TOWARDS UNDERSTANDING NORMALIZATION IN NEURAL ODES

Julia Gusak, Larisa Markeeva, Talgat Daulbaev, Alexandr Katrutsa,
Andrzej Cichocki & Ivan Oseledets
Skolkovo Institute of Science and Technology
Moscow, Russia
{y.gusak,l.markeeva,talgat.daulbaev}@skoltech.ru
aleksandr.katrutsa@phystech.edu
{a.cichocki,i.oseledets}@skoltech.ru

ABSTRACT

Normalization is an important and vastly investigated technique in deep learning. However, its role for Ordinary Differential Equation based networks (neural ODEs) is still poorly understood. This paper investigates how different normalization techniques affect the performance of neural ODEs. Particularly, we show that it is possible to achieve 93% accuracy in the CIFAR-10 classification task, and to the best of our knowledge, this is the highest reported accuracy among neural ODEs tested on this problem.

1 INTRODUCTION

Neural Ordinary Differential equations (neural ODEs) are proposed in (Chen et al. (2018)) and model the evolution of hidden representation with ordinary differential equation (ODE). The right-hand side of this ODE is represented with some neural network. If one considers classical Euler scheme to integrate this ODE, then ResNet-like architecture (He et al. (2016)) will be obtained. Thus, Neural ODEs are continuous analogue of ResNets. One of the motivation to introduce such models was assumption on smooth evolution of the hidden representation that can be violated with ResNet architecture. Also, in contrast to ResNet models, Neural ODEs share parameters of the ODE right-hand side between steps to integrate this ODE. Thus, Neural ODEs are more memory efficient.

Different normalization techniques were proposed to improve the quality of deep neural networks. Batch normalization (Ioffe & Szegedy (2015)) is a useful technique when training a deep neural network model. However, it requires computing and storing moving statistics for each time point. It becomes problematic when a number of time steps required for different inputs vary as in recurrent neural networks (Hochreiter & Schmidhuber (1997); Cooijmans et al. (2016); Ba et al. (2016)), or the time is continuous as in neural ODEs. We apply different normalization techniques (Salimans & Kingma (2016); Miyato et al. (2018); Ba et al. (2016)) to Neural ODE models and report results for the CIFAR-10 classification task. The considered normalization approaches are compared in terms of test accuracy and ability to generalize if a more powerful ODE solver is used in the inference.

2 BACKGROUND

The main ingredient of the neural ODE architecture is the ODE block. The forward pass through the ODE block is equivalent to the solve the following initial value problem (IVP)

$$\begin{cases} \frac{\mathrm{d}\boldsymbol{z}}{\mathrm{d}t} = f(\boldsymbol{z}(t), t, \boldsymbol{\theta}), \quad t \in [t_0, t_1] \\ \boldsymbol{z}(t_0) = \boldsymbol{z}_0, \end{cases} \tag{1}$$

where $\boldsymbol{z}_0$ denotes the input features, which are considered as initial value. To solve IVP, we numerically integrate system (1) using ODE solver. Depending on the solver type different number of RHS evaluations of (1) are performed. Initial value problem (1) replaces Euler discretization for the same right-hand side that arises in ResNet-like architectures. One part of the standard ResNet-like architecture is the so-called ResNet block, which consists of convolutions, batch normalizations, and ReLU layers. In practice, batch normalization is often used to regularize model, make it more robust to training hyperparameters and reduce internal covariate shift (Shimodaira (2000)). Also, it is shown that batch normalization yields smoother loss surface and makes neural network training faster and more robust (Santurkar et al. (2018)). In the context of neural ODEs training, previous studies applied layer normalization (Chen et al. (2018)) and batch normalization (Gholami et al. (2019)) but did not investigate the influence of these layers on the model performance. In this study, we focus on the role of normalization techniques in neural ODEs. We assume that proper normalization applied to the layers in ODE blocks leads to the higher test accuracy and smoother dynamic.

According to Luo et al. (2018), different problems and neural network architectures require different types of normalization. In our empirical study, we investigate the following normalization techniques to solve the image classification problem with neural ODE models.

- *Batch normalization* (BN; Ioffe & Szegedy (2015)) is the most popular choice for the image classification problem, we discuss its benefits in the above paragraph.
- *Layer normalization* (LN; Ba et al. (2016)) and *weight normalization* (WN; Salimans & Kingma (2016)) were introduced for RNNs. We consider these normalizations as a ppropriate candidates for incorporating in neural ODEs since they showed its effectiveness for RNNs that also exploit the idea of weights sharing through time.
- *Spectral normalization* (SN; Miyato et al. (2018)) was proposed for generative adversarial networks. It is natural to consider SN for neural ODEs since if the Jacobian norm is bounded by 1, one may expect better properties of the gradient propagation in the backward pass.
- We also trained neural ODEs without any normalization (NF).

To perform back-propagation, we use ANODE (Gholami et al. (2019)) approach. This is a memory-efficient procedure to compute gradients in neural ODEs with several ODE blocks. This method exploits checkpointing technique at the cost of extra computations.

3 Numerical Experiments

This section presents numerical results of applying different normalization techniques to neural ODEs in the CIFAR-10 classification task. Firstly, we compare test accuracy for neural ODE based models with different types of normalizations. Secondly, we present an $(\mathcal{S}, n)$-*criterion* to estimate quality of the trained neural ODE-like model. The source code is available at GitHub repository[1].

In our experiments we consider neural ODE based models, which are build by stacking standard layers and ODE blocks. After replacing ResNet block with ODE block in ResNet4 model, we get

$$\text{conv} \rightarrow \textit{norm} \rightarrow \text{activation} \rightarrow \text{ODE block} \rightarrow \text{avgpool} \rightarrow \text{fc}$$

an architecture, which we call *ODENet4*. For this model we test different normalization techniques for *norm* layer and inside the ODE block. Similarly, by replacing in ResNet10 architecture ResNet blocks, which do not change spacial size, with ODE blocks, we get the following model:

$$\text{conv} \rightarrow \textit{norm} \rightarrow \text{activation} \rightarrow \text{ResNet block} \rightarrow \text{ODE block} \rightarrow \text{ResNet block} \rightarrow \text{ODE block} \rightarrow \\ \text{avgpool} \rightarrow \text{fc},$$

which we call *ODENet10*. In contract to ODENet4, this model admits different normalizations in place of the *norm* layer, inside ResNet blocks and ODE blocks.

We use ANODE to train considered models since it is more robust than the adjoint method (more details see in Gholami et al. (2019)). In both forward and backward passes through ODE blocks we solve corresponding ODEs using Euler scheme. For the training schedule, we follow the settings from ANODE (Gholami et al. (2019)). In contract to ANODEDEV2 (Zhang et al. (2019)), we include activations and normalization layers to the model. We train considered models for 350

[1]`https://github.com/juliagusak/neural-ode-norm`

epochs with an initial learning rate equal to 0.1. The learning rate is multiplied by 0.1 at epoch 150 and 300. Data augmentation is implemented. The batch size used for training is 512. For all experiments with different normalization techniques, we use the same settings.

3.1 Accuracy

In our experiments, we assume that normalizations for all ResNet blocks are the same, as well as for all ODE blocks. Along with these two normalizations, we vary a normalization technique after the first convolutional layer. We report test accuracy for different normalization schedules for ODENet10. Table 1 presents test accuracy given by ODENet10 model. The best model achieves 93% accuracy. It uses batch normalization after the first convolutional layer and in the ResNet blocks, and layer normalization in the ODE blocks. Also, we observe that the elimination of batch normalization after the first convolutional layer and from the ResNet blocks leads to decreasing accuracy to 91.2%. Such quality is even worse than the quality obtained with the model without any normalizations (92%).

ODE blocks	BN	WN	SN	NF	LN
Accuracy@1	0.762	0.925	0.926	0.927	$\mathbf{0.930}$

Table 1: Comparison of normalization techniques for ODENet10 architecture on CIFAR-10. BN – batch normalization, LN – layer normalization, WN – weight normalization, SN – spectral normalization, NF – the absence of any normalization. To perform back-propagation, we exploit ANODE with a non-adaptive ODE solver. Namely, we use Euler scheme with $N_t = 8$, where N_t is a number of time steps used to solve IVP (1). The first row corresponds to the normalization in the ODE blocks. We use BN after the first convolutional layer and inside ResNet blocks, respectively. Standard ResNet10 architecture (only ResNet blocks are used) gives $\mathbf{0.931}$ test accuracy.

3.2 $(\mathcal{S}, n)$-criterion of dynamics smoothness in the trained model

Since in neural ODEs like models, we train not only parameters of standard layers, but also parameters in the right-hand side of the system (1), the test accuracy is not the only important measure. Another significant criterion is the smoothness of the hidden representation dynamic that is controlled by the trained parameters of the right-hand side (1).

To implicitly estimate this smoothness, we propose an *$(\mathcal{S}, n)$-criterion* that indicates whether more powerful solver induces performance improvement of the trained neural ODE model during evaluation. Here, $\mathcal{S}$ denotes a solver name (Euler, RK2, RK4, etc) and n denotes a number of the right-hand side evaluations necessary for integration of system (1), which corresponds to the forward pass through the ODE block. By more powerful solver we mean ODE solver that requires more right-hand side evaluations to solve (1) than ODE solver used in training for the same purpose. For example, assume one trains the model with Euler scheme and $n = 2$. Then, we say that ODE block in trained model corresponds to smooth denamics if using Euler scheme with $n > 2$ during evaluation yields higher accuracy. Otherwise, we say that $(\mathcal{S}, n)$-criterion shows the absence of learned smooth dynamics. Worth noting that the $(\mathcal{S}, n)$-criterion has limitation. Namely, it requires the solution of IVP (1) to be a Lipchitz function of the right-hand side (1) parameters and inputs (Coddington & Levinson (1955)). Otherwise, we can not rely on this criterion since the closeness in the right-hand side parameters does not induce the closeness of features that are inputs to the next layers of the model.

In our experiments we consider ODENet4 architecture with four different settings of the Euler scheme: $n = 2, 8, 16, 32$. For each setting we have trained 10 types of architectures that differ from each other by the type of normalization we apply to the first convolutionl layer and convolutional layers in the ODE block. For example, the model named "ODENet4 BN-LN (Euler, 2)" means the following: we have used ODENet4 architecture, where after conv layer follows a BN layer, after each convolutional layer in the right-hand side (1) follows an LN layer, and Euler scheme with 2 steps is used to propagate through the ODE block.

For a fixed model trained with (Euler, n_0) solver we check the fulfillment of $(\mathcal{S}, n)$-criterion by evaluating its accuracy with more powerful solver. In this case, we consider the following more powerful solvers: (Euler, n), (RK2, n) and (RK4, n), where $n > n_0$.

In Figure 1, we show how test accuracy given by ODENet4 model with different normalizations changes with varying ODE solvers to integrate IVP (1) in ODE blocks. Different line types correspond to different solver type (Euler, RK2, RK4), x-axis depicts the number of the right-hand side evaluations, while y-axis stands for the test accuracy. These models were trained with Euler scheme and after that we use Euler, RK2 and RK4 schemes to compute test accuracy. Every row from top to bottom corresponds to $n = 2, 16, 32$ used in Euler scheme.

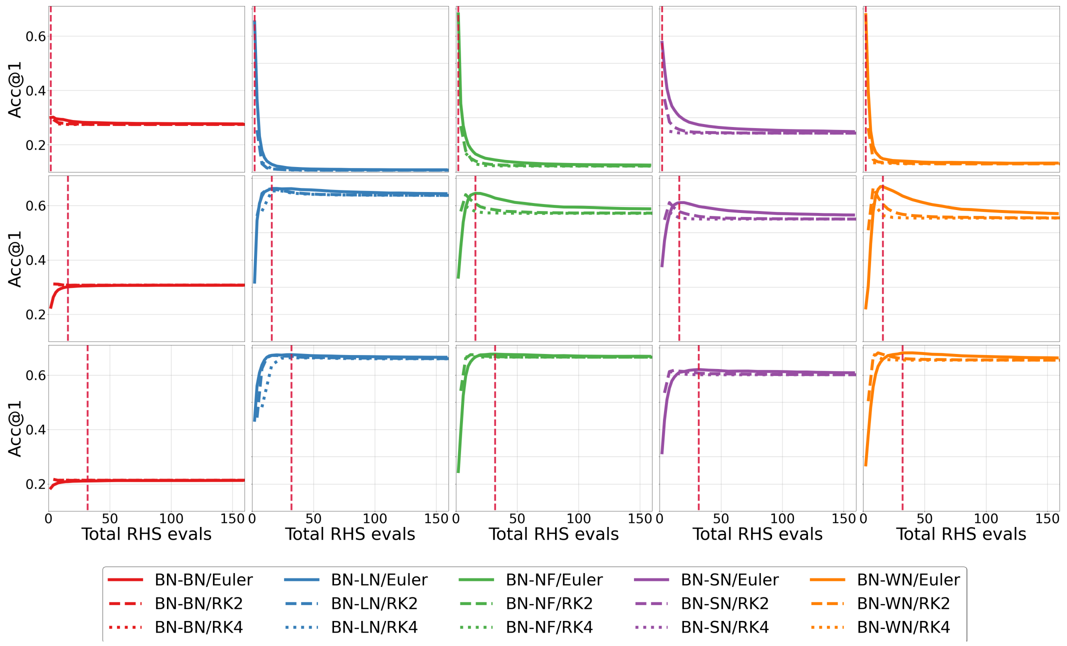

Figure 1: Illustration of how the choice of ODE solver and normalizations during training implicitly affects the smoothness of learned dynamics. Each subplot corresponds to the model trained with a fixed ODE solver and normalization scheme. Models within one row have the same type of training solver ((Euler, n), $n = 2, 16, 32$ from top to bottom). Models within one column have the same normalization technique. For example, subplot in the third row and the second column corresponds to the ODENet4 model trained with (Euler, 32) solver with BN after the first convolutional layer and LN after convolutional layers inside ODE block. Lines of different style corresponds to different types of test solvers. If model accuracy does not drop when the more powerful ODE solver is used, we conclude that, according to $(\mathcal{S}, n)$-criterion, the model provides a smooth dynamics. For example, the model (Euler, 32) BN-LN trains a smooth dynamics, while (Euler, 2) BN-LN fails to do that. Also, we can observe that to learn the smooth dynamics during training, for some normalization schemes less powerful solvers are required. If we compare BN-LN and BN-WN models, we can see that the first one learns smooth dynamics when Euler with $n = 16$ is used, but the latter one does that only for $n = 32$.

4 Discussion and Further research

We have empirically investigated the effect of normalization techniques to ODE based models. For different models, we have compared test accuracy as well as the ability to learn parameters that yield smooth dynamics of hidden representations. We have observed that both normalization and type of training solver affect the performance of the final model. Worth noting that pre-trained models, which are close in terms of test accuracy, can significantly differ when it comes to the smoothness of the hidden representations dynamics. In our further research, we plan to investigate how different tasks benefit from the presence of smoothness in hidden representations. Also, we will work on a more rigorous theoretical criterion that can be used to compare ODE based models, considering both neural networks and ODEs metrics.

Acknowledgement

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

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pp. 6571–6583, 2018.

Earl A Coddington and Norman Levinson. *Theory of ordinary differential equations*. Tata McGraw-Hill Education, 1955.

Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

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

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*, pp. 770–778, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Ping Luo, Jiamin Ren, Zhanglin Peng, Ruimao Zhang, and Jingyu Li. Differentiable learning-to-normalize via switchable normalization. *arXiv preprint arXiv:1806.10779*, 2018.

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

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pp. 901–909, 2016.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pp. 2483–2493, 2018.

Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

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.