Exploring Neural Ordinary Differential Equations

Jannik Wagner, Katrina Liang, Niclas Popp
KTH Royal Insitute of Technology
DD2424 Deep Learning in Data Science
Project Report

Abstract

Neural ordinary differential equations are a class of neural networks with infinitely many layers that generalize residual networks. In this report we present the fundamental theory behind neural ODEs together with two applications. We explain how neural ODEs can be used for descriptive time series modelling and show an example using COVID-19 infections in Sweden. Application two concerns density estimation using a family of generative models called continuous normalizing flows.

1 Introduction

At the NeurIPS conference 2018 Duvenaud et al. [1] introduced *Neural Ordinary Differential Equations* (Neural ODEs) as a new family of neural networks that combines ideas from dynamical systems and deep learning. Instead of featuring several distinct hidden layers, the architecture of Neural ODEs is given by continuous differential equations. In this report we present a fundamental explanation of the theoretical concepts behind such networks together with two applications for time series modelling and normalizing flows that make use of their special capabilities.

The main advantage of Neural ODEs for descriptive time series analysis is the possibility to continuously model data that features irregular time stamps. One example of data that is generally captured irregularly are COVID-19 infections. We train a Neural ODE network to model the newly registered infections in Sweden for a timespan of 170 days, starting from the 1st November 2022. We compare these results to a classical approach based on ordinary differential equations [2].

The second application area where Neural ODEs provide useful benefits, is density estimation with generative models. The methodology for this purpose is based on a continuous variant of normalizing flows [3]. We explain the theoretical connection between normalizing flows and Neural ODEs and exemplify how to learn a two or higher dimensional density and generate samples from it. For this purpose, we experiment with different network architectures and discuss the results.

The report for this project is structured in the following way. For better readability all necessary mathematical foundations are introduced in chapter 2. In chapter 3 we present and discuss the results from our experiments related to the two applications. We conclude with a brief summary of our findings and an outlook about Neural ODEs in general.

2 Mathematical Foundations

2.1 Machine learning vs Dynamical Systems

Machine learning and dynamical systems are approaches that both intend to encode arbitrary 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. However, sometimes there is prior information about how y evolves as x changes. If this knowledge can be encoded into a function f we can

formulate an *ordinary differential equation* (ODE).

$$\frac{d}{dx}y(x) = f(y(x), x) \tag{1}$$

$$y(x_0) = y^0 \tag{2}$$

This formulation is called an initial value problem since the dynamics of y are given together with a starting point y_0. Such ODE dynamics were first introduced to describe phenomena in classical mechanics but have since been applied to almost all scientific disciplines including the *SEIR* model for epidemics presented in section 2.2. In general, ODEs are a useful inductive bias to combine machine learning with mechanistic modelling.

2.1.1 Fundamental Concept of Neural ODEs

From ResNets to Neural ODEs The progenitor to Neural ODEs are *Residual Networks* (ResNets) [4], a special kind of neural networks. The building blocks of such networks are given by

$$h_{i+1} = \mathcal{F}(\{W_i\}, h_i) + h_i \tag{3}$$

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 6, its iteration is given by

$$y_0 = y^0 \tag{4}$$

$$y_{i+1} = y_i + hf(y_i, x_i) \tag{5}$$

Neural ODEs reverse engineer the ResNet structure given by 3 to become an ODE [5]. The hidden layers are given by

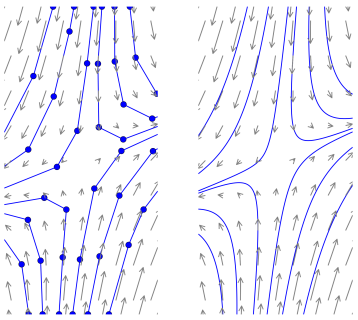


Figure 1: The left figure displays discrete transformations (ResNet) while the right figure shows continuous dynamics (Neural ODE).

$$\frac{d}{dx}h(x) = f(h(x), x, \{W\}) \tag{6}$$

$$h(x_0) = h^0 \tag{7}$$

Augmented Neural ODEs A theoretical limitation of Neural ODEs is the existence of functions that cannot be represented by them [6]. The reason for this restriction is that a continuous flow enforces the topology of the input space to be preserved. For example, the solution space of one-dimensional Neural ODEs is limited to strictly monotonic functions. In order to bypass such restrictions, the learnt space can be *augmented* from $\mathbb{R}^d$ to $\mathbb{R}^{d+p}$. Denoting $a(x) \in \mathbb{R}^d$ as a point in the augmented space, the formulation of an augmented Neural ODE is given by

$$\frac{d}{dx}\begin{bmatrix} h(x) \\ a(x) \end{bmatrix} = f\Big(\begin{bmatrix} h(x) \\ a(x) \end{bmatrix}, x, \{W\}\Big), \quad \begin{bmatrix} h(0) \\ a(0) \end{bmatrix} = \begin{bmatrix} h^0 \\ 0 \end{bmatrix} \tag{8}$$

Gradient Computation In order to train Neural ODEs through variants of the gradient descent method, it is necessary to calculate the derivatives with respect to the parameters $\{W\}$ in f. In contrast to standard neural networks, it is possible to derive the gradients as solutions to ODEs. This process is called 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.

$$a(x) = \frac{\partial L}{\partial h(x)} \tag{9}$$

The adjoint itself follows an ODE which can be derived using the chain rule [1]

$$\frac{d}{dx}a(x) = -a(x)^T \frac{\partial f(h(x), x, \{W\})}{\partial h} \tag{10}$$

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

$$\frac{dL}{dW} = -\int_{x_0}^{x_1} a(x)^T \frac{\partial f(h(x), x, \{W\})}{\partial W} dt \tag{11}$$

The adjoint sensitivity method is much more memory efficient than standard auto-differentiation since it is not necessary to store any intermediate information from the forward pass in order to carry out the backward pass. However, it requires more evaluations of the network and potentially ODE solvers with high accuracy which makes it sometimes impractical in practice.

2.1.2 Time Series Analysis

The first application of Neural ODEs are descriptive models for time series analysis. Assume we are given time dependent data $x(t_i)$ for a sequence of arbitrary time points $t_1, ..., t_n$. The general assumption to learn a dynamical systems that fits this data is to use a feed-forward model with trainable parameters $\{W\}$

$$x_{k+1} = F(x_k, t_k, t_{k+1}, \{W\}) \tag{12}$$

For this purpose, we now use a neural network $G(x, t, \{W\})$ to model the right hand side of the dynamical system we attempt to learn

$$\frac{dx}{dt} = G(x(t), t, \{W\}) \tag{13}$$

This leads to the following update formula for F [7]

$$F(x_k) = \text{ODESolve}[G(x, t, \{W\}), t_{start} = t_k, t_{end} = t_{k+1}](x_k) \tag{14}$$

The goal of this model is to train $G(x; \{W\})$ which could in theory be any neuronal network. The loss function for this purpose is given by the mean squared error

$$\sum_{i=1}^{n} ||F(x_i, t_i, t_{i+1}, \{W\}) - x(t_{i+1})||_2^2 \tag{15}$$

2.1.3 Continuous Normalizing Flows

The second application of Neural ODEs considers a group of generative models called *normalizing flows* which enable density estimation and generation of new samples according to the learned density. The approach of standard normalizing flows is to pass a random variable Z distributed according to a simple density p_Z, typically $\mathcal{N}(0, I)$, through a series of diffeomorphisms $F = F_K \circ ... \circ F_2 \circ F_1$ to receive a random variable $X = F(Z)$ distributed according the more complex density p_X underlying the data. These transformations depend on a set of parameters $\{W\}$ that we intend to learn. An example choice for the F_i are invertible neuronal networks. According to the change of variables formula, the log likelihood of X is given by

$$\log p_X(X) = \log p_Z(Z) - \log \left| \det \left(\frac{\partial F}{\partial Z} \right) \right| \tag{16}$$

$$= \log p_Z(Z) - \sum_{i=1}^{K} \log \left| \det \left(\frac{\partial F_i}{\partial Z_{i-1}} \right) \right| \tag{17}$$

where $Z_i = F_i(Z_{i-1})$ and $Z_K = X, Z_0 = Z$. In order to evaluate this log probability on some instance X, $Z = F^{-1}(X)$ has to be calculated requiring F to be invertible and the determinant of the Jacobian of F needs to be computed which is computationally expensive in general. The training over data D is performed using maximum likelihood estimation

$$\{W\}^* = \arg\max_{\{W\}} \sum_{x \in D} \log p_X(x) \tag{18}$$

For residual flows [8]

$$F_i(Z_{i-1}) = Z_{i-1} + f_i(Z_{i-1}, \{W_i\}) \tag{19}$$

where $f_i(Z_{i-1}, \{W_i\})$ can be rewritten as a shared parameter and time dependent update function $f(Z_{i-1}, i, \{W\})$, this again can be seen as a discretization of a differential equation. When generalizing this formula to infinitely many updates, the change of variable is given through the following partial differential equation

$$\frac{\partial Z(t)}{\partial t} = f(Z(t), t, \{W\}) \tag{20}$$

$$\frac{\partial \log p(Z(t))}{\partial t} = -\operatorname{Tr}\left(\frac{df}{dZ(t)}\right) \tag{21}$$

with $Z(0) = Z$ and $Z(T) = X$. This structure is called a *continuous normalizing flow* (CNF). Evaluating such a model is less expensive than the discrete iteration since it suffices to evaluate the trace of the Jacobian. With the use of the Hutchinson Trace Estimator

$$\operatorname{Tr}(A) = \mathbb{E}_{e\sim p(e)}[e^T A e] \tag{22}$$

where $\mathbb{E}[e] = 0,\ \operatorname{Cov}[e] = I$, the trace can be approximated in $O(d)$ per sample e [3]. A proof of both statements can be found in [1].

New data can be generated according to the assumed generative process, by sampling $Z \sim p_Z(Z)$ and then integrating the learned neural ODE f. The log density and the sample can be simultaneously integrated by concatenating the states.

$$\begin{bmatrix} X \\ \log p_X(X) \end{bmatrix} = \begin{bmatrix} Z \\ \log p_Z(Z) \end{bmatrix} + \int_0^T \begin{bmatrix} f(Z(t), t, \{W\}) \\ -e^T \frac{\partial f}{\partial Z(t)} e \end{bmatrix} \tag{23}$$

For the evaluation of the log probability of an instance X, the dynamical system has to be solved reversely

$$\begin{bmatrix} \Delta_X \\ \Delta_p \end{bmatrix} = \int_T^0 \begin{bmatrix} f(Z(t), t, \{W\}) \\ -e^T \frac{\partial f}{\partial Z(t)} e \end{bmatrix} \tag{24}$$

leading to $Z = X + \Delta_X$ and $\log p_X(X) = \log p_Z(Z) - \Delta_p$. Note that f is not required to be invertible anymore and can thus be represented by an arbitrary neuronal network.

2.2 Compartmental Models for Modelling Epidemics

Compartmental modelling is a very general ODEs based technique to represent real-world dynamical systems. In this project we make use of this approach to describe epidemic dynamics and investigate how they compare to Neural ODEs.
First, we split the entire population N into four compartments for susceptible S , exposed E, infected I and recovered R. The $SEIR$-model describes the interaction between the different groups in an epidemic scenario [2]. A more in-depth explanation of this approach can be found in [9].

$$\frac{dS}{dt} = -\beta\frac{SI}{N},\quad \frac{dS}{dt} = \beta\frac{SI}{N} - \mu E,\quad \frac{dS}{dt} = \mu E - \gamma I,\quad \frac{dS}{dt} = \gamma I \tag{25}$$

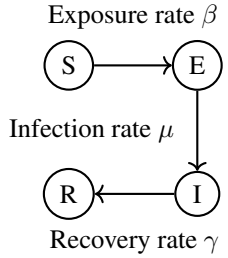


Figure 2: Dynamics of the $SEIR$-model

3 Experiments

3.1 Times Series Analysis

We apply Neural ODEs to epidemic time series data and compare the results to the classical modelling approach using the SEIR equations. The data comprises of the newly registered COVID-19 infections in Sweden from 1st November to 20th of April with irregular timestamps [10]. The results are shown in figure 3.
As explained in the foundations section, there are several technicalities to be aware of when training Neural ODEs. First, we augment the problem into 4-dimensions. The dimensionality was chosen to be the same as for the SEIR model to ensure that the neural networks can learn dynamics that are sufficiently complex for an epidemic scenario. Apart from that, local minima are an issue when fitting Neural ODEs especially when using deeper networks. To address this issue, we make use of

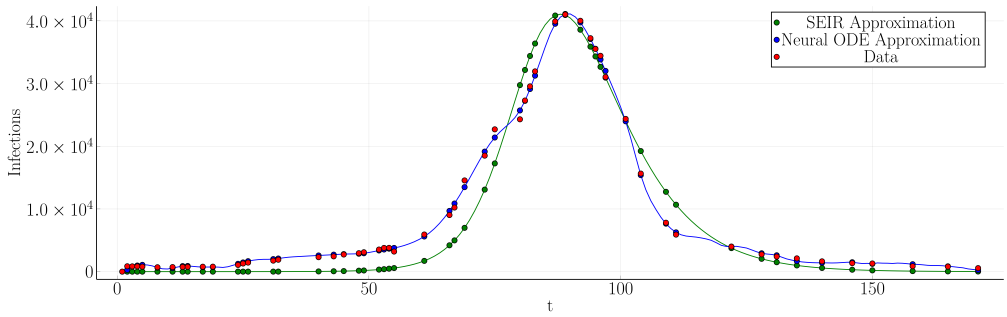


Figure 3: Comparison between a Neural ODE based approximation and an ODE based on time series data that comprises of the newly registered COVID-19 infections in Sweden starting from 1st November 2021.

a relatively narrow right hand side together with iteratively growing fits through training multiple models with extending timespans. The previous model is used as initial condition for the successive network. Since the intermediate networks are only used for initialization purposes it is not necessary for all of them to have reached convergence. For the right hand side of our final model we use a 3-layer neuronal network with [100,50,4] nodes and swish activation function [11]. This particular activation was used since it facilitated the detection of the increase in infections after day 60. The evolution of loss during training of the five intermediate and the final network can be found in figure 7 in the appendix.
The computations were performed in Julia using the DiffFluxEq.jl package [12]. The main reason for choosing Julia in this case is the efficient implementation of ODE solvers that are required for training. We applied the Tsitouras 5/4 Runge-Kutta method [13] which is a commonly used robust solver suited for most non-stiff problems.
The reference model using the SEIR equations was determined through parameter optimization using stochastic gradient descent to reduce the mean squared error with respect to the data. Although the SEIR model captures the overall dynamic of the system, it is very imprecise towards the tails of the series since the model itself lacks the necessary complexity to capture this behavior. The Neural ODE was able to learn a more complex dynamic and clearly provides a better fit.

3.2 Continuous Normalizing Flows

Artificial two dimensional density First, we applied CNFs to artificial data generated from a two dimensional density based on an image of the Sweden flag. The corresponding samples were generated based on the color of the pixels where the yellow areas are assigned a higher density value than the blue parts and then Gaussian noise is added. In figure 4, the continuous transformation from the latent distribution to the learned distribution are shown together with samples that were created according to the state at intermediate time points.
One key design decision is how to incorporate the time t of the dynamical system into the neural network $f(z(t), t, \{W\})$. For this purpose, we experimented with four different approaches. The simplest choice is to use a time independent right hand side given by a neural network with fully connected layers. Alternatively, the time t and the input to a fully connected layer can be concatenated before passing it through the linear layer. We call this time incorporation by concatenation. Instead of concatenating the time, simple neural networks $g(t)$ and $a(t)$ can be used to gate and add to the input z, i.e., $\tilde{z} = g(t) \cdot z + a(t)$, where $g(t)$ has values between 0 and 1, before passing it through a fully connected layer. We call this gate and add time incorporation. A more sophisticated approach is to use a hypernetwork $h(t, \{W_h\})$ that takes the time t as input and outputs the parameters $\{W\}$ in $f(z(t), t, \{W\})$. Specifically, for depth 2 and width w of the main network and data dimension d, the hypernetwork outputs tensors $V \in \mathbb{R}^{w \times d}, b \in \mathbb{R}^w, U \in \mathbb{R}^{d \times w}$. In the main network, the operation $f(z) = U \tanh(Vz + b)/w$ is applied, i.e., a fully connected layer with weight V and bias b, followed by a tanh non-linearity whose outputs are used as weights for a weighted average over the columns in Z. The optimization is then performed over the parameters $\{W_h\}$ that constitute the hypernetwork h.
The code that was used to perform the experiments is based on a basic implementation of continuous normalizing flows using the Python library `torchdiffeq` [14] which realizes the concepts that were presented by Chen et al. [1]. The results for all four approaches including hypernetworks with

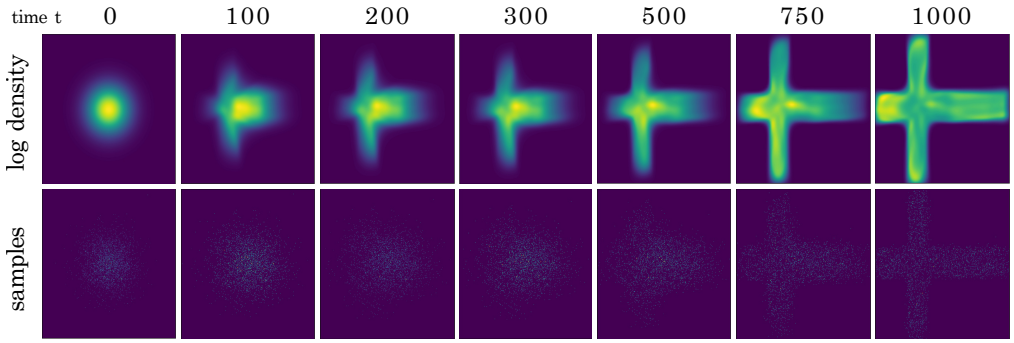


Figure 4: The continuous transformation over time of a normal distribution to the learned distribution. The right hand side of the Neural ODE was determined through a network with depth 2 and width 64 and time incorporation by the hypernetwork approach. The training was done over 1000 update steps.

different hyper-parameters can be found in table 1 in the appendix and figure 9 visualizes the log densities. We found that the best result could be achieved when using hypernetworks. The main networks that were used featured 2 fully connected layers and the weights were the output of a a 3-layer hypernetwork. After 1000 iterations the loss was lowest when both networks had a depth of at least 16. Further increasing the number of nodes in the networks only brought marginal gains. The gate and add approach achieved slightly worse but still satisfying results. Both time concatenation and a time independent right hand side lead to worse loss scores and slower convergence in comparison to the other two approaches.

High dimensional densities Apart from investigating the influence of different architectures on the performance of continuous normalizing flows when estimating a simple density, we also experimented with real world data and used the MNIST data set [15] as samples from a $1 \times 28 \times 28 = 784$ dimensional distribution. In contrast to the previous setup, the right hand side $f(z(t), t, \{W\})$ of the dynamical system is represented as a convolutional neuronal network. Instead of using a hypernetwork, we concatenated the time as another channel to the image data in each convolutional layer as suggested by Grathwohl et al. [3].
Our model consists of four of these time concatenating 2d convolutions with 64 output channels each, except for the last one with one output channel. After all but the last 2d convolutions, a softplus activation function [16] is used. The training is conducted over 10 epochs with the ADAM optimizer, batch size 200 and learning rate 0.001. Although MNIST is a rather simple data set and even with simple architectures consisting only of 4 layers, the training time was surprisingly long taking several hours per epoch on an NVIDIA Tesla K80. This prevented us from conducting more extensive experiments with higher numbers of epochs or more complex architectures and from achieving visually satisfying results. For comparison, we also trained a more complex model consisting of several stacked CNFs using the implementation by [3]. The generated images can be seen in figure 10 in the appendix. We conclude that in order to apply our approach to high dimensional real-world data and generate meaningful new samples, a more complex architecute has to be used for which it does not suffice to use a straight forward implementation but further in depth optimization is required to speed up the execution time and regularize the model which is beyond the scope of this project.

4 Conclusion

In this report we presented the theory that is necessary to develop a fundamental understanding of Neural ODEs together with two applications that showcase some of their specific capabilities. The current consensus is that Neural ODEs perform similarly to residual networks on standard supervised learning tasks such as image classification [1]. Their main advantages are the very memory efficient training using the adjoint sensitivity method and the specific applications that are possible only with this particular kind of network structure. However, since there remain some open theoretical and practical questions, the field is an area of active research. In addition to improvements for standard Neural ODEs, potential further generalizations that exploit connections to other deep learning models such as generative adversarial networks have been proposed recently [17].

References

[1] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," arXiv, 2018. DOI: 10.48550/ARXIV.1806.07366. [Online]. Available: https://arxiv.org/abs/1806.07366.

[2] S. N. Zisad, M. S. Hossain, M. S. Hossain, and K. Andersson, "An integrated neural network and seir model to predict covid-19," *Algorithms*, no. 3, 2021, ISSN: 1999-4893. DOI: 10.3390/a14030094. [Online]. Available: https://www.mdpi.com/1999-4893/14/3/94.

[3] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, *Ffjord: Free-form continuous dynamics for scalable reversible generative models*, 2018. DOI: 10.48550/ARXIV.1810.01367. [Online]. Available: https://arxiv.org/abs/1810.01367.

[4] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. DOI: 10.48550/ARXIV.1512.03385. [Online]. Available: https://arxiv.org/abs/1512.03385.

[5] W. E, *A proposal on machine learning via dynamical systems*, 2017. DOI: 10.1007/s40304-017-0103-z. [Online]. Available: https://doi.org/10.1007/s40304-017-0103-z.

[6] E. Dupont, A. Doucet, and Y. W. Teh, *Augmented neural odes*, 2019. DOI: 10.48550/ARXIV.1904.01681. [Online]. Available: https://arxiv.org/abs/1904.01681.

[7] A. F. Queiruga, N. B. Erichson, D. Taylor, and M. W. Mahoney, *Continuous-in-depth neural networks*, 2020. DOI: 10.48550/ARXIV.2008.02389. [Online]. Available: https://arxiv.org/abs/2008.02389.

[8] R. T. Q. Chen, J. Behrmann, D. Duvenaud, and J.-H. Jacobsen, *Residual flows for invertible generative modeling*, 2019. DOI: 10.48550/ARXIV.1906.02735. [Online]. Available: https://arxiv.org/abs/1906.02735.

[9] J. Müller and C. Kuttler, *Methods and Models in Mathematical Biology*. Jan. 2015, ISBN: 978-3-642-27250-9. DOI: 10.1007/978-3-642-27251-6.

[10] Folkhälsomyndigheten, *Confirmed covid-19 cases and testing for sars-cov-2 by age group and week in sweden*, May 2022. [Online]. Available: https://www.folkhalsomyndigheten.se/the-public-health-agency-of-sweden/communicable-disease-control/covid-19/statistics/.

[11] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *CoRR*, vol. abs/1710.05941, 2017. arXiv: 1710.05941. [Online]. Available: http://arxiv.org/abs/1710.05941.

[12] C. Rackauckas, M. Innes, Y. Ma, J. Bettencourt, L. White, and V. Dixit, "Diffeqflux.jl - A julia library for neural differential equations," *CoRR*, vol. abs/1902.02376, 2019. arXiv: 1902.02376. [Online]. Available: https://arxiv.org/abs/1902.02376.

[13] C. Tsitouras, "Runge-kutta pairs of orders 5(4) using the minimal set of simplifying assumptions," *AIP Conference Proceedings*, vol. 1168, Sep. 2009. DOI: 10.1063/1.3241561.

[14] R. T. Q. Chen, *torchdiffeq*, version 0.2.2, Jun. 2021. [Online]. Available: https://github.com/rtqichen/torchdiffeq.

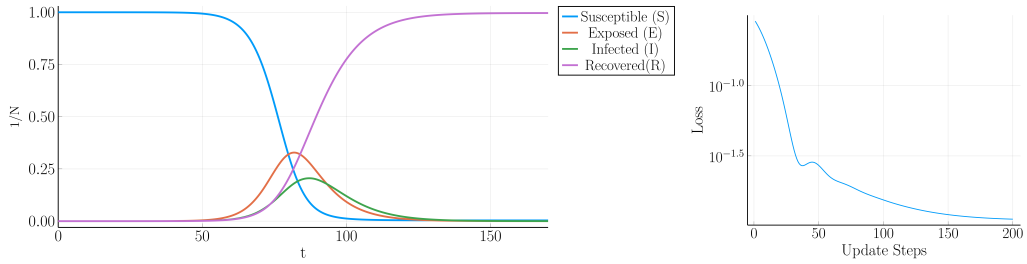
[15] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[16] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia, "Incorporating second-order functional knowledge for better option pricing.," Jan. 2000, pp. 472–478.

[17] P. Kidger, J. Foster, X. Li, H. Oberhauser, and T. Lyons, *Neural sdes as infinite-dimensional gans*, 2021. arXiv: 2102.03657 [cs.LG].

A Appendix

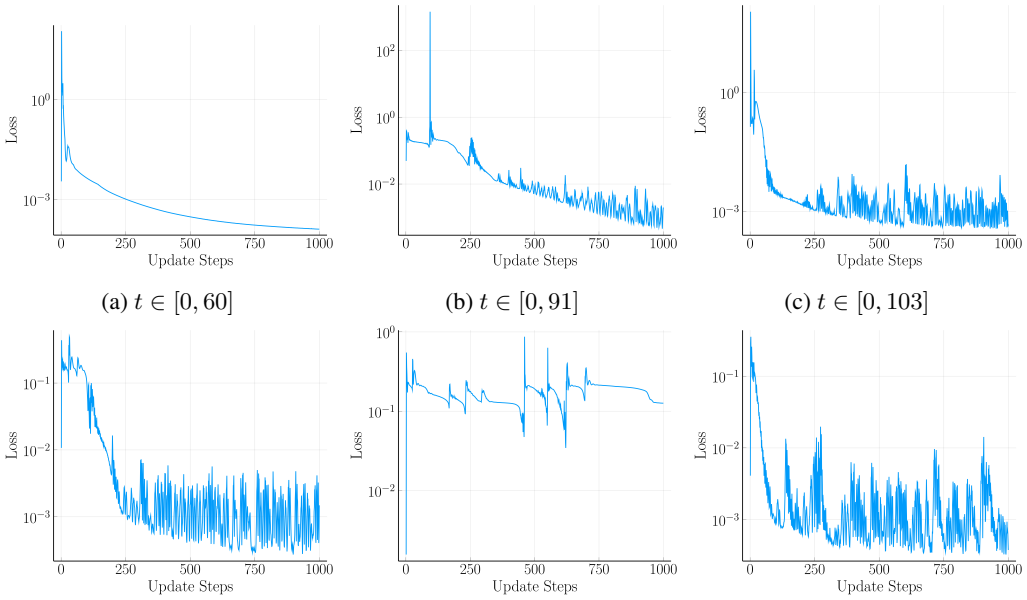
A.1 Figures



(a) Full trajectories of the final model

(b) Loss progress during training

Figure 5: The above figure depicts the full trajectory space of the final SEIR model for newly registered COVID from Sweden together with the loss during training. The training was performed using stochastic gradient descent for parameter optimization with the same loss as for the Neural ODE.



(a) $t \in [0, 60]$

(b) $t \in [0, 91]$

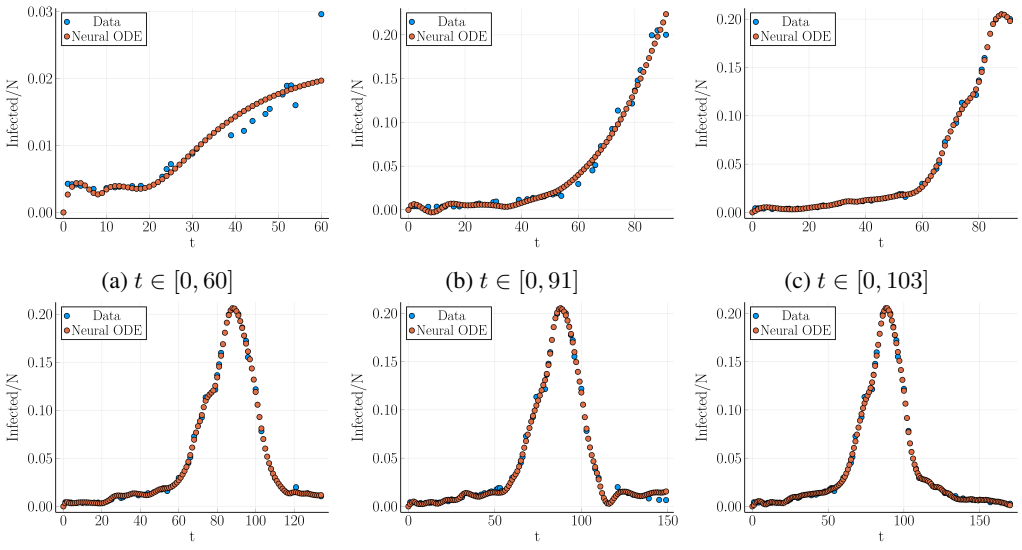
(c) $t \in [0, 103]$

(d) $t \in [0, 134]$

(e) $t \in [0, 149]$

(f) $t \in [0, 170]$

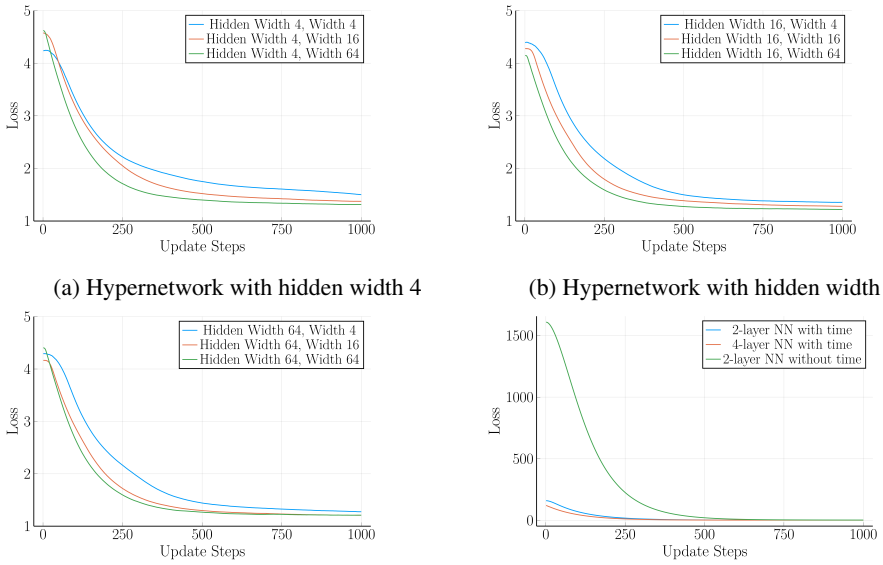
Figure 6: The figure displays the loss of the incremental Neural ODEs that were trained on time series data that contains new COVID infections in Sweden.



(a) $t \in [0, 60]$ (b) $t \in [0, 91]$ (c) $t \in [0, 103]$

(d) $t \in [0, 134]$ (e) $t \in [0, 149]$ (f) $t \in [0, 170]$

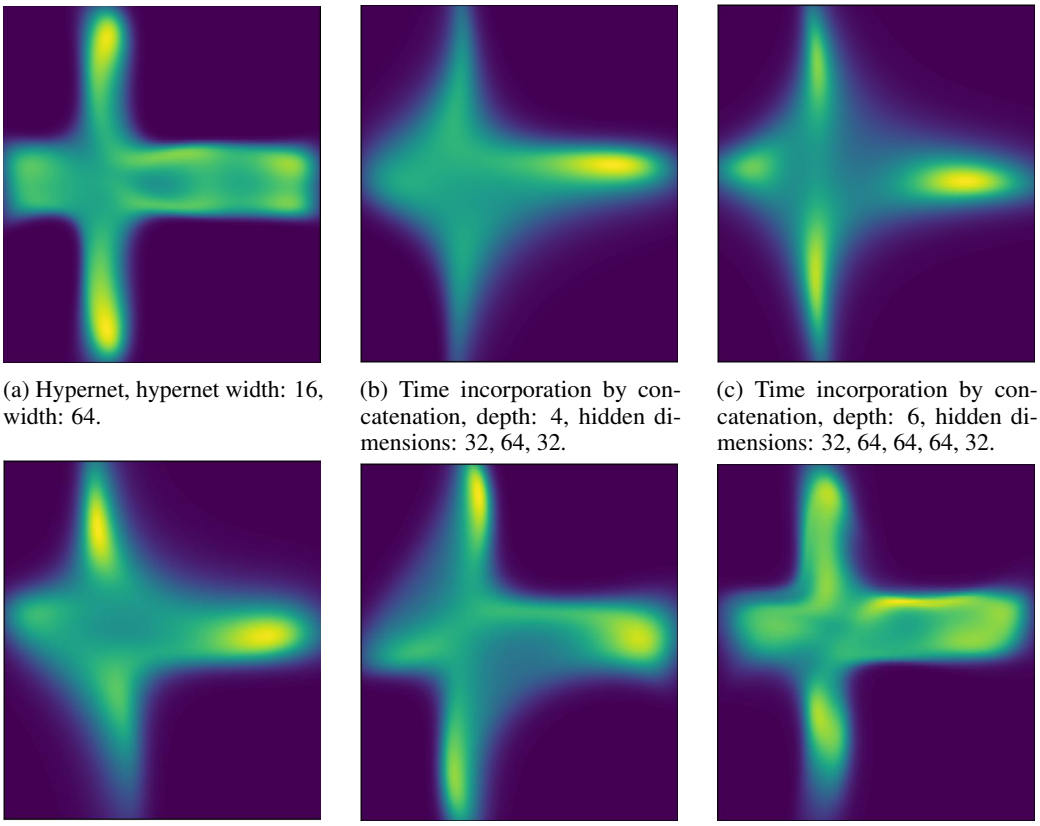
Figure 7: The figure shows the intermediate models from training our final Neural ODE on time series data featuring newly registered COVID infections in Sweden.



(a) Hypernetwork with hidden width 4 (b) Hypernetwork with hidden width 16

(c) Hypernetwork with hidden width 64 (d) Fully connected NN as right hand side

Figure 8: The figure shows the evaluation of loss during training using different setups for the right hand side of a continuous normalizing flow intended to estimate an artificial density.



(a) Hypernet, hypernet width: 16, width: 64.

(b) Time incorporation by concatenation, depth: 4, hidden dimensions: 32, 64, 32.

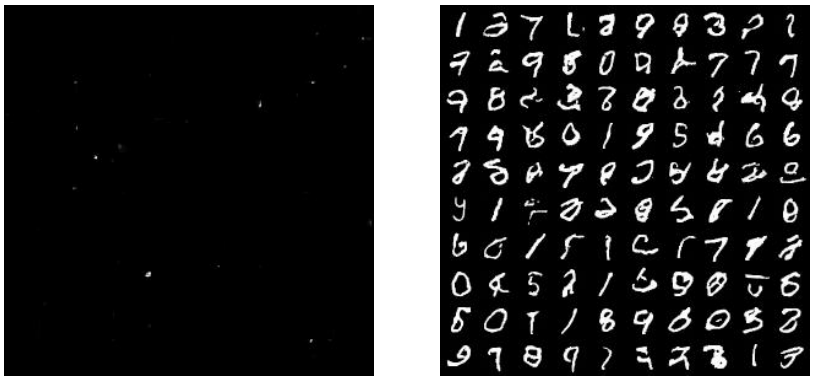
(c) Time incorporation by concatenation, depth: 6, hidden dimensions: 32, 64, 64, 64, 32.

(d) 5-layer fully connected NN (without time), hidden dimensions: 64,128,128,64.

(e) Time incorporation by concatenation, hidden dimensions: 64,128,128,64.

(f) Gate and add time, hidden dimensions: 64,128,128,64.

Figure 9: Learned log density of networks with different time incorporation methods and architectures after 1000 updates.



(a) Our CNF implementation after 11 epochs and $\approx$ 48 hours of training.

(b) FFJORD implementation using a more optimized and complex model after 31 epochs and $\approx$ 64 hours of training.

Figure 10: Generated images of simple and more complex CNF trained on MNIST.

Setting	Final Loss after 1000 Iterations
Hypernetwork with a depth of 3	
Hypernet width: 4, Width: 4	1.5013
Hypernet width: 16, Width: 4	1.3736
Hypernet width: 64, Width: 4	1.3150
Hypernet width: 4, Width: 16	1.3553
Hypernet width: 16, Width: 16	1.2792
Hypernet width: 64, Width: 16	1.2203
Hypernet width: 4, Width: 64	1.2757
Hypernet width: 16, Width: 64	1.2086
Hypernet width: 64, Width: 64	1.2097
Fully Connected Neural Network	
2-layer fully connected NN depth: 2, dimension: 5	2.7974
5-layer fully connected NN hidden dims: 64,128,128,64	1.3796
Time Incorporation through Concatenation	
2-layer NN with time incorporating layers depth: 2, hidden dimension: 5	2.0645
4-layer NN with time incorporating layers depth: 4, hidden dimensions: 32, 64, 32	1.8108
5-layer NN with time incorporating layers depth: 5, hidden dimensions: 64, 128, 128, 64	1.4956
6-layer NN with time incorporating layers depth: 6, hidden dimensions: 32, 64, 64, 64, 32	1.4972
Time Incorporating gate and add	
5-layer NN with time incorporating layers, gate and add hidden dimensions: 64, 128, 128, 64	1.4306

Table 1: Each of the models above was trained with 1000 update steps on a two dimensional example density generated from a Swedish flag and the final loss scores are shown above. The loss is the negative log probability of the data.

A.2 Code Availability

The code for this project is available at https://github.com/jannikwagner/DD2424