Neural ODEs

Training a ResNet

- So far we studied how to compute gradients in ODE-based functions
- Let's parameterise an ODE function so that it becomes a deep neural network

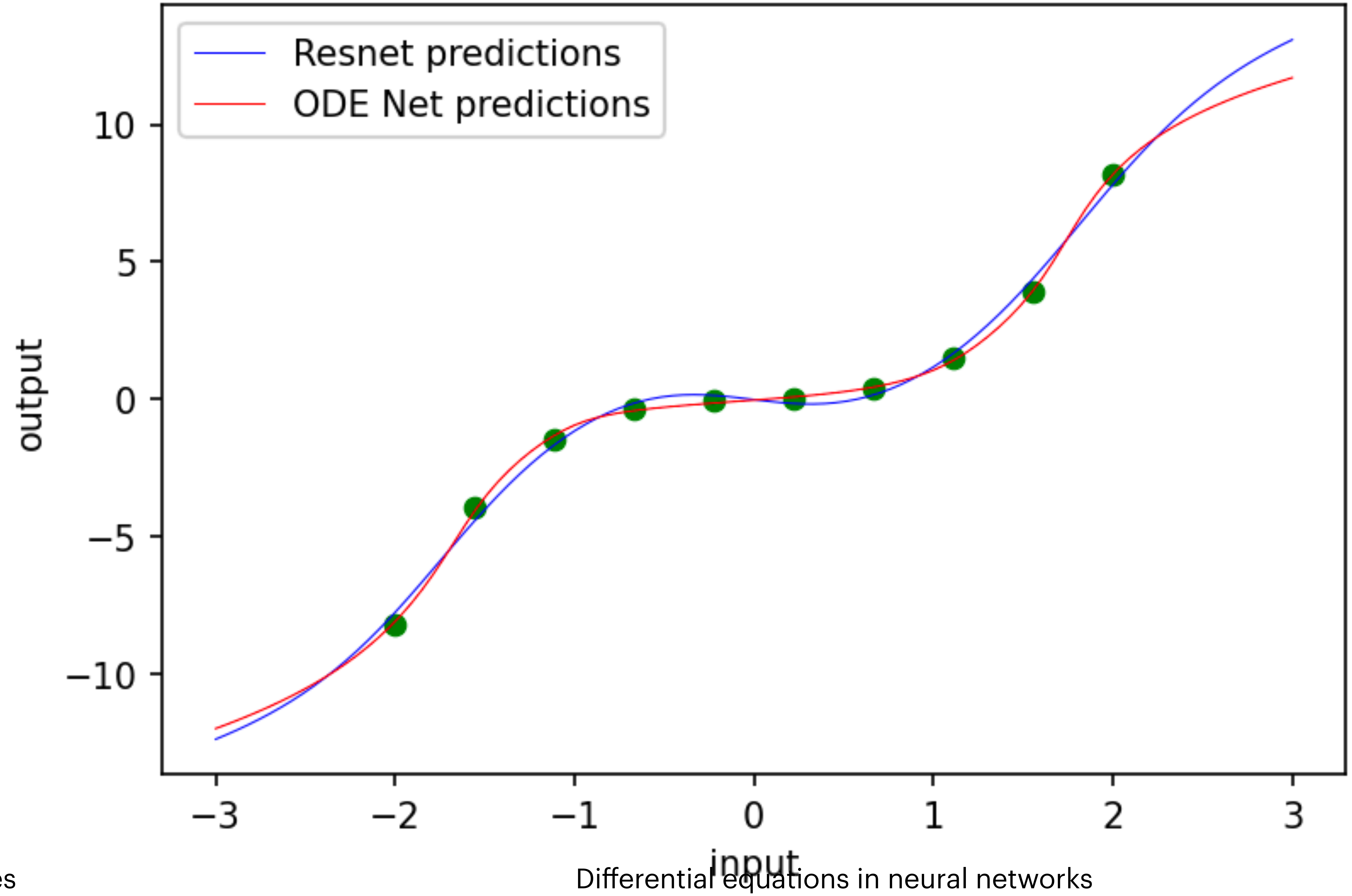
$$\partial_t y(t) = f(y(t), t, \theta), \text{ where } y(0) = y_0$$

- Parameters θ is what we called a before, that is the parameters that define f
- We must first implement a "dynamics" function, which takes the current state $y(t)$, the time t, and the parameters θ, and returns as output the $\partial_t y(t)$
- Then, we run it through and ODE solver (e.g., via `odeint` in JAX)

Batching a Neural ODE

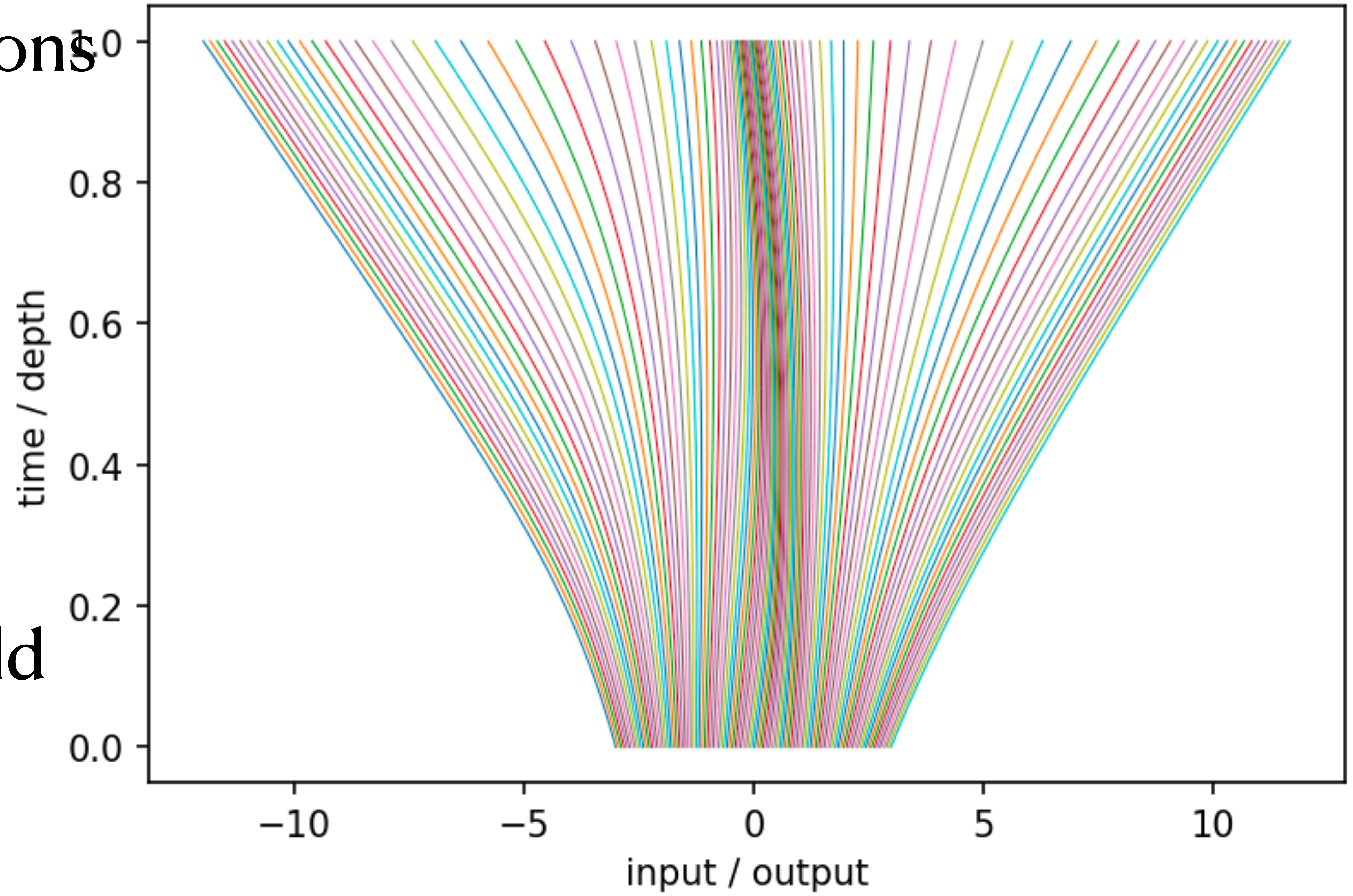
- In sophisticated frameworks you can automatically add batch dimensions (*e.g.*, in JAX using `vmap`)
- When not available, you can consider that each sample in your batch is an independent ODE, so they can be solved simultaneously
- Create an "giant"-ODE, concatenating all per sample ODEs per sample, and solve all with a single call to `odeint`

ResNet v. ODE Net



Activation trajectories

- In a deep ResNet we can check the activations per layer
- With ODE we can plot activations as trajectories over time
- Notice that the trajectories never cross (homeomorphism), which is a limitation as data can be on an entangled latent manifold
- Using auxiliary dimensions helps*



* Dupont, Doucet, Teh, Augmented Neural ODEs, 2019

What kind of dynamics?

- Almost any tractable, differentiable, parametric function can work for f
- That is, the dynamics inserted to the ODE solver can be almost any other layer
  - ConvNet
  - U-net
  - Transformer

Where to use Neural ODEs?

- Similar to where other neural networks are used
- In practice, they do not always get the same as good performance
- However, similar works like score-based matching do get state-of-the-art
- Neural ODE based functions are very strong with continuous-time settings, where the time is not "discrete" anymore
- Tractable change of variables by taking the continuous limit of a discrete process, similar to Normalising Flows
- Learning smooth homeomorphisms (*e.g.*, non-intersecting shapes)

Memory savings

- Memory savings. With ODE-based function we need only to know an initial point and can reconstruct forward/backward trajectory with constant memory cost
- In theory, forward is $\partial_t y(t) = f(y(t), t, \theta)$, backward simply $\partial_t y(t) = -f(y(t), -t, \theta)$
- In practice, if the dynamics are hard to solve, the two paths might not match or it would be too expensive (increase time resolution) to do so accurately
- In that case, some checkpointing of intermediate states can help
- With neural networks the dynamics are usually easy enough to solve

Adaptive computations

- ODE-based neural networks can spend only as many computations as needed
- The reason is that ODE solver decides itself when it has converged
- And ODE solvers were being developed for the past 120 years, so there is some experience in the domain
- The simpler the dynamics, the simpler ODE solver is needed, and fewer time steps to solve the trajectory

Trade-off precision v. speed

- Continuing on adaptive computations, with ODE-Nets one can also define how much error can they tolerate
- No need to solve the problem very accurately if not needed, and one can save time in the meantime
- That makes a lot of sense for learning algorithms, where generalisation is as important as accuracy
- Importantly, this trade-off can be also defined at both training and test time, so higher flexibility

Disadvantages of Neural ODEs

- Often slower as they require many more time steps than one would need layers
- Speed can also be hurt if the model tends to learn more complex dynamics than needed due to the nature of the data, although solutions exist*
- Some more hyperparameters than usual, like which solver or what error tolerance

* Kelly et al, Learning Differential Equations that are Easy to Solve, 2020
Finlay et al., How to train your neural ODE: the world of Jacobian and kinetic regularization, 2020

Stochastic and Partial DEs

- We can have also other types of differential equations as layers
- Stochastic differential equations
- Partial differential equations
- ICLR Workshop on Integration of Deep NN and Differential Equations

Li et al, Scalable Gradients for Stochastic Differential Equations, 2020
Beatson et al., Learning Composable Energy Surrogates for PDE Order Reduction, 2020
Sue et al., Amortized Finite Element Analysis for Fast PDE-Constrained Optimization, 2020

Conclusion

- Introduction to implicit layers
- Forward propagation with implicit layers
- Automatic differentiation with implicit layers
- Neural ordinary differential equations