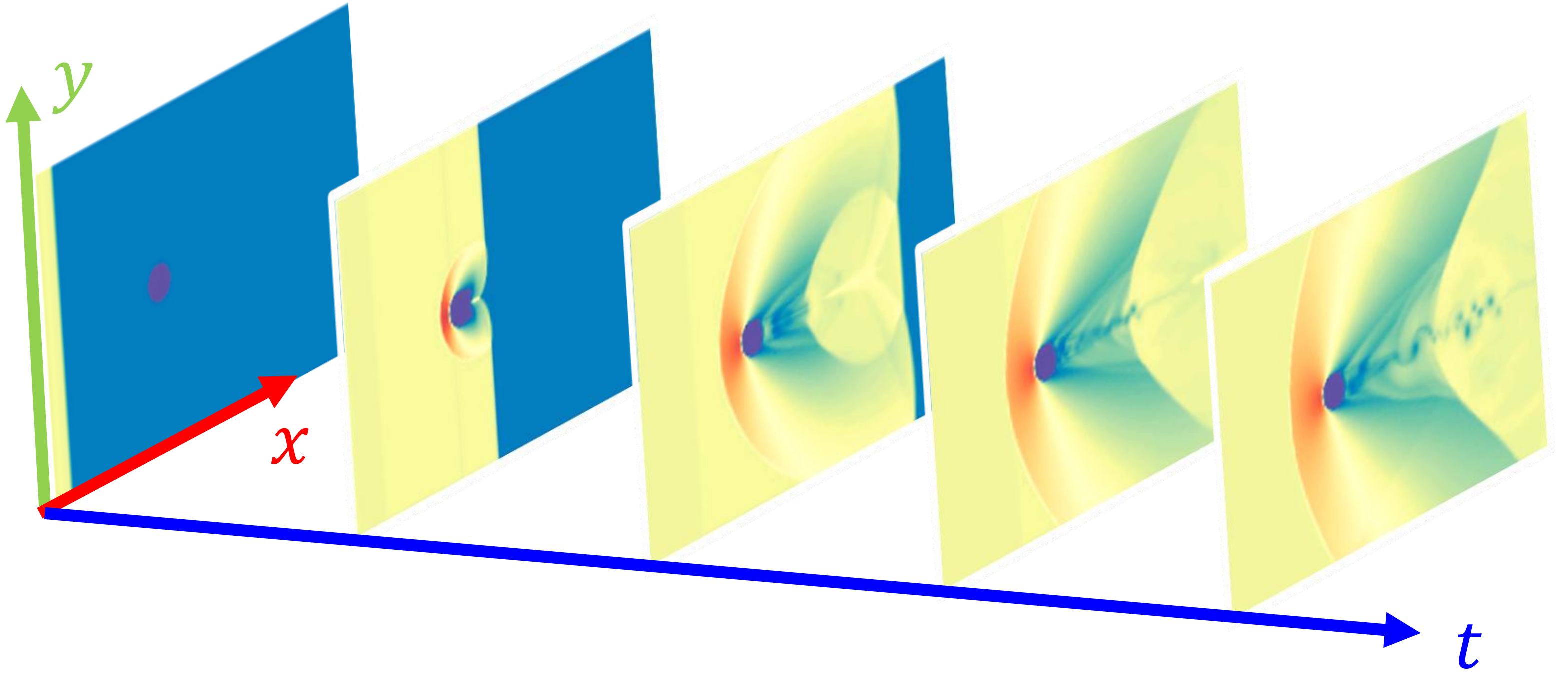
Neural Operators

Stephen Baek

Machine Learning on Function Spaces

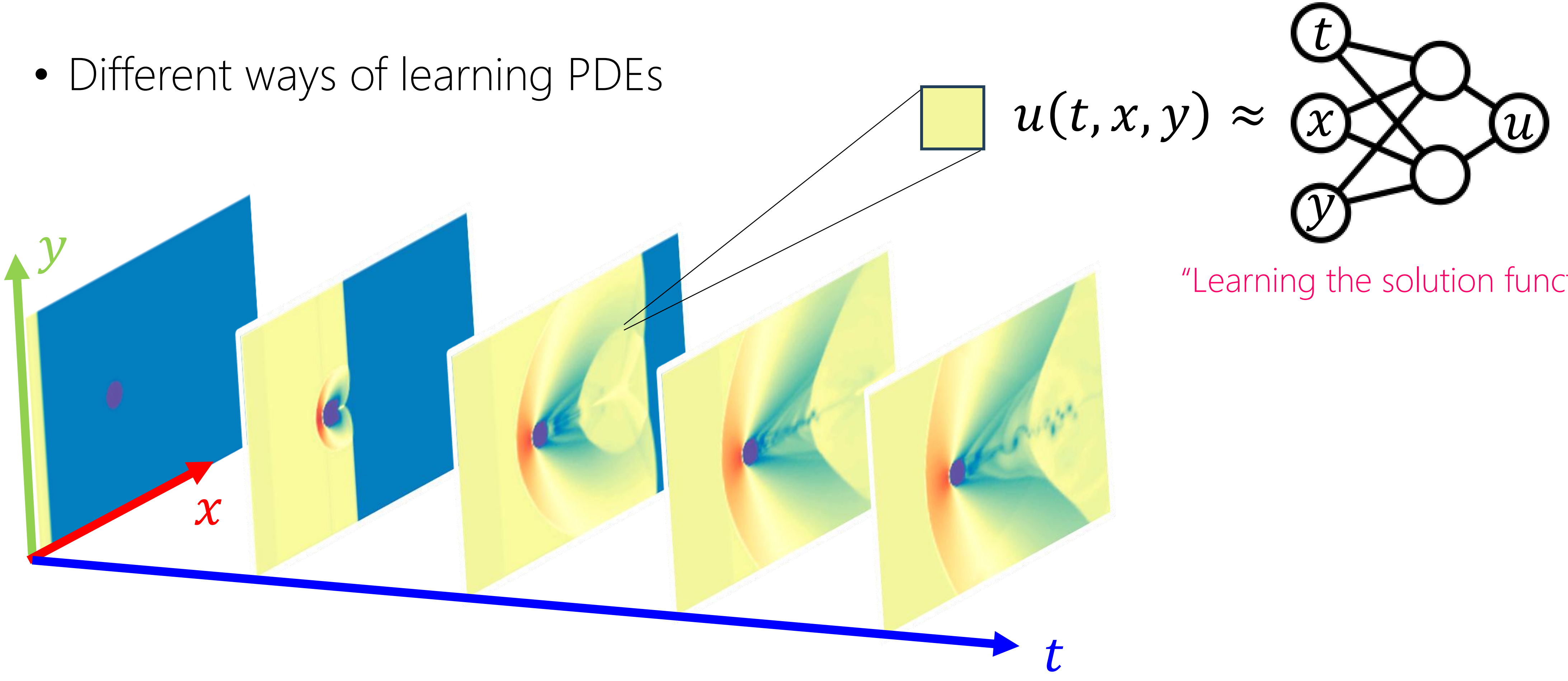
- Different ways of learning PDEs



Flow snapshots: Cheng et al. (2024). IJMF

Machine Learning on Function Spaces

- Different ways of learning PDEs

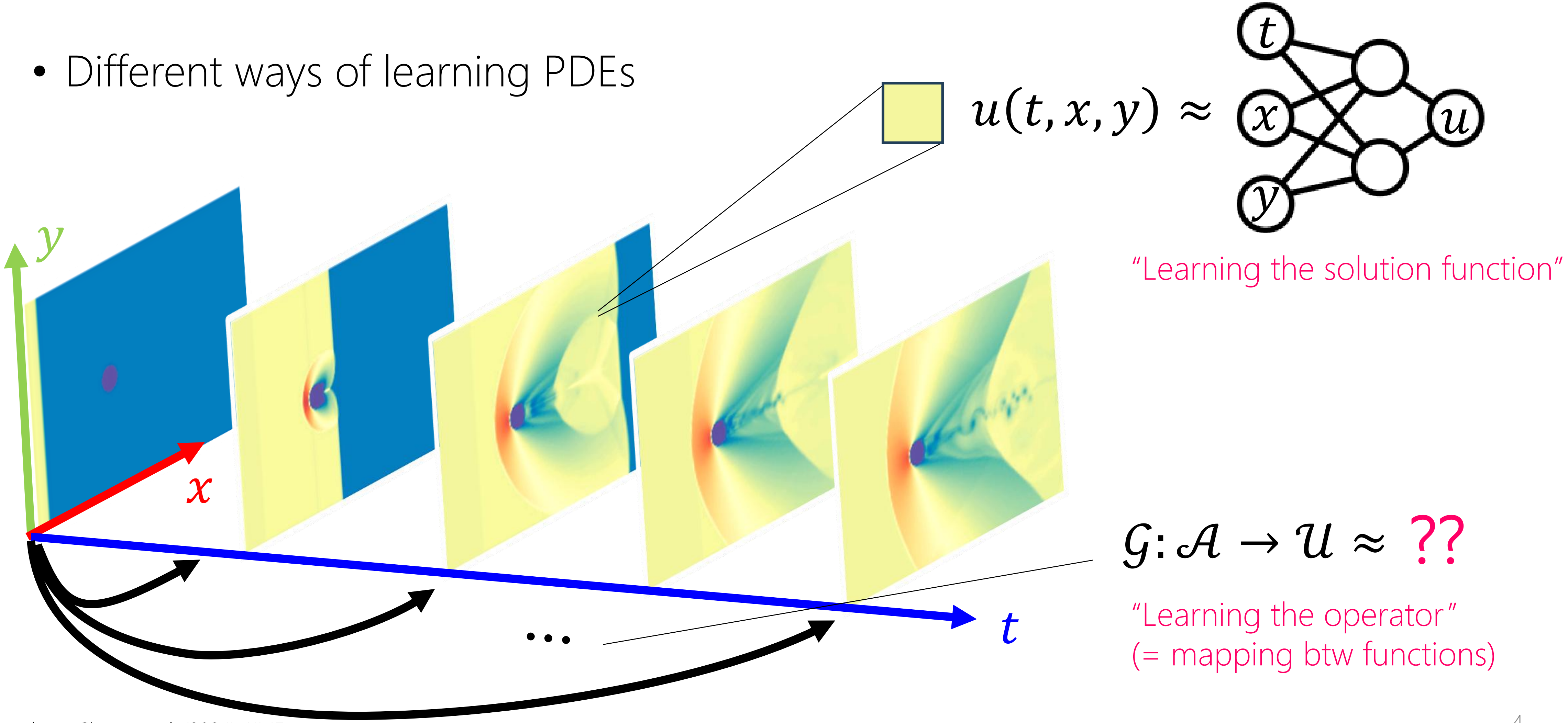


"Learning the solution function"

Flow snapshots: Cheng et al. (2024). IJMF

Machine Learning on Function Spaces

- Different ways of learning PDEs



Flow snapshots: Cheng et al. (2024). IJMF

Recap on Terminology

- Function
  - Mapping from value to value.
  - "The usual ones"
  - For instance,
    - $f(x) = \sin(x)$ maps a scalar to a scalar ($\sin: \mathbb{R} \longrightarrow \mathbb{R}$)
    - Linear function $\mathbf{y} = A\mathbf{x} + b$ maps a vector $\mathbf{x} \in \mathbb{R}^n$ to a vector $\mathbf{y} \in \mathbb{R}^m$
    - The solution function $u(t, x, y)$ of a PDE maps a given time t and location x, y to a physical quantity u.

Recap on Terminology

- Function
  - Mapping from value to value
  - “The usual ones”
  - For instance,
    - $f(x) = \sin(x)$ maps a scalar to a scalar ($\sin: \mathbb{R} \longrightarrow \mathbb{R}$)
    - Linear function $\mathbf{y} = A\mathbf{x} + b$ maps a vector $\mathbf{x} \in \mathbb{R}^n$ to a vector $\mathbf{y} \in \mathbb{R}^m$
    - The solution function $u(t, x, y)$ of a PDE maps a given time t and location x, y to a physical quantity u.

- Operator
  - Mapping from function to function.
  - May not be familiar with the term, but you may already know it.
  - For instance,
    - Differential operator d/dx maps the function $f(x) = 3x^2 + 2$ to the function $f'(x) = 6x$.
    - Integral operator $\int$ maps the function $\sin(x)$ to the function $-\cos(x) + C,\ C \in \mathbb{R}$.
    - Convolution operator T maps a function $f(x)$ to $(Tf)(x) = \int_{-\infty}^{\infty} f(y) g(x - y) dy$.

Ex: Solution Operator of ODE

- Second order systems, e.g., homogeneous spring-mass-damper system:

$$m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0$$

Ex: Solution Operator of ODE

- Second order systems, e.g., homogeneous spring-mass-damper system:

$$m\ddot{x}(t) + c\dot{x}(t) + kx(t) = \mathbf{0}$$

- Convert to first-order system:

$$\frac{d}{dt}\mathbf{u}(t) = \mathbf{A}\mathbf{u}(t)$$

  - … where: $\mathbf{u}(t) = \begin{bmatrix} x(t) \\ \dot{x}(t) \end{bmatrix}$, $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix}$

Ex: Solution Operator of ODE

- Second order systems, e.g., homogeneous spring-mass-damper system:

$$m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0$$

- Convert to first-order system:

$$\frac{d}{dt}\mathbf{u}(t) = \mathbf{A}\mathbf{u}(t)$$

  - … where: $\mathbf{u}(t) = \begin{bmatrix} x(t) \\ \dot{x}(t) \end{bmatrix}$, $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix}$

- Solution to IVP: $\mathbf{u}(t) = e^{t\mathbf{A}}\mathbf{u}(0)$

Ex: Solution Operator of ODE

- Second order systems, e.g., homogeneous spring-mass-damper system:

$$m\ddot{x}(t) + c\dot{x}(t) + kx(t) = 0$$

- Convert to first-order system:

$$\frac{d}{dt}\mathbf{u}(t) = \mathbf{A}\mathbf{u}(t)$$

  - … where: $\mathbf{u}(t) = \begin{bmatrix} x(t) \\ \dot{x}(t) \end{bmatrix}$, $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix}$

- Solution to IVP: $\mathbf{u}(t) = \boxed{e^{t\mathbf{A}}}\mathbf{u}(0)$

Solution operator

Formal Definition

- Generic family of PDEs:

$$
\begin{aligned}
(\mathrm{L}_a u)(x) &= f(x), \qquad && x \in D, \\
u(x) &= 0, && x \in \partial D
\end{aligned}
$$

  - $a \in \mathcal{A}$ where $\mathcal{A}$ is the Banach space* of physics parameters
  - $u: D \to \mathbb{R}$: solution function that lives in the Banach space $\mathcal{U}$
  - $f: D \to \mathbb{R}$: source term in the dual $\mathcal{U}^*$ of $\mathcal{U}$.
  - $D \subset \mathbb{R}^d$: a bounded physics domain
  - $\mathrm{L}_a: \mathcal{A} \to \mathcal{L}(\mathcal{U}; \mathcal{U}^*)$: mapping from the parameter Banach space $\mathcal{A}$ to the space of linear operators that map $\mathcal{U}$ to its dual $\mathcal{U}^*$

***Banach Space**

= complete normed vector space

We have a way to measure the "length" and "distance" of arrows.

If you have a sequence of arrows $v_1, v_2, \cdots$ that get closer and closer to some limit $v = \lim v_i$, the limit v is also an arrow in the same vector space.

**See Kovachki et al. (2021) for more details.

Formal Definition

- Generic family of PDEs:

$$(\mathrm{L}_a u)(x) = f(x), \qquad x \in D,$$
$$u(x) = 0, \qquad x \in \partial D$$

  - $a \in \mathcal{A}$ where $\mathcal{A}$ is the Banach space* of physics parameters
  - $u: D \to \mathbb{R}$: solution function that lives in the Banach space $\mathcal{U}$
  - $f: D \to \mathbb{R}$: source term in the dual $\mathcal{U}^*$ of $\mathcal{U}$.
  - $D \subset \mathbb{R}^d$: a bounded physics domain
  - $\mathrm{L}_a: \mathcal{A} \to \mathcal{L}(\mathcal{U}; \mathcal{U}^*)$: mapping from the parameter Banach space $\mathcal{A}$ to the space of linear operators that map $\mathcal{U}$ to its dual $\mathcal{U}^*$
- Solution operator to this PDE $\mathcal{G}: \mathcal{A} \times \mathcal{U}^* \to \mathcal{U}$

***Banach Space**

= complete normed vector space

We have a way to measure the "length" and "distance" of arrows.

If you have a sequence of arrows $v_1, v_2, \cdots$ that get closer and closer to some limit $v = \lim v_i$, the limit v is also an arrow in the same vector space.

**See Kovachki et al. (2021) for more details.

Formal Definition

- Generic family of PDEs:

$$
\begin{aligned}
(\mathrm{L}_a u)(x) &= f(x), && x \in D, \\
u(x) &= 0, && x \in \partial D
\end{aligned}
$$

  - $a \in \mathcal{A}$ where $\mathcal{A}$ is the Banach space* of physics parameters
  - $u: D \to \mathbb{R}$: solution function that lives in the Banach space $\mathcal{U}$
  - $f: D \to \mathbb{R}$: source term in the dual $\mathcal{U}^*$ of $\mathcal{U}$.
  - $D \subset \mathbb{R}^d$: a bounded physics domain
  - $\mathrm{L}_a: \mathcal{A} \to \mathcal{L}(\mathcal{U}; \mathcal{U}^*)$: mapping from the parameter Banach space $\mathcal{A}$ to the space of linear operators that map $\mathcal{U}$ to its dual $\mathcal{U}^*$
- Solution operator to this PDE $\mathcal{G}: \mathcal{A} \times \mathcal{U}^* \to \mathcal{U}$
- Parametrize $\mathcal{G}$ with a neural network! i.e. $\mathcal{G} \approx \mathcal{G}_\theta$
  (universal approximation theorem of operators says you can do it**)

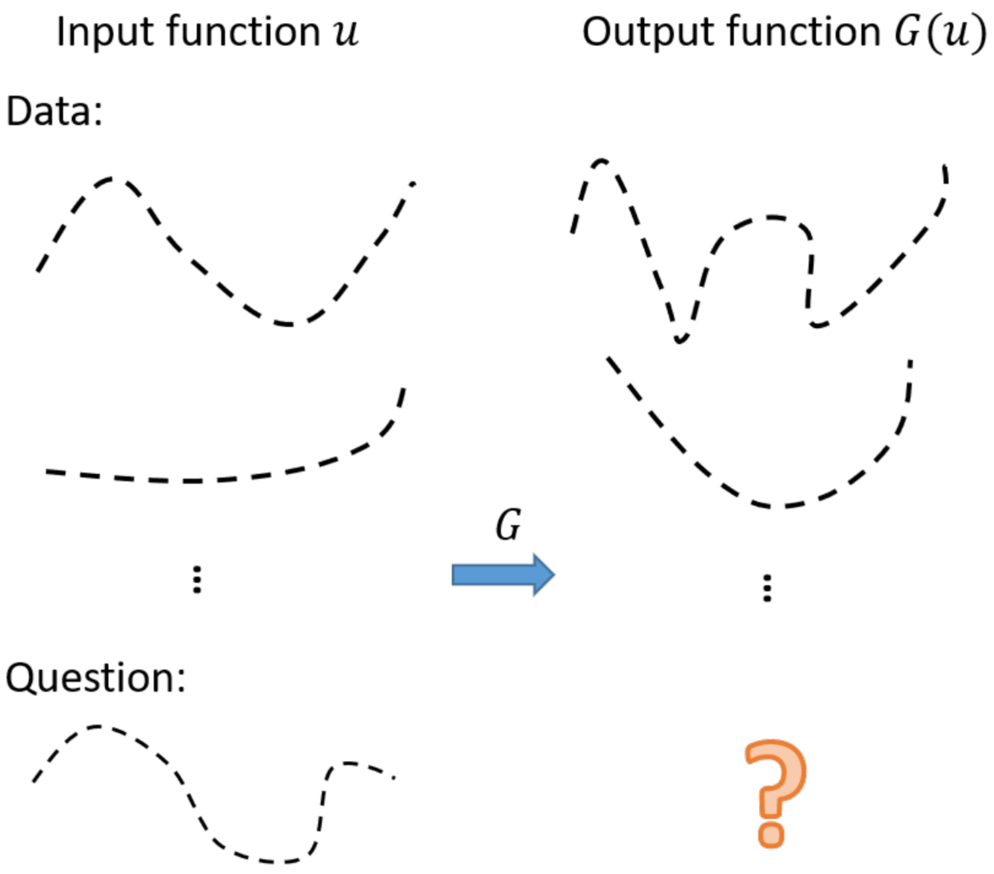
***Banach Space**
= complete normed vector space

We have a way to measure the "length" and "distance" of arrows.

If you have a sequence of arrows $v_1, v_2, \cdots$ that get closer and closer to some limit $v = \lim v_i$, the limit v is also an arrow in the same vector space.

**See Kovachki et al. (2021) for more details.

Learning Solution Operators



Universal Approximation Theorem for Operators

- (Chen & Chen, 1995) Suppose that σ is a continuous non-polynomial function, X is a Banach space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, $\mathcal{G}$ is a continuous operator, which maps V into $C(K_2)$.

  Then for any $\epsilon > 0$, there are positive integers n, p, m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \dots, n$, $k = 1, \dots, p$, $j = 1, \dots, m$, such that

$$\left| \mathcal{G}(u)(y) - \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right) \sigma(w_k \cdot y + \zeta_k) \right| < \epsilon$$

  holds for all $u \in V$ and $y \in K_2$.

Chen & Chen. (1995). Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and Its Application to Dynamical Systems. IEEE Trans. Neural Networks, 6(4):911-917

Universal Approximation Theorem for Operators

- (Chen & Chen, 1995) Suppose that σ is an activation function, X is a Banach space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, $\mathcal{G}$ is a continuous operator, which maps V into $C(K_2)$.

  Then for any $\epsilon > 0$, there are positive integers n, p, m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \dots, n$, $k = 1, \dots, p$, $j = 1, \dots, m$,
  such that

$$\left| \mathcal{G}(u)(y) - \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right) \sigma(w_k \cdot y + \zeta_k) \right| < \epsilon$$

  holds for all $u \in V$ and $y \in K_2$.

Chen & Chen. (1995). Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and Its Application to Dynamical Systems. IEEE Trans. Neural Networks, 6(4):911-917

Universal Approximation Theorem for Operators

- (Chen & Chen, 1995) Suppose that $\boldsymbol{\sigma}$ is an activation function, K_1 is an input physical domain (where the input physical quantities $\boldsymbol{u}$ are sampled), K_2 is an output physical domain (where the output physical quantities are sampled), V is a compact set in $C(K_1)$, $\mathcal{G}$ is a continuous operator, which maps V into $C(K_2)$.

  Then for any $\epsilon > 0$, there are positive integers n, p, m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \dots, n$, $k = 1, \dots, p$, $j = 1, \dots, m$,
  such that

$$\left| \mathcal{G}(u)(y) - \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right) \sigma(w_k \cdot y + \zeta_k) \right| < \epsilon$$

  holds for all $u \in V$ and $y \in K_2$.

Chen & Chen. (1995). Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and Its Application to Dynamical Systems. IEEE Trans. Neural Networks, 6(4):911-917

Universal Approximation Theorem for Operators

- (Chen & Chen, 1995) Suppose that σ is an activation function, K_1 is an input physical domain (where the input physical quantities u are sampled), K_2 is an output physical domain (where the output physical quantities are sampled), V is a set of continuous functions (physical quantities u) defined on K_1, $\mathcal{G}$ is a continuous operator, which maps V into $C(K_2)$.

  Then for any $\epsilon > 0$, there are positive integers n, p, m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \dots, n$, $k = 1, \dots, p$, $j = 1, \dots, m$, such that

$$\left| \mathcal{G}(u)(y) - \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right) \sigma(w_k \cdot y + \zeta_k) \right| < \epsilon$$

  holds for all $u \in V$ and $y \in K_2$.

Chen & Chen. (1995). Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and Its Application to Dynamical Systems. IEEE Trans. Neural Networks, 6(4):911-917

Universal Approximation Theorem for Operators

- (Chen & Chen, 1995) Suppose that $\boldsymbol{\sigma}$ is an activation function, $\boldsymbol{K_1}$ is an input physical domain (where the input physical quantities $\boldsymbol{u}$ are sampled), $\boldsymbol{K_2}$ is an output physical domain (where the output physical quantities are sampled), $\boldsymbol{V}$ is a set of continuous functions (physical quantities $\boldsymbol{u}$) defined on K_1, $\mathcal{G}$ is a solution operator, which maps input physical quantities $\boldsymbol{u}$ to output physical quantities evaluated at $\boldsymbol{y}$.

  Then for any $\boldsymbol{\epsilon > 0}$, there are positive integers $\boldsymbol{n}$, $\boldsymbol{p}$, $\boldsymbol{m}$, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \dots, n$, $k = 1, \dots, p$, $j = 1, \dots, m$,
  such that

$$\left| \mathcal{G}(u)(y) - \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right) \sigma(w_k \cdot y + \zeta_k) \right| < \epsilon$$

  holds for all $u \in V$ and $y \in K_2$.

Chen & Chen. (1995). Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and Its Application to Dynamical Systems. IEEE Trans. Neural Networks, 6(4):911-917

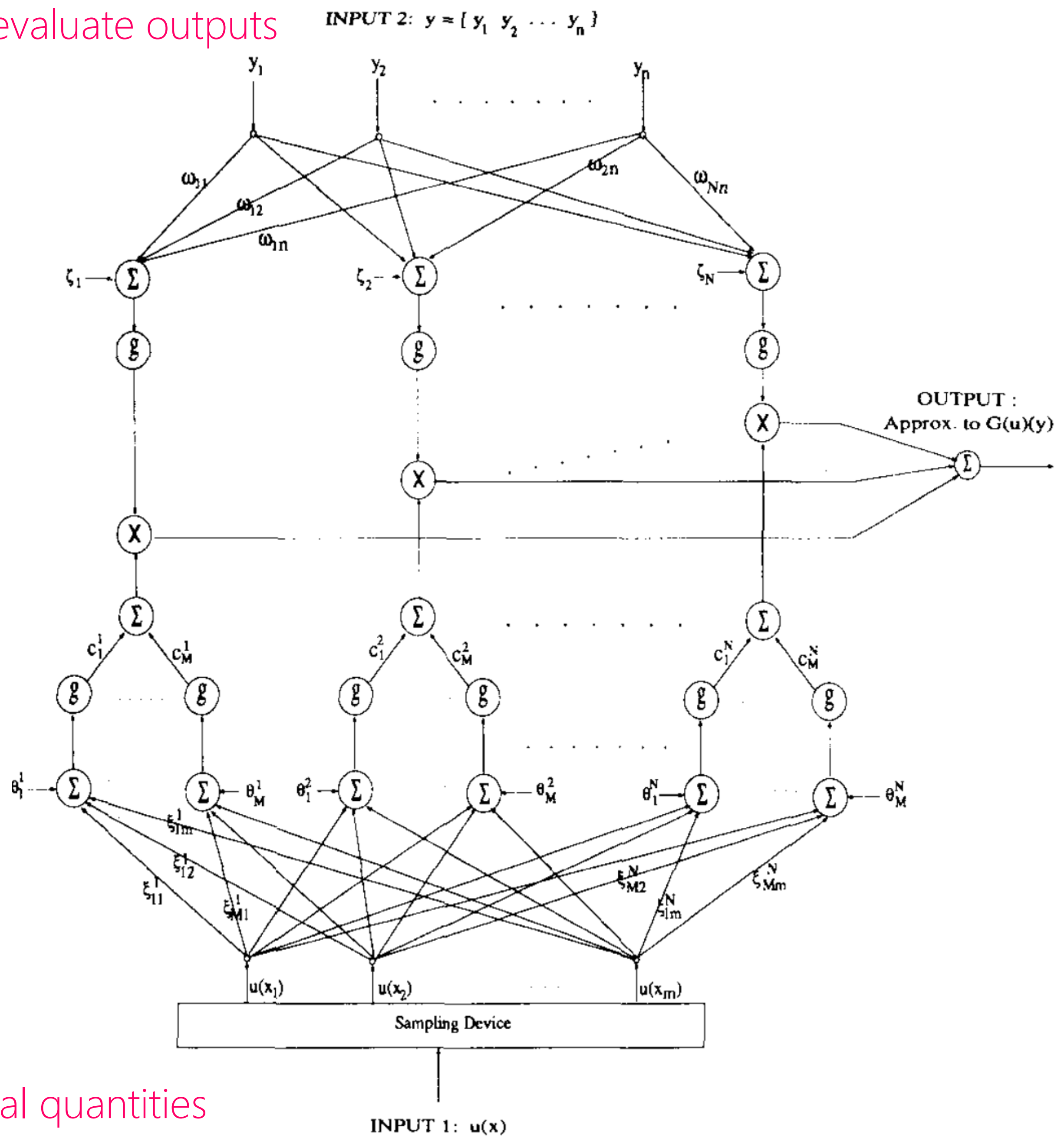
Universal Approximation Theorem for Operators

- (Chen & Chen, 1995) Suppose that σ is an activation function, K_1 is an input physical domain (where the input physical quantities u are sampled), K_2 is an output physical domain (where the output physical quantities are sampled), V is a set of continuous functions (physical quantities u) defined on K_1, $\mathcal{G}$ is a solution operator, which maps input physical quantities u to output physical quantities evaluated at y.

  Then, there exists a neural network that looks like the one in the next slide that can approximate $\mathcal{G}$ with an error bound ϵ.

Chen & Chen. (1995). Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and Its Application to Dynamical Systems. IEEE Trans. Neural Networks, 6(4):911-917

Locations where you want to evaluate outputs



Output physical quantities

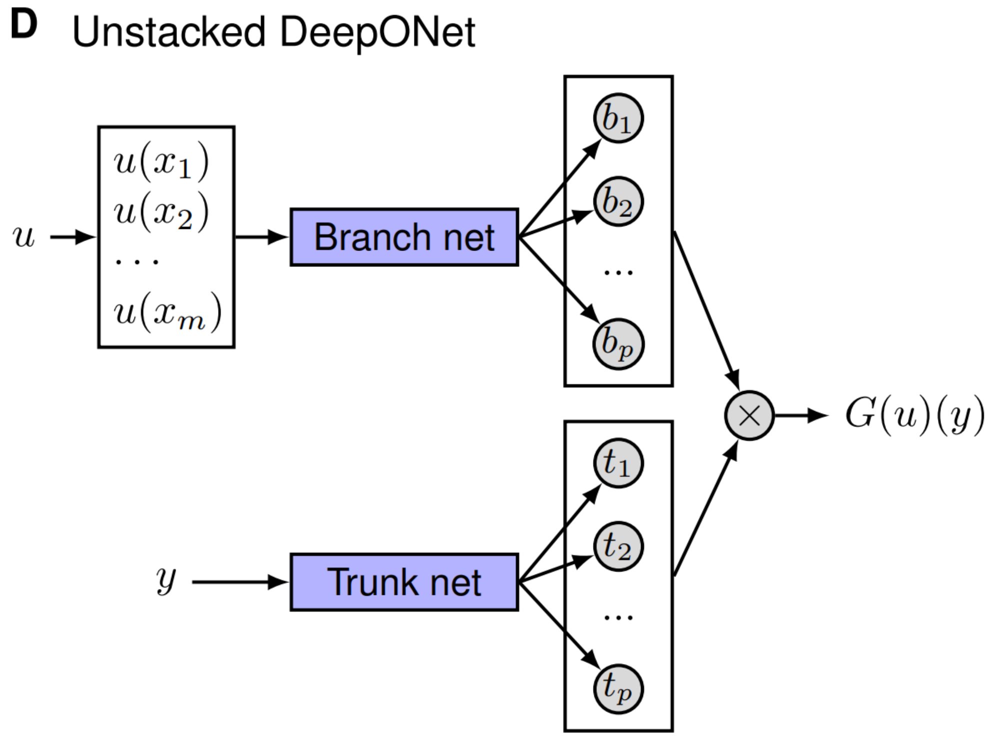
Input physical quantities

Chen & Chen. (1995). Universal Approximation to Nonlinear Operators by Neural Networks with Arbitrary Activation Functions and Its Application to Dynamical Systems. IEEE Trans. Neural Networks, 6(4):911-917

A modern twist: Deep-O-Net

Lu et al., *Nature Mach Intell*, 2021

$$\mathcal{G}(u)(y) \approx \sum_{k=1}^{p} \underbrace{b_k(u)}_{\text{branch}} \underbrace{t_k(y)}_{\text{trunk}}$$



In Simple Terms: Image-to-Image Problem

- "Operator learning can be taken as an image-to-image problem."
  - Zongyi Li website.
- …where an "Image" could be:
  - A snapshot of the solution function at a given time t.
  - Initial condition/configuration of a physical system. (snapshot at t=0)
  - Boundary geometry.
  - Distribution of system coefficients.
  - Etc.

Image-to-Image Problem

- For instance, a solution operator:
  - Consider the Darcy equation describing the flow of a fluid through a porous medium:
    $$-\mathrm{div}(a\nabla u) = f$$
  - …, where
    - $u \in \mathcal{U}$: temperature or pressure
    - $a \in \mathcal{A}$: conductance or permeability
    - $f \in \mathcal{U}^*$: source term

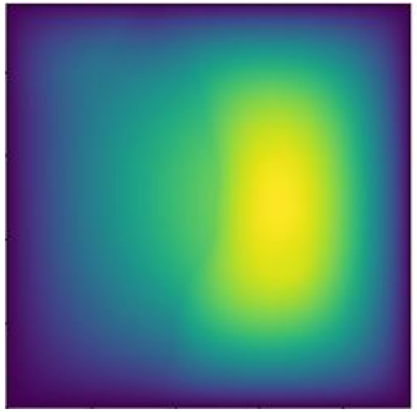
$f = 0$
Assumed in this example

$\times$

a: Permeability coefficients

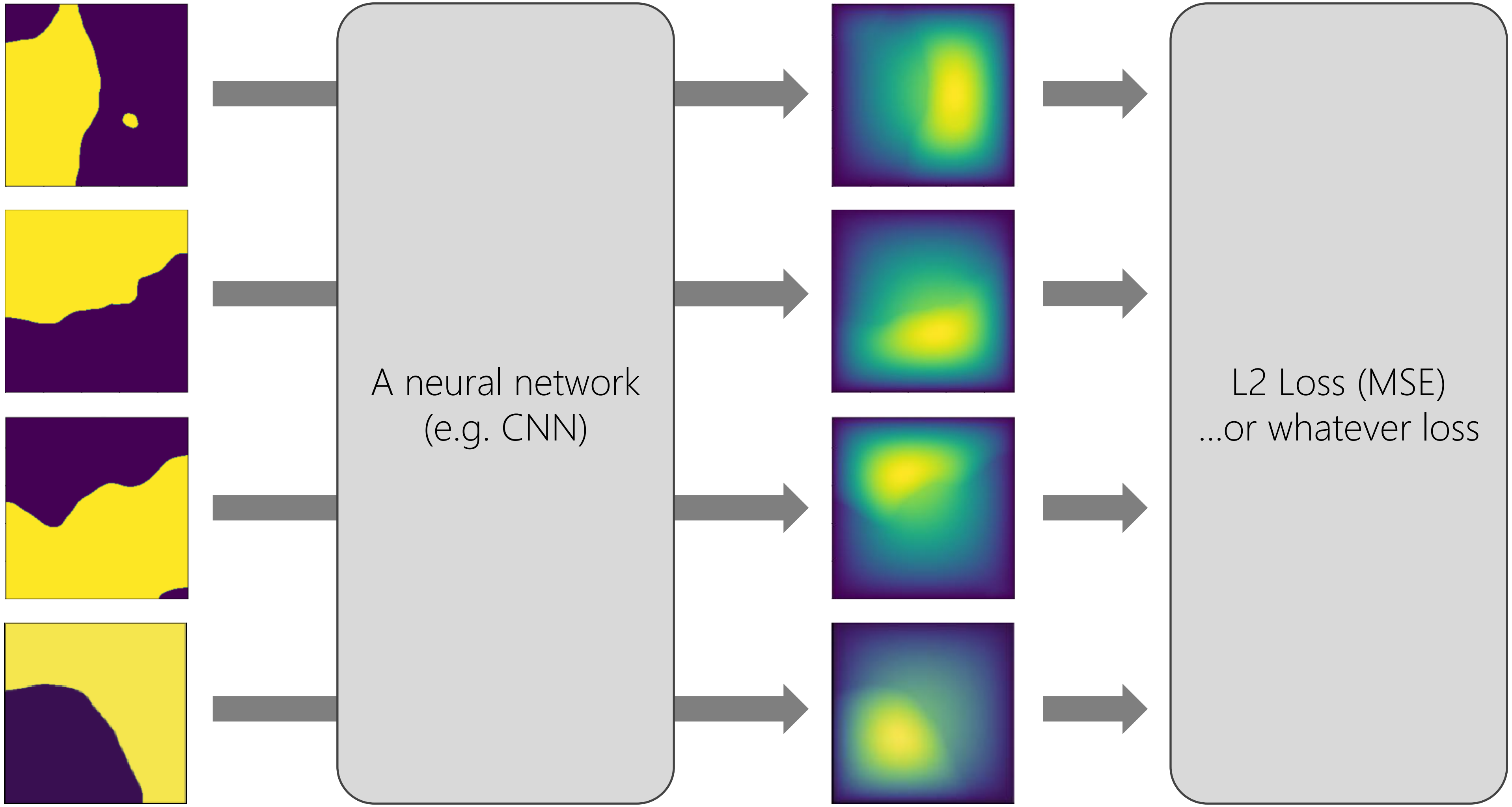
$\mathcal{G}: \mathcal{A} \times \mathcal{U}^* \to \mathcal{U}$

solution operator



u: Pressure distribution

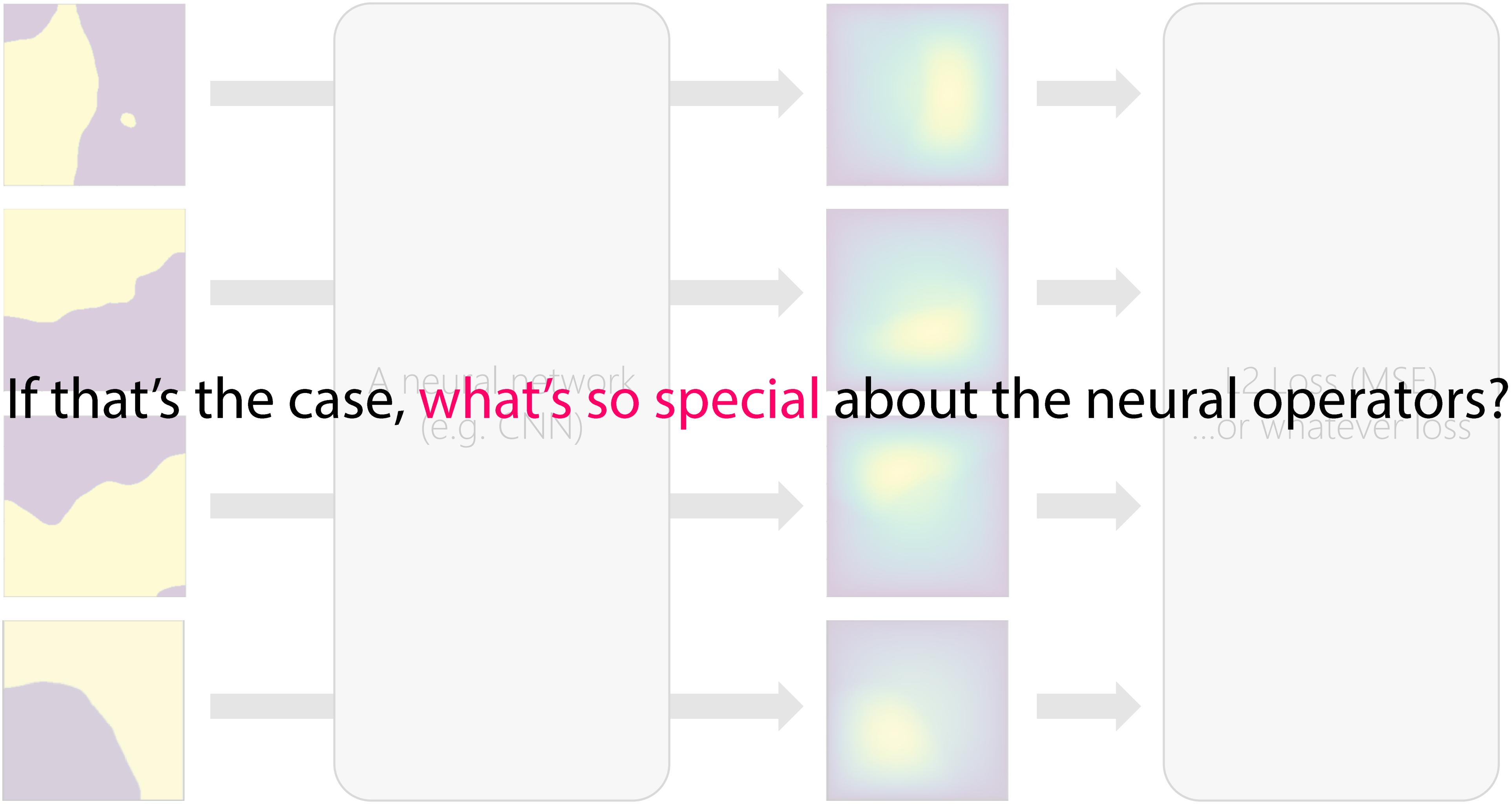
Image-to-Image Problem



Input: Coefficients

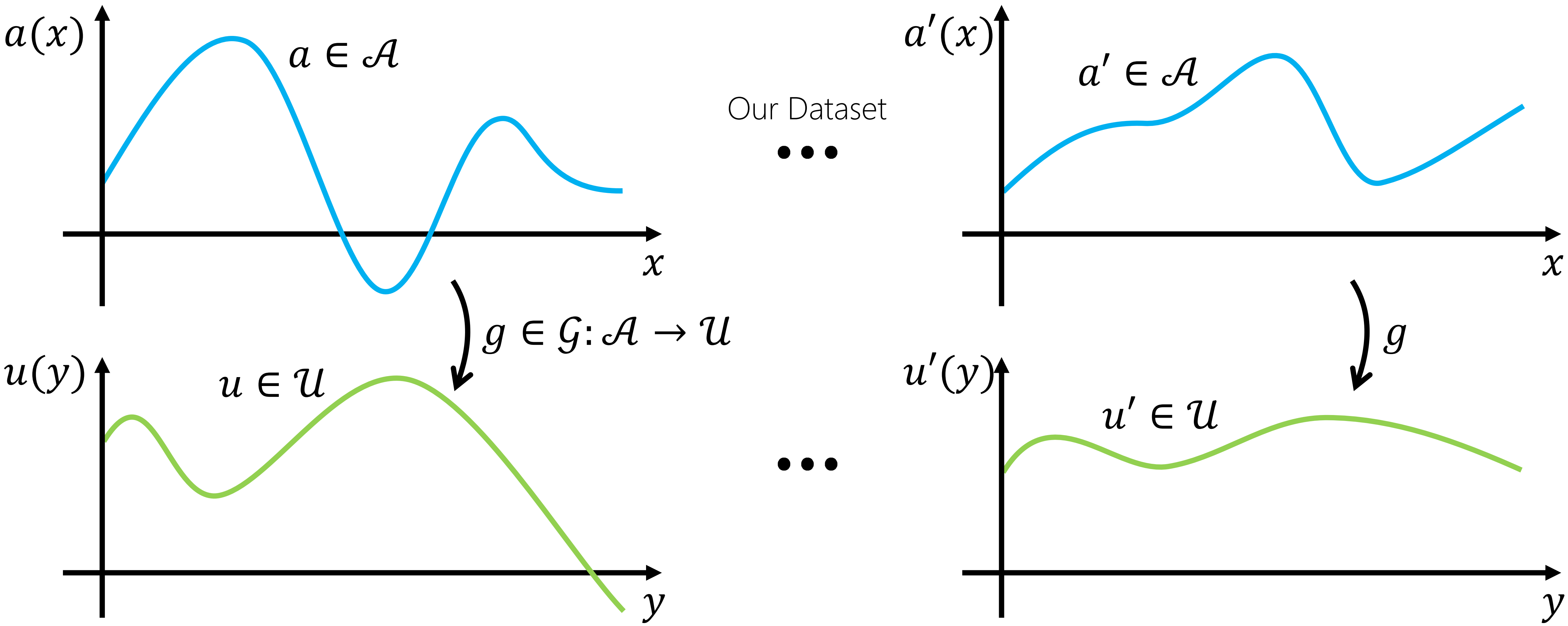
Output: Solutions

Image-to-Image Problem

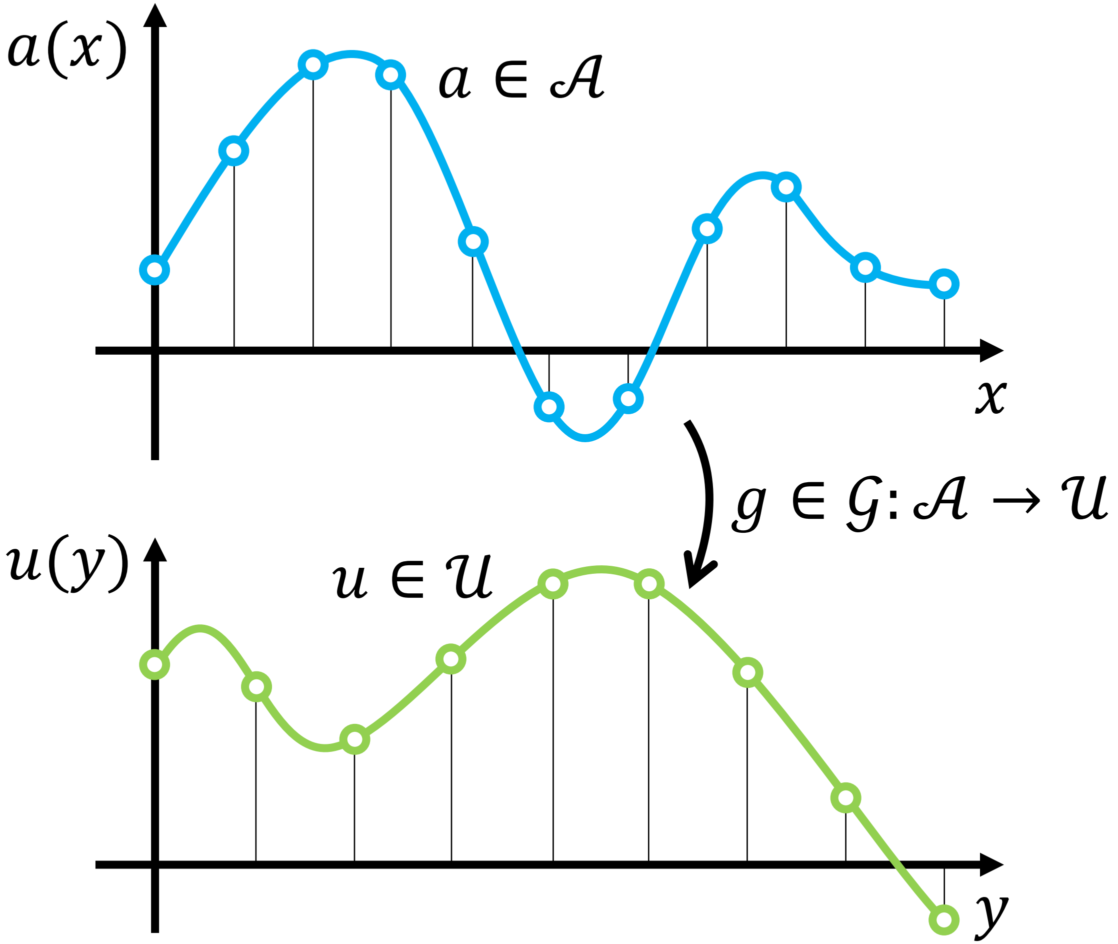


If that’s the case, what’s so special about the neural operators?

Machine Learning on Functions



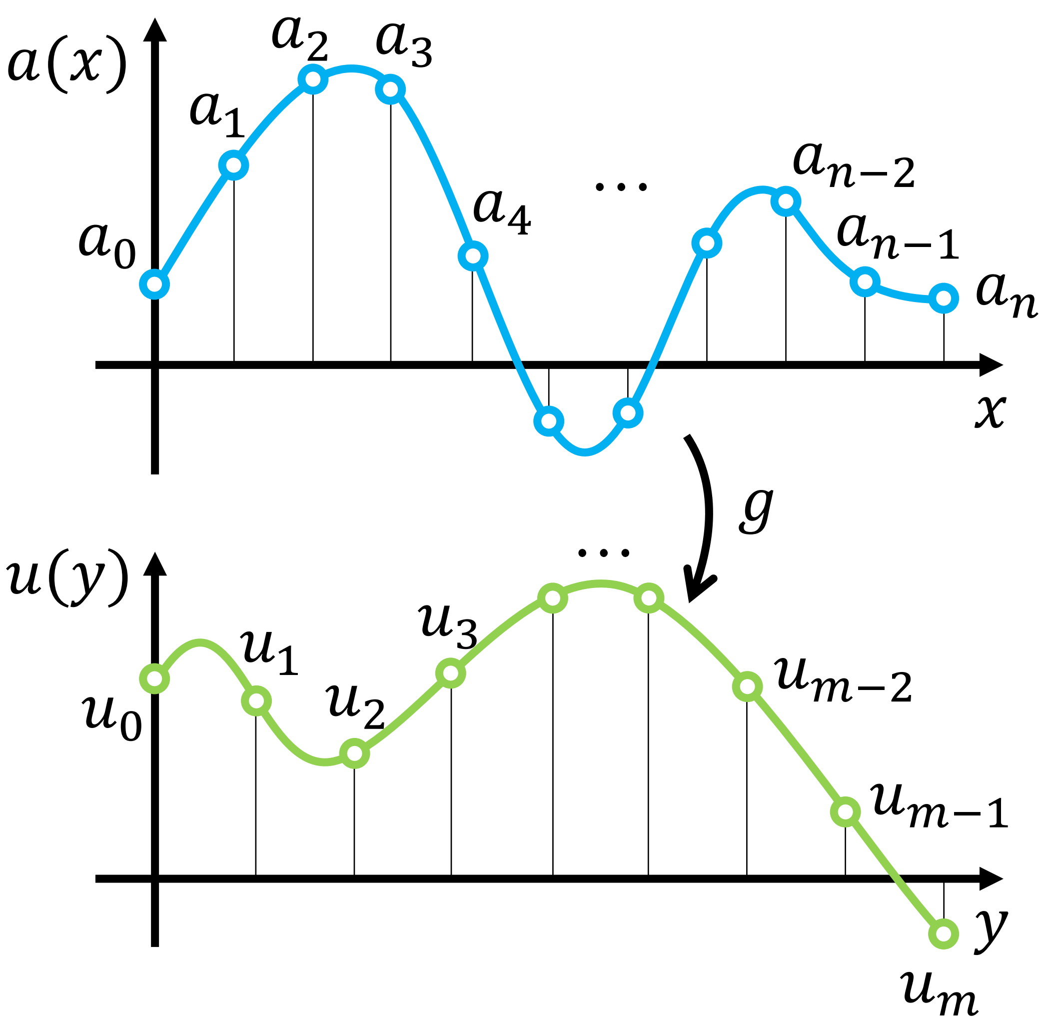
Machine Learning on Functions



Discretization:

$$a_i \coloneqq a(x_i)$$
$$u_i \coloneqq u(y_i)$$

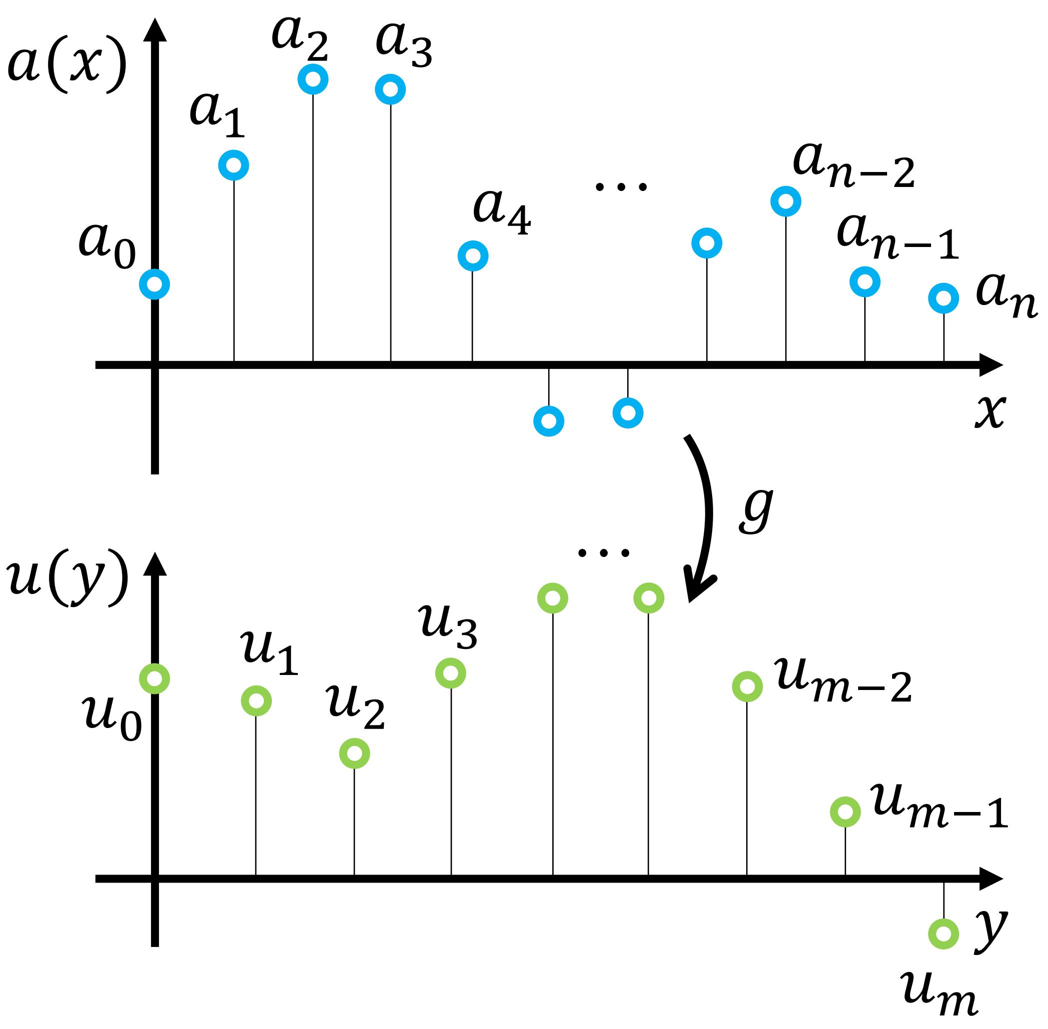
Machine Learning on Functions



Discretization:

$$a_i \coloneqq a(x_i)$$
$$u_i \coloneqq u(y_i)$$

Machine Learning on Functions



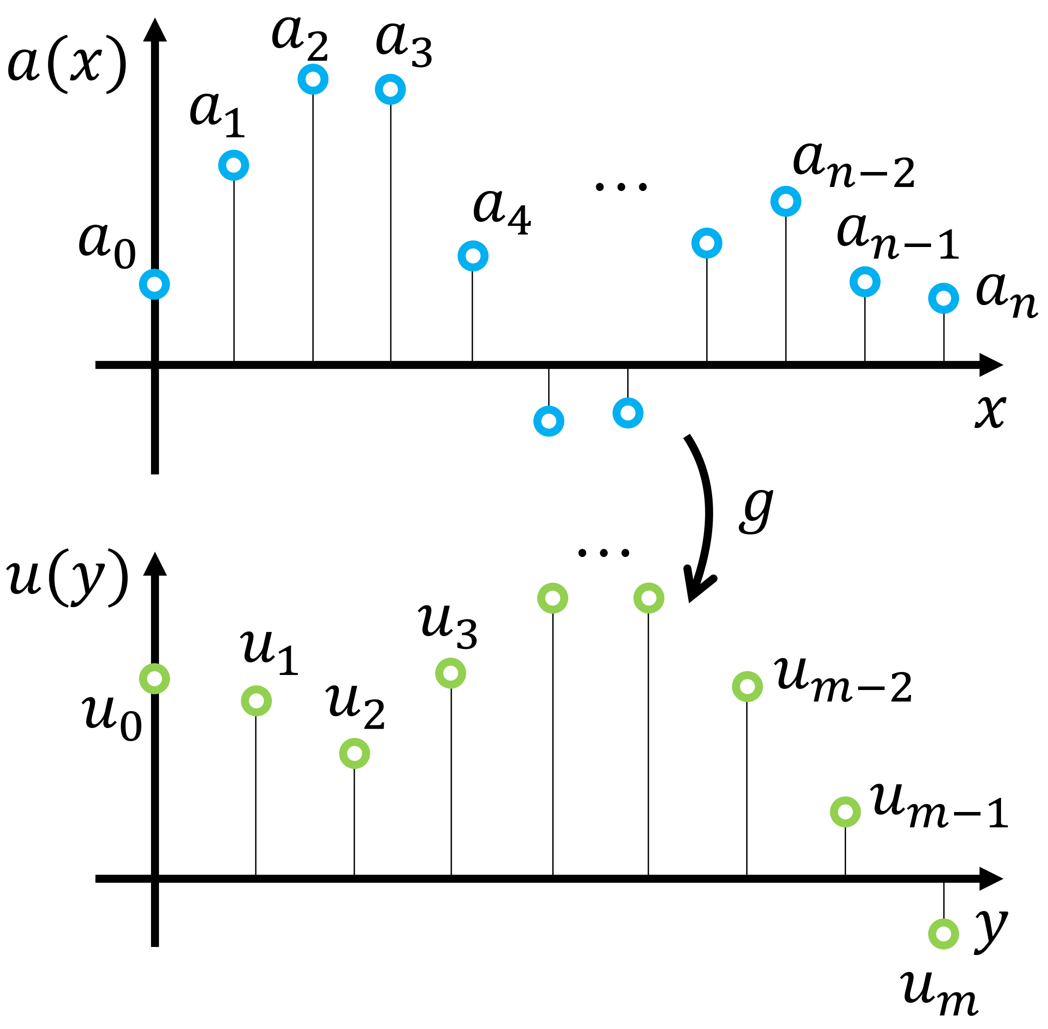
Discretization:

$$a_i \coloneqq a(x_i)$$
$$u_i \coloneqq u(y_i)$$

a_0	a_1	a_2	a_3	a_4	$\cdots$	a_n

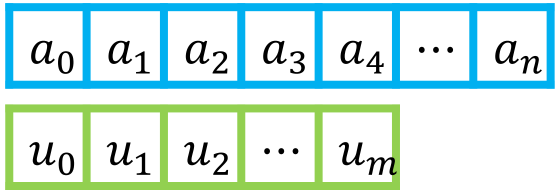
u_0	u_1	u_2	$\cdots$	u_m

Machine Learning on Functions



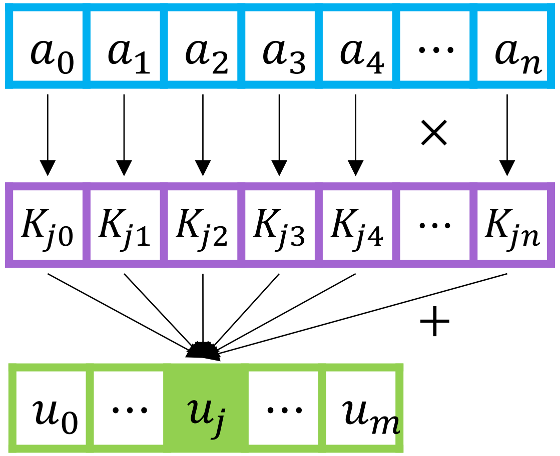
Discretization:

$$a_i := a(x_i)$$
$$u_i := u(y_i)$$



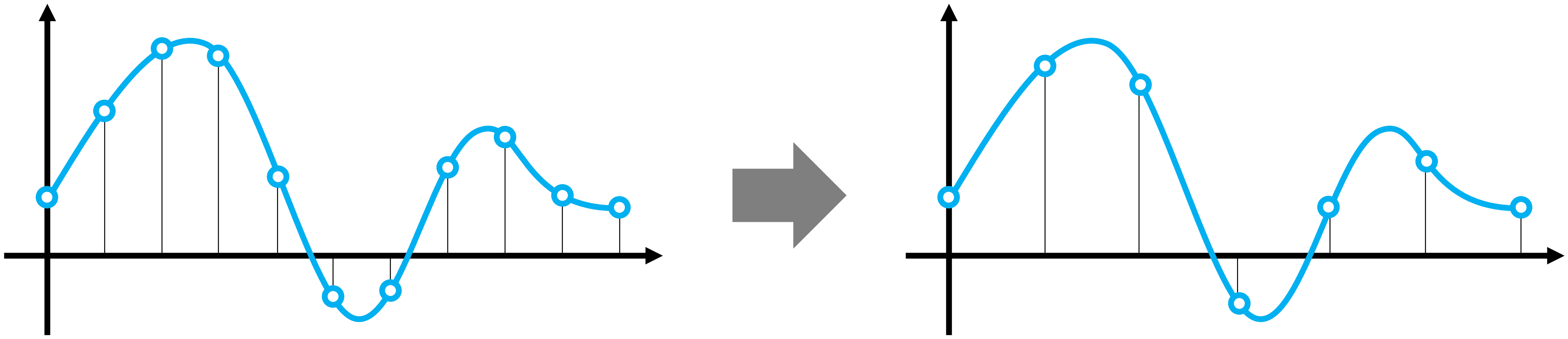
Linear neural network layer

$$u_j = \sigma\left(\sum_{i=0}^{n} K_{ji} a_i\right)$$



Machine Learning on Functions

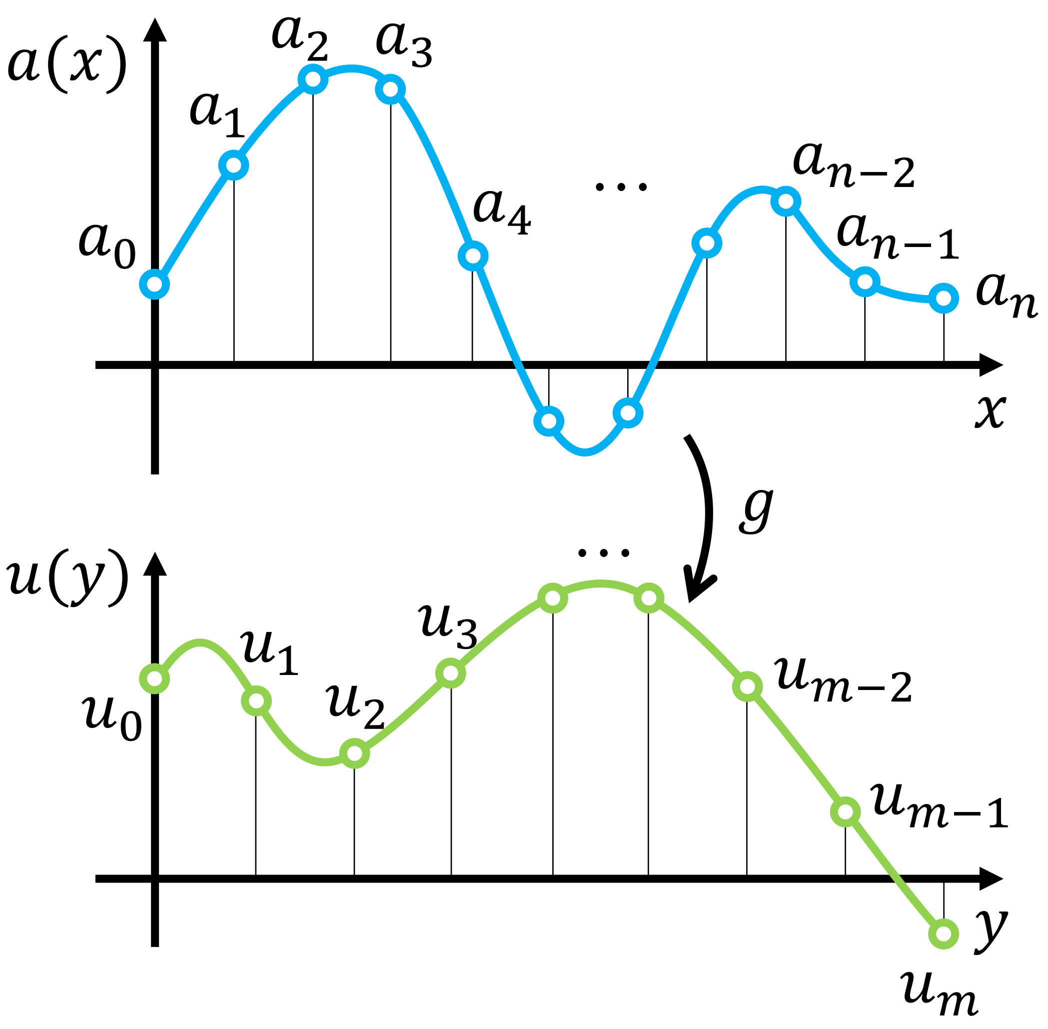
What if the discretization changes?



K_{j0}	K_{j1}	K_{j2}	K_{j3}	K_{j4}	$\cdots$	K_{jn}

→ Are these still useful?

Machine Learning on Functions

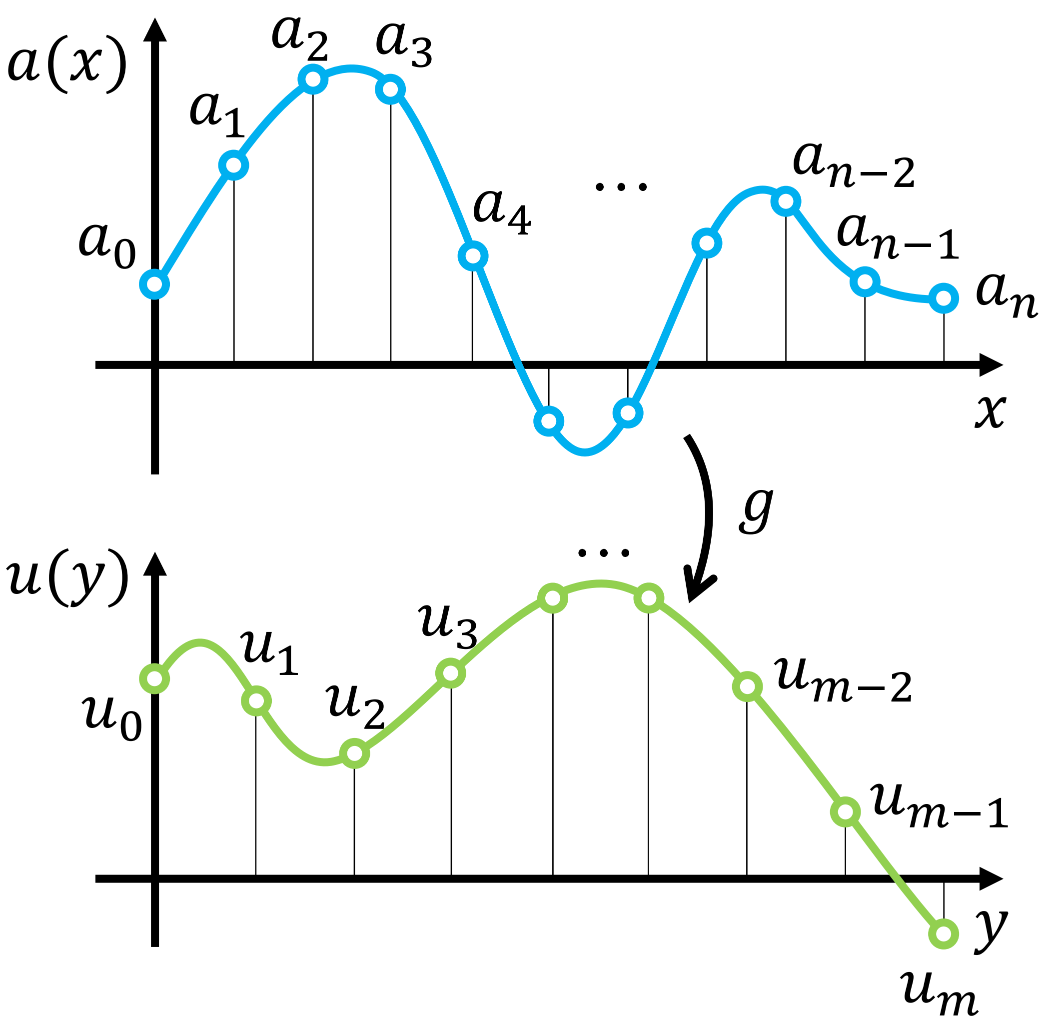


Instead, consider "indexing" the weight K with x_i and y_j

$$u_j = \sigma\left(\sum_{i=0}^{n} K_{ji} a_i\right)$$

$$\Rightarrow \quad u(y_j) = \sigma\left(\sum_{i=0}^{n} K(y_j, x_i) a(x_i)\right)$$

Machine Learning on Functions



Instead, consider "indexing" the weight K with x_i and y_j

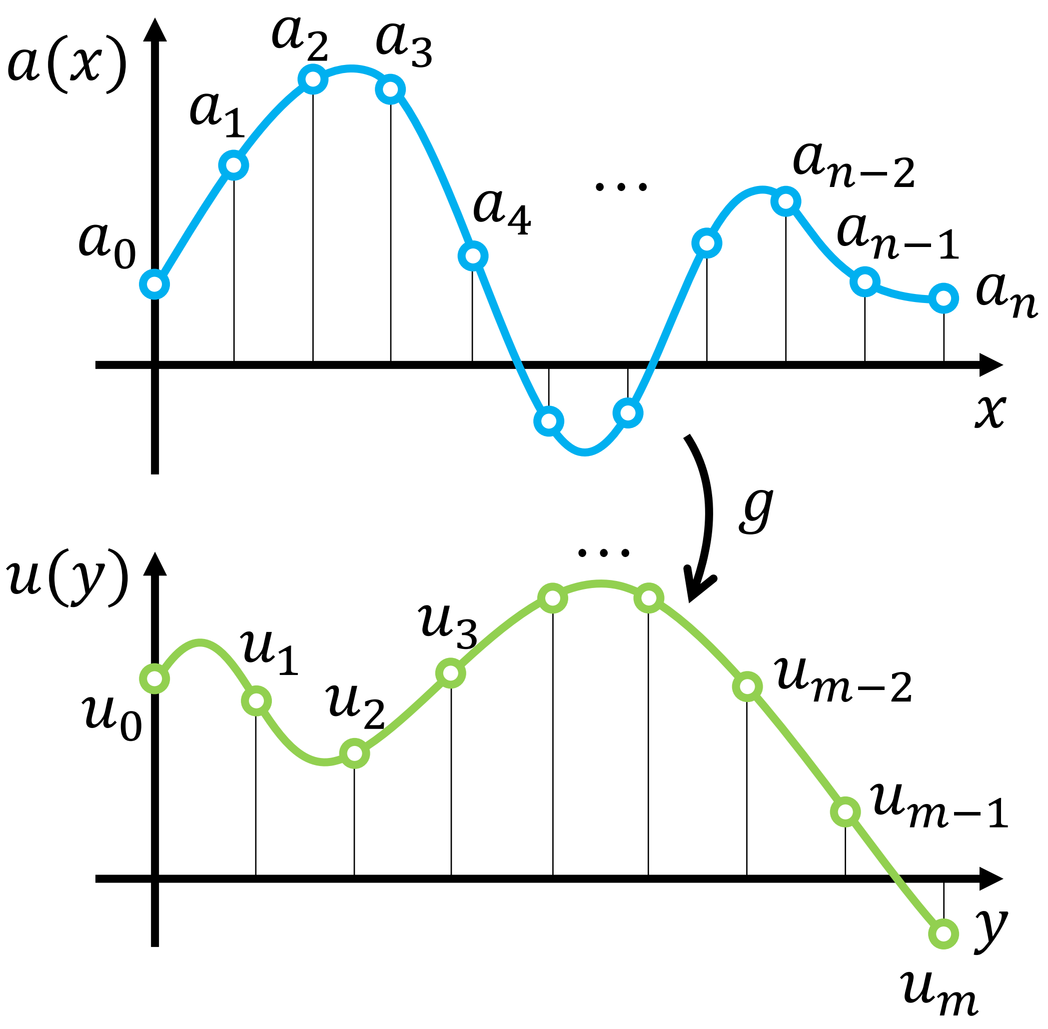
$$u_j = \sigma\left(\sum_{i=0}^{n} K_{ji} a_i\right)$$

$$\Rightarrow \quad u(y_j) = \sigma\left(\sum_{i=0}^{n} K(y_j, x_i) a(x_i)\right)$$

$$= \sigma\left(\sum_{i=0}^{n} \kappa(y_j, x_i) a(x_i) \Delta x_i\right)$$

Riemann Sum

Machine Learning on Functions



Instead, consider "indexing" the weight K with x_i and y_j

$$u_j = \sigma\left(\sum_{i=0}^{n} K_{ji} a_i\right)$$

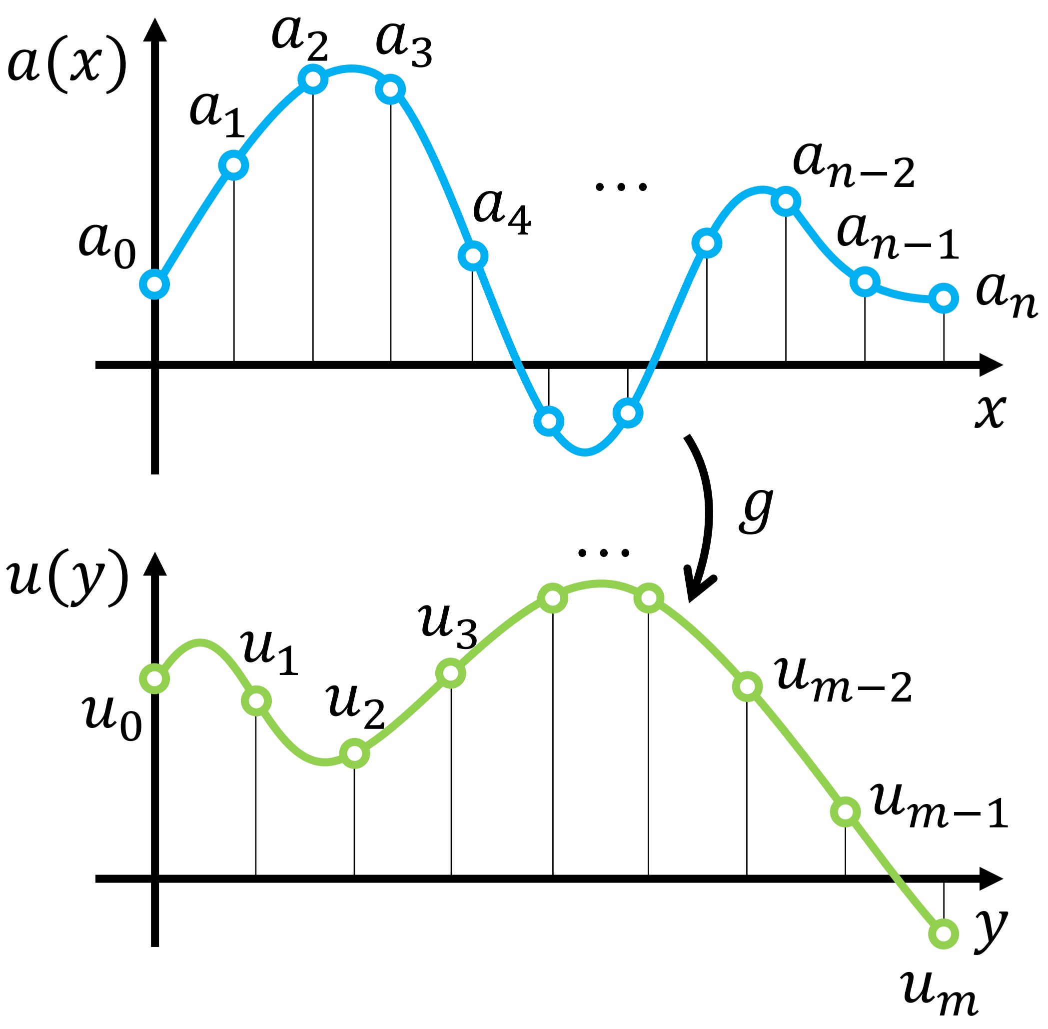
$$u(y_j) = \sigma\left(\sum_{i=0}^{n} K(y_j, x_i) a(x_i)\right)$$

$$= \sigma\left(\sum_{i=0}^{n} \kappa(y_j, x_i) a(x_i) \Delta x_i\right)$$

Riemann Sum

$$u(y) = \sigma\left(\int \kappa(y, x) a(x) dx\right)$$

Machine Learning on Functions



Instead, consider "indexing" the weight K with x_i and y_j

$$u_j = \sigma\left(\sum_{i=0}^{n} K_{ji} a_i\right)$$

Linear matrix operator acting on fixed discretization

$$u(y_j) = \sigma\left(\sum_{i=0}^{n} K(y_j, x_i) a(x_i)\right)$$

$$= \sigma\left(\sum_{i=0}^{n} \kappa(y_j, x_i) a(x_i) \Delta x_i\right)$$

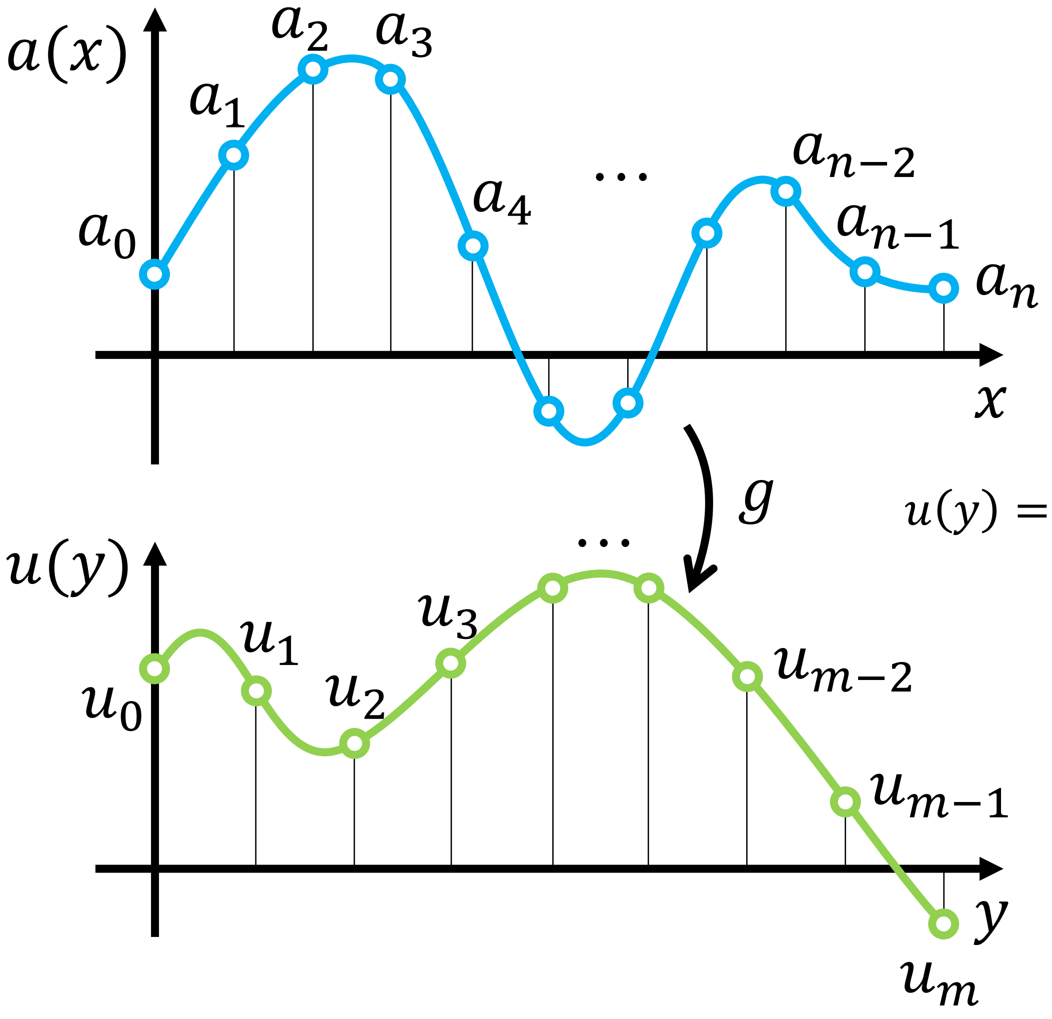
Riemann Sum

Linear integral operator acting on continuous functions

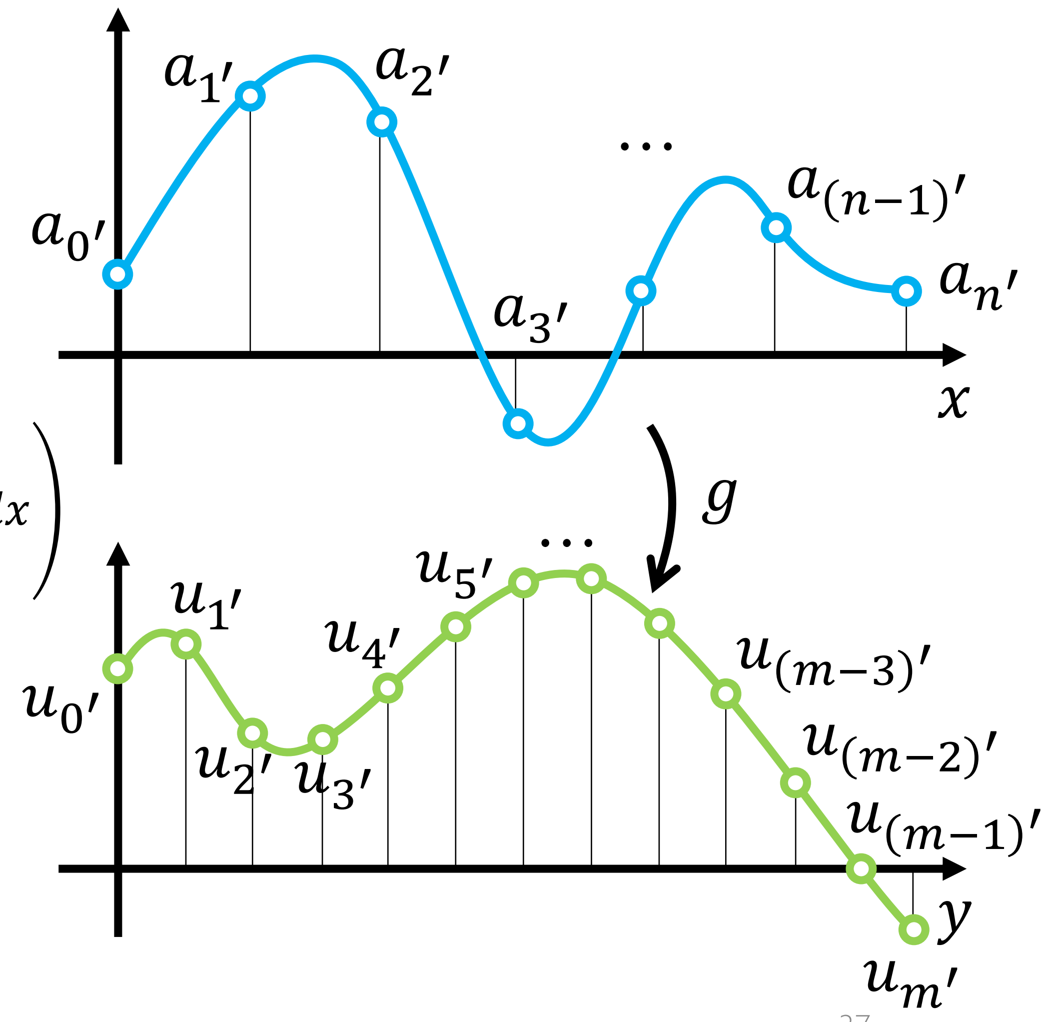
$$u(y) = \sigma\left(\int \kappa(y, x) a(x) dx\right)$$

Kernel function

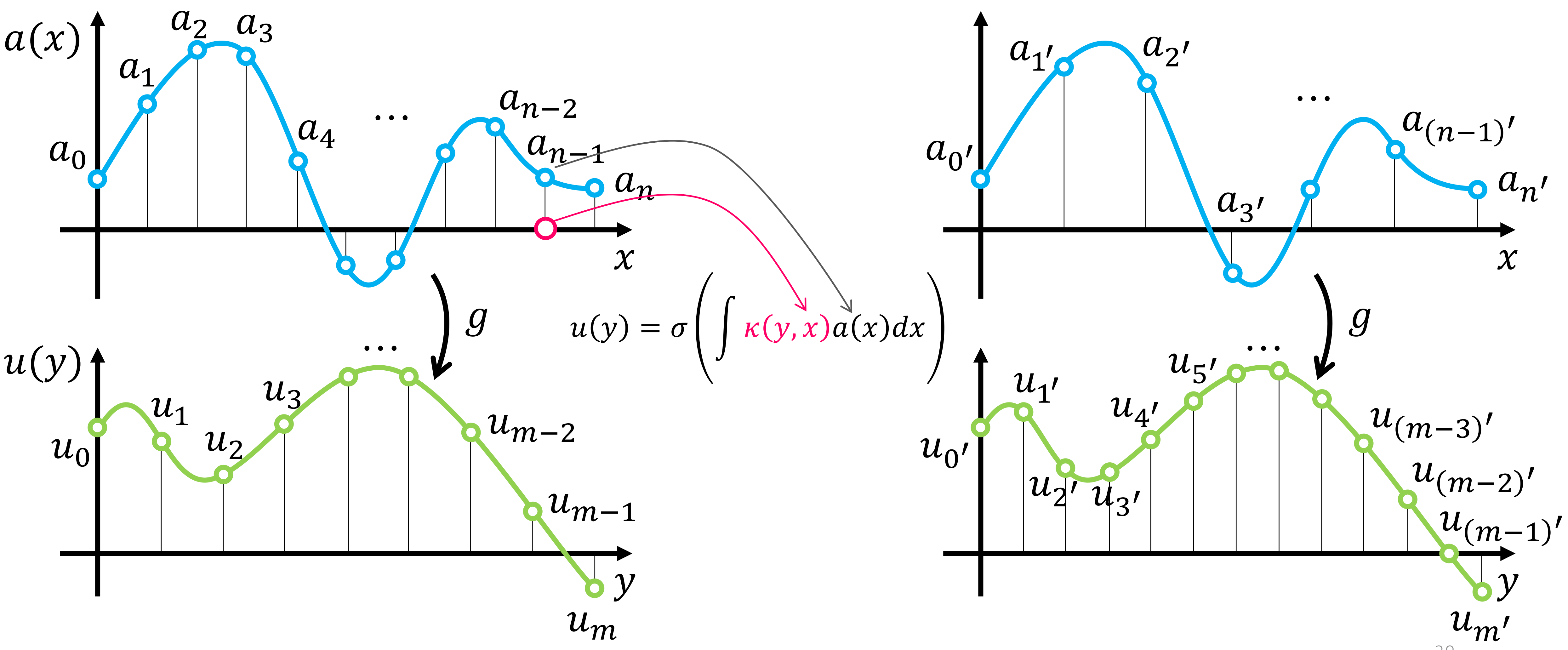
Assuming we learned the kernel function…



$$u(y) = \sigma\left(\int \kappa(y,x)a(x)dx\right)$$

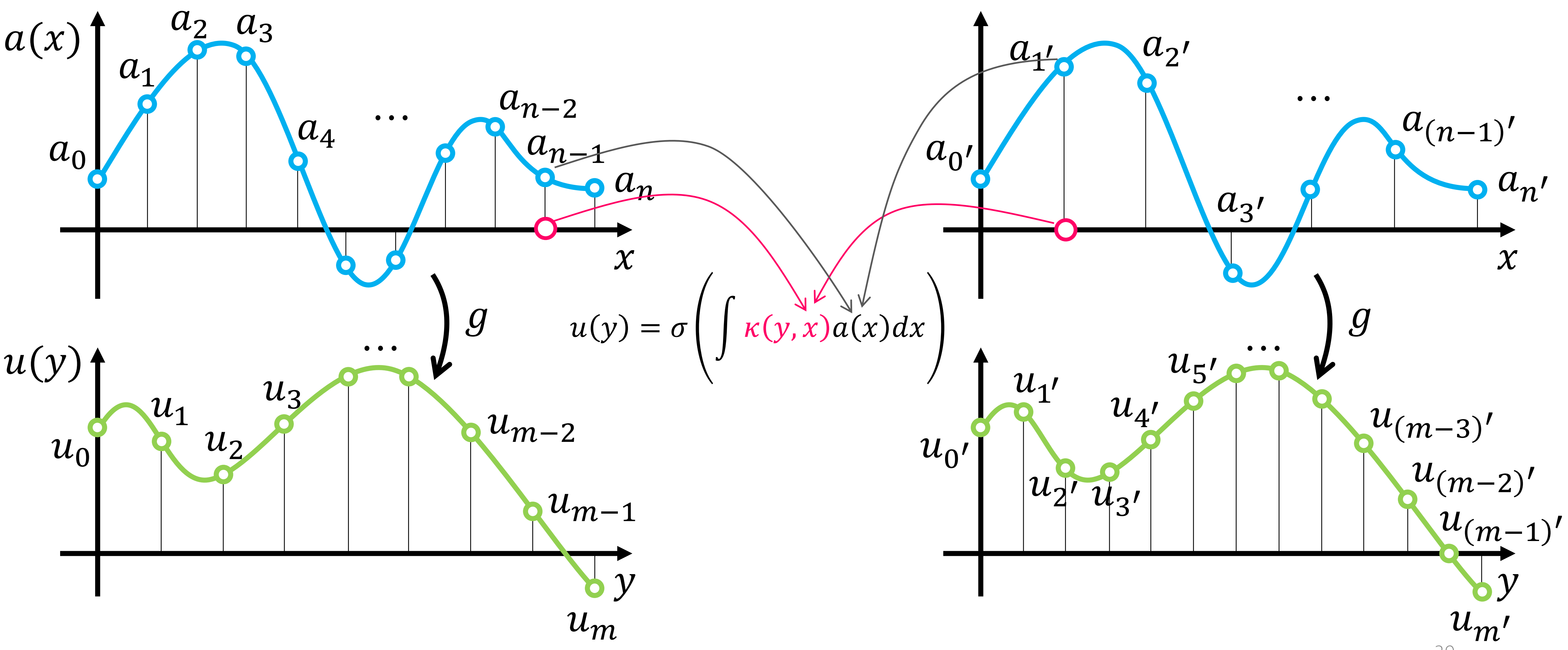


Assuming we learned the kernel function…



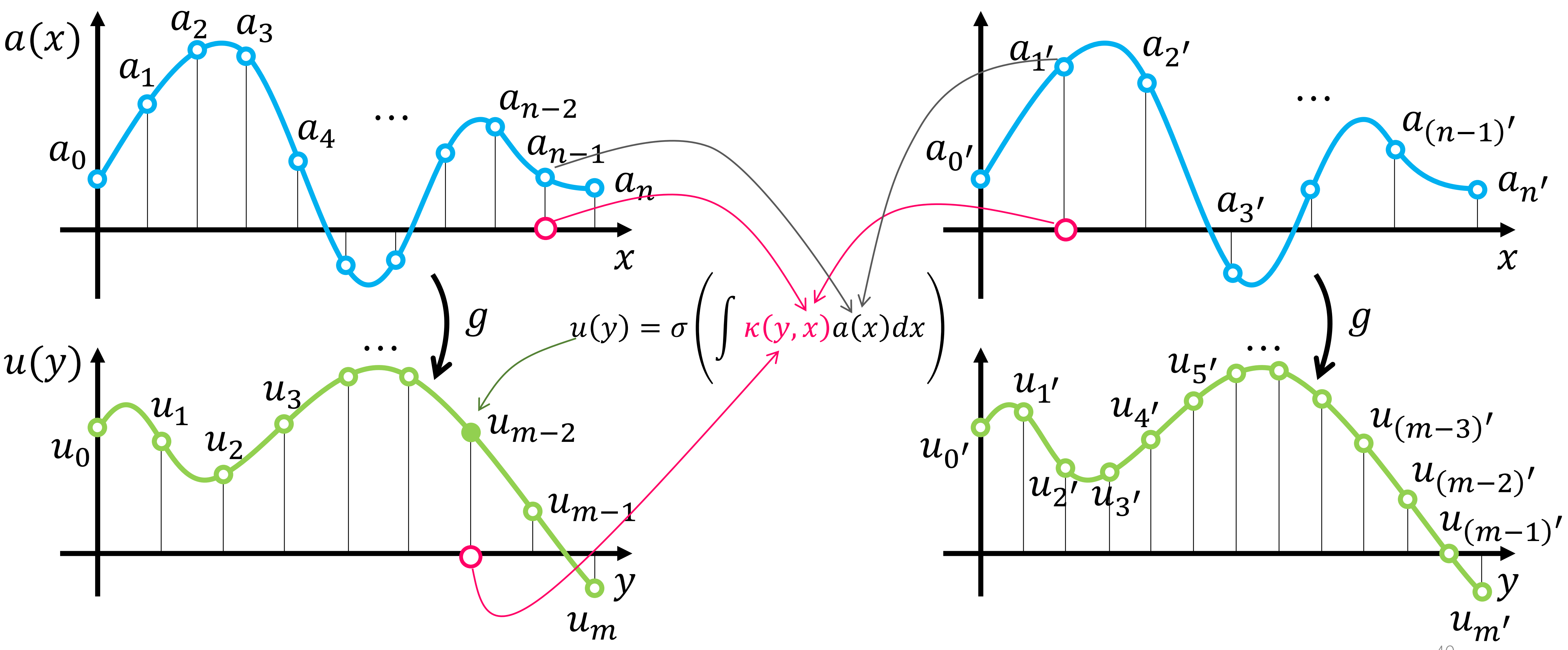
$$u(y) = \sigma\left(\int \kappa(y, x)\, a(x)\, dx\right)$$

Assuming we learned the kernel function…

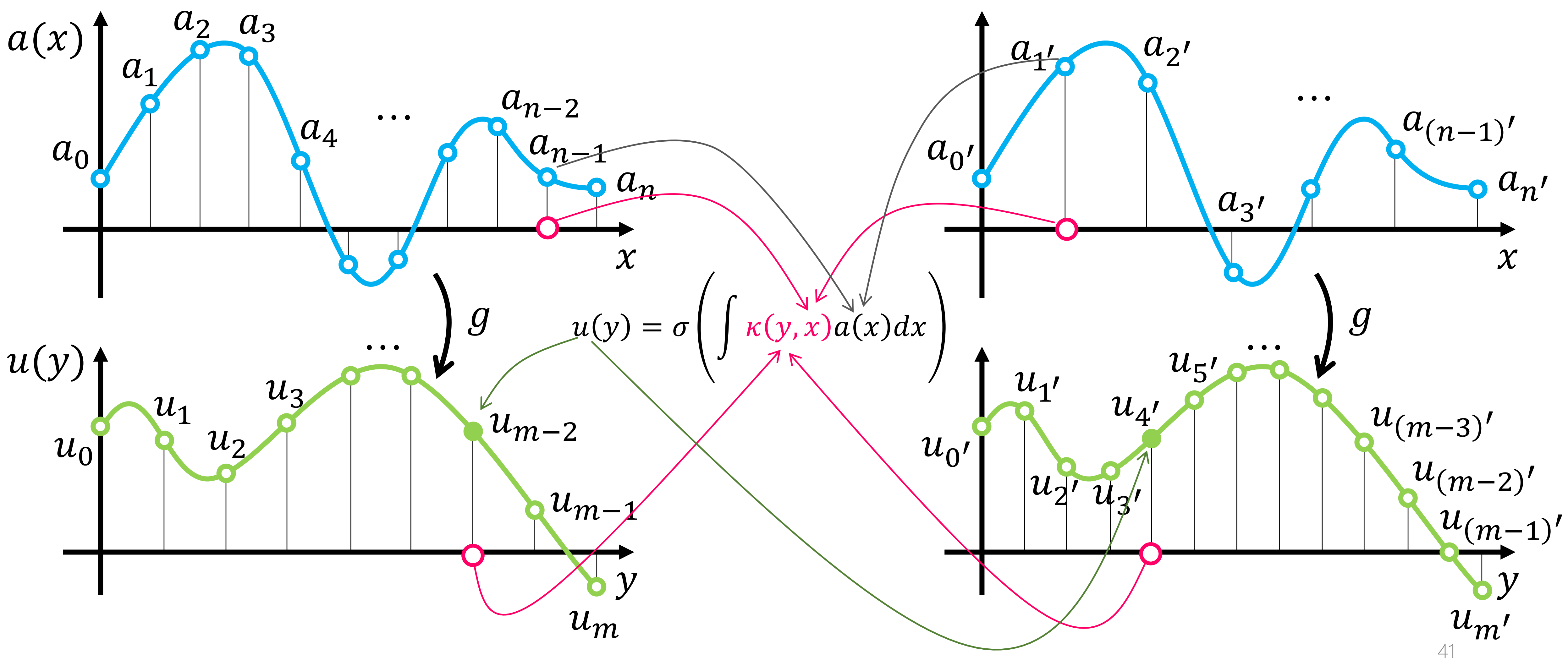


$$u(y) = \sigma\left(\int \kappa(y,x)\,a(x)\,dx\right)$$

Assuming we learned the kernel function…



Assuming we learned the kernel function…



Works great in theory, but how?

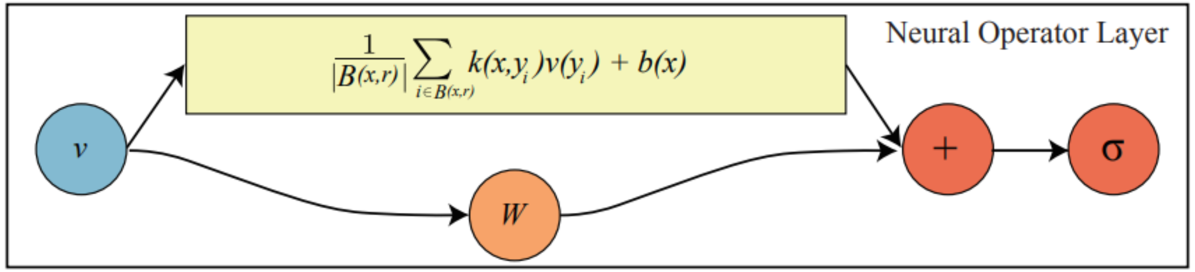
$$u(y) = \sigma\left(\int \kappa(y,x)a(x)dx\right)$$

- Practically, we need to compute (in each neural network layer):
    - Continuous kernel function $\kappa(y,x)$
    - Continuous integral $\int \cdot\, dx$
- ...which are computationally expensive!

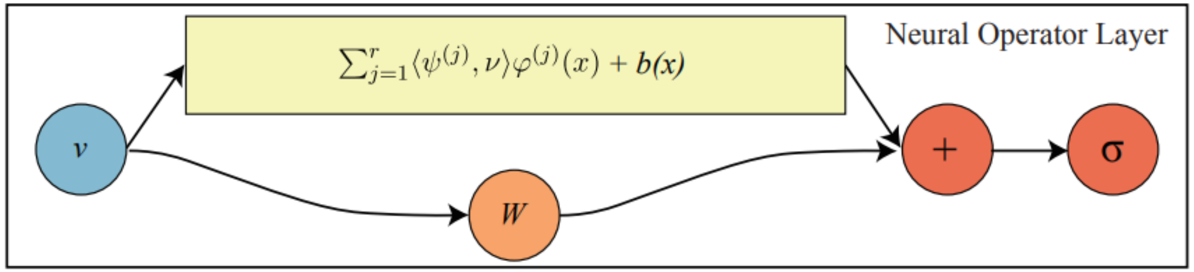
Works great in theory, but how?

- Kovachki, Li, et al. (2022) – Different versions of the linear integral operator

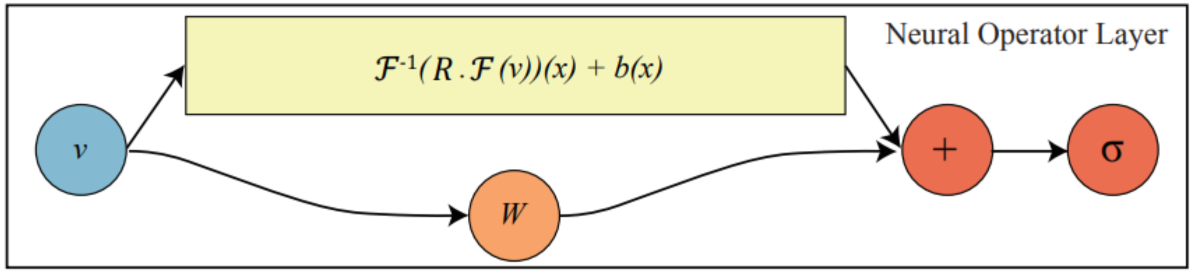
Graph Neural Operator



Low-rank Neural Operator



Fourier Neural Operator



Images from Kovachki et al. ICML 2022.

Fourier Neural Operators

- Convolution Theorem:
  - The convolution $f * g$ of two functions f and g is equivalent to a simple element-wise (Hadamard) product $\odot$ in the Fourier (spectral) domain*:

$$\mathcal{F}(f * g) = \mathcal{F}(f) \odot \mathcal{F}(g)$$

*Sun et al. (2020). ZerNet: Convolutional Neural Networks on Arbitrary Surfaces via Zernike Local Tangent Space Estimation. Computer Graphics Forum. https://arxiv.org/pdf/1812.01082

Fourier Neural Operators

- Convolution Theorem:
  - The convolution $f * g$ of two functions f and g is equivalent to a simple element-wise (Hadamard) product $\odot$ in the Fourier (spectral) domain*:
$$\mathcal{F}(f * g) = \mathcal{F}(f) \odot \mathcal{F}(g)$$

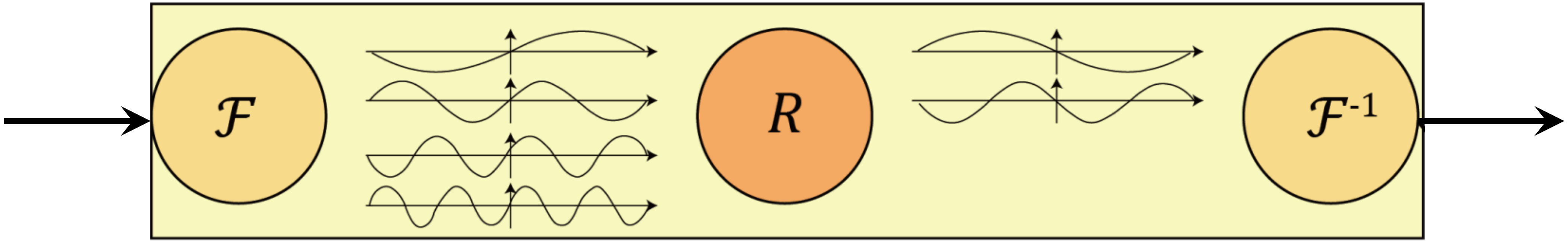
- If the integral operator was the convolution operator, i.e.:
$$u(y) = \sigma\left(\int \kappa(y,x)a(x)dx\right) \quad \rightarrow \quad u(y) \coloneqq \sigma\big((\kappa * a)(y)\big) = \sigma\left(\int \kappa(y-x)a(x)dx\right)$$
  - Then by the convolution theorem:
$$(\kappa * a)(y) = \mathcal{F}^{-1}\big(R \odot \mathcal{F}(a)\big)(y)$$
  - $\mathcal{F}(a)$: Fourier transform of the input function $a(x)$
  - $R \coloneqq \mathcal{F}(\kappa)$: Kernel function learned as Fourier coefficients (no need to evaluate κ directly)

*Sun et al. (2020). ZerNet: Convolutional Neural Networks on Arbitrary Surfaces via Zernike Local Tangent Space Estimation. Computer Graphics Forum. https://arxiv.org/pdf/1812.01082

Fourier Neural Operators

- FNO Layer
  1. Fourier transform $\mathcal{F}(a)$
  2. Linear transform R
  3. Inverse Fourier transform $\mathcal{F}^{-1}$

$$\mathcal{F}^{-1}\big(R \odot \mathcal{F}(a)\big)(y)$$



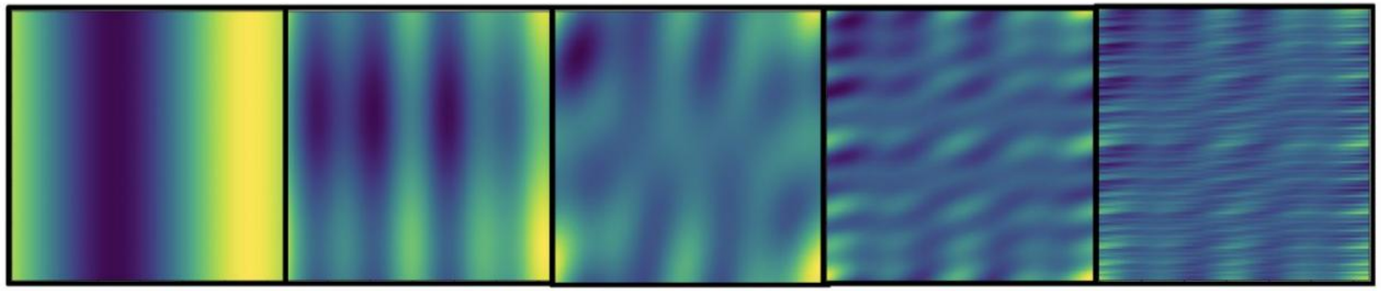
For a uniform grid, fast Fourier transform is available. Otherwise, discrete Fourier transform (slow).

Images from Kovachki et al. ICML 2022.

Fourier Neural Operators

- The Fourier Filters

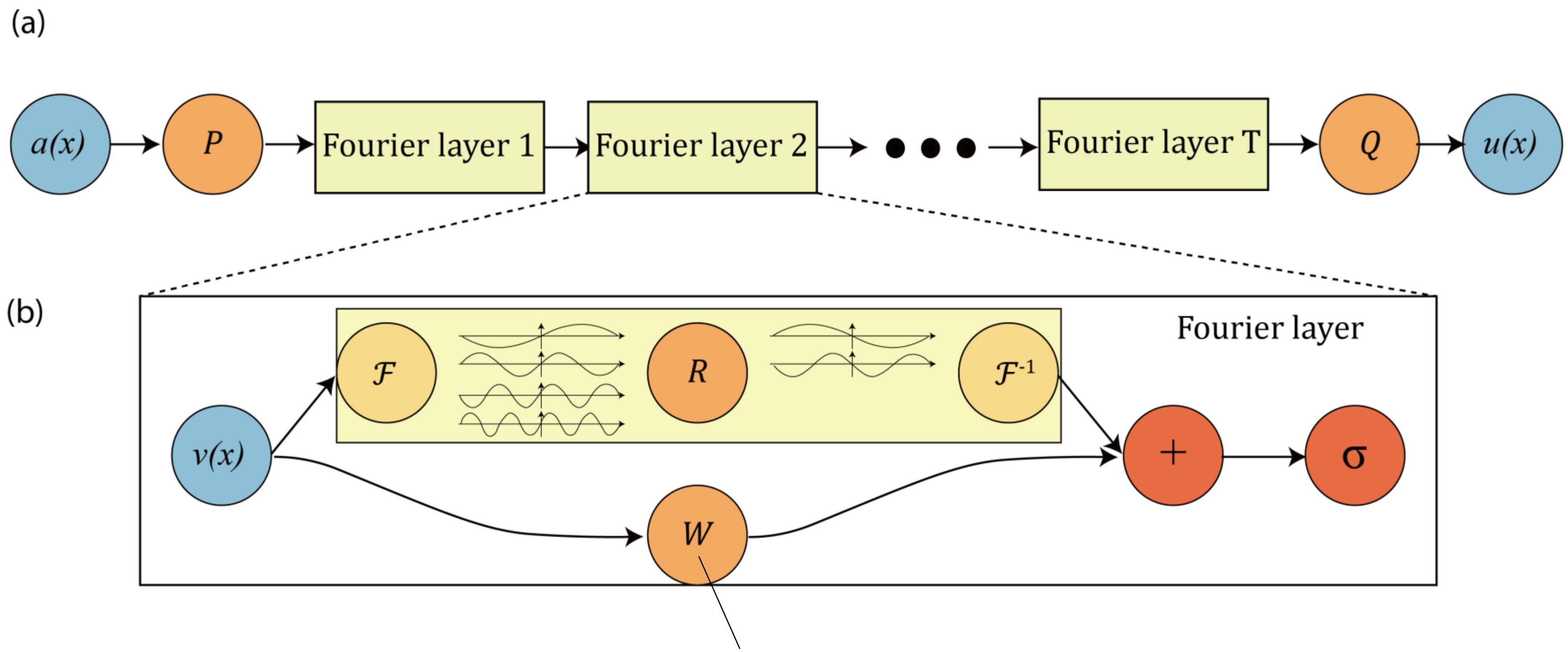
Filters in CNN



Fourier Filters

Fourier Neural Operators

- The full architecture of FNO



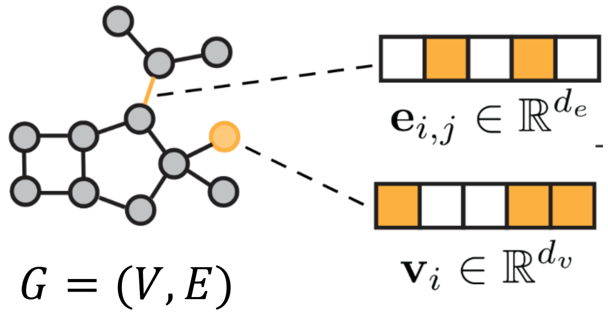
W: Local (point-wise) linear operator (skip connection)
to "keep the track of the location information and non-periodic boundary"*

Images from Kovachki et al. ICML 2022.

*Zongyi Li, ICLR Presentation (2021). https://iclr.cc/media/iclr-2021/Slides/3281.pdf

Graph Neural Operators

- Message Passing Neural Networks (Gilmer et al. 2017)
  - A graph G with node features $v_i \in V$ and edge features $e_{i,j} \in E$.



Diagrams from https://projects.volkamerlab.org/teachopencadd/talktorials/T035_graph_neural_networks.html

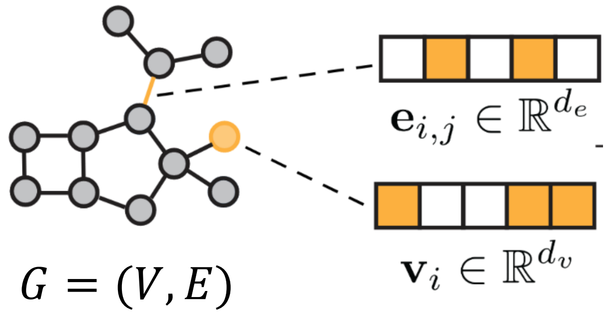
Graph Neural Operators

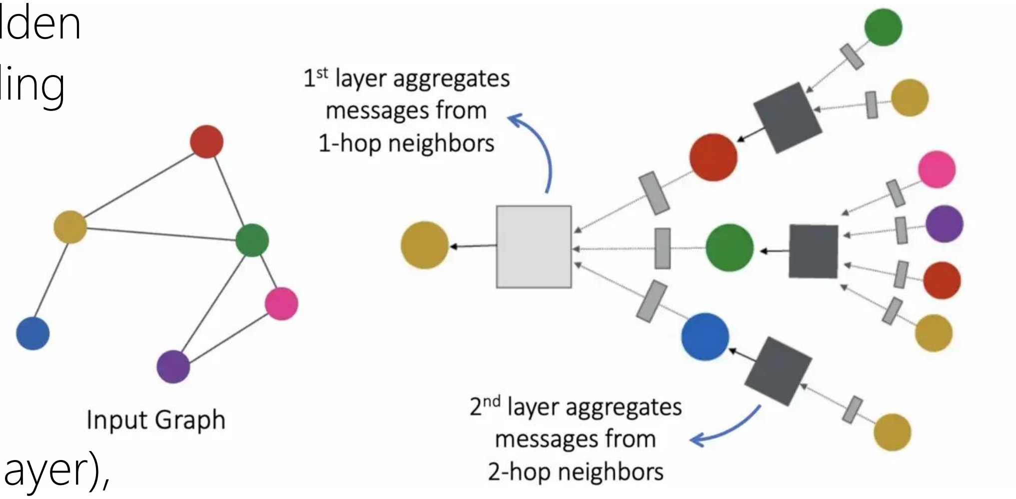
- Message Passing Neural Networks (Gilmer et al. 2017)
  - A graph G with node features $v_i \in V$ and edge features $e_{i,j} \in E$.
  - Message Passing Phase—Runs for K time steps. Update hidden states $h_i^{(k)}$ at each node based on messages $m_i^{(k+1)}$ according to:

$$m_i^{(k)} = \sum_{j \in N(i)} M^{(k)}\left(h_i^{(k-1)}, h_j^{(k-1)}, e_{i,j}\right)$$

$$h_i^{(k)} = U^{(k)}\left(h_i^{(k-1)}, m_i^{(k)}\right)$$

…, where $M^{(k)}$ is a message function (typically a neural net layer), $U^{(k)}$ is vertex update function (again, a neural network layer), and $N(i)$ is the neighbors of v_i in G.





Diagrams from https://projects.volkamerlab.org/teachopencadd/talktorials/T035_graph_neural_networks.html

Graph Neural Operator

- Graph Neural Operator using Message Passing Graph Networks

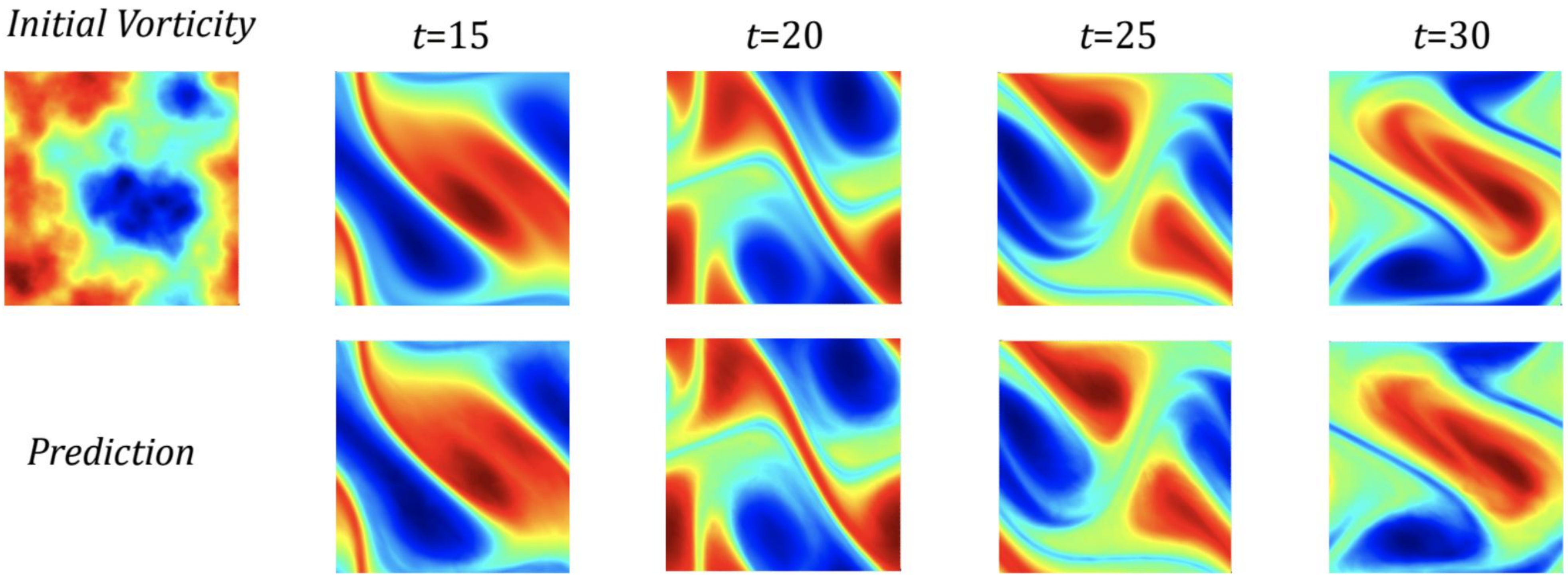
$$v_{t+1}(x) = \sigma\left(W v_t(x) + \frac{1}{|N(x)|} \sum_{y \in N(x)} \kappa_\phi\big(e(x,y)\big)\, v_t(y) \right)$$

  - $\kappa_\phi\big(e(x,y)\big)$: A message passing neural network taking edge features as input.
  - Edge features $\boldsymbol{e}(x,y)$: For example, the edge weight can be defined as

$$e(x,y) = \big(\underbrace{x, y}_{\text{Positions}}, \underbrace{a(x), a(y)}_{\text{Coefficients/physics parameters at those positions}}\big)$$

Examples & Use Cases

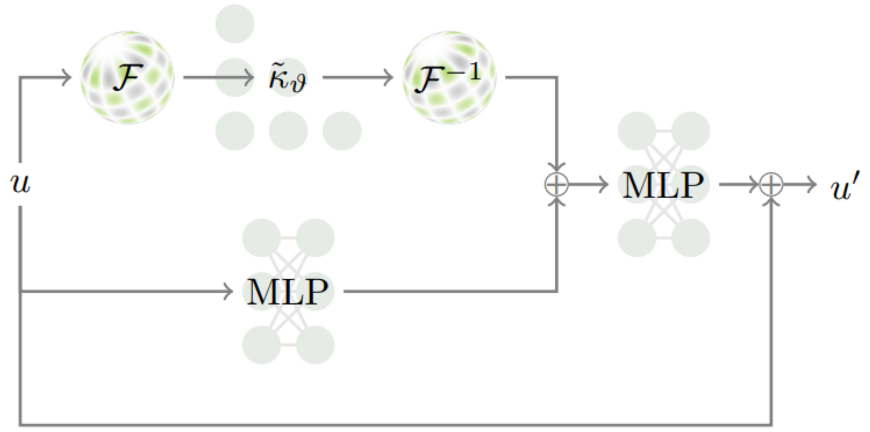
- Supervised learning of dynamics

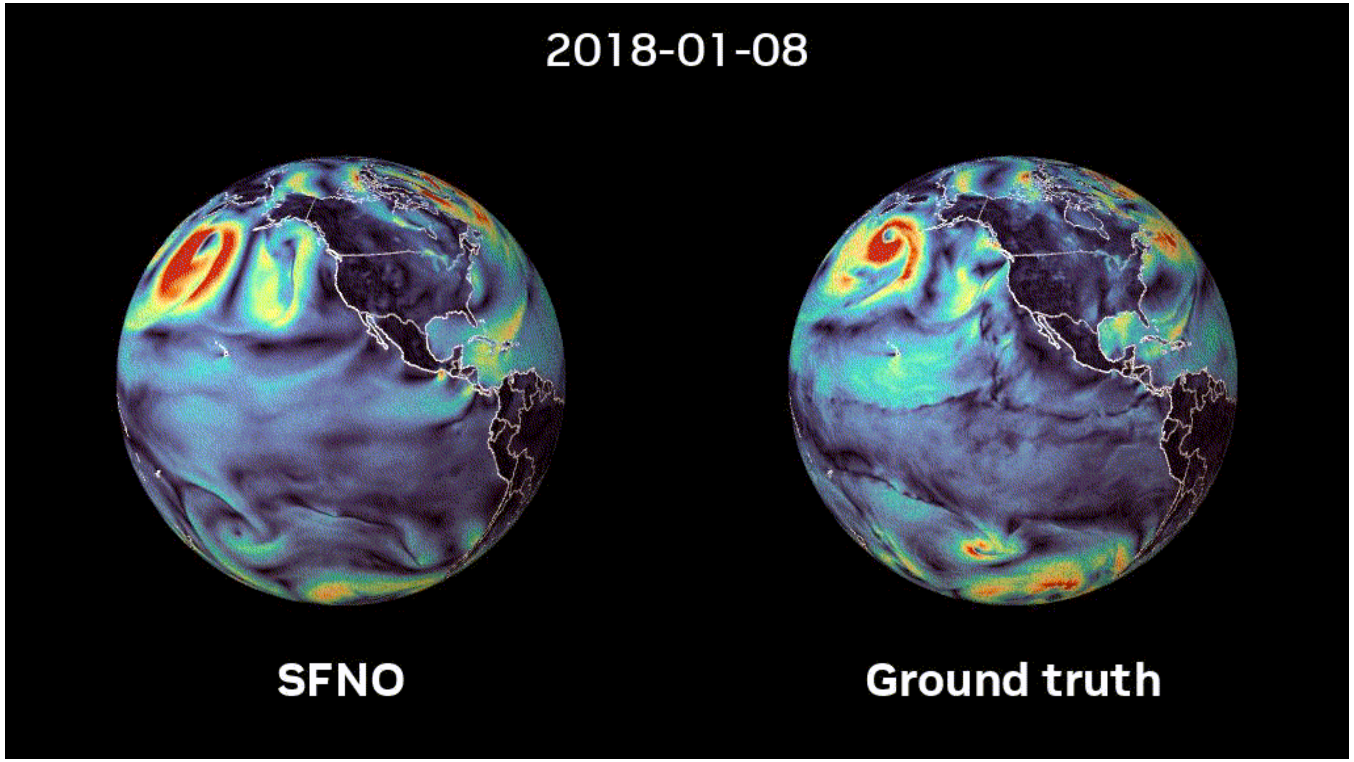


https://zongyi-li.github.io/neural-operator/

Examples & Use Cases

- Spherical Fourier Neural Operators (SFNO)

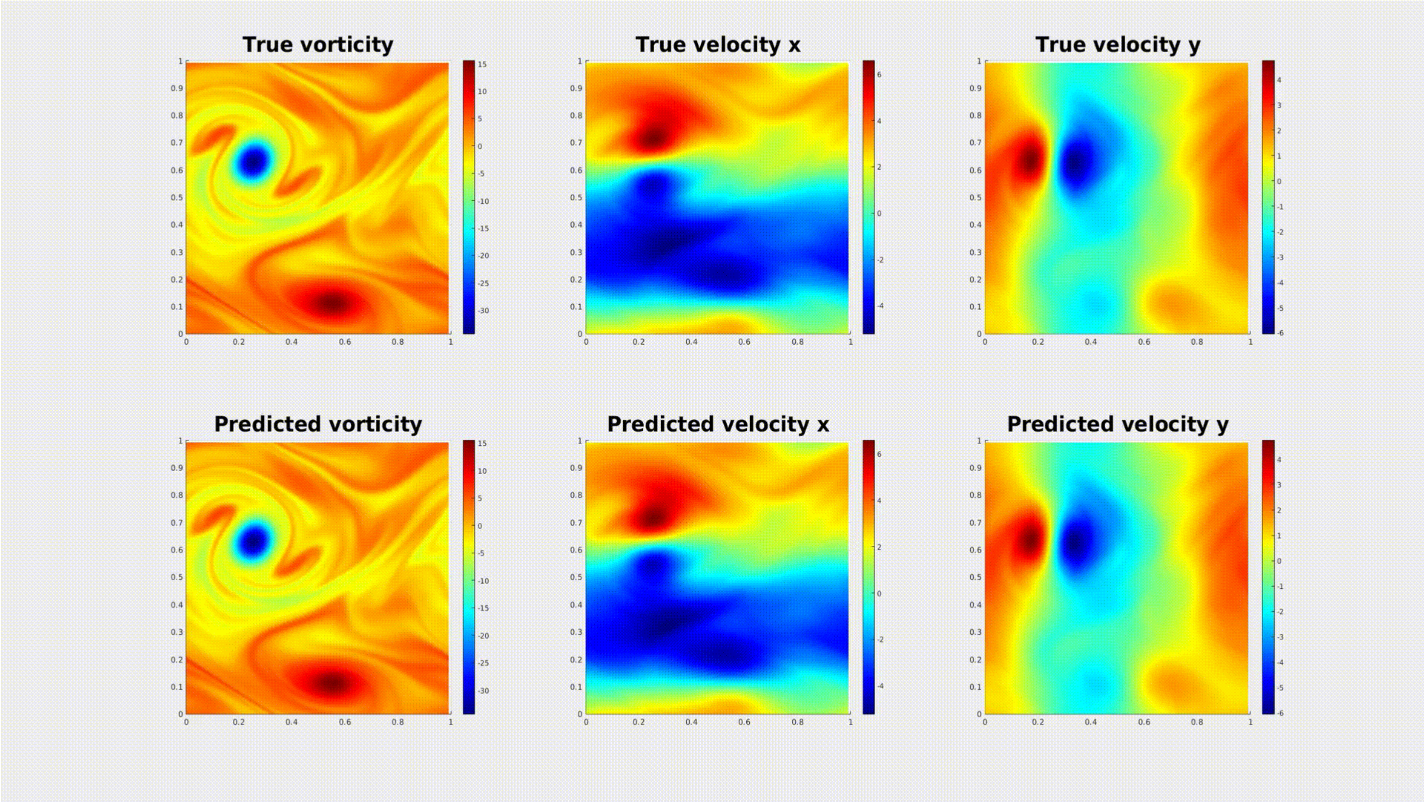




https://zongyi-li.github.io/neural-operator/

Examples & Use Cases

- Neural operators can be combined with physics-informed loss



https://zongyi-li.github.io/neural-operator/

Takeaways

- Operator learning than solving
  - Learn a family of PDE instead of solving one instance at a time.
  - Doesn’t require he explicit form, although a similar physics-loss could also be introduced (e.g. PINO)
- Learning in the function space
  - Map the values to e.g. Fourier space for continuous inputs and outputs.
  - Resolution invariant, mesh-invariant
- More accurate than other deep learning methods (discuss why?)