CS230

Learning Physically Realizable Turbulent Velocity Fields

Olivia Martin
Mechanical Engineering Department
Stanford University
ogmartin@stanford.edu

Ryan Hass
Mechanical Engineering Department
Stanford University
ryanhass@stanford.edu

Kyle Pietrzyk
Energy Resources Engineering Department
Stanford University
pietrzyk@stanford.edu

1 Abstract

In this project, we look to train a fully-connected neural network (NN) to directly correct low-resolution, large eddy simulation (LES) snapshots of wall-bounded turbulence. The NN is trained to satisfy conservation equations and reduce error in matching a variety of spatially averaged quantities obtained from high-resolution data. It is then evaluated by its ability to produce physically convincing flow fields and accurate spatiotemporally averaged profiles. Due to limitations in time and computational resources, the resulting NN displays signs of high bias, but further testing shows the potential of the presented methodology.

2 Introduction

Turbulent flows exhibit computationally prohibitive ranges of temporal and spatial scales, but commonly appear in aeroacoustics, vehicle aerodynamics, combustion, and electronics cooling [1]. By resolving larger energy-containing scales of motion and using subgrid-scale (SGS) models to account for small scale effects, LES demonstrates computational efficiency in modeling turbulence and transitional flows [2].

Despite the efficiency and robustness of LES, inaccuracies occur in wall-bounded flows, where the energy-containing scales become smaller than the grid spacing of a computational mesh near the wall [3]. Due to the increasing popularity of LES [1] and the pervasive nature of wall-bounded turbulence in engineering applications, it is of high interest to a broad research community to improve the accuracy of LES while maintaining low computational costs.

We are interested in training a NN to directly "correct"[1] the inaccuracies of coarse-resolution LES snapshots from a turbulent half-channel flow. Specifically, our NN will take in low-resolution data (i.e., velocity and pressure fields) with incorrect statistics[2], propagate the data through fully connected layers, and output "corrected" flow fields that possess accurate statistics.

3 Related Work

While we consider correcting the low-resolution data from a simulation that uses a standard, deterministic SGS model, a majority of previous works consider learning the appropriate SGS models directly. Summaries of such works are provided below:

- David Ching trained a NN on averaged flow field variables from LES data to predict a model closure for the averaged Navier Stokes equations [4].
- Gamahara *et al.* [5] trained a NN on direct numerical simulation (DNS) data to learn the residual stress tensor and act as a SGS model for LES. The improvements in performance were no better than that provided by a well-known model introduced in 1963 [6].
- Beck *et al.* [7] built upon the work of Gamahara *et al.* by making no assumptions about filtering and discretization schemes. As a result, they derive an exact LES closure and trained an RNN to predict it. However, the learned closure is numerically unstable, since it is not the exact the solution, and must be modified for practical use.

[1] Make small physical changes to the flow field.
[2] e.g. mean velocity, velocity covariance, etc.

CS230: Deep Learning, Fall 2021, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

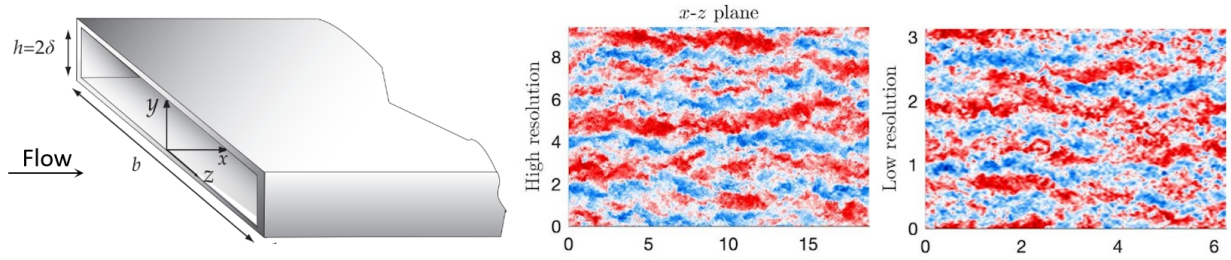

Figure 1: **Left:** The wall-bounded domain setup. **Middle:** The x-z plane of a HR data sample. **Right:** The x-z plane of a LR data sample. The contours are used to display the velocity variations in the turbulent flow.

- Park & Choi [8] used a fully connected neural network to predict the subgrid stress tensor for use in channel flow LES.

One notable exception to these works is that of Subramaniam *et al.* [9], where a NN is trained to directly synthesize physically consistent small-scale dynamics in turbulence. We draw heavily on this idea in our approach.

4 Dataset and Features

We generated our data set using *PadeOps*, an LES turbulence simulation code developed at Stanford in Professor Sanjiva Lele's lab [10]. Our data set consists of 333 low-resolution (LR) snapshots and 84 high-resolution (HR) snapshots[3] of 3D turbulent channel flow fields. The LR flow fields are used as the inputs to the NN, while the HR flow fields are used to create labels[4], as large-scale flow features are assumed to be properly resolved in the HR data. We normalize velocities by the average center-line velocity, U_0, and pressures by U_0^2. As a result, the velocity components are roughly ≤ 1 and pressure values $\sim \mathcal{O}(10)$. Snapshots are taken far enough apart in simulation time[5] to reduce temporal correlations between snapshots, which lead to longer training times [7].

We consider a training/dev/test split of 80%/10%/10%, which corresponds to 267/33/33 snapshots of LR data for inputs and 68/8/8 snapshots of HR data for labels. Though the number of LR and HR snapshots are not equal, we emphasize that our goal is to correct the LR data such that the profiles of spatiotemporally averaged[6] quantities (not the point-wise values, nor solely spatially averaged values themselves) match those of the HR data. Due to the temporally consistent nature of the problem, the profiles of the spatially averaged HR data do not change significantly from snapshot to snapshot[7]. Therefore, HR snapshots will be randomly replicated and reused for more than one input data sample to produce 267/33/33 labels for the input.

5 Methods

Table 1: The hyperparameters used and their values. The hyperparameters with the value "Varied" were tuned in the following sections due to their considerable influence on the NN performance. GPU memory constraints predominantly limited the minibatch size to 2 and the number of neurons per hidden layer to 25. The Adam optimizer values are standard.

Architecture	Value	Adam Optimizer	Value	Initialization	Type	Loss Function	Value
Learning rate	Varied	β_1	0.9	Weights	Xavier	λ_p	Varied
# of hidden layers	Varied	β_2	0.999	Biases	Zeros		
Mini-batch size	2	ϵ	10^{-4}				
Neurons per layer	25						

5.1 The Neural Network

We developed a class-based code that employs Tensorflow to create, train, and evaluate NNs with ease. The input to our NN is a single LR snapshot of the flow field unrolled into a 1D array. In general, we follow the NN architecture reported in [8] and use fully-connected layers, ReLU activation functions, and the Adam optimizer with minibatches. Further details on the hyperparameters used, and those tuned, may be found in Table 1. The output of the NN is a "corrected" flow field as well as predicted sub-grid stresses. In short, the NN provides the mapping $\{\tilde{u}_i, \tilde{p}\} \rightarrow \{\hat{u}_i, \hat{p}, \hat{\tau}_{ij}\}$.

[3] LR: $192 \times 192 \times 64$ and HR: $768 \times 768 \times 256$ grid-points in the three coordinate directions.
[4] Spatially averaged flow field profiles to compare with those of the corrected flow fields.
[5] LR snapshots were taken $5T^*$ apart and HR snapshots were taken T^* apart, where T^* is the relevant correlation timescale.
[6] Averaged over spatial dimensions, as well as snapshots (i.e., time)
[7] i.e., $\partial\langle\cdot\rangle/\partial t \sim 0$, where $\langle\cdot\rangle$ denotes a spatial average as defined in Table 2.

Table 2: The nomenclature used to describe the formulated loss function ($i,\ j \in \{1, 2, 3\}$).

Variables	Definition
u_i, p	The true velocity and pressure fields.
$\langle\cdot\rangle$	The averaging operator in x and y, unless otherwise specified.
$\overline{u}_i, \overline{p}$	The exact large-scale fields (i.e., the spatially filtered u_i and p).
τ_{ij}	The sub-filtered stress given by $\overline{u_i u_j} - \overline{u}_i \overline{u}_j$.
$\tilde{u}_i, \tilde{p}$	The inaccurate LES velocity and pressure fields (i.e., inaccurate representations of the large-scale fields due to SGS model inaccuracies and low resolution).
$(\cdot)'$	The fluctuating component with respect to the average (i.e., $(\cdot)' \equiv (\cdot) - \langle(\cdot)\rangle$).
$\hat{(\cdot)}$	An output quantity from the neural network.

5.2 The Loss Function

To have the NN "correct" LR snapshots in a physical manner, such that they match desired spatiotemporally averaged profiles from the HR data, we follow [9] to formulate a physics- and data-based loss function,

$$\mathcal{L} = (1 - \lambda_p)\mathcal{L}_{content} + \lambda_p \mathcal{L}_{physics},$$

which sums losses (weighted by hyperparameter λ_p) pertaining to 1.) the combined residuals of the momentum and mass continuity equations, $\mathcal{L}_{physics} \equiv \mathcal{L}_{mom} + \mathcal{L}_{mass}$, and 2.) the errors from matching averaged quantities and second order statistics with the HR data, $\mathcal{L}_{content} \equiv \mathcal{L}_U + \sum_{(i,j)} \mathcal{L}_{u_i u_j} + \mathcal{L}_P + \sum_{(i,j)} \mathcal{L}_{\tau_{ij}}$.[8] The partial losses are defined with the following:[9]

- $\mathcal{L}_{mom} = \text{MS}\Big(\frac{\partial^2(\hat{u}_i\hat{u}_j)}{\partial x_i \partial x_j} + \frac{\partial^2 \hat{p}}{\partial x_i \partial x_i} + \frac{\partial^2 \hat{\tau}_{ij}}{\partial x_i \partial x_j}\Big), \qquad \mathcal{L}_{mass} = \text{MS}\Big(\frac{\partial \hat{u}_i}{\partial x_i}\Big),$
- $\mathcal{L}_U = \text{MS}\Big(\langle u\rangle - \langle\hat{u}\rangle\Big), \qquad \mathcal{L}_P = \text{MS}\Big(\langle p\rangle - \langle\hat{p}\rangle\Big),$
- $\mathcal{L}_{\tau_{ij}} = \text{MS}\Big(\langle\tau_{ij}\rangle - \langle\hat{\tau}_{ij}\rangle\Big), \qquad \mathcal{L}_{u_i u_j} = \text{MS}\Big(\langle u_i' u_j'\rangle - \langle\hat{u}_i'\hat{u}_j'\rangle\Big),$

where the variables are described in Table 2.

5.3 Training Algorithm

After the NN predicts the corrected flow field, the loss is evaluated with the computation of the spatial averages, fluctuating components, and derivatives. While averaging in the x- and y-directions is trivial, the fluctuating components and sub-filter stress is computed using the definitions in Table 2. Initially, the derivatives are computed using Fast Fourier transforms in the x- and y-directions, and a 6th-order compact difference scheme in the z-direction. However, due to the high computational cost, convolutional filters are quickly adopted for computing the derivatives, which accelerate the computation by a factor of 10. With the desired averages, quantities, and derivatives for a single output, $\mathcal{L}_{physics}$ is evaluated from the residuals of the momentum and continuity equations, while $\mathcal{L}_{content}$ is evaluated using the calculated quantities and labels for each training sample.

6 The Evaluation Metrics

The main evaluation criterion of our NN considers accuracy and computational cost. We desire a maximum training time of 7 hours (satisficing metric). After training, we will apply our NN to the test set and compute spatiotemporally averaged profiles from the corrected flow fields. These profiles will then be compared with those produced from the test labels (i.e., the HR data). Upon finding the profiles of the corrected flow fields within a standard deviation of those produced with the HR data, we will consider our NN a success.

Additionally, we will qualitatively consider the visual appearance of the corrected flow fields and determine if they are distinguishable from the input data (i.e., LR snapshots). We would also like to consider $\mathcal{L}_{physics}$ as a quantitative measure for how well the corrected flow fields satisfy physical constraints. Ideally, we would minimize $\mathcal{L}_{physics}$ (optimizing metric) and use larger values of λ_p during training to emphasize the physical constraints; however, due to limitations in data, computational resources, and time, we do not expect to zero-out the physics loss.

[8] for $(i, j) \in \{(1,1), (1,2), (1,3), (2,2), (2,3)\}$

[9] $\text{MS}\Big(f\Big) \equiv \frac{1}{N}\sum_{n=1}^{N} f_n^2$, where f_n is the value of f at a discrete point n in the computational domain.

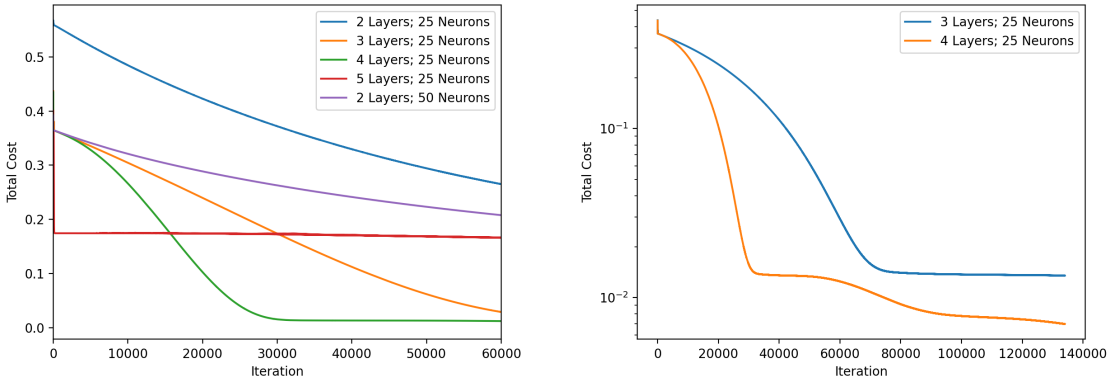

Figure 2: **Left:** The total costs per minibatch iteration found while altering the NN architecture for $\alpha = 1 \times 10^{-5}$ and $\lambda_p = 0.1$. Note that "25 Neurons" corresponds to "25 neurons per layer". **Right:** The NNs with 3 layers, 25 neurons per layer and 4 layers, 25 neurons per layer were trained longer, as they showed the best potential to quickly obtain low cost.

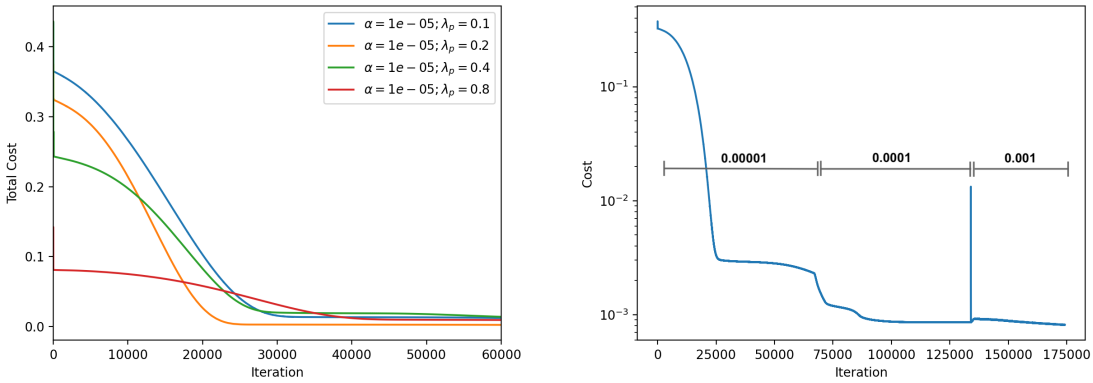

Figure 3: **Left:** The total cost per minibatch iteration found while altering λ_p for a NN with 4 layers, 25 neurons per layer, and $\alpha = 1 \times 10^{-5}$. **Right:** The total cost per minibatch iteration found for a 4 layer NN with 25 neurons per layer, $\lambda_p = 0.2$, and α varies (as specified in the plot) over time.

7 Experiments

7.1 Initial Roadblock

While we verified our derivative operators on fabricated test cases, we found non-zero residuals in the momentum and continuity equations when operating on the ground-truth data (i.e., HR data) near the walls of the domain. We have yet to reconcile this unexpected result, but due to the high residuals of the HR data in the momentum equation, we decided to only keep $\mathcal{L}_{mass}$ in the physics loss function. Since satisfying the continuity equation alone would certainly cause the NN to produce non-physical flow fields, we pursued minimization of the content loss more heavily, as matching the spatial statistics of the HR data offered another path to producing the correct spatiotemporally averaged profiles.

7.2 Hyperparameter Search

After varying other hyperparameters, including neurons per layer, optimizer, and minibatch size, to find a reasonably working NN, we used the hyperparameters in Table 1 and tuned the number of layers, λ_p, and α due to their considerable influence on the performance of the NN.

To begin, we tuned the number of layers for $\lambda_p = 0.1$, anticipating a small λ_p in light of the previously described dilemma and the results reported in [9], and $\alpha = 1 \times 10^{-5}$, anticipating large initial gradients. With limited memory for storing parameters, multiple combinations of neurons per layer and number of layers were tested, but the best (and most time feasible) results were found with using 25 neurons per layer. On the left of Figure 2, the total cost is plotted with minibatch iteration considering different numbers of layers to show the variation in the rate of total cost decay. The NN architectures that decreased the total cost the fastest were then trained for longer on the right of Figure 2. Because 4 layers and 25 neurons per layer provided the best performance, we used this NN architecture in the tests that followed.

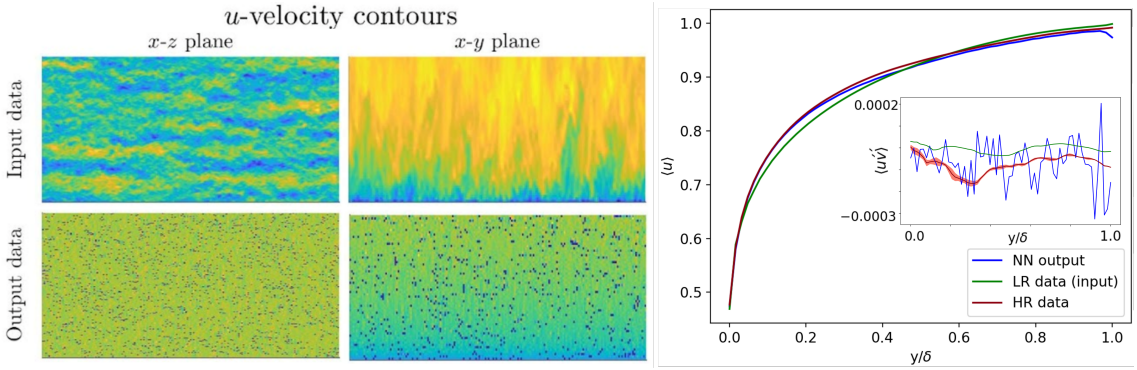

Figure 4: **Left:** Side-by-side comparison of x-z and x-y planes of the input and output x-velocity fields. **Right:** The spatiotemporally averaged profiles of the x-velocity and of $u_1' u_2'$ (inset) for the test set LR data, corrected flow fields, and test set HR data. Point-wise standard deviations of the spatially averaged HR profiles are denote by the red shading in both plots (too small to see in the main plot).

With the confirmed NN architecture, we sought an appropriate value for λ_p. To do this, we again considered $\alpha = 1 \times 10^{-5}$ and limited the training to 60000 minibatch iterations. As shown on the left of Figure 3, $\lambda_p = 0.2$ offers the fastest cost decay and the lowest converged cost for 60000 iterations. Due to limitations in time and resources, we decided $\lambda_p = 0.2$ was close enough to the previously anticipated value $\lambda_p = 0.1$, and did not repeat the previous analysis for convergence of optimal NN architecture and λ_p value.

Finally, instead of finding a single value for α to train with, we followed the advice from class and changed α over time. As shown on the right of Figure 3, we were able to reduce the final cost during training by increasing α over time. This was done for 175000 minibatch iterations (or about 7 hours, which met our satisficing evaluation metric), and resulted in our final trained NN.

8 Results and Discussion

After training, the average loss per training example was 0.001567, and the average loss per test example was 0.001586 (a 1.2% increase over the training loss). This suggested the NN had low variance, but as the results will show, the NN had high bias. As shown on the left of Figure 4, the "corrected" velocity fields of the test set do not appear to be physical when compared with the HR data. Despite achieving a physics loss of nearly 0, the momentum loss was not used, and therefore, the NN was not necessarily guided to make corrections in a physical manner. In comparing the spatiotemporally averaged profile of the corrected x-velocity to that of the HR data (right of Figure 4), we find better agreement than that between the LR and HR profiles (lower L_2 error). However, the profile of the corrected flow field does not match that of the HR data to within a standard deviation of the spatially averaged HR profiles (too small on the plot to see). Additionally, the spatiotemporally averaged profiles of other statistics (e.g., the inset of $\langle u_1' u_2' \rangle$) did not match with those of the HR data; neither qualitatively, nor within a standard deviation. Overall, we believe utilization of the momentum equation, longer training times, and larger NN architectures could help lower the high bias.

Despite the inability to match the HR data, the results from further testing suggested the idea pursued in this work could obtain better results with further tuning, debugging, and longer training times. For example, after scaling the data to keep both mean and fluctuating quantities between $\mathcal{O}(1)$ and $\mathcal{O}(10)$, we trained a NN and found the resulting spatiotemporally averaged profiles evolving towards those of the HR data with less erratic variation in space (see Appendix (Section 11.3)). A similar result was found when using $\tanh$ activation functions instead of ReLU. These results gave confidence that with more time and further investigation, better results could be achieved with this methodology by increasing training time, and perhaps increasing the number of parameters, to lower the high bias.

9 Conclusion/Future Work

In this report, we provided preliminary results for a strategy to enhance the accuracy of coarse-resolution LES simulations using a NN. With an input of LR velocity fields, the NN was trained to correct the fields by minimizing the error in spatially averaged flow field profiles (content loss) and the residuals of the mass continuity equation. Due to issues with the momentum loss and limitations in computational resources, we obtain high bias error, as we could not fully train the NN to output physically convincing velocity fields, nor match all spatiotemporally averaged profiles. However, after completing further testing with altered data scalings, we gained confidence that with longer training times and more computational resources, we could reduce the high bias and match the desired spatiotemporally averaged profiles. Upon also resolving the issues with the momentum loss, we believe the presented methodology could produce a useful NN for enhancing the accuracy of coarse-resolution LES simulations.

10 Contributions

- **Ryan Hass:** Generated all data using *PadeOps*; obtained and debugged derivative operators; wrote and debugged loss function operator; created and presented final report presentation; contributed to submitables (i.e., milestone, final report, etc.).
- **Olivia Martin:** Configured AWS; trained NNs using code and data (i.e., completed hyperparameter search); added modifications to code for NN training/evaluation; contributed to submitables (i.e., milestone, final report, etc.).
- **Kyle Pietrzyk:** Wrote code for NN creation, training, and prediction; assisted in debugging derivative and loss function operators; focused largely on submitables (i.e., milestone, final report, etc.).

11 Appendix

11.1 NN Architecture Search: Additional Content

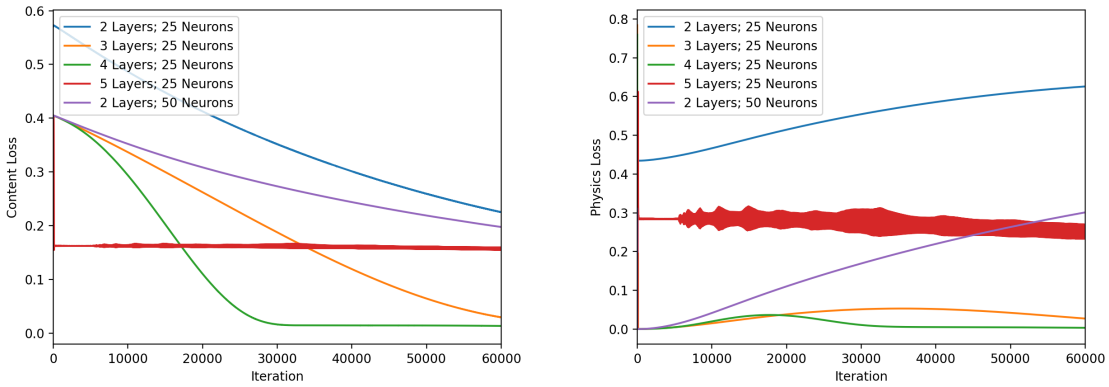

Figure 5: **Left:** The content loss per minibatch iteration found while altering the NN architecture for $\alpha = 1 \times 10^{-5}$ and $\lambda_p = 0.1$. **Right:** The physics loss per minibatch iteration found while altering the NN architecture for $\alpha = 1 \times 10^{-5}$ and $\lambda_p = 0.1$. Note that "25 Neurons" corresponds to "25 neurons per layer".

11.2 λ_p Search: Additional Content

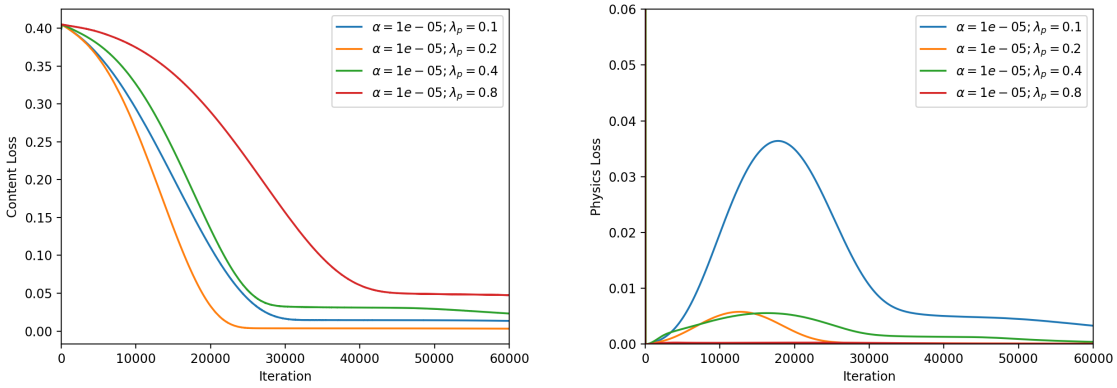

Figure 6: **Left:** The content loss per minibatch iteration found while altering λ_p for a NN with 4 layers, 25 neurons per layer, and $\alpha = 1 \times 10^{-5}$. **Right:** The physics loss per minibatch iteration found while altering λ_p for a NN with 4 layers, 25 neurons per layer, and $\alpha = 1 \times 10^{-5}$.

11.3 Matching Averaged Profiles for Altered Data Scaling

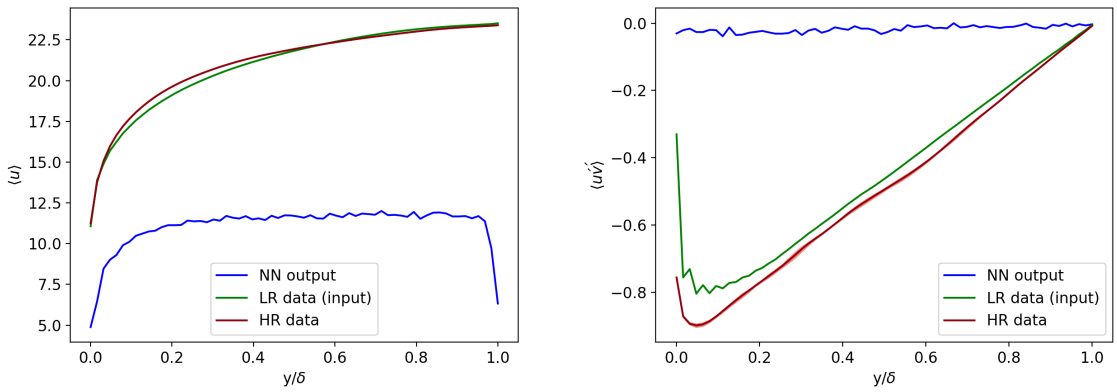

Figure 7: **Left:** The spatiotemporally averaged profiles of the x-velocity for the test set LR data, corrected flow fields, and test set HR data with rescaled velocities. **Right:** The spatiotemporally averaged profiles of $u_1' u_2'$ for the test set LR data, corrected flow fields, and test set HR data with rescaled velocities.

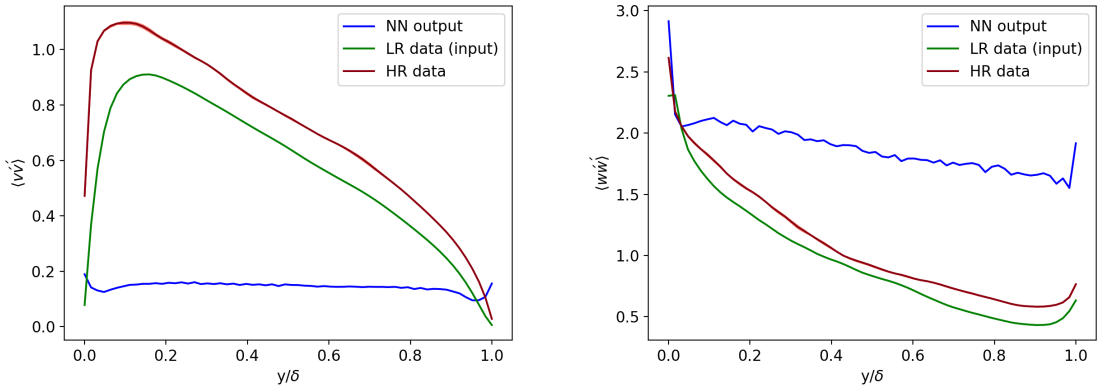

Figure 8: **Left:** The spatiotemporally averaged profiles of $u_2' u_2'$ for the test set LR data, corrected flow fields, and test set HR data with rescaled velocities. **Right:** The spatiotemporally averaged profiles of $u_3' u_3'$ for the test set LR data, corrected flow fields, and test set HR data with rescaled velocities.

References

[1] Nicholas Georgiadis, Donald Rizzetta, and Christer Fureby. "Large-Eddy Simulation: Current Capabilities, Recommended Practices, and Future Research". In: *AIAA Journal* 48 (Aug. 2010), pp. 1772–1784. DOI: 10.2514/1.J050232.

[2] Yang Zhiyin. "Large-eddy simulation: Past, present and the future". In: *Chinese Journal of Aeronautics* 28.1 (2015), pp. 11–24. ISSN: 1000-9361. DOI: https://doi.org/10.1016/j.cja.2014.12.007. URL: https://www.sciencedirect.com/science/article/pii/S1000936114002064.

[3] Javier Jiménez. "Near-wall turbulence". In: *Physics of Fluids* 25.10 (2013). DOI: 101302.

[4] David S. Ching. "Geometric sensitivity, wake dynamics, and machine learning turbulence modeling on a skewed bump". PhD thesis. Stanford University, 2019.

[5] Masataka Gamahara and Yuji Hattori. "Searching for turbulence models by artificial neural network". In: *Physical Review Fluids* 2.5 (2017). DOI: 054604.

[6] Joseph Smagorinsky. "General circulation experiments with the primitive equations: I. The basic experiment". In: *Monthly weather review* 91.3 (1963), pp. 99–164.

[7] Andrea Beck, David Flad, and Claus-Dieter Munz. "Deep neural networks for data-driven LES closure models". In: *Journal of Computational Physics* 398 (2019). DOI: 108910.

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

[9] Akshay Subramaniam et al. "Turbulence Enrichment with Physics-informed Generative Adversarial Network". In: *(Work originated from previous CS230 project)* (2020).

[10] Aditya S. Ghate. "Gabor Mode Enrichment in Large Eddy Simulation of Turbulent Flows". PhD thesis. Stanford University, 2018.