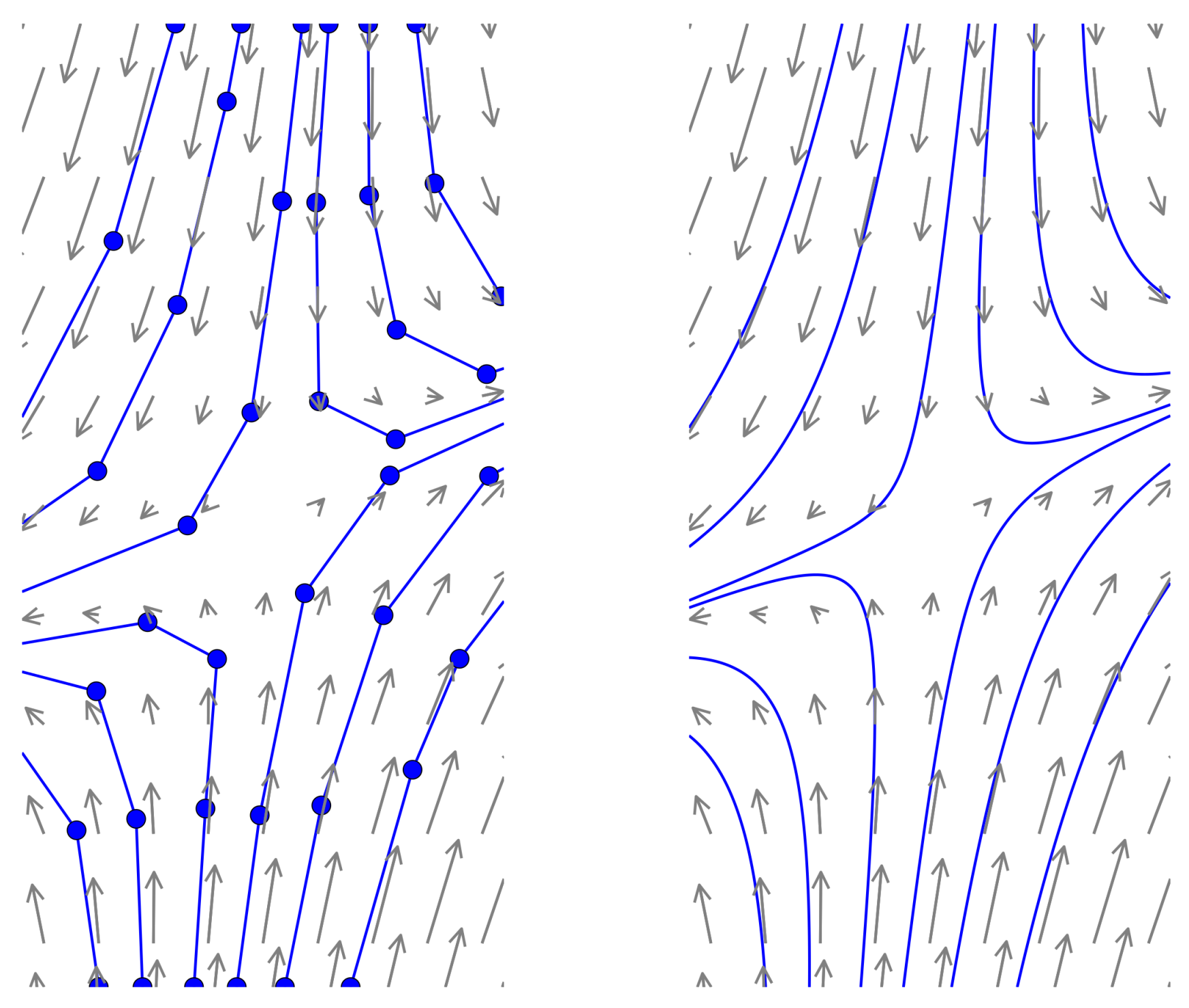
Neural Ordinary Differential Equations



Niclas Joshua Popp

KTH Royal Institute of Technology, Department of Mathematics
SF2525 Poster Session

Dynamical Sysytems vs Machine Learning

Both machine learning models and ordinary differential equations (ODEs) are intended to encode nonlinear transforms. A typical questionnaire is to predict an output y given an input x. In machine learning we train a model M_{ML} to come up with the predictions. As an example we can take a simple k-layer neural network in matrix form with sigmoid activation function

$$M_{ML}(x) = \sigma(W_k \ldots \sigma(W_2 \sigma(W_1 x))) \tag{1}$$

However, we sometimes have some prior knowledge of how y evolves as x changes. If this prior information can be encoded into a function f we can formulate an ODE.

$$\frac{d}{dx} y(x) = f(y(x), x) \tag{2}$$

$$y(x_0) = y^0 \tag{3}$$

In general, ODEs are an useful inductive bias for phyics-inspired machine learning and scientific machine learning.

From Residual Networks to Neural ODEs

Neural Ordinary Differential Equations (Neural ODEs) are a new type of deep neuronal networks that were introduced by Chen et al.[1]. The progenitor to Neural ODEs are *Residual Networks* (ResNets) [3]. The building blocks of such networks are given by

$$h_{i+1} = \mathcal{F}(\{W_i\}, h_i) + h_i \tag{4}$$

where h_i denote the hidden states and $\{W_i\}$ the parameters. These building blocks are structurally similar to one step of the *Euler Method* which is a numerical scheme to solve initial value problems. Using the formulation from equation 2 its iteration is given by

$$y_0 = y^0 \tag{5}$$

$$y_{i+1} = y_i + h f(y_i, x_i) \tag{6}$$

Neural ODEs reverse engineer the ResNet structure given by equation 4 to become an ODE [2]. The hidden layers are given by

$$\frac{d}{dx} h(x) = f(h(x), x, \{W\}) \tag{7}$$

This defines the general structure of a Neural ODE [1] which can be applied in different settings.

Gradient Computation

In order to train Neural ODEs through variants of the gradient descent method it is necessary to calculate the derivatives with respective to the parameters $\{W\}$ in f. In contrast to standard neural networks, it is possible to determine the gradients themselves as solutions to ODEs. This is realized through the *adjoint sensitivity method.*
Assume we are given a scalar-valued loss function L. The adjoint $a(x)$ models how the loss changes with respect to the hidden state.

$$a(x) = \frac{\partial L}{\partial h(x)} \tag{8}$$

The adjoint itself follows the following ODE which can be derived using the chain rule

$$\frac{d}{dx} a(x) = -a(x)^T \frac{\partial f(h(x), x, \{W\})}{\partial h} \tag{9}$$

This ODE can be solved backwards using any suitable ODE solver. This step requires the entire trajectory of $h(x)$ which can be computed simultaneously. The gradient with respect to the parameters is then given by

$$\frac{dL}{dW} = -\int_{x_0}^{x_1} a(x)^T \frac{\partial f(h(x), x, \{W\})}{\partial W} dt \tag{10}$$

Application 1: Time Series Analysis

Assume we are given time dependent data $x(t_i)$ for a sequence of time points $t_1, \ldots, t_n$. The general assumption to learn such a dynamical system is to use a feed-forward model with trainable parameters $\{W\}$

$$x_{k+1} = F(x_k, t_k, t_{k+1}, \{W\}) \tag{11}$$

which in the case of a continuous-depth neural network is given by

$$F(x_k) = \text{ODESolve}[G(x; \{W\}), t_{start} = t_k, t_{end} = t_{k+1}](x_k) \tag{12}$$

The function $G(x; \{W\})$ is the neural network we intend to train, an example is given by equation (1). The loss function for this purpose is given by the error at each time step

$$\sum_{i=1}^{n} ||F(x_i, t_i, t_{i+1}, \{W\}) - x(t_i)||_2^2 \tag{13}$$

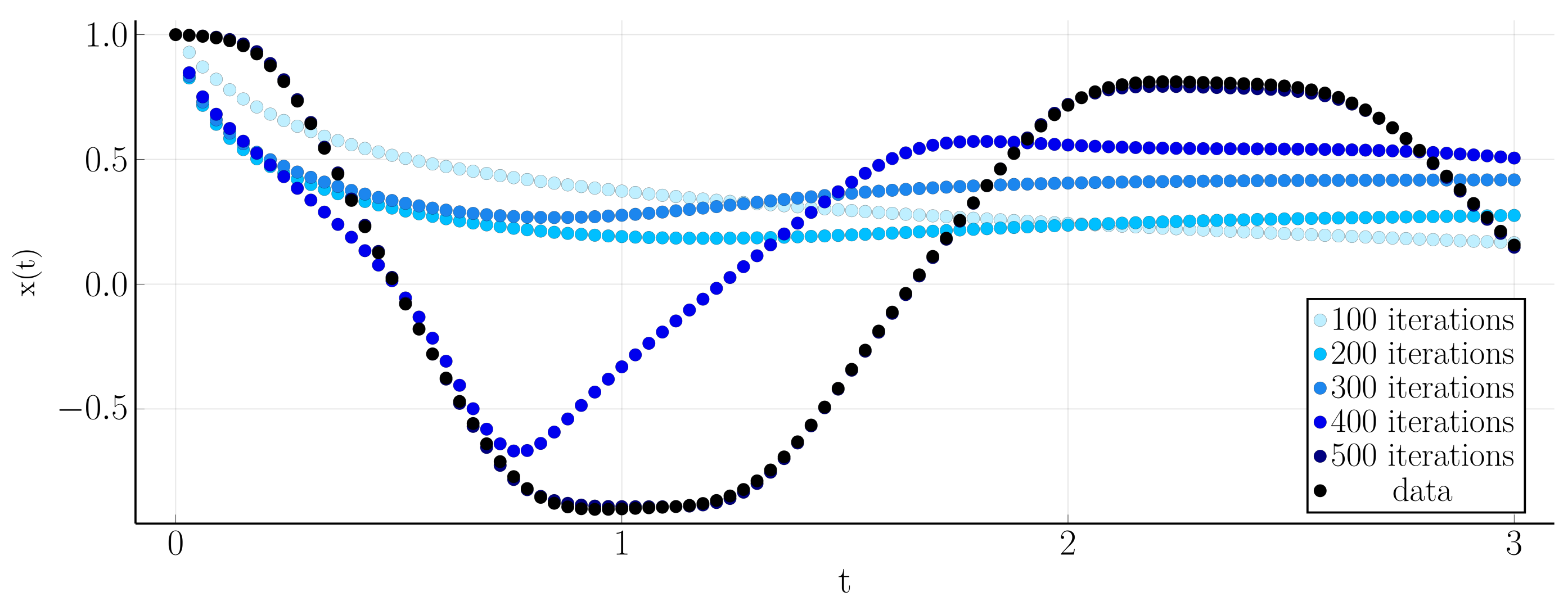


Figure 1: Continuous-depth neural network learning a simple dynamic system from time series data

Application 2: Normalizing flows

The second application of Neural ODEs is related to density estimation using flow-based generative models. The approach of standard normalizing flows is to pass a simpler density Z, typically $\mathcal{N}(0, I)$, through a diffeomorphism f to receive a more complex density $f(Z)$. This change of variables is repeated iteratively which leads to the following update formula for the log density

$$Z_k = f_K \circ \ldots \circ f_2 \circ f_1(Z_0) \tag{14}$$

$$\log(q_{i+1}(Z_{i+1})) = \log(q_i(Z_i)) - \log\left|\det\left(\frac{\partial f}{\partial Z_i}\right)\right| \tag{15}$$

Similarly when using residual flows

$$f(z) = z + F(z, \{W\}) \tag{16}$$

with a parameter dependent update function F this again can be seen as a discretization of a differential equation. When generalizing to infinitely many updates, the change of variable is given through the following partial differential equation

$$\frac{\partial z(t)}{\partial t} = f(z(t)) \tag{17}$$

$$\frac{\partial \log p(z(t)))}{\partial t} = -tr\left(\frac{df}{dz(t)}\right) \tag{18}$$

This structure characterises so called *continuous normalizing flows* which can be interpreted as Neural ODEs.

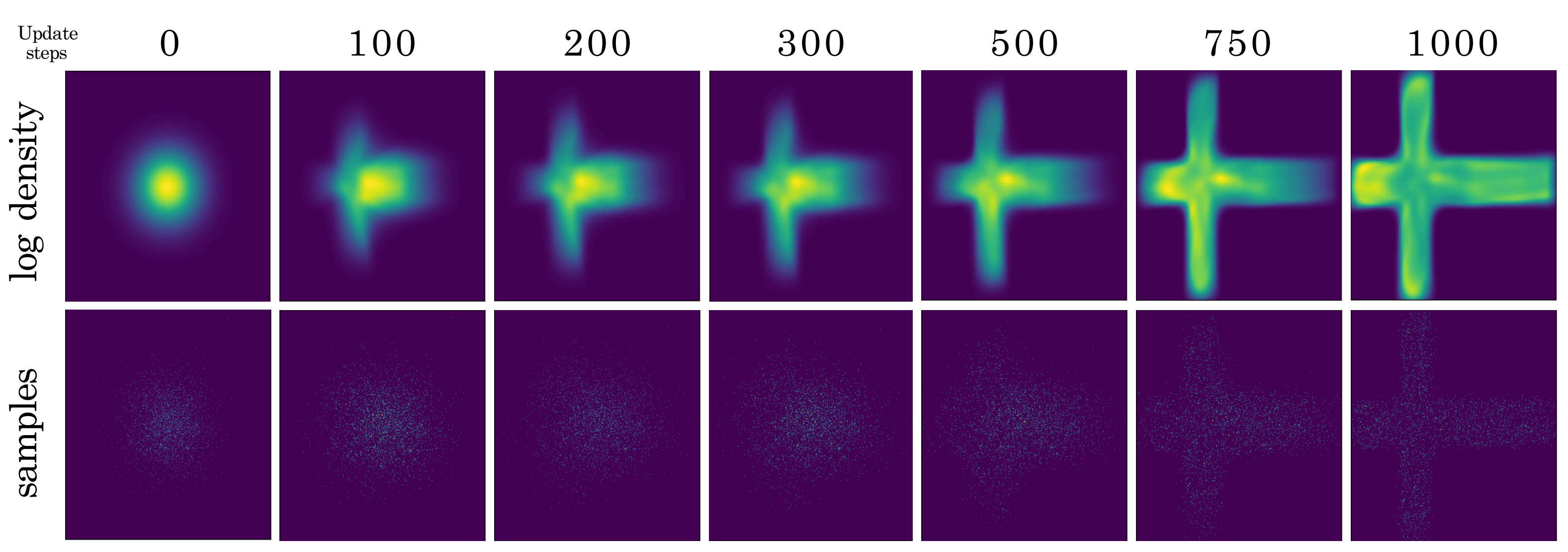


Figure 2: Continuous normalizing flow learning an example density which is higher in the yellow areas of the Swedish flag and lower in the blue areas. The bottom row shows samples that were passed through the flow.

References

[1] R. T. Q. Chen, Y. Rubanova, J. Bettencourt and D. Duvenaud. Neural ordinary differential equations, 2018. URL https://arxiv.org/abs/1806.07366

[2] W. E. A proposal on machine learning via dynamical systems, 2017. URL https://doi.org/10.1007/s40304-017-0103-z

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385

[4] A. F. Queiruga, N. B. Erichson, D. Taylor, and M. W. Mahoney. Continuous-in-depth neural networks, 2020. URL https://arxiv.org/abs/2008.02389