Efficient, Accurate and Stable Gradients for Neural Differential Equations

James Foster

University of Bath

Joint with Sam McCallum (Bath)

Outline

1. Neural Ordinary Differential Equations
2. Reversible ODE solvers
3. Towards more general reversible solvers
4. Preliminary experiments
5. Conclusion and future work
6. References

What is a neural differential equation?

These are differential equations where the vector field is parametrised as a neural network.

Standard example: Neural ODEs [1], due to Chen et al. (NeurIPS 2018).

$$\frac{dy}{dt} = f_\theta(t, y(t)),$$
$$y(0) = y_0,$$

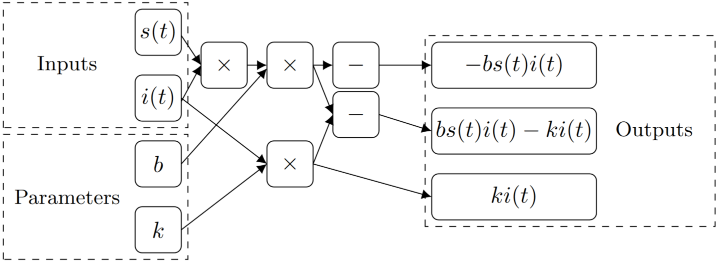
where f_θ can be any neural network (feedforward, convolutional, etc).

Examples of neural ordinary differential equations

A simple example: The SIR model for modelling infectious diseases

$$\frac{d}{dt}\begin{pmatrix} s(t) \\ i(t) \\ r(t) \end{pmatrix} = \begin{pmatrix} -bs(t)i(t) \\ bs(t)i(t) - ki(t) \\ ki(t) \end{pmatrix},$$

where b and k are parameters that are learnt from data.



At the other extreme, Neural ODEs have achieved 70% accuracy for ImageNet classification [2] (competitive with a well-tuned ResNet).

How to train your Neural ODE (backpropagation)

Step 1. Define a differentiable scalar loss function based on the data

$$L\big(y(t)\big) = L\Big(ODESolve\big(y(0), t, f_\theta\big)\Big).$$

Step 2. As "*ODESolve*" is a composition of differentiatiable operations, we can compute $\frac{dL}{d\theta}$ using automatic differentiation / backpropagation.
Step 3. Apply stochastic gradient descent (SGD) with $\frac{dL}{d\theta}$ to minimize L.

However...

When applying backpropagation, we store the full ODE trajectory $\{y_{t_k}\}$.

Thus, the memory cost scales linearly with the number of steps / depth.

How to train your Neural ODE (adjoint method)

Step 1. Define a differentiable scalar loss function based on the data

$$L\big(y(t)\big) = L\Big(ODESolve\big(y(0), t, f_\theta\big)\Big).$$

Step 2. Compute $L\big(y(T)\big)$ via ODE solver. Then $a(t) := \frac{\partial L(y(t))}{\partial y(t)}$ satisfies

$$\frac{da(t)}{dt} = -a(t)^\top \frac{\partial f_\theta\big(t, y(t)\big)}{\partial y}.$$

Step 3. Solve the above adjoint equation via ODE solver, and evaluate

$$\frac{dL}{d\theta} = \int_0^T a(t)^\top \frac{\partial f_\theta\big(t, y(t)\big)}{\partial \theta}\, dt.$$

Step 4. Apply stochastic gradient descent (SGD) with $\frac{dL}{d\theta}$ to minimize L.

Reconstruction and extrapolation of spirals with irregular time points (taken from [1])

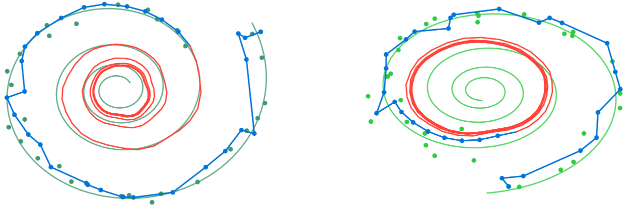
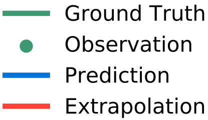


Figure: Recurrent Neural Network



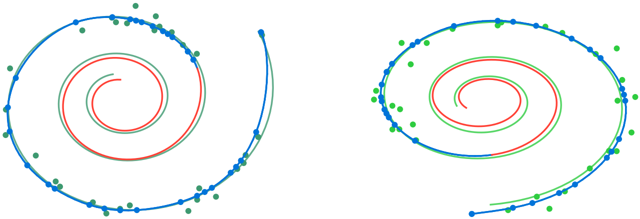


Figure: Neural ODE

Why Neural ODEs and the adjoint method?

- Flexible, includes “mechanistic” and “deep” models (+ hybrids [3])
- Continuous time, so well suited for handling (irregular) time series
- Choice of ODE solver allows trade-offs between accuracy and cost
- Adjoint method is memory efficient! (i.e. doesn’t scale with depth)

However...

Solving the ODE and its adjoint equation gives <u>inexact</u> gradients.

Outline

Reversible ODE solvers

We can compute gradients accurately using backpropagation – but that requires us to have the numerical ODE solution for the backwards pass.

In [2], it was shown that the numerical ODE solution can be dynamically recomputed (i.e. constant memory cost) using a reversible ODE solver.

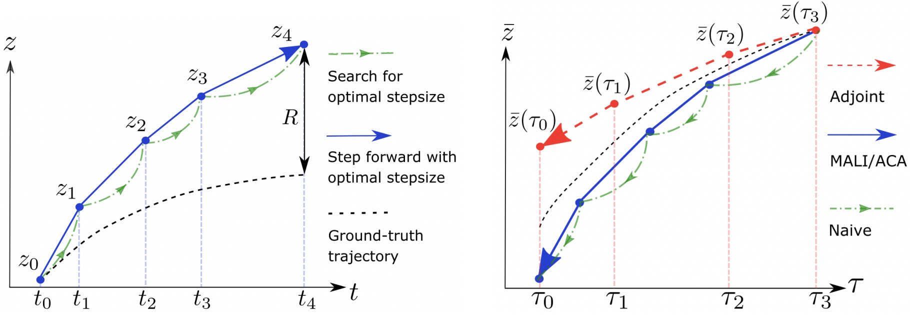


Figure: Illustration of a reversible ODE solver called “ALF” (taken from [2])

Reversible ODE solvers

Definition (ODE solver with order of convergence α)

We say an ODE solver $\Phi : \mathbb{R} \times \mathbb{R}^d \mapsto \mathbb{R}$ converges with order $\alpha > 0$ if

$$\|x(h) - \Phi_h(x)\| \leq C|h|^{\alpha+1},$$

where $x(h)$ is the solution at time $|h|$ of an ODE started at $x(0) := x$,

$$x' = f(x) \text{ if } h \geq 0, \qquad \text{or} \qquad x' = -f(x) \text{ if } h < 0.$$

Definition (Symmetric reversibility)

We say an ODE solver Φ is symmetric reversible if $\Phi_{-h}(\Phi_h(x)) = x$.

Example

For a general $f : \mathbb{R}^d \to \mathbb{R}^d$, Euler's method is not symmetric reversible.

$$(x + f_\theta(x)h) - f_\theta(x + f_\theta(x)h)h \neq x$$

Examples of reversible solvers

Example (Asynchronous Leapfrog Integrator (ICLR 2021))

$$
\begin{aligned}
X_{n+\frac{1}{2}} &:= X_n + \frac{1}{2}V_n h,\\
V_{n+1} &:= 2f\big(X_{n+\frac{1}{2}}\big) - V_n\,,\\
X_{n+1} &:= X_n + f\big(X_{n+\frac{1}{2}}\big)h,
\end{aligned}
$$

where $X_0 := x(0)$ and $V_0 := f(X_0)$.

Remark (Symmetric reversibility)

$$
\begin{aligned}
X_{n+\frac{1}{2}} &= X_{n+1} - \frac{1}{2}V_{n+1} h,\\
V_n &= 2f\big(X_{n+\frac{1}{2}}\big) - V_{n+1}\,,\\
X_n &= X_{n+1} - f\big(X_{n+\frac{1}{2}}\big)h.
\end{aligned}
$$

Examples of reversible solvers

Example (Reversible Heun's method (NeurIPS 2021))

$$Y_{n+1} := 2X_n - Y_n + f(Y_n)h,$$
$$X_{n+1} := X_n + \frac{1}{2}\big(f(Y_n) + f(Y_{n+1})\big)h,$$

where $X_0 = Y_0 = x(0)$.

Remark (Symmetric reversibility)

$$Y_n = 2X_{n+1} - Y_{n+1} - f(Y_{n+1})h,$$
$$X_n = X_{n+1} - \frac{1}{2}\big(f(Y_{n+1}) + f(Y_n)\big)h.$$

Examples of reversible solvers

Both methods...

- achieve reversibility by introducing extra state.
- have second order convergence with fixed step sizes.
- have a potentially unstable term of the form $2A - B$.
- have worked in large-scale applications:
  - A Neural ODE with the asynchronous leapfrog integrator achieved comparable performance to a ResNet-18 (≈ 11 million parameters) for classification on the ImageNet dataset [2].
  - A Neural SDE with the reversible Heun scheme was successfully used for turbulence modelling (≈ 4.6 million parameters) [4].
- can be defined for both ODEs and SDEs. However, in the SDE case, we could only prove convergence for the Reversible Heun scheme.

Examples of reversible solvers

However, [5] and [6] report that the reversible Heun method was too unstable for their applications.

Asynchronous Leapfrog Integrator	Reversible Heun method
$X_{n+\frac{1}{2}} := X_n + \frac{1}{2}V_n h,$ $V_{n+1} := 2f\big(X_{n+\frac{1}{2}}\big) - V_n,$ $X_{n+1} := X_n + \frac{1}{2}V_{n+1}h.$	$Y_{n+1} := 2X_n - Y_n + f(Y_n)h,$ $X_{n+1} := X_n + \frac{1}{2}\big(f(Y_n) + f(Y_{n+1})\big)h.$

We believe that any instability is then amplified by these solvers when

- V_n and $f(X_n)$ drift apart (for ALF)
- X_n and Y_n drift apart (for RH)

Outline

Towards more general reversible solvers

Given an ODE solver Φ_h, we define the map $\Psi_h(x) := \Phi_h(x) - x$ so that

$$\|x(h) - (x + \Psi_h(x))\| \leq C|h|^{\alpha+1},$$

where $x(h)$ is the solution at time h of the ODE started at $x(0) := x$.

Definition (Proposed reversible ODE solver [7])

We construct a numerical solution $\{(Y_n, Z_n)\}_{n\geq 0}$ by $Y_0 = Z_0 = x(0)$ and

$$\begin{aligned}
Y_{n+1} &:= \lambda Y_n + (1-\lambda)Z_n + \Psi_h(Z_n),\\
Z_{n+1} &:= Z_n - \Psi_{-h}(Y_{n+1}),
\end{aligned}$$

where $h > 0$ is the step size and $\lambda \in (0, 1]$ is a “coupling” parameter.

Towards more general reversible solvers

This approach is based on two ideas:

- Extra state allows for a reversible computation graph.
  (e.g. previous reversible solvers and coupling layers in neural nets)

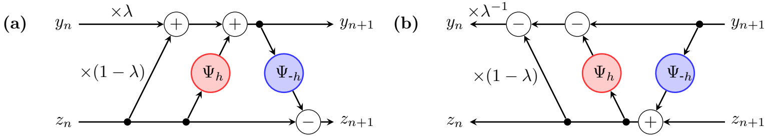


Figure: (a) Forwards ODE solve. (b) Backward ODE solve.

- ODE solvers can be applied with positive and negative step sizes.

$$\begin{aligned} x(h) \approx \Phi_h(x(0)) \;\text{“}\Rightarrow\text{”}\; & x(0) \approx \Phi_{-h}(x(h)) \\ \text{“}\Rightarrow\text{”}\; & x(0) \approx x(h) + \Psi_{-h}(x(h)) \\ \text{“}\Rightarrow\text{”}\; & x(h) \approx x(0) - \Psi_{-h}(x(0) + \Psi_h(x(0))). \end{aligned}$$

Towards more general reversible solvers

Recall the new solver is

$$Y_{n+1} := \lambda Y_n + (1-\lambda) Z_n + \Psi_h(Z_n),$$
$$Z_{n+1} := Z_n - \Psi_{-h}(Y_{n+1}).$$

The first key property to note is that this is algebraically reversible since

$$Z_n := Z_{n+1} + \Psi_{-h}(Y_{n+1}),$$
$$Y_n := \lambda^{-1} Y_{n+1} + (1-\lambda^{-1}) Z_n - \lambda^{-1}\Psi_h(Z_n).$$

Secondly, we introduce $\lambda \in (0,1]$ so that Y_n and Z_n stay close together,

$$Y_{n+1} - Z_{n+1} = \lambda(Y_n - Z_n) + \underbrace{\Psi_h(Z_n) + \Psi_{-h}(Y_{n+1})}_{\text{small if } Z_n \approx x(t_n) \text{ and } Y_{n+1} \approx x(t_{n+1})}.$$

But if λ is too small, it may cause instabilities on the backwards solve.

Towards more general reversible solvers

Theorem (Main result; any ODE solver can made reversible [7])

Suppose Ψ corresponds to an α-order numerical method for the ODE

$$x' = f(x),$$

where $t \in [0, T]$ for a fixed T. Then under a Lipschitz assumption on Ψ, there exists constants $C, h_{\max} > 0$ such that

$$\big\| Y_k - x(t_k) \big\| \leq C h^{\alpha}, \tag{1}$$

for all $k \in \{0, 1, \cdots, N\}$ where $h \in (0, h_{\max}]$, $t_k := kh \in [0, T]$ and

$$\begin{aligned} Y_{n+1} &:= \lambda Y_n + (1-\lambda) Z_n + \Psi_h(Z_n), \\ Z_{n+1} &:= Z_n - \Psi_{-h}(Y_{n+1}), \end{aligned}$$

with $\lambda \in (0, 1]$ and $Y_0 = Z_0 = x(0)$.

Stability of reversible ODE solvers

Although we can construct arbitrarily high order ODE reversible solvers, we have not yet addressed the main challenges which concern stability.

Definition (A-stability region)

Consider the following linear ODE,

$$\begin{aligned} y' &= \alpha y, \\ y(0) &= 1, \end{aligned} \tag{2}$$

where $\alpha \in \mathbb{C}$ with $\mathrm{Re}(\alpha) < 0$. A numerical solution $Y = \{Y_k\}_{k\geq 0}$ of (2) is said to be A-stable at α if $Y_k \to 0$ as $k \to \infty$. The stability region is

$$R = \{\alpha \in \mathbb{C} : \mathrm{Re}(\alpha) < 0 \text{ and } Y = \{Y_k\} \text{ is A-stable at } \alpha\}.$$

The Asynchronous Leapfrog Integrator and Reversible Heun method are not A-stable (for any $\alpha \in \mathbb{C}$).

Stability of reversible ODE solvers

Numerically computing stability regions gives some promising results:

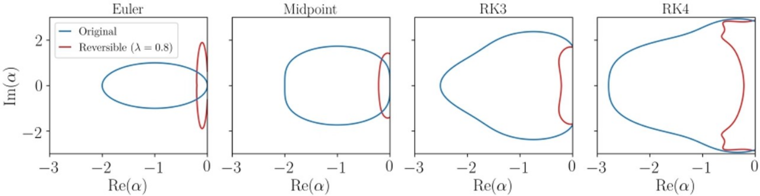


Figure: Stability regions for different reversible schemes ($h = 1$ and $\lambda = 0.8$).

We also see that decreasing $\lambda \in (0, 1]$ increases the stability region. However, if λ is too small, then the backwards solve may be unstable.

Theoretically, we have only been able to find a closed-form expression for the real part of these stability regions [7].

Outline

Preliminary experiments

We first generate synthetic time series data $\{x(t_i)\}_{i\geq 0}$ by simulating Chandrasekhar's white dwarf equation,

$$\begin{aligned}\frac{dx}{dt} &= v,\\ \frac{dv}{dt} &= -\frac{2}{t}v - \left(x^2 - C\right)^{\frac{3}{2}},\end{aligned}$$

where $(x(0), v(0)) := (1, 0)$.

We then train a Neural ODE using $\{x(t_i)\}$ to identify the above system.

In particular, we will compare against backpropagation with online recursive checkpointing. In these examples, we will set $\lambda = 0.99$.

Preliminary experiments

Method	Loss ($\times 10^{-3}$)	Time (s)	Memory (effective checkpoints)
Reversible	0.122	$\mathbf{1.90 \pm 0.04}$	**2**
Checkpointing	0.122	282.43 ± 16.73	**2**
Checkpointing	0.122	31.41 ± 0.47	4
Checkpointing	0.122	10.14 ± 0.16	8
Checkpointing	0.122	8.52 ± 0.47	16
Checkpointing	0.122	7.61 ± 0.12	32
Checkpointing	0.122	4.87 ± 0.07	44

Table: Time and memory cost incurred when training a Neural ODE to identify Chandrasekhar's white dwarf equation (1000 time and training steps).

Preliminary experiments

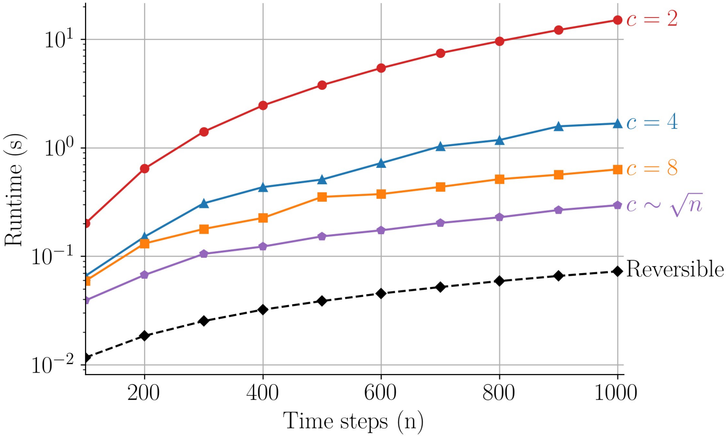


Figure: Combined runtime of a forwards solve and backpropagation through the midpoint ODE solver over n time steps. Here, we compare against backpropagation with online recursive checkpointing at c checkpoints.

Preliminary experiments

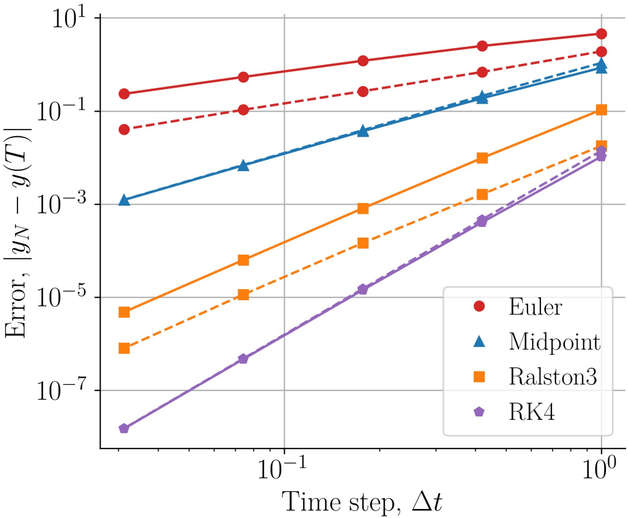


Figure: Convergence of original (solid) and reversible (dashed) ODE solvers.

Preliminary experiments

We have also tested our approach on a continuous normalising flow [8] and a neural controlled differential equation [9].

In both examples, we see similar performance compared to standard backpropagation – but with much less memory required for training.

Solver	$-\mathbb{E}\left[\log p_\theta\right]$		Memory Usage (GB)	
	Reversible	Backprop	Reversible	Backprop
Midpoint	0.888	0.891	**0.563**	3.922
RK4	0.890	0.890	**0.647**	7.467
Dopri5	0.890	0.891	**0.704**	12.79

Table: Continuous Normalising Flow on the two moons dataset [10].

Preliminary experiments

We have also tested our approach on a continuous normalising flow [8] and a neural controlled differential equation [9].

In both examples, we see similar performance compared to standard backpropagation – but with much less memory required for training.

Solver	Accuracy (%)		Memory Usage (GB)	
	Reversible	Backprop	Reversible	Backprop
Midpoint	78.4 ± 5.5	78.9 ± 6.7	**0.434**	1.09
RK4	79.0 ± 5.9	76.4 ± 5.4	**0.468**	1.86
Dopri5	80.1 ± 6.9	77.9 ± 6.7	**0.523**	3.01

Table: Neural CDE on the CharacterTrajectories dataset [11].

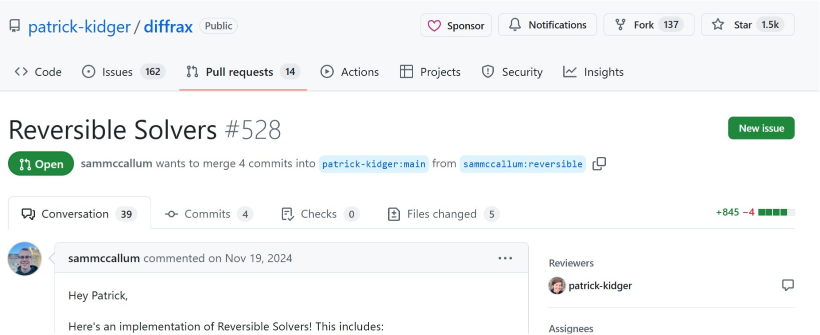
Outline

Conclusion

- Among the recent advances in neural differential equations, reversible solvers have seen utility due to the accurate and memory-efficient gradients that they provide during training.
- However, the current reversible NDE solvers have stability issues. We believe that this instability is amplified by the “$2A - B$” terms.
- We propose an approach in which an explicit ODE solver can be converted to a reversible one with the same order of convergence. Although this requires twice the function evaluations per step, we often observe faster training times due to the memory reduction.
- The reversible solvers produce stability regions and have shown promising empirical results – including against checkpointing.

Future work

- Implementation of our method into the ODE/SDE/CDE library "Diffrax" (github.com/patrick-kidger/diffrax):



- Applications of reversible solvers for learning time-evolving PDEs (which can easily have a high memory footprint).

Thank you
for your attention!

and our preprint can be found at:

Sam McCallum and James Foster. *Efficient, Accurate and Stable Gradients for Neural ODEs*, arxiv:2410.11648, 2024.

Outline

References I

R. T. Q. Chen, Y. Rubanova, J. Bettencourt and D. Duvenaud. *Neural Ordinary Differential Equations*, Neural Information Processing Systems, 2018.

J. Zhuang, N. C. Dvornek, S. Tatikonda and J. S. Duncan. *MALI: A memory efficient and reverse accurate integrator for Neural ODEs*, International Conference on Learning Representations, 2021.

C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar, D. Skinner, A. Ramadhan and A. Edelman. *Universal Differential Equations for Scientific Machine Learning*, arXiv:2001.04385, 2020.

References II

- A. Boral, Z. Yi Wan, L. Zepeda-Núñez, J. Lottes, Q. Wang, Y. Chen, J. R. Anderson and F. Sha. *Neural Ideal Large Eddy Simulation: Modeling Turbulence with Neural Stochastic Differential Equations*, Neural Information Processing Systems, 2023.

- Q. Zhang and Y. Chen. *Path Integral Sampler: A Stochastic Control Approach For Sampling*, International Conference on Learning Representations, 2022.

- A. Howe. *Possible issue with ReversibleHeun solver instability*, github.com/patrick-kidger/diffrax/issues/417, 2024.

- S. McCallum and J. Foster. *Efficient, Accurate and Stable Gradients for Neural ODEs*, arXiv:2410.11648, 2024.

References III

W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, D. Duvenaud. *NFFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models*, International Conference on Learning Representations, 2019.

P. Kidger, J. Morrill, J. Foster and T. Lyons. *Neural Controlled Differential Equations for Irregular Time Series*, Neural Information Processing Systems, 2020.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, et al. *Scikit-learn: Machine learning in Python*, Journal of Machine Learning Research, 2011.

A. Bagnall, H. A. Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam and E. Keogh. *The UEA multivariate time series classification archive, 2018*, arXiv:1811.00075, 2018.

Examples of reversible solvers

Turbulence modelling is computationally demanding due to the fine mesh and steps used to approximate the PDE. A transformer-based Neural SDE model was recently developed for such simulations [4], and was numerically discretized using the Reversible Heun method.

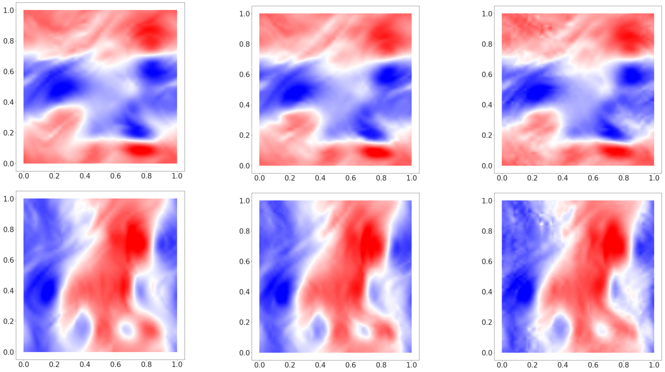


Figure: PDE simulation (left), Neural SDE (middle) and Neural network (right)