Using a Physics-Informed DeepONet to Predict Turbulent Flow Upstream of a Wind Turbine

Andreas Alexandrou
aandreas@seas.upenn.edu

Shivu Chauhan
shivuc14@seas.upenn.edu

25 April 2023

Abstract

This project focuses on predicting the wind profile downstream of a wind turbine using a physics-informed neural network. The current method of using LIDAR data to calculate the mean wind velocity has limitations as it ignores variations in the wind profile that might affect turbine operation and assumes that the wind profile remains constant. The implemented neural network uses a convolutional neural network and a feed-forward network to predict the flow field at an upstream position. The training data is split into batches and the neural network is trained using the Adam optimizer. The results show that the implemented network can predict reasonable wind profiles. More statistical methods instead of a neural network might prove to be better models.

1 Introduction

Wind forecasting seeks to predict the profile, magnitude, and direction of future wind. These are very important data pertaining to the design and placement of wind turbines and wind farms. Wind forecasting takes place over a wide spectrum of time scales, from predicting large weather patterns across years to predicting how wind will evolve over a few seconds. Somewhat recently, LIDAR devices have started to be placed atop the nacelle pointing upstream to measure the wind directly upstream [1]. These devices inform the controller of the turbine by calculating the mean wind velocity in the stream-wise direction based on the locus of data points they collect. Such LIDAR-informed controllers have been shown to improve the performance of wind turbines [2].

There are a few shortcomings with respect to the current method of using LIDAR data. First, by only calculating the mean wind velocity, the controller is ignorant of variations in the wind profile that might affect turbine operation. Second, it is assumed that the wind profile will not change from the point of measurement to when that wind reaches the turbine, however, it may. This project aimed to take on both of these challenged with a machine learning approach.

Machine learning techniques have been applied to the wind energy field in a number of ways, from controllers, to fault detection, to wind forecasting [3]. Specifically for wind directly upstream of a wind turbine, it is often turbulent and the data that is able to be collected on it is sparse. This makes it an ideal area in which to apply machine learning techniques. Recently work has been done to predict turbulence in 2D flow [4]. Being able to calculate a full 3D flow field from upstream measurements would need to take in to account these turbulent effects. We generated our data using an industry-standard turbulence generator which leverages the Mann model of turbulence. We chose Mann turbulence as it is representative of the wind flow at a number of wind farm sites [5].

In this project, we implemented a physics-informed deep operator network to predict downstream flow from upstream measurements. Our network takes in a 2D velocity profile and predicts the 2D velocity profile at a short downstream distance away. This is intended to mimic the data collection of a LIDAR sensor pointed upstream and the extrapolation of a flow field that a simulation software might do. Mainly using PyTorch, we constructed form the deep operator network and informed it with fluid physics using a PDE residual. We achieved convergent results and are able to predict subsequent velocity profiles. Our code and dataset available at: `https://github.com/AndreasA1/WindPINN`

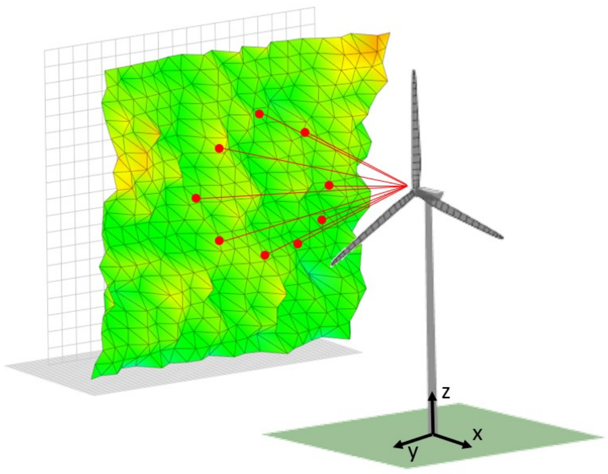


Figure 1: Example LIDAR scanning upstream wind profile

2 Methodology

We implemented a physics-informed deep operator network to predict the flow field at an upstream position. In the DeepONeT The branch net is a convolutional neural network (CNN). The trunk net a feed-forward network (FNN).

For the data processing the binary files are converted into to np.single type and its shape is used to define the variable Nx,Ny,Nz. We normalize the data by subtracting the mean and dividing by standard deviation. The data is split into training and testing for u_{in} and u_{out}.Then we create a 2D mesh grid for the y and z coordinates. This is used to create coords which concatenate the Y and Z arrays into a single array Finally, the u_{in} and coords arrays are reshaped into the appropriate dimensions for use in training and testing models. The u_{out} arrays are also reshaped into 2D arrays. The resulting X_{train} and X_{test} arrays are tuples containing u_{in} and coords. We define the DeepOnet Triple Cartesian product using X_{train}, X_{test}, $y_{train} = u_{train}$, $y_{test} = u_{outtest}$

For the CNN, the input to the network is a 2D tensor with one channel $(in_{channels} = 1)$. The first convolutional layer (conv1) has 8 output channels $(out_{channels} = 8)$, a kernel size of $5x5$ $(kernel_{size} = 3)$, a stride of 1 (stride=1), and zero-padding of 2 $(padding = 2)$. The second convolutional layer (conv2) also has 8 output channels, a kernel size of $5x5$, a stride of 2 $(stride = 2)$, and padding of 1 $(padding = 1)$. After each convolutional layer, a ReLU activation function is applied by default. A max-pooling layer (pool1) with kernel size of $5x5$ is then applied to reduce the spatial dimensions of the output feature maps.The output of the pooling layer is flattened and passed through two fully connected layers with 400 and 1024 output neurons respectively.

The FNN model is a fully connected neural network that takes two input vectors of size 2 and outputs a vector of size 1024. It consists of three fully connected layers with 200, 600, and 1024 output neurons respectively. Tanh activation function is applied to the output of the first two fully connected layers. The two input vectors are combined using the Cartesian product operation before being passed through the network.

We then define a class winddeepONet that consists of a branch network and a trunk network, which are combined by an inner product operation. The forward method takes in three input tensors (u, y, and z) and returns the output of the inner product.

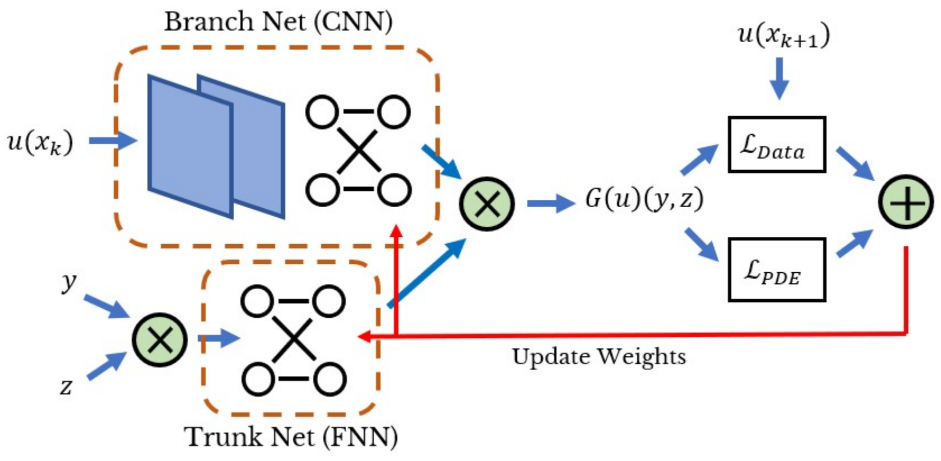


Figure 2: Model schematic

In the class we also define two loss functions: $data_{loss}$ and pde_{loss}. $data_{loss}$ calculates the mean squared error between predicted and target data. pde_{loss} calculates the residual of a partial differential equation (PDE) by sampling a random subset of points, calculating their second-order derivatives, and then averaging the result.

We then define the $total_{loss}$ function that returns the sum of the data loss and the PDE loss, weighted by the weights parameter, which is set to $[1, 1]$ by default. The PDE loss and Data loss are given as:

$$L_{Data} = \frac{1}{N_z}\frac{1}{N_y}\sum_{j=1}^{N_z}\sum_{i=1}^{N_y}|G(u(x_k))(y_i, z_j) - u(x_{k+1}, y_i, z_j)|^2$$

$$L_{PDE} = \frac{1}{m}\frac{1}{n}\sum_{j=1}^{m}\sum_{i=1}^{n}|\frac{\partial^2 G^{(i,j)}(u)(y_i, z_j)}{\partial y^2} + \frac{\partial^2 G^{(i,j)}(u)(y_i, z_j)}{\partial z^2}|$$

This is followed by training. For each iteration, the training data is split into batches of size $batch_{size}$. For each batch, the neural network predicts the output u_{pred} given the input u_{in}. The loss is computed using the $total_{loss}$ function, and the gradients are computed and used to update the neural network weights using the Adam optimizer. The average loss over the batch is stored in a list of loss values. Once the training loop is complete, the trained neural network and the list of loss values are returned from the train function.

After the model is trained, we use the trained model to predict wind velocity on the testing data. The predicted wind velocity is saved in u_pred. Finally, we plot the predicted wind velocity alongside the input wind velocity and the true wind velocity for comparison. A separate function is used to plot the velocity fields. The normalized values of the wind velocity are displayed for each plot.

In generating the data, we used parameters that would be representative of a standard turbulence condition [6]. These parameters are shown in Table 1. They were generated using the Mann Turbulence Generator from HAWC2 [7]. In total, we used 18,432 training pairs and 2,048 testing pairs. By pairs, it is meant an input and output flow field.

Table 1: Parameter values

Parameter	**Value**
Shear anisotropy	3.2
Length Scale	61 m
Energy Dissipation Term	0.11 m$^{\frac{4}{3}}s^{-1}$
y-points (N$_y$)	32
z-points (N$_z$)	32
y-spacing (dl$_y$)	2 m
z-spacing (dl$_z$)	2 m
x-spacing ($\triangle x$)	2 m

3 Results

After training the network, the network was able to output velocity profiles based on the previous velocity profile. In Figure 4, an example for the network input and output is shown. Figure 4a shows the model input. Figure 4b shows the true velocity profile at the next downstream position, and Figure 4c shows the corresponding network prediction for that position. These outputs are discussed more in the following section. Figure 3 shows the training loss of the network as a function of the epoch. The total loss reached a value of just under 0.4. The relative L2 error for the true velocity profile versus the predicted was 0.25.

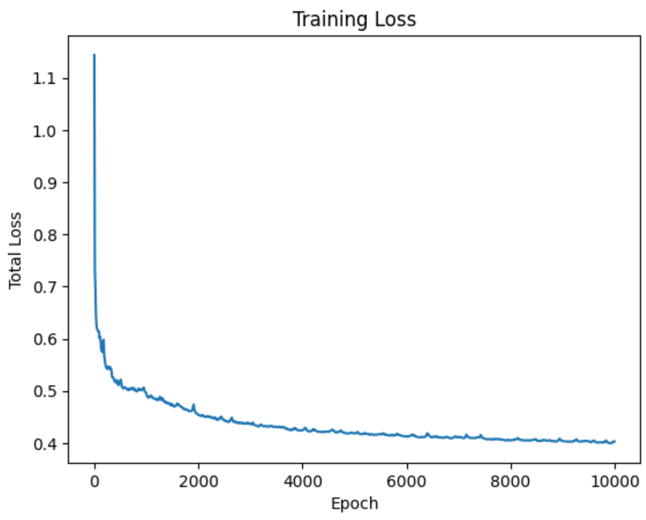
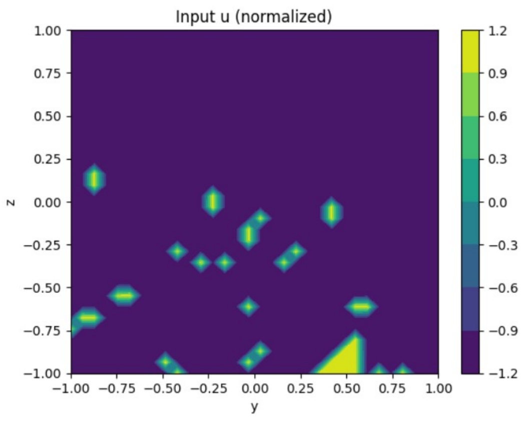
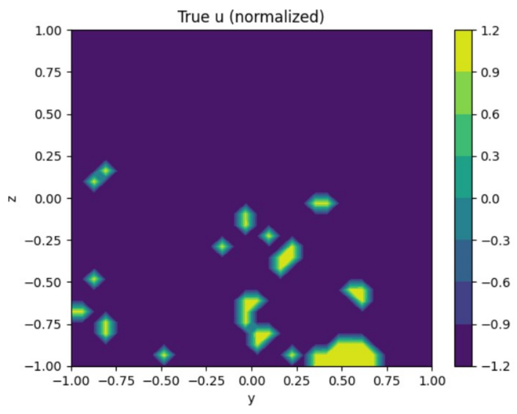


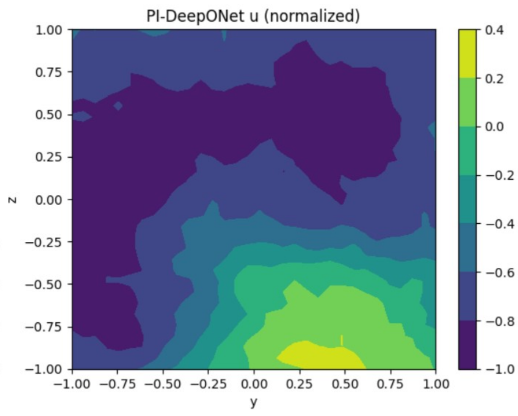
Figure 3: Loss plot during training



(a) Input u



(b) True u



(c) Predicted u

Figure 4: Comparison of true wind profile and model output

4 Discussion and Conclusions

As a predictor of turbulent behavior, the PI-DeepONet performs quite poorly. The network struggles to reproduce the small turbulent excitations contained in its input and which are expected in its output. This might be the result of the CNN we used a CNN in the branch network. A CNN was used here because the velocity profile in the yz-plane is essentially an image and we wanted to preserve the behavior across both y and z without flattening the data. However, the overall performance of the network might suffer from the convolution and pooling in the CNN, as the smaller turbulent effects get combined with and lost in larger areas. However, the network does seem adept at capturing the general contour of the velocity profile. This is shown by the relative L2 error of 0.25, which is not too high but not too low. This suggests that the network is better at capturing the larger scale turbulence. In the future, it would be interesting to see if some residual component could be added to the CNN to reintroduce the smaller-scale turbulence back into the network.

Calculating the PDE loss term by calculating the Hessians and calculating the gradient of the the network weights against the residual is by far the most computationally expensive part of the network. Due to the computation required for even one point in the network output, the PDE residual could not be calculated at every point. This meant that by increasing the number of training data, the PDE loss had to be applied to fewer points, weakening the physical constraint of the PDE. Conversely, the mean value of the PDE loss term was always near zero, meaning that the most computationally intensive operation likely played a minimal role in the training. This is further seen by the velocity profile in Figure 3c. The prediction of the network is generally smooth, meaning the 2nd derivatives of the velocity were small anyways, perhaps a result of the network architecture itself. All this suggests physical constraints should be applied only when its benefits outweigh the additional computational cost.

Some of the discrepancy might have to do with the data being generated by a turbulence model that uses purely statistical methods to generate the velocities distributions, while we are imposing a physical constraint. It would be worthwhile to examine the spectra of the turbulent fields produced by the models. The model of turbulence used to generate the turbulence in this work, the Mann model, analytically predicts a spectral curve of velocity coherence. The turbulence coherence curve calculated from the output of the network could be compared against the analytic expectation. This difference could possibly be implemented in an additional loss term that would potentially be less computationally intensive than the PDE loss term. Ideally, we would have liked to be able to feed our model's outputs back into its input to create a full 3D flow field. However, the error in the prediction compounds so much that this is not yet tenable.

5 Future Work

There are a number of areas into which this work can be extended. The most obvious one would be to integrate it into the workflow of wind-turbine simulators such as OpenFAST. The framework of the PI-DeepONet could be extended to generate time-evolved 3D wind fields (i.e., 4D wind fields). This could help with the computational costs of generating these 4D fields. If we had more time, we would have looked at how different models of turbulence would train on the model, or how the model of turbulence being considered would inform network architecture. More statistical methods instead of a neural network might prove to be better models.

6 References

1. D. Schlipf, P. Grau, S. Raach, R. Duraiski, J. Trierweiler, and P. W. Cheng, “Comparison of linear and nonlinear model predictive control of wind turbines using LIDAR,” 2014 American Control Conference, 2014.

2. B. Han, L. Zhou, and Z. Zhang, “Lidar-assisted radial basis function neural network optimization for wind turbines,” IEEJ Transactions on Electrical and Electronic Engineering, vol. 13, no. 2, pp. 195–200, 2017.

3. Alberto Pliego Marugán, Fausto Pedro García Márquez, Jesus María Pinar Perez, Diego Ruiz-Hernández, ”A survey of artificial neural network in wind energy systems”, Applied Energy, vol. 228, 2018, pp. 1822-1836, ISSN 0306-2619.

4. V. Kag, K. Seshasayanan, and V. Gopinath, “Physics-informed data based neural networks for two-dimensional turbulence,” [2203.02555] Physics-informed data based neural networks for two-dimensional turbulence, 28-May-2022. [Online]. Available: http://arxiv-export3.library.cornell.edu/abs/2203.02555.

5. On turbulence models and lidar measurements for wind turbine control - wes. (n.d.). Retrieved April 26, 2023, from https://wes.copernicus.org/preprints/wes-2021-51/wes-

2021-51.pdf

6. Y. Chen, F. Guo, D. Schlipf, and P. W. Cheng, “Four-dimensional wind field generation for the aeroelastic simulation of wind turbines with Lidars,” Wind Energy Science, 11-Mar-2022. [Online]. Available: https://wes.copernicus.org/articles/7/539/2022/wes-7-539-2022-metrics.html.

7. `https://www.hawc2.dk/install/standalone-mann-generator`