In this paper, we propose a method, which allows us to alleviate or completely avoid the notorious problem of numerical instability and stiffness of the adjoint method for training neural ODE. On the backward pass, we propose to use the machinery of smooth function interpolation to restore the trajectory obtained during the forward integration. We show the viability of our approach, both in theory and practice.

Keywords: neural ODEs, adjoint method, barycentric Lagrange interpolation, density estimation, FFJORD

1 Introduction

In this study, we propose a novel method to train neural ordinary differential equations (neural ODEs) (Chen et al., 2018). This method performs stable and memory-efficient backpropagation through the solution of initial value problems (IVP). Throughout the study, we use the term neural ODEs for all neural networks with so-called ODE blocks. Each ODE block is a continuous analog of a residual neural network (He et al., 2016). Indeed, such networks can be considered as Euler discretization of ordinary differential equations. Neural ODEs were motivated by the relationship between residual neural networks and dynamical systems (Lu et al., 2017, Chang et al., 2017, Ruthotto and Haber, 2018). Such models have already been successfully applied to various machine learning problems, including classification, generative modeling, and time series prediction (Chen et al., 2018, Grathwohl et al., 2018, Rubanova et al., 2019).

In general, the core of ODE block is the initial value problem (IVP)

\[
\begin{align*}
    \frac{dz}{dt} &= f(z(t), t, \theta), & t \in [t_0, t_1] \\
    z(t_0) &= z_0,
\end{align*}
\]

which depends on parameter \( \theta \) and considers the input features \( z_0 \) as the initial condition. The forward pass in neural ODEs is performed by standard numerical ODE solvers. In most cases, adaptive solvers (Dormand and Prince, 1980)
are preferable, because they adjust both the number and the size of steps on every iteration automatically. During the backward pass in neural ODEs, the loss function $L$, which depends on the solution of the IVP, should be differentiated with respect to parameter $\theta$. A direct application of backpropagation to ODE solvers leads to memory issues, because in this case, for every intermediate time step $\tau_k$, the output $z(\tau_k)$ must be stored as a part of computational graph. Moreover, using adaptive methods becomes risky, since we are not able to explicitly control the number of steps and, as a result, the amount of allocated memory (Eriksson et al., 2004).

However, the adjoint method (Pontryagin et al., 1961; Giles and Pierce, 2000; Marchuk, 2013) performs backpropagation through the initial value problem with a small memory footprint. In this method, backpropagation through the IVP (1) is performed by solving another IVP, which we call the backward IVP. The backward IVP consists of several groups of equations, which are solved backward-in-time: from $t_1$ to $t_0$. Therefore, the adjoint method can be viewed as a "continuous backpropagation" (see Section 2 for details).

The main property of the adjoint method is that after the forward pass we save $z_1 \triangleq z(t_1)$ and do not store other intermediate values $z(t)$ for other $t \in [t_0, t_1)$. Instead, in the backward pass, we recompute $z(t)$ in required $t \in [t_0, t_1)$. Thus, the following IVP

$$
\begin{align*}
\frac{dz}{dt} &= f(z(t), t, \theta), \quad t \in [t_0, t_1] \\
z(t_1) &= z_1
\end{align*}
$$

is a part of the backward IVP. Gholami et al. (2019) showed that this approach might often lead to catastrophic numerical instabilities and introduced a method, called ANODE, to mitigate this problem. It suggested to make $K$ checkpoints $z(\tau_k)$ at selected time points $\{\tau_k\}_{k=1}^K$ during the forward pass. After that, ANODE performs forward pass on the smaller time intervals $[\tau_k, \tau_{k+1}]$, once again to store intermediate values $z(t)$, $t \in [\tau_k, \tau_{k+1}]$, for backward pass. During the backward pass, instead of solving (2) ANODE uses stored $z(t)$. Thus, to perform backward pass, ANODE requires additional evaluations of the right-hand side in (2). Also, this method requires more memory to store intermediate activations $z(t)$. Moreover, storing intermediate activations in fixed points prevents the usage of the adaptive ODE solvers in gradient computations. Therefore, ANODE is more numerically stable than the adjoint method but suffers from the following drawback. In the backward pass, one needs to solve as many IVPs as the number of stored checkpoints.

To avoid this drawback, we propose interpolated adjoint method (IAM), which is described in detail in Section 2. This method is based on smooth function interpolation technique to approximate $z(t)$ and get rid of IVP (2). Under mild conditions on the right-hand side $f(z(t), t, \theta)$, function $z(t) \in C^1[t_0, t_1]$ as a solution of IVP. We use barycentric Lagrange interpolation (BLI) on a Chebyshev grid (Berrut and Trefethen, 2004) to approximate the solution of the backward IVP. If the approximation is accurate enough, we can replace the solution of IVP (2) with the approximation of $z(t)$. The main requirement for our method to work correctly and efficiently is that $z(t)$ can be approximated by BLI with sufficient accuracy. This can only be verified experimentally. However, the accuracy of such approximation is inherently related to the smoothness of the solution, which is also one of the natural motivations behind using neural ODEs.

Our main contribution reads as follows.

- We approximate activations $z(t)$ with the barycentric Lagrange interpolation on a Chebyshev grid instead of solving IVP (2). Thus, we eliminate equations and reduce the dimension of the backward IVP. As a result, the training becomes faster.
- We study the numerical stability of the proposed IAM and give the error bounds for the adjoint variable.
- We compare our approach with the adjoint method and ANODE in terms of forward/backward pass time, loss, and the number of adaptive solver steps on classification and density estimation problems.

## 2 Interpolated Adjoint Method (IAM)

Most deep learning problems reduce to a minimization of the loss function $L$ with respect to parameter $\theta$ by gradient-based methods. To train neural ODEs, one has to compute gradients of the loss function $L$, which depends on the solution $z(t)$ of the initial value problem (1). To compute $\frac{dL}{dz(t_1)}$, we can use the adjoint method. To use this method, one has to solve IVP (1) and save $z_1 \triangleq z(t_1)$ during the forward pass. The core idea of the adjoint method is to solve

\footnote{For simplicity, we assume that only $z(t_1)$ affects the loss function.}
the following IVP backward in time from $t_1$ to $t_0$
\[
\frac{dz}{dt} = f(z(t), t, \theta), \quad z(t_1) = z_1, \\
\frac{da(t)}{dt} = -a(t)\frac{\partial f(z(t), t, \theta)}{\partial z}, \quad a(t_1) = \frac{\partial L}{\partial z(t_1)}, \\
\frac{dL}{d\theta}/dt = -a(t)\frac{\partial f(z(t), t, \theta)}{\partial \theta}, \quad \frac{dL}{d\theta}(t_1) = 0,
\]
where $a(t)$ is the adjoint variable. We call the system of (3), (4) and (5) the backward IVP. Since the adjoint method does not store intermediate activations $z(t)$ during the forward pass, IVP (3) is used to restore these activations at all required points during the backward pass. It is assumed that solutions of (1) and (3) coincide. Nevertheless, these solutions may differ a lot (Gholami et al., 2019) see Fig. 7). Moreover, even if they coincide, it may take many ODE solver steps to solve (3).

Since IVPs (3), (4) and (5) are managed simultaneously, small time steps to solve (3) can slow down the entire adjoint method.

In the proposed interpolated adjoint method (IAM), we suggest to eliminate (3) from the backward IVP. Instead of IVP (3), the IAM approximates activations $z(t)$ obtained in the forward pass through the barycentric Lagrange interpolation (BLI) on a Chebyshev grid (Berrut and Trefethen, 2004). The detailed description of the BLI is presented below. We urge readers not to confuse the Lagrange interpolation, which is mostly of theoretical interest, with the BLI, that is widely used in practice for polynomial interpolation (Higham, 2004).

Before interpolating a vector function $z(t)$ on a segment $[t_0, t_1]$ with the BLI, one should construct a Chebyshev grid (Tyrtyshnikov, 2012)
\[
t_0 = \tau_0 < \tau_2 < \ldots < \tau_N = t_1
\]
and compute weights \(\{w_n\}_{n=0}^N\), depending only on grid \(\{\tau_n\}_{n=0}^N\). Function $z(t)$ is approximated with
\[
\hat{z}(t) = \frac{N}{n=0} \frac{w_n}{t - \tau_n} \hat{z}_n, \quad \text{where } \hat{z}_n \triangleq z(\tau_n),
\]
and $\hat{z}_n$ are stored during the forward pass. The computational complexity of this approximation as well as the memory usage is $O(N)$. Since we approximate $z(t)$, only (4) and (5) have to be solved during the backward pass. Thus, the dimension of the backward IVP is reduced by the size of $z_1$.

**IAM and ANODE.** Another method that solves the instability issue of the adjoint method is ANODE, presented in (Gholami et al., 2019). ANODE stores checkpoints in the forward pass, and integrates IVP (2) at each subinterval using an ODE solver. In this case, the stability issues are resolved, but memory issues arise. Thus, it is not possible to train neural ODEs with a large batch size. The proposed method aims to resolve both stability and memory issues. In addition, in contrast to ANODE, our method allows using adaptive solvers in the backward pass.

### 3 Approximation Estimates

In this section, we study the stability (Higham, 2002) of the adjoint variable $a(t)$. In other words, we aim to estimate how small perturbations of $z(t)$ affect the corresponding adjoint variable $a(t)$. A well-known stability measure is the norm of the Jacobian matrix. It is often bounded in the spectral normalization technique (Miyato et al., 2018). However, in the theory of ordinary differential equations, much more tight estimate can be obtained by utilizing so-called logarithmic norm (Söderlind, 2006)
\[
\mu[A] \triangleq \lim_{h \to 0^+} \frac{\| I + hA \| - 1}{h},
\]
where $A$ is an arbitrary square matrix. Note that logarithmic norm $\mu[A]$ can be negative. If to choose $\| \cdot \| = \| \cdot \|_2$, then
\[
\mu[A] = \lambda_{\max} \left( \frac{A + A^T}{2} \right),
\]
where $\lambda_{\max}(A)$ is the maximum eigenvalue of a matrix $A$. In this case, logarithmic norm can be computed by the power method (Mises and Pollaczek-Geiringer, 1929).

The following lemma helps to estimate the error of the adjoint variable approximation, which is computed with the interpolated $\hat{z}(t)$ (7) instead of the exact $z(t)$.

\[
\]
Lemma 3.1 (Söderlind and Mattheij [1985], see Lemma 1)

Let \( x(t) \) be a solution of the following non-autonomous linear system

\[
\begin{align*}
\frac{dx}{dt} &= x(t)^\top A(t) + b(t), \\
x(t_0) &= x(t_0).
\end{align*}
\]

(10)

Then \( \| x(t) \| \leq \xi(t) \), where the scalar function \( \xi \) satisfies the IVP

\[
\begin{align*}
\frac{d\xi}{dt} &= \mu [A(t)] \xi + \| b(t) \|, \\
\xi(t_0) &= \| x(t_0) \|.
\end{align*}
\]

(11)

The solution of IVP (11) looks as follows

\[
\xi(t) = \frac{e^{\mu [A(t)] t} - 1}{\mu [A(t)]} \| b(t) \| + \| x(t_0) \| e^{\mu [A(t)] t}.
\]

(12)

If the logarithmic norm \( \mu [A(t)] \) is negative, the upper bound on \( \xi(t) \) does not grow exponentially with \( t \).

Let \( z(t) \) and \( \tilde{z}(t) \) be the exact and the perturbed solutions of IVP (11). Suppose \( a(t) \) and \( \tilde{a}(t) \) are the solutions of IVP (4), which uses \( z(t) \) and \( \tilde{z}(t) \), respectively. Since Lemma 3.1 considers forward-in-time ODEs, we change variables as \( t = t_1 + t_0 - t \) and get

\[
\begin{align*}
\frac{d\hat{a}}{dt} &= \hat{a}(\tau)^\top \frac{\partial f(\hat{z}(\tau), \tau, \theta)}{\partial z} \\
\hat{a}(\tau = t_0) &= a_1.
\end{align*}
\]

(13)

Let us denote partial derivatives in the right-hand sides of (13) by \( J(\tau) \) and \( \tilde{J}(\tau) \), respectively, and the error by \( e(\tau) \equiv a(\tau) - \tilde{a}(\tau) \). Thus, we obtain the following IVP on error

\[
\begin{align*}
\frac{de}{d\tau} &= e(\tau)^\top J(\tau) + \tilde{a}(\tau)^\top (J(\tau) - \tilde{J}(\tau)), \\
e(\tau = t_0) &= 0.
\end{align*}
\]

(14)

Linear initial value problem (14) is non-autonomous, hence Lemma 3.1 states that

\[
\| e(\tau) \| \leq \frac{e^{\tau \mu [J(\tau)]} - 1}{\mu [J(\tau)]} \| \tilde{a}(\tau)^\top (J(\tau) - \tilde{J}(\tau)) \|.
\]

(15)

Lemma 3.1 gives the following upper bound on the norm of perturbed adjoint variable \( \tilde{a}(t) \)

\[
\| a(\tau) \| \leq \| \tilde{a}(\tau) \| e^{\mu [J(\tau)] t}.
\]

(16)

Summarizing (15) and (16), we obtain the following result.

**Theorem 3.2** The norm of the error \( e(\tau) \) is bounded above as

\[
\| e(\tau) \| \leq \frac{e^{2\tau \mu [J(\tau)]} - e^{\tau \mu [J(\tau)]}}{\mu [J(\tau)]} \cdot \| a_1 \| \cdot \| J(\tau) - \tilde{J}(\tau) \|.
\]

(17)

The error norm naturally depends on the smoothness of the Jacobian matrix \( J \). It also depends on the logarithmic norm \( \mu [J(\tau)] \): the smaller the logarithmic norm, the smaller the error norm. If \( \mu [J(\tau)] < 0 \), then \( \| e(\tau) \| \to 0 \) as \( \tau \to \infty \).

### 4 Numerical Experiments

In this section, we illustrate the performance of the IAM on density estimation and classification problems. The single additional hyperparameter in the IAM is the number of Chebyshev grid points \( N \). It is reported separately for each experiment, e.g., IAM(10) means that we use IAM with ten nodes. Our method is implemented on top of
torchdiffeq package. Initially, DOPR5 ODE solver from torchdiffeq provided steps that violated the considered time interval $[t_0, t_1]$; therefore, we have modified its implementation. The default ODE solver in our experiments is DOPR5.

We trace the loss and the number of the right-hand side function evaluations in the forward pass and in the backward pass, respectively. By construction of the IAM, we expect that to achieve the same loss value by performing less number of function evaluations on the backward pass and approximately the same number of function evaluations on the forward pass. The number of function evaluations is the primary metrics of computational complexity. In addition, we report time measurements in supplementary materials. The values of hyperparameters are taken from the repositories mentioned above. All experiments were conducted on a single NVIDIA Tesla V100 GPU with 16Gb of memory.

4.1 Density Estimation

The problem of density estimation is to reconstruct the probability density function using a set of given data points. We consider FFJORD method (Grathwohl et al., 2018) that exploits neural ODE approach to density estimation problem. To perform backpropagation, we test the IAM and the adjoint method on the miniboone dataset. This dataset is used in experiments since it is large enough and FFJORD with the compared methods converges in a reasonable time. Neural ODEs for density estimation problem typically requires a single ODE block. In this setting, ANODE is equal to a vanilla backpropagation with the entire computational graph stored in memory. Therefore, we do not consider ANODE in the context of the density estimation problem. Figure 1 shows the number of function evaluations on the backward pass (Figure 1a), the number of function evaluations on the forward pass (Figure 1c) and train loss (Figure 1b) versus the number of iterations. These plots illustrate that for density estimation problem the IAM requires less number of the right-hand side function evaluations during the backward pass preserving the same train loss value as the adjoint method. We also provide dependence of validation loss on the elapsed running time for the IAM and the adjoint method in supplementary materials (see Figure 5c). A more detailed comparison of the IAM and the adjoint method in terms of running time of forward and backward passes is presented in supplementary materials (see Figure 6).

Figure 1: FFJORD training on miniboone dataset. We run all simulations three times and draw mean ± 0.96σ.

https://github.com/rtqichen/torchdiffeq/
https://github.com/rtqichen/ffjord
4.2 Classification

We test the proposed method on classification problem with CIFAR10 dataset. Since ANODE proposed by Gholami et al. (2019) is superior to the adjoint method for the classification problems, we compare the IAM with ANODE. In contrast to ANODE, the IAM can manage larger number of points in grid. For a model with a single convolutional layer, a single ODEBlock and a linear layer, IAM(16) obtained 0.867 test accuracy with batch size equal to 512 and tolerance 1e-3. Such batch size limits the number of checkpoints in ANODE to 4. More checkpoints did not fit to the memory. ANODE with 4 checkpoints and the same hyperparameters gives similar test accuracy 0.868. Comparison of peak memory consumption for different number of checkpoints and batch sizes is given in Table 1.

| ODE solver   | N  | Batch size | Peak memory, Gb |
|--------------|----|------------|-----------------|
| DOPRI5       | 8  | 1024       | 15.05           |
| DOPRI5       | 16 | 512        | 9.6             |
| RungeKutta (2)| 2  | 1024       | 12.89           |
| RungeKutta (2)| 4  | 512        | 10.74           |

Table 1: Memory consumption comparison of ANODE that uses RungeKutta solver and the IAM that uses DOPRI5 solver. The IAM allows to use more points in the Chebyshev grid than can be used in ANODE for checkpointing.

In supplementary materials we provide comparison of asymptotic memory consumption and a number of the right-hand side evaluations in ANODE and the IAM (see Table 2). We consider two settings: the single ODE block and L sequentially stacked ODE blocks. The presented analysis proves the gain in memory consumption obtained with the IAM. To demonstrate this gain in practice, we plot peak memory for the different numbers of checkpoints and for different models (see Figure 4). This plot illustrates that the IAM can be used with a larger number of checkpoints (which are similar to points in Chebyshev grid) than ANODE. Therefore, an adaptive ODE solver (like DOPRI5) can solve IVP in the backward pass more precisely.

5 Related Work

There is a handful of works, which aim to develop and improve neural ODEs [Chen et al., 2018; Grathwohl et al., 2018; Rubanova et al., 2019; Nguyen et al., 2019] emphasize the importance of using adaptive solvers and introduce a procedure to learn their tolerance with reinforcement learning. A simple procedure to improve the expressive power of neural ODEs by adding dummy variables is proposed by Dupont et al. (2019). Zhang et al., (2019) propose to learn time-dependent parameters with other IVP. In addition, the stiffness of ODE is an important concept (Söderlind et al., 2015) for stable and fast training of neural ODEs, since it affects the number of steps in ODE solver during both backward and forward passes (Wanner and Hairer, 1996).

6 Conclusion

We have developed the interpolated adjoint method (IAM) to improve the adjoint method for the training of neural ordinary differential equations models. The IAM is based on barycentric Lagrange interpolation on a Chebyshev grid. Our method resolves both stability and memory issues arising during the backward pass of neural ODEs. We have compared our approach with the existing methods for classification and density estimation problems. In particular, we have shown that the IAM requires less number of right-hand side function evaluations per epoch than the standard adjoint method for FFJORD training. At the same time, in classification problems the IAM also demonstrates the same accuracy and requires a smaller memory footprint than ANODE.

Acknowledgement

Sections 2 and 4 were supported by Ministry of Education and Science of the Russian Federation grant 14.756.31.0001. Sections 3 and 4 were supported by RFBR grant 19-29-09085. High-performance computations presented in the paper were carried out on Skoltech HPC cluster Zhores.

References

Jean-Paul Berrut and Lloyd N Trefethen. Barycentric Lagrange interpolation. SIAM Review, 46(3):501–517, 2004.
Bo Chang, Lili Meng, Eldad Haber, Frederick Tung, and David Begert. Multi-level residual networks from dynamical systems view. *arXiv preprint arXiv:1710.10348*, 2017.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

John R Dormand and Peter J Prince. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural ODEs. *arXiv preprint arXiv:1904.01681*, 2019.

Kenneth Eriksson, Claes Johnson, and Anders Logg. Explicit time-stepping for stiff ODEs. *SIAM Journal on Scientific Computing*, 25(4):1142–1157, 2004.

Amir Gholami, Kurt Keutzer, and George Biros. ANODE: Unconditionally Accurate Memory-Efficient Gradients for Neural ODEs. *arXiv preprint arXiv:1902.10298*, 2019.

Michael B Giles and Niles A Pierce. An introduction to the adjoint approach to design. *Flow, turbulence and combustion*, 65(3-4):393–415, 2000.

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.

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.

Nicholas J Higham. *Accuracy and Stability of Numerical Algorithms*, volume 80. SIAM, 2002.

Nicholas J Higham. The numerical stability of barycentric Lagrange interpolation. *IMA Journal of Numerical Analysis*, 24(4):547–556, 2004.

Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *arXiv preprint arXiv:1710.10121*, 2017.

Guri I Marchuk. *Adjoint equations and analysis of complex systems*, volume 295. Springer Science & Business Media, 2013.

RV Mises and Hilda Pollaczek-Geiringer. Praktische Verfahren der Gleichungsauflosung. *ZAMM-Journal of Applied Mathematics and Mechanics* Zeitschrift für Angewandte Mathematik und Mechanik, 9(2):152–164, 1929.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Tan M Nguyen, Animesh Garg, Richard G Baraniuk, and Anima Anandkumar. InfoCNF: An efficient conditional continuous normalizing flow with adaptive solvers. *arXiv preprint arXiv:1912.03978*, 2019.

LS Pontryagin, VG Boltyanskii, RV Gamkrelidze, and EF Mishchenko. Mathematical theory of optimal processes (in Russian), 1961.

Yulia Rubanova, Tian Qi Chen, and David K Duvenaud. Latent Ordinary Differential Equations for Irregularly-Sampled Time Series. In *Advances in Neural Information Processing Systems*, pages 5321–5331, 2019.

Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *Journal of Mathematical Imaging and Vision*, pages 1–13, 2018.

Gustaf Söderlind. The logarithmic norm. History and modern theory. *BIT Numerical Mathematics*, 46(3):631–652, 2006.

Gustaf Söderlind and Robert MM Mattheij. Stability and asymptotic estimates in nonautonomous linear differential systems. *SIAM Journal on Mathematical Analysis*, 16(1):69–92, 1985.

Gustaf Söderlind, Laurent Jay, and Manuel Calvo. Stiffness 1952–2012: Sixty years in search of a definition. *BIT Numerical Mathematics*, 55(2):531–558, 2015.

Eugene E Tyrtyshnikov. *A Brief Introduction to Numerical Analysis*. Springer Science & Business Media, 2012.

Gerhard Wanner and Ernst Hairer. *Solving ordinary differential equations II*. Springer Berlin Heidelberg, 1996.

Tianjun Zhang, Zhewei Yao, Amir Gholami, Kurt Keutzer, Joseph Gonzalez, George Biros, and Michael Mahoney. ANODEV2: A Coupled Neural ODE Evolution Framework. *arXiv preprint arXiv:1906.04596*, 2019.
Supplementary materials

Classification

We evaluate the performance of the proposed method on the CIFAR-10 classification problem. We train ResNet-like architecture for 350 epochs with an initial learning rate equal to 0.1. The learning rate decays by a factor of 10 at epoch 150 and 300. Data augmentation is implemented. The batch size used for training is 512. For all experiments with different methods (the standard adjoint method, ANODE, and the IAM), we use the same setting. ResNet-like architecture is a standard ResNet such that each ResNet block, except the first and downsample blocks, is replaced with the ODE block.

![Diagram of ODE block](image1)

**Figure 2:** Forward and backward passes through a single ODE block ($L = 1$). The IAM forward pass uses Chebyshev grid points.

![Diagram of ANODE and IAM for two ODE blocks](image2)

**Figure 3:** Forward and backward passes in ANODE and the IAM for two ODE blocks ($L = 2$). For larger $L$ the plots are similar. The IAM forward pass uses Chebyshev grid points.

Memory consumption. In Table 2, we summarize the number of function evaluations (right-hand side of equation (1)) and the asymptotic memory consumption in the forward and backward passes. We consider the case of a single ODE block (Table 2a) as well as $L$ sequentially stacked ODE blocks (Table 2b).

Assume that in ANODE we use the non-adaptive solver that makes $K_{na}$ function evaluations at each of $N_t$ fixed time steps. During the forward pass through the single ODE block, ANODE performs $N_t$ solver calls, i.e., it evaluates the function (represented with a neural network) $N_t K_{na}$ times. ANODE does not store values at intermediate points of the interval. Therefore, during the backward pass, ANODE firstly recomputes values in intermediate points using $N_t K_{na}$
function calls and then computes gradients using backpropagation. The latter operation requires storing the whole computational graph that requires $O(n_z + N_l(n_z + n_\theta))$ memory, where $n_z$ denotes the number of features in the output of neural network and $n_\theta$ states for the number of network parameters.

In turn, during the forward pass, the IAM requires $NK$ function evaluations, where $N$ is a number of points in the Chebyshev grid and $K$ is a number of function evaluations per solver step. The benefit of the IAM is that during the backward pass it does not require to use the same points as during the forward pass, since $z(t)$ can be interpolated in any time point with activations pre-computed in Chebyshev grid points. Thus, in contrast to ANODE, the IAM does not recompute intermediate points using ODE solver during the backward pass. Illustration of the difference between the forward and backward passes in ANODE and the IAM is presented in Figure 2 for single ODE block) and in Figure 3 (for $L = 2$ stacked ODE blocks). Moreover, the interpolation technique makes possible to use adaptive solvers that are more general than non-adaptive (fixed step) ones.

From Table 2 follows that in contrast to ANODE, the number of function evaluations and memory consumption for IAM does not depend on the number of stacked ODE blocks $L$. Assuming $N_l \approx N$ and $K_{na} \approx K$, we can see that the IAM outperforms ANODE in terms of memory consumption in the backward pass. Although ANODE requires less amount of memory in the forward pass than the IAM, ANODE has to recompute many activations (values at intermediate points) during the backward pass. Therefore, memory-efficient forward pass in ANODE leads to extra computations and memory overhead in the backward pass.

| # function evaluations | Memory | Forward | Backward |
|------------------------|--------|---------|----------|
| ANODE                  | IAM    | $NK_{na}$ | $NK$ |
| $O(n_z)$               | $O(Nn_z)$ | $O(n_z + N_l(n_z + n_\theta))$ | $O(Nn_z + n_\theta)$ |
| (a) $L = 1$            |        |         |          |

| # function evaluations | Memory | Forward | Backward |
|------------------------|--------|---------|----------|
| ANODE                  | IAM    | $NK_{na}L$ | $NK$ |
| $O(n_zL)$              | $O(Nn_z)$ | $O(n_zL + N_l(n_z + n_\theta))$ | $O(Nn_z + n_\theta)$ |
| (b) $L > 1$            |        |         |          |

Table 2: Complexity of ANODE and the IAM during the forward and backward passes through $L \geq 1$ sequentially stacked ODE blocks. We denote by $N$ the number of Chebyshev grid points used in the IAM, and by $N_l$ the number of checkpoints in ANODE. The right-hand side $f$ is represented with a neural network. The number of features in the neural network output is denoted by $n_z$, while $n_\theta$ states for the number of parameters. At one step of adaptive solver, function $f$ is evaluated $K$ times. Since only non-adaptive solvers are appropriate to work with ANODE, we denote the corresponding number of the function $f$ calls by $K_{na}$.

In practice, the IAM requires a smaller memory footprint than ANODE if a small number of ODE blocks is used. Particularly, current neural ODEs for density estimation problems exploit a single ODE block. At the same time, we train neural ODEs with multiple ODE blocks for classification problems. The comparison of memory consumption in such models is provided below.

Figure 4 demonstrates memory requirements for the compared methods. ANODE uses non-adaptive solvers. Therefore, the time step is equal to $1/N_l$, where $N_l$ is the number of checkpoints. If to make the number of checkpoints greater for the considered setting, it will not fit into 16Gb GPU memory.

**Density Estimation**

To demonstrate the benefits of the proposed method in density estimation problem, we carry out experiments using the FFJORD approach. We compare FFJORD performance on miniboone dataset combining this approach with the adjoint method and the IAM. The results have been obtained for the same setting and model architecture as in the original paper [Grathwohl et al., 2018].

The IAM significantly reduces training time compared with the adjoint method due to more effective backward pass. Figure 6 demonstrates that FFJORD with the IAM requires a much smaller number of function evaluations during the backward pass (Figure 6d), while having the number of function calls during the forward pass on par with the adjoint method (Figure 6b). Thus, the IAM reduces the total number of function evaluations (Figure 6c) and total training time (Figure 6e) of FFJORD. Moreover, the overall performance of the model remains the same (Figure 5).
Figure 4: Peak memory dependence on the number of checkpoints for the IAM and ANODE during ODE based model training. The IAM with adaptive solver DOPRI5 allows to use more checkpoints than ANODE, which works with non-adaptive solvers (Euler, RK2, or RK4). The number of checkpoints for the IAM means the number of points in the Chebyshev grid. In this setting, the batch size is equal to 512. We consider a ResNet-like model, which is obtained by replacing each ResNet block (all except the first block and downsample blocks) to the ODE block in a standard ResNet architecture. For example, ResNet-10 stands for “Conv → NormLayer → ReLU → ResNet block → ODE block → ResNet block → ODE block → AvgPooling → Linear” network. If there is only the one block in the model, as in ResNet-4, we replace it to ODE block.

To better illustrate the difference between the IAM and the adjoint method, we provide Figure 5c, where we show validation loss depending on running time.

Figure 5: Density estimation on miniboone dataset with FFJORD. Training and validation losses demonstrate the same convergence behaviour for FFJORD with the IAM and FFJORD with the adjoint method. However, if we consider the dependence of validation loss on running time, we can see slightly faster convergence at the beginning of the training process.
Figure 6: Density estimation on miniiboone dataset with FFJORD. FFJORD with the IAM significantly reduces the total training time and the number of total function evaluations. Comparing to FFJORD with the adjoint method, FFJORD with the IAM performs backward passes faster and requires fewer number of function evaluations. IAM(10) and IAM(14) denote the FFJORD with the interpolated adjoint method that uses 10 and 14 Chebyshev grid points, respectively.