where K_s is a convolutional kernel and b is the bias. In Algorithm 1, the building block is (20) and (23), which is solved by (21) and (24). In fact, (28) (or problem (24)) and (21) have the same form. Specifically, in the first equation of (21) we have

$$\bar{u} = u^* + \gamma\tau\left(\sum_{s=1}^{c} \hat{A}_s * u_s^*\right) + \gamma\tau\hat{b} = \sum_{s=1}^{c}\left(\mathbb{1}/c + \gamma\tau\hat{A}_s\right) * u_s^* + \gamma\tau\hat{b}, \tag{29}$$

where $\mathbb{1}$ denotes the identity kernel satisfying $\mathbb{1} * g = g$ for any function g. In (28), set

$$K_s = \mathbb{1}/c + \gamma\tau\hat{A}_s, \quad b = \gamma\tau\hat{b}. \tag{30}$$

We have $\bar{v} = \bar{u}$, and $v = u$. Essentially, Algorithm 1 and UNet are the same. Thus, we have shown that a simplified UNet structure (with only 1 convolution layer at each data resolution) is equivalent to one iteration of Algorithm 1. UNet architecture consists of four components: encoder, decoder, bottleneck, and skip-connections, each of which has a corresponding component in the structure of Algorithm 1:

1. **Encoder**: Encoder in UNet corresponds to the left branch of the V-cycle in Algorithm 1. The number of data resolution levels corresponds to the number of grid levels J.
2. **Decoder**: Decoder in UNet corresponds to the right branch of the V-cycle in Algorithm 1.
3. **Bottleneck**: Bottleneck in UNet corresponds to the computations at the coarsest grid level (grid level J) in Algorithm 1.
4. **Skip-layer connection**: Skip-layer connections in UNet correspond to the relaxation steps (17) in Algorithm 1.

Therefore, a one-step operator splitting of the control problem (15) is exactly equivalent to a simplified UNet.

Algorithm 2: A hybrid splitting method to solve the control problem (15) for PottsMGNet

Data: The initial condition solution U^n at time step t^n.
Result: The computed solution U^{n+1}.
Set $c_0 = 1, v_1^0 = \bar{v}^0 = U^n$.
for $j = 1, \cdots, J$ **do**
  for $k = 1, \cdots, c_j$ **do**
    Compute $v_k^j \in \mathscr{V}^j$ by solving

$$\frac{v_k^j - \mathscr{D}(v^{j-1})}{2^{j-1}c_j\tau} - \sum_{s=1}^{c_{j-1}} A_{k,s}^j(t^n) * \mathscr{D}(v_s^{j-1}) - b_k^j(t^n) - S_k^j(v_k^j) \ni 0. \tag{31}$$

  end for
  Compute $v^j = \frac{1}{c_j}\sum_{k=1}^{c_j} v_k^j$.
end for
Set $\bar{u}^J = v^J$, and $\bar{u}_k^J = v_k^J$ for $k = 1, ..., c_J$.
for $j = J-1, \cdots, 1$ **do**
  for $k = 1, \cdots, c_j$ **do**
    Compute $u_k^j \in \mathscr{V}^j$ by solving

$$\frac{u_k^j - \mathscr{U}(\bar{u}^{j+1})}{2^j c_j\tau} - \sum_{s=1}^{c_{j+1}} \tilde{A}_{k,s}^j(t^n) * \mathscr{U}(\bar{u}_s^{j+1}) - \tilde{b}_k^j(t^n) - \tilde{S}_k^j(u_k^j) \ni 0. \tag{32}$$

  end for
  Compute for $k = 1, ..., c_j$

$$\bar{u}_k^j = \frac{1}{2}v_k^j + \frac{1}{2}u_k^{j+1} \tag{33}$$

  Compute $\bar{u}^j = \frac{1}{c_j}\sum_{k=1}^{c_j} \bar{u}_k^j$.
end for
Compute U^{n+1} by solving

$$\frac{U^{n+1} - \bar{u}^1}{\tau} - A^*(t^n) * \bar{u}^1 - b^*(t^n) - S^*(U^{n+1}) \ni 0. \tag{34}$$

5. Case study: PottsMGNet

In the previous section, we presented how to use the framework introduced in Section 2-3 to show that the operator splitting scheme of (15) is equivalent to a UNet. In this section, we show that PottsMGNet proposed in [18] is another instance of this framework. Specifically, we will consider a modified control problem of (15) and a modified splitting scheme of Algorithm 1.

5.1. Constructing a PottsMGNet

We consider the control problem (15) with decomposition (5)-(10). Here, the decomposition of the nonlinear operator S and $\tilde{S}$ as in (14) are defined as

$$S_k^j(u) = -\frac{(2^{j-1})^{-1}}{\kappa}\ln\frac{u}{1-u}, \quad \tilde{S}_k^j(u) = -\frac{(2^j)^{-1}}{\kappa}\ln\frac{u}{1-u}, \quad S^*(u) = -\frac{1}{\kappa}\ln\frac{u}{1-u} - \eta G_\sigma * (1-2u),$$

where κ is a normalization term, η is a weight parameter, and G_σ is the Gaussian smoothing kernel with variance σ^2. To solve this new system, we consider a multi-step operator splitting algorithm (Algorithm 2) using the hybrid splitting method in Appendix A for approximating the solution of (15) with the above given split of operators.

When solving the subproblems (31) and (32), we adopt a similar sequential splitting method as (21). The second substep is then defined as

$$u = (\mathscr{I} - \gamma\tau S)^{-1}(\bar{u}),$$

which can be approximately solved by a fixed point iteration as for the second substep in (24). By unrolling Algorithm 2, we can get a simplified PottsMGNet [18].

5.2. Comparison with UNet

Compared with the UNet structure, the PottsMGNet starts with a control problem with a different nonlinear term S and $\tilde{S}$, which results in a different activation function at each layer of the resulted neural network. The position of the relaxation step is also different, which leads to a different skip-connection structure. Additionally, the UNet is a one-step algorithm while the PottsMGNet is a multi-step algorithm. Similar to UNet, we can also extend Algorithm 2 to a multi-channel case by further split the control variables.

6. Conclusion

In this work, we present a framework to design novel neural networks using operator splitting of some control problems. Networks designed in this way are usually robust to disturbances in the input, because concerned control problems usually include some regularization terms. We also presented how to derive UNet and PottsMGNet from this framework. In the future, we would apply this framework to other types of networks like graph neural networks.

Declaration on Generative AI

During the preparation of this work, the authors used DeepSeek in order to: Grammar and spelling check. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

Appendix

A. Hybrid Splitting Schemes for Initial Value Problems and Neural Network Design

We discuss a hybrid splitting scheme proposed in [18] as a powerful technique for solving initial value problems, which can be effectively leveraged to design and understand the architecture of deep neural networks. A hybrid splitting scheme combines the principles of parallel and sequential splitting schemes to decompose complex problems into more manageable subproblems.

Consider the initial value problem:

$$u_t + \sum_{m=1}^{M}\left(\sum_{k=1}^{c_m}\sum_{s=1}^{d_m} B_{k,s}^m(\mathbf{x},t;u) + \sum_{k=1}^{c_m} C_k^m(\mathbf{x},t;u) + \sum_{k=1}^{c_m} f_k^m(\mathbf{x},t)\right) = 0 \quad \text{on } \Omega\times[0,T],\ u(0)=u_0.$$

We can first split it into M sequential steps, where each step consists of c_m parallel substeps. At the $(n+1)$-th time step, denote the intermediate result computed in the k-th branch of the m-th sequential step by $u_k^{n+m/M}$, and define $u^{n+(m-1)/M} = \frac{1}{c_m}\sum_{k=1}^{c_m} u_k^{n+(m-1)/M}$. The function $u_k^{n+m/M}$ is computed by solving:

$$\frac{u_k^{n+m/M} - u^{n+(m-1)/M}}{c_m\tau} = -\sum_{s=1}^{d_m} B_{k,s}^m(\mathbf{x},t^n;u_s^{n+(m-1)/M}) - C_k^m(\mathbf{x},t^{n+1};u_k^{n+m/M}) - f_k^m(\mathbf{x},t^n).$$

Here, the operators $B_{k,s}^m$ are treated explicitly, while the operators C_k^m are treated implicitly. Functions f_k^m can be treated explicitly as shown here. Starting with $u^{n,0} = u^n$, this algorithm iterates through m from 1 to M and k from 1 to c_m to compute u^{n+1}.

When $B_{k,s}^m$ and C_k^m are Lipschitz continuous, this hybrid splitting scheme is first-order accurate [18, Theorem D.1], meaning that the error $\|u^{n+1} - u(t^{n+1})\|_\infty$ is of the order $O(\tau)$.

References

[1] O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in: International Conference on Medical image computing and computer-assisted intervention, Springer, 2015, pp. 234–241.

[2] Z. Zhou, M. M. R. Siddiquee, N. Tajbakhsh, J. Liang, Unet++: Redesigning skip connections to exploit multiscale features in image segmentation, IEEE transactions on medical imaging 39 (2019) 1856–1867.

[3] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, A. L. Yuille, Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, IEEE transactions on pattern analysis and machine intelligence 40 (2017) 834–848.

[4] K. Zhang, W. Zuo, Y. Chen, D. Meng, L. Zhang, Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising, in: IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2017, pp. 5743–5752.

[5] T. Wu, C. Huang, S. Jia, W. Li, R. Chan, T. Zeng, S. K. Zhou, Medical image reconstruction with multi-level deep learning denoiser and tight frame regularization, Applied Mathematics and Computation 477 (2024) 128795.

[6] A. Graves, A.-r. Mohamed, G. Hinton, Speech recognition with deep recurrent neural networks, in: 2013 IEEE international conference on acoustics, speech and signal processing, Ieee, 2013, pp. 6645–6649.

[7] A. R. Barron, Universal approximation bounds for superpositions of a sigmoidal function, IEEE Transactions on Information Theory 39 (1993) 930–945.

[8] L. Song, Y. Liu, J. Fan, D.-X. Zhou, Approximation of smooth functionals using deep relu networks, Neural Networks 166 (2023) 424–436.

[9] H. Liu, H. Yang, M. Chen, T. Zhao, W. Liao, Deep nonparametric estimation of operators between infinite dimensional spaces, Journal of Machine Learning Research 25 (2024) 1–67.

[10] W. E, A Proposal on Machine Learning via Dynamical Systems, Communications in Mathematics and Statistics 5 (2017) 1–11. doi:10.1007/s40304-017-0103-z.

[11] M. Benning, E. Celledoni, M. Ehrhardt, B. Owren, C. Schhönlieb, Deep learning as optimal control problems: models and numerical methods, Journal of Computational Dynamics (2019).

[12] K. Gregor, Y. LeCun, Learning fast approximations of sparse coding, in: Proceedings of the 27th international conference on international conference on machine learning, 2010, pp. 399–406.

[13] Y. Yang, J. Sun, H. Li, Z. Xu, Admm-csnet: A deep learning approach for image compressive sensing, IEEE transactions on pattern analysis and machine intelligence 42 (2018) 521–538.

[14] L. Ruthotto, E. Haber, Deep neural networks motivated by partial differential equations, Journal of Mathematical Imaging and Vision 62 (2020) 352–364.

[15] E. Haber, L. Ruthotto, Stable architectures for deep neural networks, Inverse Problems 34 (2018) 1–23. doi:10.1088/1361-6420/aa9a90. arXiv:1705.03341.

[16] Y. Zang, G. Bao, X. Ye, H. Zhou, Weak adversarial networks for high-dimensional partial differential equations, Journal of Computational Physics 411 (2020) 109409.

[17] G. Bao, D. Wang, B. Zou, Wanco: Weak adversarial networks for constrained optimization problems, arXiv preprint arXiv:2407.03647 (2024).

[18] X.-C. Tai, H. Liu, R. Chan, Pottsmgnet: A mathematical explanation of encoder-decoder based neural networks, SIAM Journal on Imaging Sciences 17 (2024) 540–594.

[19] H. Liu, X.-C. Tai, R. Chan, Connections between operator-splitting methods and deep neural networks with applications in image segmentation, Ann. Appl. Math 39 (2023) 406–428.

[20] H. Liu, J. Liu, R. Chan, X.-C. Tai, Double-well net for image segmentation, arXiv preprint arXiv:2401.00456 (2023).

[21] J. He, J. Xu, MgNet: A unified framework of multigrid and convolutional neural network, Science China Mathematics 62 (2019) 1331–1354. doi:10.1007/s11425-019-9547-2. arXiv:1901.10415.

[22] R. Glowinski, Finite element methods for incompressible viscous flow, Handbook of numerical analysis 9 (2003) 3–1176.

[23] R. Glowinski, T.-W. Pan, X.-C. Tai, Some facts about operator-splitting and alternating direction methods, in: Splitting Methods in Communication, Imaging, Science, and Engineering, Springer, 2016, pp. 19–94.
[24] T. Lu, P. Neittaanmaki, X.-C. Tai, A parallel splitting-up method for partial differential equations and its applications to navier-stokes equations, ESAIM: Mathematical Modelling and Numerical Analysis 26 (1992) 673–708.
[25] X. Tai, L. Li, E. Bae, The potts model with different piecewise constant representations and fast algorithms: a survey, Handbook of Mathematical Models and Algorithms in Computer Vision and Imaging: Mathematical Imaging and Vision (2021) 1–41.
[26] X.-C. Tai, H. Liu, R. H. Chan, L. Li, A mathematical explanation of unet, Mathematical Foundations of Computing (2024) 0–0.
[27] T. F. Chan, T. P. Mathew, Domain decomposition algorithms, Acta numerica 3 (1994) 61–143.
[28] J. Xu, Iterative methods by space decomposition and subspace correction, SIAM review 34 (1992) 581–613.
[29] X.-C. Tai, Rate of convergence for some constraint decomposition methods for nonlinear variational inequalities, Numerische Mathematik 93 (2003) 755–786.
[30] X.-C. Tai, J. Xu, Global and uniform convergence of subspace correction methods for some convex optimization problems, Mathematics of Computation 71 (2002) 105–124.