On the Effectiveness of Bayesian AutoML methods for Physics Emulators

Peetak P. Mitra*
Department of Mechanical Engineering
University of Massachusetts
Amherst, MA 01003, USA
pmitra@umass.edu

Niccolò Dal Santo
MathWorks Inc.,
Cambridge Business Park
Cambridge, CB4 0HH, UK
ndalsant@mathworks.com

Majid Haghshenas
Department of Mechanical Engineering,
University of Massachusetts
Amherst, MA 01003, USA
mhaghshenas@umass.edu

Shounak Mitra
MathWorks Inc.,
1 Apple Hill Drive
Natick, MA 01002, USA
shmitra@mathworks.com

Conor Daly
MathWorks Inc.,
Cambridge Business Park
Cambridge, CB4 0HH, UK
cdaly@mathworks.com

David P. Schmidt
Department of Mechanical Engineering
University of Massachusetts
Amherst, MA, 01003 USA
schmidt@acad.umass.edu

Abstract

The adoption of Machine Learning (ML) for building emulators for complex physical processes has seen an exponential rise in the recent years. While neural networks are good function approximators, optimizing the hyper-parameters of the network to reach a global minimum is not trivial, and often needs human knowledge and expertise. In this light, automatic ML or autoML methods have gained large interest as they automate the process of network hyper-parameter tuning. In addition, Neural Architecture Search (NAS) has shown promising outcomes for improving model performance. While autoML methods have grown in popularity for image, text and other applications, their effectiveness for high-dimensional, complex scientific datasets remains to be investigated. In this work, a data driven emulator for turbulence closure terms in the context of Large Eddy Simulation (LES) models is trained using Artificial Neural Networks and an autoML framework based on Bayesian Optimization, incorporating priors to jointly optimize the hyper-parameters as well as conduct a full neural network architecture search to converge to a global minima, is proposed. Additionally, we compare the effect of using different network weight initializations and optimizers such as ADAM, SGDM and RMSProp, to explore the best performing setting. Weight and function space similarities during the optimization trajectory are investigated, and critical differences in the learning process evolution are noted and compared to theory. We observe ADAM optimizer and Glorot initialization consistently performs better, while RMSProp outperforms SGDM as the latter appears to have been stuck at a local minima. Therefore, this autoML BayesOpt framework provides a means to choose the best hyper-parameter settings for a given dataset.

Keywords: *Machine Learning, Bayesian Optimization, Fluid Turbulence*

*Corresponding author

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 October 2020 doi:10.20944/preprints202010.0595.v1

1 Introduction

Fluid turbulence is a multi-scale phenomenon and is an essential component of modeling engineering-relevant flows. While solving the full Navier Stokes using Direct Numerical Simulation (DNS) results in the most accurate representation of the complicated, non-linear, non-local, multi-scale phenomenon, DNS is often computationally intractable. Engineering level solutions based on Reynolds Averaged Navier-Stokes (RANS) and Large Eddy Simulations (LES) alleviate this issue by resolving the larger integral length scales and modelling the smaller unresolved scales by introducing a linear operator to the Navier-Stokes equation to reduce the simulation complexity. These models however suffer from the curse of turbulence closure. The linear eddy-viscosity model represents one of the most popular methods for Reynolds stress closure for two-equation RANS as well as Smagronisky-LES models [1, 2]. However, these approximates models are commonly phenomenological/heuristic in nature and thus require fitting to high fidelity DNS datasets for idealized flows [3].

With the incredible strides in the development of sophisticated Machine Learning (ML) algorithms made in the last decade it is only logical that these tools be widely adopted for use with scientific applications (refer to [4, 5, 6] for a review). In particular, Artificial Neural Networks (ANN) have shown great performance for function approximations [7] and recently Deep Neural Network (DNN)-based approaches for fluid problems have gained wide attention [3, 8, 9, 10, 11, 12, 13, 14, 15]. Specifically, the use of ML methods for developing data-driven Reynolds stress closures has shown a lot of promise for canonical as well as complex engineering flows [5, 6, 16, 17, 18, 19]. We extend this effort to emulate a data-driven closure term for compressible flows relevant to modeling advanced propulsion systems.

When training a neural network, one of the the main difficulties is setting the tunable hyper-parameters which need to be chosen and directly dictate the performance of the data-driven model. Manually tuning these hyper-parameters requires expertise and a-priori information about their sensitivities, which can be difficult to develop as more of these techniques are applied to large-scale scientific datasets. In addition, there are some challenges in transferring the best practices from other ML areas to Scientific ML. Primarily because the data used to develop these algorithms are different than scientific data as the latter can be high-dimensional, multi-modal, complex, structured and/or sparse [20]. Often, these suggested settings for hyper-parameters are incompatible due to the nature of the complex non-convex loss manifold for these datasets.

Automatic Machine Learning, or autoML, promises to be an important effort in this area where the optimization occurs without human intervention, and often times yield robust results. Along with tuning parameters, searching for optimal neural network architecture or Neural Architecture Search (NAS), in terms of layer depth and width, has previously shown performance boost [21, 22].

1.1 Our Contributions

In this work, a data driven physics emulator is built for Reynolds Stress closure term using a simple feed-forward neural network and the a-priori performance reported. The effectiveness of using an autoML framework based on Bayesian Optimization to automatically discover optimal parameter as well as identify the best neural network architecture for emulating closure terms based on large resolved scale quantities are investigated. Different optimizers (RMSProp, ADAM and SGDM) and weight initialization (Glorot, He, Narrow-Normal) strategies are used to identify best performing settings that is hyper-parameter and architecture. In addition, the training process of the different optimizer and initialization strategies are compared using weight and function space similarities.

2 Physics of the Problem

The LES-filtered governing equations (using Favre-averaging) for the balance of mass and momentum are as below:

$$\frac{\partial \bar{\rho}}{\partial t} + \frac{\partial}{\partial x_j}(\bar{\rho}\tilde{u_j}) = 0 \tag{1}$$

$$\frac{\partial(\bar{\rho}\tilde{u_i})}{\partial t} + \frac{\partial \bar{\rho}\tilde{u_i}\tilde{u_j}}{\partial x_j} = \frac{\partial}{\partial x_j}[\bar{\rho}\bar{\nu}(\frac{\partial \tilde{u_j}}{\partial x_i} + \frac{\partial \tilde{u_i}}{\partial x_j}) - \frac{2}{3}\bar{\rho}\bar{\nu}\frac{\partial \tilde{u_k}}{\partial x_k}\delta_{ij} - \rho\tau_{ij}^{sgs}] - \frac{\partial \bar{p}}{\partial x_i} + \bar{p}g_i \tag{2}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 October 2020 doi:10.20944/preprints202010.0595.v1

where u represents the velocity, p is the pressure, ρ the fluid density, ν the dynamic the viscosity and τ the subgrid stress term. The effect of the sub-grid scale appears on the right hand side of the governing equations through the sub-grid scale stresses, τ_{ij}, which are modelled using the Boussinesq approximation [23], and the assumption by Smagorinsky that the smallest scales are isotropic [2]. Based on Prandtl mixing length theory, the subgrid viscosity can be derived in terms of characteristic length and one velocity scale [24] as follows, therefore helping to close the Reynolds stress term

$$\tau_{ij}{}^{sgs} - \frac{1}{3}\tau_{kk}{}^{sgs}\delta_{ij} = -\mu_{sgs}S_{ij} \tag{3}$$

$$\mu_{sgs} = \rho(C_s\Delta)^2|\bar{S}| \tag{4}$$

where the superscript sgs stands for subgrid scale terms, Δ is the filter width, and C_s is the Smagorinsky constant, and S_{ij} is the strain rate tensor and is calculated by taking off-diagnal gradients of velocities. In conclusion, the above approximation for the eddy viscosity assumes that changes in the resolved fields are slow, so that subgrid eddies can adjust themselves quickly to the rate-of-strain tensor. Thus, a closure based on a single constant is not universally true and the constant value may have to be adjusted [25], based on fitting the model parameters to high-fidelity data. Since some form of data-fitting is needed to optimize the subgrid scale model parameters even for this simplified approach, one can envisage a purely data-driven method to optimally approximate this changing constant based on large scale resolved terms, motivating our approach. More details about the LES implementation as well as the accuracy of results is reported in [26].

In this work we aim to derive a functional relationship between the large scale resolved flow features and the sub-grid scale unresolved terms, and specifically to approximate the subgrid scale viscosity

$$\mu_{sgs} = f(Re_c, S, \Omega, \Delta K, Y) \tag{5}$$

where Re_c is the Cell Reynolds number, S is the Strain-rate tensor and has six components, Ω is the rotation-rate tensor, and has three components, ΔK is the Kinetic energy gradient, and Y is a non-dimensional term that is a measure of the mesh resolution. The non-dimensionalized input features are chosen in order to impose Galliean-invariance [27, 12].

3 Machine Learning Approach

Differential programming, based on principles of automatic differentiation, is a paradigm in deep learning where parameters of the neural network are trained by gradient-based optimizations [28, 29]. This is indeed useful for scientific ML tasks and helps to improve the parameterization of the approximations of the neural network [30]. In this work, fully connected ANNs are used to find the functional mapping between the target (μ_{sgs}) to the non-dimensional input features. While the neural network implementation is fairly straightforward and done using MATLAB's Deep Learning Toolbox, the a-priori estimation of the best network hyper-parameters, and the architecture itself is non-trivial due to the multi-scale, non-local, non-linear nature of the dataset. To discover the best performing settings for our dataset, an Automatic Machine Learning (autoML) strategy is used and, while there are many available optimization methods [31], the Bayesian Optimization (BayesOpt) approach, which has shown good performance for other data-driven tasks ([32, 33]) is used. Our overall approach is shown in Figure 1 and described in detail in [18, 34]. For the purposes of this paper we limit ourselves to identifying the best performing ANN, given by the BayesOpt algorithm. The coupling of the ANN to a CFD solver and comparing *a-posteriori* performance will be the subject of a subsequent article.

3.1 Automatic Hyper-parameter Tuning using Bayesian Optimization

We explore an automatic hyper-parameter tuning setup within the Bayesian Optimization framework, in which the learning algorithm's generalization performance is modeled as a sample from a Gaussian Process (GP). The posterior distribution induced by the GP leads to efficient use of information gathered by previous experiments, enabling optimal choices for what parameters to try next. To pick the hyper-parameters of the next experiment, one can optimize the expected improvement (EI) [35] over the current best result or the Gaussian process upper confidence bound (UCB) [36]. EI and UCB

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 October 2020 doi:10.20944/preprints202010.0595.v1

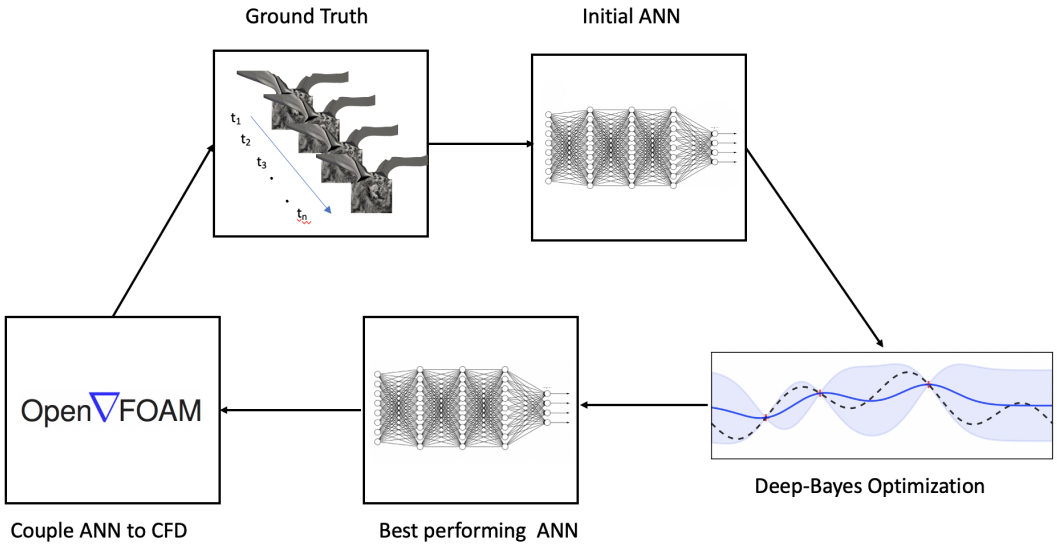

Figure 1: Schematic of the Bayesian Optimization based data-driven physical emulator workflow in identifying the best performing network hyper-parameters and network architecture

have been shown to be efficient in the number of function evaluations required to find the global optimum of many multimodal black-box functions [36, 37]. While there are many hyper-parameters that have an effect on the ANN performance and accuracy, we limit set of parameters to the ones in Table 1, due to their leading order effect on the network performance.

Table 1: Optimization Matrix with interpolation strategies and range of investigation for each hyper-parameter. *The Network Depth is kept constant for all layers, to reduce design space search complexity.

Hyper-Parameter	Min. Range	Max. Range	Interpolation
***Init* LR**	1e-6	1e-2	Logarithmic
LR Drop Factor	10	1000	Integer
Batch Size	100	16000	Logarithmic
Network Depth*	2	10	Integer
Network Width	10	100	Integer

3.1.1 Acquisition Functions for Bayesian Optimization

The acquisition functions evaluate the effectiveness of a point, x, based on the posteriori distribution function, Q [33]. The Expected Improvement function is used due to its superior performance over other methods [33]. The Expected-Improvement acquisition function evaluates the expected amount of improvement in the objective function, ignoring values that cause an increase in the objective. In other words, it is shown mathematically as:

$$EI(x, Q) = E_Q[max(0, \mu_Q(X_{best} - f(x)))] \tag{6}$$

where $\mu_Q(X_{best})$ is the lowest value of the posterior mean, and X_{best} is the location of the lowest posterior mean.

3.2 Solvers and Initialization of Learnable Parameters

The functional mapping, using a neural network, can be viewed as a function of the learnable parameters such as weights and biases. In fact the training process of the neural network is often stochastic - very often two separate runs do not yield the same result. This can be attributed to the choice of optimizer, the precision of the learning process as well as the complex, non-convex loss manifold and the initialization of the weights. This last step has a determination on the performance

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 October 2020 doi:10.20944/preprints202010.0595.v1

of the neural network during the optimization process itself [38, 39, 40]. In this work we explore the effect of choosing different initialization, from well known ones including Glorot[38], He [39] and narrow-normal. In addition to the initialization of the weights the performance and evolution of the training process itself are studied by using different solvers such as ADAM [41], RMSProp [42] and SGDM [40], within the autoML framework.

4 Results

A well-resolved and validated Large Eddy Simulation dataset [26] is used with over 10 million data points, taken across four time-steps from a simulation of a compressible flow within the cylinder of an advanced propulsion system generated using OpenFOAM C++ libraries [43]. 20% of the dataset is set aside for *a-priori* testing. Fifty function evaluations are run for the BayesOpt workflow, for each case, and after each iteration of the BayesOpt run, an error is calculated, ε on the test dataset defined as:

$$\varepsilon = \frac{abs(y_{true} - y_{pred})}{y_{true}} \tag{7}$$

where y_{true} is the test data point, and the y_{pred} is the value predicted by the ANN. The best performing settings for all the possible combinations of optimizers and initializations are reported in Table 2. For each of our network evaluation, the architecture is optimized by self-repeating blocks of Dense Layer -> Leaky ReLU activation -> Dropout Layer. The p value of the Dropout layer is set to 0.2, per best practices [44]. For the specific case of optimizing the network architecture, the best set of Layer Width, W, and Layer Depth, D, the latter optimizing the number of the self-repeating blocks for a given BayesOpt evaluation, are identified and reported.

4.1 Network Performance

The final objective of this effort is to not only identify the best settings for a given choice of solver and initialization as reported in Table 2, but also understand the effective cost one can expect to pay when this network is coupled to a CFD solver (see Figure 1) to make run-time inferences across multiple timesteps and grid points, in a realistic CFD simulation. This inference cost is directly proportional to the size of the network, width (W) and depth (D). Therefore total number of terms based on the formulation $N = I * W + D * W * W + W * O$, where I refers to the input features, in this case 14, O refers to the output features, which is 1 is also reported in Table 2.

Table 2: Best Performing hyper-parameters from autoML BayesOpt

Optimizer	Initialization	Batch Size	Learning Rate	Width	Depth	ε	N
ADAM	Glorot	1426	9.56e-04	91	10	1e-05	84094
ADAM	He	12719	3.77e-04	50	7	2e-04	18207
ADAM	narrow-normal	12553	1.65e-04	59	3	1e-03	11272
SGDM	Glorot	6814	0.0098	91	9	2e-03	75812
SGDM	He	1161	0.0098	89	6	1e-03	48778
SGDM	narrow-normal	9609	0.0015	64	5	7e-02	21381
RMSProp	Glorot	291	1e-04	78	4	8e-04	25432
RMSProp	He	11630	1.86e-05	55	9	1e-03	28004
RMSProp	narrow-normal	1085	2e-05	89	2	1e-02	17090

It is found that the ADAM Glorot and ADAM He combinations perform the best in terms of absolute errors, although the ADAM Glorot configuration has the highest number of parameters. The RMPSProp on average, performs better than SGDM optimizer. This can be explained as RMSProp is an adaptive learning rate method and is capable in navigating regions of local optima and whereas that SGDM performs poorly navigating ravines and makes hesitant progress towards local optima [45]. It is observed that the Glorot initialization consistently has the best performance in terms of error, ε, which is an interesting result as He and ReLU activation have performed well for Convolutional Neural Networks [39].

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 October 2020 doi:10.20944/preprints202010.0595.v1

4.2 Visualizing Weight and Function Space Similarity

We investigate the similarities in the weight and function space for similar as well as differently initialized trajectories, in an effort to better understand the commonalities in the optimization process. In order to do that, the simulations are check-pointed after every epoch and the cosine similarity among the weights are computed, defined by $cos(\theta_1, \theta_2) = \frac{\theta_1^T \theta_2}{||\theta_1|| ||\theta_2||}$, where θ_1 and θ_2 are the vectorized weights of the ANN. From the left panel in Figure 2, it is observed the checkpoints along a trajectory are largely similar in the weight space, in this case for ADAM-Glorot configuration. However, when compared to the same initialization, He, and different optimizers, ADAM and SGDM we observe major differences in the trajectories in the weight space and this can be attributed to the optimization methods and the stochasticity of the learning process therein. Therefore we see that functions within a single trajectory exhibit higher similarity and this similarity map is optimizer-initialization specific.

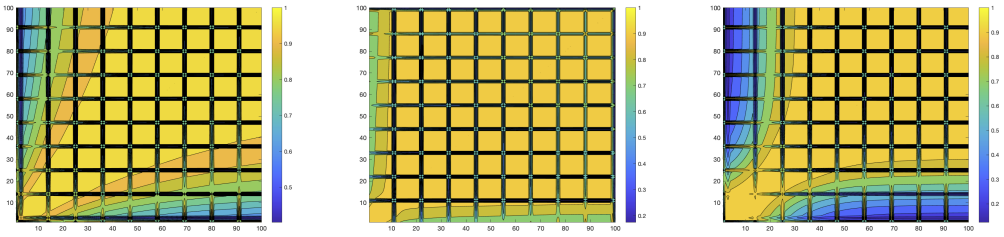

Figure 2: Cosine-similarity between checkpoints to measure weight space alignment for the ADAM-Glorot (left), SGDM-He (middle) and ADAM-He (right) configurations. This shows the stochasticity of the weights trajectories that can occur during the training process.

In addition to investigating the similarities in the weight space which are inherently high-dimensional and therefore non-intuitive, the use of dimensionality reduction methods are used, such as t-SNE [46, 47, 48] to observe the trajectory of the checkpoints in a 2D space in Figure 3. It is observed that the Glorot and He have overlapping similarities, which makes sense as they both have similar functional forms and theoretical analysis: they both find a good variance for the distribution from which the initial parameters are drawn and only differ in the type of distribution they use - Gaussian for He, and Uniform for Glorot.

On the other hand, the t-SNE trajectory of the narrow-normal initialization which independently samples from the normal distribution with zero mean and a standard deviation of 0.01, thereby not incorporating information from the data, only occupies a small region in the phase space. For the same initialization, and different solvers, it is observed all the three different solutions start off from the same point but quickly diverge and follow different trajectories, consistent with previously reported observations [49]. This shows that the functions explored by different optimizers are far away and this leads to the divergence and differences in predictions, while functions explored within a single trajectory tend to be much more similar. This observation is independently verified by comparing weight-space similarities in the first few dense layer for two separate fully converged networks, trained using the same initialization, and different solvers and observe no discernible common patterns (high similarity scores) in the map, see Figure 5 in the Appendix.

Lastly another dimensionality reduction method in Singular Value Decomposition (SVD) are used and the singular values for each set of runs plotted. It is observed that the narrow-normal initialization has the least significant number of indices, which can be explained due to the nature of the initialization. Whereas for the He and Glorot initializations RMSProp and SGDM tends to have higher singular values compared to ADAM, thereby making ADAM the most suitable candidate for a pruning-type operation to reduce the number of overall parameters in the network. This is important for the primary objective of this effort to help isolate and identify candidate network settings that can be successfully coupled to a non-linear PDE solver for run-time predictions.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 October 2020 doi:10.20944/preprints202010.0595.v1

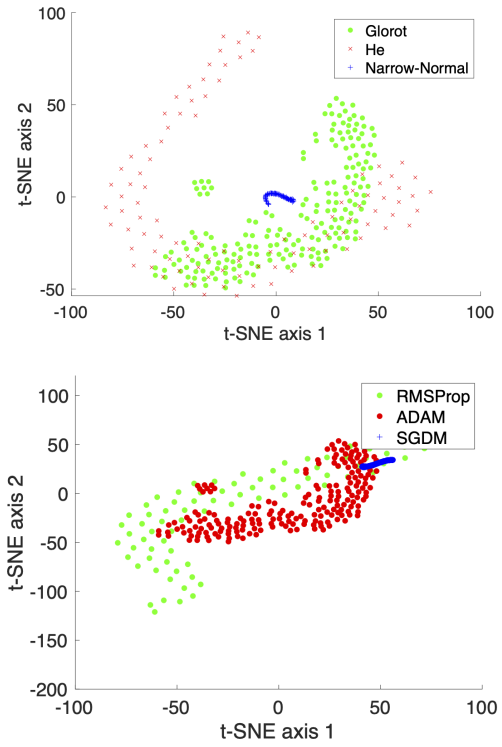

Figure 3: The left panel shows the 2D point rendering using t-SNE for different initialization for the same solver, ADAM. Glorot and He have similar functional space, whereas the narrow-normal indicates a very small region of exploration in the function space, explaining the high error observed in Table 2. The right panel shows the trajectories of the different optimizer, given same weight initialization and shows the difference in the function space exploration of each optimizer.

5 Conclusions

In this work, the effectiveness of an automatic Machine Learning framework, based on Bayesian Optimization, is demonstrated in identifying the best network parameters in a differential programming paradigm relevant to scientific computing. We provide a measure of efficiency for using these settings in terms of expected advantage-to-compute. Comparing different initialization and solver settings, and dimensionality reduction techniques, the differences in the learning process are observed, and suitability in identifying candidate network settings for pruning tasks matrix reduction for cheaper run-time inference are noted.

6 Acknowledgements

The authors would like to acknowledge the funding and compute support from partners of The ICEnet Consortium, including AVL, Cummins, Convergent Science Inc., SIEMENS/CD-Adapco, MathWorks Inc., and NVIDIA Corporation.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 October 2020 doi:10.20944/preprints202010.0595.v1

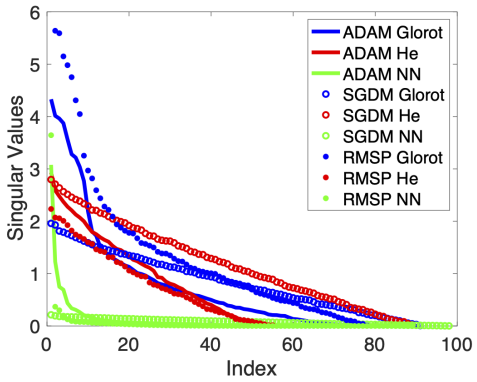

Figure 4: Singular values for different initialization and optimizers for the first dense layer in the trained network, showing key differences in the nature of the weight matrix. RMSProp and SGDM have higher singular values compared to ADAM for the same settings. Specifically, ADAM-Glorot stands as best candidate for a low-rank approximation pruning-type operation.

References

[1] 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.

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

[3] Gavin D Portwood, Peetak P Mitra, Mateus Dias Ribeiro, Tan Minh Nguyen, Balasubramanya T Nadiga, Juan A Saenz, Michael Chertkov, Animesh Garg, Anima Anandkumar, Andreas Dengel, et al. Turbulence forecasting via neural ode. *arXiv preprint arXiv:1911.05180*, 2019.

[4] J Nathan Kutz. Deep learning in fluid dynamics. *Journal of Fluid Mechanics*, 814:1–4, 2017.

[5] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

[6] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. Turbulence modeling in the age of data. *Annual Review of Fluid Mechanics*, 51:357–377, 2019.

[7] Zarita Zainuddin and Ong Pauline. Function approximation using artificial neural networks. *WSEAS Transactions on Mathematics*, 7(6):333–338, 2008.

[8] Gavin D Portwood, Balasubramanya T Nadiga, Juan A Saenz, and Daniel Livescu. Analysis and interpretation of out-performing neural network residual flux models. *arXiv preprint arXiv:2004.07207*, 2020.

[9] Niccolò Dal Santo, Simone Deparis, and Luca Pegolotti. Data driven approximation of parametrized pdes by reduced basis and neural networks. *Journal of Computational Physics*, 416:109550, 2020.

[10] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

[11] Qian Wang, Nicolò Ripamonti, and Jan S. Hesthaven. Recurrent neural network closure of parametric pod-galerkin reduced-order models based on the mori-zwanzig formalism. *Journal of Computational Physics*, 410:109402, 2020.

[12] Julia Ling, Andrew Kurzawski, and Jeremy Templeton. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics*, 807:155–166, 2016.

[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] Peetak Mitra, Majid Haghshenas, and David Schmidt. Super resolution approach for accelerating fluid simulations. *Bulletin of the American Physical Society*, 2020.

[15] Krzysztof Rojek, Marcin Rojek, Mariusz Kolanko, and Peetak Mitra. Ai accelerated simulations for cfd. *Bulletin of the American Physical Society*, 2020.

[16] Peetak Mitra, Mateus Dias Ribeiro, and David Schmidt. A data-driven approach to modeling turbulent flows in an engine environment. *APS*, pages G16–003, 2019.

[17] Mateus Dias Ribeiro, Gavin D Portwood, Peetak Mitra, Tan Mihn Nyugen, Balasubramanya T Nadiga, Michael Chertkov, Anima Anandkumar, and David P Schmidt. A data-driven approach to modeling turbulent decay at non-asymptotic reynolds numbers. *Bulletin of the American Physical Society*, 2019.

[18] Majid Haghshenas, Peetak Mitra, Niccolo Dal Santo, Mateus Dias Ribeiro, Shounak Mitra, and David Schmidt. Les turbulence model with learnt closure; integration of dnn into a cfd solver. *Bulletin of the American Physical Society*, 2020.

[19] Varun Shankar, Gavin Portwood, Arvind Mohan, Peetak Mitra, Venkat Viswanathan, and David Schmidt. Rapid spatiotemporal turbulence modeling with convolutional neural odes. *Bulletin of the American Physical Society*, 2020.

[20] Rick Stevens, Valerie Taylor, Jeff Nichols, Arthur Barney Maccabe, Katherine Yelick, and David Brown. Ai for science. Technical report, Argonne National Lab.(ANL), Argonne, IL (United States), 2020.

[21] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[22] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956, 2019.

[23] Edward A Spiegel and G Veronis. On the boussinesq approximation for a compressible fluid. *The Astrophysical Journal*, 131:442, 1960.

[24] W Malalasekera and HK Versteeg. *An introduction to computational fluid dynamics: the finite volume method*. Pearson Prentice Hall Upper Saddle River, NJ, 2007.

[25] M Meldi, Didier Lucor, and P Sagaut. Is the smagorinsky coefficient sensitive to uncertainty in the form of the energy spectrum? *Physics of Fluids*, 23(12):125109, 2011.

[26] Mateus Dias Ribeiro, Alex Mendonca Bimbato, Mauricio Araujo Zanardi, Jose Antonio Perrella Balestieri, and David P Schmidt. Large-eddy simulation of the flow in a direct injection spark ignition engine using an open-source framework. *INTERNATIONAL JOURNAL OF ENGINE RESEARCH*, 2020.

[27] Charles G Speziale. Galilean invariance of subgrid-scale stress models in the large-eddy simulation of turbulence. *Journal of fluid mechanics*, 156:55–62, 1985.

[28] Atılım Günes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637, 2017.

[29] Li Li, Stephan Hoyer, Ryan Pederson, Ruoxi Sun, Ekin D Cubuk, Patrick Riley, and Kieron Burke. Kohn-sham equations as regularizer: building prior knowledge into machine-learned physics. *arXiv preprint arXiv:2009.08551*, 2020.

[30] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[31] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *arXiv preprint arXiv:1908.00709*, 2019.

[32] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[33] Michael A Gelbart, Jasper Snoek, and Ryan P Adams. Bayesian optimization with unknown constraints. *arXiv preprint arXiv:1403.5607*, 2014.

[34] Peetak Mitra, Majid Haghshenas, Niccolo Dal Santo, Conor Daly, Shounak Mitra, and David Schmidt. Towards building robust neural network models for fluid simulations. *Bulletin of the American Physical Society*, 2020.

[35] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129):2, 1978.

[36] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009.

[37] Ruben Martinez-Cantin. Bayesopt: A bayesian optimization library for nonlinear optimization, experimental design and bandits. *The Journal of Machine Learning Research*, 15(1):3735–3739, 2014.

[38] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[40] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[41] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[42] Tijmen Tieleman and Geoffrey Hinton. Rmsprop: Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. *COURSERA Neural Networks Mach. Learn*, 2012.

[43] Hrvoje Jasak, Aleksandar Jemcov, Zeljko Tukovic, et al. Openfoam: A c++ library for complex physics simulations. In *International workshop on coupled methods in numerical dynamics*, volume 1000, pages 1–20. IUC Dubrovnik Croatia, 2007.

[44] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[45] Richard Sutton. Two problems with back propagation and other steepest descent learning procedures for networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986*, pages 823–832, 1986.

[46] Laurens Van Der Maaten. Barnes-hut-sne. *arXiv preprint arXiv:1301.3342*, 2013.

[47] Robert A Jacobs. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4):295–307, 1988.

[48] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[49] Dami Choi, Christopher J Shallue, Zachary Nado, Jaehoon Lee, Chris J Maddison, and George E Dahl. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*, 2019.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 October 2020 doi:10.20944/preprints202010.0595.v1

7 Appendix

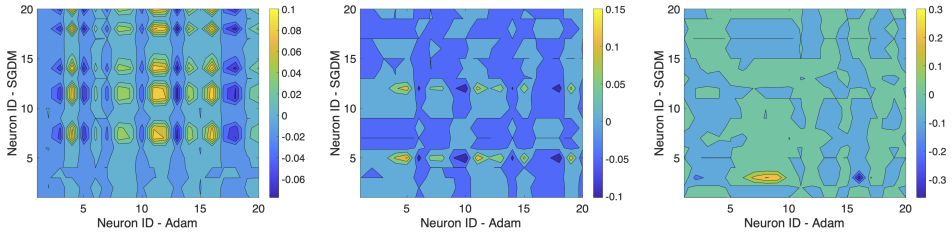

Figure 5: We compare the ADAM and SGDM optimizers for the same settings of initialization at different dense layers of the network and find no resemblance of similarity between their weights indicating very different function and weight space trajectories for their optimized settings.