A posteriori learning of quasi-geostrophic turbulence parametrization: an experiment on integration steps

Hugo Frezat
Univ. Grenoble Alpes, CNRS UMR LEGI, Grenoble, France
Univ. Grenoble Alpes, CNRS UMR IGE, Grenoble, France
IMT Atlantique, CNRS UMR Lab-STICC, Brest, France
`hugo.frezat@univ-grenoble-alpes.fr`

Julien Le Sommer
Univ. Grenoble Alpes, CNRS UMR IGE, Grenoble, France

Ronan Fablet
IMT Atlantique, CNRS UMR Lab-STICC, Brest, France

Guillaume Balarac
Univ. Grenoble Alpes, CNRS UMR LEGI, Grenoble, France
Institut Universitaire de France (IUF), Paris, France

Redouane Lguensat
Learning, Data and Robotics Lab, ESIEA, Paris, France
LOCEAN-IPSL, Sorbonne Université, Institut Pierre Simon Laplace, Paris, France

Abstract

Modeling the subgrid-scale dynamics of reduced models is a long standing open problem that finds application in ocean, atmosphere and climate predictions where direct numerical simulation (DNS) is impossible. While neural networks (NNs) have already been applied to a range of three-dimensional flows with success, two dimensional flows are more challenging because of the backscatter of energy from small to large scales. We show that learning a model jointly with the dynamical solver and a meaningful *a posteriori*-based loss function lead to stable and realistic simulations when applied to quasi-geostrophic turbulence.

1   Introduction

Understanding and predicting the evolution of various natural systems would not be possible without simulations of turbulent flows. However, solving all the spatial and temporal scales of the associated partial differential equation (PDE), i.e., the Navier-Stokes equations remains computationally prohibitive. One popular solution (e.g. [14, 8, 4]) is to resolve only the largest scales and use subgrid closures (or physical parametrizations) to represent the smaller ones.

Recently, neural networks (NNs) have been proposed as a promising alternative to algebraic parametrizations in three-dimensional incompressible turbulence [6, 15, 5]. Two-dimensional problems, however, are more challenging due to the inverse cascade of energy that leads to negative viscosities, and it has been demonstrated that numerical stability of the trained model in decaying turbulence requires either the removal of negative eddy viscosities [13] or a large training dataset [9].
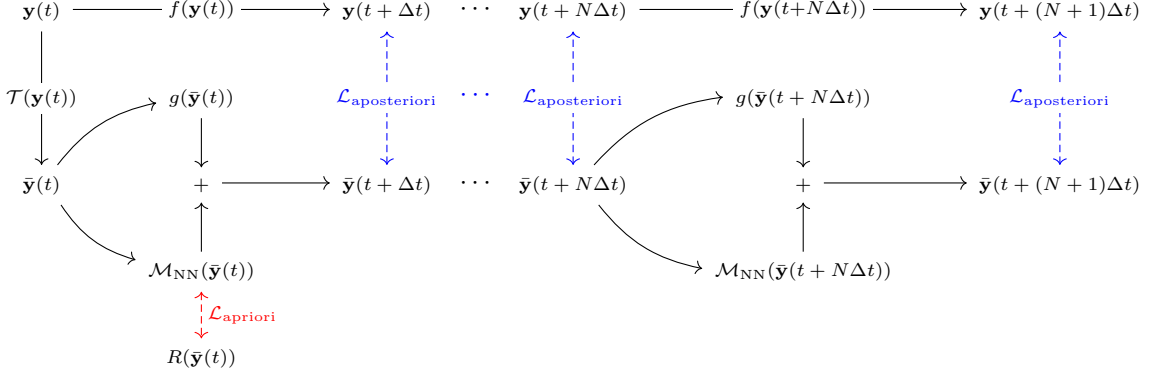
Figure 1: Sketch of one learning step for the presented strategies. The *a priori* loss is computed at instantaneous time t (dashed, red), while the *a posteriori* loss is evaluated on states forward in time (dashed, blue).

2D turbulence is also a simple model that well approximates more complex rotating stratified flows (found in atmosphere and ocean dynamics).

We show that on a forced configuration of 2D turbulence, stability can be obtained when the NN is trained end-to-end on a temporal horizon, i.e. jointly with the forward solver (see Fig. 1). At equivalent computational complexity, the standard learning strategy does not lead to a stable parametrization and is thus not usable in practice. Additionally, the presented strategy allows us to constrain the calibration of the model on *a posteriori* metrics, i.e. the evolution of quantities of interest over time, which are the absolute performance measures in turbulence modeling. Finally, we show that the number of iterations performed during the training process have a significant impact on the long-time statistics of the model.

2 A priori and a posteriori learning

In the machine learning community, it is well known that models can take advantage of additional informations during training. For instance, the basic, purely data-based approach has been extended to train models end-to-end using an entire system. In practice, some examples of those systems include autonomous vehicles [2], objects detection [19] and super-resolution [3]. This idea has also recently found many applications in the physical sciences community (e.g. [10, 11]).

In the context of partial differential equation parametrization, we are given with a high-resolution system $f(\mathbf{y})$ and a low resolution system $g(\bar{\mathbf{y}})$ describing the evolution of high and low resolution variables $\mathbf{y}(t)$ and $\bar{\mathbf{y}}(t)$ respectively by,

$$\begin{cases} \dfrac{\partial \mathbf{y}}{\partial t} = f(\mathbf{y}), & \mathbf{y} \in \Omega \\ \dfrac{\partial \bar{\mathbf{y}}}{\partial t} = g(\bar{\mathbf{y}}) + R(\bar{\mathbf{y}}), & \bar{\mathbf{y}} \in \bar{\Omega} \\ \mathcal{T}(\mathbf{y}) = \bar{\mathbf{y}} \end{cases} \quad (1)$$

where $\bar{\Omega}$ is a smaller domain than Ω and $\mathcal{T}$ is a known projection operator that goes from high to low resolution variables.

In this framework, we are interested in modeling the residual term, or parametrization $R(\bar{\mathbf{y}})$ with a neural network $\mathcal{M}_{\mathrm{NN}}(\bar{\mathbf{y}})$, based on high resolution data coming from a direct numerical simulation of $f(\mathbf{y})$, collected in a dataset

$$\mathcal{D} := \{\bar{\mathbf{y}}(t)\} \to \{R(\bar{\mathbf{y}}(t))\}. \quad (2)$$

If we connect the machine learning naming to fluid dynamics concepts, we could say that the data-based approach only gives the ability to optimize a model on *a priori* metrics, i.e., instantaneous predictions of the targeted term. With the end-to-end approach, however, we have access to simulation quantities over time, which can be used to train the model on *a posteriori* metrics (See [17] for

more details). Using the data $\mathcal{D}$, a standard learning strategy based on the direct prediction of the residual term can be written as $\arg\min_\theta \mathcal{L}(R(\bar{\mathbf{y}}), \mathcal{M}_{\mathrm{NN}}(\bar{\mathbf{y}}|\theta))$, which has already been applied to fluid dynamics parametrization, and corresponds to an *a priori* minimization problem. Now, if we have a differentiable solver for $g(\bar{\mathbf{y}})$, the standard minimization problem performed during learning can be rewritten as a series of *a posteriori* losses on low resolution variables that are advanced in time, such that

$$\arg\min_\theta \mathcal{L}(\mathbf{y}(t), \bar{\mathbf{y}}(t_0) + \Phi_\theta(\bar{\mathbf{y}}(t_0), t_0, t)), t \in [0, T] \tag{3}$$

where $\mathcal{L}$ is the loss function, $[0, T]$ is the temporal horizon on which the forward solver is integrated and Φ the flow operator that advances the reduced model in time, expressed as

$$\Phi_\theta(\bar{\mathbf{y}}, t_0, t) = \int_{t_0}^t g(\bar{\mathbf{y}}(\tau)) + \mathcal{M}_{\mathrm{NN}}(\bar{\mathbf{y}}(\tau)|\theta)\, \mathrm{d}\tau. \tag{4}$$

In practice, the forward solver performs N discrete timesteps, which defines the temporal horizon $T = N\Delta t$. A sketch of the *a posteriori* learning strategy is shown in Fig. 1.

3 Application to quasi-geostrophic turbulence parametrization

The described strategies are applied to a classical two-dimensional single-layer barotropic quasi-geostrophic (QG) potential vorticity equation [12], defined by

$$\partial_t \omega + J(\psi, \omega) = \nu\nabla^2\omega - \mu\omega + F \tag{5}$$

with

$$\mathbf{u} = (-\partial_y\psi, \partial_x\psi) \tag{6}$$
$$\omega = \nabla^2\psi \tag{7}$$

where ω is the vorticity, ψ the stream function, $\mathbf{u}$ the velocity vector and $J(\psi, \omega) = \partial_x\psi\partial_y\omega - \partial_y\psi\partial_x\omega$ is the Jacobian operator. The model is parametrized by a viscosity ν, a linear drag μ and a source term F.

Projecting to the reduced system. In turbulence modeling, a common approach is to suppose that the reduced system has been filtered so that the smallest scales are removed, and instead predicted by a model. In practice, the projection operator $\mathcal{T}$ is a spatial filter $\Lambda_\delta(x, y)$ at spatial scale $\delta > 0$ defined such that

$$\mathcal{T}(\omega) := \omega * \Lambda_\delta = \bar{\omega}(x, y, t) \tag{8}$$

and the reduced system becomes

$$\partial_t\bar{\omega} + J(\bar{\psi}, \bar{\omega}) = \nu\nabla^2\bar{\omega} - \mu\bar{\omega} + \bar{F} + \underbrace{J(\bar{\psi}, \bar{\omega}) - \overline{J(\psi, \omega)}}_{R(\psi, \omega)} \tag{9}$$

where $R(\psi, \omega)$ is the subgrid-scale (SGS) term or parametrization that needs to be modeled.

Configuration. In our experiment, we discretize the full solutions on 2048×2048 grid points using a pseudo-spectral method and use $\delta = 16$ with a spectral cut-off filter $\Lambda_\delta(k) = 0, \forall k > \pi/\delta\Delta$ where Δ is the filter width so that our reduced solution lies on a 128×128 grid. The source term F is a time-dependent wind-forcing acting at large scales $k = 4$,

$$F = C_F(t)\left[\cos(4y + \pi\sin(1.4t)) - \cos(4x + \pi\sin(1.5t))\right] \tag{10}$$

with steady enstrophy rate C_F such that $\langle F^2 \rangle/2 = 3$. Following the procedure given by [8], we spin-up multiple simulations with 1024×1024 grid initialized with random modes at $k = 4$ for approximately 1300 days. The simulations are then integrated for 35 days on the high-resolution grid to obtain a turbulent state that will be used as initial conditions for training and testing. Finally, training data is extracted from 10 independent ≈ 65 days long (48000 iterations) trajectories. The parameters of the simulations are summarized in dimensional and non-dimensional units in Table 3.

Our model [1] is implemented with PyTorch [16] in order to benefit from automatic differentiation, which is required when computing the gradient of the flow operator (7).

---

[1] `https://github.com/hrkz/torchqg`

Table 1: Parameters of the forced configuration in dimensional and non-dimensional units, with $L_d(x) = 1.2 \times 10^6 s$ and $T_d(x) = 504 \times 10^4/\pi m$.

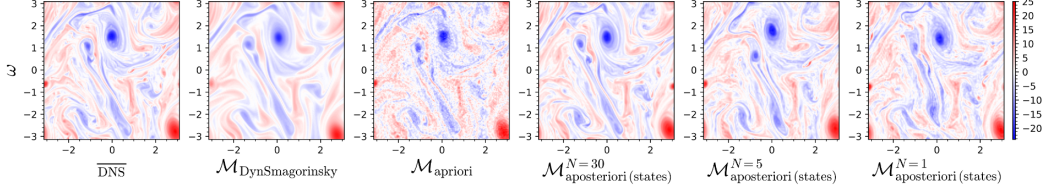| Parameter	Dimensional	Numerical
Domain length (L)	10^4 km	2π
Spatial resolution (Δx)	10 km	3.00×10^{-3}
Time step (Δt)	120 s	1.00×10^{-4}
Linear drag (μ)	1.25×10^{-8} m^{-1}	2.00×10^{-2}
Viscosity (ν)	22.0 m^2 s^{-1}	1.02×10^{-5}
Reynolds number (Re)	≈ 220000	

Figure 2: Vorticity fields at the end of the testing simulation with the filtered DNS (leftmost) and the discussed models. Note that the model trained with *a priori* learning was numerically unstable and its states diverged to infinity.

Learning components. The neural network $\mathcal{M}_{NN}$ used in this experiment for both *a priori* and *a posteriori* strategies is realized with a fully convolutional architecture. It consist of 10 convolutional layers with 64 features each, interspresed with ReLU activation functions using kernel sizes of 5×5. The parameters are optimized for a fixed number of epochs with Adam and a learning rate of 10^{-4}.

In particular, the *a posteriori* version is trained for a different number of iterations $N = \{1, 5, 30\}$, with a loss function that minimizes the mean squared error (MSE) of vorticity $\bar{\omega}$ over time, such that

$$\mathcal{L}_{\text{aposteriori}} := \frac{1}{N} \sum_{i=1}^{N} (\omega(i\Delta t) * \Lambda_\delta - \bar{\omega}(i\Delta t))^2 \tag{11}$$

Results. We run a new simulation at $\delta = 16$, i.e. on a 128×128 grid, comparing the NNs to the reference direct numerical simulation $\overline{\text{DNS}}$ and the dynamic Smagorinsky model [18, 7], a purely dissipative algebraic model which is a common baseline in the fluid dynamics community. The run is performed for 3000 and 48000 iterations of the low resolution and direct numerical simulation systems, respectively, since the timestep for low resolution systems can be δ times larger than the DNS Δt.

First, we can see in Fig. 2 the vorticity fields ω at the end of the new simulation. As expected, the dynamic Smagorinsky model is only dissipative and thus the smallest scales of the simulation are not visible anymore. On the other side, the model learnt from the *a priori* strategy accumulated too much energy on the smallest scales which led to numerical instabilities, as expected from similar studies [13, 9]. The 3 models trained with the *a posteriori* strategy remain stable at the end of the simulation. Moreover, these models are preserving the small-scale features of the flow. Now, we look at quadratic invariants of 2D turbulence, which develops as a double cascade in statistically stationary conditions [1]. In particular, the enstrophy wavenumber spectrum $Z(k)$ and flux $\Pi_Z(k)$ are relevant measures of the success of turbulence parametrizations, as proposed by [8].

In Fig. 3 (left), the enstrophy spectra confirm a large deviation at the smallest scales (highest wavenumbers k) for the Smagorinsky model, and a visible deviation at the largest scales (smallest wavenumbers k) for the *a priori* model and *a posteriori* model trained with $N = 1$. Overall statistical performance of the model is verified by the enstrophy flux in Fig. 3 (right), where the NN trained with the *a posteriori* strategy at $N = 30$ is the closest to the exact flux predicted by the DNS.
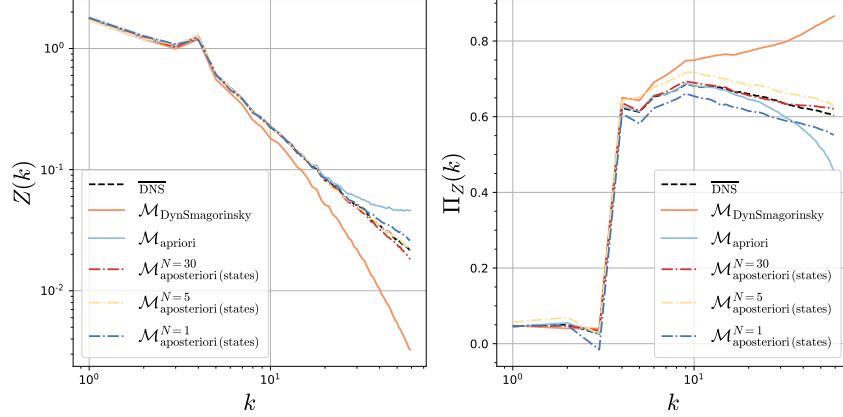
4

Figure 3: Enstrophy spectrum (left) and time-averaged enstrophy flux (right) of the different models w.r.t the reference DNS (dashed). For the NNs models, only those trained with the *a posteriori* strategy are stable and thus have valid statistics.

4 Conclusion

We show that a NN trained jointly with the forward solver on *a posteriori* informations lead to stable simulations and outperforms the classical data-driven strategy and algebraic models based on different statistical metrics for the prediction of quasi-geostrophic turbulence parametrization. However, it is important to consider that this requires a differentiable solver for the reduced order system, which is not the case for most global circulation models developed nowadays.

References

[1] Guido Boffetta. Energy and enstrophy fluxes in the double cascade of two-dimensional turbulence. *Journal of Fluid Mechanics*, 589:253–260, 2007.

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[3] Yu Chen, Ying Tai, Xiaoming Liu, Chunhua Shen, and Jian Yang. Fsrnet: End-to-end learning face super-resolution with facial priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2492–2501, 2018.

[4] Anurag Dipankar, Bjorn Stevens, Rieke Heinze, Christopher Moseley, Günther Zängl, Marco Giorgetta, and Slavko Brdar. Large eddy simulation using the general circulation model icon. *Journal of Advances in Modeling Earth Systems*, 7(3):963–986, 2015.

[5] Hugo Frezat, Guillaume Balarac, Julien Le Sommer, Ronan Fablet, and Redouane Lguensat. Physical invariance in neural networks for subgrid-scale scalar flux modeling. *Physical Review Fluids*, 6(2):024607, 2021.

[6] Masataka Gamahara and Yuji Hattori. Searching for turbulence models by artificial neural network. *Physical Review Fluids*, 2(5):054604, 2017.

[7] Massimo Germano, Ugo Piomelli, Parviz Moin, and William H Cabot. A dynamic subgrid-scale eddy viscosity model. *Physics of Fluids A: Fluid Dynamics*, 3(7):1760–1765, 1991.

[8] Jonathan Pietarila Graham and Todd Ringler. A framework for the evaluation of turbulence closures used in mesoscale ocean large-eddy simulations. *Ocean Modelling*, 65:25–39, 2013.

[9] Yifei Guan, Ashesh Chattopadhyay, Adam Subel, and Pedram Hassanzadeh. Stable a posteriori les of 2d turbulence using convolutional neural networks: Backscattering analysis and generalization to higher re via transfer learning. *arXiv preprint arXiv:2102.11400*, 2021.

[10] Philipp Holl, Nils Thuerey, and Vladlen Koltun. Learning to control pdes with differentiable physics. In *International Conference on Learning Representations*, 2019.

[11] Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021.

[12] Andrew Majda and Xiaoming Wang. *Nonlinear dynamics and statistical theories for basic geophysical flows*. Cambridge University Press, 2006.

[13] Romit Maulik, Omer San, Adil Rasheed, and Prakash Vedula. Subgrid modelling for two-dimensional turbulence using neural networks. *Journal of Fluid Mechanics*, 858:122–144, 2019.

[14] Charles Meneveau and Joseph Katz. Scale-invariance and turbulence models for large-eddy simulation. *Annual Review of Fluid Mechanics*, 32(1):1–32, 2000.

[15] Jonghwan Park and Haecheon Choi. Toward neural-network-based large eddy simulation: application to turbulent channel flow. *Journal of Fluid Mechanics*, 914, 2021.

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8026–8037, 2019.

[17] Stephen B Pope. Turbulent flows, 2001.

[18] Joseph Smagorinsky. General circulation experiments with the primitive equations: I. the basic experiment. *Monthly Weather Review*, 91(3):99–164, 1963.

[19] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4490–4499, 2018.

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] Yes, thoroughly.
   (b) Did you describe the limitations of your work? [Yes] Yes, see Conclusion.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] Yes, no potential negative societal impacts have been identified related to this paper.

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] The code is available at `https://github.com/hrkz/torchqg`.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Yes, we did describe the training details in the "Learning components" part of Section 3.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] No

(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)? [No] No, but runtime is not an important factor for this paper.

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] Yes, we used PyTorch.
   (b) Did you mention the license of the assets? [Yes] We cited the paper, which then refers to the licence.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

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]