Jacobian norm regularisation and conditioning in neural ordinary differential equations

Shane Josias[1,2] Willie Brink[1]

[1]Applied Mathematics, Mathematical Sciences, Stellenbosch University

[2]School for Data Science and Computational Thinking, Stellenbosch University

06 December 2022

What to expect

Overview of neural ordinary differential equations (ODEs)

- learnable input-output mapping defined as the solution to an ODE

What to expect

Overview of neural ordinary differential equations (ODEs)

- learnable input-output mapping defined as the solution to an ODE

Neural ODE challenges and Jacobian regularisation

What to expect

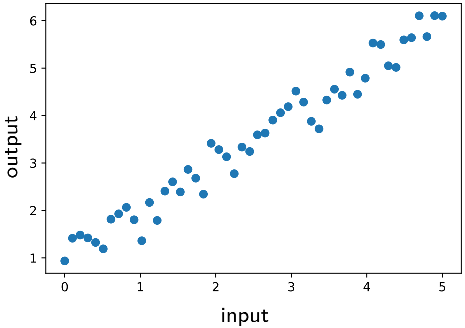
Overview of neural ordinary differential equations (ODEs)

- learnable input-output mapping defined as the solution to an ODE

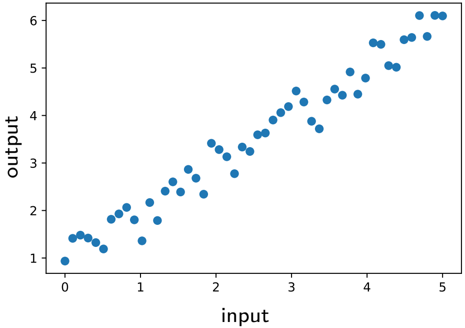
Neural ODE challenges and Jacobian regularisation

Review selected results

Learnable input-output mapping

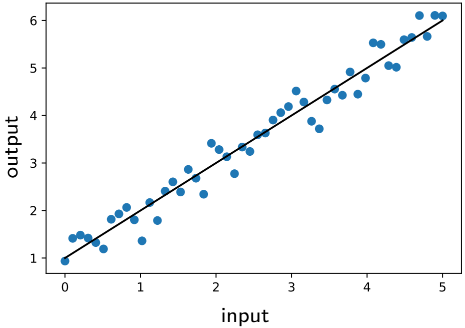


Learnable input-output mapping



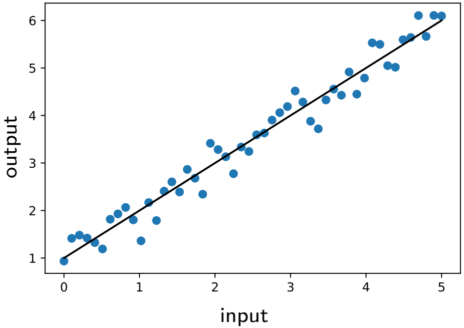
1. Choose a function class
$$f(x) = wx + b$$

Learnable input-output mapping



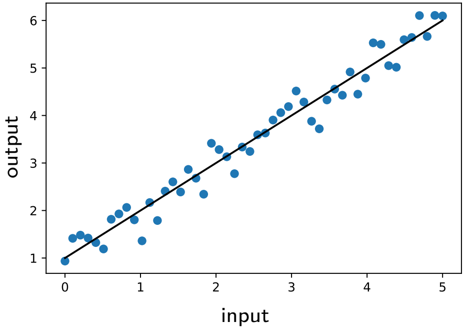
1. Choose a function class
$$f(x) = wx + b$$
2. Determine parameters w and b via gradient based optimisation.

Learnable input-output mapping



1. Choose a function class
$$f(x) = wx + b$$
2. Determine parameters w and b via gradient based optimisation.
3. Done by defining an objective function (error).

Learnable input-output mapping

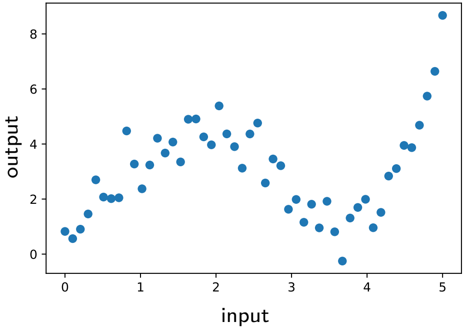


1. Choose a function class
$$f(x) = wx + b$$
2. Determine parameters w and b via gradient based optimisation.
3. Done by defining an objective function (error).

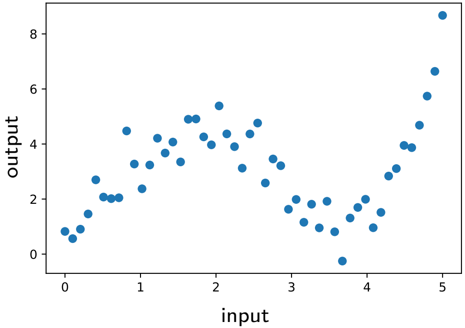
Regularisation adds a penalty to the objective.

- faster convergence, better generalisation

Learnable input-output mapping: neural networks



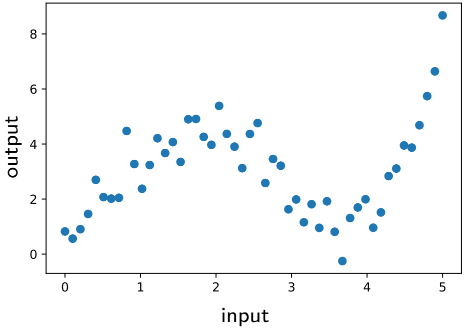
Learnable input-output mapping: neural networks



1. Choose a function class

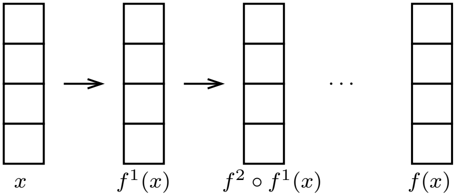
$$f(x) = f^n \circ f^{n-1} \circ \cdots \circ f^1(x)$$

Learnable input-output mapping: neural networks

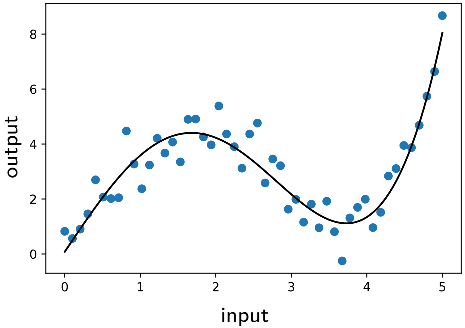


1. Choose a function class

$$f(x) = f^n \circ f^{n-1} \circ \cdots \circ f^1(x)$$

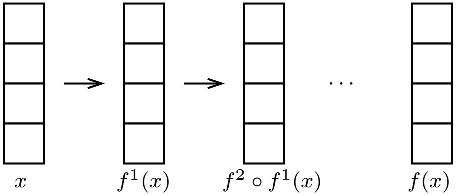


Learnable input-output mapping: neural networks



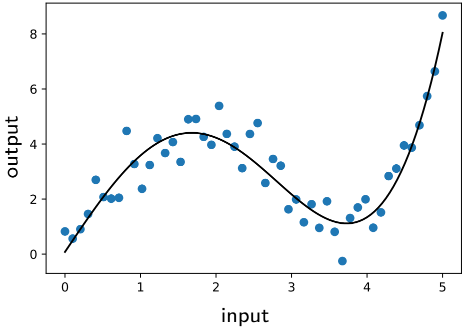
1. Choose a function class

$$f(x) = f^n \circ f^{n-1} \circ \cdots \circ f^1(x)$$



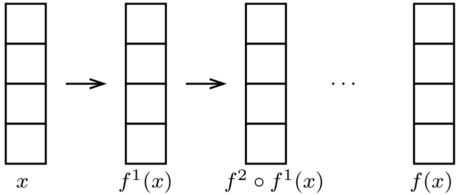
2. Determine parameters via gradient-based optimisation.

Learnable input-output mapping: neural networks



1. Choose a function class

$$f(x) = f^n \circ f^{n-1} \circ \cdots \circ f^1(x)$$

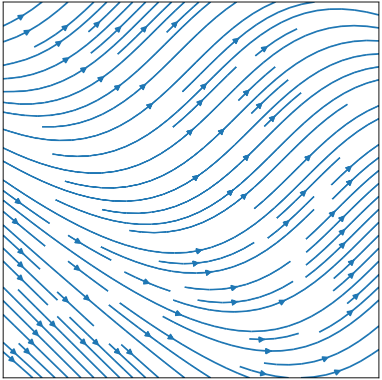


2. Determine parameters via gradient-based optimisation.

Regularisation adds a penalty to the objective.

- faster convergence, better generalisation

Neural ordinary differential equations

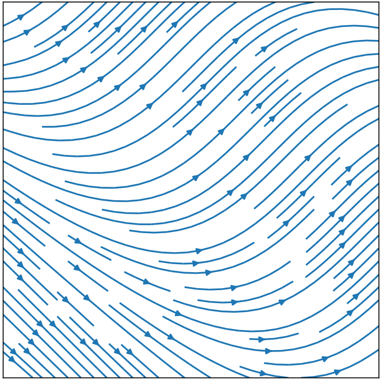


A vector $\boldsymbol{h}(t)$ follows the dynamics f:

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t)$$

Neural ordinary differential equations

$\boldsymbol{h}(t_0)$

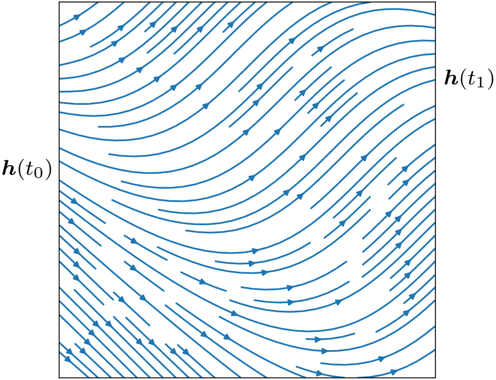


A vector $\boldsymbol{h}(t)$ follows the dynamics f:

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t)$$

For an input $\boldsymbol{h}(t_0)$ determine output as

Neural ordinary differential equations



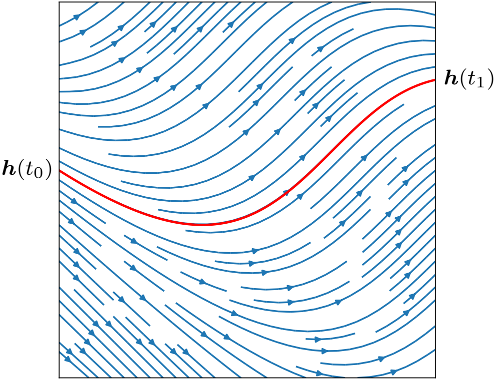
A vector $\boldsymbol{h}(t)$ follows the dynamics f:

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t)$$

For an input $\boldsymbol{h}(t_0)$ determine output as

$$\boldsymbol{h}(t_1) = \boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t)\, dt$$

Neural ordinary differential equations



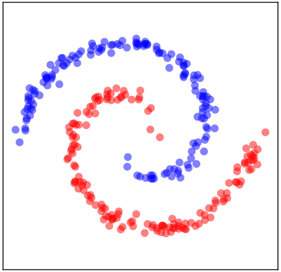
A vector $\boldsymbol{h}(t)$ follows the dynamics f:

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t)$$

For an input $\boldsymbol{h}(t_0)$ determine output as

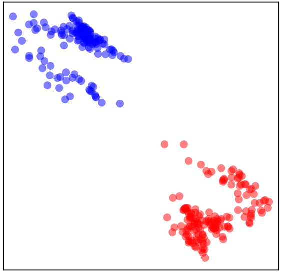
$$\boldsymbol{h}(t_1) = \boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t)\, dt$$

An example: binary classification



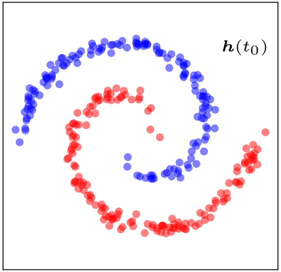
input

use a neural network
as the function

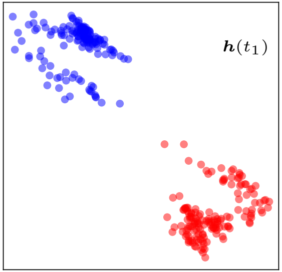


output

An example: binary classification

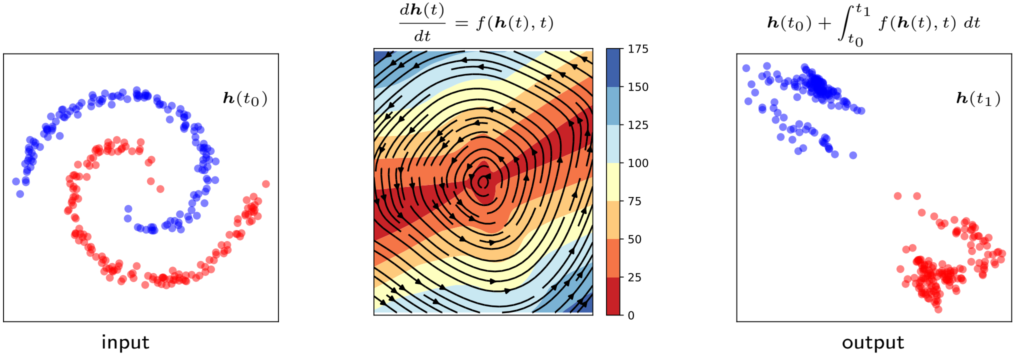


input

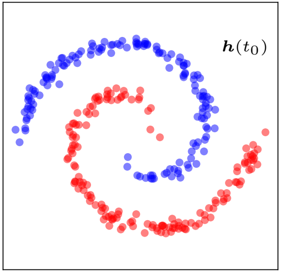


output

An example: binary classification

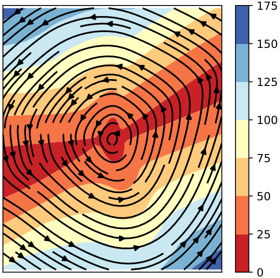


An example: binary classification

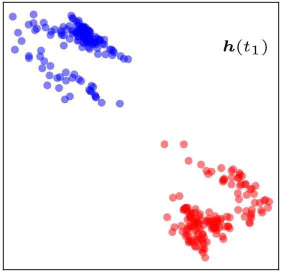


input

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t)$$



$$\boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t)\ dt$$



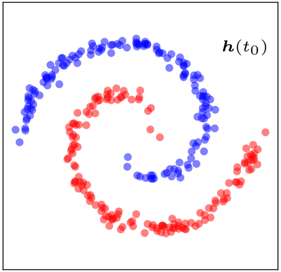
output

More generally useful for

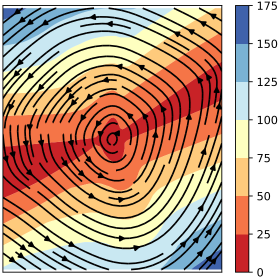
An example: binary classification

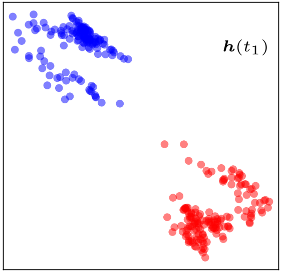
$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t)$$

$$\boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t)\, dt$$



input





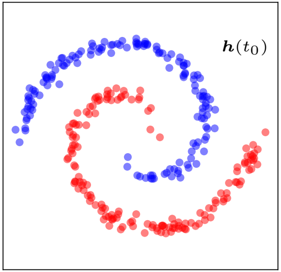
output

More generally useful for

1. Modelling data from continuous-time systems

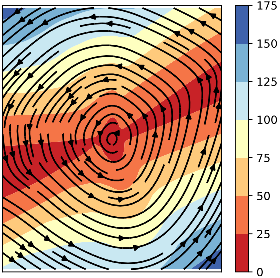
dynamical systems, time-series

An example: binary classification

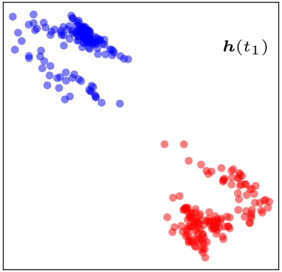


input

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t)$$



$$\boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t)\ dt$$

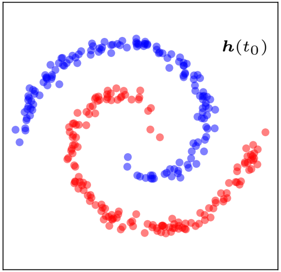


output

More generally useful for

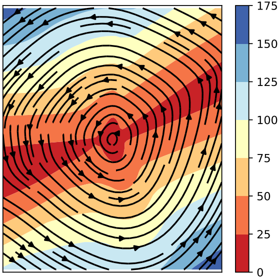
1. Modelling data from continuous-time systems — dynamical systems, time-series
2. Continuous normalising flows for density estimation

An example: binary classification

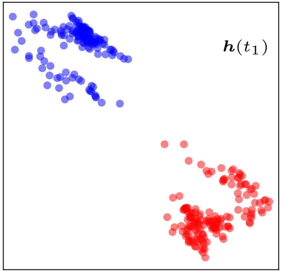


input

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t)$$



$$\boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t)\ dt$$

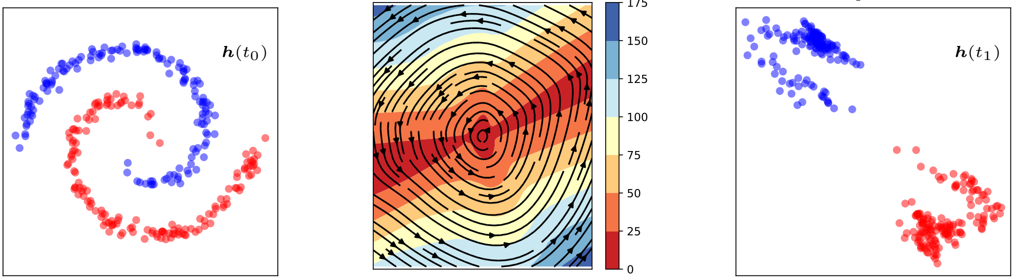


output

We care about

An example: binary classification

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t) \qquad\qquad \boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t)\ dt$$



input

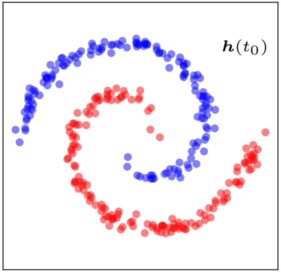
output

We care about

1. Generalisation and robustness to input perturbations

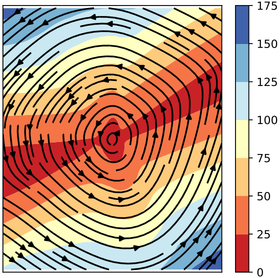
in high dimensions

An example: binary classification

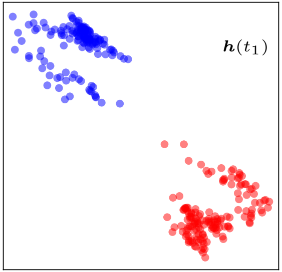


input

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t)$$



$$\boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t)\, dt$$



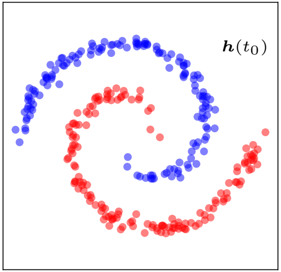
output

We care about

1. Generalisation and robustness to input perturbations
2. Computational efficiency

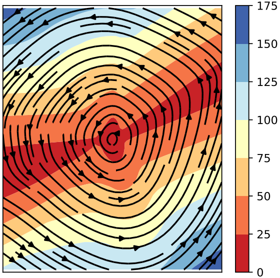
in high dimensions

An example: binary classification

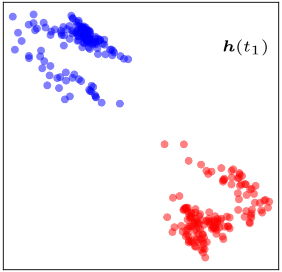


input

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t)$$



$$\boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t)\ dt$$



output

We care about

1. Generalisation and robustness to input perturbations — in high dimensions
2. Computational efficiency

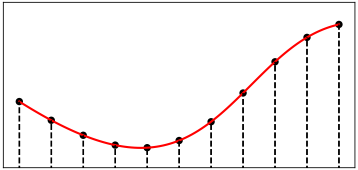
How should we regularise our objective function when training neural ODEs?

The problem: rising number of function evaluations (NFE)

$$\boldsymbol{h}(t_1) = \boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t) \, dt$$

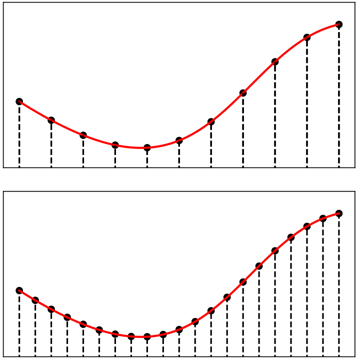
The problem: rising number of function evaluations (NFE)

$$\boldsymbol{h}(t_1) = \boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t)\, dt$$



The problem: rising number of function evaluations (NFE)

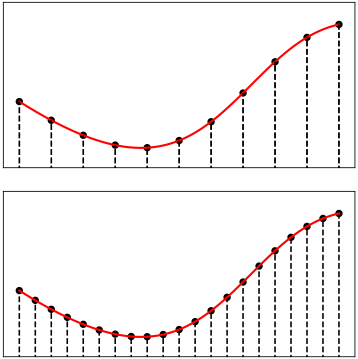
$$\boldsymbol{h}(t_1) = \boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t)\, dt$$



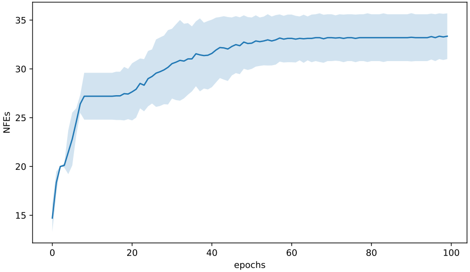
higher accuracy requires higher NFE

The problem: rising number of function evaluations (NFE)

$$\boldsymbol{h}(t_1) = \boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t)\, dt$$



higher accuracy requires higher NFE



NFE rises during training

Comments on conditioning

"poorly conditioned dynamics will lead to difficulties during numerical integration"

Finlay et al. How to train your neural ODE: the world of Jacobian and kinetic regularization, 2020.

Comments on conditioning

"poorly conditioned dynamics will lead to difficulties during numerical integration"

Finlay et al. How to train your neural ODE: the world of Jacobian and kinetic regularization, 2020.

"flows that need to stretch and squeeze the input space in such a way are likely to lead to ill-posed ODE problems that are numerically expensive to solve"

Dupont et al. Augmented Neural ODEs, 2019.

Jacobian norm regularisation

If $\boldsymbol{h}(t) = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}$, and $f(\boldsymbol{h}(t), t) = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$, then $\boldsymbol{J} = \begin{bmatrix} \frac{\partial f_1}{\partial h_1} & \frac{\partial f_1}{\partial h_2} \\ \frac{\partial f_2}{\partial h_1} & \frac{\partial f_2}{\partial h_2} \end{bmatrix}$

Jacobian norm regularisation

If $\boldsymbol{h}(t) = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}$, and $f(\boldsymbol{h}(t), t) = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$, then $\boldsymbol{J} = \begin{bmatrix} \frac{\partial f_1}{\partial h_1} & \frac{\partial f_1}{\partial h_2} \\ \frac{\partial f_2}{\partial h_1} & \frac{\partial f_2}{\partial h_2} \end{bmatrix}$

In general: $\boldsymbol{J} = \nabla_{\boldsymbol{h}(t_0)} f(\boldsymbol{h}(t), t)$

Jacobian norm regularisation

If $\boldsymbol{h}(t) = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}$, and $f(\boldsymbol{h}(t), t) = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$, then $\boldsymbol{J} = \begin{bmatrix} \frac{\partial f_1}{\partial h_1} & \frac{\partial f_1}{\partial h_2} \\ \frac{\partial f_2}{\partial h_1} & \frac{\partial f_2}{\partial h_2} \end{bmatrix}$

In general: $\boldsymbol{J} = \nabla_{\boldsymbol{h}(t_0)} f(\boldsymbol{h}(t), t)$

$\|\boldsymbol{J}\|_F = \sqrt{\sum_{i=1}^{d} \sum_{j=1}^{d} |\boldsymbol{J}_{i,j}|^2} \downarrow$

Frobenius: neural ODE

Jacobian norm regularisation

If $\boldsymbol{h}(t) = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}$, and $f(\boldsymbol{h}(t), t) = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$, then $\boldsymbol{J} = \begin{bmatrix} \frac{\partial f_1}{\partial h_1} & \frac{\partial f_1}{\partial h_2} \\ \frac{\partial f_2}{\partial h_1} & \frac{\partial f_2}{\partial h_2} \end{bmatrix}$

In general: $\boldsymbol{J} = \nabla_{\boldsymbol{h}(t_0)} f(\boldsymbol{h}(t), t)$

$\|\boldsymbol{J}\|_F = \sqrt{\sum_{i=1}^{d} \sum_{j=1}^{d} |\boldsymbol{J}_{i,j}|^2} \downarrow$ Frobenius: neural ODE

$\|\boldsymbol{J}\|_2 = \sigma_{\text{max}}(\boldsymbol{J}) \downarrow$ spectral: neural network

Jacobian norm regularisation

If $\boldsymbol{h}(t) = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}$, and $f(\boldsymbol{h}(t), t) = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$, then $\boldsymbol{J} = \begin{bmatrix} \frac{\partial f_1}{\partial h_1} & \frac{\partial f_1}{\partial h_2} \\ \frac{\partial f_2}{\partial h_1} & \frac{\partial f_2}{\partial h_2} \end{bmatrix}$

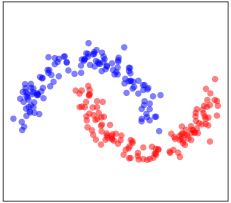
In general: $\boldsymbol{J} = \nabla_{\boldsymbol{h}(t_0)} f(\boldsymbol{h}(t), t)$

$\|\boldsymbol{J}\|_F = \sqrt{\sum_{i=1}^{d} \sum_{j=1}^{d} |\boldsymbol{J}_{i,j}|^2} \downarrow$ Frobenius: neural ODE

$\|\boldsymbol{J}\|_2 = \sigma_{\text{max}}(\boldsymbol{J}) \downarrow$ spectral: neural network

$\kappa(\boldsymbol{J}) = \frac{\sigma_{\text{max}}(\boldsymbol{J})}{\sigma_{\text{min}}(\boldsymbol{J})} \to 1$ condition number: our work

Jacobian norm regularisation



Binary classification

Intertwining moons dataset

$$\|\boldsymbol{J}\|_F = \sqrt{\sum_{i=1}^{d} \sum_{j=1}^{d} |\boldsymbol{J}_{i,j}|^2} \ \downarrow$$

Frobenius: neural ODE

$$\|\boldsymbol{J}\|_2 = \sigma_{\mathsf{max}}(\boldsymbol{J}) \ \downarrow$$

spectral: neural network

$$\kappa(\boldsymbol{J}) = \frac{\sigma_{\mathsf{max}}(\boldsymbol{J})}{\sigma_{\mathsf{min}}(\boldsymbol{J})} \to 1$$

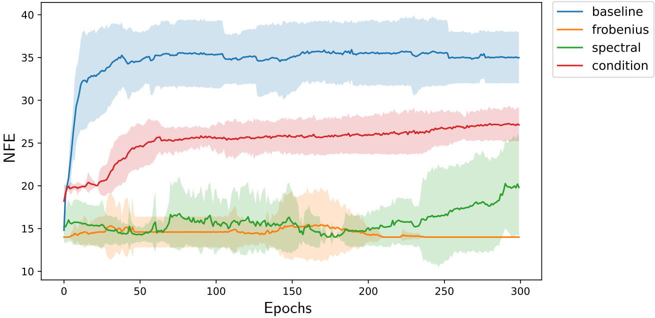
condition number: our work

NFE reduction

Frobenius, spectral, and condition number regularisation reduce NFE.

NFE reduction

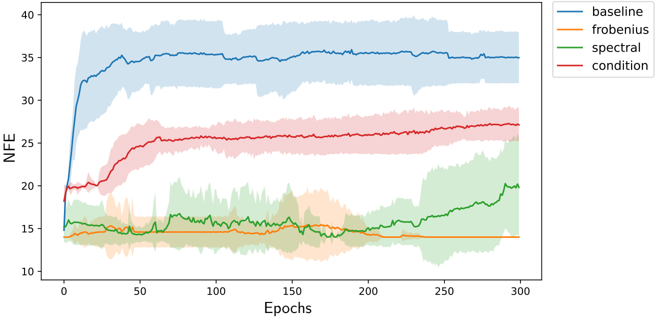
Frobenius, spectral, and condition number regularisation reduce NFE.



Solid curves and shaded regions indicate mean and standard deviation over 10 runs.

NFE reduction

Frobenius, spectral, and condition number regularisation reduce NFE.



Solid curves and shaded regions indicate mean and standard deviation over 10 runs.

Good! But at what cost?

Performance and robustness

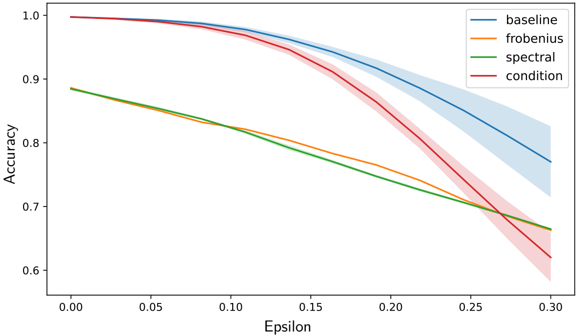
a) Jacobian norm regularisation sacrifices performance for NFE reduction.

b) Robustness to input noise for condition number regularisation.

Performance and robustness

a) Jacobian norm regularisation sacrifices performance for NFE reduction.

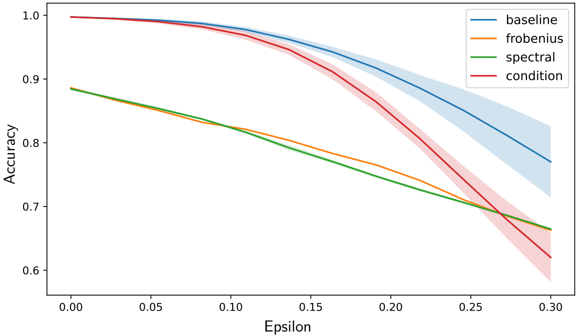
b) Robustness to input noise for condition number regularisation.



Performance and robustness

a) Jacobian norm regularisation sacrifices performance for NFE reduction.

b) Robustness to input noise for condition number regularisation.



Jacobian norm regularisation leads to increased distance to decision boundary.

Effect on conditioning

Condition number an explanation for robustness?

Effect on conditioning

Condition number an explanation for robustness?

Intertwining moons	
Regularisation	Condition number
None	5.3 ± 3.5
Frobenius	27.3 ± 34.1
Spectral	45.9 ± 70.6
Condition number	6.1 ± 5.2

Conclusion

Recall that we want generalisation and a reduced NFE.

Conclusion

Recall that we want generalisation and a reduced NFE.

1. Jacobian norm regularisation reduces NFE, potentially at a cost to generalisation and robustness. Condition number regularisation seems to help.

Conclusion

Recall that we want generalisation and a reduced NFE.

1. Jacobian norm regularisation reduces NFE, potentially at a cost to generalisation and robustness. Condition number regularisation seems to help.
2. Jacobian norm regularisation can lead to an increased distance to the decision decision boundary.

Conclusion

Recall that we want generalisation and a reduced NFE.

1. Jacobian norm regularisation reduces NFE, potentially at a cost to generalisation and robustness. Condition number regularisation seems to help.
2. Jacobian norm regularisation can lead to an increased distance to the decision decision boundary.

Ongoing work:

Conclusion

Recall that we want generalisation and a reduced NFE.

1. Jacobian norm regularisation reduces NFE, potentially at a cost to generalisation and robustness. Condition number regularisation seems to help.
2. Jacobian norm regularisation can lead to an increased distance to the decision decision boundary.

Ongoing work:

1. Efficient condition number estimation.

Conclusion

Recall that we want generalisation and a reduced NFE.

1. Jacobian norm regularisation reduces NFE, potentially at a cost to generalisation and robustness. Condition number regularisation seems to help.
2. Jacobian norm regularisation can lead to an increased distance to the decision decision boundary.

Ongoing work:

1. Efficient condition number estimation.
2. Characterise conditions for rising NFE (stiffness?).

Conclusion

Recall that we want generalisation and a reduced NFE.

1. Jacobian norm regularisation reduces NFE, potentially at a cost to generalisation and robustness. Condition number regularisation seems to help.
2. Jacobian norm regularisation can lead to an increased distance to the decision decision boundary.

Ongoing work:

1. Efficient condition number estimation.
2. Characterise conditions for rising NFE (stiffness?).
3. Other ways to parameterise the ODE or solution?