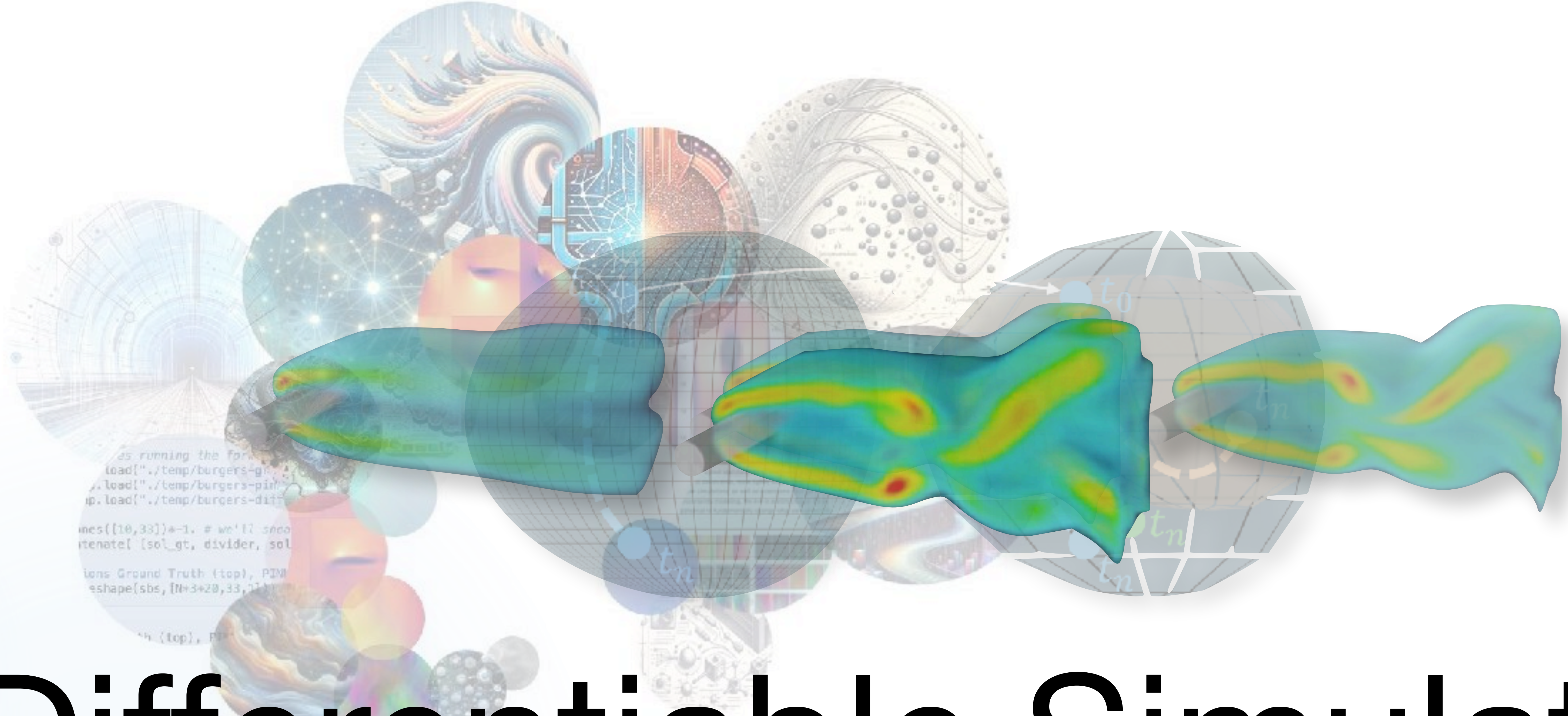


Differentiable Simulations

DEEP LEARNING FROM AND WITH NUMERICAL PDE SOLVERS (PART 1)

Contents

Physical Loss Terms

Differentiable Physics Simulations

- Examples

Differentiable Physics Training

- Examples

Physical Loss Terms

Physical Models and Residuals

Formulating a “Classic” Numerical Simulator

- Physical model PDE $\mathscr{P}$ with phase space states $\mathbf{u}(\mathbf{p}, t)$, shortened to $\mathbf{u}(t)$
- Reformulate $\mathscr{P}$ to compute new state or solution
- Time derivative $\mathbf{u}_t = \mathscr{F}(\mathbf{u}_x, \mathbf{u}_{xx}, \ldots \mathbf{u}_{xx\ldots x})$
- Apply standard time integration; Euler step gives $\mathbf{u}(t + \Delta t) = \mathbf{u}(t) + \Delta t \; \mathbf{u}_t$
- More complex PDEs like *Navier-Stokes* $\frac{\mathrm{D}u}{\mathrm{D}t} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 u + g, \; \nabla \cdot u = 0$

  require *operator splitting* or custom time integration schemes

Physical Models and Residuals

Residual Equations

Example setup for time integration (many variations possible)

Solution given by NN $f(\mathbf{x};\theta) \approx \mathbf{y}^* = \mathbf{u}(t+\Delta t)$ with $\mathbf{x} = \mathbf{u}(t)$, evaluate $\mathscr{F}$ here

$R := \mathbf{u}(t+\Delta t) - \mathbf{u}(t) - \Delta t \ \mathscr{F}(\mathbf{u}_x, \mathbf{u}_{xx}, \ldots \mathbf{u}_{xx\ldots x})$ for $\mathscr{F}$ evaluated at time $(t+\Delta t)$

Minimize: $\left| f(\cdots) - \mathbf{x} - \Delta t \ \mathscr{F}(\mathbf{u}_x, \mathbf{u}_{xx}, \cdots) \right|^2$

⇨ Does not require pre-computed solutions of $\mathscr{P}$ anymore

Physical Models and Residuals in NNs

Integration into Neural Networks

Add residual as additional loss term

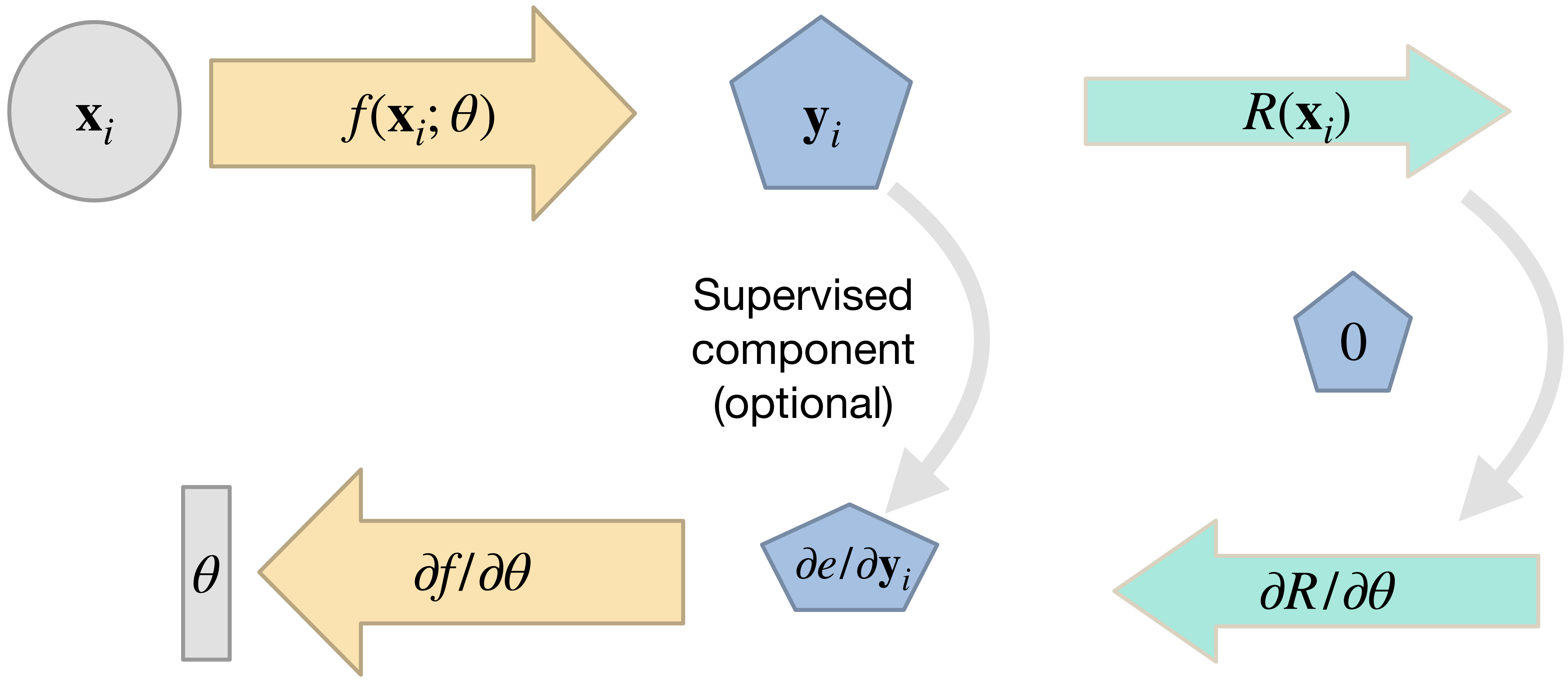
Including residual R in DL loss gives: $\arg\min_\theta \sum_i \alpha_0 \left(f(\mathbf{x}_i;\theta) - y_i^*\right)^2 + \alpha_1 R(\mathbf{x}_i)$

Relative weighting of terms via α_0, α_1 factors

Backpropagate through expressions to train NN

Physical Models and Residuals in NNs

Visual Outline



⇨ So far generic, 2 variants will follow…

Physical Models and Residuals

Side Note: Steady-state Problems also fit in …

- No explicit time dimension t, but typically still involve an *imaginary* time
  - Iterative solvers are used to compute converged solutions
  - Solver iterations can be treated as *imaginary / virtual time*
  - $\mathbf{u}(t)$ at initial time $t = 0$ typically zero
- Thus: special case. We'll focus on time dependent problems

Physical Models and Residuals in NNs (v1)

Flexible NNs with Traditional Discretization - Variant 1

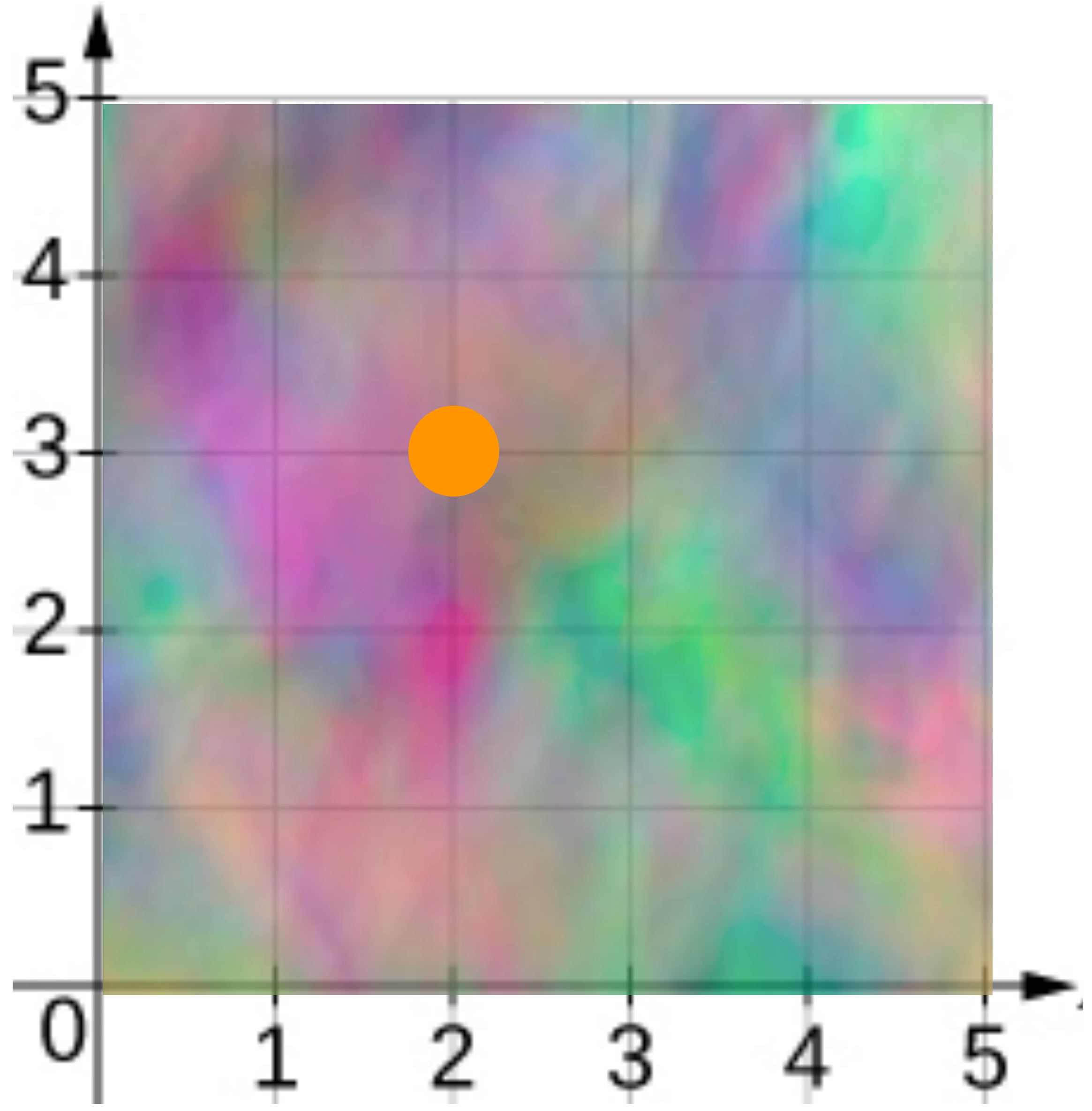
Work with chosen spatial discretization (grid, mesh, points …)

E.g., $\mathbf{u}$ represented by dense array in space

Discretize equations of R accordingly, and provide $(\partial R / \partial \theta)^T$

For $\mathbf{u} = f(\mathbf{x}; \theta)$ with $R(\mathbf{u})$: provide $(\partial R / \partial \mathbf{u})^T$, then rely on backprop

E.g., compute $\partial \mathbf{u} / \partial x$ via finite difference on grid



E.g.: value stored at array location [2,3]

Physical Models and Residuals in NNs (v1)

Example: Learning Divergence-Freeness

Specific example: Continuity equation $\nabla \cdot \mathbf{u} = 0$

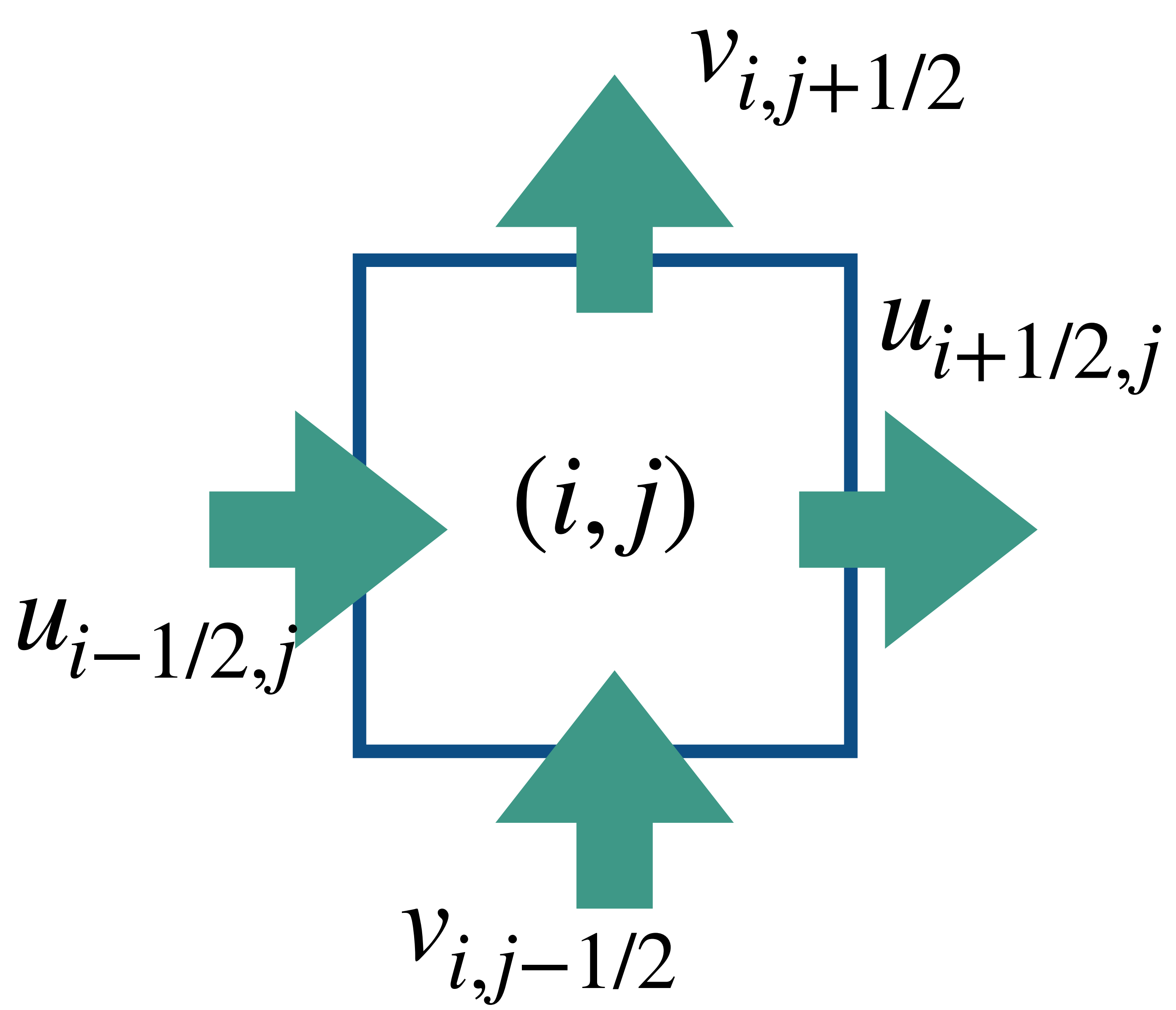
Single time step of the form $\mathbf{u}(1) = \mathbf{u}(0) - \nabla p;\ \ p = \nabla^{-2}\big(\nabla \cdot \mathbf{u}(0)\big)$

Learn to predict pressure field via NN: $p = f(\mathbf{u}(0); \theta)$

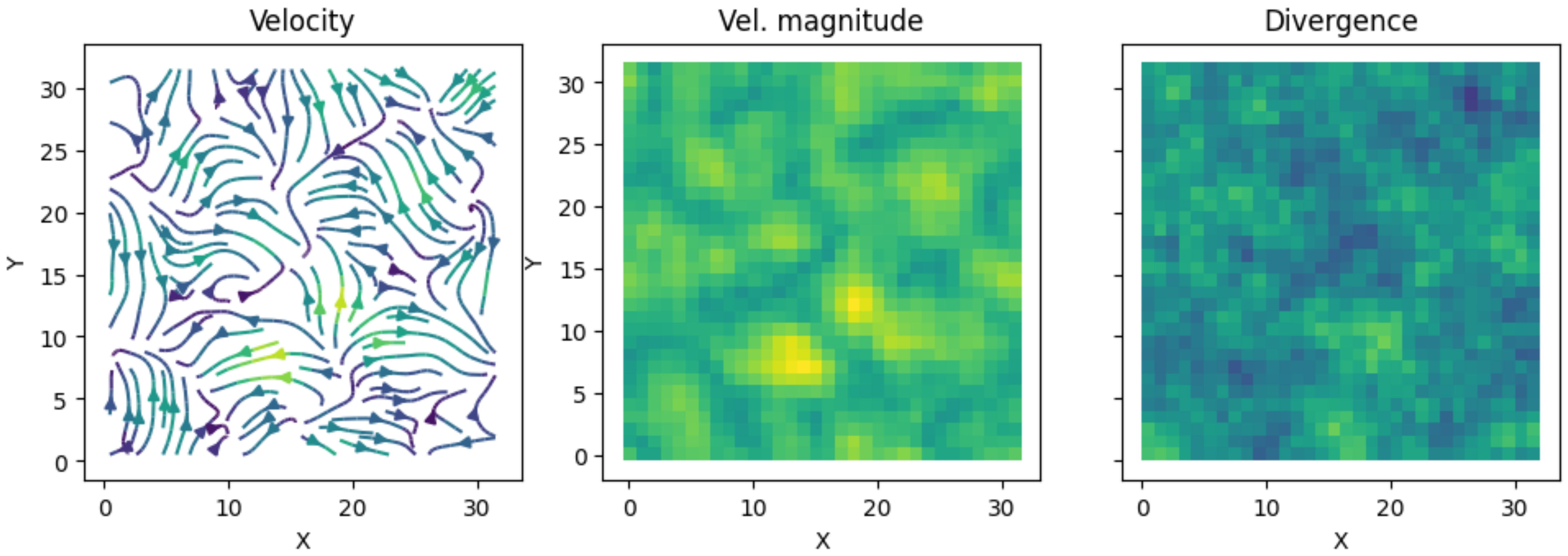
Goal is $\nabla \cdot \mathbf{u}(1) = 0$ and hence minimize $\nabla \cdot \big(\mathbf{u}(0) - \nabla f(\mathbf{u}(0); \theta)\big)$

- Note: has to hold at all times, hence time step of 1
- Use Eulerian grid and CNN, discretize divergence and gradient operators via finite differences for all computational cells

Divergence for cell (i,j):



Physical Models and Residuals in NNs (v1)



Classic Paper by Tompson et. al, reimplemented in:

https://www.physicsbaseddeeplearning.org/physicalloss-div.html

Physical Models and Residuals in NNs (v1)

v1 Discussion

✅ Accurate and fast residual calculations, updates whole solutions $\mathbf{u}$

✅ No pre-computations and large data sets needed

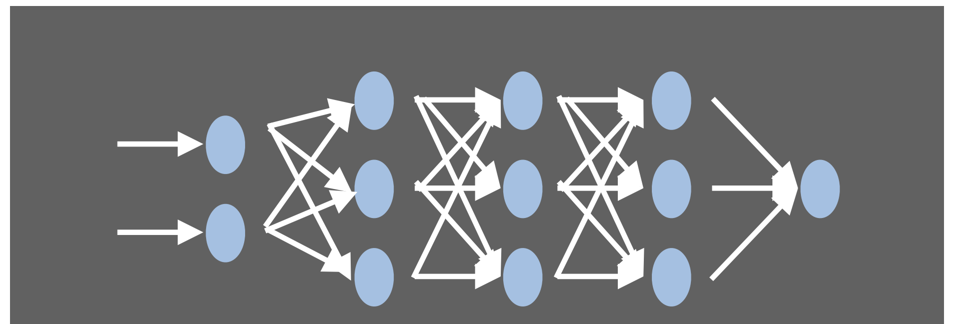
✅ Good for multi-modal problems

❌ Not “adaptive” , discretization needs to be chosen carefully

❌ Slower than supervised approaches (evaluation and backprop of R)

Physical Models and Residuals in NNs (v2)

Special Case with Fully connected NNs - Variant 2



Fully-connected NN to represent solution and for computing derivatives

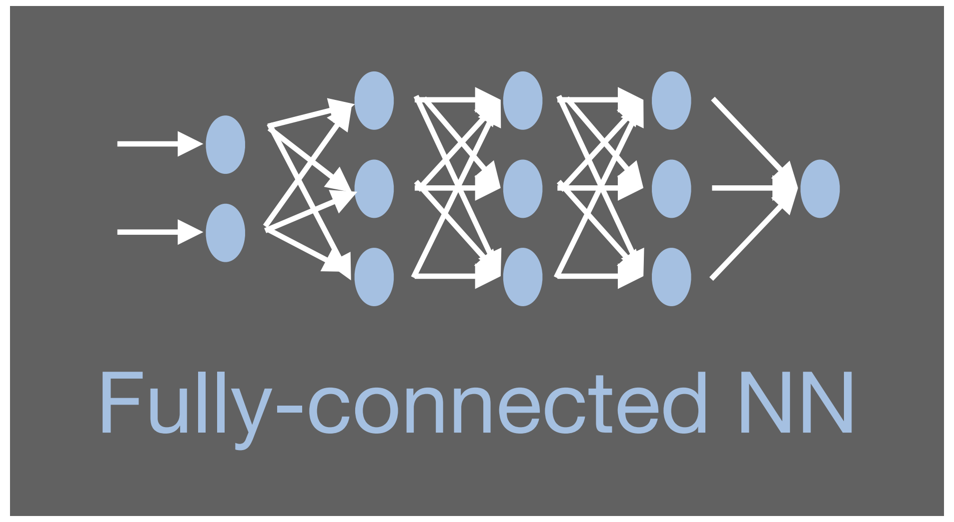
"Neural field" representation

Value retrieved
by querying network
$f(\mathbf{p}, \cdot\,; \theta) \quad \mathbf{p} = (2{,}3)^T$

E.g.: value stored in
neural network
at location [2,3]

Physical Models and Residuals in NNs (v2)

Special Case with Fully connected NNs - Variant 2

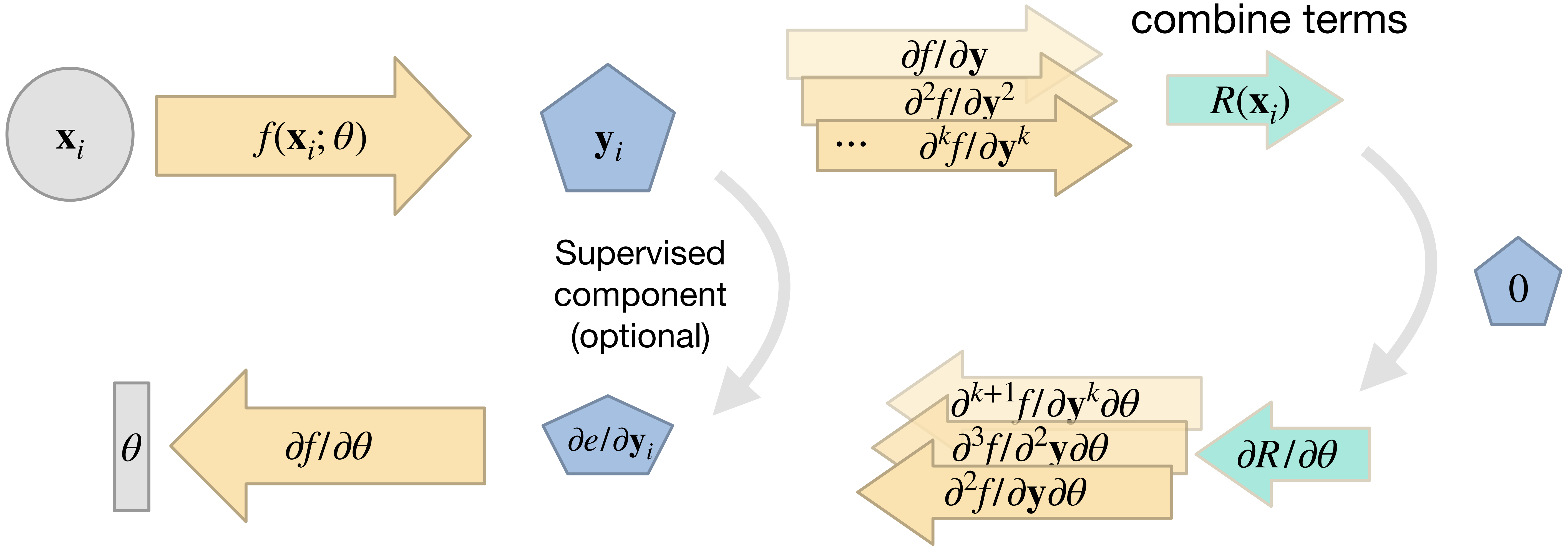


Fully-connected NN to represent solution and for computing derivatives

Spatial coordinates $\mathbf{p}$ are a part of $\mathbf{x}$ now, i.e. an input to f; evaluate f to sample $\mathbf{u}$

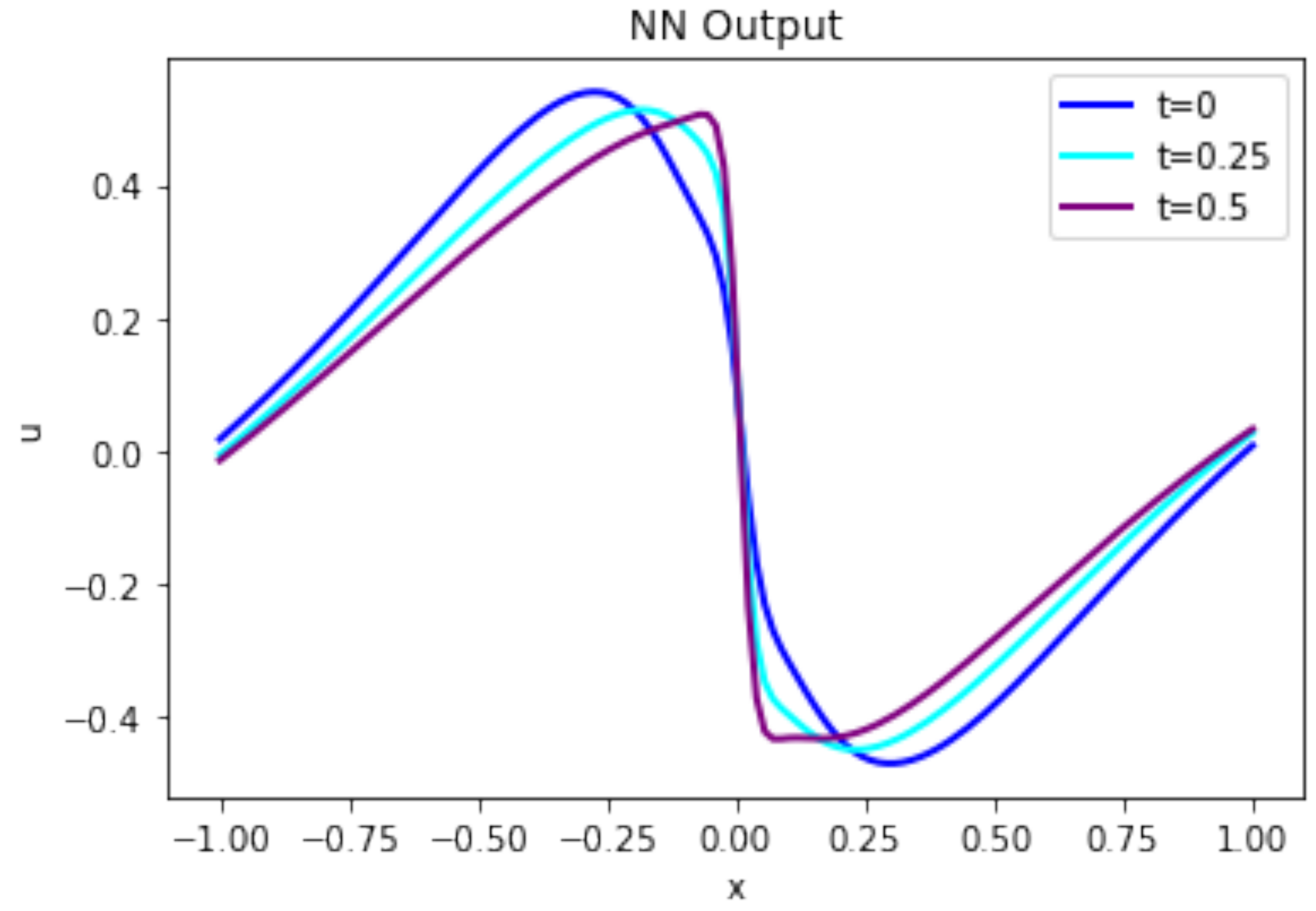
Commonly known as "*physics-informed neural networks*" PINNs

Formulate residual via NN f, e.g., compute $\partial \mathbf{u} / \partial x$ via derivative of NN $\partial f / \partial x$



Physical Models and Residuals in NNs (v2)

Example from Raissi et al. *"Physics-informed neural networks"*



https://www.physicsbaseddeeplearning.org/physicalloss-code.html

Physical Models and Residuals in NNs (v2)

v2 Discussion

✅ Likewise: No pre-computations and large data storages needed

✅ Likewise: Good for multi-modal problems

✅ Fully-connected NN “learns” discretization

❌ Slow derivatives (especially for higher orders)

❌ Localized updates, typically require supervised constraints

❌ Accuracy relies on current learned state

Physical Residuals v1 vs v2

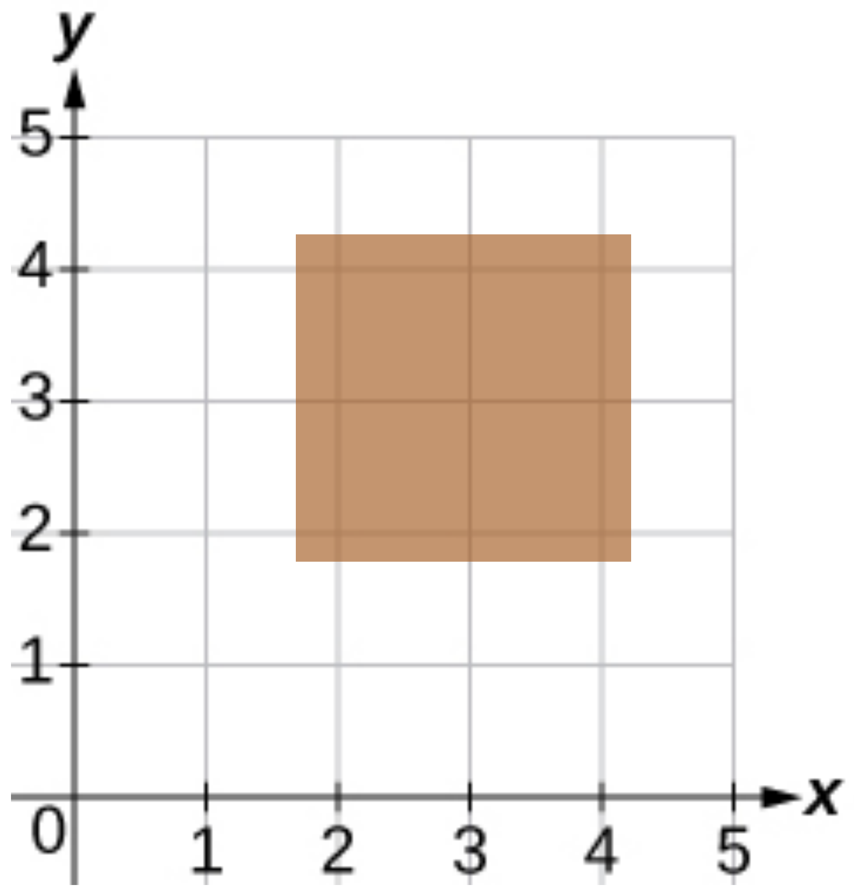
Summary / Discussion

v1:

✅ / ❌ Chosen discretization

✅ Reliable and fast evaluation of residual

✅ Each learning step provides gradient for whole domain



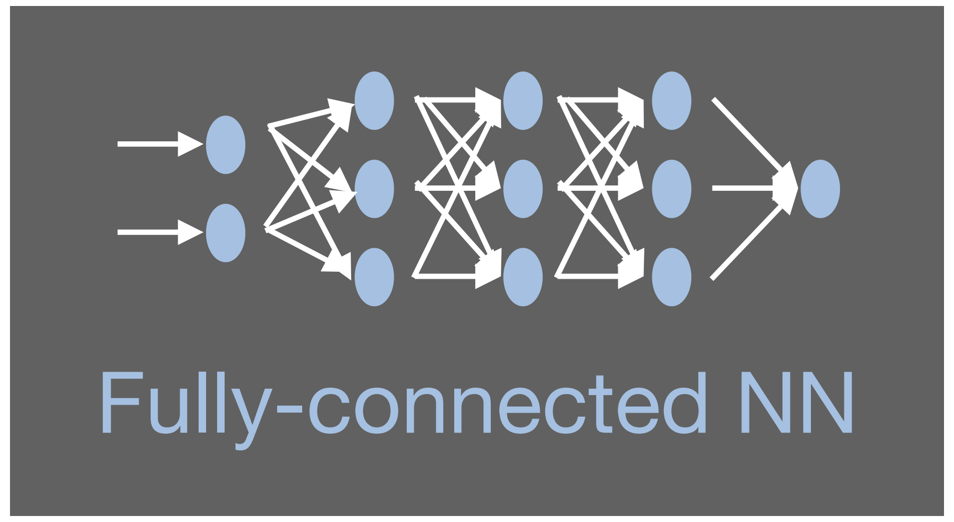
v2:

✅ / ❌ Discretization adapts at training time, influences derivatives

✅ Easy to start with

❌ Incompatible with most existing numerical techniques

❌ Each learning step provides only “local” updates



Physical Residuals v1 vs v2

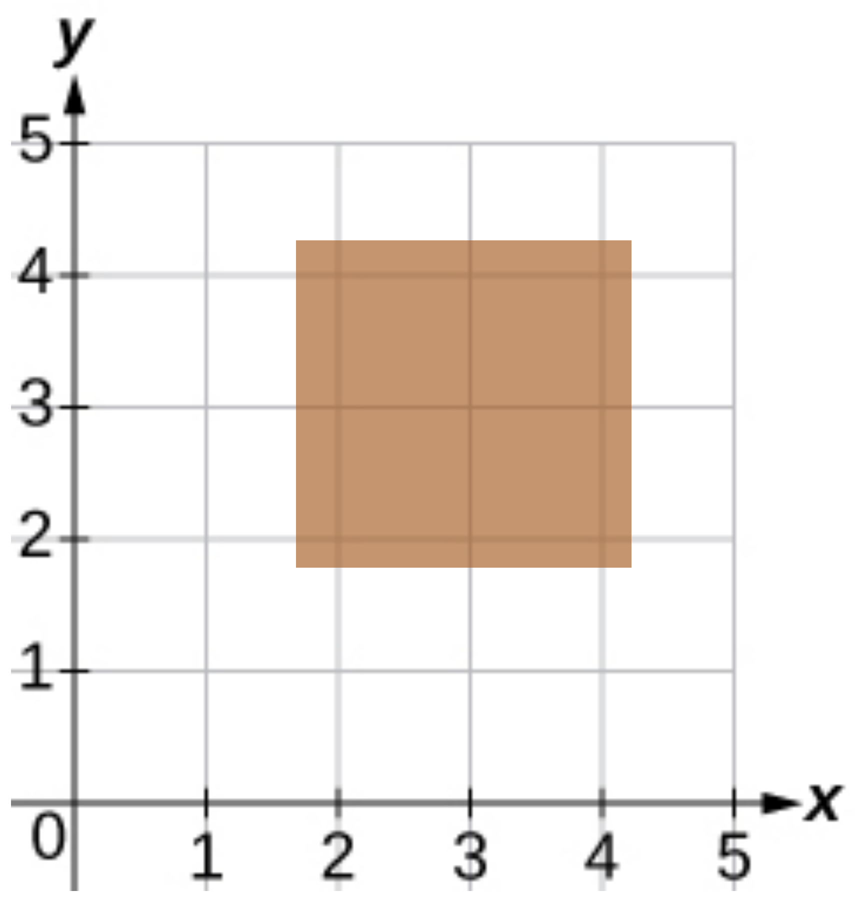
Summary / Discussion

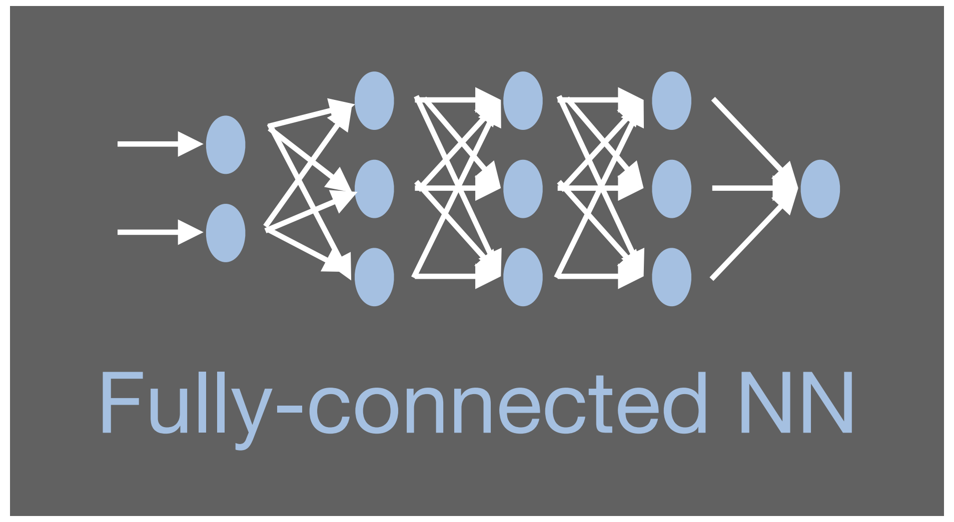
Compare loss propagation at training time:

v1: ResNet/Unet based - learning signal for full domain, potentially influences full NN state

v2: MLP based - localized sampling points, only partial updates of NN state

Strongly influences convergence





Physical Residuals in General

✅ Uses physical model

❌ Soft constraints, no guarantees

Outlook "Neural Fields":

- Hamiltonian / Lagrangian NNs , interesting variants of v2 residuals:

  https://arxiv.org/pdf/1906.01563 and https://arxiv.org/pdf/2003.04630.pdf

Next:

How can we use all of the "traditional knowledge" of physical simulations in DL?

Differentiable Physics Simulations

Differentiable Numerical Physics Simulations

Same Starting Point: “Classic” Numerical Simulator

- Physical model PDE $\mathscr{P}$ with phase space states $\mathbf{u}$
- $\mathscr{P}$ to compute new state, time derivative $\mathbf{u}_t = \mathscr{F}(\mathbf{u}_x, \mathbf{u}_{xx}, \ldots \mathbf{u}_{xx\ldots x})$
- Apply time integrator of choice
- ⇨ Augment $\mathscr{P}$ with NN, train via simulation derivatives $(\partial\mathscr{P}/\partial\mathbf{u})^T$
- “Simply” broader picture than before: full simulator rather than (partial) residual
- … before involving NNs, let’s look at how to compute and work with $(\partial\mathscr{P}/\partial\mathbf{u})^T$

Nomenclature

- Note: Naming schemes not (yet) unified
- Physics- constrained / based / augmented …
- Closely related to "classical" *Adjoint methods*
- Equivalent: Reverse mode differentiation (== backpropagation)
- Distinguish physical simulations from financial, crowd, light simulations etc., hence *"differentiable physics"* (DP)

Jacobian Vector Products

- Solver as sequence of m operations, e.g.: $\mathbf{u}(t+\Delta t) = \mathscr{P}_m \circ \dots \mathscr{P}_2 \circ \mathscr{P}_1(\mathbf{u}(t), \nu)$, with parameter ν (e.g., viscosity)
- Jacobian for $\mathbf{u}$ of solver operation i: $\dfrac{\partial \mathscr{P}_i}{\partial \mathbf{u}} = \begin{bmatrix} \partial \mathscr{P}_{i,1}/\partial u_1 & \cdots & \partial \mathscr{P}_{i,1}/\partial u_d \\ & \vdots & \\ \partial \mathscr{P}_{i,d}/\partial u_1 & \cdots & \partial \mathscr{P}_{i,d}/\partial u_d \end{bmatrix}$ (square in this case)
- We could compute $\partial \mathscr{P}_i / \partial \nu$, but Jacobians only needed for optimized quantities; here that's $\mathbf{u}$ (later on θ)
- Evaluate chain rule, e.g., for two operators: $\left.\dfrac{\partial(\mathscr{P}_1 \circ \mathscr{P}_2)}{\partial \mathbf{u}}\right|_{\mathbf{u}^n} = \left.\dfrac{\partial \mathscr{P}_1}{\partial \mathbf{u}}\right|_{\mathscr{P}_2(\mathbf{u}^n)} \left.\dfrac{\partial \mathscr{P}_2}{\partial \mathbf{u}}\right|_{\mathbf{u}^n}$ $\quad f(g(x)' = f'(g(x)) \; g'(x)$
- *Always* scalar loss as final operation with 1-column $\left(\dfrac{\partial L}{\partial \mathbf{u}}\right)^T$, i.e. gradient vector
- Leads to sequence of Jacobian-vector products: $\left(\dfrac{\partial \mathscr{P}_1}{\partial \mathbf{u}(t)}\right)^T \left(\dfrac{\partial \mathscr{P}_2}{\partial \mathscr{P}_1}\right)^T \left(\dfrac{\partial L}{\partial \mathscr{P}_2}\right)^T$

Jacobian Vector Products

- Solver as sequence of m operations, e.g.: $\mathbf{u}(t+\Delta t) = \mathscr{P}_m \circ \ldots \mathscr{P}_2 \circ \mathscr{P}_1(\mathbf{u}(t), \nu)$, with parameter ν (e.g., viscosity)
- In this notation: $\mathscr{P}_1$ applied first, $\mathscr{P}_m$ is last operation
- Solver derivative $\left(\frac{\partial \mathscr{P}}{\partial \mathbf{u}(t)}\right)^T$ Leads to sequence of m Jacobians:

$$\left(\frac{\partial \mathscr{P}_1}{\partial \mathbf{u}(t)}\right)^T \left(\frac{\partial \mathscr{P}_2}{\partial \mathscr{P}_1}\right)^T \cdots \left(\frac{\partial \mathscr{P}_{m-1}}{\partial \mathscr{P}_{m-2}}\right)^T \left(\frac{\partial \mathscr{P}_m}{\partial \mathscr{P}_{m-1}}\right)^T$$

Differentiable Physics Simulations: Fluids as an Example

Re-cap Fluid Simulations

Traditional Numerical Solver

Navier-Stokes equations

$$\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} = -\frac{1}{\rho}\nabla p + \nu \nabla \cdot \nabla \mathbf{u} \;, \quad \nabla \cdot \mathbf{u} = 0$$

Many advanced techniques available

Two main steps via operator splitting:

- Advection - transport $D\mathbf{u}/Dt = 0$
- Pressure projection - enforce divergence-freeness $\nabla \cdot u = 0$
- Viscosity would require additional implicit solve, analogous to pressure solve

Advection Only

Example Problem with Explicit Update Step

Example: advection of passive scalar density $d(\mathbf{x}, t)$ in velocity field $\mathbf{u}$

Physical model $\mathscr{P}$ performs time update $d(t + \Delta t) = \mathscr{P}(d(t), \mathbf{u}, t + \Delta t)$ via advection equation

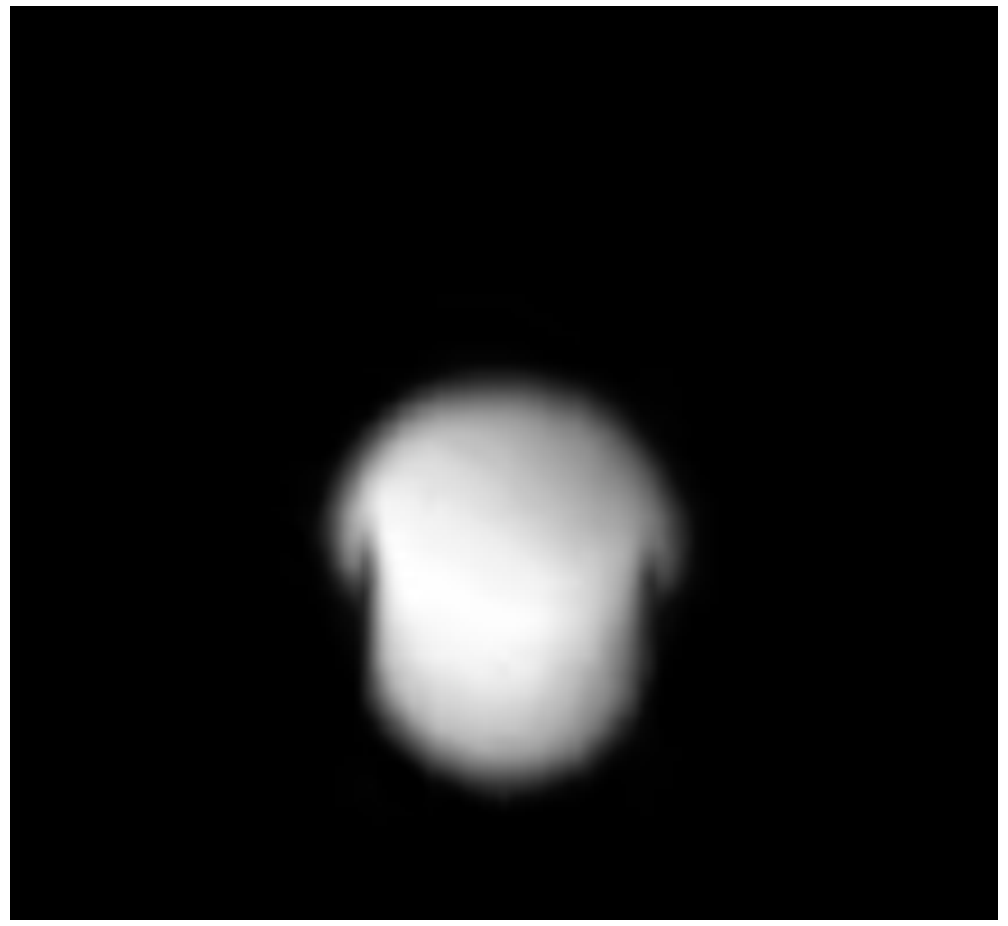
$$\frac{\partial d}{\partial t} + \mathbf{u} \cdot \nabla d = 0$$

Discretize the advection step $\mathscr{P}$:

- For simplicity, with a simple upwinding scheme
- Semi-Lagrangian or higher order schemes analogous

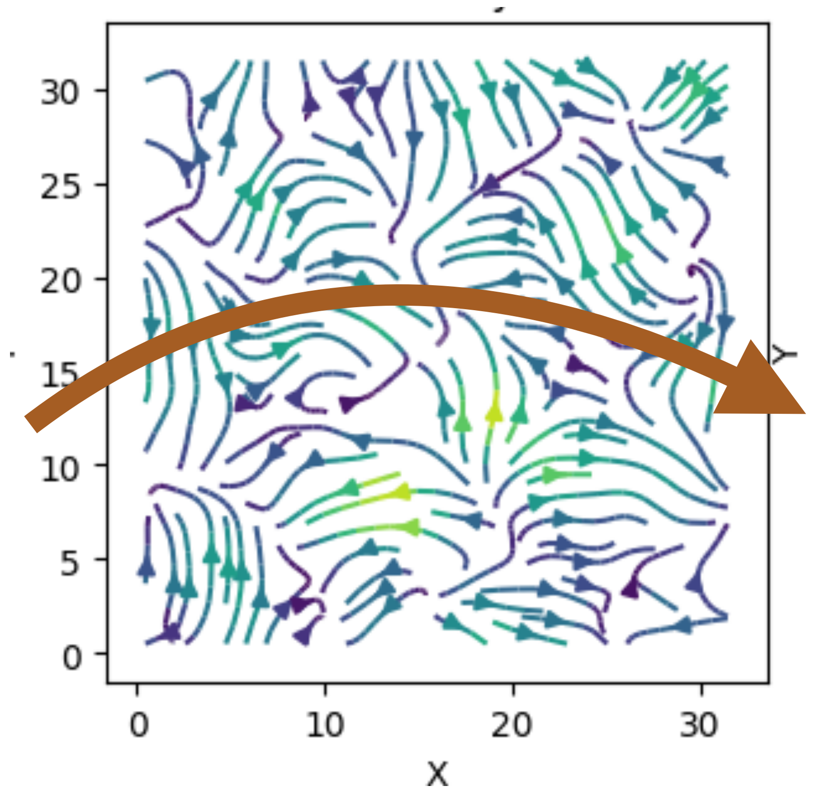
Example Problem with Explicit Update Step

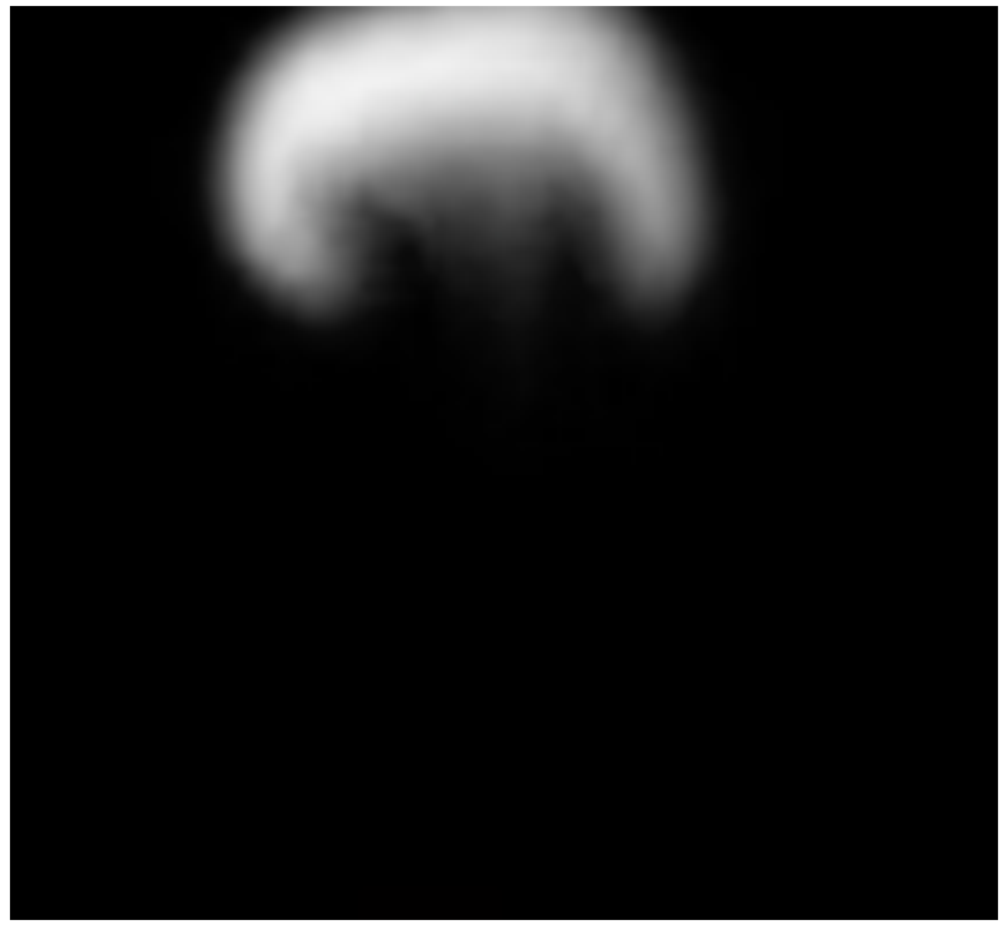
Find velocity such that density matches given target d^{target} at t^e, with $d(t^0) = d^0$



$d(t^0) = d^0$

u





$d(t^e) = d^{target}$

Advection

Example Problem with Explicit Update Step

Find velocity such that density matches given target d^{target} at t^e, with $d(t^0) = d^0$

Loss computed after one advection step: arg min $_{\mathbf{u}} |\mathcal{P}(d^0, \mathbf{u}, t^e) - d^{target}|^2$

Gradient for velocity $\Delta \mathbf{u} = \dfrac{\partial \mathcal{P}}{\partial \mathbf{u}}^T \dfrac{\partial L}{\partial \mathcal{P}}^T = \dfrac{\partial d}{\partial \mathbf{u}}^T \dfrac{\partial L}{\partial d}^T$

Loss Jacobian is simple enough: $\dfrac{\partial L}{\partial d} = \dfrac{\partial |\mathcal{P}(d^0, \mathbf{u}, t^e) - d^{target}|^2}{\partial d} = 2\left(d(t^e) - d^{target}\right)$

Jacobian of advection $\dfrac{\partial \mathcal{P}}{\partial \mathbf{u}}$ from discretized advection operator

Advection

Example Problem with Explicit Update Step

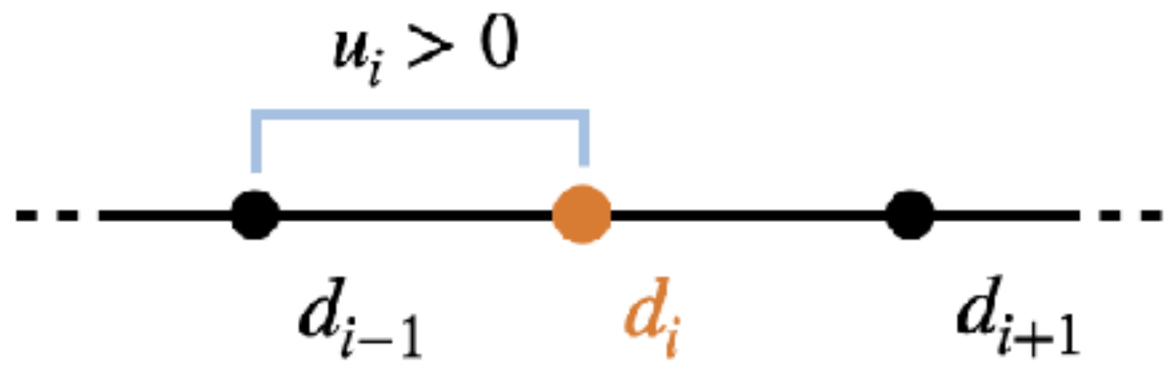
First-order upwinding scheme:

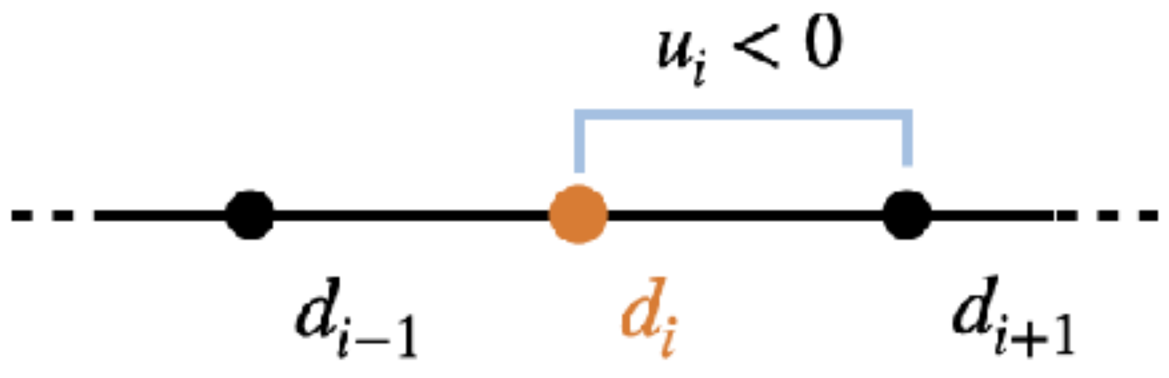
$$d_i(t + \Delta t) = d_i - \Delta t\left[u_i^+(d_{i+1} - d_i) + u_i^-(d_i - d_{i-1})\right] \text{ with}$$

$$u_i^+ = \min(u_i/\Delta x, 0)$$

$$u_i^- = \max(u_i/\Delta x, 0)$$

I.e., either forward or backward finite difference:





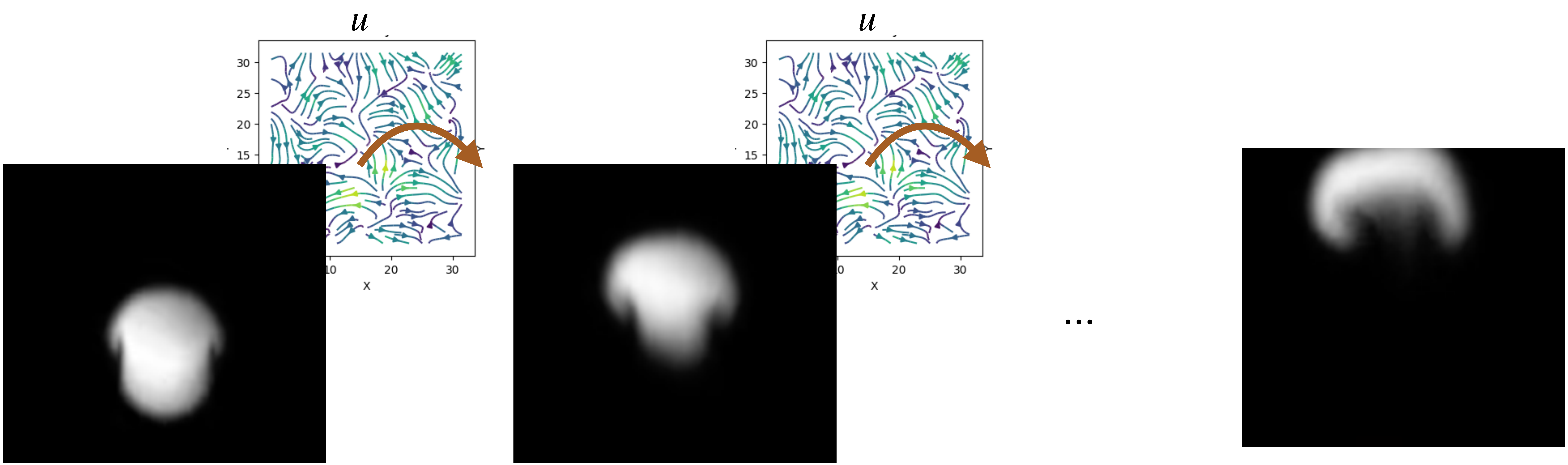
Negative velocity at i gives $\mathscr{P}(d_i(t), \mathbf{u}(t), t + \Delta t) = (1 + \frac{u_i \Delta t}{\Delta x})d_i - \frac{u_i \Delta t}{\Delta x} d_{i+1}$

Velocity derivative is simply $\partial \mathscr{P} / \partial u_i = \frac{\Delta t}{\Delta x} d_i - \frac{\Delta t}{\Delta x} d_{i+1}$

Advection

Multiple Simulation Steps

Find velocity so that density matches given target d^{target} at t^e, with $d(t^0) = d^0$ after applying velocity multiple times



$d(t^0) = d^0$ ... $d(t^e) = d^{target}$

Advection over Time

Multiple Simulation Steps

Simplified setup:

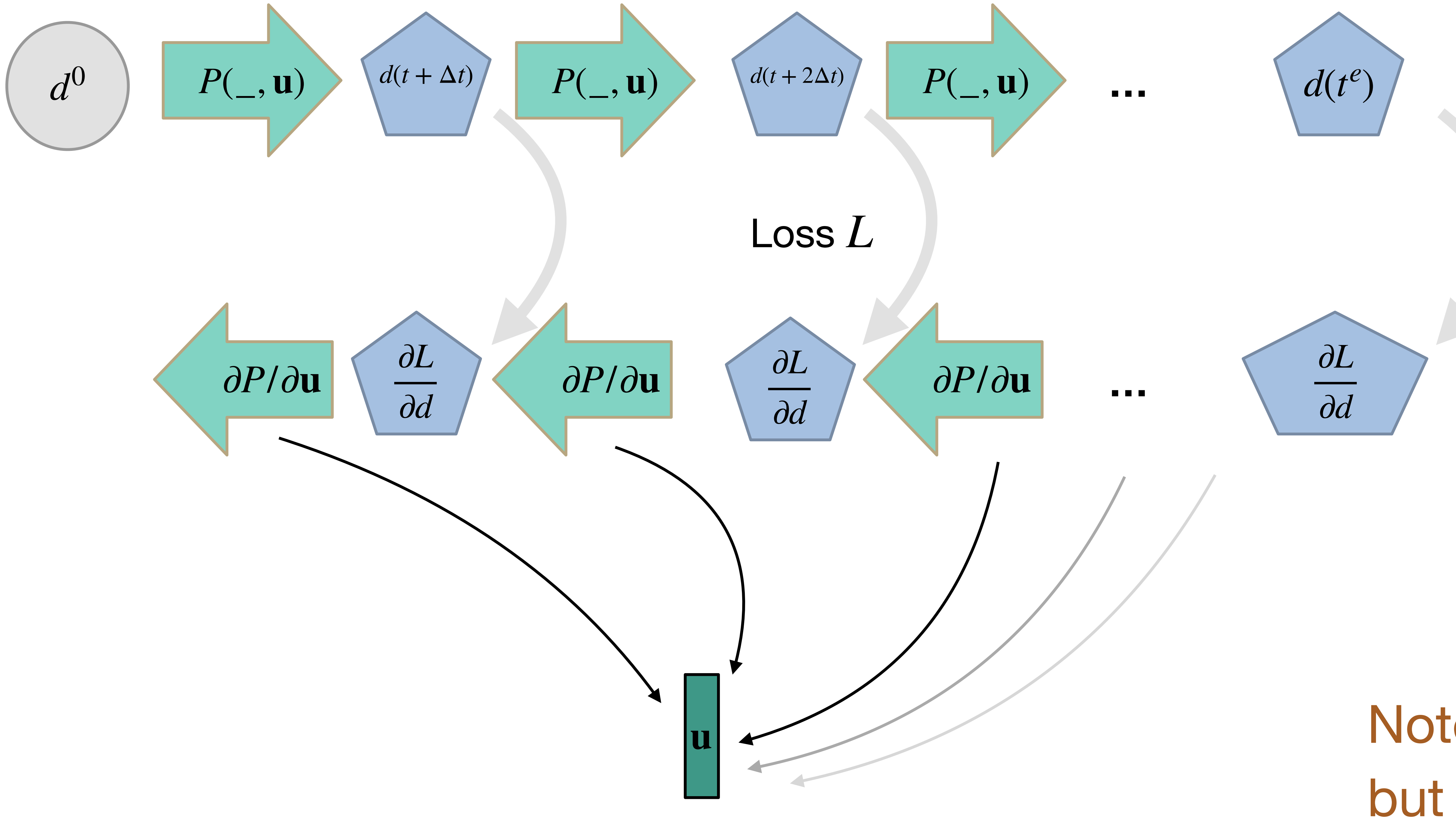
- Try to match target every step along the way, evaluate loss multiple times

- Keep single constant velocity and density target

Loss computed after multiple advection steps: $\arg\min_{\mathbf{u}} \sum_{s} |\mathcal{P}(d^{0}, \mathbf{u}, t^{s}) - d^{target}|^2$

Gives s loss terms each of which change $\mathbf{u}$

Multiple Simulation Steps - Schematic



Note: still no NN involved, but $\mathbf{u}$ used in each P !

Advection over Time

Term of velocity gradient only for s'th step (*transpose* of Jacobians omitted for brevity here!)

$$\Delta \mathbf{u}_s = \frac{\partial d(t^s)}{\partial \mathbf{u}} \frac{\partial L}{\partial d(t^s)}$$
$$+ \frac{\partial d(t^s - \Delta t)}{\partial \mathbf{u}} \frac{\partial d(t^s)}{\partial d(t^s - \Delta t)} \frac{\partial L}{\partial d(t^s)}$$
$$+ \cdots$$
$$+ \left(\frac{\partial d(t^0)}{\partial \mathbf{u}} \cdots \frac{\partial d(t^s - \Delta t)}{\partial d(t^s - 2\Delta t)} \frac{\partial d(t^s)}{\partial d(t^s - \Delta t)} \frac{\partial L}{\partial d(t^s)} \right)$$

Full gradient $\Delta \mathbf{u} = \sum_s \Delta \mathbf{u}_s$, e.g. for $s = 3$ we have $1 + 2 + 3 = 6$ terms…

⇨ Single gradient consists of $O(s^2)$ updates of $\mathbf{u}$

In line with NN training: treat each evaluation of $\mathscr{P}$ separately to compute Jacobian-vector product

Re-cap Fluid Simulations

Traditional Numerical Solver

Navier-Stokes equations

$$\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} = -\frac{1}{\rho}\nabla p + \nu\nabla\cdot\nabla\mathbf{u} \ , \ \ \nabla\cdot\mathbf{u} = 0$$

Two main steps via operator splitting:

- Advection - *done*
- Pressure projection - next
- Viscosity (either implicit / explicit, more of the same…)

Divergence Freeness Revisited

Example Problem with Implicit Update Step

Implicit step typically requires solving a linear system $\mathbf{u}^n = \mathbf{u} - \nabla p$

E.g., Poisson's equation for pressure $p = (\nabla^2)^{-1} b = A^{-1} b$

For fluids, right hand side is given by divergence: $b = \nabla \cdot \mathbf{u}$

Gradient for velocity, i.e. right hand side: $\dfrac{\partial p}{\partial b}\dfrac{\partial L}{\partial p} = A^{-1}\dfrac{\partial L}{\partial p}$, with $p = A^{-1} b$

Solver for the forward step can be re-used directly (applied to $\partial L / \partial p$)

Performance Considerations

Theoretically: Full iterative solver (e.g., Conjugate Gradient) could be implemented via matrix-vector ops of DL framework

Every op is tracked and intermediate state registered and stored (for backprop)

… slow and inefficient.

Rather: Use efficient numerical solver components for custom ops

(E.g., matrix inversion via fast CG solve in previous example.)

Note: We're essentially computing an *implicit derivative* here (cf. Implicit Function Theorem)

Differentiable Physics Example

(Compare to Burgers Simulation via PINN)

https://www.physicsbaseddeeplearning.org/diffphys-code-burgers.html

End