Enforcing Analytic Constraints in Neural Networks Emulating Physical Systems

Tom Beucler, Michael Pritchard, Stephan Rasp, Jordan Ott, Pierre Baldi, and Pierre Gentine

Phys. Rev. Lett. **126**, 098302 — Published 4 March 2021

DOI: 10.1103/PhysRevLett.126.098302

Enforcing Analytic Constraints in Neural-Networks Emulating Physical Systems

Tom Beucler,[1, 2, *] Michael Pritchard,[1] Stephan Rasp,[3] Jordan Ott, Pierre Baldi,[4] and Pierre Gentine[2]

[1] *Department of Earth System Science, University of California, Irvine, CA, USA*
[2] *Department of Earth and Environmental Engineering, Columbia University, New York, NY, USA*
[3] *Technical University of Munich, Munich, Germany*
[4] *Department of Computer Science, University of California, Irvine, CA, USA*

(Dated: February 2, 2021)

Neural networks can emulate nonlinear physical systems with high accuracy, yet they may produce physically-inconsistent results when violating fundamental constraints. Here, we introduce a systematic way of enforcing nonlinear analytic constraints in neural networks via constraints in the architecture or the loss function. Applied to convective processes for climate modeling, architectural constraints enforce conservation laws to within machine precision without degrading performance. Enforcing constraints also reduces errors in the subsets of the outputs most impacted by the constraints.

`Main Repository: https://github.com/raspstephan/CBRAIN-CAM`
`Figures and Tables: https://github.com/tbeucler/CBRAIN-CAM/blob/master/notebooks/tbeucler_devlog/042_Figures_PRL_Submission.ipynb`

I. INTRODUCTION

Many fields of science and engineering (e.g., fluid dynamics, hydrology, solid mechanics, chemistry kinetics) have exact, often *analytic*, closed-form constraints, i.e. constraints that can be explicitly written using analytic functions of the system's variables. Examples include translational or rotational invariance, conservation laws, or equations of state. While physically-consistent models should enforce constraints to within machine precision, data-driven algorithms often fail to satisfy well-known constraints that are not explicitly enforced. In particular, neural networks (NNs, [1]), powerful regression tools for nonlinear systems, may severely violate constraints on individual samples while optimizing overall performance.

Despite the need for physically-informed NNs for complex physical systems [2–5], enforcing *hard* constraints [6] has been limited to physical systems governed by specific equations, such as advection equations [7–9], Reynolds-averaged Navier-Stokes equations [10, 11], boundary conditions of idealized flows [12], or quasi-geostrophic equations [13]. To address this gap, we introduce a systematic method to enforce analytic constraints arising in more general physical systems to within machine precision, namely the Architecture-Constrained NN or ACnet. We then compare ACnets to unconstrained (UCnets) and loss-constrained NNs (LCnets, in which soft constraints are added through a penalization term in the loss function [e.g., 14–16]) in the particular case of climate modeling, where the system is high-dimensional and the constraints (such as mass and energy conservation) are few but crucial [17].

II. THEORY

A. Formulating the Constraints

Consider a NN mapping an input vector $\boldsymbol{x} \in \mathbb{R}^m$ to an output vector $\boldsymbol{y} \in \mathbb{R}^p$. Enforcing constraints is easiest for linearly-constrained NNs, i.e. NNs for which the constraints $(\mathcal{C})$ can be written as a linear system of rank n:

$$(\mathcal{C}) \overset{\text{def}}{=} \left\{ \boldsymbol{C} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} = \boldsymbol{0} \right\}. \tag{1}$$

We call $\boldsymbol{C} \in \mathbb{R}^n \times \mathbb{R}^{m+p}$ the constraints matrix, and use bold font for vectors and tensors to distinguish them from scalars. For the regression problem to have non-unique solutions, the number of independent constraints n has to be strictly less than $m + p$.

In Figure 1, we consider a generic regression problem subject to analytic constraints $(\mathcal{C})$ that may be nonlinear, and propose how to formulate a linearly-constrained NN. First, define the regression's inputs $\boldsymbol{x_0}$ and outputs $\boldsymbol{y_0}$, which respectively become the *temporary* NN's features and targets. Then (**Formulation 1**), write the constraints $(\mathcal{C})$ as an identically zero function $\boldsymbol{c}$ of the inputs, the outputs, and additional parameters $\boldsymbol{z}$ the constraints may involve. We recommend non-dimensionalizing all variables to facilitate the design, interpretation, and performance of the loss function. While the function $\boldsymbol{c}$ may be nonlinear, it can always be written as the sum of: (1) terms $\boldsymbol{x}$ that *only* depend on inputs and (2) terms $\boldsymbol{y}$ that depend on inputs, outputs and additional parameters. Thus the constraints can be written as:

$$\boldsymbol{c}\left(\boldsymbol{x_0}, \boldsymbol{y_0}, \boldsymbol{z}\right) = \boldsymbol{C} \begin{bmatrix} \boldsymbol{x}\left(\boldsymbol{x_0}\right) \\ \boldsymbol{y}\left(\boldsymbol{x_0}, \boldsymbol{y_0}, \boldsymbol{z}\right) \end{bmatrix}, \tag{2}$$

where $\boldsymbol{C}$ is a matrix. Finally (**Formulation 2**), choose $\boldsymbol{x}$ and $\boldsymbol{y}$ as the NN's new inputs and outputs. If $\boldsymbol{x}$ and $\boldsymbol{y}$ are not bijective functions of $(\boldsymbol{x_0}, \boldsymbol{y_0})$, add variables to the

[*] tom.beucler@gmail.com

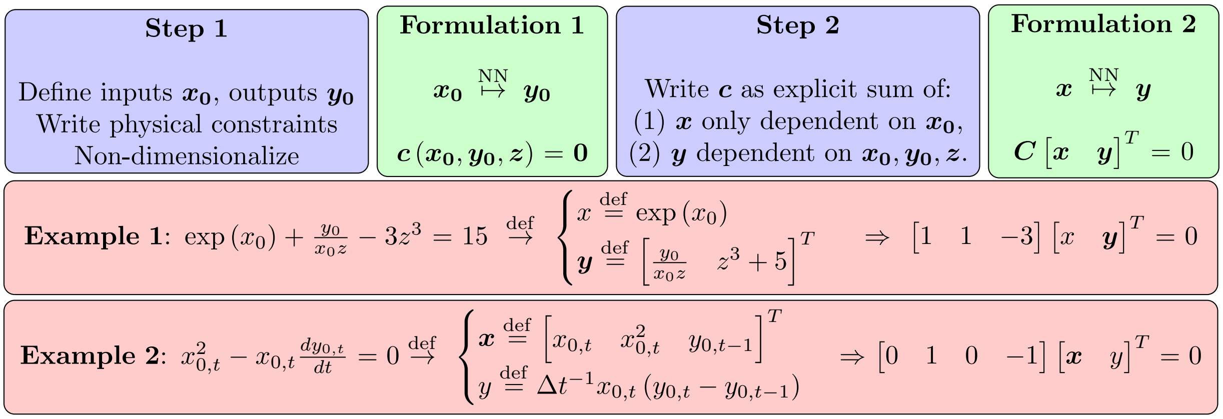
Step 1
Define inputs $\boldsymbol{x_0}$, outputs $\boldsymbol{y_0}$
Write physical constraints
Non-dimensionalize
Formulation 1
$\boldsymbol{x_0} \overset{\text{NN}}{\mapsto} \boldsymbol{y_0}$
$\boldsymbol{c}\left(\boldsymbol{x_0},\boldsymbol{y_0},\boldsymbol{z}\right)=\boldsymbol{0}$
Step 2
Write $\boldsymbol{c}$ as explicit sum of:
(1) $\boldsymbol{x}$ only dependent on $\boldsymbol{x_0}$,
(2) $\boldsymbol{y}$ dependent on $\boldsymbol{x_0},\boldsymbol{y_0},\boldsymbol{z}$.
Formulation 2
$\boldsymbol{x} \overset{\text{NN}}{\mapsto} \boldsymbol{y}$
$\boldsymbol{C}\begin{bmatrix}\boldsymbol{x} & \boldsymbol{y}\end{bmatrix}^T=0$
Example 1: $\exp\left(x_0\right)+\frac{y_0}{x_0 z}-3z^3=15 \overset{\text{def}}{\to} \begin{cases} x \overset{\text{def}}{=} \exp\left(x_0\right) \\ \boldsymbol{y} \overset{\text{def}}{=} \begin{bmatrix}\frac{y_0}{x_0 z} & z^3+5\end{bmatrix}^T \end{cases} \Rightarrow \begin{bmatrix}1 & 1 & -3\end{bmatrix}\begin{bmatrix}x & \boldsymbol{y}\end{bmatrix}^T=0$
Example 2: $x_{0,t}^2-x_{0,t}\frac{dy_{0,t}}{dt}=0 \overset{\text{def}}{\to} \begin{cases} \boldsymbol{x} \overset{\text{def}}{=} \begin{bmatrix}x_{0,t} & x_{0,t}^2 & y_{0,t-1}\end{bmatrix}^T \\ y \overset{\text{def}}{=} \Delta t^{-1}x_{0,t}\left(y_{0,t}-y_{0,t-1}\right) \end{cases} \Rightarrow \begin{bmatrix}0 & 1 & 0 & -1\end{bmatrix}\begin{bmatrix}\boldsymbol{x} & y\end{bmatrix}^T=0$

FIG. 1. Framework to treat constrained regression problems using linearly-constrained NNs, with two examples: (1) A regression problem with one nonlinear constraint, and (2) a time-prediction problem with one differential nonlinear constraint that we discretize using a forward Euler method of timestep Δt. Note that the choice of $\boldsymbol{x},\boldsymbol{y}$, and $\boldsymbol{C}$ is not unique.

NN's inputs and outputs to recover $\boldsymbol{x_0}$ and $\boldsymbol{y_0}$ after optimization (e.g., we add $x_{0,t}$ and $y_{0,t-1}$ to $\boldsymbol{x}$ in **Example 2**). We are now in a position to build a computationally-efficient NN that satisfies the linear constraints ($\mathcal{C}$).

B. Enforcing the Constraints

Consider a NN trained on preexisting measurements of $\boldsymbol{x}$ and $\boldsymbol{y}$. For simplicity's sake, we measure the quality of its output $\boldsymbol{y_{\text{NN}}}$ using a standard mean-squared error (MSE) misfit:

$$\text{MSE}\left(\boldsymbol{y_{\text{Truth}}},\boldsymbol{y_{\text{NN}}}\right) \overset{\text{def}}{=} \left\|\boldsymbol{y_{\text{Err}}}\right\|_2 \overset{\text{def}}{=} \frac{1}{p}\sum_{k=1}^{p} y_{\text{Err},k}^2, \quad (3)$$

where we have introduced the error vector, defined as the difference between the NN's output and the "truth":

$$\boldsymbol{y_{\text{Err}}} \overset{\text{def}}{=} \boldsymbol{y_{\text{NN}}}-\boldsymbol{y_{\text{Truth}}}. \quad (4)$$

In the reference case of an "unconstrained network" (UCnet), we optimize a multi-layer perceptron [e.g., 18, 19] using MSE as its loss function $\mathcal{L}$. To enforce the constraints ($\mathcal{C}$) within NNs, we consider two options:

(1) Constraining the loss function (LCnet, soft constraints): We first test a *soft* penalization of the NN for violating physical constraints using a penalty $\mathcal{P}$, defined as the mean-squared residual from the constraints:

$$\begin{aligned}\mathcal{P}\left(\boldsymbol{x},\boldsymbol{y_{\text{NN}}}\right) &\overset{\text{def}}{=} \left\|\boldsymbol{C}\begin{bmatrix}\boldsymbol{x} \\ \boldsymbol{y_{\text{NN}}}\end{bmatrix}\right\|_2, \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{m}C_{ij}x_j+\sum_{k=1}^{p}C_{i(k+m)}y_{\text{NN},k}\right)^2,\end{aligned} \quad (5)$$

and given a weight $\alpha\in[0,1]$ in the loss function $\mathcal{L}$:

$$\mathcal{L}\left(\alpha\right)=\alpha\mathcal{P}\left(\boldsymbol{x},\boldsymbol{y_{\text{NN}}}\right)+\left(1-\alpha\right)\text{MSE}\left(\boldsymbol{y_{\text{Truth}}},\boldsymbol{y_{\text{NN}}}\right). \quad (6)$$

(2) Constraining the architecture (ACnet, hard constraints): Alternatively, we treat the constraints as *hard* and augment a standard, optimizable NN with n fixed conservation layers that sequentially enforce the constraints ($\mathcal{C}$) to within machine precision (Figure 2), while keeping the MSE as the loss function:

$$(\text{ACnet}) \Rightarrow \left\{\min \text{MSE} \;\; \text{s.t.} \;\; \boldsymbol{C}\begin{bmatrix}\boldsymbol{x} & \boldsymbol{y_{\text{NN}}}\end{bmatrix}^T=\boldsymbol{0}\right\} \quad (7)$$

The optimizable NN calculates a "direct" output whose size is $p-n$. We then calculate the remaining output's components of size n as exact "residuals" from the constraints. Concatenating the "direct" and "residual" vectors results in the full output $\boldsymbol{y_{\text{NN}}}$ that satisfies the constraints to within machine precision. Since our loss uses the full output $\boldsymbol{y_{\text{NN}}}$, the gradients of the loss function are passed through the constraints layers during optimization, meaning that the final NN's weights and biases depend on the constraints ($\mathcal{C}$). ACnet improves upon the common approach of calculating "residual" outputs *after* training because ACnet exposes the NN to "residual" output data *during* training (SM C.3). A possible implementation of the constraints layer uses custom (Tensorflow in our case) layers with fixed parameters that solve the system of equations ($\mathcal{C}$) , in row-echelon form, from the bottom to the top row (SM B.1). Note that we are free to choose which outputs to calculate as "residuals", which introduces n new hyperparameters (SM B.2).

C. Linking Constraints to Performance

Intuitively, we might expect the NNs' performance to improve once we enforce constraints arising in physical systems with few degrees of freedom, but this may not hold true with many degrees of freedom. We formalize the link between constraints and performance by: (1) decomposing the NN's prediction into the "truth" and error vectors following equation 4; and (2) assuming that constraints exactly hold for the "truth" (no errors in measurement). This yields:

$$\boldsymbol{C}\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}_{\mathbf{NN}}\end{bmatrix}\overset{\mathrm{def}}{=}\overbrace{\boldsymbol{C}\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{y}_{\mathbf{Truth}}\end{bmatrix}}^{\mathbf{0}}+\boldsymbol{C}\begin{bmatrix}\mathbf{0}\\ \boldsymbol{y}_{\mathbf{Err}}\end{bmatrix}. \tag{8}$$

Equation 8 relates how much the constraints are violated to the error vector. More explicitly, if we measure performance using the MSE, we may square each component of Equation 8. The resulting equation links how much physical constraints are violated to the squared error for each constraint of index $i \in [\![1,n]\!]$:

$$\underbrace{\left(\boldsymbol{C}\begin{bmatrix}x\\ y_{\mathrm{NN}}\end{bmatrix}\right)_i^2}_{\text{Physical constraints}}=\underbrace{\sum_{k=1}^{p}C_{i(k+m)}^2 y_{\mathrm{Err},k}^2}_{\text{Squared}-\text{error}>0}+\underbrace{\sum_{k=1}^{p}\sum_{l\neq k}C_{i(k+m)}C_{i(l+m)}y_{\mathrm{Err},k}y_{\mathrm{Err},l}}_{\text{Cross}-\text{term}} \tag{9}$$

In ACnets, we strictly enforce physical constraints, setting the left-hand side of Equation 9 to 0, within numerical errors. As the squared error is positive-definite, the cross-term is always negative in ACnets as both terms sum up to 0. It is difficult to predict the cross-term before optimization, hence Equation 9 does not provide a-priori predictions of performance, even for ACnets. Instead, it links how much the NN violates constraints to how well it predicts outputs that appear in the constraints equations: the more negative the cross-term, the larger the squared error for a given violation of physical constraints.

III. APPLICATION

A. Convective Parameterization for Climate Modeling

The representation of subgrid-scale processes in coarse-scale, numerical models of the atmosphere, referred to as subgrid *parameterization*, is a large source of error and uncertainty in numerical weather and climate prediction [e.g., 20, 21]. Machine-learning algorithms trained on fine-scale, process-resolving models can improve subgrid parameterizations by faithfully emulating the effect of fine-scale processes on coarse-scale dynamics [e.g., 22–25, see Section 2 of Rasp [26] for a detailed review]. The problem is that none of these parameterizations exactly follow conservation laws (e.g., conservation of mass, energy). This is critical for long-term climate projections, as the spurious energy production may both exceed the projected radiative forcing from greenhouse gases and result in large thermodynamic drifts or biases over a long time-period. Motivated by this shortcoming, we build a NN parameterization of convection and clouds that we *constrain* to conserve 4 quantities: column-integrated energy, mass, longwave radiation, and shortwave radiation.

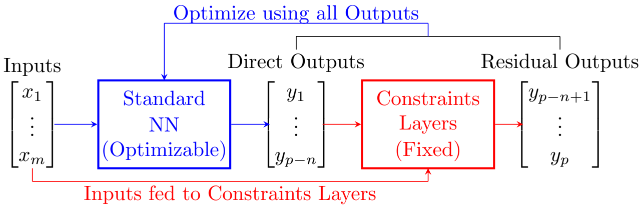
Optimize using all Outputs
Inputs
x_1
x_m
Standard NN (Optimizable)
Direct Outputs
y_1
y_{p-n}
Constraints Layers (Fixed)
Residual Outputs
y_{p-n+1}
y_p
Inputs fed to Constraints Layers

FIG. 2. ACnet: Direct outputs are calculated using a standard NN, while the remaining outputs are calculated as residuals from the fixed constraints layers.

B. Model and Data

We use the Super-Parameterized Community Atmosphere Model 3.0 [27] to simulate the climate for two years in aquaplanet configuration [28], where the surface temperatures are fixed with a realistic equator-to-pole gradient [29]. Following [24]'s sensitivity tests, we use 42M samples from the simulation's first year to train the NN (training set) and 42M samples from the simulation's second year to validate the NN (validation set). Since we use the validation set to adjust the NN's hyperparameters and avoid overfitting, we additionally introduce a test set using 42M different samples from the simulation's second year to provide an unbiased estimator of the NNs' performances. Note that each sample represents a single atmospheric column at a given time, longitude, and latitude.

C. Formulating the Conservation Laws in a Neural Network

The parameterization's goal is to predict the rate at which sub-grid convection vertically redistributes heat and water based on the current large-scale thermodynamic state. We group all variables describing the local climate in an input vector $\boldsymbol{x}$ of size 304 (5 vertical profiles with 30 levels each, prescribed large-scale conditions $\mathbf{LS}$ for all profiles of size 150, and 4 scalars):

$$\boldsymbol{x}=\begin{bmatrix}(\boldsymbol{q_v},\boldsymbol{q_l},\boldsymbol{q_i},\boldsymbol{T},\boldsymbol{v},\mathbf{LS},p_s,S_0) & \mathrm{SHF} & \mathrm{LHF}\end{bmatrix}^T, \tag{10}$$

where all variables are defined in SM A. We then concatenate the time-tendencies from convection and the additional variables involved in the conservation laws to form an output vector $\boldsymbol{y}$ of size 216 (7 vertical profiles with 30 levels, followed by 6 scalars):

$$\boldsymbol{y} = \left[\; \boldsymbol{\dot{q}_v} \;\; \boldsymbol{\dot{q}_l} \;\; \boldsymbol{\dot{q}_i} \;\; \boldsymbol{\dot{T}} \;\; \boldsymbol{\dot{T}_{KE}} \;\; \mathbf{lw} \;\; \mathbf{sw} \;\; \mathrm{LW_t} \;\; \mathrm{LW_s} \;\; \mathrm{SW_t} \;\; \mathrm{SW_s} \;\; P \;\; P_i \;\right]^T, \tag{11}$$

We normalize all variables to the same units before non-dimensionalizing them using the constant 1W m^{-2} (SM A.5). Finally, we derive the dimensionless conservation laws (SM A.1-A.4) and write them as a sparse matrix of size $4 \times (304 + 218)$:

$$\boldsymbol{C} = \begin{bmatrix} \mathbf{0} & 1 & \ell_s & -\ell_s\boldsymbol{\delta p} & -\ell_f\boldsymbol{\delta p} & \mathbf{0} & -\boldsymbol{\delta p} & \boldsymbol{\delta p} & \mathbf{0} & \mathbf{0} & -1 & 1 & 1 & -1 & -\ell_f & \ell_f \\ \mathbf{0} & 0 & 1 & -\boldsymbol{\delta p} & -\boldsymbol{\delta p} & -\boldsymbol{\delta p} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 0 & 0 & 0 & -1 & 0 \\ \mathbf{0} & 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{\delta p} & \mathbf{0} & 1 & -1 & 0 & 0 & 0 & 0 \\ \mathbf{0} & 0 & 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{\delta p} & 0 & 0 & -1 & 1 & 0 & 0 \end{bmatrix}, \tag{12}$$

that acts on $\boldsymbol{x}$ and $\boldsymbol{y}$ to yield Equation 1.

Each row of the constraints matrix $\boldsymbol{C}$ describes a different conservation law: The first row is column-integrated enthalpy conservation (here equivalent to energy conservation), the second row is column-integrated water conservation (here equivalent to mass conservation), the third row is column-integrated longwave radiation conservation and the last row is column-integrated shortwave radiation conservation.

D. Implementation

We implement the three NN types and a multi-linear regression baseline using the Tensorflow library [30] version 1.13 with Keras [31] version 2.2.4: (1) *LCnets* for which we vary the weight α given to conservation laws from 0 to 1 (Equation 6), (2) our reference *ACnet*, and (3) *UCnet*, i.e. an unconstrained LCnet of weight $\alpha = 0$. In our reference ACnet, we write the constraints layers in Tensorflow to solve the system of equations ($\mathcal{C}$) from bottom to top, and calculate surface tendencies as residuals of the conservation equations (SM B.1); switching the "residual" outputs to different vertical levels does not significantly change the validation loss nor the constraints penalty (SM B.3). After testing multiple architectures and activation functions (SM C.2), we chose 5 hidden layers of 512 nodes with leaky rectified linear-unit activations as our standard multi-layer perceptron architecture, resulting in ~1.3M trainable parameters. We optimized the NN's weights and biases with the RMSprop optimizer [32] for LCnets (because it was more stable than the Adam optimizer [33]), used Sherpa for hyperparameter optimizations [34], and saved the NN's state of minimal validation loss over 20 epochs.

E. Results

In Figure 3a, we compare mean performance (measured by MSE) and by how much physical constraints are violated (measured by $\mathcal{P}$) for the three NN types. As expected, we note a monotonic trade-off between performance and constraints as we increase α from 0 to 1 in the loss function. This trade-off is well-measured by MSE and $\mathcal{P}$ across the training, validation, and test sets (SM Table V). Interestingly, the physical constraints are easier to satisfy than reducing MSE in our case, likely because it is difficult to deterministically predict precipitation, which is strongly non-Gaussian, inherently stochastic, and whose error contributes to a large portion of MSE. Despite this, UCnet may violate physical constraints more than our multi-linear regression baseline.

Our first key result is that *ACnet performs nearly as well as our lowest-MSE UCnet on average* (to within 3%) *while satisfying constraints* to $\sim \left(10^{-9}\%\right)$ (SM C.1). This result holds across the training, validation and test sets (SM Table IV). In our case, ACnets perform slightly less well than UCnet because they are harder to optimize and the "residual" outputs exhibit systematically larger errors (SM B.2). This systematic, unphysical bias can be remedied by multiplying the weights of these "residual" outputs in the loss function (SM B.3) by a factor $\beta > 1$ (SM Equation 12 and SM Figure 2). β can be objectively chosen alongside the "residual" outputs via formal hyperparameter optimization (SM C.2).

In Figure 3b, we compare how much the NNs violate column energy conservation (RESID) to the prediction of a variable that appears in that constraint: the total thermodynamic tendency in the enthalpy conservation equation (THERMO):

$$\overbrace{\left(\boldsymbol{C}\begin{bmatrix} x \\ y_{\mathrm{NN}} \end{bmatrix}\right)_1}^{\text{RESID}} = \overbrace{\boldsymbol{\delta p}\cdot\left(\boldsymbol{\dot{T}_{\mathrm{KE}}} - \boldsymbol{\dot{T}} - \boldsymbol{\ell_s}\boldsymbol{\dot{q}_v} - \boldsymbol{\ell_f}\boldsymbol{\dot{q}_l}\right)}^{\text{THERMO}} + ..., \tag{13}$$

where the ellipsis includes the surface fluxes, radiation, and precipitation terms. ACnet predicts THERMO more accurately than all NNs (full blue line) by an amount closely related to how much each NN violates enthalpy consevation (dashed lines), followed by LCnet (full green line). This yields our second key result: *Enforcing con-*

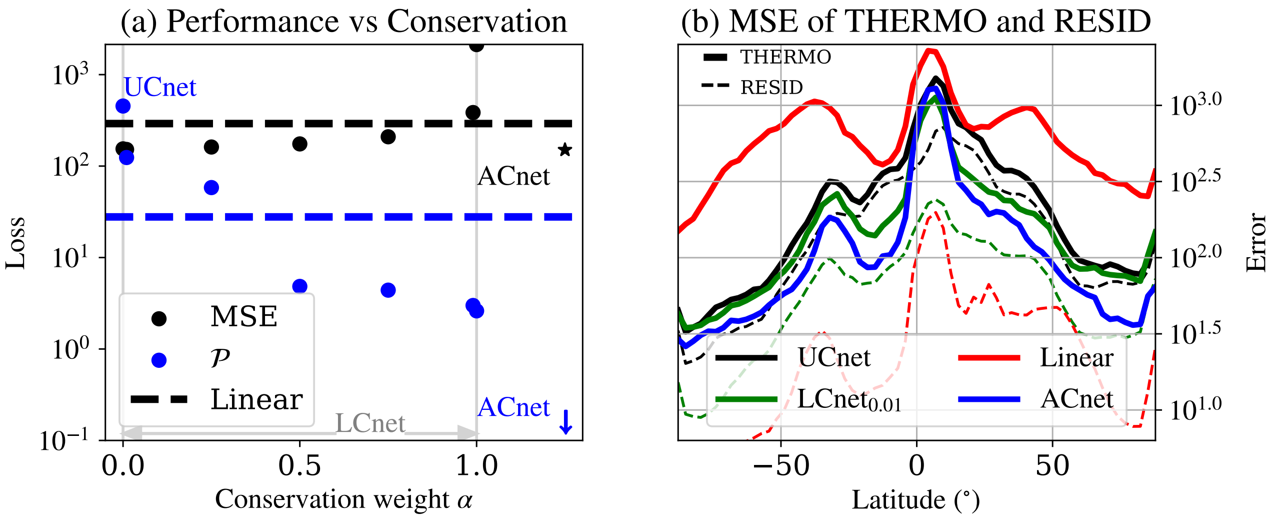

FIG. 3. (a) MSE and $\mathcal{P}$ averaged over all samples of the test dataset for UCnet, LCnets of varying α, and ACnet. The dashed lines indicate MSE and $\mathcal{P}$ for our multi-linear regression baseline. (b) Mean-squared error in the thermodynamic term (THERMO) and the enthalpy residual (RESID) versus latitude for our lowest-MSE NN in each category.

straints, whether in the architecture or the loss function, can systematically reduce the error of variables that appear in the constraints. This result holds true across the training, validation, and test sets (SM Figure 4). However, possibly since our case has many degrees of freedom, it does not hold true for individual components of THERMO as their cross-term in Equation 9 is more negative for ACnet, nor does it hold for variables that are hard to predict deterministically (e.g., precipitation). Additionally, obeying conservation laws does not guarantee the ability to generalize well far outside of the training set, e.g. in the Tropics of a warmer climate (see Figure 3 of [35]). These results nuance the finding that physically constraining NNs systematically improves their generalization ability, which has been documented for machine learning emulation of low-dimensional idealized flows [5, 12], and motivate physically-constraining machine-learning algorithms capable of stochastic predictions [36] that are consistent across climates [35].

Finally, although the mapping presented in Section III has linear constraints, ACnets can also be applied to nonlinearly constrained mappings by using the framework presented in Figure 1. We give a concrete example in SM D, where we introduce the concept of “conversion layers” that transform nonlinearly constrained mappings into linearly-constrained mappings within NNs and without overly degrading performance (SM Table IX). Additionally, ACnets can be extended to incorporate inequality constraints on their “direct” outputs (by using positive-definite activation functions, discussed in SM E), making ACnets applicable to a broad range of constrained optimization problems.

ACKNOWLEDGMENTS

TB is supported by NSF grants OAC-1835769, OAC-1835863, and AGS-1734164. PG acknowledges support from USMILE ERC synergy grant. The work of JO and PB is in part supported by grants NSF 1839429 and NSF NRT 1633631 to PB. We thank Eric Christiansen, Imme Ebert-Uphoff, Bart Van Merrienboer, Tristan Abbott, Ankitesh Gupta, and Derek Chang for advice. We also thank the meteorology department of LMU Munich and the Extreme Science and Engineering Discovery Environment supported by NSF grant number ACI-1548562 (charge numbers TG-ATM190002 and TG-ATM170029) for computational resources. The Supplemental Material (SM) can be found at [INSERT URL] and includes references [13, 23, 37–43].

[1] P. Baldi, *Deep Learning in Science: Theory, Algorithms, and Applications* (Cambridge University Press, Cambridge, UK, 2021) in press.

[2] M. Reichstein, G. Camps-Valls, B. Stevens, M. Jung, J. Denzler, N. Carvalhais, and Prabhat, Deep learning and process understanding for data-driven Earth system science, Nature **566**, 195 (2019).

[3] K. J. Bergen, P. A. Johnson, M. V. De Hoop, and G. C. Beroza, Machine learning for data-driven discovery in solid Earth geoscience (2019).

[4] A. Karpatne, G. Atluri, J. H. Faghmous, M. Steinbach, A. Banerjee, A. Ganguly, S. Shekhar, N. Samatova, and V. Kumar, Theory-guided data science: A new paradigm for scientific discovery from data, IEEE Transactions on

Knowledge and Data Engineering **29**, 2318 (2017).

[5] J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar, Integrating Physics-Based Modeling with Machine Learning: A Survey, (2020), arXiv:2003.04919.

[6] P. Márquez-Neila, M. Salzmann, and P. Fua, Imposing Hard Constraints on Deep Networks: Promises and Limitations, (2017), arXiv:1706.02025.

[7] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations, (2017), arXiv:1711.10561.

[8] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, Learning data-driven discretizations for partial differential equations, Proceedings of the National Academy of Sciences **116**, 15344 (2019).

[9] E. de Bezenac, A. Pajot, and P. Gallinari, Deep Learning for Physical Processes: Incorporating Prior Scientific Knowledge, (2017), arXiv:1711.07970.

[10] J. Ling, A. Kurzawski, and J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, Journal of Fluid Mechanics **807**, 155 (2016).

[11] J. L. Wu, H. Xiao, and E. Paterson, Physics-informed machine learning approach for augmenting turbulence models: A comprehensive framework, Physical Review Fluids **7**, 074602 (2018).

[12] L. Sun, H. Gao, S. Pan, and J. X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Computer Methods in Applied Mechanics and Engineering **361**, 112732 (2020).

[13] T. Bolton and L. Zanna, Applications of Deep Learning to Ocean Data Inference and Subgrid Parameterization, Journal of Advances in Modeling Earth Systems **11**, 376 (2019).

[14] A. Karpatne, W. Watkins, J. Read, and V. Kumar, Physics-guided Neural Networks (PGNN): An Application in Lake Temperature Modeling, (2017), arXiv:1710.11431.

[15] X. Jia, J. Willard, A. Karpatne, J. Read, J. Zwart, M. Steinbach, and V. Kumar, Physics guided RNNs for modeling dynamical systems: A case study in simulating lake temperature profiles, in *SIAM International Conference on Data Mining, SDM 2019* (2019) pp. 558–566, arXiv:1810.13075v2.

[16] M. Raissi, A. Yazdani, and G. E. Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, Science **367**, 1026 (2020).

[17] T. Beucler, S. Rasp, M. Pritchard, and P. Gentine, Achieving Conservation of Energy in Neural Network Emulators for Climate Modeling, (2019), arXiv:1906.06622.

[18] A. K. Jain, J. Mao, and K. M. Mohiuddin, Artificial neural networks: A tutorial (1996).

[19] M. W. Gardner and S. R. Dorling, Artificial neural networks (the multilayer perceptron) - a review of applications in the atmospheric sciences, Atmospheric Environment **32**, 2627 (1998).

[20] T. Palmer, G. Shutts, R. Hagedorn, F. Doblas-Reyes, T. Jung, and M. Leutbecher, Representing Model Uncertainty in Weather and Climate Prediction, Annual Review of Earth and Planetary Sciences **33**, 163 (2005).

[21] T. Schneider, J. Teixeira, C. S. Bretherton, F. Brient, K. G. Pressel, C. Schär, and A. P. Siebesma, Climate goals and computing the future of clouds, Nature Climate Change **7**, 3 (2017).

[22] V. M. Krasnopolsky, M. S. Fox-Rabinovitz, and A. A. Belochitski, Using Ensemble of Neural Networks to Learn Stochastic Convection Parameterizations for Climate and Numerical Weather Prediction Models from Data Simulated by a Cloud Resolving Model, Advances in Artificial Neural Systems **2013**, 1 (2013).

[23] P. Gentine, M. Pritchard, S. Rasp, G. Reinaudi, and G. Yacalis, Could Machine Learning Break the Convection Parameterization Deadlock?, Geophysical Research Letters **45**, 5742 (2018).

[24] S. Rasp, M. S. Pritchard, and P. Gentine, Deep learning to represent sub-grid processes in climate models, Proceedings of the National Academy of Sciences of the United States of America **115**, 9684 (2018), arXiv:1806.04731.

[25] N. D. Brenowitz and C. S. Bretherton, Prognostic Validation of a Neural Network Unified Physics Parameterization, Geophysical Research Letters **45**, 6289 (2018).

[26] S. Rasp, Coupled online learning as a way to tackle instabilities and biases in neural network parameterizations 10.5194/gmd-2019-319 (2019), arXiv:1907.01351.

[27] M. Khairoutdinov, D. Randall, and C. DeMott, Simulations of the Atmospheric General Circulation Using a Cloud-Resolving Model as a Superparameterization of Physical Processes, Journal of the Atmospheric Sciences **62**, 2136 (2005).

[28] M. S. Pritchard, C. S. Bretherton, and C. A. Demott, Restricting 32-128 km horizontal scales hardly affects the MJO in the Superparameterized Community Atmosphere Model v.3.0 but the number of cloud-resolving grid columns constrains vertical mixing, Journal of Advances in Modeling Earth Systems **6**, 723 (2014).

[29] J. A. Andersen and Z. Kuang, Moist static energy budget of MJO-like disturbances in the atmosphere of a zonally symmetric aquaplanet, Journal of Climate **25**, 2782 (2012).

[30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, (2016), arXiv:1603.04467.

[31] F. Chollet, Keras (2015).

[32] T. Tieleman, G. E. Hinton, N. Srivastava, and K. Swersky, Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, COURSERA: Neural Networks for Machine Learning **4**, 26 (2012).

[33] D. P. Kingma and J. Ba, Adam: A Method for Stochastic Optimization, (2014), arXiv:1412.6980.

[34] L. Hertel, J. Collado, P. Sadowski, J. Ott, and P. Baldi, Sherpa: Robust hyperparameter optimization for machine learning, SoftwareX (2020), in press.

[35] T. Beucler, M. Pritchard, P. Gentine, and S. Rasp, Towards Physically-consistent, Data-driven Models of Convection, (2020), arXiv:2002.08525.

[36] J.-L. Wu, K. Kashinath, A. Albert, D. Chirila, Prabhat, and H. Xiao, Enforcing Statistical Constraints in Generative Adversarial Networks for Modeling Chaotic

Dynamical Systems, (2019), arXiv:1905.06841.

[37] M. F. Khairoutdinov and D. a. Randall, Cloud Resolving Modeling of the ARM Summer 1997 IOP: Model Formulation, Results, Uncertainties, and Sensitivities, Journal of the Atmospheric Sciences **60**, 607 (2003).

[38] W. D. Collins, P. J. Rasch, B. A. Boville, J. J. Hack, J. R. McCaa, D. L. Williamson, B. P. Briegleb, C. M. Bitz, S. J. Lin, and M. Zhang, The formulation and atmospheric simulation of the Community Atmosphere Model version 3 (CAM3), Journal of Climate **19**, 2144 (2006).

[39] J. Yuval and P. A. O'Gorman, Stable machine-learning parameterization of subgrid processes for climate modeling at a range of resolutions, Nature Communications **11**, 1 (2020).

[40] A. Krizhevsky and G. Hinton, Learning multiple layers of features from tiny images.(2009), Cs.Toronto.Edu (2009).

[41] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, ImageNet Large Scale Visual Recognition Challenge, International Journal of Computer Vision **115**, 211 (2015), arXiv:1409.0575.

[42] L. Hertel, P. Sadowski, J. Collado, and P. Baldi, Sherpa : Hyperparameter Optimization for Machine Learning Models, Conference on Neural Information Processing Systems (NIPS) (2018).

[43] J. Bergstra and Y. Bengio, Random search for hyperparameter optimization, Journal of Machine Learning Research **13**, 281 (2012).