The Journal of Open Source Software

InvertibleNetworks.jl: A Julia package for scalable normalizing flows

Rafael Orozco[1], Philipp Witte[4], Mathias Louboutin[2], Ali Siahkoohi[3], Gabrio Rizzuti[6], Bas Peters[5], and Felix J. Herrmann[1]

1 Georgia Institute of Technology (GT), USA **2** Main contribution done at GT now at Devito Codes, UK **3** Main contribution done at GT now at Rice University, USA **4** Main contribution done at GT now at Microsoft Research, USA **5** Main contribution done at GT now at Computational Geosciences Inc, Canada **6** Main contribution done at GT now at Shearwater GeoServices, UK

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 30 November 2023
Published: unpublished

Summary

Normalizing flows is a density estimation method that provides efficient exact likelihood estimation and sampling (Dinh et al., 2014) from high dimensional distributions. This method depends on the use of the change of variables formula which requires an invertible transform. Thus normalizing flow architectures are built to be invertible by design (Dinh et al., 2014). In theory, the invertibility of architectures constrains the expressiveness but the use of coupling layers allows normalizing flows to exploit the power of arbitrary neural networks that need not be invertible (Dinh et al., 2016) and layer invertibility means if properly implemented many layers can be stacked to increase expressiveness without creating a training memory bottleneck.

The package we present, InvertibleNetworks.jl, is a pure Julia (Bezanson et al., 2012) implementation of normalizing flows. We have implemented many relevant neural network layers, including GLOW 1x1 invertible convolutions (Kingma & Dhariwal, 2018), affine/additive coupling layers (Dinh et al., 2014), Haar wavelet multiscale transforms (Haar, 1909) and Hierarchical invertible neural transport (HINT) (Kruse et al., 2021) among others. These modular layers are easily composed and modified to create different types of normalizing flows. As starting points, we have implemented RealNVP, GLOW, HINT, Hyperbolic networks (Lensink et al., 2022) and their conditional counterparts for users to quickly implement their individual applications.

Statement of need

This software package focuses on memory efficiency. The promise of neural networks is in learning high-dimensional distributions from examples thus normalizing flow packages should allow easy application to large dimensional inputs such as images or 3D volumes. Interestingly, the invertibility of normalizing flows naturally alleviates memory concerns since intermediate networks activations can be recomputed instead of saved in memory, greatly reducing the memory needed during backpropagation. The problem is that directly implementing normalizing flows in automatic differentiation frameworks such as PyTorch (Paszke et al., 2017) will not automatically exploit this invertibility. The available packages for normalizing flows such as: nflows (Durkan et al., 2020), normflows (Stimper et al., 2023) and FrEIA (Ardizzone et al., 2018-2022) are built depending on automatic differentiation frameworks thus do not exploit invertibility for memory efficiently.

Memory efficiency

By implementing gradients by hand instead of depending completely on automatic differentiation, our layers are capable of scaling to large inputs. By scaling, we mean that these codes are not prone to out-of-memory errors when training on GPU accelerators. Indeed, previous literature has described memory problems when using normalizing flows as their invertibility requires the latent code to maintain the same dimensionality as the input (Khorashadizadeh et al., 2023).

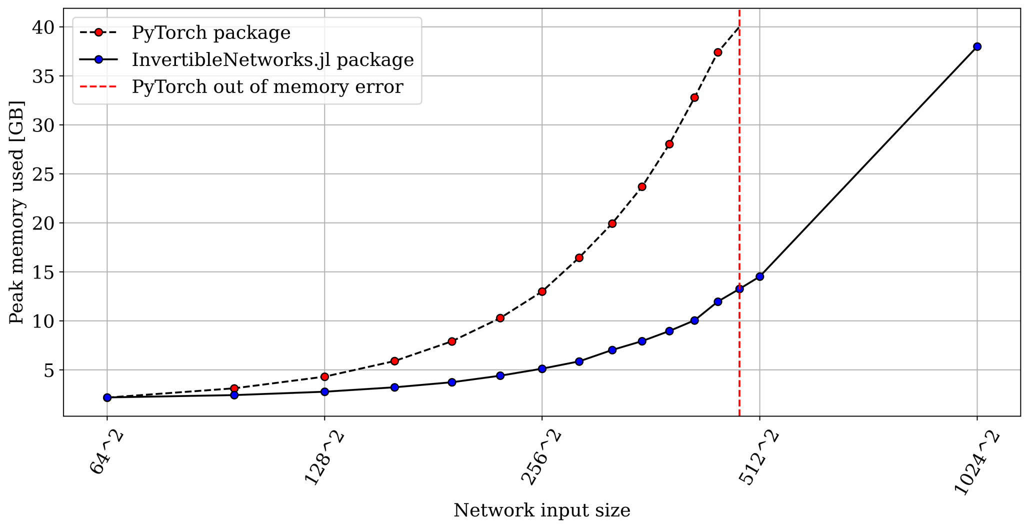

Figure 1: Our package InvertibleNetworks.jl provides memory frugal implementations of normalizing flows. Here, we compare our implementation of GLOW with an equivalent implementation in a PyTorch package. Using a 40GB A100 GPU, the PyTorch package can not train on image sizes larger than 480x480, while our package can handle sizes larger than 1024x1024.

In Figure 1, we show the relation between input size and the memory required for a gradient calculation in a PyTorch normalizing flow package (normflows (Stimper et al., 2023)) as compared to our package. The two tests were run with identical normalizing flow architectures. We note that the PyTorch implementation quickly increases the memory load and throws an out of memory error on the 40GB A100 GPU at the spatial image size of 480x480 while our InvertibleNetworks.jl implementation still has not run out of memory at spatial size 1024x1024. Note that this is in the context of a typical learning routine, so the images include 3 channels (RGB) and we used a batchsize of 8.

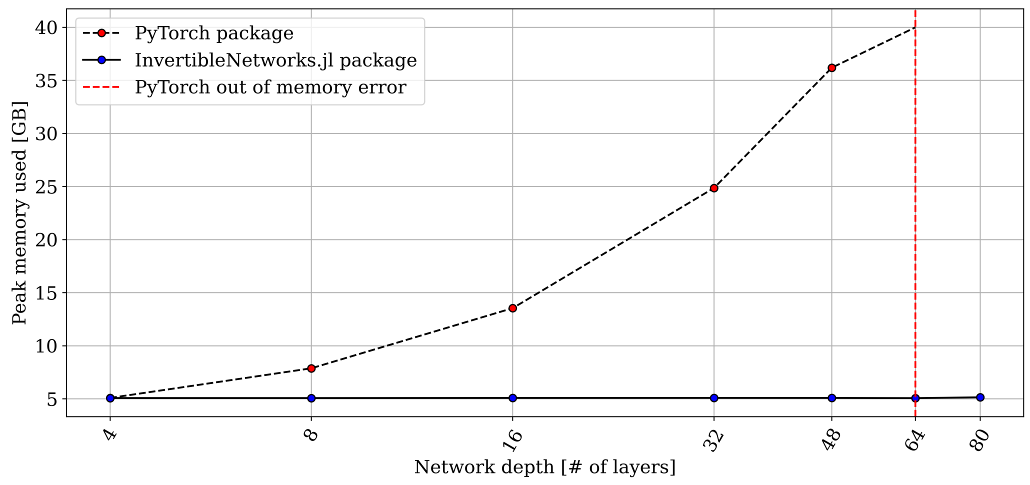

Figure 2: Due to the invertibility of the normalizing flow architecture, the memory consumption does not increase as we increase the depth of the network. Our package properly exploits this property thus shows constant memory consumption whereas the PyTorch package does not.

Since traditional normalizing flow architectures need to be invertible these might be less expressive as compared to non-invertible counterparts. In order to increase their expressiveness, practitioners stack many invertible layers to increase the overall expressive power. Increasing the depth of a neural network would in most cases increase the memory consumption of the network but in this case since normalizing flows are invertible, the memory consumption does not increase. Our package displays this phenomena as shown in Figure 2 while the PyTorch (normflows) package that has been implemented with automatic differentiation does not display this constant memory phenomena.

Ease of use

Although the normalizing flow layers gradients are hand-written, the package is fully compatibly with ChainRules (White et al., 2023) in order to integrate with automatic differentiation frameworks in Julia such as Zygote (Innes et al., 2019). This integration allows users to add arbitrary neural networks which will be differentiated by automatic differentiation while the memory bottleneck created by normalizing flow gradients will be dealt with InvertibleNetworks.jl. The typical use case for this combination is the summary networks used in amortized variational inference such as BayesFlow (Radev et al., 2020) which has been implemented in our package.

All implemented layers are tested for invertibility and correctness of their gradients with continuous integration testing via GitHub actions. There are many example for layers, networks and application workflows allowing new users to quickly build networks for a variety of applications. The ease of use is demonstrated by the publications that made use of the package.

Many publications have used InvertibleNetworks.jl for diverse applications including: change point detection, (Peters, 2022), acoustic data denoising (Kumar et al., 2021), seismic imaging (Louboutin et al., 2023; Rizzuti et al., 2020; Siahkoohi et al., 2021, 2022, 2023), fluid flow dynamics (Yin et al., 2023), medical imaging (Orozco, Siahkoohi, Rizzuti, et al., 2023; Orozco et al., 2021; Orozco, Louboutin, et al., 2023; Orozco, Siahkoohi, Louboutin, et al., 2023) and monitoring CO2 for combating climate change (Gahlot et al., 2023).

Future work

The neural network primitives (convolutions, non-linearities, pooling etc) are implemented in NNlib.jl abstractions thus support for AMD, Intel and Apple GPU can be trivially extended. Also, while our package currently can handle 3D inputs and has been used on large volume-based medical imaging (Orozco et al., 2022) there are interesting avenues of research regarding the "channel explosion" seen in invertible down and upsampling used in invertible networks (Peters et al., 2019).

References

Ardizzone, L., Bungert, T., Draxler, F., Köthe, U., Kruse, J., Schmier, R., & Sorrenson, P. (2018-2022). *Framework for Easily Invertible Architectures (FrEIA)*. https://github.com/vislearn/FrEIA

Bezanson, J., Karpinski, S., Shah, V. B., & Edelman, A. (2012). Julia: A fast dynamic language for technical computing. *arXiv Preprint arXiv:1209.5145*.

Dinh, L., Krueger, D., & Bengio, Y. (2014). Nice: Non-linear independent components estimation. *arXiv Preprint arXiv:1410.8516*.

Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2016). Density estimation using real nvp. *arXiv Preprint arXiv:1605.08803*.

Durkan, C., Bekasov, A., Murray, I., & Papamakarios, G. (2020). *nflows: Normalizing flows in PyTorch* (Version v0.14). Zenodo. https://doi.org/10.5281/zenodo.4296287

Gahlot, A. P., Erdinc, H. T., Orozco, R., Yin, Z., & Herrmann, F. J. (2023). Inference of CO2 flow patterns–a feasibility study. *arXiv Preprint arXiv:2311.00290*.

Haar, A. (1909). *Zur theorie der orthogonalen funktionensysteme*. Georg-August-Universitat, Gottingen.

Innes, M., Edelman, A., Fischer, K., Rackauckas, C., Saba, E., Shah, V. B., & Tebbutt, W. (2019). A differentiable programming system to bridge machine learning and scientific computing. *arXiv Preprint arXiv:1907.07587*.

Khorashadizadeh, A., Kothari, K., Salsi, L., Harandi, A. A., Hoop, M. de, & Dokmanić, I. (2023). Conditional injective flows for bayesian imaging. *IEEE Transactions on Computational Imaging*, *9*, 224–237.

Kingma, D. P., & Dhariwal, P. (2018). Glow: Generative flow with invertible 1x1 convolutions. *Advances in Neural Information Processing Systems*, *31*.

Kruse, J., Detommaso, G., Köthe, U., & Scheichl, R. (2021). HINT: Hierarchical invertible neural transport for density estimation and bayesian inference. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*, 8191–8199.

Kumar, R., Kotsi, M., Siahkoohi, A., & Malcolm, A. (2021). Enabling uncertainty quantification for seismic data preprocessing using normalizing flows (NF)—an interpolation example. *First International Meeting for Applied Geoscience & Energy*, 1515–1519.

Lensink, K., Peters, B., & Haber, E. (2022). Fully hyperbolic convolutional neural networks. *Research in the Mathematical Sciences*, *9*(4), 60.

Louboutin, M., Yin, Z., Orozco, R., Grady, T. J., Siahkoohi, A., Rizzuti, G., Witte, P. A., Møyner, O., Gorman, G. J., & Herrmann, F. J. (2023). Learned multiphysics inversion with differentiable programming and machine learning. *The Leading Edge*, *42*(7), 474–486.

Orozco, R., Louboutin, M., & Herrmann, F. J. (2022). Memory efficient invertible neural networks for 3D photoacoustic imaging. *arXiv Preprint arXiv:2204.11850*.

Orozco, R., Louboutin, M., Siahkoohi, A., Rizzuti, G., Leeuwen, T. van, & Herrmann, F. (2023). Amortized normalizing flows for transcranial ultrasound with uncertainty quantification. *arXiv Preprint arXiv:2303.03478*.

Orozco, R., Siahkoohi, A., Louboutin, M., & Herrmann, F. J. (2023). Refining amortized posterior approximations using gradient-based summary statistics. *arXiv Preprint arXiv:2305.08733*.

Orozco, R., Siahkoohi, A., Rizzuti, G., Leeuwen, T. van, & Herrmann, F. J. (2023). Adjoint operators enable fast and amortized machine learning based bayesian uncertainty quantification. *Medical Imaging 2023: Image Processing*, *12464*, 357–367.

Orozco, R., Siahkoohi, A., Rizzuti, G., Leeuwen, T. van, & Herrmann, F. J. (2021). Photoacoustic imaging with conditional priors from normalizing flows. *NeurIPS 2021 Workshop on Deep Learning and Inverse Problems*.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). *Automatic differentiation in pytorch*.

Peters, B. (2022). Point-TO-SET DISTANCE FUNCTIONS FOR OUTPUT-CONSTRAINED NEURAL NETWORKS. *Journal of Applied & Numerical Optimization*, *4*(2).

Peters, B., Haber, E., & Lensink, K. (2019). Symmetric block-low-rank layers for fully reversible multilevel neural networks. *arXiv Preprint arXiv:1912.12137*.

Radev, S. T., Mertens, U. K., Voss, A., Ardizzone, L., & Köthe, U. (2020). BayesFlow: Learning complex stochastic models with invertible neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *33*(4), 1452–1466.

Rizzuti, G., Siahkoohi, A., Witte, P. A., & Herrmann, F. J. (2020). Parameterizing uncertainty by deep invertible networks: An application to reservoir characterization. *SEG International Exposition and Annual Meeting*, D031S057R006.

Siahkoohi, A., Orozco, R., Rizzuti, G., & Herrmann, F. J. (2022). Wave-equation-based inversion with amortized variational bayesian inference. *arXiv Preprint arXiv:2203.15881*.

Siahkoohi, A., Rizzuti, G., Louboutin, M., Witte, P. A., & Herrmann, F. J. (2021). Preconditioned training of normalizing flows for variational inference in inverse problems. *arXiv Preprint arXiv:2101.03709*.

Siahkoohi, A., Rizzuti, G., Orozco, R., & Herrmann, F. J. (2023). Reliable amortized variational inference with physics-based latent distribution correction. *Geophysics*, *88*(3), R297–R322.

Stimper, V., Liu, D., Campbell, A., Berenz, V., Ryll, L., Schölkopf, B., & Hernández-Lobato, J. M. (2023). Normflows: A PyTorch package for normalizing flows. *arXiv Preprint arXiv:2302.12014*.

White, F., Abbott, M., Zgubic, M., Revels, J., Axen, S., Arslan, A., Schaub, S., Robinson, N., Ma, Y., Sam, Dhingra, G., Tebbutt, W., Widmann, D., Heim, N., Schmitz, N., Rackauckas, C., Lucibello, C., Fischer, K., Heintzmann, R., … Wennberg, D. (2023). *JuliaDiff/ChainRules.jl: v1.58.0* (Version v1.58.0). Zenodo. https://doi.org/10.5281/zenodo.10100624

Yin, Z., Orozco, R., Louboutin, M., & Herrmann, F. J. (2023). Solving multiphysics-based inverse problems with learned surrogates and constraints. *Advanced Modeling and Simulation in Engineering Sciences*, *10*(1), 14.