Sparse Flows: Pruning Continuous-depth Models

Anonymous Author(s)
Affiliation
Address
email

Abstract

Continuous deep learning architectures enable learning of flexible probabilistic models for predictive modeling as neural ordinary differential equations (ODEs), and for generative modeling as continuous normalizing flows. In this work, we design a framework to decipher the internal dynamics of these continuous depth models by pruning their network architectures. Our empirical results suggest that pruning improves generalization for neural ODEs in generative modeling. Moreover, pruning finds minimal and efficient neural ODE representations with up to 98% less parameters compared to the original network, without loss of accuracy. Finally, we show that by applying pruning we can obtain insightful information about the design of better neural ODEs. We hope our results will invigorate further research into the performance-size trade-offs of modern continuous-depth models.

1 Introduction

The continuous analog of normalizing flows (CNFs) (Chen et al., 2018) efficiently (Grathwohl et al., 2019) maps a latent space to data by ordinary differential equations (ODEs), relaxing the strong constraints over *discrete* normalizing flows (Dinh et al., 2016, Durkan et al., 2019, Huang et al., 2020, Kingma and Dhariwal, 2018, Papamakarios et al., 2017, Rezende and Mohamed, 2015). CNFs enable learning flexible probabilistic models by arbitrarily-chosen neural network topologies. While recent works investigated ways to improve CNFs' efficiency (Finlay et al., 2020, Grathwohl et al., 2019, Li et al., 2020), regularize the flows (Onken et al., 2020, Yang and Karniadakis, 2020), or solving their fundamental shortcomings such as crossing trajectories (Dupont et al., 2019, Massaroli et al., 2020), less is understood about their inner dynamics during and post training.

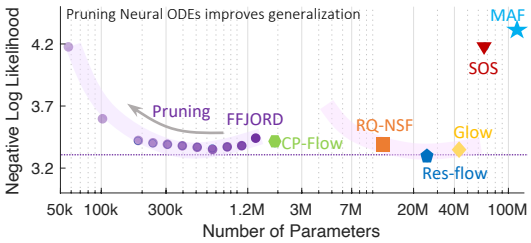


Figure 1: Pruning Neural ODEs improves their generalization with at least 1 order of magnitude less parameters. CIFAR-10 density estimation. Values and methods are described in Table 3.

In this paper, we set out to use standard pruning algorithms to investigate generalization properties of sparse neural ODEs and continuous normalizing flows. In particular, we investigate how the inner dynamics and the modeling performance of a continuous flow varies if we methodologically prune its neural network architecture. Reducing unnecessary weights of a neural network (pruning) (Han et al., 2015b, Hassibi and Stork, 1993, LeCun et al., 1990, Li et al., 2016) without loss of accuracy results in smaller network size (Hinton et al., 2015, Liebenwein et al., 2021), computational efficiency (Luo et al., 2017, Molchanov et al., 2019, Yang et al., 2017), faster inference (Frankle and Carbin, 2019), and enhanced interpretability (Baykal et al., 2019, Lechner et al., 2020, Liebenwein et al., 2020).

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

Here, our main objective is to better understand CNFs' dynamics in density estimation tasks as we increase network sparsity and to show that pruning can improve generalization in neural ODEs.

Pruning improves generalization in neural ODEs. Our results consistently suggest that a certain ratio of pruning of fully connected neural ODEs leads to lower empirical risk in density estimation tasks, thus obtaining better generalization. We validate this observation on a large series of experiments with increasing dimensionality. See an example here in Figure 2.

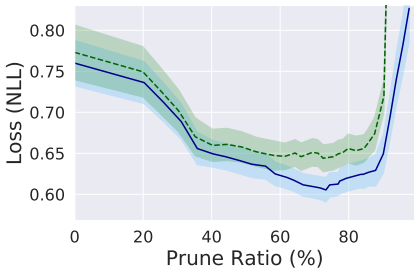


Figure 2: Pruning enhances generalization of continuous-depth models. Structured pruning (green), unstructured pruning (blue). More details in Section 4.

Pruning flattens the loss surface of neural ODEs. Additionally, we conduct a Hessian-based empirical investigation on the objective function of the flows-under-test in density estimation tasks to better understand why pruning results in better generalization. We find that for Neural ODEs, pruning decreases the value of the Hessian's eigenvalues, and as a result, flattens the loss which leads to better generalization, c.f., Keskar et al. (2017) (Figure 3).

Pruning helps avoiding mode-collapse in generative modeling. In a series of multi-modal density estimation tasks, we observe that densely connected CNFs often get stuck in a sharp local minimum (See Figure 3) and as a result, cannot properly distinguish different modes of data. This phenomena is known as mode-collapse. Once we sparsify the flows, the quality of the density estimation task increases significantly and consequently mode-collapse does not occur.

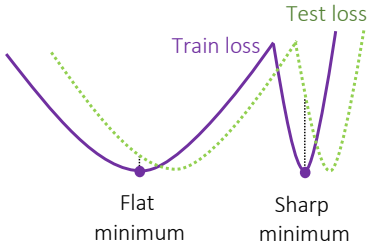


Figure 3: Flat minima results in better generalization compared to sharp minima. Pruning neural ODEs flattens the loss around local minima. Figure is reproduced from Keskar et al. (2017).

Pruning finds minimal and efficient neural ODE representations. Our framework finds highly optimized and efficient neural ODE architectures via pruning. In many instances, we can reduce the parameter count by 70-98% (6x-50x compression rate). Notably, one cannot directly train such sparse and efficient continuous-depth models from scratch.

2 Background

In this section, we describe the necessary background topics to construct our framework. We show how to perform generative modeling by continuous depth models using the change of variables formula.

Generative modeling via change of variables. The change of variables formula uses an invertible mapping $f : \mathbb{R}^D \to \mathbb{R}^D$, to wrap a normalized base distribution $p_z(\mathbf{z})$, to specify a more complex distribution. In particular, given $z \sim p_z(\mathbf{z})$, a random variable, the log density for function $f(\mathbf{z}) = \mathbf{x}$ can be computed by Grathwohl et al. (2019):

$$\log p_x(\mathbf{x}) = \log p_z(\mathbf{z}) - \log \det \left| \frac{\partial f(\mathbf{z})}{\partial \mathbf{z}} \right|, \tag{1}$$

where the Jacobian of f is determined by $\frac{\partial f(\mathbf{z})}{\partial \mathbf{z}}$. While theoretically Eq. 1 demonstrates a straightforward way to find the log density, from a practical standpoint computation of the Jacobian determinant has a time complexity of $\mathcal{O}(D^3)$. Restricting neural network architectures can make the computation of this term more tractable. Examples include designing normalizing flows (Berg et al., 2018, Papamakarios et al., 2017, Rezende and Mohamed, 2015), autoregressive transformations (Durkan et al., 2019, Jaini et al., 2019, Kingma et al., 2016, Müller et al., 2019, Oliva et al., 2018a, Wehenkel and Louppe, 2019), partitioned transformations (Dinh et al., 2016, Kingma and Dhariwal, 2018), universal flows (Kong and Chaudhuri, 2020, Teshima et al., 2020), and the use of optimal transport theorem (Huang et al., 2020).

Alternative to these discrete transformation algorithms, one can construct a generative model similar to (1), and declare f by a continuous-time dynamics (Chen et al., 2018, Grathwohl et al., 2019).

Given a sample from the base distribution, one can parametrize an ordinary differential equations (ODEs) by a function $f(\mathbf{z}(t), t, \theta)$, and solve the ODE to obtain the observable data. When f is a neural network, the system is called a neural ODE (Chen et al., 2018).

Neural ODEs. More formally, a neural ODE is defined by finding the solution to the initial value problem (IVP): $\frac{\partial \mathbf{z}(t)}{\partial t} = f(\mathbf{z}(t), t, \theta)$, $\mathbf{z}(t_0) = \mathbf{z}_0$, with $\mathbf{z}_0 \sim p_{z_0}(\mathbf{z}_0)$, to get $\mathbf{z}(t_n)$ the desired output observations at a terminal integration step n (Chen et al., 2018).

Continuous Normalizing Flows. If $\mathbf{z}(t_n)$ is set to our observable data, given samples from the base distribution $\mathbf{z}_0 \sim p_{z_0}(\mathbf{z}_0)$, the neural ODE described above forms a continuous normalizing flow (CNF). CNFs modify the change in log density by the left hand-side differential equation and as a result the total change in log-density by the right hand-side equation (Chen et al., 2018, Grathwohl et al., 2019):

$$\frac{\partial \log p(\mathbf{z}(t))}{\partial t} = -\text{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right), \quad \log p(\mathbf{z}(t_n)) = \log p(\mathbf{z}(t_0)) - \int_{t_0}^{t_1} \text{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) dt. \tag{2}$$

The system of two differential equations (the neural ODE ($\frac{\partial \mathbf{z}(t)}{\partial t} = f(\mathbf{z}(t), t, \theta)$) and (2) can then be solved by automatic differentiation algorithms such as backpropagation through time (Rumelhart et al., 1986) or the adjoint sensitivity method (Chen et al., 2018, Pontryagin, 2018). Computation of $\text{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right)$ costs $\mathcal{O}(D^2)$. A method called the Free-form Jacobian of Reversible Dynamics (FFJORD) (Grathwohl et al., 2019) improved the cost to $\mathcal{O}(D)$ by using the Hutchinson's trace estimator (Adams et al., 2018, Hutchinson, 1989). Thus, the trace of the Jacobian can be estimated by: $\text{Tr}\left(\frac{\partial f}{\partial \mathbf{z}(t)}\right) = \mathbb{E}_{p(\boldsymbol{\varepsilon})}\left[\boldsymbol{\varepsilon}^T \frac{\partial f}{\partial \mathbf{z}(t)} \boldsymbol{\varepsilon}\right]$, where $p(\varepsilon)$ is typically set to a Gaussian or Rademacher distribution (Grathwohl et al., 2019). Throughout the paper, we investigate the properties of FFJORD CNFs by pruning their neural network architectures.

3 Pruning Neural ODEs

We enable sparsity in neural ODEs and CNFs by removing, i.e., pruning, redundant weights from the underlying neural network architecture during training. Pruning can tremendously improve the parameter efficiency of neural networks across numerous tasks, such as computer vision (Liebenwein et al., 2020) and natural language processing (Maalouf et al., 2021).

3.1 A General Framework for Training Sparse Flows

Our approach to training Sparse Flows is inspired by *iterative learning rate rewinding*, a recently proposed and broadly adopted pruning framework as used by Liebenwein et al. (2020), Renda et al. (2020) among others.

In short, our pruning framework proceeds by first training an unpruned, i.e., dense network to obtain a warm initialization for pruning. Subsequently, we proceed by iteratively pruning and retraining the network until we either obtain the desired level of sparsity, i.e., prune ratio or when the loss for a pre-specified hold-out dataset (validation loss) starts to deteriorate (early stopping).

We note that our framework is readily applicable to any continuous-depth model and not restricted to FFJORD-like models. Moreover, we can account for various types of pruning, i.e., unstructured pruning of weights and structured pruning of neurons or filters. An overview of the framework is provided in Algorithm 1 and we provide more details below.

3.2 From Dense to Sparse Flows

TRAIN a dense flow for a warm initialization. To initiate the training process, we first train a densely-connected network to obtain a warm initialization (Line 2 of Algorithm 1). We use Adam with a fixed step learning decay schedule and weight decay in some instances. Numerical values for each network and dataset are provided in the supplementary material. Based on the warm initialization, we start pruning the network.

Algorithm 1 SPARSEFLOW(f, Φ_{train}, PR, e)

Input: f: neural ODE model with parameter set θ; Φ_{train}: hyper-parameters for training; PR: relative prune ratio; e: number of training epochs per prune-cycle.
Output: $f(\cdot; \hat{\theta})$: Sparse Flow; m: sparse connection pattern.

1: $\theta_0 \leftarrow$ RANDOMINIT()
2: $\theta \leftarrow$ TRAIN($\theta_0, \Phi_{\text{train}}, e$) ▷ Initial training stage with dense neural ODE ("warm start").
3: $m \leftarrow 1^{|\theta_0|}$ ▷ Initialize binary mask indicating neural connection pattern.
4: **while** validation loss of Sparse Flow decreases **do**
5: $m \leftarrow$ PRUNE($m \odot \theta, PR$) ▷ Prune $PR\%$ of the *remaining* parameters and update binary mask.
6: $\theta \leftarrow$ TRAIN($m \odot \theta, \Phi_{\text{train}}, e$) ▷ Restart training with updated connection pattern.
7: **end while**
8: $\hat{\theta} \leftarrow m \odot \theta$
9: **return** $f(\cdot; \hat{\theta})$, m

PRUNE for Sparse Flow. For the prune step (Line 5 of Algorithm 1) we either consider unstructured or structured pruning, i.e., weight or neuron/filter pruning, respectively. At a fundamental level, unstructured pruning aims at inducing sparsity into the parameters of the flow while structured pruning enables reducing the dimensionality of each flow layer. For unstructured pruning, we use magnitude pruning (Han et al., 2015a)), where we prune weights across all layers (global) with magnitudes below a pre-defined threshold. For structured pruning, we use the ℓ_1-norm of the weights associated with the neuron/filter and prune the structures with lowest norm for constant per-layer prune ratio (local) as proposed by Li et al. (2016). See Table 1 for an overview.

Table 1: Pruning Methods.

	Unstructured (Han et al., 2015a)	Structured (Li et al., 2016)
Target	Weights	Neurons
Score	$\lvert W_{ij} \rvert$	$\lVert W_{i:} \rVert_1$
Scope	Global	Local

TRAIN the Sparse Flow. Following the pruning step, we re-initiate the training with the new sparsity pattern and the unpruned weights (Line 6 of Algorithm 1). Note that we generally follow the same hyperparameters as during the initial, dense training stage.

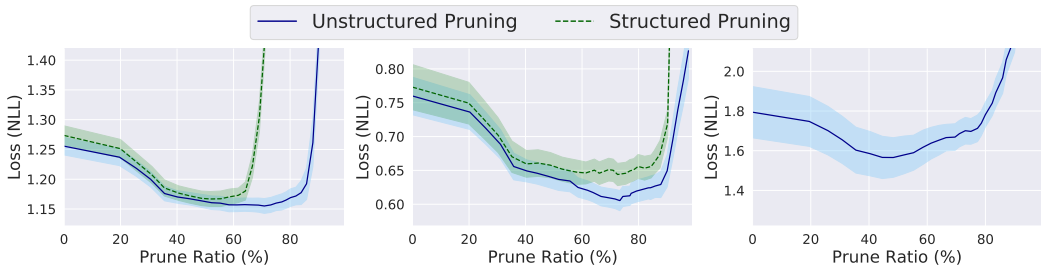
ITERATE for increased sparsity and performance. Naturally, we can iteratively repeat the PRUNE and TRAIN step to further sparsify the flow (Lines 4-7 of Algorithm 1). Moreover, the resulting sparsity-performance trade-off is affected by the total number of iterations, the relative prune ratio PR per iteration, and the amount of training between PRUNE steps. We find that a good trade-off is to keep the amount of training constant across all iterations and tune it such that the initial, dense flow is essentially trained to (or close to) convergence. Depending on the difficulty of the task and the available compute resources we can then adapt the per-iteration prune ratio PR. Note that the overall relative sparsity after n iterations is given by $(1 - PR)^n$. Detailed hyperparameters for each experiment are provided in the supplementary material.

4 Experiments

We perform a diverse set of experiments demonstrating the effect of pruning on the generalization capability of continuous-depth models. Our experiments include pruning ODE-based flows in density estimation tasks with increasing complexity, as well as pruning neural-ODEs in supervised inference tasks. The density estimation tasks were conducted on flows equipped with Free-form Jacobian of Reversible Dynamics (FFJORD) (Grathwohl et al., 2019), using adaptive ODE solvers (Dormand and Prince, 1980). We used two code bases (FFJORD from Grathwohl et al. (2019) and TorchDyn (Poli et al., 2020a)) over which we implemented our pruning framework.[1]

Baselines. In complex density estimation tasks we compare the performance of Sparse Flows to a variety of baseline methods including: FFJORD (Grathwohl et al., 2019), masked autoencoder density

[1] All code and data are submitted as supplements for revision and will be released upon acceptance.



(a) Gaussians (b) Gaussian Spiral (c) Spiral

Figure 4: Negative log likelihood of Sparse Flow as function of prune ratio.

estimation (MADE) (Germain et al., 2015), Real NVP (Dinh et al., 2016), masked autoregressive flow (MAF) (Papamakarios et al., 2017), Glow (Kingma and Dhariwal, 2018), convex potential flows (CP-Flow) (Huang et al., 2020), transformation autoregressive networks (TAN) (Oliva et al., 2018b), neural autoregressive flows (NAF) (Huang et al., 2018), and sum-of-squares polynomial flow (SOS) (Jaini et al., 2019).

4.1 Density Estimation on 2D Data

In the first set of experiments, we train FFJORD on a multi-modal Gaussian distribution, a multi-model set of Gaussian distributions placed orderly on a spiral as well as a spiral distribution with sparse regions. Figure 5 (first row) illustrates that densely connected flows (prune ratio = 0%) might get stuck in sharp local minima and as a result induce mode collapse (Srivastava et al., 2017). Once we perform unstructured pruning, we observe that the quality of the density estimation in all tasks considerably improves, c.f. Figure 5 (second and third rows). If we continue sparsifying the flows, depending on the task at hand, the flows get disrupted again.

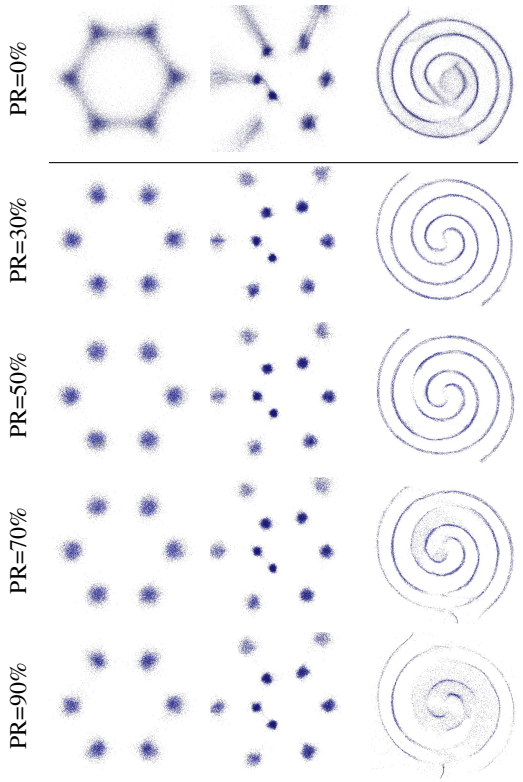


Figure 5: Unstructured pruning of FFJORD (PR= Prune ratio).

Therefore, there is a certain threshold for pruning flows required to avoid generative modeling issues such as mode-collapse in continuous flows. We validate this observation by plotting the negative log-likelihood loss as a function of the prune ratio in all three tasks with both unstructured and structured pruning. As shown in Figure 4, we confirm that sparsity in flows improves the performance of continuous normalizing flows.

We further explore the inner dynamics of the flows between unpruned and pruned networks on the multi-modal case, with the aim of understanding how pruning enhances density estimation performance. Figure 6 represents the vector-field constructed by each flow to model 6-Gaussians independently. We observe that sparse flows with PR of 70% attract the vector-field directions uniformly towards the mean of each Gaussian distribution. In contrast, unpruned flows do not exploit this feature and contain converging vectors in between Gaussians. This is how the mode-collapse occurs.

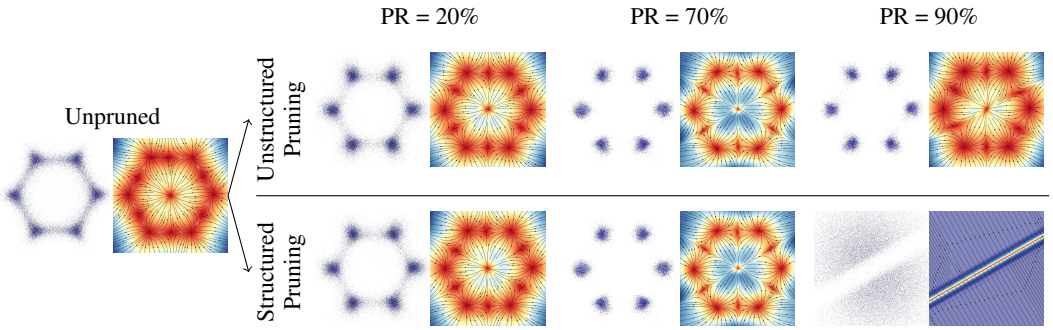


Figure 6: Multi-modal Gaussian flow and pruning. We observe that Sparse Flows attract the vector-field directions uniformly towards the mean of each Gaussian distribution, while an unpruned flow does not exploit this feature and contains converging vectors in between Gaussians.

Table 2: Negative test log-likelihood (NLL) in nats of tabular datasets from (Papamakarios et al., 2017) and corresponding architecture size in number of parameters (#params). Sparse Flow (based on FFJORD) with lowest NLL and competing baseline with lowest NLL are bolded.

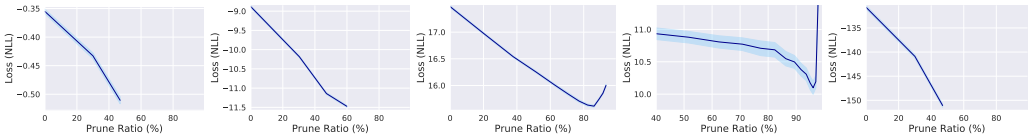
Model	POWER		GAS		HEPMASS		MINIBOONE		BSDS300	
	nats	#params	nats	#params	nats	#params	nats	#params	nats	#params
MADE (Germain et al., 2015)	3.08	5.90K	-3.56	6.20K	20.98	147K	15.59	164K	-148.85	621K
Real NVP (Dinh et al., 2016)	-0.17	212K	-8.33	216K	18.71	5.46M	13.84	5.68M	-153.28	22.3M
MAF (Papamakarios et al., 2017)	-0.24	59.0K	-10.08	62.0K	17.70	1.47M	11.75	1.64M	-155.69	6.21M
Glow (Kingma and Dhariwal, 2018)	-0.17	N/A	-8.15	N/A	18.92	N/A	11.35	N/A	-155.07	N/A
CP-Flow (Huang et al., 2020)	-0.52	5.46M	-10.36	2.76M	16.93	2.92M	10.58	379K	-154.99	2.15M
TAN (Oliva et al., 2018b)	**-0.60**	N/A	**-12.06**	N/A	**13.78**	N/A	11.01	N/A	**-159.80**	N/A
NAF (Huang et al., 2018)	**-0.62**	**451K**	**-11.96**	**443K**	15.09	10.7M	**8.86**	**8.03M**	-157.73	42.3M
SOS (Jaini et al., 2019)	**-0.60**	**212K**	**-11.99**	**256K**	15.15	4.43M	**8.90**	**6.87M**	-157.48	9.09M
FFJORD (Grathwohl et al., 2019)	-0.35	43.3K	-8.58	279K	17.53	547K	10.50	821K	-128.33	6.70M
Sparse Flow	-0.45	30.3K	-10.79	194K	16.53	340K	10.84	397K	-145.62	4.69M
	-0.50	**23K**	-11.19	147K	15.82	160K	10.81	186K	-148.72	3.55M
			-11.59	**85K**	**15.60**	**75K**	**9.95**	**32K**	**-149.92**	**1.86M**
			-11.47	64K	15.99	46K	10.54	18K		

4.2 Density Estimation on Real Data - Tabular

We scale our experiments to a set of five real-world tabular datasets (prepared based on the instructions given by Papamakarios et al. (2017) and Grathwohl et al. (2019)) to verify our empirical observations about the effect of pruning on the generalizability of continuous normalizing flows. Table 2 summarizes the results. We observe that sparsifying FFJORD flows substantially improves their performance in all 5 tasks. In particular, we gain up to 42% performance gain in the POWER, 35% in GAS, 12% in HEPMASS, 5% in MINIBOONE and 19% in BSDS300.

More importantly, this is achieved with flows with 1 to 3 orders of magnitude less parameters compared to other advanced flows. On MINIBOONE dataset for instance, we found a sparse flow with only 4% of its original network that outperforms its densely-connected FFJORD flow. On MINIBOONE, Autoregressive flows (NAF) and sum-of-squares models (SOS) which outperform all other models possess 8.03 and 6.87 million parameters. In contrast, we obtain a Sparse Flow with only 32K parameters that outperform all models except NAF and SOS.

Let us now look at the loss versus prune-ratio trends in all experiments to conclude our empirical observations on real-world tabular datasets. As shown in Figure 8, we observe that pruning considerably improves the performance of flows at larger scale as well.



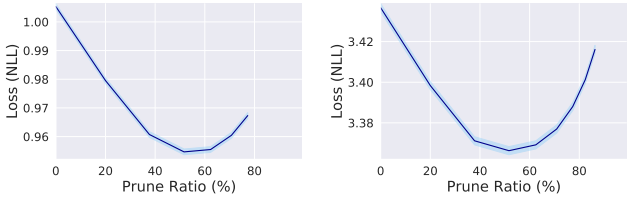
(a) Power (b) Gas (c) Hepmass (d) Miniboone (e) Bsds300

Figure 8: Negative log-likelihood versus prune ratio on tabular datasets with unstructured Pruning.

4.3 Density Estimation on Real-Data - Vision

Next, we extend our experiments to density estimation for image datasets, MNIST and CIFAR10. We observe a similar case on generative modeling with both datasets, where pruned flows outperform densely-connected FFJORD flows. On MNIST, a sparse FFJORD flow with 63% of its weights pruned outperforms all other benchmarks. Compared to the second best flow (Residual flow), our sparse flow contains 70x less parameters (234K vs 16.6M). On CIFAR10, we achieve the second best performance with over 38x less parameters compared to Residual flows which performs best.

Furthermore, on CIFAR10, Glow with 44 million parameters performs on par with our sparse FFJORD network with 657k parameters. It is worth mentioning that FFJORD obtains this results by using a simple Gaussian prior, while Glow takes advantage of learned base distribution (Grathwohl et al., 2019). Figure 7 illustrates the improvement of the loss (negative log-likelihood) in density estimation on image datasets as a result of pruning neural ODEs. Around 60% sparsity of a continuous normalizing flow leads to better generative modeling compared to densely structured flows.



(a) MNIST. (b) CIFAR10.

Figure 7: Loss vs. prune ratio for MNIST and CIFAR10. Unstructured Pruning is applied.

Table 3: Negative test log-likelihood (NLL) in bits/dim for image datasets and corresponding architecture size in number of parameters (#params). Sparse Flow (based on FFJORD) with lowest NLL and competing baseline with lowest NLL are bolded.

Model	MNIST		CIFAR-10	
	bits/dim	#params	bits/dim	#params
MADE (Germain et al., 2015)	1.41	1.20M	5.80	11.5M
Real NVP (Dinh et al., 2016)	1.05	N/A	3.49	N/A
MAF (Papamakarios et al., 2017)	1.91	12.0M	4.31	115M
Glow (Kingma and Dhariwal, 2018)	1.06	N/A	3.35	44.0M
CP-Flow (Huang et al., 2020)	1.02	2.90M	3.40	1.90M
TAN (Oliva et al., 2018b)	1.19	N/A	3.98	N/A
SOS (Jaini et al., 2019)	1.81	17.2M	4.18	67.1M
RQ-NSF (Durkan et al., 2019)		N/A	3.38	11.8M
Residual Flow (Chen et al., 2019)	**0.97**	**16.6M**	**3.28**	**25.2M**
FFJORD (Grathwohl et al., 2019))	1.01	801K	3.44	1.36M
Sparse Flows (PR=20%)	0.97	641K	3.38	1.09M
Sparse Flows (PR=38%)	0.96	499K	3.37	845K
Sparse Flows (PR=52%)	0.95	387K	**3.36**	**657K**
Sparse Flows (PR=63%)	**0.95**	**302K**	3.37	510K
Sparse Flows (PR=71%)	0.96	234K	3.38	395K
Sparse Flows (PR=77%)	0.97	182K	3.39	308K
Sparse Flows (PR=82%)	0.98	141K	3.40	239K
Sparse Flows (PR=86%)	0.97	109K	3.42	186K

Table 4: Eigenanalysis of the Hessian H in terms of the largest eigenvalue (λ_{max}), trace (tr), and condition number (κ) of pruned and unpruned continuous normalizing flows on the mixture of Gaussian task. Numbers are normalized with respect to the unpruned flow.

Model	NLL	$\lambda_{max}(H)$	tr(H)	$\kappa(H)$
Unpruned FFJORD	1.309	1.000	1.000	1.000
Sparse Flows (PR=20%)	1.163	0.976	0.858	0.825
Sparse Flows (PR=60%)	1.125	0.356	0.583	0.717
Sparse Flows (PR=70%)	**1.118**	**0.295**	**0.340**	**0.709**

4.4 Pruning Flattens the Loss Surface for Neural ODEs

What could be the potential reason for the enhanced generalization of pruned CNFs besides their ability to resolve mode-collapse which we observed before? To investigate this further, we conducted a Hessian-based analysis on the flows-under-test. Dissecting the properties of the Hessian by Eigenanalysis allows us to gain useful insights about the behavior of neural networks (Erichson et al., 2021, Ghorbani et al., 2019, Hochreiter and Schmidhuber, 1997, Sagun et al., 2017). We use PyHessian (Yao et al., 2020) tool-set to analyze the Hessian H w.r.t. the parameters of the CNF. This enables us to study the curvature of the loss function as the eigenvalues of the Hessian determines (locally) the loss gradients' changes.

Larger Hessian eigenvalues therefore, stand for sharper curvatures and their sign identifies upward or downward curvatures. In Table 4, we report the maximum eigenvalue of the Hessian λ_{max}, Hessian's Trace, and the condition number $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$.[2] Smaller λ_{max} and tr(H) indicates that our normalizing flow found a flatter minimum. As shown in Figure 3, a flat minimum leads to a better generalization error as opposed to a sharp minimum. We find that up to a certain prune ratio, the maximum of the Hessian decreases and increases thereafter. Therefore, we claim that pruned continuous flows finds flatter local minimum, therefore, it generalize better than their unpruned version. Moreover, the Hessian condition number κ could be an indicator of the robustness and efficiency of a deep model (Bottou and Bousquet, 2008). Smaller κ corresponds to obtaining a more robust learned agent. We observe that κ also follows the same trend and up to a certain prune ratio it shrinks and then increases again confirming our hypothesis.

4.5 On the Robustness of Decision Boundaries of Pruned ODE-based Flows

While pruning improves performance, it is imperative to investigate their robustness properties in constructing decision boundaries. For feedforward deep models it was recently shown that although pruned networks perform as well as their unpruned version, their robustness significantly decreases due to smaller network size (Liebenwein et al., 2021). Is this also the case for neural ODEs?

We design an experiment to investigate this. We take a simple 2-Dimensional moon dataset and perform classification using unpruned and pruned neural ODEs.[3] This experiment (shown in Figure 9) demonstrates another valuable property of neural ODEs: We observe that neural ODE instances pruned up to 84%, manage to establish a safe decision boundary for the two half moon classes. This insight about neural ODEs which was obtained by our pruning framework is another testimony of the merits of using neural ODE in decision-critical applications. Additionally, we observe that the decision-boundary gets very close to one of the classes for a network pruned up to 96% which reduces robustness. Though even this network instance provide the same classification accuracy compared to the densely connected version.

The state-space plot illustrates that a neural ODE's learned vector-field becomes edgier as we keep pruning the network. Nonetheless, we observe that the distribution of the vector-field's intensity (the color map in the second column) get attenuated as we go further away from its center, in highly pruned networks. This indicates that classification near the decision boundary is more sensitive to perturbations in extremely pruned networks.

[2]This experiment was inspired by the Hessian-based robustness analysis performed in Erichson et al. (2021).
[3]This experiment is performed using the TorchDyn library (Poli et al., 2020a).

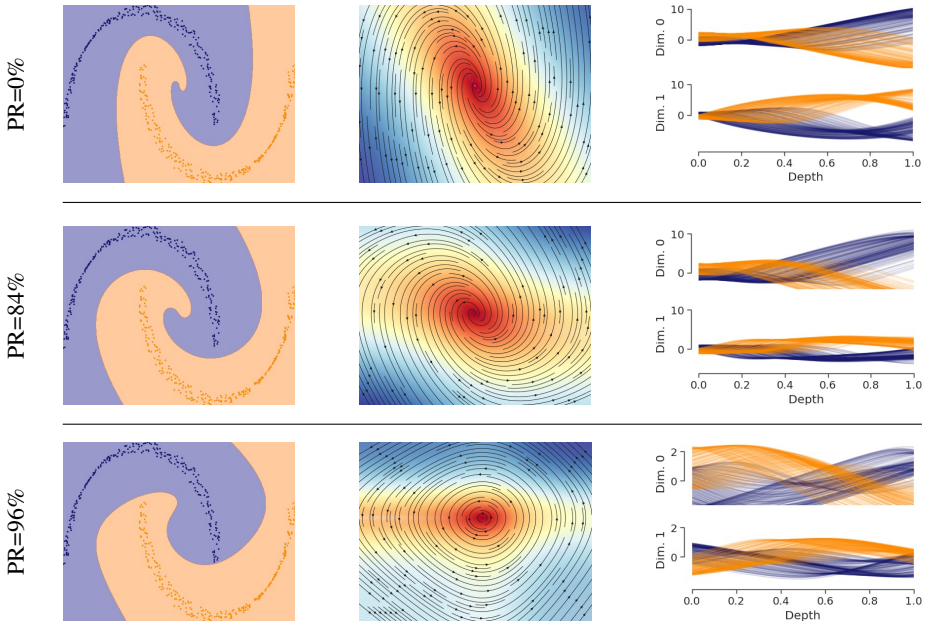


Figure 9: Robustness of decision boundaries for pruned networks. Column 1 is the decision boundary. Column 2 = state-space, and column 3 = the flow of data points.

5 Discussions, Scope and Conclusions

We showed the effectiveness of pruning for continuous neural networks. Pruning improved generalization performance of continuous normalizing flows in density estimation tasks at scale. Additionally, pruning allows us to obtain performant minimal network instances with at least one order of magnitude less parameter count. By providing key insights about how pruning improves generative modeling and inference, we enabled the design of better neural ODE instances.

Ensuring sparse Hessian computation for sparse continuous-depth models. As we prune neural ODE instances, their weight matrices will contain zero entries. However the Hessian with respect to those zero entries is not necessarily zero. Therefore, when we compute the eigenvalues of the Hessian, we must ensure to make the decompositon vector sets the Hessian of the pruned weights to zero before performing our eigenanalysis.

What are the limitations of Sparse Flows? Similar to any ODE-based learning system, the computational efficiency of sparse flows is highly determined by the choice of their ODE solvers, data and model parameters. As the complexity of any of these fronts increases, the number of function evaluations for a given task increases. Thus we might have a slow training process. This computational overhead can be relaxed in principle by the use of efficient ODE solvers (Poli et al., 2020b) and flow regularization schemes (Massaroli et al., 2020).

What design notes did we learn from applying pruning to neural ODEs? Our framework suggested that to obtain a generalizable sparse neural ODE representation, the choice of activation function is important. In particular activations that are Lipschitz continuous, monotonous, and bounded are better design choices for density estimation tasks. Moreover, we find that the generalizability of sparse neural ODEs is more influenced by their neural network's width than their depth (number of layers). Furthermore, pruning neural ODEs allows us to obtain better hyperparameters for the optimization problem by setting a trade-off between the value of the learning rate and weight decay.

In summary, we hope to have shown compelling evidence for the effectiveness of having sparsity in ODE-based flows.

References

Ryan P Adams, Jeffrey Pennington, Matthew J Johnson, Jamie Smith, Yaniv Ovadia, Brian Patton, and James Saunderson. Estimating the spectral density of large implicit matrices. *arXiv preprint arXiv:1802.03451*, 2018.

Cenk Baykal, Lucas Liebenwein, Igor Gilitschenski, Dan Feldman, and Daniela Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJfwJ2A5KX.

Rianne van den Berg, Leonard Hasenclever, Jakub M Tomczak, and Max Welling. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*, 2018.

Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2008. URL https://proceedings.neurips.cc/paper/2007/file/0d3180d672e08b4c5312dcdafdf6ef36-Paper.pdf.

Ricky T. Q. Chen, Jens Behrmann, David K Duvenaud, and Joern-Henrik Jacobsen. Residual flows for invertible generative modeling. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/5d0d5594d24f0f955548f0fc0ff83d10-Paper.pdf.

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.

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

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. In *Advances in Neural Information Processing Systems*, pages 3134–3144, 2019.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. *arXiv preprint arXiv:1906.04032*, 2019.

N. Benjamin Erichson, Omri Azencot, Alejandro Queiruga, Liam Hodgkinson, and Michael W. Mahoney. Lipschitz recurrent neural networks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=-N7PBXqOUJZ.

Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M Oberman. How to train your neural ode. *arXiv preprint arXiv:2002.02798*, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 881–889, Lille, France, 07–09 Jul 2015. PMLR. URL http://proceedings.mlr.press/v37/germain15.html.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. In *International Conference on Machine Learning*, pages 2232–2241. PMLR, 2019.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *International Conference on Learning Representations*, 2019.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015a. URL http://arxiv.org/abs/1510.00149.

Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 1*, pages 1135–1143, 2015b.

Babak Hassibi and David G Stork. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

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

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. Neural autoregressive flows. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2078–2087. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/huang18d.html.

Chin-Wei Huang, Ricky TQ Chen, Christos Tsirigotis, and Aaron Courville. Convex potential flows: Universal probability distributions with optimal transport and convex optimization. *arXiv preprint arXiv:2012.05942*, 2020.

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

Priyank Jaini, Kira A Selby, and Yaoliang Yu. Sum-of-squares polynomial flow. In *International Conference on Machine Learning*, pages 3009–3018. PMLR, 2019.

Nitish Shirish Keskar, Jorge Nocedal, Ping Tak Peter Tang, Dheevatsa Mudigere, and Mikhail Smelyanskiy. On large-batch training for deep learning: Generalization gap and sharp minima. In *5th International Conference on Learning Representations, ICLR 2017*, 2017.

Diederik P Kingma and Prafulla Dhariwal. Glow: generative flow with invertible $1\times$ 1 convolutions. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 10236–10245, 2018.

Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improving variational inference with inverse autoregressive flow. *arXiv preprint arXiv:1606.04934*, 2016.

Zhifeng Kong and Kamalika Chaudhuri. The expressive power of a class of normalizing flow models. In *International Conference on Artificial Intelligence and Statistics*, pages 3599–3609. PMLR, 2020.

Mathias Lechner, Ramin Hasani, Alexander Amini, Thomas A Henzinger, Daniela Rus, and Radu Grosu. Neural circuit policies enabling auditable autonomy. *Nature Machine Intelligence*, 2(10): 642–652, 2020.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics*, pages 3870–3882. PMLR, 2020.

Lucas Liebenwein, Cenk Baykal, Harry Lang, Dan Feldman, and Daniela Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJxkOlSYDH.

Lucas Liebenwein, Cenk Baykal, Brandon Carter, David Gifford, and Daniela Rus. Lost in pruning: The effects of pruning neural networks beyond test accuracy. *Proceedings of Machine Learning and Systems*, 3, 2021.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

Alaa Maalouf, Harry Lang, Daniela Rus, and Dan Feldman. Deep learning meets projective clustering. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=EQfpYwF3-b.

Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asma. Dissecting neural odes. In *34th Conference on Neural Information Processing Systems, NeurIPS 2020*. The Neural Information Processing Systems, 2020.

P Molchanov, S Tyree, T Karras, T Aila, and J Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations, ICLR 2017-Conference Track Proceedings*, 2019.

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

Junier Oliva, Avinava Dubey, Manzil Zaheer, Barnabas Poczos, Ruslan Salakhutdinov, Eric Xing, and Jeff Schneider. Transformation autoregressive networks. In *International Conference on Machine Learning*, pages 3898–3907. PMLR, 2018a.

Junier Oliva, Avinava Dubey, Manzil Zaheer, Barnabas Poczos, Ruslan Salakhutdinov, Eric Xing, and Jeff Schneider. Transformation autoregressive networks. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3898–3907. PMLR, 10–15 Jul 2018b. URL http://proceedings.mlr.press/v80/oliva18a.html.

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

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

Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Torchdyn: A neural differential equations library. *arXiv preprint arXiv:2009.09346*, 2020a.

Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, Jinkyoo Park, et al. Hypersolvers: Toward fast continuous-depth models. *Advances in Neural Information Processing Systems*, 33, 2020b.

Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. Routledge, 2018.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing fine-tuning and rewinding in neural network pruning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1gSj0NKvB.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538. PMLR, 2015.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

Akash Srivastava, Lazar Valkov, Chris Russell, Michael U. Gutmann, and Charles Sutton. Veegan: Reducing mode collapse in gans using implicit variational learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/44a2e0804995faf8d2e3b084a1e2db1d-Paper.pdf.

Takeshi Teshima, Isao Ishikawa, Koichi Tojo, Kenta Oono, Masahiro Ikeda, and Masashi Sugiyama. Coupling-based invertible neural networks are universal diffeomorphism approximators. *arXiv preprint arXiv:2006.11469*, 2020.

Antoine Wehenkel and Gilles Louppe. Unconstrained monotonic neural networks. *arXiv preprint arXiv:1908.05164*, 2019.

Liu Yang and George Em Karniadakis. Potential flow generator with l2 optimal transport regularity for generative models. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5687–5695, 2017.

Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W Mahoney. Pyhessian: Neural networks through the lens of the hessian. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 581–590. IEEE, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]
5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]