Preserving Properties of Neural Networks by Perturbative Updates

Andreas Krämer, Jonas Köhler, Frank Noé[*]
Freie Universität Berlin
Department of Mathematics and Computer Science
Germany
`{andreas.kraemer, jonas.koehler, frank.noe}@fu-berlin.de`

Abstract

Deep learning applications in physics usually require neural network architectures that obey certain symmetries and equivariances. Retaining such mathematical properties during training with stochastic gradient-based optimizers is a challenging task. To this end, we present a novel, general approach to preserve network properties by using parameterized perturbations. In lieu of directly optimizing the network parameters, the introduced P^4 *update* optimizes perturbations and merges them into the actual parameters infrequently such that the desired property is preserved. As a demonstration, we use this concept to preserve invertibility of linear layers during training. This P^4*Inv update* allows keeping track of inverses and determinants using rank-one updates. We show how such invertible blocks improve mode separation when applied to normalizing flows and Boltzmann generators.

1 Introduction

Many deep learning applications depend critically on the neural network parameters having a certain mathematical structure. As an important example, reversible generative models rely on invertibility and, in the case of normalizing flows [28], efficient computation of the Jacobian determinant [21]. Other models require (or benefit from) orthogonal linear layers or bounded Lipschitz constants. Finally, applications in physics often rely on networks that obey the relevant physical invariances and equivariances (e.g. [13, 2, 12, 11, 23, 24]).

Preserving parameter properties during training can be challenging and many approaches are currently in use. The most basic way of incorporating constraints is by network design. Many examples could be listed, like defining convolutional layers to obtain equivariances, constraining network outputs to certain intervals through bounded activation functions, Householder flows [29] to enforce layer-wise orthogonality, or coupling layers [5, 6] that enforce tractable inversion through their two-channel structure. A second approach concerns the optimizers used for training. Optimization routines have been tailored for example to enforce Lipschitz bounds [30] or efficiently optimize orthogonal linear layers [3].

The present work introduces a novel algorithmic concept for training neural networks in a property-preserving manner, see Figure 1. In lieu of directly changing the network parameters, the optimizer operates on perturbations to these parameters. The actual network parameters are frozen, while a parameterized perturbation serves as a proxy for optimization. Inputs are passed through the perturbed network during training. In regular intervals, the perturbed parameters are merged into the actual network.

[*]also at Rice University, Dept. of Chemistry, Houston, TX 77005, USA, and FU Berlin, Dept. of Physics

This stepwise reparameterization trick has several advantages for optimizing neural networks under constraints. First, it is usually easier to generate new sets of parameters that obey the constraints through suitable perturbations than from scratch. Therefore, we will refer to these updates as *property-preserving parameter perturbations*, or *P^4 updates*. Second, the merging step can occur infrequently (e.g. every 100 iterations) and perform the heavy-lifting so that the perturbations occurring in every step are kept computationally efficient. Specifically, the constraints need not be rigorously obeyed in every optimization step. Rather, numerical inaccuracies can be corrected before merging, which avoids propagation of errors into the actual (frozen) network parameters. Third, the perturbative update does not affect the network structure or computational cost outside of training. Finally, the general method can be used to retain desirable properties of either individual layers or the deep neural network as a whole.

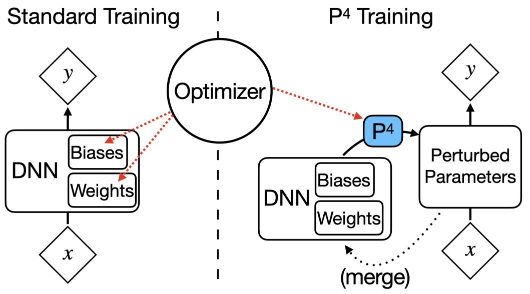

Figure 1: Training of deep neural networks (DNN). Standard DNN transform inputs x into outputs y through activation functions and linear layers, which are tuned by an optimizer. In contrast, P^4 training operates on perturbations to the parameters. Those are defined to retain certain network properties. The perturbed parameters are merged in regular intervals.

To demonstrate these benefits in a practical example, we efficiently train invertible linear layers while keeping track of their inverses and determinants. Previous work [7, 10, 19, 18, 29, 17, 22, 8] has mostly focused on orthogonal matrices, which can be trivially inverted and have unity determinant. Only most recently, Gresele et al. [9] presented a first method to optimize general invertible matrices implicitly using relative gradients, thereby providing greater flexibility and expressivity.

The novel P^{4}Inv scheme provides an alternative approach to train arbitrary invertible matrices $\boldsymbol{A} \in \mathrm{GL}(n)$. Notably, it avoids any explicit computations of inverses or determinants. We show how such invertible blocks can be utilized in normalizing flows by combining them with nonlinear, bijective activation functions or with coupling layers. The resulting neural networks are validated as normalizing flows [28, 27, 21] for density estimation and as deep generative models within Boltzmann generators.

2 Related Work

Constrained matrices can be optimized using Riemannian gradient descent on the manifold [1]. A reparameterization trick for general Lie groups has been introduced in [7]. For the unitary/orthogonal group there are multiple more specialized approaches, including using the Cayley transform [10], Householder Reflections [19, 18, 29], Givens rotations [25, 22] or the exponential map [17, 8].

Lezcano-Casado [15] recently introduced the concept of dynamic trivializations. This method performs training on manifolds by combining ideas from Riemannian gradient descent and trivializations (parameterizations of the manifold via an unconstrained space). Dynamic trivializations were derived in the general settings of Riemannian exponential maps and Lie groups. Convergence results were recently proven in follow-up work [16]. P^4 training resembles dynamic trivializations in that both perform a number of iteration steps in a fixed basis and infrequently lift the optimization problem to a new basis. In contrast to dynamic trivializations, the P^4 method allows leaving the set of feasible parameters and reenter it in a dedicated merging step. For example, the rank-one updates used in P^{4}Inv layers do not strictly parameterize $\mathrm{GL}(n)$ but instead can access all of $\mathbb{R}^{n\times n}$. This introduces the need for numerical stabilization, but enables efficient computation of the inverse and determinant during training.

3 P^4 Updates: Preserving Properties through Perturbations

3.1 General Concept

A deep neural network is a parameterized function $M_{\mathbf{A}} : \mathbb{R}^n \to \mathbb{R}^m$ with a high-dimensional parameter tensor $\mathbf{A}$. Now, let $\mathbb{S}$ define the subset of feasible parameter tensors so that the network satisfies a certain desirable property. In many situations, generating elements of $\mathbb{S}$ from scratch is much harder than transforming any $\mathbf{A} \in \mathbb{S}$ into other elements $\mathbf{A}' \in \mathbb{S}$, i.e. to move within $\mathbb{S}$.

The efficiency of perturbative updates can be leveraged as an incremental approach to retain certain desirable properties of the network parameters during training. Rather than optimizing the parameter tensors directly, we instead use a transformation $R_{\mathbf{B}} : \mathbb{S} \to \mathbb{S}$, which we call a *property-preserving parameter perturbation* (P^4). A P^4 transforms a given parameter tensor $\mathbf{A} \in \mathbb{S}$ into another tensor with the desired property $\mathbf{A}' \in \mathbb{S}$. The P^4 itself is also parameterized, by a tensor $\mathbf{B}$.

During training, the network is evaluated using the perturbed parameters $\tilde{\mathbf{A}} = R_{\mathbf{B}}(\mathbf{A})$. The parameter tensor of the perturbation, $\mathbf{B}$, is trainable via gradient-based stochastic optimizers, while the actual parameters $\mathbf{A}$ are frozen. In regular intervals, every N iterations of the optimizer, the optimized parameters of the P^4, $\mathbf{B}$, are merged into $\mathbf{A}$ as follows:

$$\mathbf{A}_{\text{new}} \leftarrow R_{\mathbf{B}}(\mathbf{A}), \tag{1}$$

$$\mathbf{B}_{\text{new}} \leftarrow \mathbf{B}_0. \tag{2}$$

Here, $\mathbf{B}_0$ is a tensor that produces the identity, $R_{\mathbf{B}_0} = \mathrm{id}_{\mathbb{S}}$, so that this update does not modify the effective (perturbed) parameters of the network $\tilde{\mathbf{A}}$ and hence enables a steady, iterative transformation.

3.2 P^{4}Inv: Invertible Linear Layers via Rank-One Updates

The P^4 algorithm can in principle be applied to properties concerning either individual blocks or the whole network. Here we train individual invertible linear layers (P^{4}Inv). To this end, we define the rank-one perturbation

$$R_{\boldsymbol{u},\boldsymbol{v}}(\boldsymbol{A}) : \boldsymbol{A} \mapsto \boldsymbol{A} + \boldsymbol{u}\boldsymbol{v}^T.$$

The inverse and determinant of rank-one perturbed matrices are given through the Sherman-Morrison formula

$$(\boldsymbol{A} + \boldsymbol{u}\boldsymbol{v}^T)^{-1} = \boldsymbol{A}^{-1} - \frac{1}{1 + \boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u}}\boldsymbol{A}^{-1}\boldsymbol{u}\boldsymbol{v}^T\boldsymbol{A}^{-1} \tag{3}$$

and the matrix determinant lemma

$$\det(\boldsymbol{A} + \boldsymbol{u}\boldsymbol{v}^T) = (1 + \boldsymbol{v}^T\boldsymbol{A}^{-1}\boldsymbol{u})\det(\boldsymbol{A}). \tag{4}$$

We define $\mathbb{S}$ as the set of invertible matrices, for which we know the inverse and determinant. Due to the above equations, the rank-one update is a P^4 on $\mathbb{S}$. The perturbation is reset by setting $\boldsymbol{u}$ to zero and reinitializing $\boldsymbol{v}$ from Gaussian noise. The inverse matrix and determinant are stored in the P^4 layer alongside $\boldsymbol{A}$ and updated according to equations 3 and 4.

3.3 Numerical Stabilization

The update to the inverse and determinant can become ill-conditioned if the denominator in equation 3 is close to zero. To tame potential numerical issues, the following additional strategies are applied. Firstly, merges are skipped whenever the determinant update falls out of predefined bounds. This allows the optimization to continue without propagating numerical errors into the actual weight matrix $\boldsymbol{A}$. Note that numerical errors in the perturbed parameters $\tilde{\boldsymbol{A}}$ are instantaneous and vanish when the optimization leaves the ill-conditioned regime. Secondly, to maintain a small error of the inverse throughout training, the inverse is corrected after every 50-th merging step by one iteration of an iterative matrix inversion [26]. This operation is $O(n^3)$, yet highly parallel so that it did not significantly affect the total training time in the experiments considered below. The reasoning behind this correction is to equip the algorithm with a means to recover from any numerical inaccuracies that may propagate into the stored inverse via merge steps (equation 3). Whether such a correction is in fact needed depends on various factors, including the determinant bounds for accepting merge steps, the total number of merges, the floating point precision, and the problem at hand.

4 Experiments

4.1 2D Distributions

To access the effectiveness of P^{4}Inv layers in deep networks, density estimation of common 2D toy distributions was performed by stacking 200 $2{\times}2$ P^{4}Inv layers with bijective activation functions (Bent identities and their inverses). For comparison, a RealNVP (RNVP) [6] normalizing flow was constructed with the same number of tunable parameters as the P^{4}Inv flow.

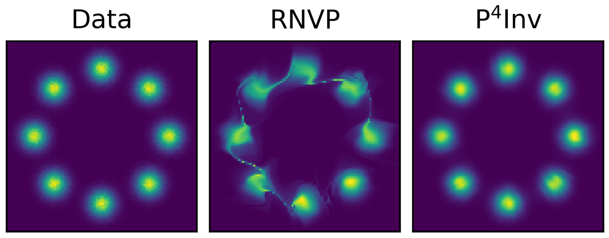

Figure 2: Density estimation for two-dimensional distributions from RealNVP (RNVP) and P^{4}Inv networks with similar numbers of tunable parameters.

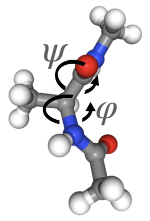

Figure 3: Alanine dipeptide with backbone torsions φ and ψ.

Figure 2 compares the generated distributions from the two models. The samples from the P^{4}Inv model aligned favorably with the ground truth. In contrast to RNVP, P^{4}Inv cleanly separated the modes. This underlines the favorable mixing achieved by general linear layers with elementwise nonlinear activations.

4.2 Boltzmann Generators of Alanine Dipeptide

Boltzmann generators [20] combine normalizing flows with statistical mechanics in order to draw direct samples from a given target density, e.g. given by a many-body physics system. This setup is ideally suited to assess the inversion of normalizing flows as its highly sensitive potential energy provides a quantitative measure for the sample quality. In molecular examples, specifically, the target densities are multimodal and contain singularities. Therefore, the generation of the 66-dimensional alanine dipeptide conformations (test case adopted from [4]) is a highly nontrivial test for generative models, see Figure 3.

The training efficiency and expressiveness of Boltzmann Generators were compared between pure RNVP baseline models as used in [20] and models augmented by P^{4}Inv. Concretely, P^{4}Inv layers were inserted between any RNVP layers replacing the usual swapping of input channels. Both flows consisted of 50 RNVP layers with 735,050 RNVP parameters. P^{4}Inv blocks only added 9,000 tunable parameters and neglibile additional cost. Both models were trained via density estimation (40000 steps on batches of 256 with a 10^{-3} learning rate) and subsequent energy-based training (2000 steps, batch size 4000, learning rate 10^{-5}).

Figure 4 (left) shows the energy statistics of generated samples. To demonstrate the sensitivity of the potential energy, the training data was first perturbed by 0.004 nm (less than 1% of the total length of the molecule) and energies were evaluated for the perturbed data set. As a consequence, the mean of the potential energy distribution increased by 13 $k_B T$.

In comparison, the Boltzmann generators produced much more accurate samples. The energy distributions from RNVP and P^{4}Inv blocks were only shifted upward by $\approx$ 2.6 $k_B T$ and rarely generated samples with infeasibly large energies. The performance of both models was comparable with slight advantages for models with P^{4}Inv swaps. This shows that the P^{4}Inv inverses remained intact during training. Finally, Figure 4 (right) shows the joint distribution of the two backbone torsions. Both Boltzmann generators reproduced the most important local minima of the potential energy. As in the 2D toy problems, the P^{4}Inv layers provided a cleaner separation of modes.

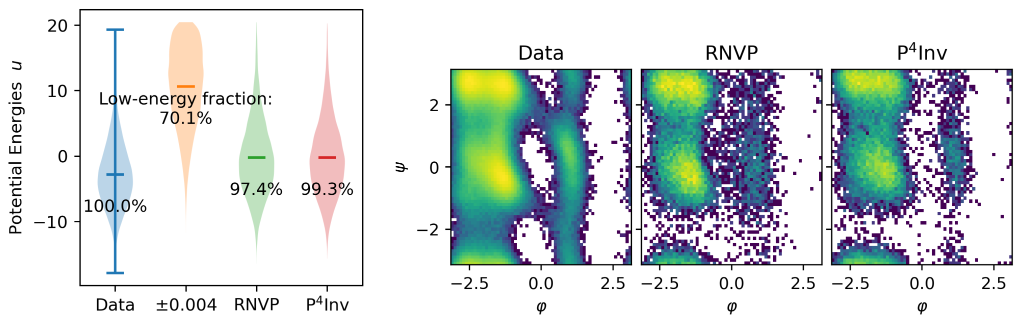

Figure 4: Left: Energy distributions of generated samples; the second (orange) violin plot shows energies when the training data was perturbed by normal distributed random noise with 0.004 nm standard deviation. The low-energy fraction for each column denotes the fraction of samples that had potential energy u lower than the maximum energy from the training set ($\approx 20\ k_BT$). Right: Joint marginal distribution of the backbone torsions φ and ψ: training data compared to samples from Boltzmann generators with and without P^4Inv swaps (denoted *P^4Inv* and *RNVP*, respectively).

5 Discussion

We have introduced P^4 updates, a novel algorithmic concept to preserve properties of neural networks using parameterized perturbations. As an example, invertible linear layers (P^4Inv) were trained with stochastic optimizers while efficiently keeping track of their inverses and determinants. A crucial aspect of the P^4 method is its decoupled merging step, which allows stable and efficient updates. As a consequence, the invertible linear P^4Inv layers can approximate any well-conditioned regular matrix.

The efficiency of P^4Inv training is studied in [14]. There, optimization of linear toy problems required a similar number of iterations as standard training of the full matrix despite the much smaller number of trainable parameters ($2n$ instead of n^2). In practical nonlinear situations, the convergence rate became slower than for standard training when the optimizer approached an optimum. On the contrary, P^4Inv layers provided a two orders-of-magnitude speedup of the inverse pass over alternative methods. Consequently, these layers are most useful when evaluating the loss function requires both the forward and inverse pass of the network.

Since perturbation theorems like the rank-one update exist for many classes of linear and nonlinear functions, we believe that the P^4 concept presents an efficient and widely applicable way of preserving desirable network properties during training.

Broader Impact

Machine learning applications in the physical sciences critically depend on neural networks that encode the relevant invariances and equivariances. To this end, the present work develops a new approach to retain mathematical properties of models during training. We demonstrate the method for invertibility of linear layers. Applications to other properties are straightforward if suitable perturbations can be defined. Since perturbation theorems are ubiquitous in both mathematics and physics, we expect that the P^4 approach will be useful in many other situations.

Acknowledgements

The authors acknowledge funding by the European Commission (ERC CoG 772230), German Ministry for Education and Research (Berlin Institute for the Foundations of Learning and Data BIFOLD), Deutsche Forschungsgemeinschaft (SFB1114/A04 and GRK DAEDALUS), and The Berlin Mathematics research center Math+ (projects AA1-6 and EF2-1).

References

[1] P. A. Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

[2] Denis Boyda, Gurtej Kanwar, Sébastien Racanière, Danilo Jimenez Rezende, Michael S Albergo, Kyle Cranmer, Daniel C Hackett, and Phiala E Shanahan. Sampling using $su(n)$ gauge equivariant flows. *arXiv preprint arXiv:2008.05456*, 2020.

[3] Krzysztof Choromanski, David Cheikhi, Jared Davis, Valerii Likhosherstov, Achille Nazaret, Achraf Bahamou, Xingyou Song, Mrugank Akarte, Jack Parker-Holder, Jacob Bergquist, Yuan Gao, Aldo Pacchiano, Tamas Sarlos, Adrian Weller, and Vikas Sindhwani. Stochastic flows and geometric optimization on the orthogonal group. In *37th International Conference on Machine Learning (ICML 2020)*, 2020.

[4] Manuel Dibak, Leon Klein, and Frank Noé. Temperature-steerable flows. *Machine Learning and the Physical Sciences, Workshop at the 34th Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[5] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

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

[7] Luca Falorsi, Pim de Haan, Tim R. Davidson, and Patrick Forré. Reparameterizing distributions on lie groups. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS) 2019*, volume 89 of *Proceedings of Machine Learning Research*, pages 3244–3253. PMLR, 2019.

[8] Adam Golinski, Mario Lezcano-Casado, and Tom Rainforth. Improving normalizing flows via better orthogonal parameterizations. In *ICML Workshop on Invertible Neural Networks and Normalizing Flows*, 2019.

[9] Luigi Gresele, Giancarlo Fissore, Adrián Javaloy, Bernhard Schölkopf, and Aapo Hyvärinen. Relative gradient optimization of the jacobian term in unsupervised deep learning. In *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, 2020.

[10] Kyle Helfrich, Devin Willmott, and Qiang Ye. Orthogonal recurrent neural networks with scaled cayley transform. In *International Conference on Machine Learning*, pages 1969–1978. PMLR, 2018.

[11] Jan Hermann, Zeno Schätzle, and Frank Noé. Deep-neural-network solution of the electronic Schrödinger equation. *Nat. Chem.*, 12(10):891–897, 2020.

[12] Gurtej Kanwar, Michael S. Albergo, Denis Boyda, Kyle Cranmer, Daniel C. Hackett, Sébastien Racanière, Danilo Jimenez Rezende, and Phiala E. Shanahan. Equivariant flow-based sampling for lattice gauge theory. *Phys. Rev. Lett.*, 125:121601, 2020.

[13] Jonas Köhler, Leon Klein, and Frank Noé. Equivariant flows: exact likelihood generative learning for symmetric densities. *arXiv preprint arXiv:2006.02425*, 2020.

[14] Andreas Krämer, Jonas Köhler, and Frank Noé. Training invertible linear layers through rank-one perturbations. *arXiv preprint arXiv:2010.07033*, 2020.

[15] Mario Lezcano-Casado. Trivializations for gradient-based optimization on manifolds. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, volume 32, pages 9157–9168, 2019.

[16] Mario Lezcano-Casado. Curvature-dependant global convergence rates for optimization on manifolds of bounded geometry. *arXiv preprint arXiv:2008.02517*, 2020.

[17] Mario Lezcano-Casado and David Martínez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 3794–3803. PMLR, 2019.

[18] Chenlin Meng, Yang Song, Jiaming Song, and Stefano Ermon. Gaussianization flows. *arXiv preprint arXiv:2003.01941*, 2020.

[19] Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In *International Conference on Machine Learning*, pages 2401–2409. PMLR, 2017.

[20] Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457):eaaw1147, 2019.

[21] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv preprint arXiv:1912.02762*, 2019.

[22] Tomas Pevny, Vasek Smidl, Martin Trapp, Ondrej Polacek, and Tomas Oberhuber. Sum-product-transform networks: Exploiting symmetries using invertible transformations. *arXiv preprint arXiv:2005.01297*, 2020.

[23] David Pfau, James S. Spencer, Alexander G. D. G. Matthews, and W. M. C. Foulkes. Ab initio solution of the many-electron schrödinger equation with deep neural networks. *Phys. Rev. Research*, 2:033429, 2020.

[24] Danilo Jimenez Rezende, Sébastien Racanière, Irina Higgins, and Peter Toth. Equivariant hamiltonian flows. *arXiv preprint arXiv:1909.13739*, 2019.

[25] Uri Shalit and Gal Chechik. Coordinate-descent for learning orthogonal matrices through Givens rotations. In *31st International Conference on Machine Learning (ICML 2014)*, volume 1, pages 833–845, 2014.

[26] Fazlollah Soleymani. A fast convergent iterative solver for approximate inverse of matrices. *Numerical Linear Algebra with Applications*, 21(3):439–452, 2014.

[27] Esteban G Tabak and Cristina V Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.

[28] Esteban G Tabak, Eric Vanden-Eijnden, et al. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217–233, 2010.

[29] Jakub M Tomczak and Max Welling. Improving variational auto-encoders using householder flow. *arXiv preprint arXiv:1611.09630*, 2016.

[30] Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning, 2017.