Forward Gradients for Data-Driven CFD Wall Modeling

Jan Hückelheim
Argonne National Laboratory
Lemont, IL, USA
jhueckelheim@anl.gov

Tadbhagya Kumar
Argonne National Laboratory
Lemont, IL, USA
tkumar@anl.gov

Krishnan Raghavan
Argonne National Laboratory
Lemont, IL, USA
kraghavan@anl.gov

Pinaki Pal
Argonne National Laboratory
Lemont, IL, USA
pal@anl.gov

Abstract

Computational Fluid Dynamics (CFD) is used in the design and optimization of gas turbines and many other industrial/ scientific applications. However, the practical use is often limited by the high computational cost, and the accurate resolution of near-wall flow is a significant contributor to this cost. Machine learning (ML) and other data-driven methods can complement existing wall models. Nevertheless, training these models is bottlenecked by the large computational effort and memory footprint demanded by back-propagation. Recent work has presented alternatives for computing gradients of neural networks where a separate forward and backward sweep is not needed and storage of intermediate results between sweeps is not required because an unbiased estimator for the gradient is computed in a single forward sweep. In this paper, we discuss the application of this approach for training a subgrid wall model that could potentially be used as a surrogate in wall-bounded flow CFD simulations to reduce the computational overhead while preserving predictive accuracy.

1 Introduction

Reverse mode automatic differentiation and back-propagation can efficiently compute gradients, but often require large amounts of memory. This is due to the need for storing intermediate state information. Moreover, these computations are not supported on all hardware, particularly on some AI accelerators. In this work, we explore the use of forward gradients as an alternative to back-propagation in the context of data-driven machine learning models to augment conventional computational fluid dynamics (CFD) methods for simulating film cooling of turbo-machinery components such as gas turbines.

To perform parametric design studies of such components, it is often desired to have high-fidelity simulations that capture a wide range of spatio-temporal scales. Wall-resolved large-eddy simulations (WRLES) [1, 2, 3, 4, 5] capture the unsteady features and provide a feasible way for simulating complex turbulent flows at a reasonable cost. However, the computational cost incurred due to very high resolution needed to resolve the viscous scales near the wall is still high. To overcome this, wall models are used to avoid the need to resolve the near-wall region, providing a feasible way for LES of wall-bounded flows at high Reynolds numbers.

Machine Learning and the Physical Sciences Workshop, NeurIPS 2023.

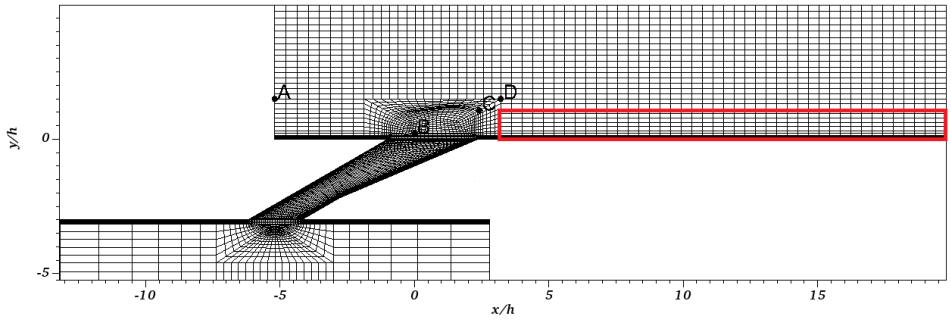


Figure 1: Wall resolved LES mesh of the film cooling setup. The highlighted region (red) depicts the domain of data collection for training the ML wall model

Recent advancements in machine learning and high performance computing have motivated new efforts to develop data-driven methods that complement existing wall models. Some popular methods and architectures in the literature that facilitate such development are Random Forest regression ([6, 7]), artificial neural networks [8, 9, 10, 11], and convolutional neural networks[12, 13, 14]. In most of these approaches, high fidelity WRLES are performed to gather data corresponding to flow features (velocity of the fluid, pressure gradients, etc.). Then, a data-driven regression model is trained to predict the wall-shear stress from these flow features, which is used as a wall model for LES of turbulent channel flows with coarser near-wall resolution.

2 Application

In this application, we seek to develop a data-driven wall model to predict wall shear stress in large-eddy simulation (LES) of a gas turbine film cooling configuration. Film cooling is a popular technique used to reduce turbine temperature and protect them from thermal failure. As the hot gases exit from the combustor and enter the turbine stage, the temperatures can be higher than the melting point of turbine stage materials. Cooling strategies thus become necessary for thermal management of the turbine blades. Since conventional CFD models suffer from the aforementioned limitations, neural network-based wall models are being investigated as a surrogate in wall-bounded flow simulations to reduce the computational overhead while preserving predictive accuracy.

2.1 Data Generation

The data used in this work to train the data-driven model is derived from WRLES of a 7-7-7 cooling hole configuration detailed in [5] for two blowing ratios (BR): 1 and 1.5. The film cooling configuration consists of three parts: a plenum feeding cool air to a flat surface via a single row of cooling holes (represented with a single hole with periodic boundary conditions in the spanwise direction). The flow was simulated using the Nek5000 [15] platform, a higher-order spectral element CFD code, using its Low Mach flow solver. Figure 1 depicts the WRLES mesh used to perform the CFD simulation, and the red box highlights the flow domain which is used to train the data-driven model. The flow data in the highlighted region was collected at 91 planes for $y \in [0.05, 0.5]$ such that $z \in [-3, 3]$ and $x \in [3.2, 19.6]$.

2.2 Data-driven Modeling

Fluid velocity and velocity gradients extracted from the WRLES serve as the Ml model input features, whereas the streamwise shear stress is used as the output. To make shear stress predictions at a grid point located at (i, j, k), pointwise flow features from neighboring x, y, z locations are combined in a stencil-like form. If the pointwise feature set consisting of fluid velocity components and gradients is denoted by $\mathbf{X}_{i,j,k}$, the corresponding 3D stencil feature set is given by:

$$\mathbf{X}_{st} = \mathbf{X}_{i,j,k} + \mathbf{X}_{i-1,j,k} + \mathbf{X}_{i+1,j,k} + \mathbf{X}_{i,j,k+1} + \mathbf{X}_{i,j,k-1} + \mathbf{X}_{i,j+1,k} + \mathbf{X}_{i,j+2,k} \tag{1}$$

The feature stencil at time $t = T$ is combined with stencil features with time delay ($t = T - 1$) to form the input features to the network, as shown in Figure 2. The output label is the streamwise

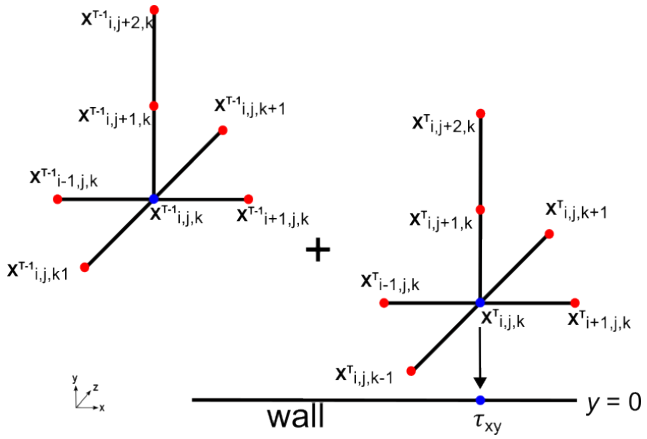


Figure 2: Combining features in a stencil form at two timew, $t = T$ and $t = T - 1$ to make streamwise shear stress predictions (τ_{xy}) at a point. $\mathbf{X}$ denotes the feature set at a point consisting of velocity and velocity gradients, the superscipt denotes the time, and the subscripts denote the coordinates.

shear stress (τ_{xy}) at time $t = T$. The network consists of an input layer of size equal to the number of features used and 3 hidden layers with 64 neurons each with a dropout value of 0.1 and ReLU activation function. The output layer is a dense linear layer of size one and predicts the wall shear stress in the streamwise direction (τ_{xy}). The prepared input - output data is scaled in the range [-1,1] using Min-Max scaling. The dataset for BR1 is randomly shuffled and an 80%-20% split is performed to generate training and validation datasets, respectively. The loss function used to train the network is the mean squared error (E_w):

$$E_w = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2}$$

The neural network model is implemented in PyTorch and is trained using the Adam optimizer with early stopping criterion.

3 Forward Gradients

The key contribution of this paper is the implementation and use of forward gradients for a physical science use-case, not demonstrated earlier. We summarize the theory as shown in [16], followed by a description of our implementation strategy.

3.1 Theory

Given a objective function $f(\theta) : \mathbb{R}^n \to \mathbb{R}^n$, we seek to solve the optimization problem

$$\underset{\theta \in \mathbf{\Omega}}{min} f(\theta). \tag{3}$$

Intuitively, we seek to find a θ in the search space $\mathbf{\Omega}$ such that the objective function value is minimized. The objective function can take many forms but is usually expected to be twice differentiable. This is required because, typically, the process of finding the minima involves a search using the gradients of the objective function denoted $\nabla_\theta f(\theta)$. Since exact calculation of $\nabla_\theta f(\theta)$ is usually expensive, several simplifications are performed based on the assumption that, what is usually required for efficient learning is the directional derivative of f with respect to θ. This can be accomplished by forward gradients [16] $g(\theta)$ such that

$$g(\theta) = (\nabla_\theta f(\theta).\mathbf{v})\mathbf{v} \tag{4}$$

where $\mathbf{v}$ is a perturbation vector chosen as a multivariate random variable sampled $\mathbf{v} \sim p(\mathbf{v})$ such that the scalar components v_i are independent with zero mean and unit variance for all i's. The key insight here is that, $g(\theta)$ through the perturbation vector allows us to estimate the directional derivatives without needing to explicitly compute the gradient $\nabla_\theta f(\theta)$. This allows for a runtime

advantage over typical back-propagation where we can interpret the directions of update without needing to distinguish the updates for each scalar parameter in θ. The process of evaluating forward gradients typically involves a three step process where we

1. sample a random perturbation $\mathbf{v} \sim p(\mathbf{v})$, which has the same size with f's argument,
2. run forward mode autodiff to evaluate $(\nabla_\theta f(\theta).\mathbf{v})$ and $f(\theta)$ simultaneously, and finally
3. multiply $(\nabla_\theta f(\theta).\mathbf{v})$ with the perturbation vector $\mathbf{v}$ to obtain the forward gradients $g(\theta)$.

As long as $g(\theta)$ is similar to the true gradients, or points in a descent direction, the optimization problem can still be solved efficiently. It is shown in [16] that the forward gradients are an unbiased estimator of $\nabla_\theta f(\theta)$ and can be used to solve the optimization problem (4).

3.2 Implementation

Forward gradients are propagated through the model using the forward mode, also known as tangent mode, of automatic differentiation [17]. Since the code used by [16] is not yet publicly available at the time of writing, we used our own framework, which not only supports the computation of randomized forward gradients, but also computes entire Jacobians of the network or individual layers, as well as mixed-mode differentiation to freely combine forward gradients with conventional back-propagation across layers. The implementation is already freely available on github[1].

The framework implements a so-called *tangent model* for selected PyTorch layers, which implement the forward mode of automatic differentiation for these layers. It also provides a simple way to combine multiple layers in a way that mimics standard PyTorch usage, and can therefore be used as a drop-in replacement for PyTorch in many cases. The tangent model of individual layers is implemented by using existing PyTorch layers underneath, which allows us to support GPUs, AI accelerators, and other hardware platforms efficiently. Since our framework can compute forward gradients without using PyTorch back-propagation, it can even be used to obtain approximate gradients on AI accelerators.

We discuss here the tangent model of a 2D convolution layer to illustrate the method. For each output pixel and output channel N_i and C_{out_j}, `Conv2d` computes an output value o by adding a bias to a weighted sum of cross correlations over the input values u at nearby pixels across input channels as

$$o(N_i, C_{out_j}) = b(C_{out_j}) + \sum_{k=0}^{C_{in}-1} w(C_{out_j}, k) \star u(N_i, k). \tag{5}$$

Differentiating with respect to the bias, weights and inputs yields

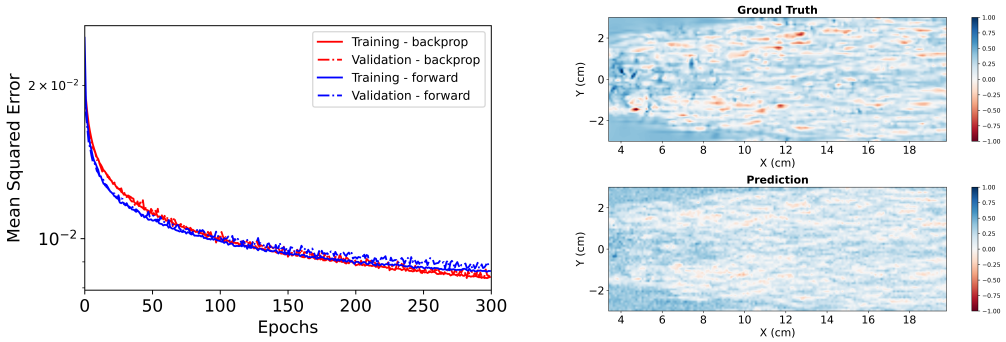
$$\dot{o}(N_i, C_{out_j}) = \dot{b}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \dot{w}(C_{out_j}, k) \star u(N_i, k) + \sum_{k=0}^{C_{in}-1} w(C_{out_j}, k) \star \dot{u}(N_i, k), \tag{6}$$

where derivative terms are denoted using a dot above. To compute forward gradients, the term $\dot{w}$ needs to be replaced with a random vector as discussed in Section 3.1. The term $\dot{u}$ needs to be replaced with the output of the preceding tangent layer, or dropped in case of the first layer of the network. The tangent model shown in (6) can itself be computed using two calls to `Conv2d`.

4 Evaluation

Figure 3a shows the loss function evolution for both the forward gradient approach and back-propagation approach. The network is trained using ADAM optimizer, with an initial learning rate $\eta = 2 \times 10^{-4}$. As training progresses, the learning rate is decayed by a factor $\gamma = 0.5$ if the validation loss does not change for 20 epochs. Figure 3b compares the instantaneous shear stress predictions from the neural network trained with forward gradient approach against the ground truth at a time instant out of the training sample. It is seen that the trained network captures the overall shear stress distribution in the domain.

[1] URL removed for anonymization



(a) (b)

Figure 3: Comparison of a) loss evolution between the forward gradient approach and standard back-propagation for training on the wall-resolved LES dataset, and b) out-of-sample predictions for normalized instantaneous wall shear stress from the forward gradient approach versus the ground truth.

5 Conclusion and next steps

We demonstrated that forward gradients can be used effectively for training without back-propagation in the context of computational fluid dynamics applications. In future work, we plan to investigate different optimizer settings and network architectures, which may lead to setups that are more suited to the randomized nature of forward gradients, and perform even better or more robustly. We also plan to experiment with computing the mean from multiple forward gradient computations with varying random seeds, which is likely to yield a more accurate representation of the true gradient at a higher computational cost, but still without the memory footprint of conventional back-propagation.

The generation of (pseudo-)random numbers is a performance bottleneck in our current implementation, since the dense layers have a large number of parameters, each of which requiring its own random number that will be subsequently used only for a small number of floating point operations. This may be a fundamental challenge with the forward gradient approach that was not discussed in [16], and provides further motivation for exploring other neural network architectures that have fewer parameters that get re-used more often.

6 Acknowledgments

This research used resources of the Argonne Leadership Computing Facility, and was supported by Laboratory Directed Research and Development (LDRD) funding from Argonne National Laboratory, provided by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-06CH11357. The gas turbine film cooling datasets were generated under a research project in collaboration with Raytheon Technologies Research Center (RTRC) funded by the DOE Advanced Manufacturing Office (AMO) through the High Performance Computing for Energy Innovation (HPC4EI) program. Lastly, the authors would like to acknowledge the computing core hours available through the Bebop cluster provided by the Laboratory Computing Resource Center (LCRC) at Argonne National Laboratory and National Energy Research Scientific Computing Center (NERSC) Cori supercomputer for the generation of the high-fidelity simulation datasets.

References

[1] Kunlun Liu and Richard Pletcher. Large eddy simulation of discrete-hole film cooling in a flat plate turbulent boundary layer. In *38th AIAA Thermophysics Conference*, page 4944, 2005.

[2] X Guo, W Schröder, and M Meinke. Large-eddy simulations of film cooling flows. *Computers & Fluids*, 35(6):587–606, 2006.

[3] Yulia V Peet and Sanjiva K Lele. Near field of film cooling jet issued into a flat plate boundary layer: Les study. In *Turbo Expo: Power for Land, Sea, and Air*, volume 43147, pages 409–418, 2008.

[4] Todd A Oliver, Joshua B Anderson, David G Bogard, Robert D Moser, and Gregory Laskowski. Implicit les for shaped-hole film cooling flow. In *Turbo Expo: Power for Land, Sea, and Air*, volume 50879, page V05AT12A005. American Society of Mechanical Engineers, 2017.

[5] Austin C Nunno, Sicong Wu, Muhsin Ameen, Pinaki Pal, Prithwish Kundu, Ahmed Abouhussein, Yulia Peet, Michael Joly, and Peter Cocks. Wall-resolved les study of shaped-hole film cooling flow for varying hole orientation. In *AIAA SCITECH 2022 Forum*, page 1404, 2022.

[6] Jian-Xun Wang, Jin-Long Wu, and Heng Xiao. Physics-informed machine learning approach for reconstructing reynolds stress modeling discrepancies based on dns data. *Physical Review Fluids*, 2(3):034603, 2017.

[7] Tadbhagya Kumar, Pinaki Pal, Austin C Nunno, Sicong Wu, Opeluwa Owoyele, Michael Joly, and Dima Tretiak. Development of a data-driven wall model for large-eddy simulation of gas turbine film cooling flows. In *AIAA SciTech Forum and Exposition*, page 1254, 2023.

[8] Michele Milano and Petros Koumoutsakos. Neural network modeling for near wall turbulent flow. *Journal of Computational Physics*, 182(1):1–26, 2002.

[9] Zhideng Zhou, Guowei He, and Xiaolei Yang. Wall model based on neural networks for les of turbulent flows over periodic hills. *Physical Review Fluids*, 6(5):054610, 2021.

[10] XIA Yang, Suhaib Zafar, J-X Wang, and Heng Xiao. Predictive large-eddy-simulation wall modeling via physics-informed neural networks. *Physical Review Fluids*, 4(3):034602, 2019.

[11] Xinyi LD Huang, Xiang IA Yang, and Robert F Kunz. Wall-modeled large-eddy simulations of spanwise rotating turbulent channels—comparing a physics-based approach and a data-based approach. *Physics of Fluids*, 31(12):125105, 2019.

[12] Junhyuk Kim and Changhoon Lee. Prediction of turbulent heat transfer using convolutional neural networks. *Journal of Fluid Mechanics*, 882, 2020.

[13] Luca Guastoni, Miguel P Encinar, Philipp Schlatter, Hossein Azizpour, and Ricardo Vinuesa. Prediction of wall-bounded turbulence from wall quantities using convolutional neural networks. In *Journal of Physics: Conference Series*, volume 1522, page 012022. IOP Publishing, 2020.

[14] Naoki Moriya, Kai Fukami, Yusuke Nabae, Masaki Morimoto, Taichi Nakamura, and Koji Fukagata. Inserting machine-learned virtual wall velocity for large-eddy simulation of turbulent channel flows, 2021.

[15] Paul Fischer, James Lottes, and Henry Tufo. Nek5000. [Computer Software] `https://doi.org/10.11578/dc.20210416.29`, jun 2007.

[16] Atılım Güneş Baydin, Barak A Pearlmutter, Don Syme, Frank Wood, and Philip Torr. Gradients without backpropagation. *arXiv preprint arXiv:2202.08587*, 2022.

[17] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.