GO WITH THE FLOW: ADAPTIVE CONTROL FOR NEURAL ODES

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite their elegant formulation and lightweight memory cost, neural ordinary differential equations (NODEs) suffer from known representational limitations. In particular, the single flow learned by NODEs cannot express all homeomorphisms from a given data space to itself, and their static weight parametrization restricts the type of functions they can learn compared to discrete architectures with layer-dependent weights. Here, we describe a new module called neurally-controlled ODE (N-CODE) designed to improve the expressivity of NODEs. The parameters of N-CODE modules are dynamic variables governed by a trainable map from initial or current activation state, resulting in forms of *open-loop* and *closed-loop* control, respectively. A single module is sufficient for learning a distribution on non-autonomous flows that adaptively drive neural representations. We provide theoretical and empirical evidence that N-CODE circumvents limitations of previous models and show how increased model expressivity manifests in several domains. In supervised learning, we demonstrate that our framework achieves better performance than NODEs as measured by both training speed and testing accuracy. In unsupervised learning, we apply this control perspective to an image autoencoder endowed with a latent transformation flow, greatly improving representational power over a vanilla model and leading to state-of-the-art image reconstruction on CIFAR-10.

1 INTRODUCTION

The interpretation of artificial neural networks as continuous-time dynamical systems opens has led to both theoretical and practical advances in representation learning. According to this interpretation, the separate layers of a deep neural network are understood to be a discretization of a continuous-time operator so that, in effect, the net is infinitely deep. One important class of continuous-time models, neural ordinary differential equations (NODEs) (Chen et al., 2018), have found natural applications in generative variational inference (Grathwohl et al., 2018) and physical modeling (Köhler et al., 2019; Ruthotto et al., 2020) because of their ability to take advantage of black-box differential equation solvers and interpretation as dynamical systems.

Nevertheless, NODEs suffer from known representational limitations, which researchers have tried to alleviate either by lifting the NODE activation space to higher dimensions or by allowing the transition operator to change in time, making the system *non-autonomous* (Dupont et al., 2019) . For example, Zhang et al. (2020) showed that NODEs can arbitrarily approximate maps from $\mathbb{R}^d$ to $\mathbb{R}$ if NODE dynamics operate with an additional time dimension in $\mathbb{R}^{d+1}$ and the system is affixed with an additional linear layer. The same authors showed that NODEs could approximate homeomorphisms from $\mathbb{R}^d$ to itself if the dynamics were lifted to $\mathbb{R}^{2d}$. Yet, the set of homeomorphisms from $\mathbb{R}^d$ to itself is in fact quite a conservative function space from the perspective of representation learning, since these mappings preserve topological invariants of the data space, preventing them from "disentangling" data classes like those of the annulus data in Fig. 1 (lower left panel). In general, much remains to be understood about the continuous-time framework and its expressive capabilities.

In this paper, we propose a new approach that we call neurally-controlled ODEs (N-CODE) designed to increase the expressivity of continuous-time neural nets by using tools from control theory.

Whereas previous continuous-time methods learn a single, time-varying vector field for the whole input space, our system learns a *family* of vector fields parameterized by data. We do so by mapping the input space to a collection of control weights which then interact with neural activity to optimally steer model dynamics. These control functions form the time-varying weights of a dynamical system, making N-CODE akin to a continuous-time hypernetwork (Ha et al., 2016). The implications of this new formulation are critical for model expressivity. In particular, the transformation of the input space is no longer constrained to be a homeomorphism, since the flows associated with each datum are specifically adapted to that point. Consequently, our system can easily "tear" apart the two annulus classes in Fig. 1 (lower right panel) without directly lifting the data space to a higher dimension.

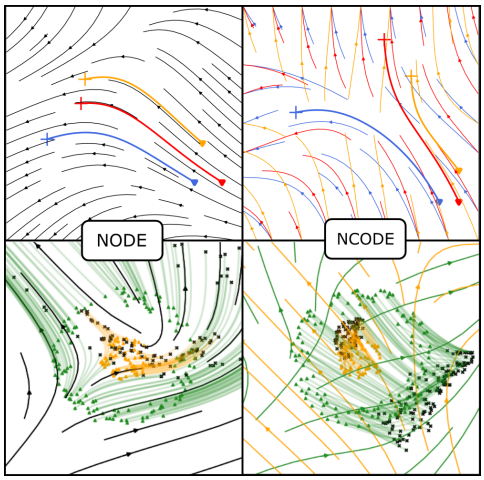

Figure 1: Vector fields for continuous-time neural networks. Integral curves with arrows show the trajectories of data points under the influence of the network. (*Top left*) Standard NODEs learn a single time-independent flow (in black) that must account for the whole data space. (*Top right*) N-CODE learns a family of vector fields (red vs yellow vs blue), enabling the system to flexibly adjust the trajectory for every data point. (*Bottom left*) Trained NODE trajectories of initial values in a data set of concentric annuli, colored green and yellow. The NODE transformation is a homeomorphism on the data space and cannot separate the classes as a result. Colored points are the initial state of the dynamics, and black points are the final state. (*Bottom right*) Corresponding flows for N-CODE which easily separate the classes. Transiting from the inner to outer annulus effects a flow bifurcation which linearly separates the data.

The rest of the paper proceeds as follows. First, we will lay out the background for N-CODE and its technical formulation. Then, we will demonstrate its efficacy in two cases, one supervised and one unsupervised. In the supervised case, we show how N-CODE can classify data by learning to bifurcate its dynamics along class boundaries. Then, we show how the flows learned by N-CODE can be used as latent representations in an autoencoder, leading to state-of-the-art reconstruction on CIFAR10.

2 Background

Neural ODEs (NODEs) (Chen et al., 2018) consist of a differentiable nonlinear mapping $f : \mathcal{X} \times \mathbb{R} \times \Theta \mapsto \mathcal{X}$ parameterized by a vector $\boldsymbol{\theta} \in \Theta$ and defining a continuous vector field over a feature space $\mathcal{X}$. The corresponding equation of motion

$$\frac{d\boldsymbol{x}}{dt} = f(\boldsymbol{x}, \boldsymbol{\theta}, t) \tag{1}$$

gives rise to a *flow*, i.e a triple $(\mathcal{X}, \mathbb{R}, \Phi_{\boldsymbol{\theta}})$ with $\Phi_{\boldsymbol{\theta}} : \mathcal{X} \times \mathbb{R} \mapsto \mathcal{X}$ defined by:

$$\Phi_{\boldsymbol{\theta}}(\boldsymbol{x}_0, T) = \boldsymbol{x}_0 + \int_0^T f(\boldsymbol{x}(t), \boldsymbol{\theta}, t) dt \tag{2}$$

which relates an initial point $\boldsymbol{x}_0$ to an *orbit* of points $\{\boldsymbol{x}(t) = \Phi_\theta(\boldsymbol{x}_0, t), t \in \mathbb{R}\}$. Integrating $\Phi_{\boldsymbol{\theta}}$ over a specified time horizon results in a homeomorphism of the feature space parametrized by $\boldsymbol{\theta}$.

Several properties of such flows make them appealing for machine learning. For example, the ODEs that govern such such flows can be solved with off-the shelf solvers and they can potentially model irregularly sampled data. Moreover, such flows are reversible maps by construction whose inverse is just the system integrated backward in time, $\Phi_\theta(., t)^{-1} = \Phi_\theta(., -t)$. This property enables depth-constant memory cost of training thanks to the adjoint sensitivity method (Pontryagin, 1962) and the modeling of time-continuous generative normalizing flows algorithms (Grathwohl et al., 2019).

Interestingly, discretizing Eq. 2 coincides with the recursive formulation of a residual network (He et al., 2015) with a single residual operator $f_{\boldsymbol{\theta}}$:

$$\Phi_{ResNet}(\boldsymbol{x}_0, T) = \boldsymbol{x}_0 + \sum_{t=1}^{T} f_{\boldsymbol{\theta}}(\boldsymbol{x}_{t-1}) \tag{3}$$

In this sense, NODEs with a time-independent (autonomous) transition operator, f, are the infinitely deep limit of weight-tied residual networks.

3 N-CODE: LEARNING TO CONTROL DATA-DEPENDENT FLOWS

General formulation - The main idea of N-CODE is to consider the parameters, θ, in Eq. 1 as data-dependent control weights for the dynamical state, x. Model dynamics are then governed by a coupled system of equations on the extended state $\boldsymbol{z}(t) = (\boldsymbol{x}(t), \boldsymbol{\theta}(t))$. The initial value of the control weights, $\theta(0)$, is given by a mapping $\gamma : \mathcal{X} \to \Theta$. The full trajectory of control weights, $\theta(t)$, is then output by a controller, g, given by another differentiable nonlinear mapping $g : \Theta \times \mathbb{R} \times \mathcal{X} \mapsto \Theta$ with initial condition $\gamma(x_0)$. Given an initial point, $\boldsymbol{x}_0$, we can solve the initial value problem (IVP):

$$\begin{cases} \dfrac{d\boldsymbol{z}}{dt} = h(\boldsymbol{z}, t) \\ \boldsymbol{z}(t_0) = \boldsymbol{z}_0, \end{cases} = \begin{cases} \left(\dfrac{d\boldsymbol{x}}{dt}, \dfrac{d\boldsymbol{\theta}}{dt}\right) = (f(\boldsymbol{x}, \boldsymbol{\theta}, t), g(\boldsymbol{\theta}, \boldsymbol{x}, t)) \\ (\boldsymbol{x}(t_0), \boldsymbol{\theta}(t_0)) = (\boldsymbol{x}_0, \gamma(\boldsymbol{x}_0)) \end{cases} \tag{4}$$

where h corresponds to the augmented map $h = (f, g)$. We may think of g and γ as a collective controller of the dynamical system with equation of motion, f. We model g and γ as neural networks parametrized by $\mu \in \mathbb{R}^{n_\mu}$ and use gradient descent techniques where the gradient can be computed through the solving of an adjoint problem (Pontryagin, 1962) that we describe in the next section. Our goal here is to use the meta-parameterization via Θ to create richer dynamic behavior for a given f than directly optimizing its weights $\boldsymbol{\theta}$.

Well-posedness - If f and g are continuously differentiable with respect to $\boldsymbol{x}$ and $\boldsymbol{\theta}$ and continuous with respect to t, then, for all initial conditions $\boldsymbol{z}_0$, there exists a unique solution $\boldsymbol{z}$ for equation 4 thanks to the Cauchy-Lipschitz theorem. This result leads to the existence and uniqueness of the augmented flow $(\mathcal{X} \times \Theta, \mathbb{R}, \Phi_{\boldsymbol{\mu}})$ with $\Phi_{\boldsymbol{\mu}} : (\mathcal{X} \times \Theta) \times \mathbb{R} \mapsto \mathcal{X} \times \Theta$. Moreover, considering the restriction of such a flow on $\mathcal{X}$, we are now endowed with an universal approximator for at least the set of homeomorphisms on $\mathcal{X}$ given that this restriction constitutes a non-autonomous system (Zhang et al., 2019a). We discuss now how g and γ affect the evolution of the variable $\boldsymbol{x}(t)$, exhibiting two forms of control.

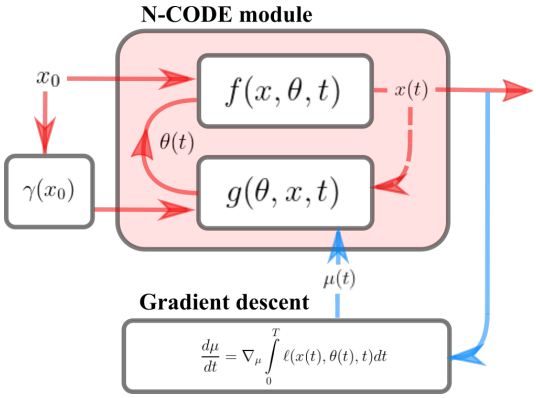

Figure 2: Diagram of a general N-CODE module: Given a initial state $\boldsymbol{x}_0$, the module consists of an augmented dynamical system that couples activity state $\boldsymbol{x}$ and weights $\boldsymbol{\theta}$ over time (red arrows). A mapping γ infers initial control weights $\boldsymbol{\theta}_0$ defining an inital flow (open-loop control). This flow can potentially evolve as $\boldsymbol{\theta}$ might be driven by a feedback signal from $\boldsymbol{x}$ (closed-loop, doted line). This meta-parameterization of f can be trained with gradient descent by solving an augmented adjoint sensitivity system (blue arrows)

3.1 OPEN AND CLOSED-LOOP CONTROLLERS

If the controller outputs control weights as a function of the current state, $\boldsymbol{x}(t)$, then we say that the it is a *closed-loop* controller. Otherwise, it is an *open-loop* controller.

Open-loop control - First, we consider the effect of using only the mapping γ in equation 4 as a controller. Here, γ maps the input space $\mathcal{X}$ to Θ so that f is conditioned on $\boldsymbol{x}_0$ but not necessarily on $x(t)$ for $t > 0$. In other words, each initial value $\boldsymbol{x}_0$ evolves according to its own learned flow $(\mathcal{X}, \mathbb{R}, \Phi_{\boldsymbol{\gamma}(\boldsymbol{x}_0)})$. This allows for particle trajectories to evolve more freely than within a single flow that must account for the whole data-distribution and resolves the problem of non-intersecting orbits (see Figure 3). At the distribution level, optimizing γ for a given cost function amounts

to find a push-forward mapping from the distribution on $\mathcal{X}$ to the distribution of flows $(\mathcal{X}, \mathbb{R}, \Phi)$ that minimizes this cost. Note that γ is independent of the way $\boldsymbol{\theta}$ parameterizes f, and thus can be combined with recent proposals that perform NODE parametrization in a discretized functional space, allowing for time-dependent parameters (Choromanski et al., 2020).

Closed-loop control - Defining a differentiable mapping g outputting the time-dependent control weights $\boldsymbol{\theta}(t)$ given the state of the variable $\boldsymbol{x}(t)$ yields a non-autonomous system on $\mathcal{X}$. This can be seen as a specific dimension augmentation technique where additional variables correspond to the parameters $\boldsymbol{\theta}$. However, contrary to appending extra dimensions which doesn't change the autonomous property of the system, here this augmentation results in a module describing a time-varying transformation $\Phi_{\boldsymbol{\theta(t)}}(\boldsymbol{x_0}, t)$. Additionally, the presence of closed-loop feedback might result in interesting dynamical regimes such as stability, chaos, etc. (HAL, 1969).

4 TRAINING

Loss function - We consider now the following generalized loss function that integrates a cost over some interval $[0, T]$:

$$l(\boldsymbol{z}) := \int_0^T \ell(\boldsymbol{z}(t), t)dt = \int_0^T \ell(\boldsymbol{x}(t), \boldsymbol{\theta}(t), t)dt \tag{5}$$

This loss formulation is more general than in (Chen et al., 2018) as one can imagine ℓ as any Lebesgue-measurable function. In particular, this encompasses discrete time point penalization or regularization of activations or weights over the whole trajectory rather than the final state $\boldsymbol{z}(T)$ of the system. The following result enables us to compute the gradients of equation 5 with respect to $\boldsymbol{\mu}$.

Theorem 1 - Augmented adjoint method: Given the (IVP) of equation 4 and for ℓ defined in equation 5, we have:

$$\frac{\partial l}{\partial \boldsymbol{\mu}} = \int_0^T \boldsymbol{a}(t)^T \frac{\partial h}{\partial \boldsymbol{\mu}} dt, \quad \text{such that } \boldsymbol{a} \text{ verifies } \begin{cases} \dfrac{d\boldsymbol{a}}{dt} = -\boldsymbol{a}^T . \dfrac{\partial h}{\partial \boldsymbol{z}} - \dfrac{\partial \ell}{\partial \boldsymbol{z}} \\ \boldsymbol{a}(T) = \mathbb{O}_{|\mathcal{X}|+|\Theta|} \end{cases} \tag{6}$$

where $\dfrac{\partial h}{\partial \boldsymbol{z}}$ is the Jacobian of h with respect to $\boldsymbol{z}$: $\quad \frac{\partial h}{\partial \boldsymbol{z}} = \begin{pmatrix} \frac{\partial f}{\partial \boldsymbol{x}} & \frac{\partial f}{\partial \boldsymbol{\theta}} \\ \frac{\partial g}{\partial \boldsymbol{x}} & \frac{\partial g}{\partial \boldsymbol{\theta}} \end{pmatrix}$

In practice, we computate the Jacobian for this augmented dynamics with open source automatic differentiation libraries using Pytorch (Paszke et al., 2019), mediating the definition of modules with specific backward methods to integrate them seamlessly in bigger architectures. We show an example of such modules in the appendix.

5 EXPERIMENTS

5.1 TOY PROBLEMS

Reflection - We introduce our approach with the 1-dimensional problem posed in (Dupont et al., 2019) consisting of learning the reflection map $\varphi(\boldsymbol{x}) = -\boldsymbol{x}$. It has been shown that the vanilla NODE module cannot learn such a function. However, both the open and closed loop formulations of N-CODE easily fit φ, as shown in Figure 3. The conditioning on data allows the N-CODE to learn a family of vector fields that circumvents the topological limitation of the single flow learned by NODEs. We may interpret the classification decision as resulting from a bifurcation in the model dynamics. In particular, as x goes from -1 to 1, we can observe the vector field given by $f(x)$ transitioning from a negative (blue) to a positive (orange) orientation.

Concentric annuli - We also evaluate our proposed models on the concentric annuli dataset proposed in (Lin & Jegelka, 2018; Dupont et al., 2019). Specifically, we define a class of equations of motion, f, acting on $\mathbb{R}^2$ in the form of a two-layer MLP with 16 hidden units. We train the models such that the final representations $\boldsymbol{x}(T)$ must linearly separate the classes on the concentric

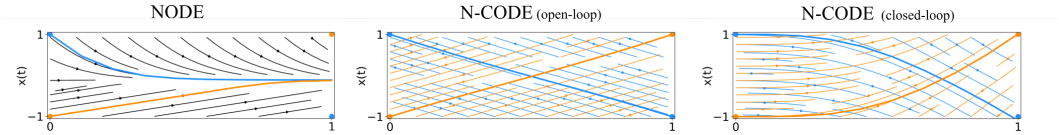

Figure 3: Trajectories over time for three types of continuous-time neural networks learning the 1-dimensional reflection map $\varphi(\boldsymbol{x}) = -\boldsymbol{x}$. **Left:** (*NODE*) Irrespective of the form of f, a NODE module cannot learn such a function as trajectories cannot intersect (Dupont et al., 2019). **Middle:** (*Open-loop N-CODE*). The model is able to learn a family of vector fields by controlling a single parameter of f conditioned on $\boldsymbol{x}_0$. **Right:** (*Closed-loop N-CODE*) With a fixed initialization $\boldsymbol{\theta}_0$, the controller model still learns a deformation of the vector field to learn φ. Note that the vector field is time-varying, contrary to the two others versions.

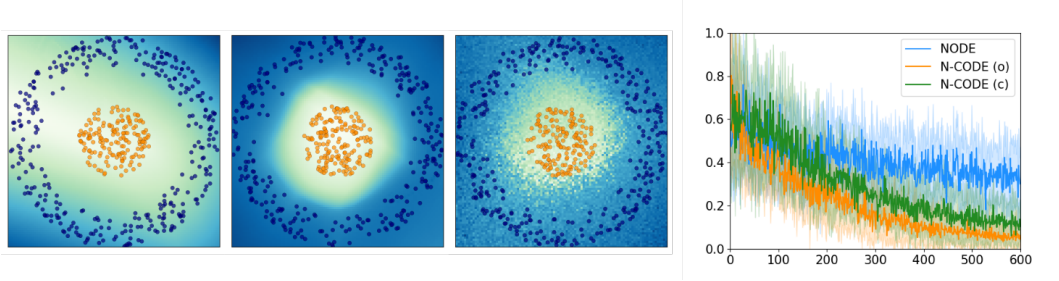

Figure 4: The *concentric annuli* problem: **Left** Decision boundaries for NODE (*First*) and N-CODE open-loop ((*Second*) and closed-loop (*First*) models. **Right** Training curve for the three models.

classes dataset. We qualitatively evaluated the learned flows complexity and their generalization capabilities by inspecting decision boundaries in Fig. 4. As previously discussed, NODE transformations are homeomorphisms and preserve the topology of the space, notably the linking number of data manifolds, which might lead to unstable and complex flows for intricated data manifold. In comparison, our proposed method is not affected by this problem, as every trajectory belongs to its own specific flow, thus breaking the topological constraint of these transformations and allowing for simpler paths that ease model convergence (see Fig 4). Again, we may think of the classification decisions as being mediated by a bifurcation parametrized by the data space. Transiting across the true classification boundary switches the vector field from one pushing points to one side of the classification boundary to the other. Moreover, the construction of γ as a neural network guarantees a smooth variation of the flow with respect to the initial condition, which can potentially provide interesting regularization properties on the family learned by the model.

5.2 Supervised image classification

Model	MNIST	CIFAR-10
NODE	96.4%	53.7%
ANODE	98.2%	60.6%
NCODE (open)	**99.2%**	**66.7%**

Figure 5: Test accuracy on MNIST and CIFAR10 for different Neural ODEs.

We now compare our open-loop control method to earlier continuous-time neural network models, like NODEs, on a image classification task. We show that our method not only outperforms NODEs and Augmented NODEs but also trains quicker, as the model converges earlier than NODEs variants. We perform this experiment on the MNIST and CIFAR-10 image datasets. Our model consists of the dynamical system where transformation f is expressed as a single convolutional block $(1 \times 1 \mapsto 3 \times 3 \mapsto 1 \times 1)$ with 50 channels. The control weights $\boldsymbol{\theta} \in \mathbb{R}^K$ is a real-valued vector of K concatenated parameters instantiating the convolution kernels. The 10-class prediction vector is obtained as a linear transformation of the final state $\boldsymbol{x}(T)$ of the dynamics with a sigmoid activation. We aimed at simplicity by defining the mapping γ as a two-layer perceptron with 10 hidden units. We did not perform any fine-tuning, although we did augment the state dimension, akin to ANODE, by increasing the channel dimension by 10, which helped stabilize training. Results are shown in Fig. 5.

5.3 Unsupervised learning : Image autoencoding with controlled flow

We now use our control approach in an autoencoding model for low-dimensional representation learning (Fig. 6). Our idea is to invoke a standard class of ordinary differential equations to "encode"

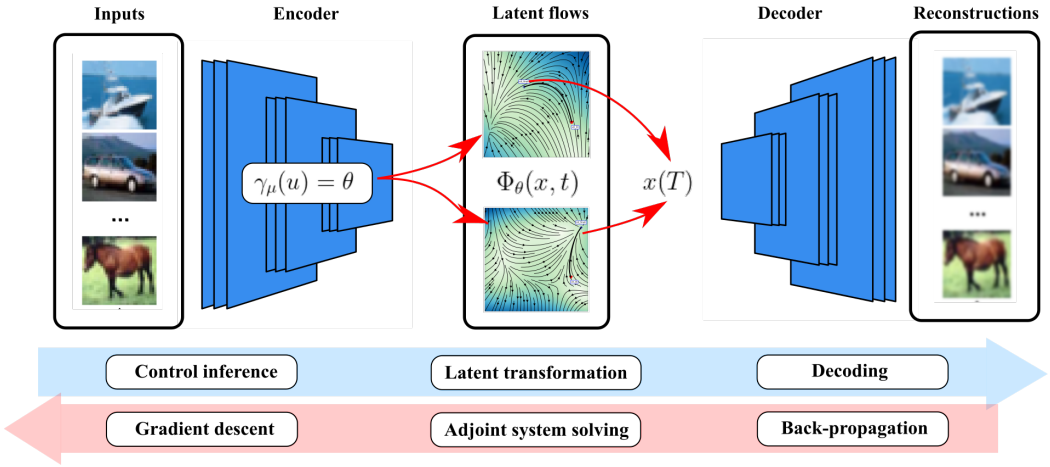

Figure 6: Diagram of AutoN-CODE: Very similar to a vanilla auto-encoder, the model uses two sets of convolution filters to map the data distribution to a low dimensional representation space. However, rather than mapping to a to-be-decoded representation, the encoder infers a control parameterizing a linear system. This data-depndent system is subsequently solved and used as a low-dimensional representation for the decoder reconstruction. The whole architecture remains entirely differentiable and gradients can be estimated by solving and adjoint state system.

the model's latent representations taken as the system state at a fixed point in time.

$$\begin{aligned} \gamma_{\boldsymbol{\mu}} &: \mathcal{U} \mapsto \Theta \subset \mathbb{R}^n \\ \boldsymbol{u} &\mapsto \gamma_{\boldsymbol{\mu}}(\boldsymbol{u}) = \boldsymbol{\theta} = (\boldsymbol{\theta}_{\boldsymbol{i},\boldsymbol{j}})_{1 \leq i,j \leq n} \end{aligned} \tag{7}$$

The latent code is subsequently interpreted as the solution of the autonomous differential system presented in equation 8 for a specified time T and given the initial condition $(\boldsymbol{x}_0, \gamma_{\boldsymbol{\mu}}(u))$ with $\boldsymbol{x}_0$ a constant initial vector.

$$\begin{cases} \left(\dfrac{d\boldsymbol{x}}{dt}, \dfrac{d\boldsymbol{\theta}}{dt}\right) = (f(\boldsymbol{x}, \boldsymbol{\theta}, t), 0) \\ (\boldsymbol{x}(t_0), \boldsymbol{\theta}(t_0)) = (\boldsymbol{x}_0, \gamma_{\boldsymbol{\mu}}(\boldsymbol{u})) \end{cases} = \begin{cases} \left(\dfrac{d\boldsymbol{x}}{dt}, \boldsymbol{\theta}(t)\right) = (\langle \boldsymbol{\theta}_{\boldsymbol{i}}, \boldsymbol{x}(t) \rangle, \gamma_{\boldsymbol{\mu}}(u)) \\ \boldsymbol{x}(t_0) = (\boldsymbol{x}_0) \end{cases} \tag{8}$$

This model can be interpreted as a hybrid version of autoencoders and generative normalizing flows, where the latent generative flow is data-dependant and parametrized by the encoder output. We empirically found that a simple linear homogeneous differential system was already more expressive than a linear layer to shape the latent representation. However, since the output dimension of the encoder grows quadratically with the desired latent code dimension of the bottleneck, we adopted a sparse prediction strategy, where as few as two elements of each row of $(\boldsymbol{\theta}_{i,j})_{1 \leq i,j \leq n}$ are non-zero, making our model match exactly the number of parameters of a VAE with same architecture. We believe that the increase in representational power over a vanilla autoencoder, as shown by FID improvements, despite these simplistic assumptions, shows the potential of this approach for representation learning (Fig. 7). For the encoder and decoders, we slightly simplify the convolution architecture tested in (Ghosh et al., 2019; Tolstikhin et al., 2017) and apply batch normalization to all layers (see Appendix). We make no additional

Dataset	FID	
Model	CIFAR-10	CelebA (64x64)
Variational Bayes AE† (Kingma & Welling, 2013)	105.45	68.07
Auto-encoder† (100 components)	73.24	63.11
PixelCNN (van den Oord et al., 2016)	65.93	-
Auto-encoder† (1000 components)	55.55	55.60
DCGAN (Radford et al., 2015)	37.11	-
EBM (10 hist. ensemble) Du & Mordatch (2019)	38.2	-
MoLM (Ravuri et al., 2018)	33.8	-
NCSN (Song & Ermon, 2019)	25.32	-
SNGAN (Miyato et al., 2018)	21.7	-
AutoGAN (Gong et al., 2019)	12.42	-
AutoNCODE (100 components)	34.90	55.27
AutoNCODE (1000 components)	24.19	52.47

Figure 7: Frechet Inception Distance (FID) for several recent architectures. (lower is better) Models that are evaluated using authors code are noted with †. CelebA results are given for our implementations only as data processing is inconsistent across models.

change to the decoder. We train the parameters of the encoder and decoder module for minimizing the mean-squared error (MSE) on CIFAR-10 (Krizhevsky, 2012) and CelebA (Liu et al., 2015) or alternatively the Kullback-Leibler divergence between the data distribution and the ouput of the decoder for MNIST (Y. LeCun & Haffner., 1998) (formulas in Appendix.) Gradient descent is performed for 50 epochs with the Adam (Kingma & Ba, 2014) optimizer with learning rate $\lambda = 1e^{-3}$ reduced by half every time the loss plateaus. All experiments are run on a single GPU GeForce Titan X with 12 GB of memory.

5.4 IMAGE GENERATION

In order to endow our deterministic model with the ability to generate image samples, we perform, similarly to (Ghosh et al., 2019), a post hoc density estimation of the distribution of the latent trajectories final state. Namely, we employ a gaussian mixture model (GMM) fitted with Expectation-Maximization, and explore the effect of increasing the number of components in the mixture. We report in Fig. 7, contrary to (Ghosh et al., 2019), a substantial decrease of the Frechet Inception Distance (FID, (Heusel et al., 2017)) of our sampled images to the image test set with increasing number of components in the mixture (see Fig. 8 for sampled examples and Fig. 9 for reconstructions along controlled orbits). These results suggest, considering the identical architecture of the encoder and decoder to a vanilla model, that the optimal encoding control formulation produced a structural change in the latent manifold organization. We further show that our approach compares favorably over recent generative techniques (see Fig. 10). Finally, we note that this model offers several original possibilities to explore, such as sampling from the control coefficients, or adding a regularization on the dynamic evolution as in (Finlay et al., 2020), which we leave for future work.

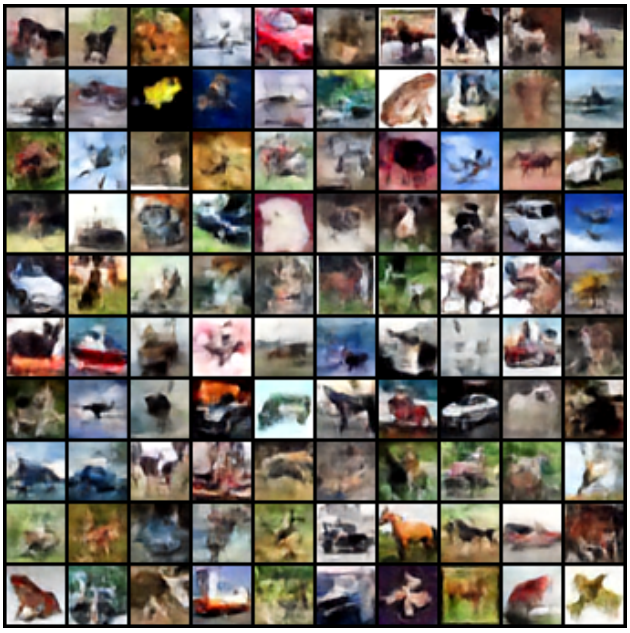

Figure 8: Random samples generated with **AutoN-CODE** trained on CIFAR-10 from a Gaussian Mixture Model with 5000 components.

Figure 9: Left to right: Temporal evolutions of an image reconstruction along controlled *orbits* of **AutoN-CODE** for representative CIFAR-10 and CelabA test images. Last column: ground truth image.

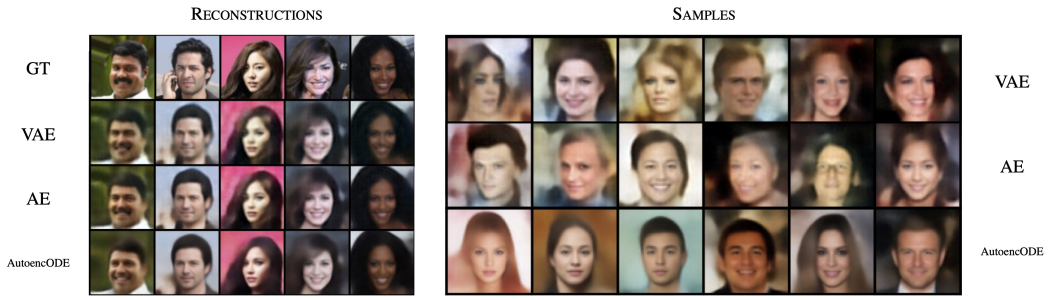

Figure 10: **Left** Reconstructions of random test images from CelebA for different autoencoding models. **Right**: Random samples from the latent space. Deterministic models are fitted with a 100 components GMM.

6 RELATED WORK

Neural ODEs - This work builds on the recent development of Neural ODEs, which have shown great promise for building generative normalizing flows (Grathwohl et al., 2018) such as for modeling molecular interactions (Köhler et al., 2019) or in mean field games (Ruthotto et al., 2019). More recent work has focused on the topological properties of these models (Dupont et al., 2019; Zhang et al., 2020), introducing non-static parametrization (Zhang et al., 2019b; Massaroli et al., 2020b; Choromanski et al., 2020), developing novel regularization methods using optimal transport or stochastic perturbations (Finlay et al., 2020; Oganesyan et al., 2020), adapting them to stochastic differential equations (Tzen & Raginsky, 2019).

Optimal Control - Several works have interpreted deep learning optimization as an optimal control problem (Li et al., 2017; Benning et al., 2019; Liu & Theodorou, 2019; Zhang et al., 2019a; Seidman et al., 2020) providing strong error control and yielding convergence results for alternative algorithms to standard gradient-based learning methods. Of particular interest is the *mean-field optimal control* formulation of (E et al., 2018), which notes the dependence of an unique control variable governing the network dynamics on the whole data population. Finally, the adjoint method at the heart of NODEs training arise from the well known maximum principle of Pontryagin (1962) from control theory.

Hypernetworks - Finally, this work is also related to network architectures with adaptive weights such as Hypernetworks (Ha et al., 2016), dynamic filters (Brabandere et al., 2016) and spatial transformers (Jaderberg et al., 2015). Though not explicitly formulated as optimal neural control, these approaches effectively implement a form of modulation of neural network activity as a function of input data and activation state, resulting in compact and expressive models. These methods demonstrate, in our opinion, the significant untapped potential value in developing dynamically controlled modules for deep learning.

7 CONCLUSION

In this work, we have presented an original modulation system for continuous-time neural feature transformations that theoretically relates to optimal control. We have shown that it is possible to dynamically shape the trajectories of the transformation module applied to the data, by augmenting the network with a trained control inference mechanism, and further demonstrated that this can be applied in the context of unsupervised image representation learning. In future work, we would like to investigate the robustness and generalization properties of such controlled models as well as their similarities with fast-synaptic modulation systems observed in neuroscience, and test this on natural applications such as recurrent neural networks and robotics. An additional avenue for further research is the connection between our system and the theory of bifurcations in dynamical systems and neuroscience.

REFERENCES

Dynamical systems and stability. *Journal of Mathematical Analysis and Applications*, 26(1):39 – 59, 1969. ISSN 0022-247X. doi: https://doi.org/10.1016/0022-247X(69)90175-9.

Martin Benning, Elena Celledoni, Matthias J. Ehrhardt, Brynjulf Owren, and Carola-Bibiane Schönlieb. Deep learning as optimal control problems: models and numerical methods, 2019.

Bert De Brabandere, Xu Jia, Tinne Tuytelaars, and Luc Van Gool. Dynamic filter networks, 2016.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2018.

Krzysztof Choromanski, Jared Quincy Davis, Valerii Likhosherstov, Xingyou Song, Jean-Jacques Slotine, Jacob Varley, Honglak Lee, Adrian Weller, and Vikas Sindhwani. An ode to an ode, 2020.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. ISSN 00359246. URL http://www.jstor.org/stable/2984875.

Yilun Du and Igor Mordatch. Implicit generation and generalization in energy-based models, 2019.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes, 2019.

Weinan E, Jiequn Han, and Qianxiao Li. A mean-field optimal control formulation of deep learning, 2018.

Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ode: the world of jacobian and kinetic regularization, 2020.

Partha Ghosh, Mehdi S. M. Sajjadi, Antonio Vergari, Michael Black, and Bernhard Schölkopf. From variational to deterministic autoencoders, 2019.

Xinyu Gong, Shiyu Chang, Yifan Jiang, and Zhangyang Wang. Autogan: Neural architecture search for generative adversarial networks, 2019.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models, 2018.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, and David Duvenaud. Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJxgknCcK7.

David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2017.

Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks, 2015.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.

Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

Jonas Köhler, Leon Klein, and Frank Noé. Equivariant flows: sampling configurations for multi-body systems with symmetric energies, 2019.

Qianxiao Li, Long Chen, Cheng Tai, and Weinan E. Maximum principle based algorithms for deep learning, 2017.

Hongzhou Lin and Stefanie Jegelka. Resnet with one-neuron hidden layers is a universal approximator, 2018.

Guan-Horng Liu and Evangelos A. Theodorou. Deep learning theory review: An optimal control and dynamical systems perspective, 2019.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Stefano Massaroli, Michael Poli, Michelangelo Bin, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Stable neural flows, 2020a.

Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes, 2020b.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks, 2018.

Viktor Oganesyan, Alexandra Volokhova, and Dmitry Vetrov. Stochasticity in neural odes: An empirical study, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

V G ; Gamkrelidze R V ; Mishchenko E F Pontryagin, Lev Semenovich ; Boltyanskii. *The mathematical theory of optimal processes*. 1962.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

Suman Ravuri, Shakir Mohamed, Mihaela Rosca, and Oriol Vinyals. Learning implicit generative models with the method of learned moments, 2018.

Lars Ruthotto, Stanley Osher, Wuchen Li, Levon Nurbekyan, and Samy Wu Fung. A machine learning framework for solving high-dimensional mean field game and mean field control problems, 2019.

Lars Ruthotto, Stanley J. Osher, Wuchen Li, Levon Nurbekyan, and Samy Wu Fung. A machine learning framework for solving high-dimensional mean field game and mean field control problems. *Proceedings of the National Academy of Sciences*, 117(17):9183–9193, 2020. ISSN 0027-8424. doi: 10.1073/pnas.1922204117. URL https://www.pnas.org/content/117/17/9183.

Jacob H. Seidman, Mahyar Fazlyab, Victor M. Preciado, and George J. Pappas. Robust deep learning as optimal control: Insights and convergence guarantees. volume 120 of *Proceedings of Machine Learning Research*, pp. 884–893, The Cloud, 10–11 Jun 2020. PMLR.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution, 2019.

Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein auto-encoders, 2017.

Belinda Tzen and Maxim Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit, 2019.

Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders, 2016.

Y. Bengio Y. LeCun, L. Bottou and P. Haffner. "gradient-based learning applied to document recognition.". *Proceedings of the IEEE*, 1998.

Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 227–238. Curran Associates, Inc., 2019a.

Han Zhang, Xi Gao, Jacob Unterman, and Tom Arodz. Approximation capabilities of neural odes and invertible residual networks, 2020.

Tianjun Zhang, Zhewei Yao, Amir Gholami, Joseph E Gonzalez, Kurt Keutzer, Michael W Mahoney, and George Biros. Anodev2: A coupled neural ode framework. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 5151–5161. Curran Associates, Inc., 2019b.

A APPENDIX

A.1 PROOF OF THEOREM 1

Proof. The proof is inspired from Massaroli et al. (2020a). We place our analysis in the Banach space $\mathbb{R}^n$. Since ℓ and h are continuously differentiable, let us form the *Lagrangian* functional $\mathcal{L}$ where $\boldsymbol{a}$ is an element of the dual space of $\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\theta} \end{bmatrix}$:

$$\mathcal{L}(\boldsymbol{z}, \boldsymbol{a}, t) := \int_0^T [\ell - \langle \boldsymbol{a}, \frac{\partial \boldsymbol{z}}{\partial t} - h \rangle] dt \tag{9}$$

The constraint $\langle \boldsymbol{a}(t), \frac{\partial z}{\partial t} - h(\boldsymbol{z}, t) \rangle$ is always *active* on the admissible set by construction of $\boldsymbol{z}$ in equation 4, such that we have $\forall \boldsymbol{z}, \boldsymbol{a}, \quad \frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}}(\boldsymbol{z}, \boldsymbol{a}, t) = \frac{\partial l}{\partial \boldsymbol{\mu}}(\boldsymbol{z}, \boldsymbol{a}, t)$. Moreover, integrating the left part of the integral in equation 9 gives:

$$\int_0^T \langle \boldsymbol{a}, \frac{\partial \boldsymbol{z}}{\partial t} \rangle dt = \langle \boldsymbol{a}, \boldsymbol{z} \rangle \big|_0^T - \int_0^T \langle \frac{\partial \boldsymbol{a}}{\partial t}, \boldsymbol{z} \rangle dt \tag{10}$$

Now, given that $\ell - \langle \frac{d\boldsymbol{a}}{dt}, \boldsymbol{z} \rangle$ is derivable in μ for any $t \in [0, T]$, Leibniz integral rule allows to write

$$\frac{\partial l}{\partial \boldsymbol{\mu}} = \int_0^T \frac{\partial}{\partial \boldsymbol{\mu}} [\ell + \langle \frac{d\boldsymbol{a}}{dt}, \boldsymbol{z} \rangle + \langle \boldsymbol{a}, h \rangle] dt + \langle \boldsymbol{a}(T), \frac{\partial \boldsymbol{z}(T)}{\partial \boldsymbol{\mu}} \rangle - \langle \boldsymbol{a}(0), \cancel{\frac{\partial \boldsymbol{z}(0)}{\partial \boldsymbol{\mu}}} \rangle \tag{11}$$

$$\frac{\partial l}{\partial \boldsymbol{\mu}} = \int_0^T [\frac{\partial \ell}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\mu}} + \langle \frac{d\boldsymbol{a}}{dt}, \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\mu}} \rangle + \langle \boldsymbol{a}, \frac{\partial h}{\partial \boldsymbol{\mu}} \rangle] dt + \langle \boldsymbol{a}(T), \frac{\partial \boldsymbol{z}(T)}{\partial \boldsymbol{\mu}} \rangle \tag{12}$$

$$\frac{\partial l}{\partial \boldsymbol{\mu}} = \int_0^T [\frac{\partial \ell}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\mu}} + \langle \frac{d\boldsymbol{a}}{dt}, \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\mu}} \rangle + \langle \boldsymbol{a}, \frac{\partial h}{\partial \boldsymbol{\mu}} + \cancel{\frac{\partial h}{\partial t} \frac{\partial t}{\partial \boldsymbol{\mu}}} + \frac{\partial h}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\mu}}] dt + \langle \boldsymbol{a}(T), \frac{\partial \boldsymbol{z}(T)}{\partial \boldsymbol{\mu}} \rangle \tag{13}$$

The last equation can be reordered as:

$$\frac{\partial l}{\partial \boldsymbol{\mu}} = \int_0^T \langle \frac{d\boldsymbol{a}}{dt} - \boldsymbol{a} \frac{\partial h}{\partial \boldsymbol{z}} - \frac{\partial \ell}{\partial \boldsymbol{z}}, \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\mu}} \rangle dt + \int_0^T \langle \boldsymbol{a}, \frac{\partial h}{\partial \boldsymbol{\mu}} \rangle dt + \langle \boldsymbol{a}(T), \frac{\partial \boldsymbol{z}(T)}{\partial \boldsymbol{\mu}} \rangle \tag{14}$$

Posing that $\boldsymbol{a}(T) = \mathbb{O}_{|\mathcal{X}|+|\Theta|}$ and $\langle \frac{d\boldsymbol{a}}{dt} - \boldsymbol{a} \frac{\partial h}{\partial \boldsymbol{z}} - \frac{\partial \ell}{\partial \boldsymbol{z}}, \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\mu}} \rangle = 0$, the result follows. □

Remark 1: This result allows to compute gradient estimate assuming that $\boldsymbol{\mu}$ directly parametrizes h such that the gradient $\frac{\partial h}{\partial \boldsymbol{\mu}}$ are straightforward to compute. However, in the case of optimizing the initial mapping γ that infer the variable $\boldsymbol{\theta}$, this result can be combined with usual chain rule to estimate the gradient:

$$\frac{\partial l}{\partial \boldsymbol{\mu}_\gamma} = \frac{\partial \theta}{\partial \boldsymbol{\gamma}} \frac{\partial l}{\partial \boldsymbol{\theta}} \tag{15}$$

A.2 IMPLEMENTATION

Generic N-CODE module - We provide here a commented generic PyTorch implementation for the N-CODE module in the open-loop setting.

```
class NCODE(torch.autograd.Function):
    @staticmethod
    def forward(ctx, x_t, theta_t, t, model, control, model_names, parameters_size, device, *adjoint_params):
        """
         - ctx : context object that can be used to stash information for backward computation.
         - z_t : the tuple of tensors representing the state and controls of the system
         - model_names, parameters_size : arguments to initiate the model update

        """
        ### 1) Instantiate f_model with control parametrization
        state_dict = OrderedDict(zip(model_names,\theta_t))
        model.load_state_dict(state_dict)
        d_x = model(y_t)

        ### 2) Compute control update as per equation dw/dt =
        d_u = control(y_t)

        ### 3) return dz/dt = [dx/dt, dtheta_1/dt, ..., dtheta_N/dt]
        dz = (d_y, *d_u)

        ### 4) Save models and tensors
        ctx.save_for_backward(h, d_x, x_t, *\theta_t)
        ctx.model_names = model_names
        ctx.parameters_size = parameters_size
        ctx.device = device
        ctx.intermediate = d_theta , theta_t
        ctx.theta_t = theta_t
        ctx.model = model
        ctx.control = control

        return dz

    @staticmethod
    def backward(ctx, grad_out, a=None, b=None, c=None, d=None):
        """
         - grad_out : Tensor (or tuple of tensors) containing the gradient of the loss
        function with respect to y_t and u_t

        """

        with torch.no_grad():
            grad_x_out = grad_out
            h, d_x, x_t, *theta_t = ctx.saved_tensors
            model = ctx.model
            control = ctx.control
            model_names = ctx.model_names
            parameters_size = ctx.parameters_size

            state_dict = OrderedDict(zip(model_names, theta_t))
            model.load_state_dict(state_dict)
            grad_weigths = [torch.zeros_like(param) for param in tuple(ctx.control.parameters())]

```

```
            with torch.enable_grad():
                model_eval = model(x_t)
                control_eval = [layer(control[1](control[0](x_t))).reshape(parameters_size[i]) for i, layer in enumerate(control[2:])]
                control_eval_param = [nn.Parameter(weight,requires_grad=True) for weight in control_eval]

                #compute jacobian for f_model

                d_mod_dx, *d_mod_dtheta = torch.autograd.grad(model_eval, (x_t,) + tuple(ctx.model.parameters()), grad_x_out, create_graph=False)

                for i, grad_u in enumerate(d_mod_du):
                    d_con_dmu = torch.autograd.grad(control_eval[i], tuple(ctx.control.parameters()), grad_theta, create_graph=False, retain_graph=True, allow_unused=True)
                    grad_weigths = [ grad_weigths[i] + d_con_dmu[i] if d_con_dmu[i] is not None else grad_weigths[i] for i in range(len(d_con_dmu))]
                d_con_du = None


        grad_y_in = d_mod_dy
        adj_params = grad_weigths

        return (grad_y_in, None, None, None, None, None, None, None, *adj_params)
```

Listing 1: Generic N-CODE module forward and backward methods

A.3 Experimental results

Resources - Our experiments were run on a 12GB NVIDIA® Titan Xp GPUs cluster equipped with CUDA 10.1 driver. Neural ODEs were trained using torchdiffeq (Chen et al., 2018) PyTorch package.

A.4 AutoN-CODE architectures

	MNIST	**CIFAR-10**	**CelebA**
	$\mathbf{x} \in \mathbb{R}^{3\times28\times28}$	$\mathbf{x} \in \mathbb{R}^{3\times32\times32}$	$\mathbf{x} \in \mathbb{R}^{3\times64\times64}$
Encoder:	Flatten $\text{FC}_{400} \mapsto$ Relu $\text{FC}_{25} \mapsto$ Norm	$\text{Conv}_{256} \mapsto$ BN $\mapsto$ Relu $\text{Conv}_{512} \mapsto$ BN $\mapsto$ Relu $\text{Conv}_{1024} \mapsto$ BN $\mapsto$ Relu $\text{FC}_{128*2} \mapsto$ Norm	$\text{Conv}_{256} \mapsto$ BN $\mapsto$ Relu $\text{Conv}_{512} \mapsto$ BN $\mapsto$ Relu $\text{Conv}_{1024} \mapsto$ BN $\mapsto$ Relu $\text{FC}_{64*10} \mapsto$ Norm
Latent dynamics:	ODE[0,10]	ODE[0,10]	ODE[0,10]
Decoder:	$\text{FC}_{400} \mapsto$ Relu $\text{FC}_{784} \mapsto$ Sigmoid	$\text{FC}_{1024*8*8} \mapsto$ Norm $\text{ConvT}_{512} \mapsto$ BN $\mapsto$ Relu $\text{ConvT}_{256} \mapsto$ BN $\mapsto$ Relu ConvT_{3}	$\text{FC}_{1024*8*8} \mapsto$ Norm $\text{ConvT}_{512} \mapsto$ BN $\mapsto$ Relu $\text{ConvT}_{256} \mapsto$ BN $\mapsto$ Relu ConvT_{3}

Table A.1: Model architectures for the different data sets tested. FC_n and Conv_n represents respectively fully connected and convolutional layer with n output/filters. We apply a component-wise normalisation of the controls components which proved crucial for good performance of the model. The dynamic is ran on the time segment [0,10] which empirically yield good results.

Adapting the models adopted by Tolstikhin et al. (2017) and Ghosh et al. (2019), we use a latent space dimension of 25 for MNIST, 128 for CIFAR-10 and 64 for CelebA. All convolutions and transposed convolutions have a filter size of 4×4 for MNIST and CIFAR-10 and 5×5 for CELEBA.

They all have a stride of size 2 except for the last convolutional layer in the decoder. We use Relu non-linear activation and batch normalisation at the end of every convolution filter. Official train and test splits are used for the three datasets. For training, we use a mini-batch size of 64 in MNIST and CIFAR and 16 for CelebA in AutoN-CODE. (64 for control models.) All models are trained for a maximum of 50 epochs on MNIST and CIFAR and 40 epochs on CelebA.

A.5 VISUALIZATION OF LATENT CODE DYNAMICAL EVOLUTION

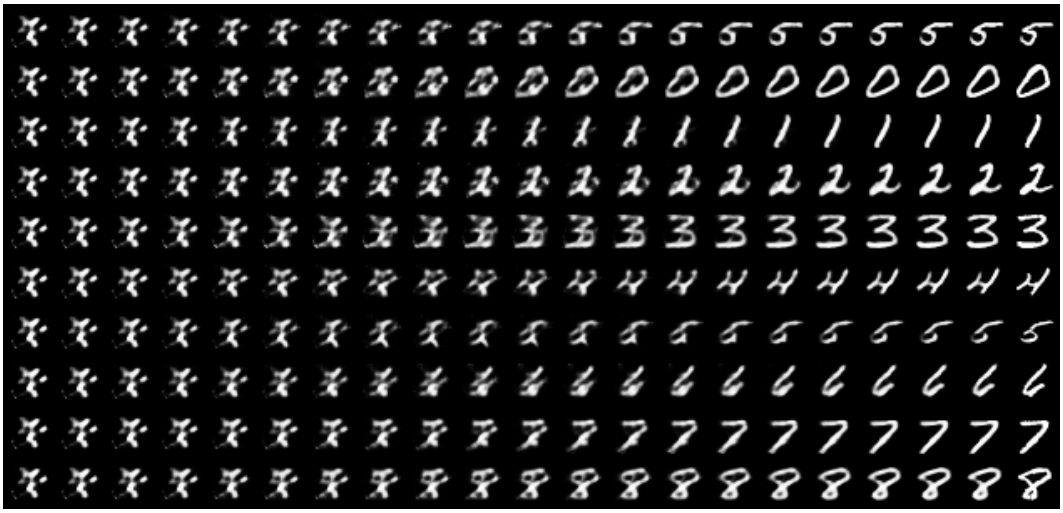

Figure A.1: Reconstructions of the image along the controlled *orbits* of AutoN-CODE for MNIST. The temporal dimension reads from left to right. Last column: ground truth image

Figure A.2: Reconstructions of the image along the controlled *orbits* of AutoN-CODE for CIFAR-10. The temporal dimension reads from left to right. Last column: ground truth image.

A.6 EXPLORING SAMPLING DISTRIBUTIONS

For random samples evaluation, we train the VAE with a $\mathcal{N}(0, I)$ prior. For the deterministic models, samples are drawn from a mixture of multivariate gaussian distributions fitted using the testing set embeddings. The distribution is obtained through Expectation-Maximization Dempster et al. (1977) with one single k-means initialization, tolerance $1e^{-3}$ and run for at most 100 iterations.

Figure A.3: Reconstructions of the image along the controlled *orbits* of AutoN-CODE for CelebA. The temporal dimension reads from left to right. Last column: ground truth image

We compute the FID using 10k generated samples evaluated against the whole test set for all FID evaluations, using the standard 2048-dimensional final feature vector of an Inception V3 following Heusel et al. (2017) implementation.

Further experiments showed that 10K components (matching the number of data points in the test sets) logically overfits on the data distribution. Generated images display marginal changes compared to test images. However, 1000 components does not, showing that our AutoN-CODE sampling strategy describe a trade-off between sample quality and generalization of images. We alternatively tested non-parametric kernel density estimation with varying kernel variance to replace our initial sampling strategy. We report similar result to gaussian mixture with an overall lower FID of AutoN-CODE for small variance kernels. As the fiting distribution becomes very rough ($\sigma \sim 5$), the generated image quality is highly deteriorating.(see Figure S6).

Figure A.4: Nearest neighbors search of random samples from the gaussian mixture with (**Left**) 1000 components (**Left**) and (**Right**) 10K components in the testing set of CIFAR-10.

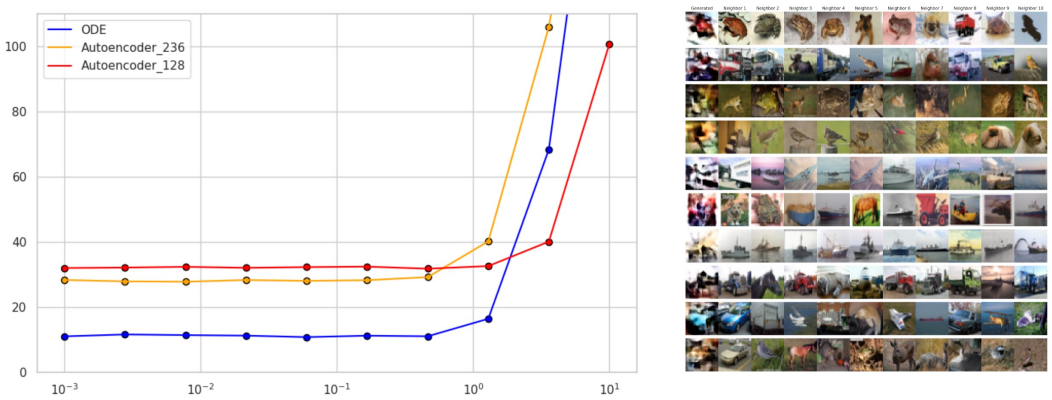

Figure A.5: (**Left**) Evolution of FID between test set and generated samples as a function of the gaussian kernel variance used to perform kernel estimation of the the latent code distribution. (**Right**) Nearest neighbors search of random samples from the latent distribution fitted with a gaussian kernel estimation with variance $\sigma = 2$ in the testing set.

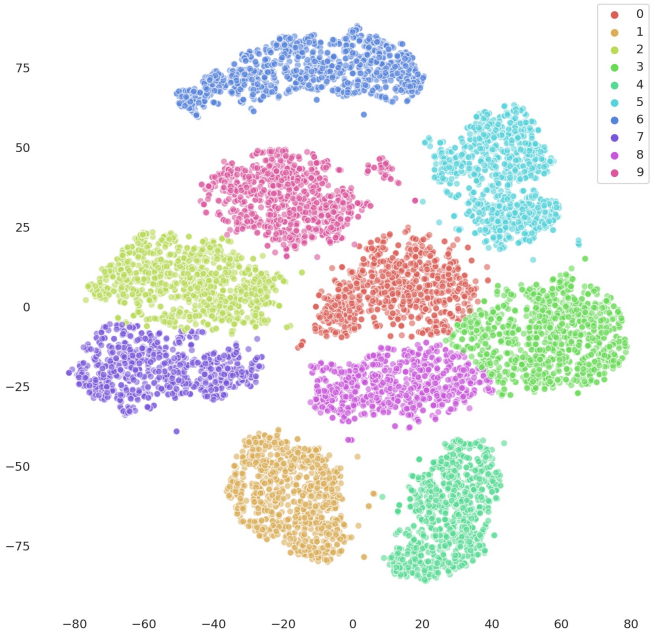

Figure A.6: t-SNE embeddings of the latent code at $t = t_1$ for MNIST test set colored with a 10 component gaussian mixture model.

A.7 Latent code interpolation

Figure A.7: Interpolation: We further explore the latent space of AutoN-CODE by interpolating between the latent vectors of the **CelebA** test set. (**Upper panel**) Linear interpolation between random samples reconstructed with AutoN-CODE. (**Middle panel**) Interpolation comparison between AutoN-CODE and a vanilla auto-encoder for a single pair of vectors. (**Lower panel**) 2d interpolation with three anchor reconstructions corresponding to the three corners of the square (A:up-left,B:up-right and C:down-left. Left square corresponds to AutoN-CODE reconstructions and right to a vanilla auto-encoder.