TO-FLOW: Efficient Continuous Normalizing Flows with Temporal Optimization adjoint with Moving Speed

Shian Du[1]*, Yihong Luo[1,2]*, Wei Chen[1]*, Jian Xu[1], Delu Zeng[1]†
[1] South China University of Technology [2] Hong Kong University of Science and Technology
201930230264@mail.scut.edu.cn, yihongluo@ust.hk, {202120130414,202010106028}@mail.scut.edu.cn,
dlzeng@scut.edu.cn

Abstract

Continuous normalizing flows (CNFs) construct invertible mappings between an arbitrary complex distribution and an isotropic Gaussian distribution using Neural Ordinary Differential Equations (neural ODEs). It has not been tractable on large datasets due to the incremental complexity of the neural ODE training. Optimal Transport theory has been applied to regularize the dynamics of the ODE to speed up training in recent works. In this paper, a temporal optimization is proposed by optimizing the evolutionary time for forward propagation of the neural ODE training. In this appoach, we optimize the network weights of the CNF alternately with evolutionary time by coordinate descent. Further with temporal regularization, stability of the evolution is ensured. This approach can be used in conjunction with the original regularization approach. We have experimentally demonstrated that the proposed approach can significantly accelerate training without sacrifying performance over baseline models.

1. Introduction

As an impressive example of unsupervised learning, deep generative models have exhibited powerful modeling performance across a wide range of tasks, including variational autoencoders (VAE) [1], Generative Adversarial Nets (GAN) [2], autoregressive models [3] and Flow-based models [4, 5].

Generative models based on normalizing flows have been recently realized with great success in the problems of probabilistic modeling and inference. Normalizing flows [6] provide a general and extensible framework for modelling highly complex and multimodal distributions through a series of differentiable and invertible transformations. Since these transformations are invertible, the framework of normalizing flows allows for powerful exact density estimation by computing the Jacobian determinant [7]. Like a fluid flowing through a set of tubes, the initial density 'flows' through the sequence of invertible mappings by repeatedly applying the rule for change of variables until a desired probability at the end of this sequence is obtained. Normalizing flows are an increasingly active area of machine learning research. Applications include image generation [5, 8], noise modelling [9], video generation [10], audio generation [11–13], graph generation [14], reinforcement learning [15–17], computer graphics [18], and physics [19–23]. A key strength of normalizing flows is their expressive power as generative models due to its ability to approximate the posterior distribution arbitrarily, while maintaining explicit parametric forms.

Thanks to recent advances in deep generative architectures using maximum likelihood estimation and approximate approach like VAE for large-scale probabilistic models, continuous normalizing flows (CNF) obtained by solving an ordinary differential equations (ODE) were later developed in neural ODEs [24]. Neural ODEs form a family of models that approximate the ResNet architecture by using continuous-time ODEs. The neural ODE's dynamics can be chosen almost arbitrarily while ensuring invertibility. The jump to continuous-time dynamics affords a few computational benefits over its discrete-time counterpart, namely the presence of a trace in place of a determinant in the evolution formulae for the density, as well as the adjoint method for memory-efficient backpropagation. Due to their desirable properties, such as invertibility and parameter efficiency, neural ODEs have attracted increasing attention recently. For example, Grathwohl et al [25] proposed a neural ODE-based generative model—the FFJORD—to solve inverse problems; Quaglino et al [26] used a higher-order approximation of the states in a neural ODE,and proposed the SNet to accelerate computation. Further algorithmic improvements to the framework were presented by YAN et al [27] and Anumasa et al [28], exploring the robustness properties of neural ODEs. Effective neural ODE

*Equal contribution. Order determined by coin toss.
†Corresponding author: Delu Zeng.

architectures remain the subject of ongoing research — see for example [29–31].

Training neural ODEs consists of minimizing a loss function over the network weights subject to the nonlinear ODE constraint. To some extent, training can be seen as an optimal control problem. Applying optimal control theory to improve the training has become an appealing research area and more attention has been paid in recent years. For example, Pontryagin's maximum principle has been used to efficiently train networks with discrete weights [32], multigrid methods have been proposed to parallelize forward propagation during training [33], and analyzing the convergence on the continuous and discrete level has led to novel architectures [34]. Our goal in this paper is to extend this discussion by optimizing the integral interval of ODEs and perform similar experiments for continuous normalizing flows using neural ODEs from a novel perspective.

In summary, our contributions are as follows:

- Firstly, we are the first to propose an improved algorithm based on temporal optimization, which is simple yet effective in significantly boosting the training of neural ODEs. We find that the temporal optimization can attain competitive performance compared to original models but with significantly less training time.
- Secondly, we introduce temporal regularization and clipping function, which effectively stablize the training process and do not result in degradation of model performance.
- Moreover, we optimize the stopping time T alternately with the parameters $\boldsymbol{\theta}$ of the movement speed f, and end up with more compatible T and $\boldsymbol{\theta}$ to cause less number of function evaluation (NFE), and also obtain the decreasing training loss.

2. Preliminaries

2.1. Background

The data distributions encountered in real life are usually complicated, causing the essence behind the data difficult to be explored. One way often be used is to introduce the change of variables formula by $\mathbf{z} = g(\mathbf{x})$, and we have

$$\log p_{\mathbf{x}}(\mathbf{x}) = \log q_{\mathbf{z}}(\mathbf{z}) + \log |\det \nabla g(\mathbf{x})| \,, \tag{1}$$

where $g : \mathbb{R}^D \to \mathbb{R}^D$ is bijective, ∇g is the Jacobian of g and $\det(\cdot)$ is its determinant, $p_{\mathbf{x}}(\mathbf{x})$ and $q_{\mathbf{z}}(\mathbf{z})$ are the distributions of x and z, respectively. In this way, we can warp the distribution $p_{\mathbf{x}}(\mathbf{x})$ into $q_{\mathbf{z}}(\mathbf{z})$.

In practice, the computational cost to calculate the determinant is $\mathcal{O}(D^3)$, which is the main bottleneck of using Eqn.(1). Alteratively, Chen et al [24] use continuous normalizing flow (CNF) to characterize the recursively continuous transformations instead of characterizing g directly in (1), and it's computational cost is $\mathcal{O}(D^2)$. For this method, a so called instantaneous change of variables formula is obtained as follows:

$$\begin{aligned} \partial_t \begin{bmatrix} \mathbf{z}(t) \\ \log p(\mathbf{z}(t)) \end{bmatrix} &= \begin{bmatrix} f(\mathbf{z}(t), t; \boldsymbol{\theta}) \\ -\operatorname{Tr}(\mathbf{J}(t, \boldsymbol{\theta})) \end{bmatrix}, \\ \begin{bmatrix} \mathbf{z}(t_0) \\ \log p(\mathbf{z}(t_0)) - \log p_{\mathbf{x}}(\mathbf{x}) \end{bmatrix} &= \begin{bmatrix} \mathbf{x} \\ \mathbf{0} \end{bmatrix}, \end{aligned} \tag{2}$$

where $t \in [t_0, T]$ and $\boldsymbol{\theta}$ are parameters of f which is called the movement speed and is a neural network being trained, $\mathbf{J}(t, \boldsymbol{\theta}) = \frac{\partial f(\mathbf{z}(t), t; \boldsymbol{\theta})}{\partial \mathbf{z}(t)}$ is the partial derivative of $f(\mathbf{z}(t), t; \boldsymbol{\theta})$ w.r.t $\mathbf{z}(t)$.

Then by integrating across time from t_0 to T, the change of $\mathbf{z}$ can be derived:

$$\begin{aligned} \mathbf{z}(T) &= \mathbf{z}(t_0) + \int_{t_0}^{T} f(\mathbf{z}(t), t; \theta) \mathrm{d}t \\ &\approx \mathbf{z}(t_0) + \sum_{i=1}^{N} f(\mathbf{z}(t_i), t_i; \theta) \Delta t_i, \end{aligned} \tag{3}$$

where N denote the number of function evaluations (NFEs).

If training the dynamics (2) from the perspective of maximum likelihood estimation, we can switch the estimation of likelihood from $\mathbf{x}$ to $\mathbf{z}$. If $\mathbf{z}$ is an isotropic Gaussian variable, then the likelihood of $\mathbf{x}$ can be computed easily by integrating across time:

$$\begin{aligned} \min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} L(\boldsymbol{\theta}) &= -\mathbb{E}_{p_{\mathbf{x}}} \{\log p(\mathbf{z}(t_0); \boldsymbol{\theta})\} \\ &= -\mathbb{E}_{p_{\mathbf{x}}} \left\{ \log p(\mathbf{z}(T); \boldsymbol{\theta}) + \int_{t_0}^{T} Tr(\mathbf{J}(t, \boldsymbol{\theta})) dt \right\}, \end{aligned} \tag{4}$$

where $\mathbf{x} = \mathbf{z}(t_0)$.

Furthermore, Grathwohl et al [25] use the Hutchinson's trace estimator [35] and Onken et al [36] design a refined network structure to compute the trace in Eqn.(2), where the cost are both reduced to $\mathcal{O}(D)$.

Despite the calculation of the log determinant for a single f becomes faster, there still remain some hurdles causing the total evolutionary time unacceptable, like complex structure of f and undesirable large NFEs that increase over time.

To accelerate the training process of CNF, Finlay et al [37] and Onken et al [36] introduce several regularization related to f based on the optimal transport (OT) theory. Both of them add transport loss $OT(\boldsymbol{\theta})$ to the objective function, which can be described as:

$$OT(\boldsymbol{\theta}) = \int_{t_0}^{T} \int_{\mathbb{R}^D} \|f(\mathbf{z}(t), t; \boldsymbol{\theta})\|^2 \, p(\mathbf{z}(t)) \mathrm{d}\mathbf{z}\mathrm{d}t. \tag{5}$$

Hence, the objective function becomes:

$$\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \{L(\boldsymbol{\theta}) + OT(\boldsymbol{\theta})\}. \tag{6}$$

Besides, they treat the evolutionary process of $\mathbf{z}$ as the motion of particles and limit the speed of particles $f(\mathbf{z}(t), t; \boldsymbol{\theta})$ at t from different angles.

However, within the above methods, the stopping time, T, is treated as a fixed hyperparameter. Given T, they have been trying to find the optimal dynamics depending on network weights $\boldsymbol{\theta}$.

2.2. Related work

Finite Flows Normalizing flows [6, 38–40] use a finite number of transformations to construct differentiable bijections between complex unknown distributions and simple distributions. NICE [41] and REALNVP [4] first use coupling layers to construct the transformation, thus ensuring the reversibility of the model. Improved from REALNVP, a 1×1 convolution has been introduced in GLOW [5] to increase the flexibility of the model. Then, an attention mechanism has been developed in FLOW++ [8] to obtain a more expressive architecture. An autoregressive structure has been proposed in IAF [42] and MAF [43] to enhance the expressiveness of the model. Benefiting from numerous improvements in autoregressive flows [44–47], its expressive power is gradually recognized in flow-based models.

Infinitesimal Flows Inspired from the presentation of Resnet [48], many recent works [24, 49] have used ordinary differential equations to construct invertible transformations between random variables. A more flexible architecture and lower computational complexity of Jacobian determinant are obtained in FFJORD [25] by means of unbiased trace estimation. Augmented-NODE [29] and NANODE [50] used increased dimensionality to lift the restriction that the trajectories of ordinary differential equations cannot intersect. Most continuous flows back-propogate and update the gradient via the adjoint method, saving memory while back-propogating inaccurate state values. Some models introduce a check-point mechanism to store some of the state nodes during forward propogation to solve the backward integral accurately [30, 31, 51]. These infinite-depth approaches from a novel perspective theoretically bridges the gap between deep learning and dynamical systems. In addition, Neural ODEs have shown great promise for numerous tasks such as image registration [52], video generation [53], reinforcement learning [54] and system identification [26]. Recent work has also extended neural ODEs to stochastic differential equations [55, 56], Riemannian manifold [57], Bayesian learning frameworks [58] and graph-structured data [59].

Flows with Optimal Transport To enforce straight trajectories and accelerate training, RNODE [37] and OT-FLOW [36] regularized the FFJORD model by adding a transport cost in the form of L_2 norm to the original loss function. RNODE also introduced the Frobenius norm to stablize training. Furthermore, Tay-NODE [60] generalized the form of the L_2 norm and obtained a regularization term of arbitrary order, but is slower to train than RNODE because of the extra computational cost introduced. TNODE [61], on the other hand, proposed a novel view to regularize trajectories into polynomials. STEER [62], similar to our method, also optimizes time, but only by random sampling of end time, whereas our method optimizes by coordinate descent as described in Section 3.1.

Algorithm 1 log-density estimaion using the TO-FLOW

```
Input: dynamics $f_{\boldsymbol{\theta}}$, start time $t_0$, initial stopping time $T_0$, minibatch of samples $\mathbf{x}$, number of iterations $n$, network optimizer $\mathcal{P}$, temporal optimizer $\mathcal{Q}$.
  Initialize: $\mathbf{z}(t_0) = \mathbf{x}, T = T_0$
  for $i = 1 \to n$ do
      $[\mathbf{z}(T), -\hat{\text{Tr}}] \leftarrow \text{odeint}(f_{\boldsymbol{\theta}}, [\mathbf{x}, \mathbf{0}], t_0, T)$   ◇ Solve the ODE
      $\hat{\boldsymbol{\theta}} \leftarrow \mathcal{P}(\nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\mathbf{x}}}\{-\log p(\mathbf{z}(t_0))\}, \boldsymbol{\theta})$   ◇ Update the network weights
      $\hat{T} \leftarrow \mathcal{Q}(\partial_T \mathbb{E}_{p_{\mathbf{x}}}\{-\log p(\mathbf{z}(t_0))\}, T)$   ◇ Update the stopping time
  end for
Output: final dynamics $f_{\hat{\boldsymbol{\theta}}}$, stopping time $\hat{T}$
```

3. Method

Motivated by the similarities between training CNFs and solving OT problems [63, 64], some previous works regularize the minimization problem (5) to enforce a straight trajectory and significantly faster training [37, 60, 61].

From another perspective, if we express Eqn. (2) explicitly by the first-order Euler method:

$$\mathbf{z}(t_i + \Delta t_i) = \mathbf{z}(t_i) + f(\mathbf{z}(t_i), t_i; \boldsymbol{\theta})\Delta t_i \tag{7}$$

$$T = t_0 + \sum_{i=1}^{n} \Delta t_i, \tag{8}$$

where n denote total number of time steps. It is clear that total evolutinary time $T - t_0$ interacts with $f(\mathbf{z}(t), t; \boldsymbol{\theta})$ to infuence the evolution of $\mathbf{z}(t)$ in an intricate way. Without placing demands on the total evolutionary time $T - t_0$, an under-regularized trajectory is formed and results in unnecessarily large training time [62].

How can the regularization of the total evolutionary time be designed? The formulation of continuous normalizing flows is given by (3) where t_0 and T are both hyperparameters that fixed before training. One intuitive approach is to optimize t_0 and T together to find an appropriate integral

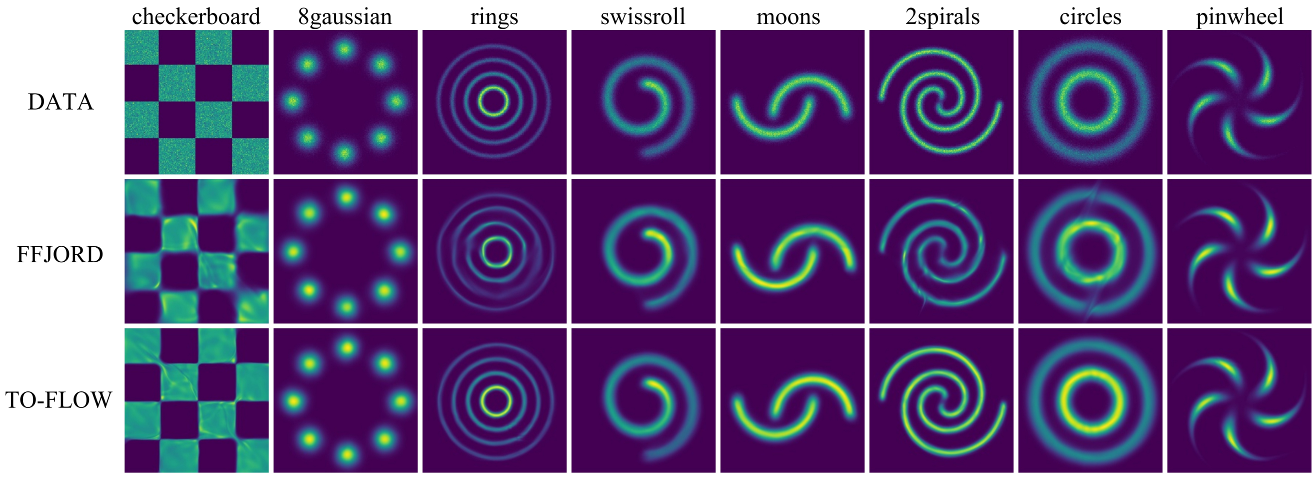

Figure 1. Comparison of FFJORD and TO-FLOW on 2-dimensional distributions.

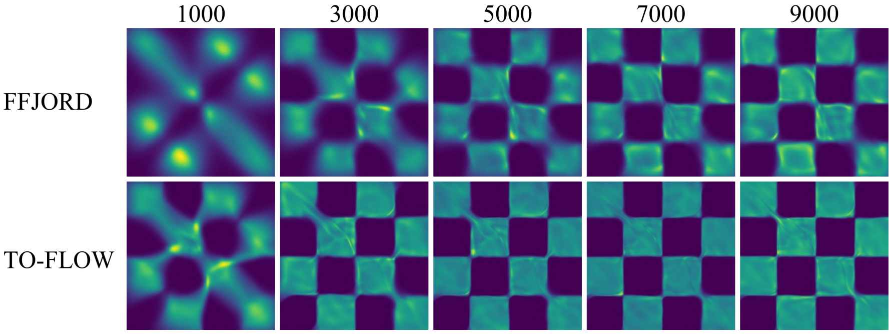

Figure 2. Comparison of FFJORD and TO-FLOW on checkerboard data set. The numbers at the top of the images represent the number of iterations of the model.

horizon. For simplicity, we fix t_0 and only optimize T (for the derivation of the generic form, see App.A).

Then combing with the optimization of the moving speed depending on θ, we could revise the optimization problem 4 as follows:

$$\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}, T\in\mathbb{R}} \{L(\boldsymbol{\theta}, T)\}. \tag{9}$$

3.1. Coordinate descent

The problem is split into two smaller subproblems: one trains the network weights $\boldsymbol{\theta}$, the other optimizes the stopping time T to form an appropriate trajectory. In the following, the details of each subproblem are discussed.

Step 1: Training of the network weights. The initial network weights are chosen randomly, then updated by fixing the stopping time $\tilde{T}$ and solving the subproblem:

$$\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} L(\boldsymbol{\theta}, \tilde{T}) = -\mathbb{E}_{p_{\mathbf{x}}}\{\log p(\mathbf{z}(t_0); \boldsymbol{\theta})\}, \tag{10}$$

Once the objective function is solved, then we calculate the gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \tilde{T})$ to update the network weights $\boldsymbol{\theta}$ with some general SGD-like optimizer $\mathcal{P}$, such as Adam [65].

Step2: Coordinate descent on evolutionary time. Once the network weights $\boldsymbol{\theta}$ are updated, we then fix the updated network weights $\tilde{\theta}$, and solve the subproblem:

$$\begin{aligned}\min_{T\in\mathbb{R}} L(\tilde{\boldsymbol{\theta}}, T) &= -\mathbb{E}_{p_{\mathbf{x}}}\{\log p(\mathbf{z}(t_0); \tilde{\boldsymbol{\theta}})\} \\ &= -\mathbb{E}_{p_{\mathbf{x}}}\left\{\log p(\mathbf{z}(T); \tilde{\boldsymbol{\theta}}) + \int_{t_0}^{T} Tr(\mathbf{J}(t, \tilde{\boldsymbol{\theta}}))dt\right\},\end{aligned} \tag{11}$$

The stopping time T is updated by calculating the derivative of $L(\tilde{\boldsymbol{\theta}}, T)$ with respect to T:

$$\frac{\partial L(\tilde{\boldsymbol{\theta}}, T)}{\partial T} = -\frac{\partial \mathbb{E}_{p_{\mathbf{x}}}\{\log p(\mathbf{z}(T); \tilde{\boldsymbol{\theta}})\}}{\partial T} - \mathbb{E}_{p_{\mathbf{x}}}\{Tr(\mathbf{J}(T, \tilde{\boldsymbol{\theta}}))\} \tag{12}$$

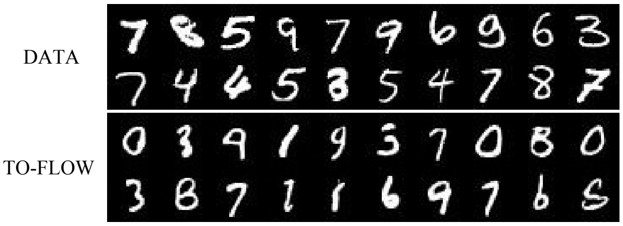

Figure 3. Samples of MNIST data set.

Figure 4. Samples of CIFAR-10 data set.

Figure 5. Samples of FASHION-MNIST data set.

However, partial derivative $\frac{\partial \log p}{\partial T}$ cannot be derived directly, since we only encode the stopping time T onto the channels of the feature map without introducing a direct correspondence. Instead, we introduce the chain rule to obtain a feasible calculation:

$$\frac{\partial \mathbb{E}_{p_{\mathbf{x}}}\{\log p(\mathbf{z}(T);\tilde{\boldsymbol{\theta}})\}}{\partial T} = \mathbb{E}_{p_{\mathbf{x}}}\left\{\frac{\partial \log p(\mathbf{z}(T);\tilde{\boldsymbol{\theta}})}{\partial \mathbf{z}(T)} \circ \frac{\partial \mathbf{z}(T)}{\partial T}\right\}, \tag{13}$$

where $\circ$ denotes the dot product and $\frac{\partial \mathbf{z}(T)}{\partial T} = f(\mathbf{z}(T), T; \tilde{\boldsymbol{\theta}})$, which is relatively well calculated.

Once the derivative is calculated, we update T also with some SGD-like optimizer $\mathcal{Q}$, such as Adam [65]:

$$\hat{T} = \mathcal{Q}\left(\frac{\partial L(\tilde{\boldsymbol{\theta}}, T)}{\partial T}, T\right), \tag{14}$$

where $\widetilde{T}$ denote the stopping time updated in one iteration. Pseudo-code of our method is given in Algorithm 1.

3.2. Temporal regularization

Within our experiments, we find that T changes quickly at the beginning of training. Finlay et al [37] and Onken et al [36] constrain the movement speed $f(\mathbf{z}(t), t; \boldsymbol{\theta})$ of particles to reduce the loss of transmission from the perspective of OT theory. Inspired by their work, we add constraints to the total evolutionary time $T - t_0$ to stablize training. This trick is called temporal regularization (TR), which can be described as:

$$TR(T) = \alpha \cdot |T|, \tag{15}$$

where α denotes the power of TR on the training process of CNF. Then the total objective function becomes:

$$\min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}, T\in\mathbb{R}} \{L(\boldsymbol{\theta}, T) + TR(T)\}. \tag{16}$$

How does TR affect the training process of CNF? It can be seen in the left side of Figure 6 and 7 that a smaller α can lead to instability in the training process. Hence the hyperparameter α can be measured as the strength of temporal regularization.

We also propose an operation of applying a clipping function to the stopping time T. In this case, the T obtained at each iteration of the model is in the interval $[t_0 + \varepsilon, 2T_0 - t_0 - \varepsilon]$, the center of which is T_0. Empirically, an obvious advantage of this is that T will not evolve drastically during each iteration. The so called clipping function at each iteration is defined as:

$$\text{Clip}\,(T) = \begin{cases} 2T_0 - t_0 - \varepsilon & T \geq 2T_0 - t_0 - \varepsilon \\ t_0 + \varepsilon & T \leq t_0 + \varepsilon \\ T & \text{otherwise} \end{cases} \tag{17}$$

where ε is the clipping parameter. t_0 and T_0 denote the initial value of the lower and upper bound of the integral, respectively.

4. Experiments

We demonstrate the benefits of the proposed method on a variety of density estimation tasks. We compare our results with FFJORD [25], the baseline of our method, and STEER [62], another model that only by random sampling the stopping time.

Two metrics are evaluated, testing loss and training time. We want to see whether our model leads to faster training process compared to FFJORD and STEER in training, while keeping comparable training quality.

To compare the training speed, we count total training time and average time per training iteration. We also count the average NFE per training iteration. NFE is defined as the number of function of evaluating the right-hand-side of the ODE 2 when solving it. The lower NFE, the faster training speed.

Data Set	Model	Bits/dim	Param	Time(h)	Iter	Time/Iter(s)	NFE
MNIST	FFJORD	1.017	400K	79.641	60K	6.409	750.67
	STEER	1.024	400K	138.212	60K	12.368	1265.48
	TO-FLOW (ours)	1.026	400K	**46.363**	60K	**3.353**	**396.81**
Fashion-MNIST	FFJORD	2.806	400K	87.845	60K	7.010	811.40
	STEER	2.803	400K	147.197	60K	12.405	1308.82
	TO-FLOW (ours)	2.807	400K	**63.482**	60K	**5.415**	**513.79**
CIFAR-10	FFJORD	3.414	670K	108.314	50K	10.299	1228.04
	STEER	3.424	670K	168.649	50K	15.502	1749.17
	TO-FLOW (ours)	3.429	670K	**82.607**	50K	**7.373**	**716.85**

Table 1. Density estimation on image data sets. We present the testing loss (Bits/dim), number of parameters (Param), total training time (Time), total number of iterations (Iter), average time per iteration (Time/Iter) and average number of function evaluations (NFE). We use moving average instead of summation average [25].

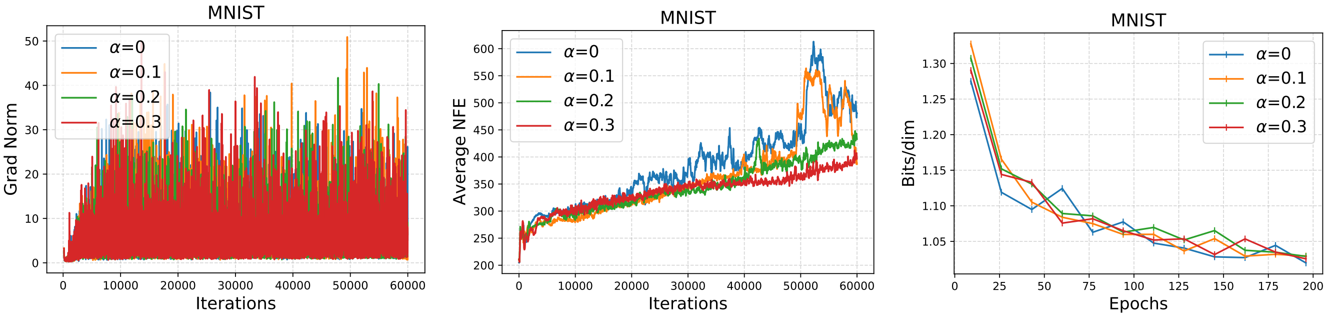

Figure 6. Model performance under different temporal regularization on MNIST data set.

To evaluate the training quality, we compute bits/dim as a metric:

$$\mathrm{BPD} = -\mathbb{E}_{p_{\boldsymbol{x}}}\left\{\frac{\log \hat{p}(x)/d - \log 256}{\log 2}\right\}, \tag{18}$$

where $\log \hat{p}(x)$ denote estimated log-likelihood of our model and d denote the dimension of data. It is a classical metric to measure the approximation of the distribution transformed by flow-based model to the isotropic Gaussian distribution. A low BPD value means that the model can effectively transform an unknown data distribution into a simple known distribution.

In all experiments, we use exactly the same architectures of neural network as in FFJORD. What we do is integrating our temporal optimization to training process. The experiment settings of temporal optimization are described below. For temporal optimizer, we choose Adam [65] as an optimizer and set the learning rate $lr = 10^{-2}$. The initial value of the stopping time T_0 is set to be the same fixed value as in FFJORD, which is 0.5 in toy data and 1 in image data. The hyperparameter of the temporal regularization is $\alpha = 0.1$. The hyperparameter of the clipping function is $\epsilon = 0.1$. These hyper-parameters above are shared in all experiments. Furthermore, we discuss the influences of different choice of hyperparameters in Section 5

4.1. Density estimation on toy 2D data

We first train TO-FLOW on eight simple 2D toy data that serve as standard benchmarks [25]. In Figure 1, we show that TO-FLOW can fit both multi-modal and discontinuous distributions compared against FFJORD by warping a simple isotropic Gaussian.

The distributions of the eight 2D data used for the experiment are shown in the first row of Figure 1. The learned distributions using FFJORD and our method are shown in the second row and the last row, respectively. Both FFJORD and TO-FLOW trained 10000 iterations for a fair comparison. For checkerboard, rings, 2spirals and circles, our model produces images with a higher degree of reduction, which also shows that our approach can learn multi-modal and discontinuous distributions more efficiently. We compare different stages of FFJORD and TO-FLOW on checkerboard data set in Figure 2. For a more detailed comparison of different stages during training, see App.B.

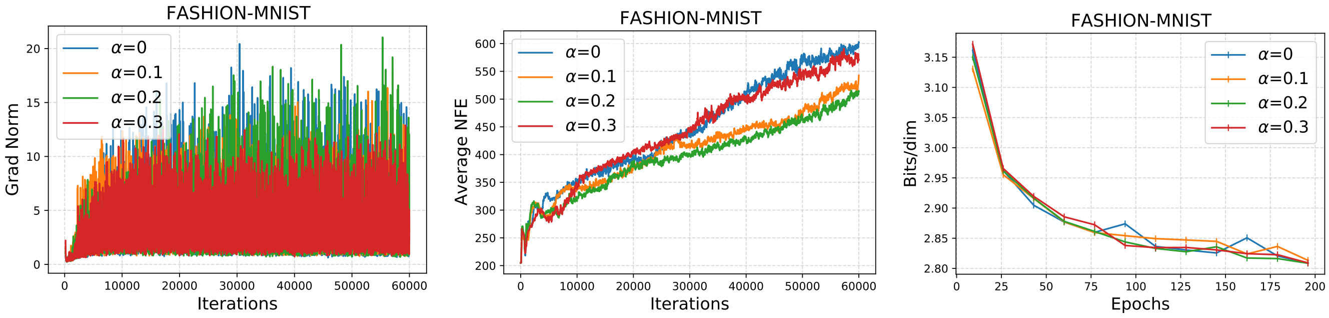

Figure 7. Model performance under different temporal regularization on FASHION-MNIST data set.

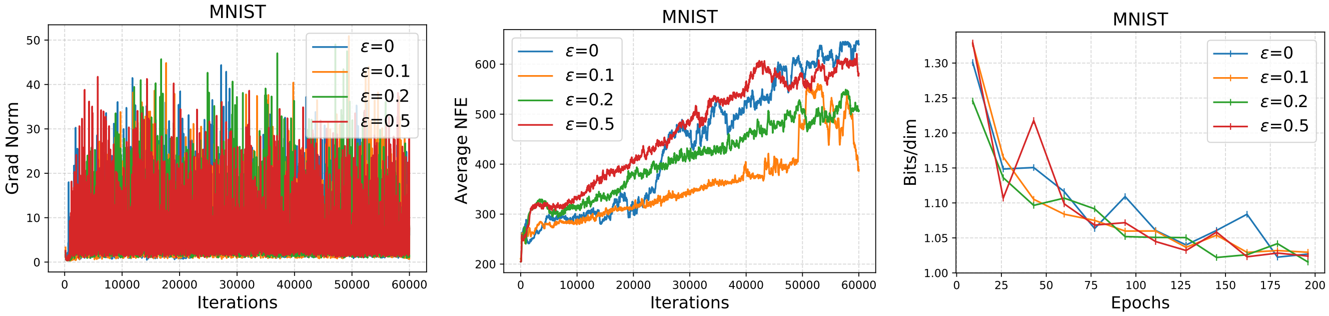

Figure 8. Model performance via different clipping function on MNIST data set.

4.2. Density estimation on image data

We compare our model's performance on three image data sets: MNIST [66], CIFAR-10 [67] and FASHION-MNIST [68]. On three data sets, we train with a batch size of 200 and train for 200 epochs on a single GPU[1].

A comparison of TO-FLOW against FFJORD and STEER[2] is presented in Table 1. For training speed, our model significantly outperforms FFJORD and STEER. On all data sets, our model uses fewer NFE and a shorter training time, which also leads to a faster convergence of the model. The reduction of total training time ranges from 23.7% to 41.7%. On some large data sets, such as CIFAR-10, our model is 23.7% faster than the baseline model, which demonstrates the great potential of our model to scale to larger datasets.

For testing loss (18), our model is also comparable with FFJORD and STEER, which shows that there is no performance penalty for adding temporal optimization. We also visualize the images generated by our model in Figure 3, 4 and 5. As can be seen, the introduction of temporal optimization still maintains the quality of image generation. More images generated by different settings can be seen in App.C.

In general, the proposed approach results in comparable performance in tesing loss, but significantly speeds up training. It allows us to use larger network structures and batch sizes, which also preserves the possibility of further performance improvements.

5. Analysis and discussion

We perform a series of ablation experiments to gain more insight into our model.

5.1. Stable training via temporal regularization

Grad norm denote the clipped portion of the norm of the gradient of the network parameters and is often used in training to characterize the stability of the training process. We compare the performance of our model under different coeffieient of temporal regularization on MNIST and FASHION-MNIST. We plot the grad norm in the left side of Figure 6 and 7. We plot average NFE and testing loss in the middle and right side of Figure 6 and 7 respectively to measure the effect of temporal regularization on training speed and density estimation. We find that the introduction of temporal regularization significantly stablizes the training process while maintaining training speed and the accuracy of density estimation.

[1] We use Tesla-V100 for all experiments.

[2] We obtained results that differ from the original paper. Since the author did not release their code, We will continue to try other strategies to replicate their results.

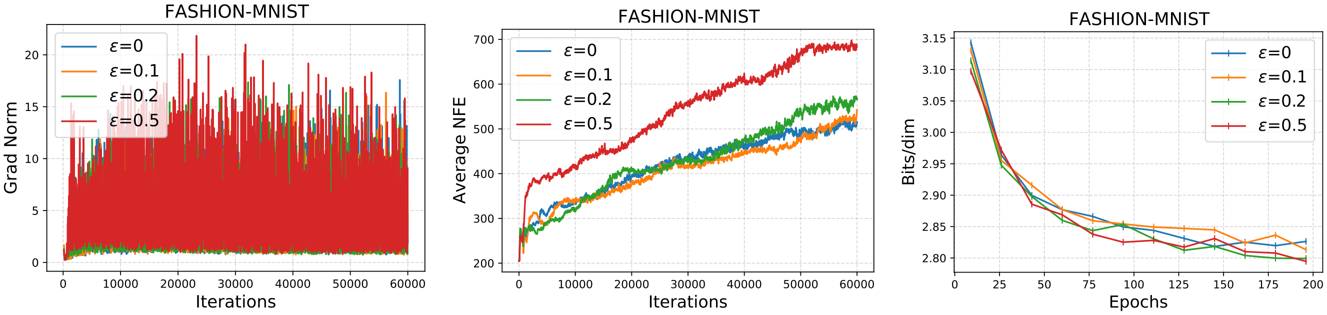

Figure 9. Model performance via different clipping function on FASHION-MNIST data set.

5.2. The choice of clipping function

The performance of Temporal Optimization is closely related to the choice of clipping parameter ϵ since it represents the boundary of the evolutionary time $T - t_0$. We compare the model performance of different size of clipping parameter ϵ and plot in Figure 8 and 9 respectively. We can conclude that a more compact boundary also leads to more stable training and does not result in degradation of model performance.

5.3. Future work

Finlay et al [37] and Onken et al [36] are dedicated to optimising trajectory spatially, thus constraining it to be straight to improve training speed. In this paper, We optimize the trajectory in terms of time, which also has the effect of significantly accelerating training. An intuitive idea is to simply combine the above spatial optimization models with our approach. How to combine temporal and spatial optimization more effectively remains the focus of subsequent research.

6. Conclusion

We have presented TO-FLOW, a model that optimizes time and does not introduce additional computational cost. Excessive computational costs are a major bottleneck to scaling CNF to large applications. We integrate an additional temporal optimization step to the training process, which regularizes the trajectory from another perspective and significantly improves computational efficiency. Furthermore, our method is compatible with other regularization methods and can be applied to other more expressive architectures to further improve performance.

Acknowledgements: This work was supported in part by grants from National Science Foundation of China (61571005), the fundamental research program of Guangzhou, China (2020B1515310023).

References

[1] Diederik P Kingma and Max Welling. Stochastic gradient vb and the variational auto-encoder. In *Second International Conference on Learning Representations, ICLR*, volume 19, page 121, 2014. 1

[2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 1

[3] Dzmitry Bahdanau, Dmitriy Serdyuk, Philemon Brakel, Nan Rosemary Ke, Jan Chorowski, Aaron Courville, and Yoshua Bengio. Task loss estimation for sequence prediction. *arXiv preprint arXiv:1511.06456*, 2015. 1

[4] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016. 1, 3

[5] Diederik P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018. 1, 3

[6] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015. 1, 3

[7] Erhan Çınlar and Robert J Vanderbei. *Real and Convex Analysis*. Springer Science & Business Media, 2013. 1

[8] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *International Conference on Machine Learning*, pages 2722–2730. PMLR, 2019. 1, 3

[9] Abdelrahman Abdelhamed, Marcus A Brubaker, and Michael S Brown. Noise flow: Noise modeling with conditional normalizing flows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3165–3173, 2019. 1

[10] Manoj Kumar, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, and Durk Kingma. Videoflow: A flow-based generative model for video. *arXiv preprint arXiv:1903.01434*, 2(5), 2019. 1

[11] Philippe Esling, Naotake Masuda, Adrien Bardet, Romeo Despres, et al. Universal audio synthesizer control with normalizing flows. *arXiv preprint arXiv:1907.00971*, 2019. 1

[12] Sungwon Kim, Sang-gil Lee, Jongyoon Song, Jaehyeon Kim, and Sungroh Yoon. Flowavenet: A generative flow for raw audio. *arXiv preprint arXiv:1811.02155*, 2018. 1

[13] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621. IEEE, 2019. 1

[14] Kaushalya Madhawa, Katushiko Ishiguro, Kosuke Nakago, and Motoki Abe. Graphnvp: An invertible flow model for generating molecular graphs. *arXiv preprint arXiv:1905.11600*, 2019. 1

[15] Bogdan Mazoure, Thang Doan, Audrey Durand, Joelle Pineau, and R Devon Hjelm. Leveraging exploration in off-policy algorithms via normalizing flows. In *Conference on Robot Learning*, pages 430–444. PMLR, 2020. 1

[16] Patrick Nadeem Ward, Ariella Smofsky, and Avishek Joey Bose. Improving exploration in soft-actor-critic with normalizing flows policies. *arXiv preprint arXiv:1906.02771*, 2019. 1

[17] Ahmed Touati, Harsh Satija, Joshua Romoff, Joelle Pineau, and Pascal Vincent. Randomized value functions via multiplicative normalizing flows. In *Uncertainty in Artificial Intelligence*, pages 422–432. PMLR, 2020. 1

[18] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *ACM Transactions on Graphics (TOG)*, 38(5):1–19, 2019. 1

[19] Danilo Jimenez Rezende, George Papamakarios, Sébastien Racanière, Michael Albergo, Gurtej Kanwar, Phiala Shanahan, and Kyle Cranmer. Normalizing flows on tori and spheres. In *International Conference on Machine Learning*, pages 8083–8092. PMLR, 2020. 1

[20] Jonas Köhler, Leon Klein, and Frank Noé. Equivariant flows: exact likelihood generative learning for symmetric densities. In *International Conference on Machine Learning*, pages 5361–5370. PMLR, 2020. 1

[21] Victor Garcia Satorras, Emiel Hoogeboom, Fabian B Fuchs, Ingmar Posner, and Max Welling. E (n) equivariant normalizing flows for molecule generation in 3d. *arXiv preprint arXiv:2105.09016*, 2021. 1

[22] Peter Wirnsberger, Andrew J Ballard, George Papamakarios, Stuart Abercrombie, Sébastien Racanière, Alexander Pritzel, Danilo Jimenez Rezende, and Charles Blundell. Targeted free energy estimation via learned mappings. *The Journal of Chemical Physics*, 153(14):144112, 2020. 1

[23] Kaze WK Wong, Gabriella Contardo, and Shirley Ho. Gravitational-wave population inference with deep flow-based generative network. *Physical Review D*, 101(12):123005, 2020. 1

[24] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2018. 1, 2, 3

[25] Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018. 1, 2, 3, 5, 6

[26] Alessio Quaglino, Marco Gallieri, Jonathan Masci, and Jan Koutník. Snode: Spectral discretization of neural odes for system identification. *arXiv preprint arXiv:1906.07038*, 2019. 1, 3

[27] Hanshu Yan, Jiawei Du, Vincent YF Tan, and Jiashi Feng. On robustness of neural ordinary differential equations. *arXiv preprint arXiv:1910.05513*, 2019. 1

[28] Srinivas Anumasa and PK Srijith. Improving robustness and uncertainty modelling in neural ordinary differential equations. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 4053–4061, 2021. 1

[29] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. *arXiv preprint arXiv:1904.01681*, 2019. 2, 3

[30] Amir Gholami, Kurt Keutzer, and George Biros. Anode: Unconditionally accurate memory-efficient gradients for neural odes. *arXiv preprint arXiv:1902.10298*, 2019. 2, 3

[31] Juntang Zhuang, Nicha Dvornek, Xiaoxiao Li, Sekhar Tatikonda, Xenophon Papademetris, and James Duncan. Adaptive checkpoint adjoint method for gradient estimation in neural ode. In *International Conference on Machine Learning*, pages 11639–11649. PMLR, 2020. 2, 3

[32] Qianxiao Li, Long Chen, Cheng Tai, et al. Maximum principle based algorithms for deep learning. *arXiv preprint arXiv:1710.09513*, 2017. 2

[33] Stefanie Gunther, Lars Ruthotto, Jacob B Schroder, Eric C Cyr, and Nicolas R Gauger. Layer-parallel training of deep residual neural networks. *SIAM Journal on Mathematics of Data Science*, 2(1):1–23, 2020. 2

[34] Martin Benning, Elena Celledoni, Matthias J Ehrhardt, Brynjulf Owren, and Carola-Bibiane Schönlieb. Deep learning as optimal control problems: Models and numerical methods. *arXiv preprint arXiv:1904.05657*, 2019. 2

[35] Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989. 2

[36] Derek Onken, Samy Wu Fung, Xingjian Li, and Lars Ruthotto. Ot-flow: Fast and accurate continuous normalizing flows via optimal transport. *arXiv preprint arXiv:2006.00104*, 2020. 2, 3, 5, 8

[37] Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam Oberman. How to train your neural ode: the world of jacobian and kinetic regularization. In *International Conference on Machine Learning*, pages 3154–3164. PMLR, 2020. 2, 3, 5, 8

[38] Esteban G Tabak and Cristina V Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013. 3

[39] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019. 3

[40] Ivan Kobyzev, Simon Prince, and Marcus Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. 3

[41] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014. 3

[42] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29:4743–4751, 2016. 3

[43] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *arXiv preprint arXiv:1705.07057*, 2017. 3

[44] Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. In *International Conference on Machine Learning*, pages 2078–2087. PMLR, 2018. 3

[45] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. *Advances in Neural Information Processing Systems*, 32:7511–7522, 2019. 3

[46] Zachary Ziegler and Alexander Rush. Latent normalizing flows for discrete sequences. In *International Conference on Machine Learning*, pages 7673–7682. PMLR, 2019. 3

[47] Antoine Wehenkel and Gilles Louppe. Unconstrained monotonic neural networks. *Advances in Neural Information Processing Systems*, 32:1545–1555, 2019. 3

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 3

[49] Tim Salimans, Diederik Kingma, and Max Welling. Markov chain monte carlo and variational inference: Bridging the gap. In *International Conference on Machine Learning*, pages 1218–1226. PMLR, 2015. 3

[50] Jared Quincy Davis, Krzysztof Choromanski, Jake Varley, Honglak Lee, Jean-Jacques Slotine, Valerii Likhosterov, Adrian Weller, Ameesh Makadia, and Vikas Sindhwani. Time dependence in non-autonomous neural odes. *arXiv preprint arXiv:2005.01906*, 2020. 3

[51] Juntang Zhuang, Nicha C Dvornek, Sekhar Tatikonda, and James S Duncan. Mali: A memory efficient and reverse accurate integrator for neural odes. *arXiv preprint arXiv:2102.04668*, 2021. 3

[52] Junshen Xu, Eric Z Chen, Xiao Chen, Terrence Chen, and Shanhui Sun. Multi-scale neural odes for 3d medical image registration. *arXiv preprint arXiv:2106.08493*, 2021. 3

[53] Sunghyun Park, Kangyeol Kim, Junsoo Lee, Jaegul Choo, Joonseok Lee, Sookyung Kim, and Edward Choi. Vid-ode: Continuous-time video generation with neural ordinary differential equation. *arXiv preprint arXiv:2010.08188*, 2020. 3

[54] Jianzhun Du, Joseph Futoma, and Finale Doshi-Velez. Model-based reinforcement learning for semi-markov decision processes with neural odes. *arXiv preprint arXiv:2006.16210*, 2020. 3

[55] Xuanqing Liu, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. Neural sde: Stabilizing neural ode networks with stochastic noise. *arXiv preprint arXiv:1906.02355*, 2019. 3

[56] Viktor Oganesyan, Alexandra Volokhova, and Dmitry Vetrov. Stochasticity in neural odes: An empirical study. *arXiv preprint arXiv:2002.09779*, 2020. 3

[57] Emile Mathieu and Maximilian Nickel. Riemannian continuous normalizing flows. *arXiv preprint arXiv:2006.10605*, 2020. 3

[58] Mohamed Aziz Bhouri and Paris Perdikaris. Gaussian processes meet neuralodes: A bayesian framework for learning the dynamics of partially observed systems from scarce and noisy data. *arXiv preprint arXiv:2103.03385*, 2021. 3

[59] Zijie Huang, Yizhou Sun, and Wei Wang. Coupled graph ode for learning interacting system dynamics. In *Proc. of 2021 ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'21)*, 2021. 3

[60] Jacob Kelly, Jesse Bettencourt, Matthew James Johnson, and David Duvenaud. Learning differential equations that are easy to solve. *arXiv preprint arXiv:2007.04504*, 2020. 3

[61] Han-Hsien Huang and Mi-Yen Yeh. Accelerating continuous normalizing flow with trajectory polynomial regularization. *arXiv preprint arXiv:2012.04228*, 2020. 3

[62] Arnab Ghosh, Harkirat Singh Behl, Emilien Dupont, Philip HS Torr, and Vinay Namboodiri. Steer: Simple temporal regularization for neural odes. *arXiv preprint arXiv:2006.10711*, 2020. 3, 5

[63] Jean-David Benamou and Yann Brenier. A computational fluid mechanics solution to the monge-kantorovich mass transfer problem. *Numerische Mathematik*, 84(3):375–393, 2000. 3

[64] Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019. 3

[65] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4, 5, 6

[66] Yann LeCun, Corinna Cortes, and C Burges. Mnist handwritten digit database, 1998. *URL http://www. research. att. com/˜ yann/ocr/mnist*, 1998. 7

[67] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images.(2009), 2009. 7

[68] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 7