Machine Learning for Flow Evolution Ahead of an Aircraft

a project presented to

The Faculty of the Department of Aerospace Engineering

San Jose State University

in partial fulfillment of the requirements for the degree

Master of Science in Aerospace Engineering

by

Cesar Gonzalez

December 2022

approved by

Dr. Periklis Papadopoulos

Faculty Advisor

ABSTRACT

Machine Learning for Flow Evolution Ahead of an Aircraft

by Cesar Gonzalez

The Navier-Stokes equations are heavily used to simulate flows around aircraft due to their accuracy. However, supercomputers are often needed as the Navier-Stokes equations require vast computational resources to be resolved. Approximations of the Navier-Stokes equations speed up processing time so that commercial hardware can run these simulations, but the computation time remains high. During the design process this slow computation of a flow field may be acceptable, but an airborne aircraft requires prompt knowledge of the flow field to take corrective action. For flow field information to be used on aircraft, faster algorithms must be developed. This purpose of this project is to develop a method to use Lidar airflow measurements ahead of an aircraft to predict the flow field encountered as the aircraft passes through. Machine learning will be used to develop spatiotemporal algorithms capable of interpolating between measurements in space and extrapolating a flow field into the future.

Acknowledgements

The author made use of the SciANN API to program the neural network and data from the Johns Hopkins Turbulence Database. Additionally the video lectures and writings of Steve Brunton of the University of Washington were often referenced. The information regarding data driven mathematics, machine learning, and referenced papers was invaluable.

Table of Contents

List of Figures

Symbols

Symbol	Definition	Units [SI]
L	Number of Layers in Neural Network	---
$\mathcal{L}$	Loss Function	---
p	Pressure	N/m^2
t	Time	seconds
$\boldsymbol{u}$	Velocity vector	m/s
Greek Symbols:		
β	Stan Scaling Term	---
$\boldsymbol{\theta}$	Tunable Parameters of a Neural Network	---
λ	Dynamic Weight of each term	---
μ	Viscosity	Ns/m^2
ν	Kinematic Viscosity= (viscosity μ)/(density ρ)	m^2/s
ρ	Density	kg/m^3
σ	Activation Function	---
Subscripts:		
k	Layer of Neural Network	---
r	Referring to PDE Residual	---
s	Referring to Data Points	---
Superscripts:		
$*$	Nondimensionalized	---
i	ith Neuron in a Layer	---
Acronyms:		
Adam	Adaptive Moment Estimation	---
CFD	Computational Fluid Dynamics	---
DWL	Doppler Wind Lidar	---
JHTD	Johns Hopkins Turbulence Database	---
Lidar	Light Imaging Detection And Ranging	---
L-BFGS-B	Limited Memory Broyden–Fletcher–Goldfarb–Shanno Bound variant	---
MSE	Mean Squared Error	---
NS	Navier-Stokes	---
PDE	Partial Differential Equation	---
PINN	Physics Informed Neural Network	---
ReLU	Rectified Linear Unit	---
RPCA	Robust Principal Control Analysis	---
Stan	Self-Scalable Tanh	---

1. Introduction

1.1 Motivation

Sensing of air currents is an active field of research for commercial passenger aviation due to clear air turbulence being invisible to conventional weather radars installed on current aircraft [1]. When turbulence is encountered by an aircraft, passenger comfort and control of the aircraft are negatively affected. It would be beneficial to the aerospace industry if areas of dangerous airflow could be sensed remotely so that the areas could be avoided. Predicting fluid flow near an aircraft is typically done via computational fluid dynamics (CFD) simulations based on the Navier-Stokes (NS) equations. In flight however, there is little time to be running such expensive simulations. Once solution to this problem is to use machine learning.

Machine learning uses large amounts of data to "learn" how a system behaves using low computation cost models with good accuracy [2]. An existing method of remotely measuring air flow is Doppler Wind Lidar (DWL). Using the Doppler effect, infrared, visible, ultraviolet, or X-rays may emitted from a sensor and the reflected data analyzed to determine flow properties [3,4]. A combination of DWL and machine learning could allow aircraft to predict the flow that will be encountered quick enough for adverse areas to be avoided.

1.2 Literature Review

1.2.1 Doppler Wind Lidar Background

DWL is being actively used and researched in the wind energy sector. Knowing how much energy can be extracted from the wind in an area is a large driver of research into DWL systems [5]. Older research focused on single beam Lidar, which could only determine radial wind velocity relative to a sensor, as opposed to absolute wind speed and direction. This issue was referred to as the "cyclops dilemma" owing to the lack of depth perception from a single sensor. In 2012 it was shown that by using a windmill mounted dual Lidar system (two separate beams with an angle between them) it was possible to "compute estimates of the radial and azimuthal components of the wind field" [5]. That is, by using two separate Lidar beams, 2D wind vector components could at certain points could be found. The problem described by this paper is shown in Figure 1.1. Computation time for 10 iterations, giving under 1 m/s root mean square error was 0.133 seconds, suitable for real time use. A later paper by [6] considered reconstruction of a 3D flow field from Lidar beams varying in vertical angle as well as horizontal angle. In this case the computation time of 7.5 hours using a supercomputer, due to the computational cost of reconstructing turbulent flow using large eddy simulation [6]. As it is unknown whether 3D reconstruction will be similarly expensive when using machine learning, a 2D reconstruction of flow will be focused on for this project.

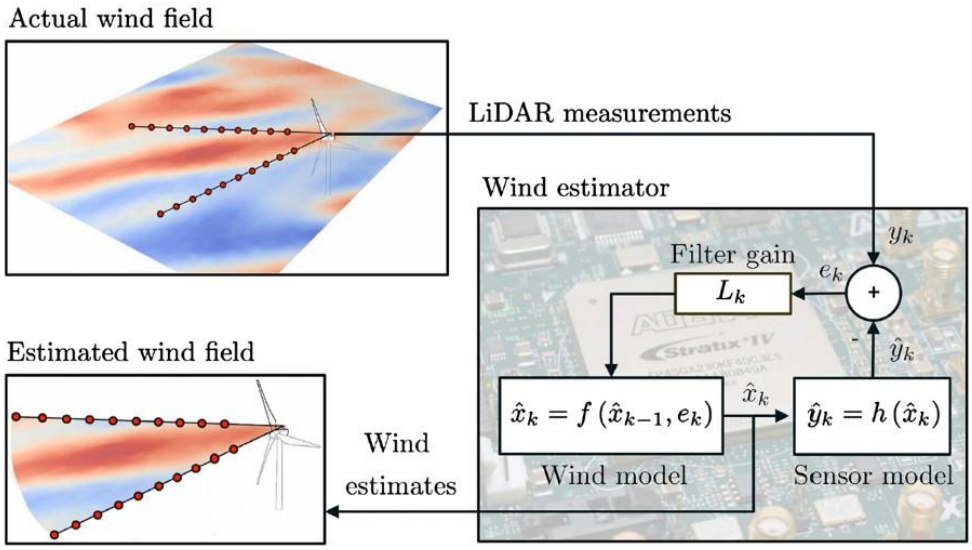


Figure 1.1 – Estimation of 2D flow field from dual Lidar system [5]

These works [5,6] did not make use of machine learning, but instead used a simplified model of the atmospheric boundary layer over flat terrain (surface roughness of 0.1 m [5]). In the case of an aircraft travelling over terrain, such a simplified model would likely fail to generate accurate results. According to [7], "Flows over ridgelines, around terrain, or within turbine wakes may have different vertical components on each side of the measurement volume, causing an error in the estimated velocity". Thus, a different approach is required for a DWL system mounted on a moving aircraft capable of experiencing a variety of flow conditions.

In 2020 the first use of Lidar on a UAV in scientific literature was done by [8]. In this proof of concept, the UAV was attached umbilically to a Lidar base station, with Lidar telescopes mounted to the UAV. This paper focused on the accuracy of an airborne Lidar system vs conventional sonic anemometers (devices that measure wind velocity). An airborne Lidar system had under 5% error compared to a sonic anemometer when measurements are taken outside of the downwash influenced area of 3 meters [8]. Using these results, for this project the virtual Lidar measurements taken will have a random error of within 5% added in for realism.

1.2.2 Machine Learning for Spatial Flow Interpolation and Temporal Flow Extrapolation

Over the course of the last 20 years, machine learning in fluid dynamics has swelled in popularity due the expanded availability of fluid data sets and fast computer hardware [9]. Once implementation of machine learning is in neural networks. Figure 1.2 shows the basic form of a neural network, which is comprised of several interconnected functions. Each function in a circle is called in neuron. Each layer takes the results of the previous layer as inputs to neurons on that layer. The multiple inputs to a neuron are multiple by coefficients called weights, and then have a constant called a bias added to the weighted sum. This is shown in Figure 1.3. Each layer of functions is dependent on the one previous, with more layers allowing for better modeling of nonlinear behavior [2]. A neural network with many layers is referred to as a deep neural network. A neural network is trained using known inputs and outputs. As weights and biases are changed though many iterations, the computer keeps track of which model had the lowest error/loss from the known output. The strength of machine learning is the generalizability of

trained models. For example, a model trained on airfoil pressure and velocity distributions was able predict such distributions over previously unseen airfoils with under 3% error [10]. Due to the number of degrees of freedom present in real life air flows, it is unlikely that all possible flows exist in fluid data sets. Therefore, a method of predicting novel flow field patterns from limited measurements is needed. In this project machine learning will be used to fulfill this task.

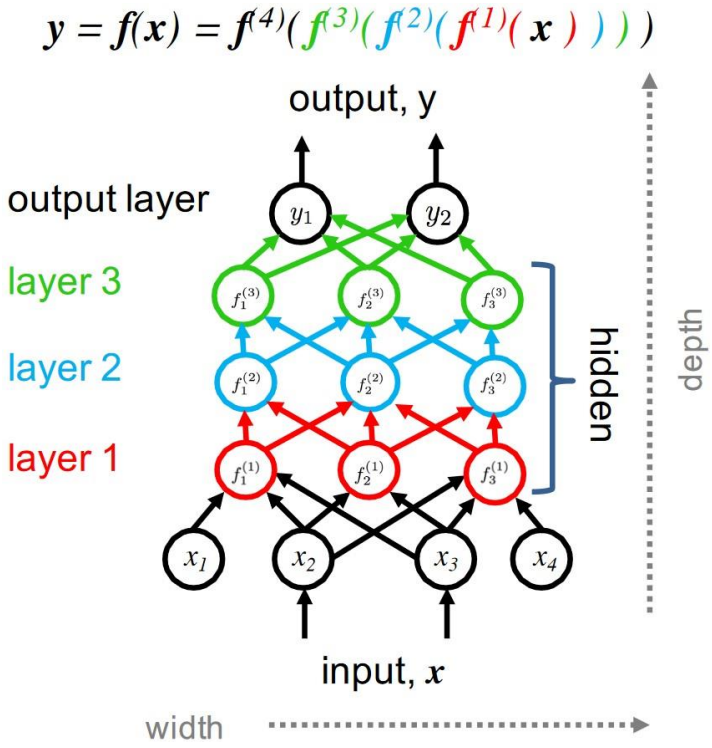


Figure 1.2 – Simple neural network with three hidden (in between) layers [2]

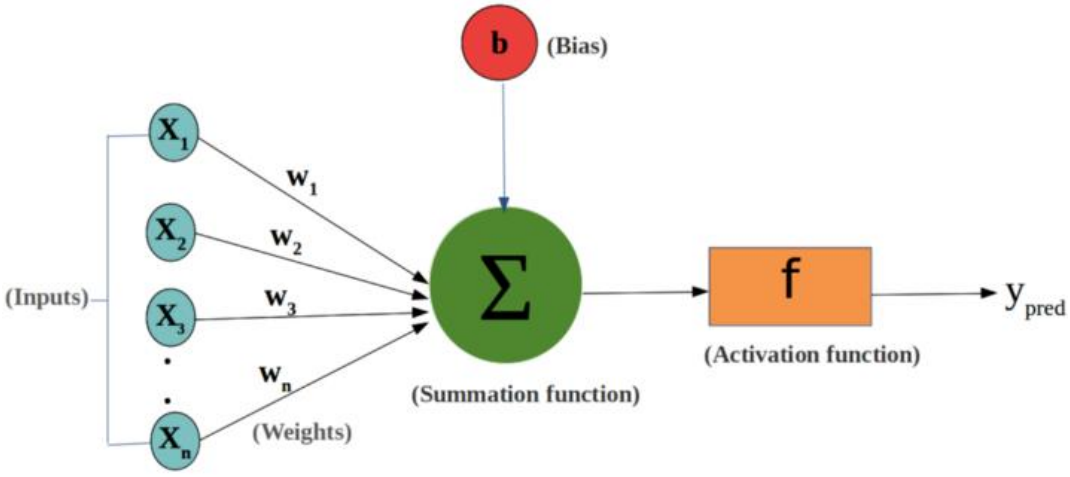


Figure 1.3 – How multiple inputs are handled in a neuron [11]

When viewing a fluid field changing in time, two questions arise- what occurs between measurement points, and what will occur in the future as the flow evolves in time. In this project

both questions are of interest. A moving aircraft will only have time to take a coarse grid of the flow, and must then determine if the flow conditions in an area will become adverse in the time it takes the aircraft to arrive.

The problem of data interpolation depends on the quality of measurements given. In the case of a Lidar system, an error of about 5% is to be expected from commercial hardware [8]. Flow reconstructed from this data will likely be noisy. Using machine learning, a neural network can be developed to extract more accurate data by removing noise. Machine learning using robust principal component analysis (RPCA) has been used to "uncover and isolate the dominant low-rank coherent structures from sparse outliers" [12]. Underlying patterns in fluid flows can be identified by a computer to remove noise, even if severe. This is shown in Figure 1.4. Since cheaper Lidar systems are noisier and less accurate, this approach may be needed for use in general aviation aircraft.

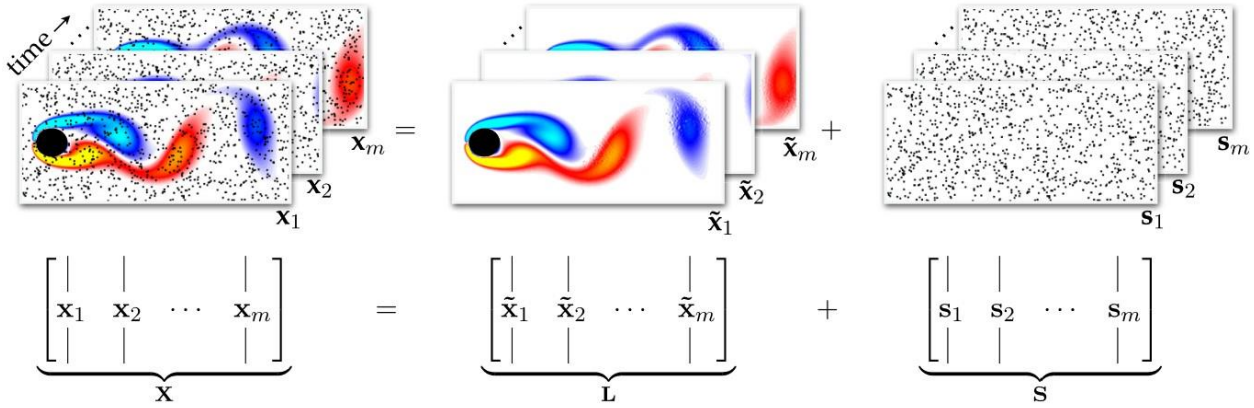


Figure 1.4 – Removal of noise with RPCA via machine learning [12]

In October 2021, the first paper using physical laws with machine learning to predict wind flow was published [13]. In this paper a flow field was simulated, and virtual Lidar measurements taken. A neural network was set up to predict a 2D flow field from these coarse measurements. Vitally, the output to this neural network was compared to both the true Navier-Stokes solution and the Lidar measurement at each point. This meant that the neural network learned to obey conservation of mass, momentum, and energy as it was trained. Figure 1.5 shows the setup of the neural network used in [13], capable of reconstructing an 81 x 41 grid in 0.012 seconds using an Nvidia K80 GPU. Due to the high speed and accuracy of this physics informed neural network (PINN), a similar approach will be used in this project. More computationally expensive machine learning interpolation work has been done in [14]. In this paper a turbulent 3D flow field is simulated with large eddy simulation. Coarse data points are taken from this flow field and closure models created to reduce error compared to direct numerical simulation. The end result is a reduction in discrepancies between turbulence scales at orders of magnitude less computational cost [14]. Unfortunately, the process described in this paper is still far too expensive for real time use, requiring a supercomputer cluster to be used.

Flow interpolation from points is discussed in [13] and flow extrapolation in time is discussed in [15]. As discussed in the previous paragraph, [13] uses a physics informed neural network, resulting in high accuracy compared to the true Naver-Stokes equations. Since the

neural network is computationally light, each 0.02 second time step can be computed in 0.012 seconds on average. This allows for prediction of flow ahead of a DWL system. Figure 1.6 from [13] shows high speed flow moving from the left side of the 240-meter domain to the right side over the course of 20 seconds. The method proposed in this paper has good agreement between the prediction and the actual simulated flow. In [15], the acceleration of flow field computation via machine learning is discussed. In direct numerical simulation the Navier-Stokes equations are directly used on a grid. In practice the Navier-Stokes equation cannot be calculated continuously, there must be a finite grid which gives a certain amount of error. A CFD practitioner must choose an acceptable amount of error in the form of lost vorticity correlation. The result of [15] is shown in Figure 1.7, and is quite promising. Using machine learning, a grid ~10 times coarser can be calculated 86 times faster while having the same accuracy as direct numerical simulation. This paper demonstrates that real time flow prediction is a good use case for machine learning.

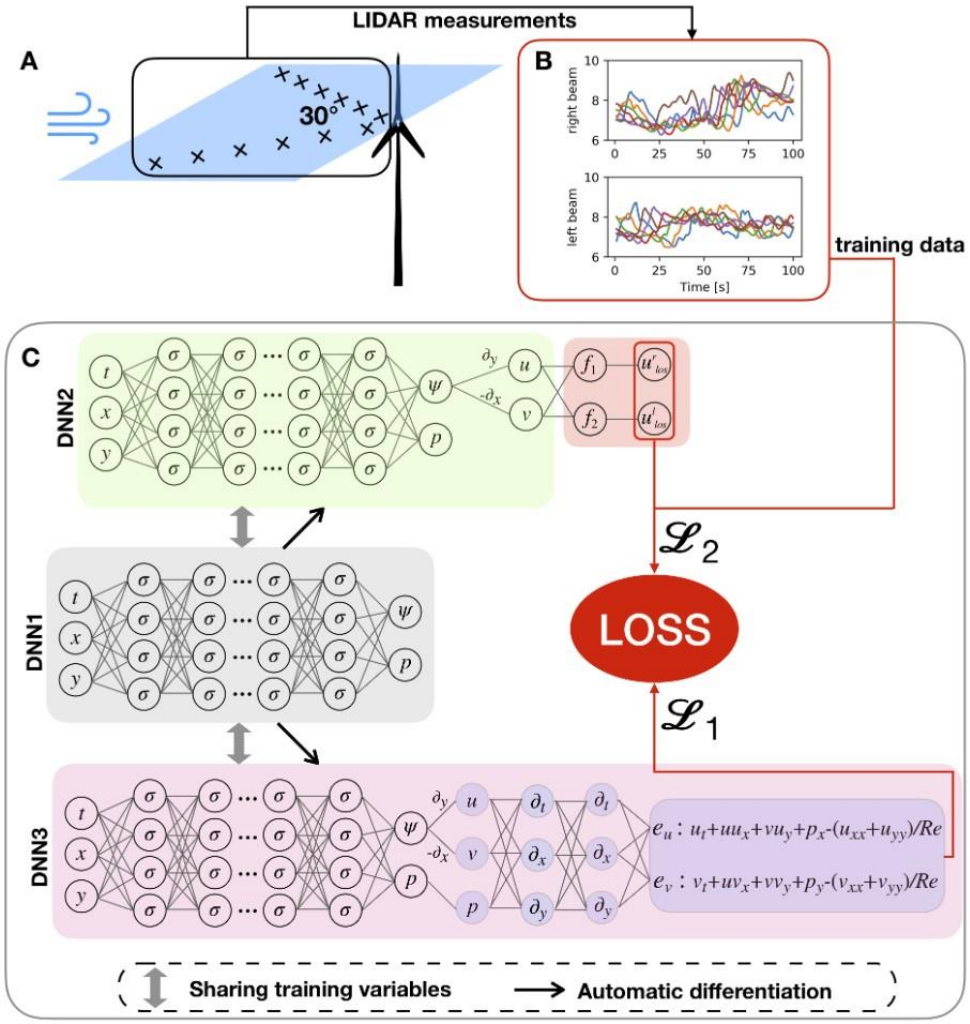


Figure 1.5 – Physics informed deep neural network [13]

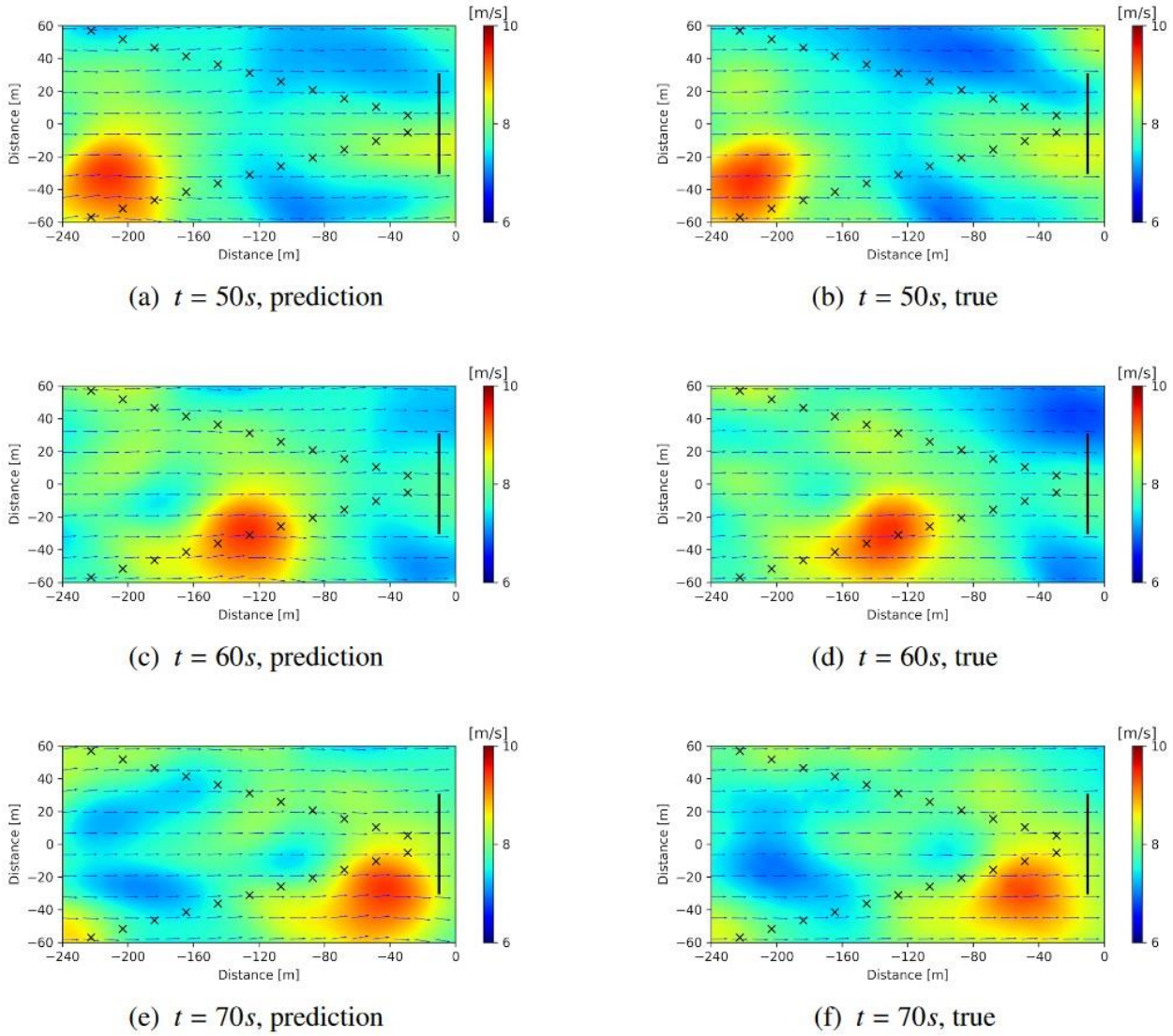


Figure 1.6 – Predicted 2D flow vs true simulated atmospheric flow [13]

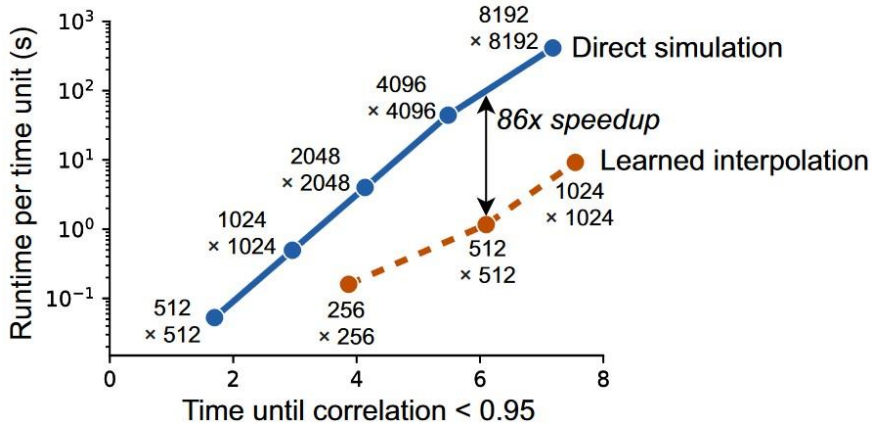


Figure 1.7 – Speedup from use of machine learning instead of direct simulation [15]

1.3 Proposal

This project aims to use measurements from a Lidar system integrated onto an aircraft to reconstruct the flow field ahead of the aircraft. Existing airborne weather radar cannot detect clear air turbulence [1], and the coarseness of Lidar measurements means that small pockets of adverse flow cannot be detected easily by a Lidar system [3]. As airborne Doppler Wind Lidar systems have been tested before [1,8], this type of remote flow measurement will be used for this project. As data from a lightweight DWL will be noisy and coarse, physics informed machine learning will be used to reconstruct the flow field in real time. The machine learning algorithm will closely approximate the Navier-Stokes equations to ensure the physical accuracy of the predicted flow at each time step. The resulting algorithm will be computationally cheap enough to run in real time using commercial hardware.

1.4 Methodology

A balance will need to be found between the Lidar technology, aircraft velocity, and machine learning algorithm chosen for use on the aircraft. The reason being that both data measurement time and algorithm computational time must be considered for a moving aircraft. As a moving aircraft will "run into" a flow field after a certain time from measurement, the change in the flow over time must be considered.

No sensor system is perfect. In the case of DWL sensors, there are limitations regarding range, spatial resolution (coarseness of data points), and accuracy. The QinetiQ 1.5 μm Lidar is one such DWL system. It is relatively low power, using 50 μJ per pulse, but has a range of only 0.5 km, a spatial resolution of 25 m, and an accuracy of ±0.5 m/s [3]. For such data to be usable to an aircraft, data cleanup and interpolation between points will be required. For data cleanup, work by [12] shows that even with heavy "salt and pepper" corruption, underlying flow properties can be found through machine learning. The method discussed in [12] will be adapted for use in this project.

Once air flow data has been cleaned, interpolation between points at that time must occur so that extrapolation in time can be done at a later stage. The problem of interpolation between coarse fluid measurements in real time has been investigated by [13]. This paper used machine learning to reconstruct the flow in a flat plane going outwards from Lidar measurements. The resulting physics informed deep learning algorithm was capable of resolving a 2D 81 x 41 grid from 22 Lidar measurements in 0.012 seconds when running on an Nvidia K80 GPU [13]. This paper was published in October 2021 and is the only real time deep learning flow predicter the author of this report was able to find. Due to the speed and low computational cost of the method described in [13], the method will be adapted for used in this project.

2. Computational Fluid Dynamics Setup

2.1 CFD Mathematical Background

The Navier-Stokes (NS) equations describe fluid flow through the conservation of mass, momentum, and energy. In a nonreacting flow all three must be conserved for the flow to be real, as there is no change in the three quantities. In this project only low speed air flow commonly encountered by general aviation aircraft will be considered. While this does limit the scope of this project, smaller general aviation aircraft are more affected by adverse flow than more massive aircraft. At low speeds the incompressible NS equation in (2.1) can be used, saving compute resources. As there are limited compute resources available to the author, only the incompressible isotropic NS equations will be considered. The isotropic assumption allows for density and viscosity to be assumed constant in every direction.

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} - \nu \nabla^2 \boldsymbol{u} = -\frac{1}{\rho}\nabla p \tag{2.1}$$

2.2 Computational Approximation Model

2.2.1 Training Data

The layer of the atmosphere in which general aviation aircraft fly is the troposphere. As this layer is the closest to the surface of the Earth, air density is high enough for the air to be considered continuous. This assumption in conjunction with the incompressible and isotropic assumptions allows for the use of the Johns Hopkins Turbulence Database (JHTD) [16].

For a neural network to be trained, large amounts of data are required so that the network can "learn" fluid behavior. Producing large amounts of high-fidelity fluid simulations is prohibitively expensive due to computational cost. In [13], it took 256 processors in a high performance computing cluster 2 hours to run a single large scale air flow simulation. To rectify this issue, the JHTD is used in this project. This database contains the results of direct numerical simulation of fluid flow, as shown in Figure 2.1. In direct numerical simulation, all scales of turbulence are simulated- down to the Kolmogorov dissipative scales [17]. At the Kolmogorov microscales turbulent kinetic energy is converted to heat, preventing the formation of smaller eddies. That is, flow does not become more physically complicated under these scales. This makes the JHTD datasets highly accurate and therefore good for training a neural network. The dataset chosen for this project is the "Forced Isotropic Turbulence Data Set", which is similar to turbulent air flow. The properties of this dataset are shown in Figure 2.2. As the full dataset is several dozen terabytes, subsets of the simulated cube of flow are used. In [13], the 2D plane from which data was taken was of dimensions 81x41, so these dimensions will be used for the data slices from the JHTD dataset. Small slices allow the neural network to be trained at many different locations at different times. This variety improves the comprehensiveness of training. Following the setup of shown in Figure *1.5*, data points are taken in two lines 15° off the centerline (30° separation from each other). The nearest data points to these lines are used when a data point does not lie directly on one of these lines.

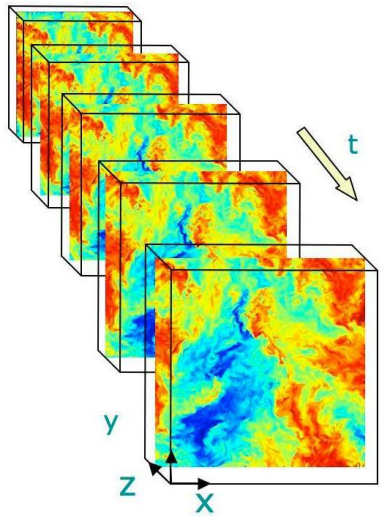


Figure 2.1 – Snapshots of direct numerical simulation of turbulent flow [16]

Simulation parameters:
Domain: 2π x 2π x 2π (i.e. range of x_1 , x_2 and x_3 is [0,2π])
Grid: 1024^3
Viscosity (ν) = 0.000185
Simulation time-step Δt = 0.0002
Data are stored separated by δt = 0.002 (i.e. every 10 DNS time-steps is stored)
Time stored: between t=0 and 10.056 (5028 time samples separated by δt)

Statistical characteristics of turbulence, time averaged over t=0 and 10.056:

Total kinetic energy, $E_{tot}=\frac{1}{2}\langle u_i u_i\rangle$:	$E_{tot}=0.705$
Dissipation, $\varepsilon=2\nu\langle S_{ij}S_{ij}\rangle$:	$\varepsilon=0.103$
Rms velocity, $u'=\sqrt{(2/3)E_{tot}}$:	$u'=0.686$
Taylor Micro. Scale $\lambda=\sqrt{15\nu u'^2/\varepsilon}$:	$\lambda=0.113$
Taylor-scale Reynolds #, $R_\lambda=u'\lambda/\nu$:	$R_\lambda=418$
Kolmogorov time scale $\tau_\eta=\sqrt{\frac{\nu}{\varepsilon}}$:	$\tau_\eta=0.0424$
Kolmogorov length scale $\eta=\nu^{3/4}\varepsilon^{-1/4}$:	$\eta=0.00280$
Integral scale: $L=\frac{\pi}{2u'^2}\int\frac{E(k)}{k}dk$:	$L=1.364$
Large eddy turnover time: $T_L=L/u'$:	$T_L=1.99$

Figure 2.2 – JHTD forced isotropic turbulence dataset parameters [16]

2.2.2 Test Data

Separate from the training data is the test data. Once the neural network is trained, the performance of the network needs to be determined. While large amounts of training data are required to train the neural network, only a few simulations are required for testing. The testing data is also derived from the JHTD dataset at random times and locations, with no overlap with training data. This lack of overlap is critical to prevent overtraining the network to specific data. This should prove sufficient to compare the neural network's prediction with the simulation.

3. Physics Informed Neural Network Setup

3.1 PINN Mathematical Background

A neural network is a series of interconnected functions that take in inputs and pass on those inputs to the next layer, as shown in Figure 1.2. A standard neural network does not possess knowledge of physical laws, simply adjusting each layer of functions until the output matches what is expected. This can lead to a neural network only following conservations laws in certain cases. In a physics informed neural network (PINN), the physics of the problem is embedded within the network. PINNs work by comparing their output to the true PDE. This calculated PDE residual is then backpropagated through the network so that weights and biases can be updated. The goal during training is to adjust the weights and biases of the PINN so that error (the residual) is minimized. In the case of fluid flow, these are the NS equations. As shown in Figure 1.5, the output to the neural network at each iteration is compared to the incompressible NS equations and Lidar measurements.

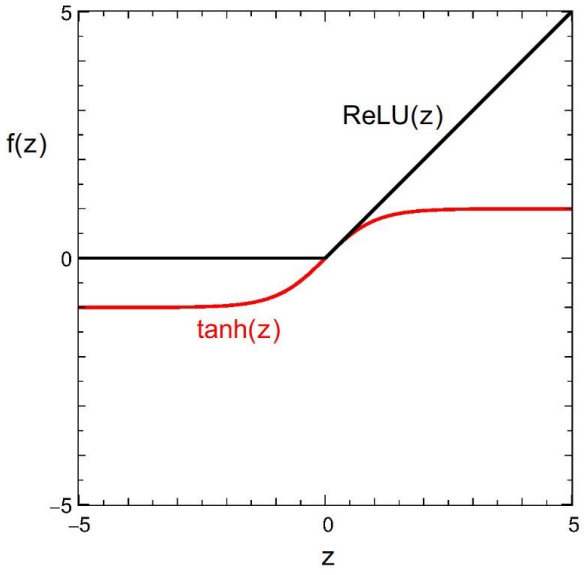


Figure 3.1 – Comparison of ReLU and tanh activation functions [2]

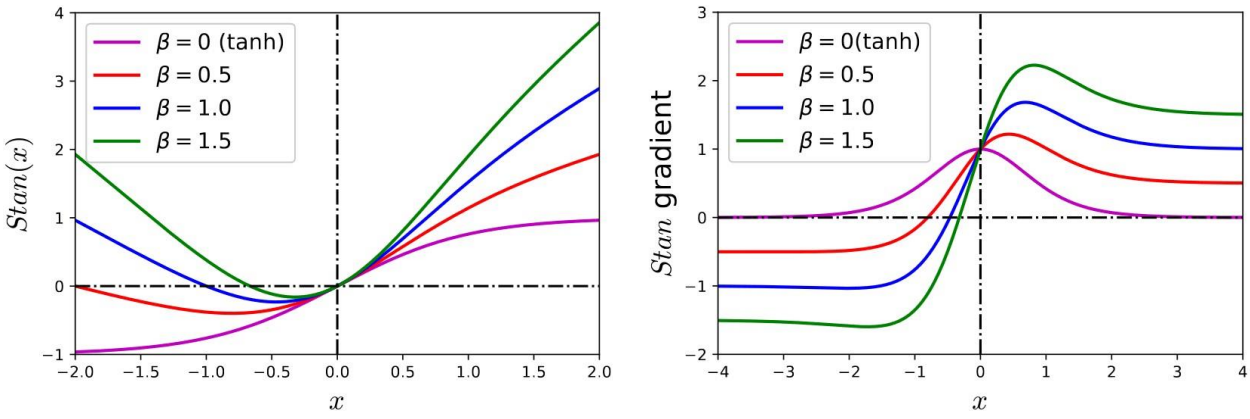
The function chosen for use in each node of a neural network is known as the activation function. An activation function determines what value is passed on to each layer based the weights (coefficients) and biases (added constants). The activation function is chosen by the practitioner, while the weights and biases are changed with each iteration to reduce error. Several activation functions exist, with sigmoidal functions and tanh(z) being the standard until recently. In this past decade the rectified linear unit (ReLU) has become the most popular activation function due to solving the vanishing gradient problem [2]. Sigmoidal functions only have a range of -1 or 0 to 1, so extreme weights and biases are not able to affect the overall network very much, slowing down training and reducing accuracy. ReLU is a simple linear (mx+b) function with a minimum value of zero, where values below zero are clamped to be zero. This is shown in Figure 3.1. ReLU continues to increase to positive infinity, so there continues to be a gradient for the positive half of the function. The ReLU approach gives an activation function that is nonlinear overall (vital for an accurate neural network [2]) and easily differentiable.

Unfortunately, ReLU is not a good choice of activation function for PINNs. According to [18,19] PINNs require smooth (differentiable throughout) activation functions to ever converge to the exact solution. Since ReLU is not smooth when it transitions from flat to a sloped line, a PINN using ReLU will not converge even if given infinite time. Another activation function than is standard is required.

According to [20,21], activation functions used for PINNs must follow three conditions:

- Must be smooth. If the activation function is not differentiable throughout, then the PINN will not ever converge to the exact solution.
- Must not have vanishing gradients. If the derivative of the function as x approaches infinity is zero, then the PINN will require infinite time to train.
- Must be centered about zero. There must be both positive and negative gradient for the weights and biases to be both increased and decreased. If the function center is not zero then the PINN will be optimized to be more positive or negative than the exact solution.

ReLU fails all three criteria, while the tanh function suffers from vanishing gradients due to the asymptotes at -1 and 1. Through literature review, the author found two activation functions proposed in 2022 that fulfill all three conditions and are designed for PINNs. [20] proposes the Self-scalable Tanh (Stan) function, which is shown in Figure 3.2. The value of β is trained per neuron and helps the PINN scale across orders of magnitude. This is especially useful for the NS equations which have multiple turbulence scales. [22] proposes the Rowdy modification of activation functions. Rowdy is not an activation function itself but a modification that can be applied to any general activation function. Rowdy adds sinusoidal noise to existing activation functions to ensure smoothness and prevent saturation at extreme values. An example of different levels of the Rowdy modification on ReLU is shown in Figure 3.3. As more sinusoidal noise is added, the gradient of the overall function is increased.



(a) Stan activation function (b) Stan activation function gradient

Figure 3.2 – Different levels of Stan activation function and gradient (derivative) [20]

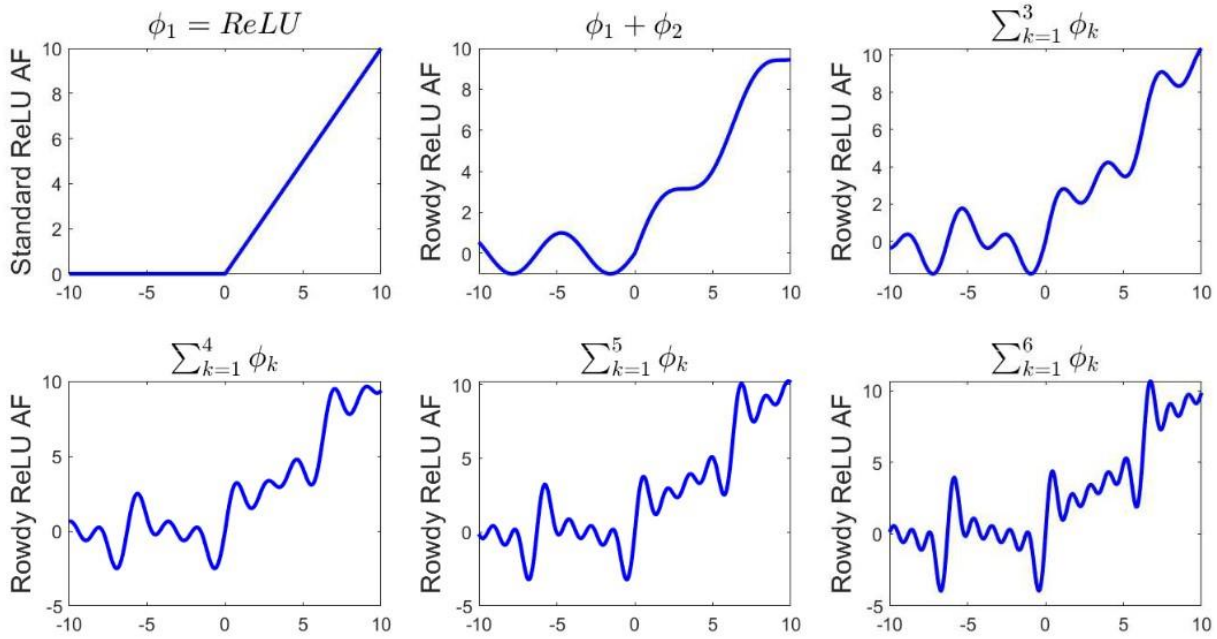


Figure 3.3 – Levels of Rowdy modification of ReLU [22]

When deciding between the Stan and Rowdy one factor stood out – computational cost. Rowdy adds many sinusoidal waves to an activation function, resulting in a difficult to compute derivative. [22] states that 9th order Rowdy required 75% more time to train than tanh. By contrast, the derivative of Stan is easier to compute than Rowdy due to multiple sine terms not being added on. It is unknown which approach gives more accurate results, as both Stan and Rowdy are new enough to not have been compared yet. With limited time to complete this project, the author has elected to use Stan.

With the PINN structure determined by Figure 1.5 and the activation function chosen as Stan [20], the loss function and optimizer are the last details to determine. During training the loss function is the value to minimize, while the optimizer is what determines how the weights and biases are changed to accomplish this. In this project the goal is for the PINN to predict fluid flow, so the discrepancy between predicted and actual velocity values at certain points will make up the loss function. The mean squared error (MSE) shown in (3.1) is what is typically used in neural networks [2,9,13,20]. Due to the common inclusion of MSE in machine learning frameworks, this loss function has been chosen for use in this project.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_{i,predict} - y_{i,actual}\right)^2 \tag{3.1}$$

For improved optimization during training, adaptive weights for multiple terms in the incompressible NS equations have been included in the neural network. During training, gradient descent is used to determine how to adjust the weights of the neurons. In the case of the NS equations, each term contributes to the loss function to be optimized. According to [23], improved accuracy can be gained by having each loss term have a variable weight. In a standard PINN, there may be terms that are more variable than others, and so small changes to these terms

causes a greater change in the overall loss. This can cause a PINN to focus only on these terms and ignore the other smaller terms that do not contribute as much. For a PINN to be physically accurate, all terms must replicated accurately, not simply the most chaotic ones. For this reason the Gaussian Process adaptive weights approach from [23] is used in this project. Equations relating x-velocity u, y-velocity v, pressure P, and stream function ψ to true values given by data are used, as shown in Figure 3.4.

All optimizers have strengths and weaknesses, which makes choosing an optimizer difficult. [18] tested different permutations (order mattered) of optimizers at different epochs (iterations of training done so far) to find the best permutation for use in PINNs. Of the optimizers tested, the best permutation was found to be Adaptive Moment Estimation (Adam) followed by Limited Memory Broyden–Fletcher–Goldfarb–Shanno Bound–variant (L-BFGS-B) [24,25]. The L-BFGS-B optimizer is lightweight computationally but tends to get caught in local minimums during early training, which limits error reduction. When solely the L-BFGS-B optimizer is used a PINN trains quickly, but has high error. The Adam optimizer avoids local minimums for longer, but also plateaus. When solely the Adam optimizer is used a PINN takes longer to train and error does not decrease even if the number of epochs is increased. [18] shows than when Adam is used first, followed by L-BFGS-B, error can be reduced by 10-100 times in a PINN vs only using a single optimizer. In essence, using Adam for early training avoids the local minimum problem. The use of L-BFGS-B after the error first plateaus then allows for error to be further decreased. Due to the effectiveness of this approach in [18], the Adam then L-BFGS-B permutation will be used. These optimizers will affect the weights and biases of the network.

3.2 Neural Network Model and Training

The neural network framework SciANN was used to program this project [26]. In addition to ease of use, the framework contains the Gaussian Process constraint weigher as an option for the neural network [23]. Custom activation functions were also possible to import easily, a necessity for the use of Stan [20]. The general form for the code is based off the NS example is the SciANN applications repository [27].

The activation function used in the network is Stan, shown in (3.2) [20]. In this function k is the layer of the neural network, L is the total number of layers, i is the numbered neuron of the layer, and N_k is the number of neurons per layer. The reason that $k < L$ is that the final layer of a neural network is the output layer, which contains the results instead of an activation function. β is a parameter which determines how large of a gradient flow exists for extreme values, which allows for a single PINN to model multiple length scales effectively. It should be noted that if β is zero, then the function is simply tanh, as shown by Figure 3.2. A nonzero β allows for gradients to exist throughout the domain of input values x. Through these gradients and the loss function, an optimizer can determine how to change the weights and biases of a neural network. For this work, a constant value of $\beta = 0.25$ has been selected.

$$\sigma_k^i = \tanh(x) + \beta x * \tanh(x) \;;\; k = 1,2,\dots,L-1 \;;\; i = 1,2,\dots,N_k \tag{3.2}$$

The NS PDE that is used in the PINN is shown in (3.3), with velocity and pressure data originating from the Johns Hopkins Turbulence Database [16]. The MSE loss function used is given by (3.4). The * designates the actual values which come from the data. The term $\boldsymbol{\theta}$ refers

to the tunable parameters of the network, e.g., the weights and biases as shown in Figure 1.3. The groups of terms used for individual weighing are given by Figure 3.4. Terms d1, d2, and d3 are data terms to be compared directly with the actual values. Standard neural networks only have these values, attempting to fit to data regardless of physics. Terms c1, c2, and c3 are the physical equations that the PINN must also follow. Terms c1 and c2 are the NS equations for the x and y respectively, while c3 is the stream function definition. The Gaussian Process approach given by [23] weighs the importance of each constraint in reducing error. The process of determining each weight is given by (3.5). Here n is the number of steps taken between updates of the weights (n number of data points are considered). $\mathcal{L}_{c_{total}}$ is the total loss attributed to physical equations, while $\mathcal{L}_{f_{num}}$ is the loss due to the specific term whose weight is being found.

For PDE of form $\boldsymbol{u}_t + \mathcal{N}[\boldsymbol{u}] = 0 \;;\; t \in [0,T] \;;\; \boldsymbol{x} \in \Omega$

For NS equations shown in (2.1), $\mathcal{N}[\boldsymbol{u}] = (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} - \nu\nabla^2\boldsymbol{u} + \frac{1}{\rho}\nabla p$ (3.3)

Or in scalar form, the NS PDE is $u_t + \left(u * u_x + v * u_y\right) - \nu(u_{xx} - u_{yy}) + P_x = 0$

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N_t}\sum_{a=1}^{N_t}((\boldsymbol{u}_t + \mathcal{N}[\boldsymbol{u}])^* - (\boldsymbol{u}_t + \mathcal{N}[\boldsymbol{u}])_{\boldsymbol{\theta}})^2 \tag{3.4}$$

$$\hat{\lambda}_{f_{num}} = \frac{\max_\theta(\nabla_\theta \mathcal{L}_{c_{total}}(\theta_n))}{mean(\nabla_\theta \mathcal{L}_{f_{num}}(\theta_n)} \tag{3.5}$$

Then update weights with moving average: $\lambda_{f_{num}} = 0.1 * \lambda_{d_{num}} + 0.1 * \hat{\lambda}_{f_{num}}$

```
[ ]  # Define constraints
     d1 = sn.Data(u)
     d2 = sn.Data(v)
     d3 = sn.Data(P)

     c1 = sn.Tie(-p_x, u_t+(u*u_x+v*u_y)-nu*(u_xx+u_yy))#x velocity equation rearranged, must equal 0
     c2 = sn.Tie(-p_y, v_t+(u*v_x+v*v_y)-nu*(v_xx+v_yy))#y velocity equation rearranged, must equal 0
     c3 = sn.Data(u_x + v_y)#streamfunction definition, must equal 0
```

Figure 3.4 – Different constraints to be individually weighed via Gaussian Process

4. Results

Table 4.1 – Comparison of Training Configurations

Case	Network	Data	Epochs	Epochs per adaptive weights updates (Lower # means higher frequency)	Loss (Error)
1	8*[20]	Full	2500	N/A, adaptive weights not used	0.0754
2	8*[20]	Full	2500	20	3.5684e-07
3	8*[20]	Sparse	2500	N/A, adaptive weights not used	0.1762
4	8*[20]	Sparse	2500	20	9.1917e-07
5	11*[120]	Sparse	2500	N/A, adaptive weights not used	0.1586
6	11*[120]	Sparse	2500	125	0.0117
7	11*[120]	Sparse	2500	20	1.807e-07
8	11*[120]	Sparse	5000	25	1.449e-10

All plots shown are from time step 500.

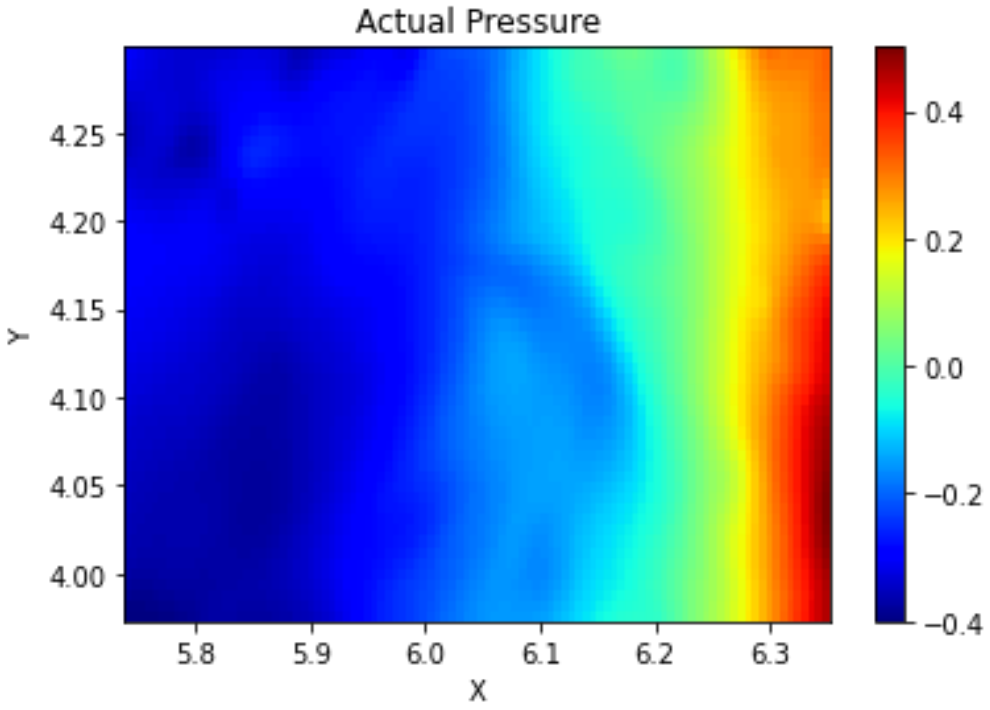


Figure 4.1 – Actual pressure from data

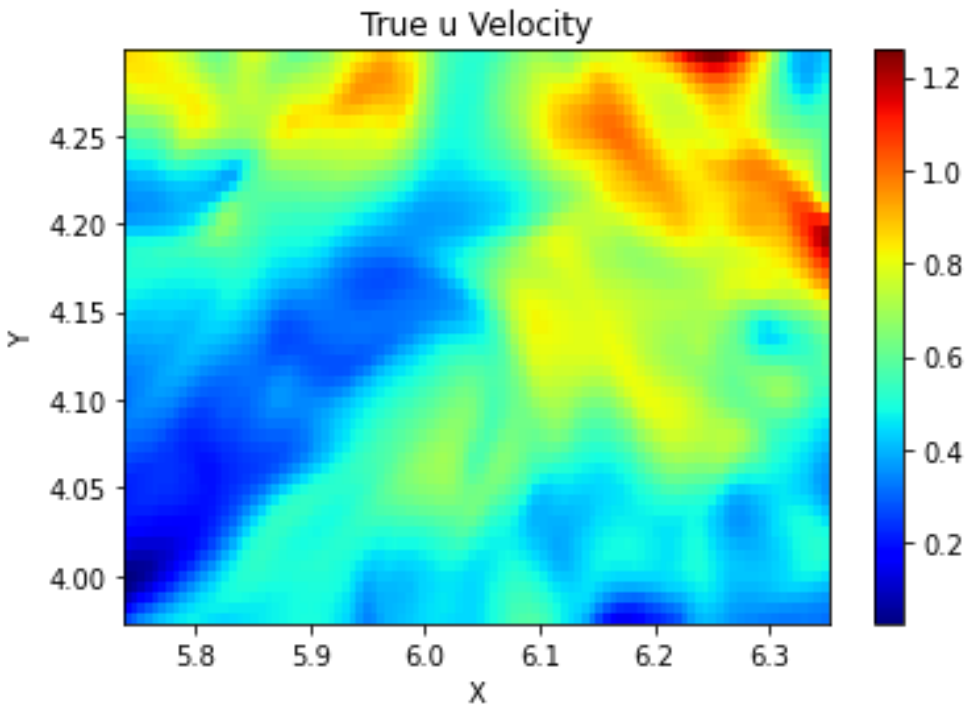


Figure 4.2 – True velocity in x from data

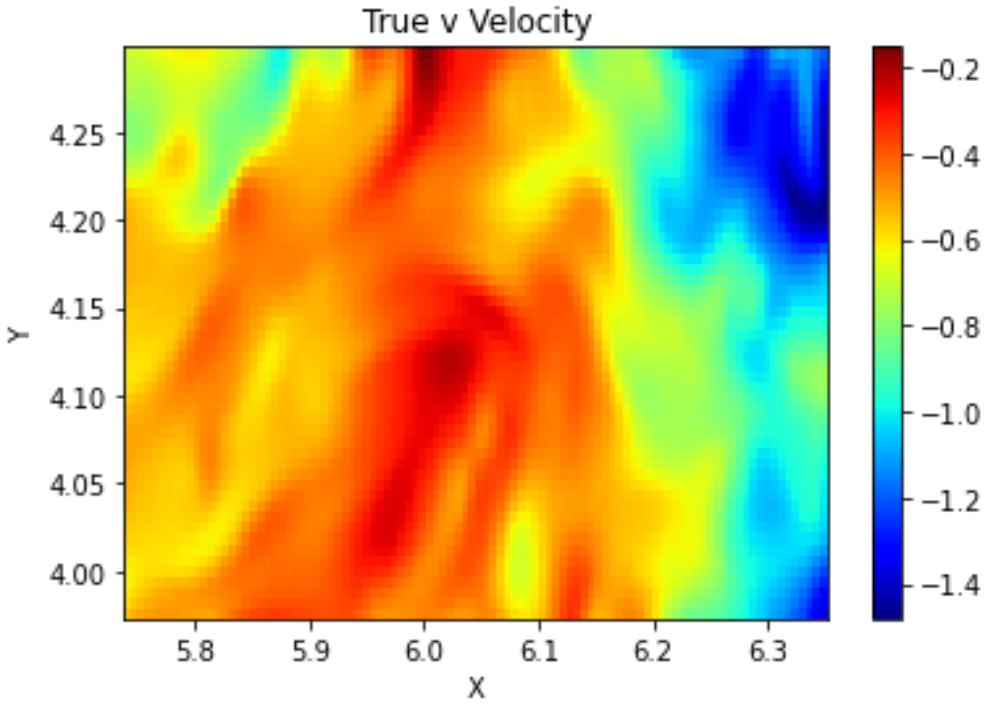


Figure 4.3 – True velocity in y from data

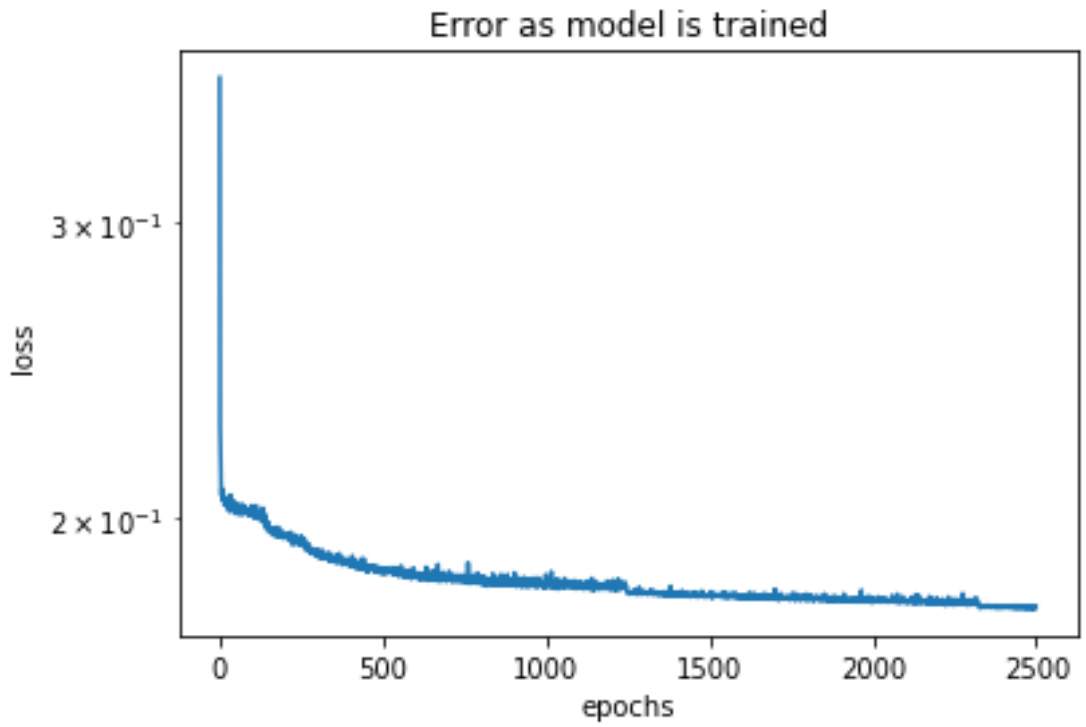


Figure 4.4 – Case 1: Error reduction plateaus

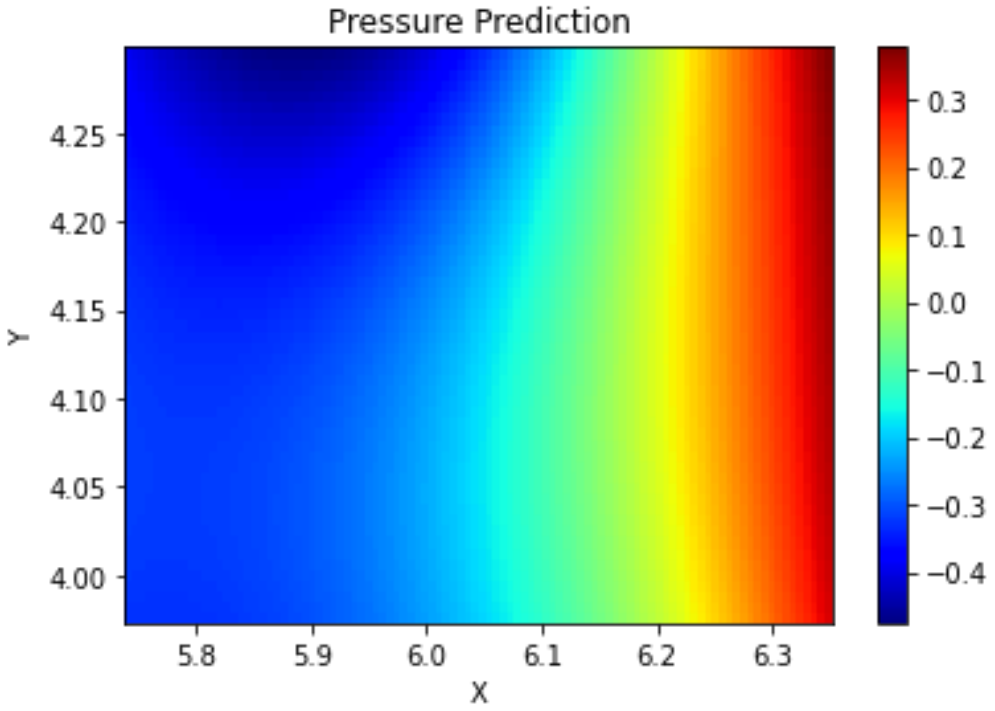


Figure 4.5 – Case 1 predicted pressure from PINN

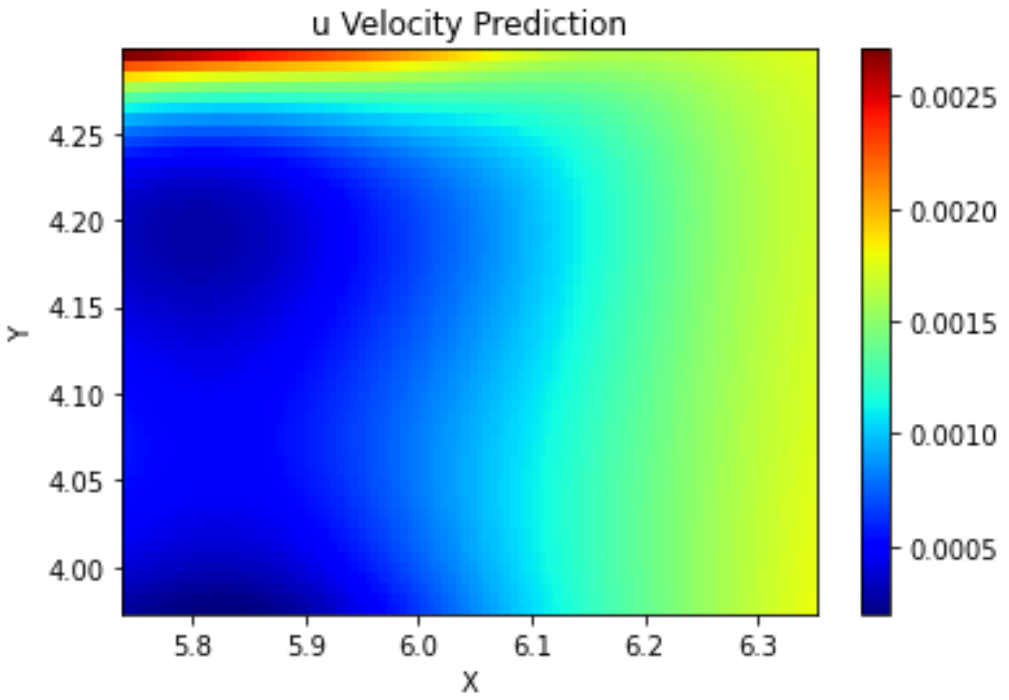


Figure 4.6 – Case 1 predicted velocity in x from PINN

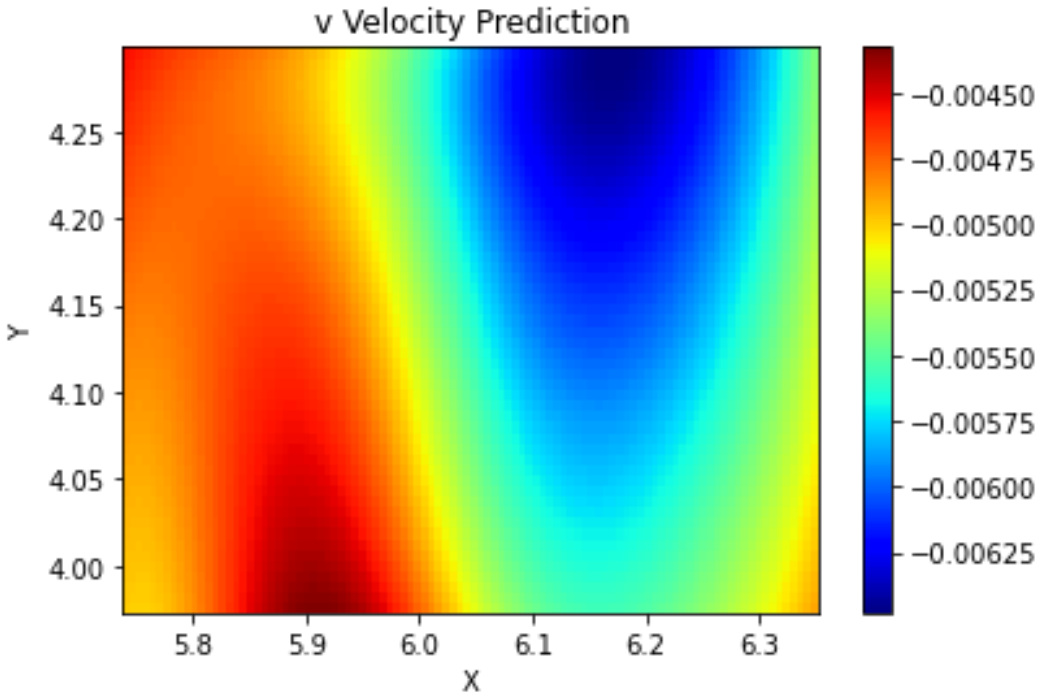


Figure 4.7 – Case 1 predicted velocity in y from PINN

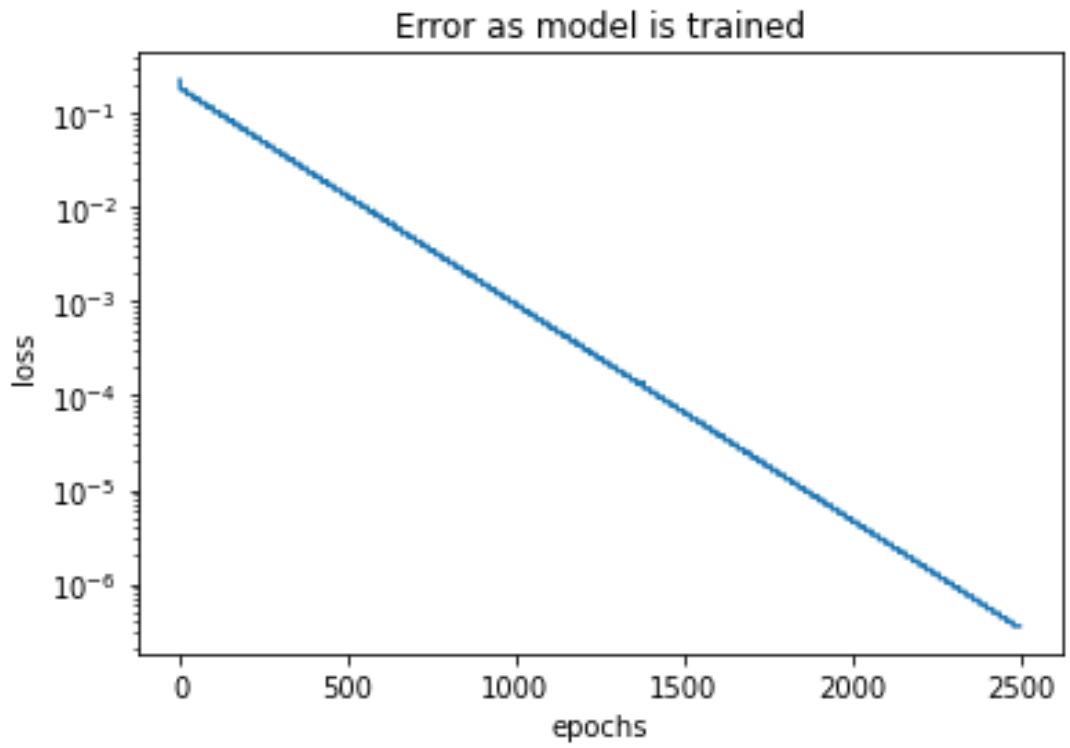


Figure 4.8 – Case 2: Error is reduced as the model is allowed to train

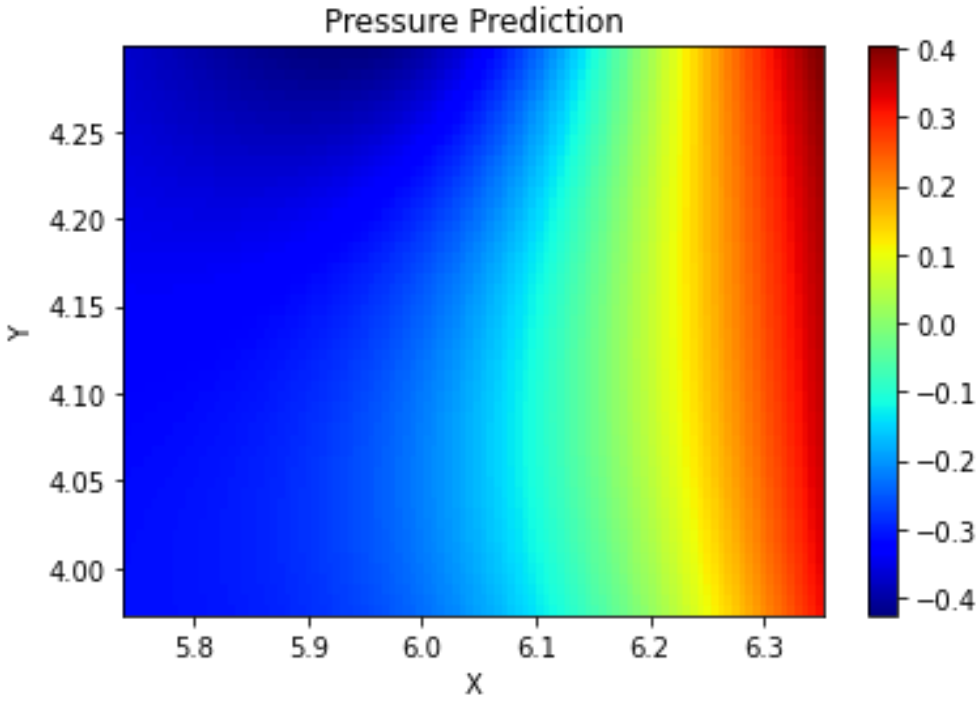


Figure 4.9 – Case 2 predicted pressure from PINN

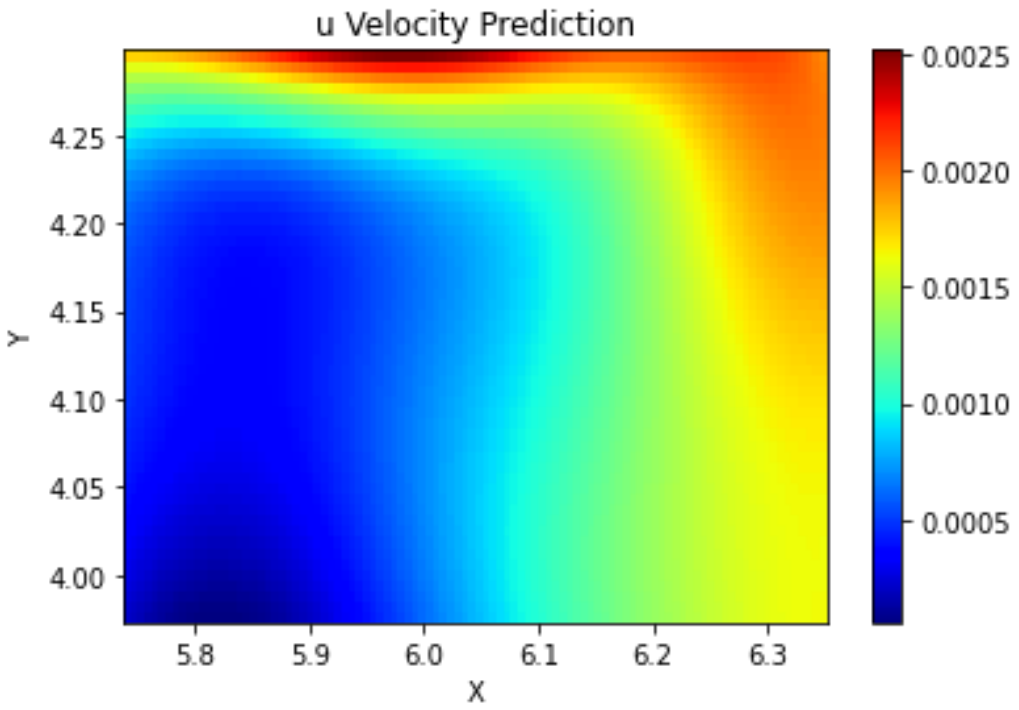


Figure 4.10 – Case 2 predicted velocity in x from PINN

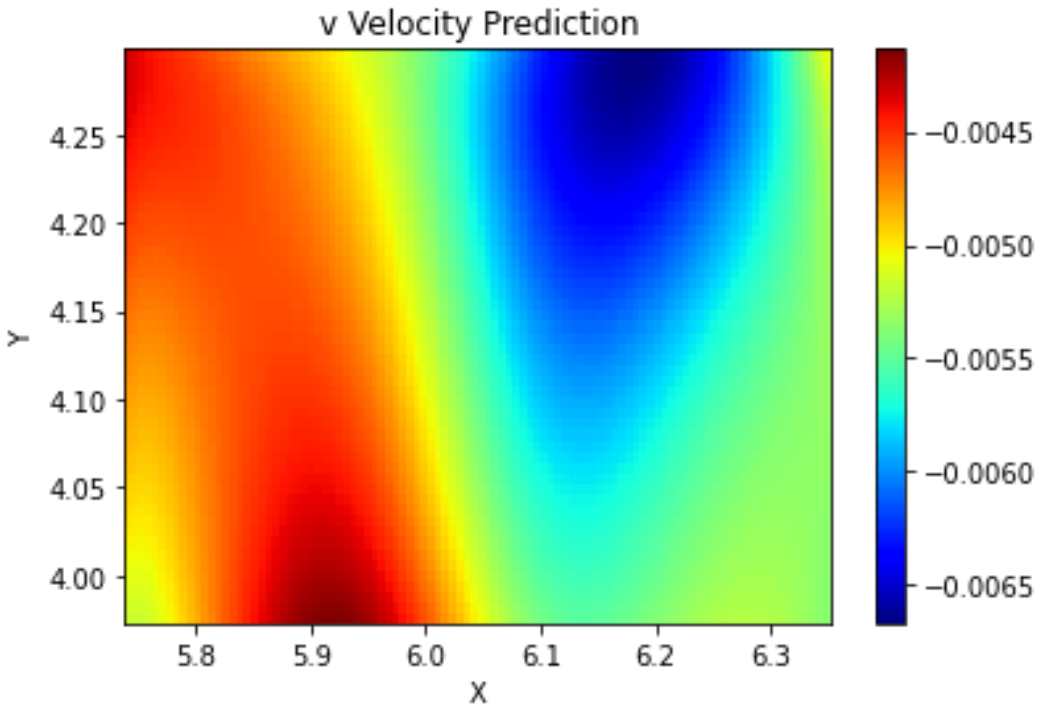


Figure 4.11 – Case 2 predicted velocity in y from PINN

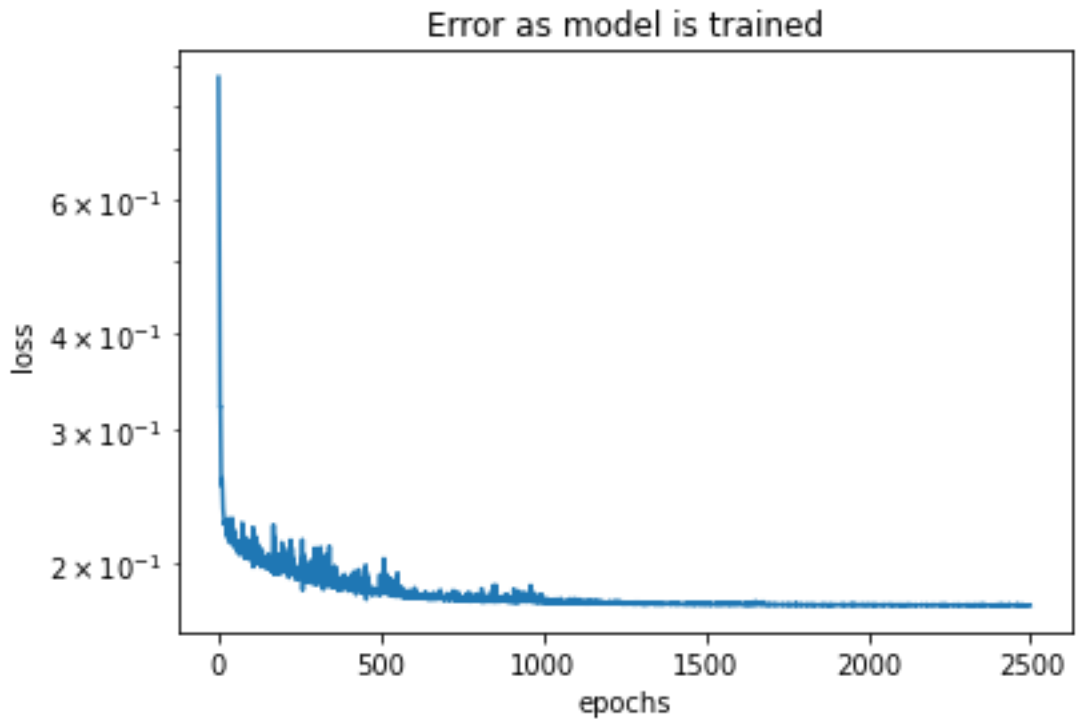


Figure 4.12 – Case 3: Error reduction plateaus

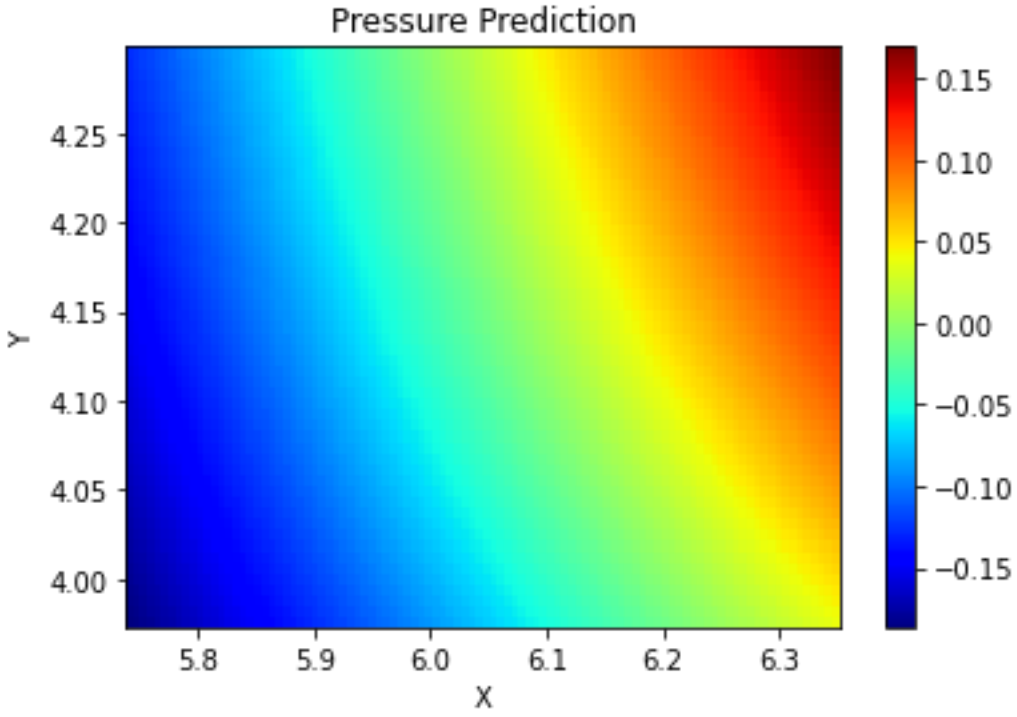


Figure 4.13 – Case 3 predicted pressure from PINN

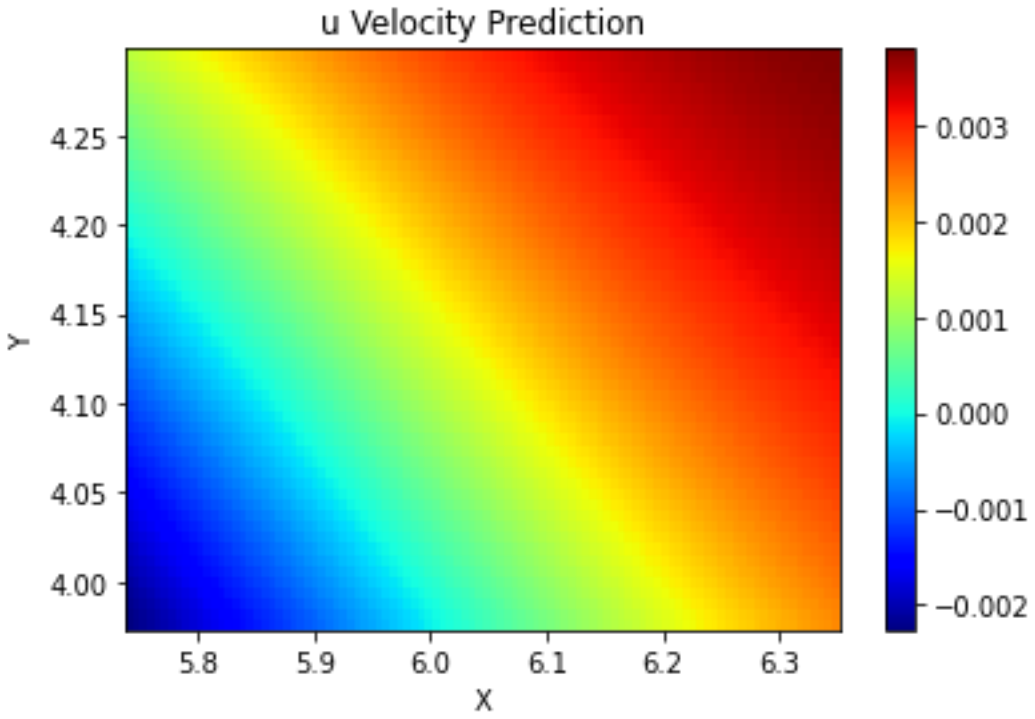


Figure 4.14 – Case 3 predicted velocity in x from PINN

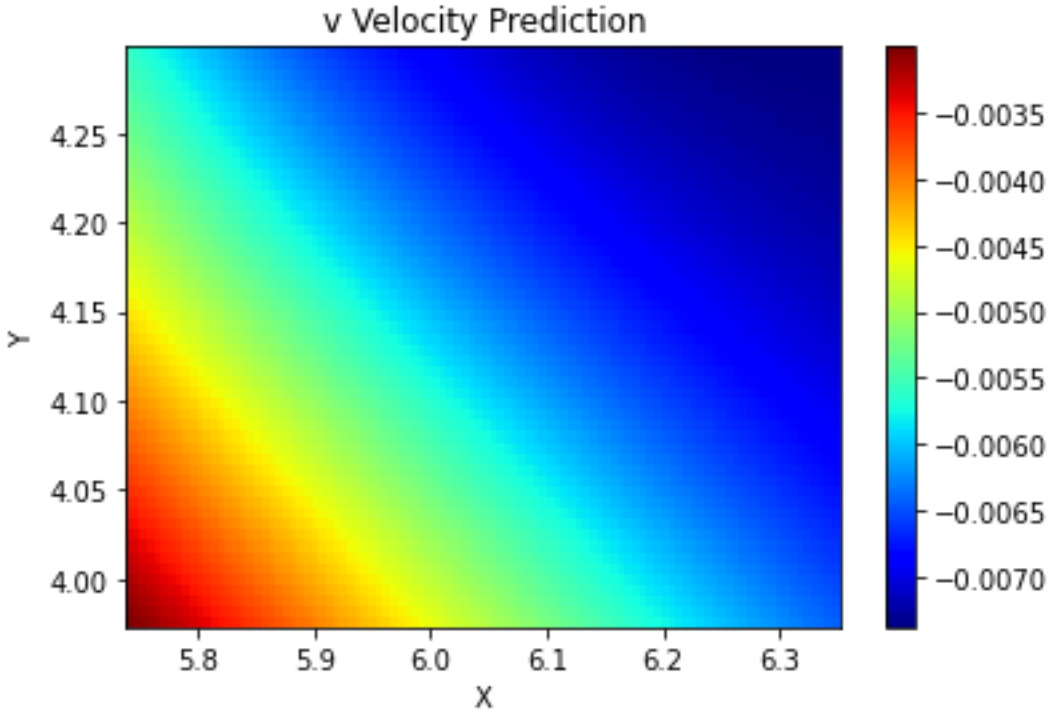


Figure 4.15 – Case 3 predicted velocity in y from PINN

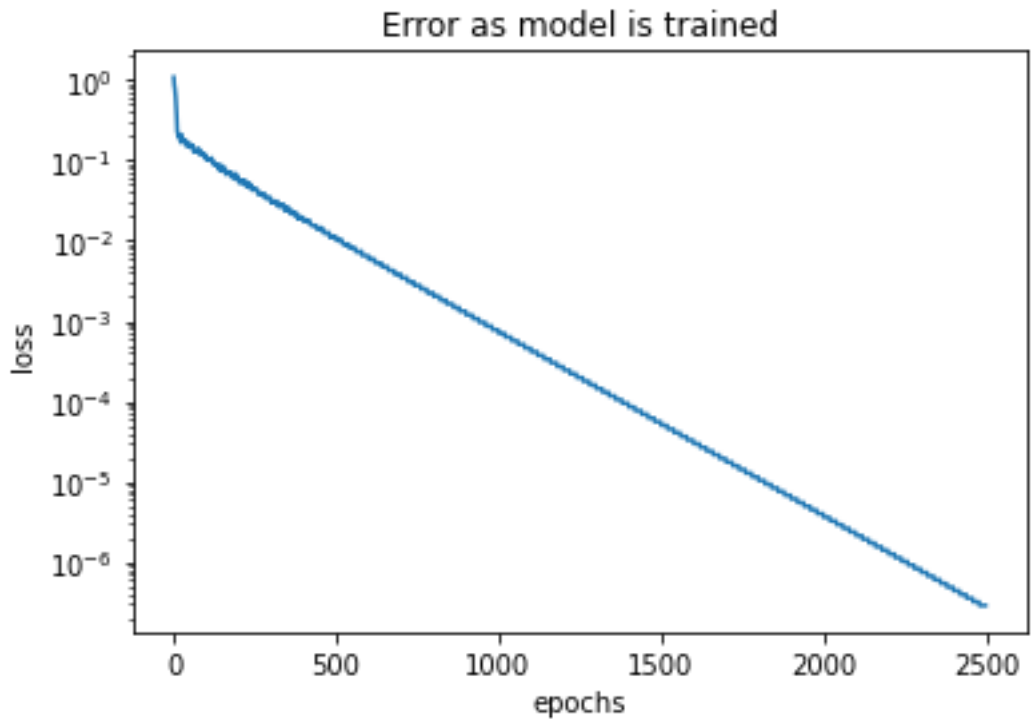


Figure 4.16 – Case 4: Error is reduced as the model is allowed to train

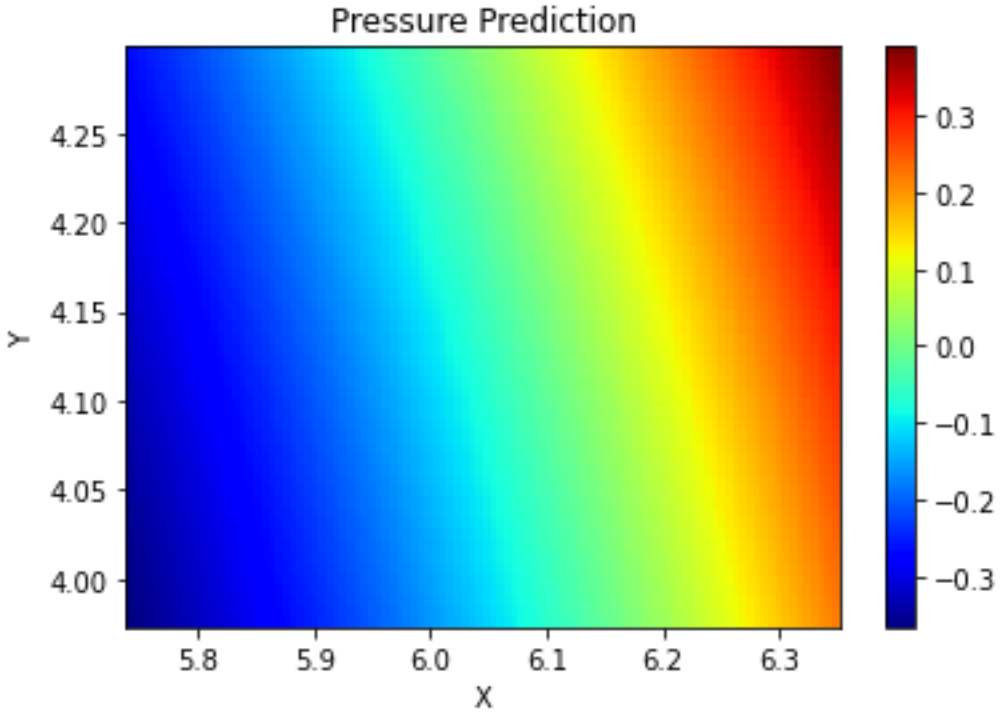


Figure 4.17 – Case 4 Predicted pressure from PINN

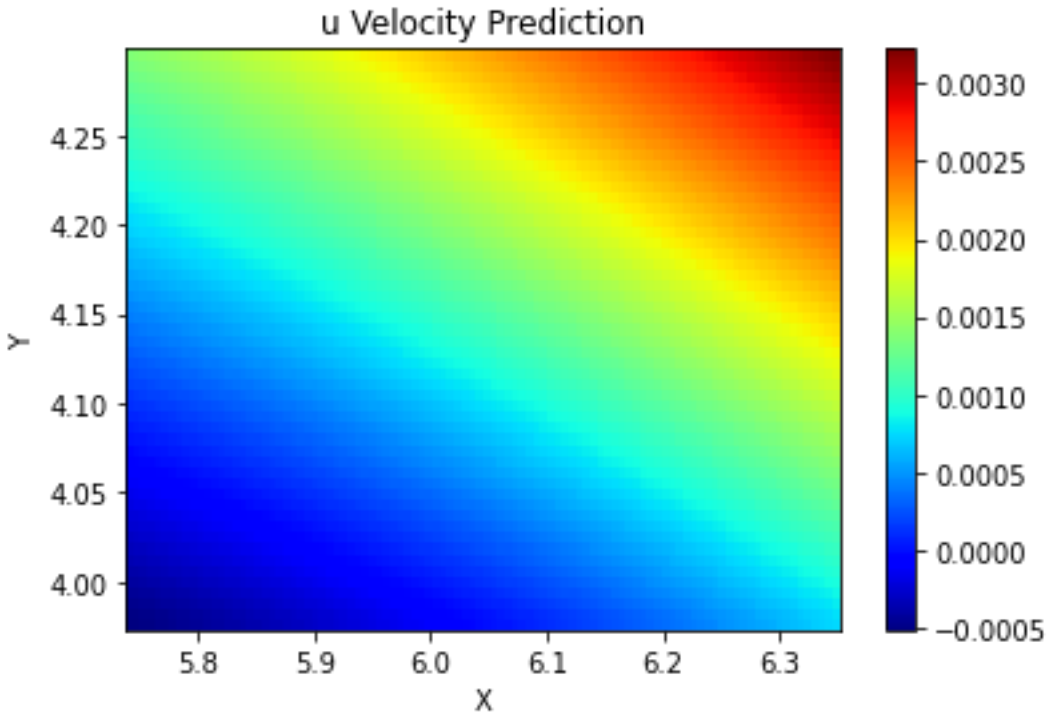


Figure 4.18 – Case 4 predicted velocity in x from PINN

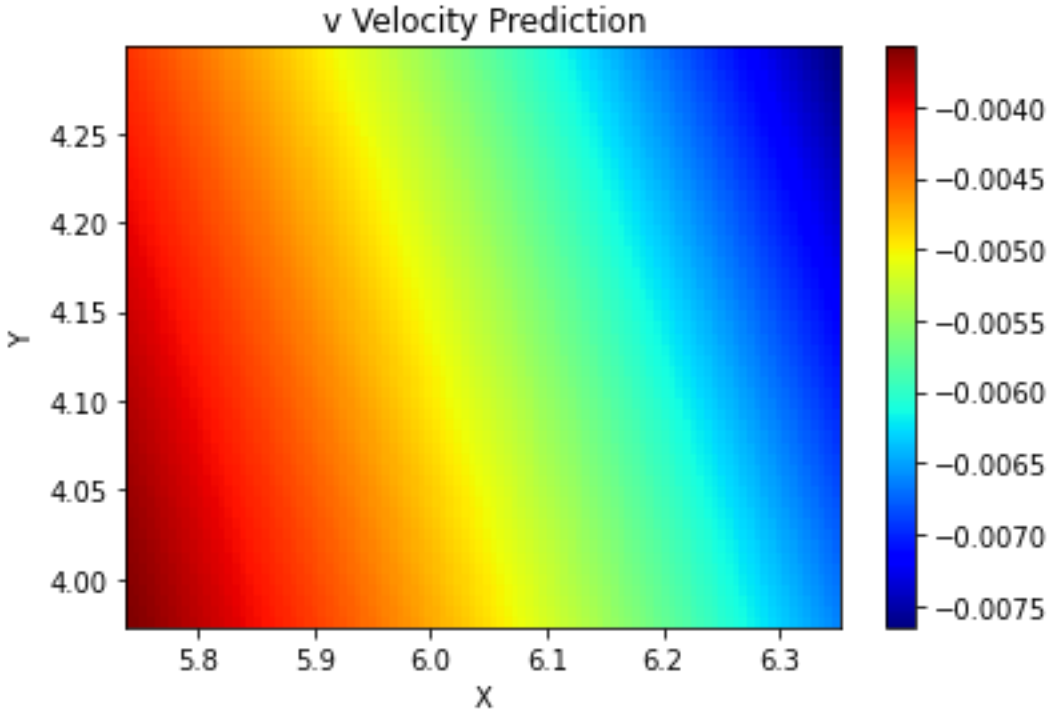


Figure 4.19 – Case 4 predicted velocity in y from PINN

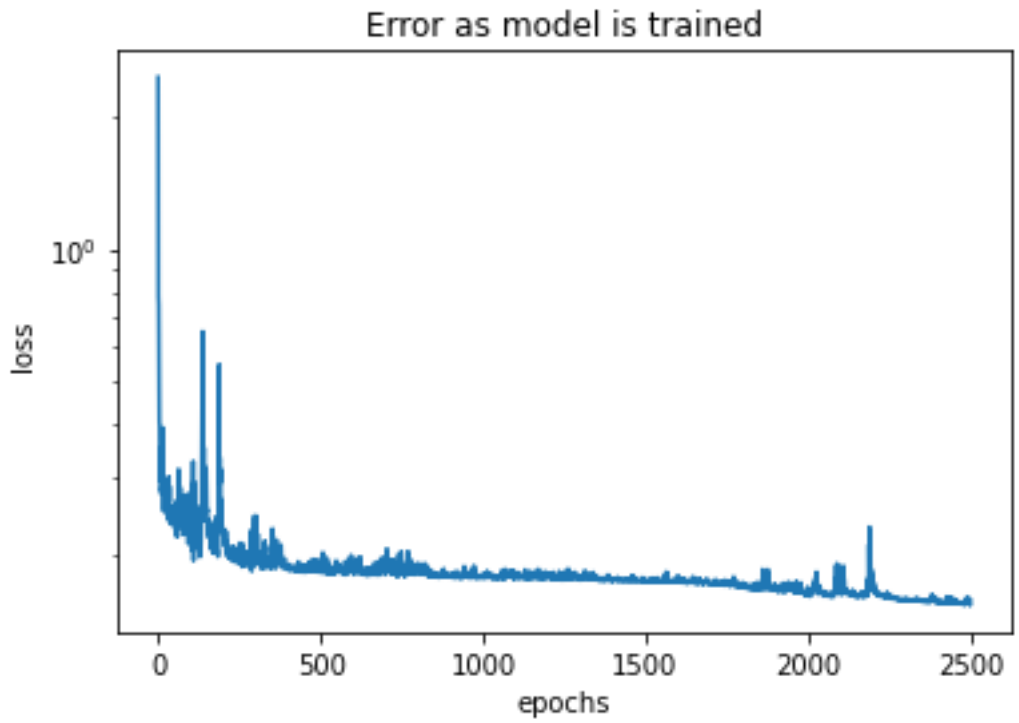


Figure 4.20 – Case 5: Error reduction plateaus

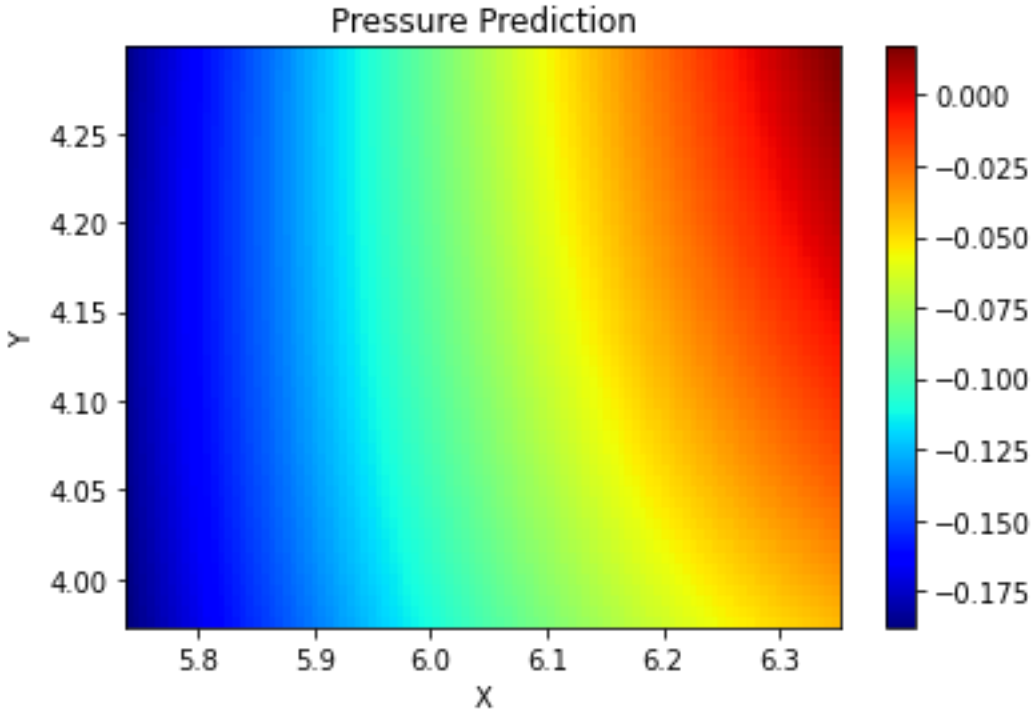


Figure 4.21 – Case 5 predicted pressure from PINN

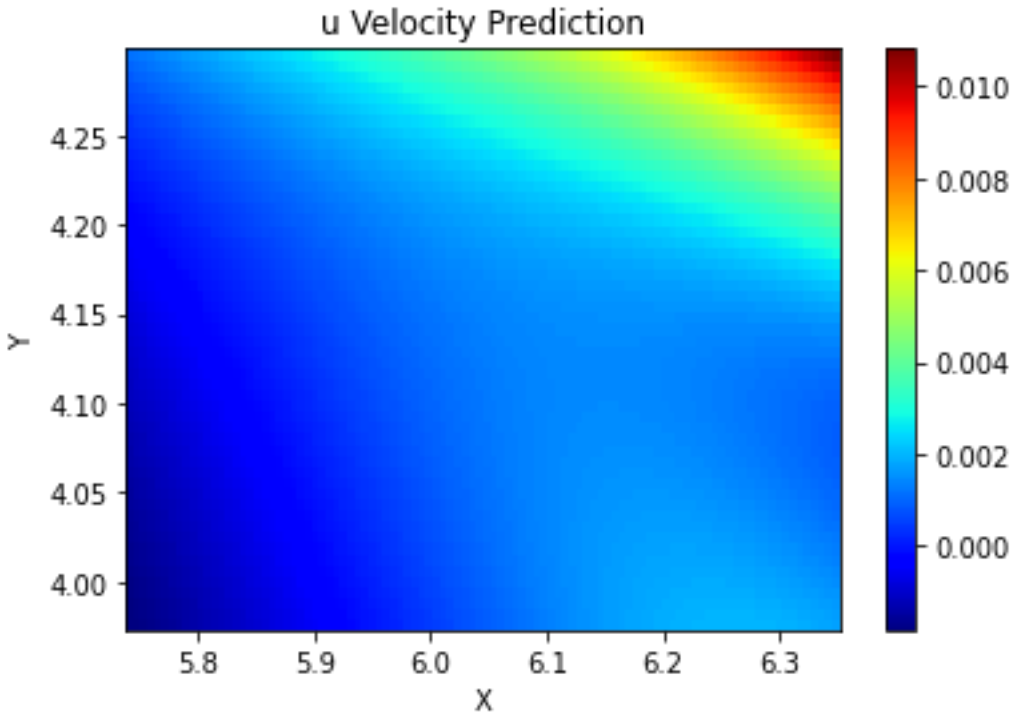


Figure 4.22 – Case 5 predicted velocity in x from PINN

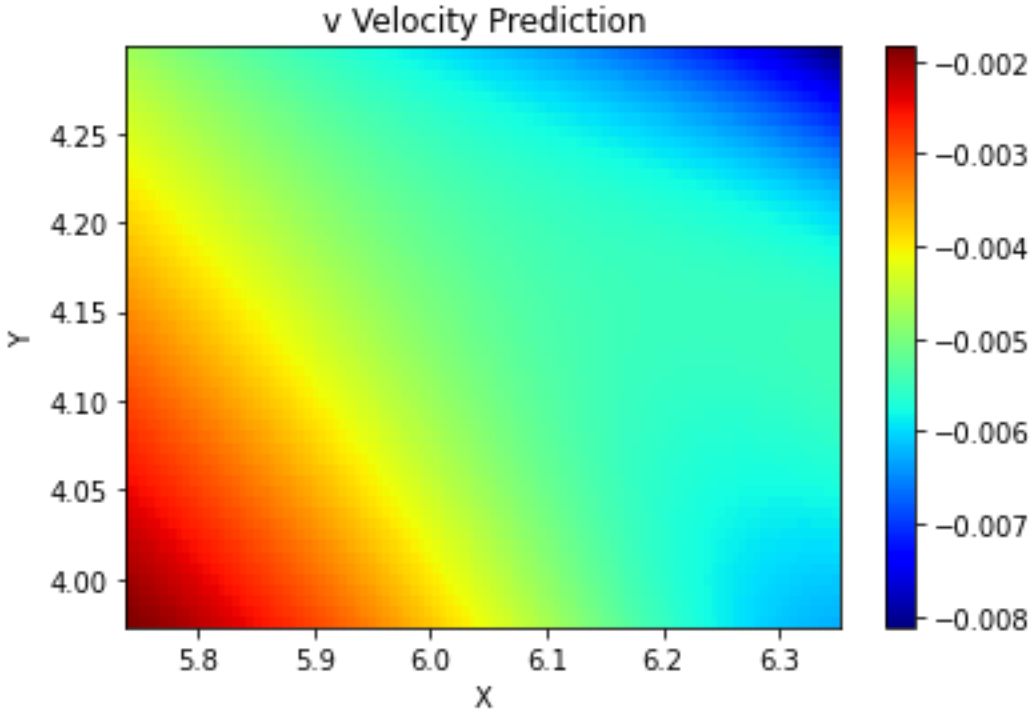


Figure 4.23 – Case 5 predicted velocity in y from PINN

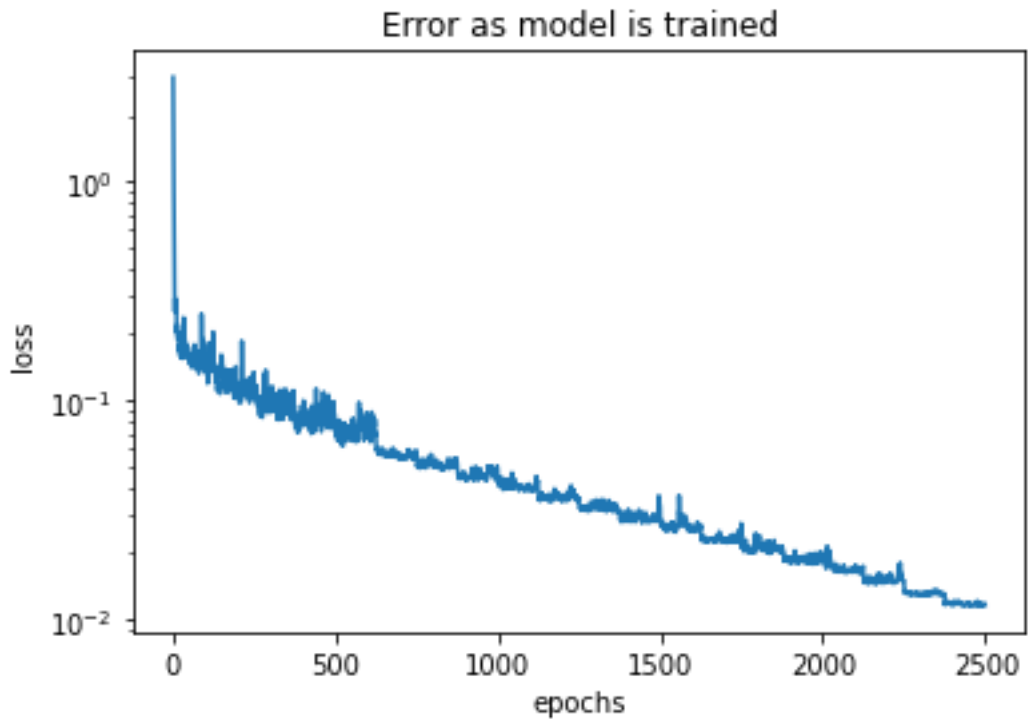


Figure 4.24 – Case 6: Error is reduced as the model is allowed to train

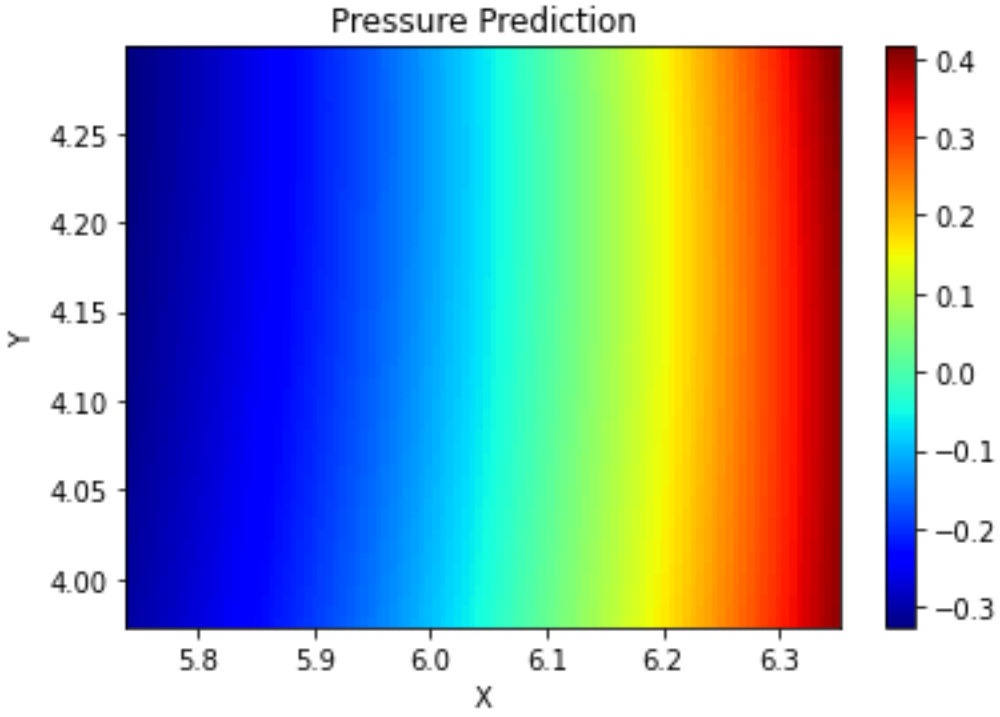


Figure 4.25 – Case 6 predicted pressure from PINN

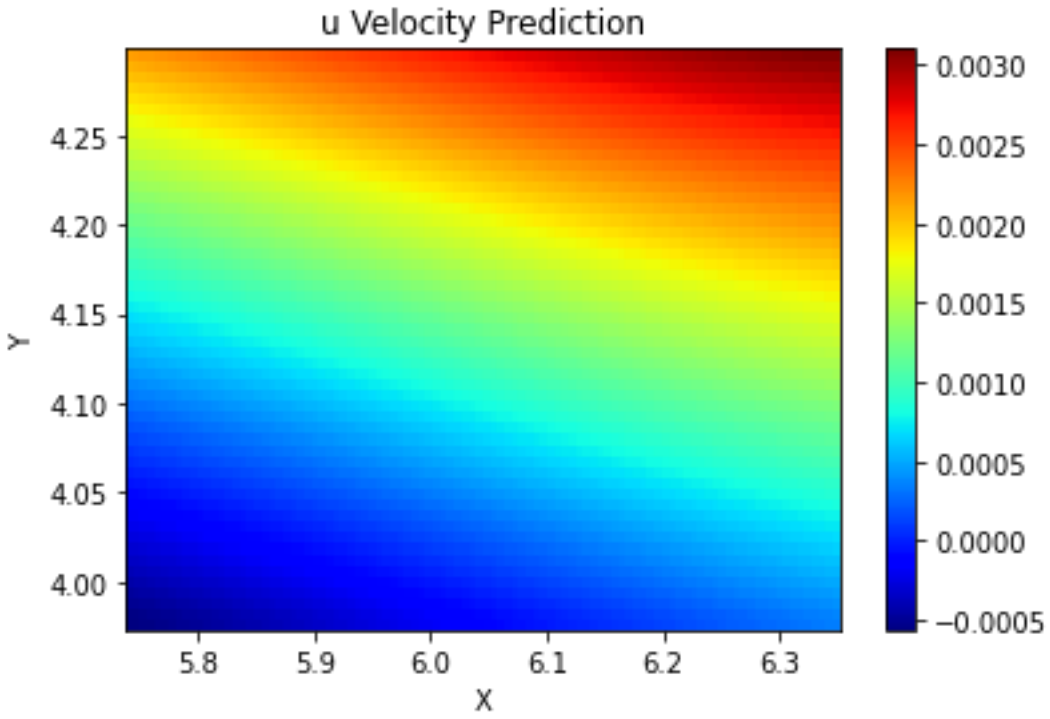


Figure 4.26 – Case 6 predicted velocity in x from PINN

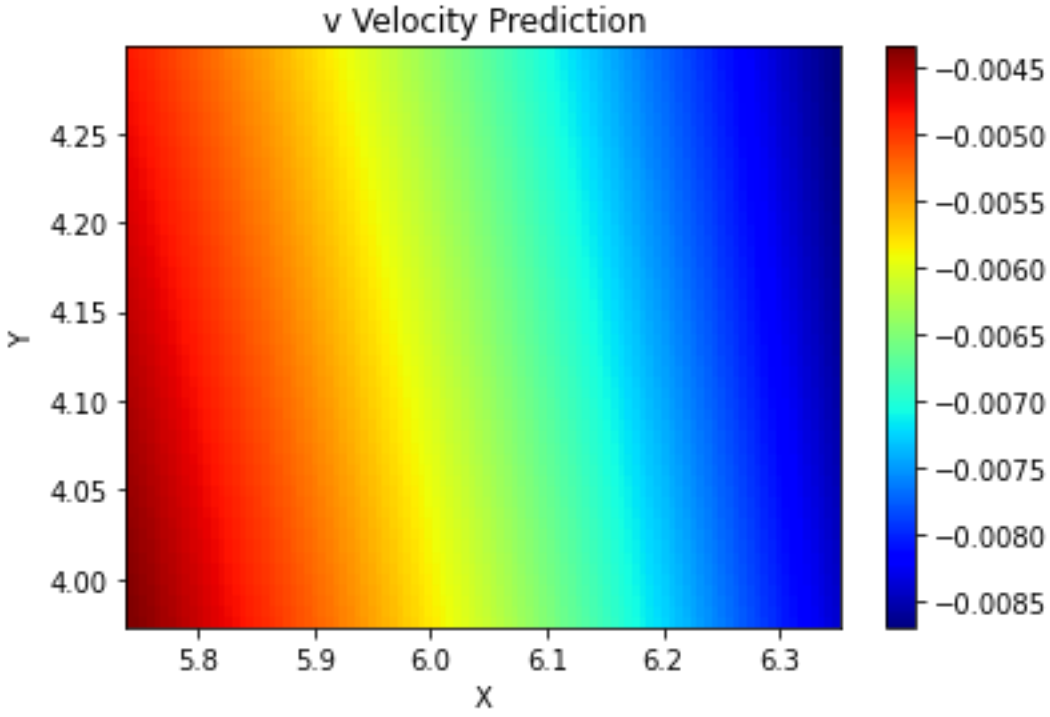


Figure 4.27 – Case 6 predicted velocity in y from PINN

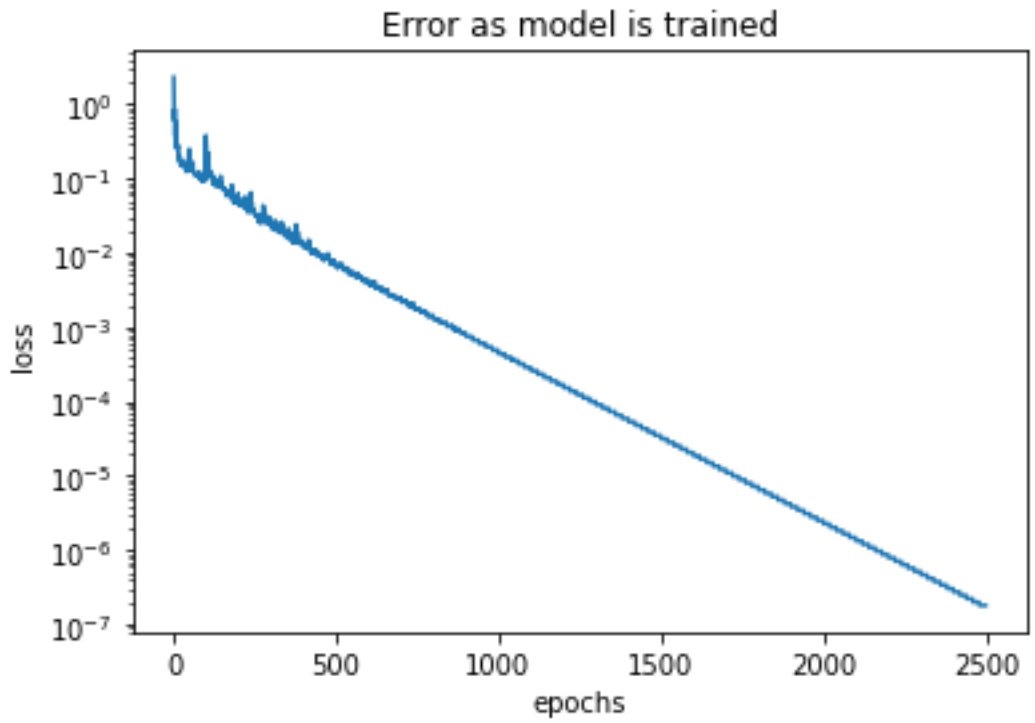


Figure 4.28 – Case 7: Error is reduced as the model is allowed to train

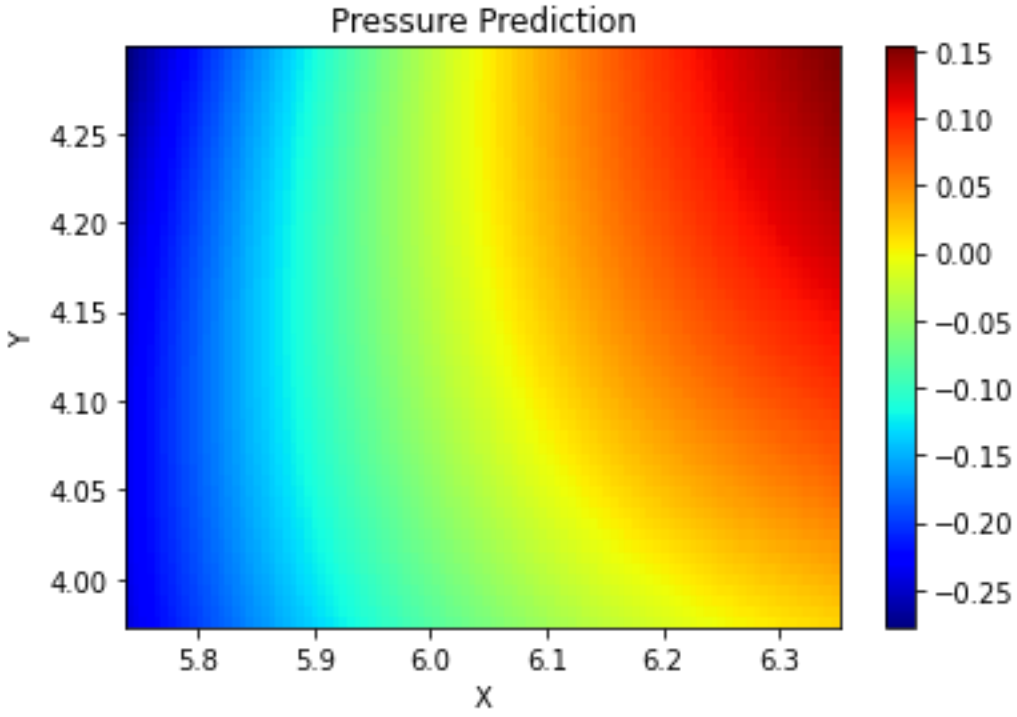


Figure 4.29 – Case 7 predicted pressure from PINN

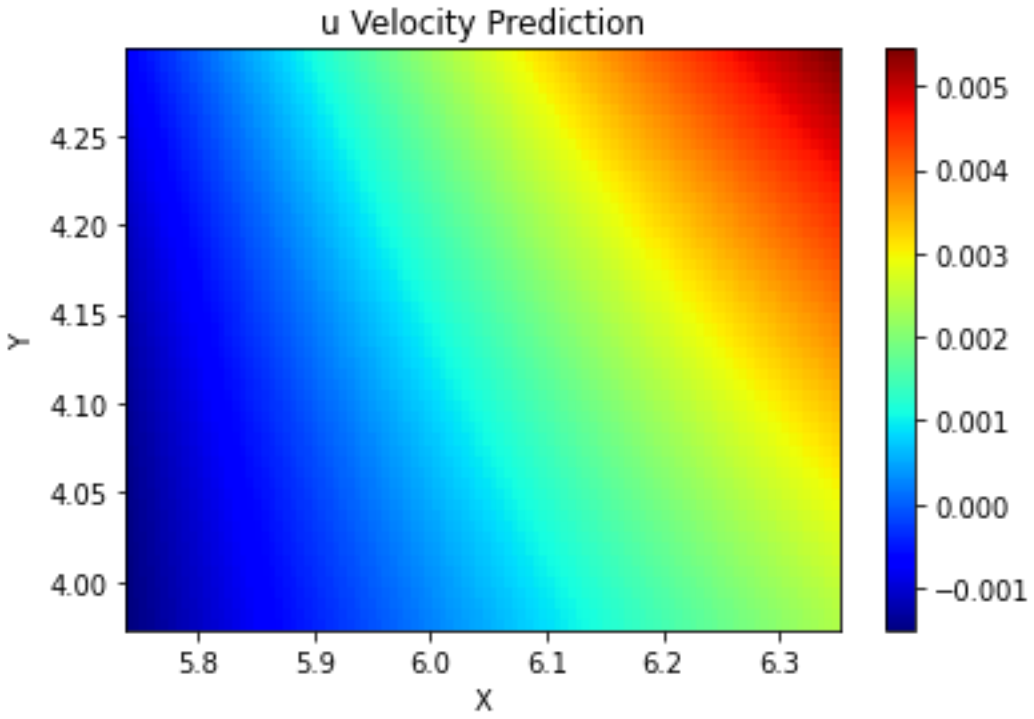


Figure 4.30 – Case 7 predicted velocity in x from PINN

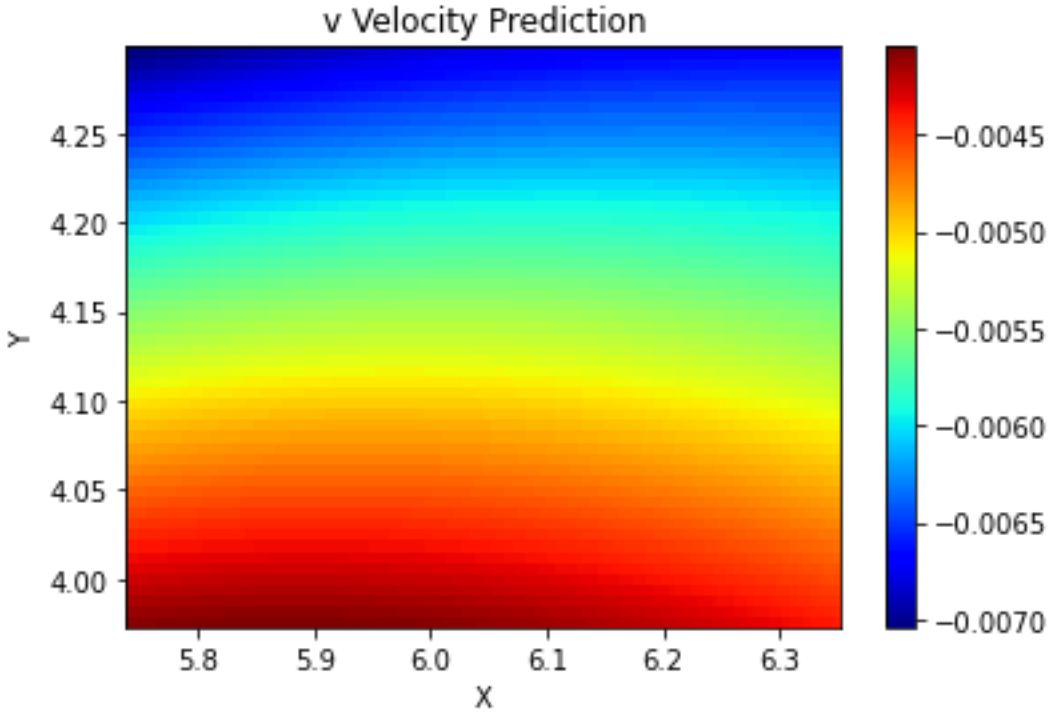


Figure 4.31 – Case 7 predicted velocity in y from PINN

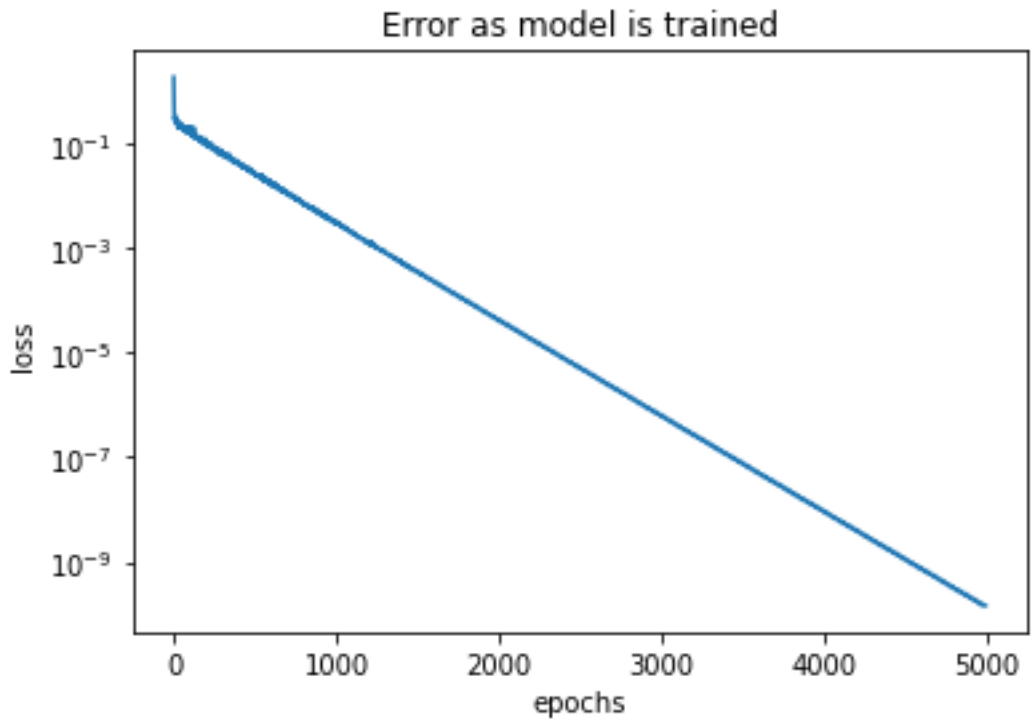


Figure 4.32 – Case 8: Error is reduced as the model is allowed to train

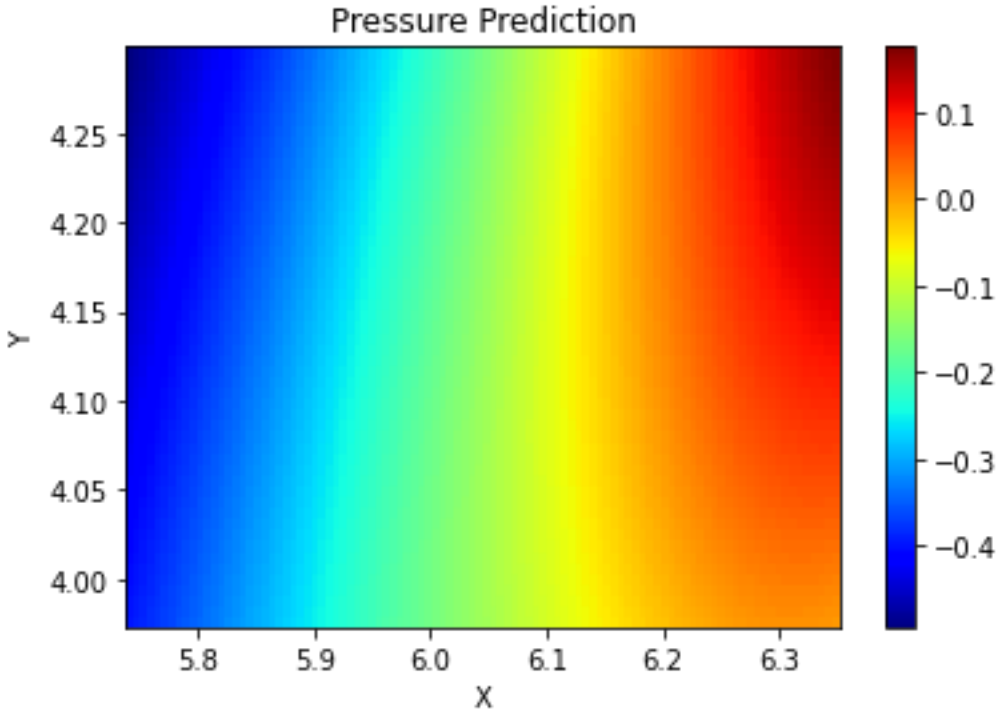


Figure 4.33 – Case 8 predicted pressure from PINN

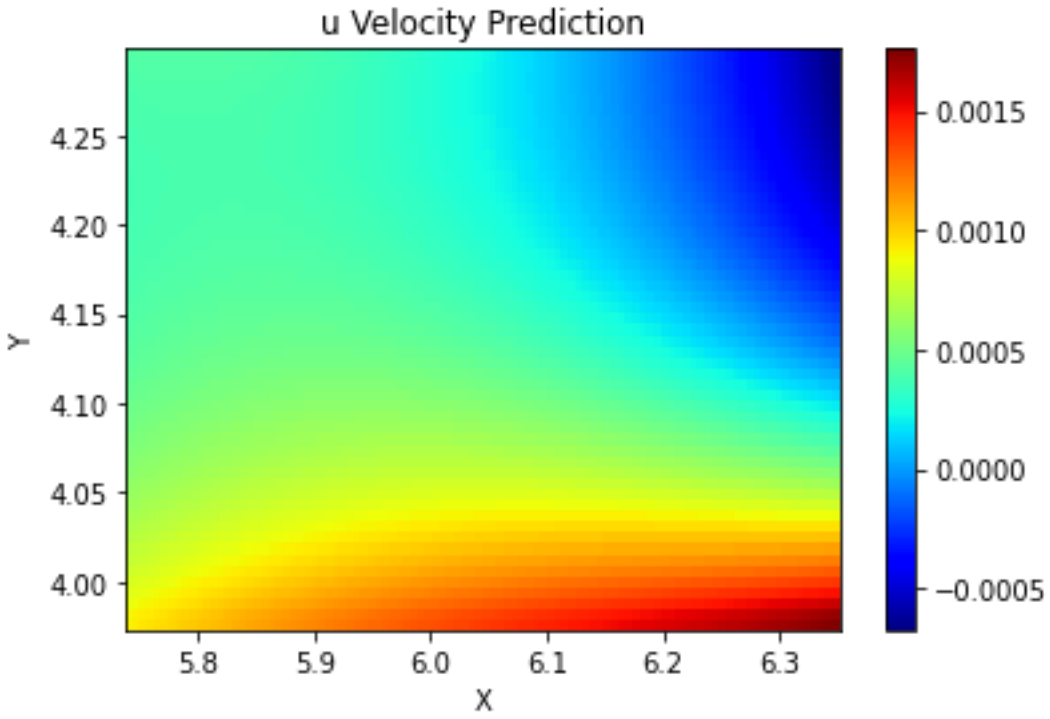


Figure 4.34 – Case 8 predicted velocity in x from PINN

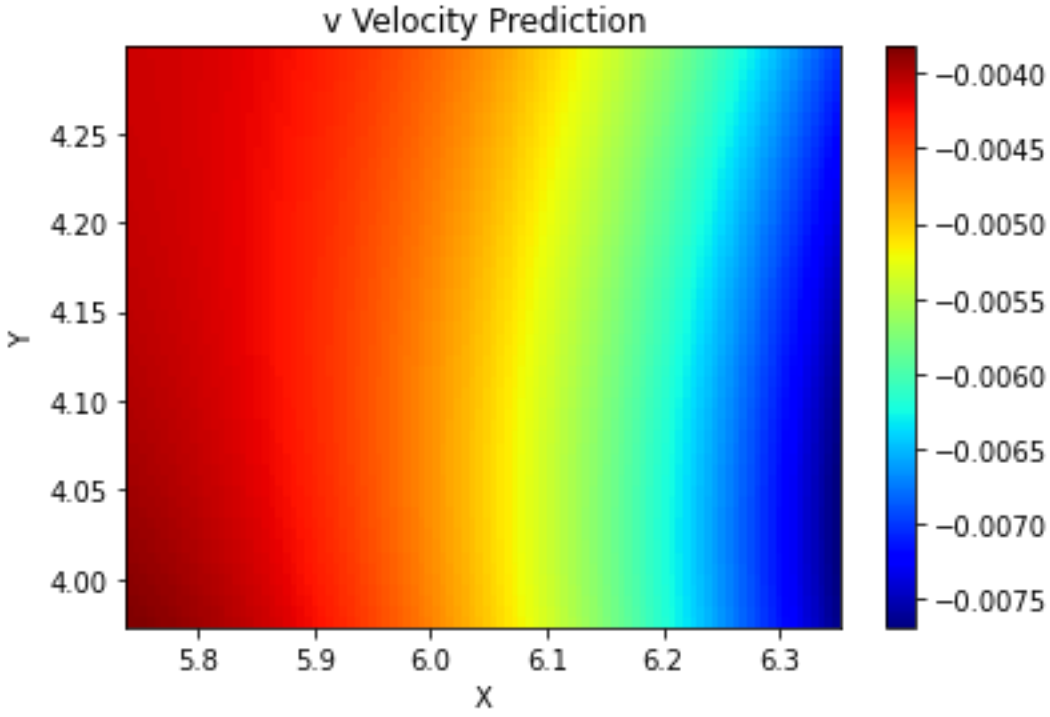


Figure 4.35 – Case 8 predicted velocity in y from PINN

5. Discussion

As a baseline, Cases 1 and 2 were trained using all points in the dataset. That is, all 5454 points in space were available for training, versus the 22 sparse points in the other cases. Naturally the prediction made from the PINNs in these cases were more accurate than the PINNs with far fewer data points available for training.

Cases 1, 3, and 5 did not use adaptive weights during training, and as a result failed to train much at all. Figure 4.4, Figure 4.12, and Figure 4.20 show that the PINN is unable to reduce training error past early training. This is likely due to the error terms being of different magnitude, incentivizing the neural network to focus only on certain certain terms, regardless of the physics. This results in the neural network being trapped in a local minimum. In Cases 3 and 5 the predictions made or wildly inaccurate and of little use in practice. In predicting velocities, however, the PINN did not perform as expected. Despite having access to the full dataset, the PINN was unable to match the values. In all cases only the general shape (gradient of velocities) of a plot is able to be predicted. The magnitudes of velocity for both the x and y cases are unfortunately inaccurate. The author notes that it is odd that the PINN predicted the wrong magnitude for velocities, yet still produced a somewhat similar gradient plot. The current hypothesis for this is that JHTD is simply too turbulent for a 2D approximation to fully describe a slice. Large amounts of flow move in the Z axis, making the 2D approximation too inaccurate for a PINN to use in training. That is, even when the 2D NS equations are fulfilled exactly, the neural network will still have significant error, and so will train away from the correct solution. As pressure is scalar quantity rather than a vector one, it was easier for the neural network to predict. As a result pressure predictions in all cases were more accurate than those of velocities.

As Case 1 had the full 5454 spatial points to train from, the prediction made are not as inaccurate as those made in Cases 3 and 5. Comparing Cases 1 and 2 via Figure 4.5, Figure 4.6, and Figure 4.7 to Figure 4.9, Figure 4.10, and Figure 4.11 respectively, it is apparent that the adaptive weights had little effect on the shape of the gradient. This is in spite of the error of Case 2 being several orders of magnitude lower than that of Case 1, as shown in Table 4.1. This paradox is solved by knowing how adaptive weights work. More important terms have their weight, and therefore value in the loss function increased. Less important terms have their impact on the loss function decreased. Thus it is possible for error of certain terms to be very high if they are deemed “unimportant” by the neural network. This downside is the reason that adaptive weights must be used moderation. If the update rate is too high a neural network will remain inaccurate even if loss decreases by several orders of magnitude.

Cases 3 and 4 are used to show the effect of network width and depth. Cases 1 through 4 have 8 layers with 20 neurons each, while Cases 5 through 8 have 11 layers with 120 neurons each. Neural networks with greater numbers of layers and neurons per layer are able to learn more complex behavior, at the cost of higher computational time. When comparing Cases 3 and 5 there is not much of a difference in loss due to the lack of adaptive weights. In both of those cases loss plateaued early without much training. When comparing Cases 4 and 7, differences become apparent. As can be seen from Table 4.1, loss in Case 7 is ~80% lower than in Case 4.

However, the overuse of adaptive weights in Cases 4 and 7 (updated every 20 epochs) has resulted in both being inaccurate in predicting velocities.

Case 6 is the most successful of the PINNs in predicting velocities. As can be seen in Figure 4.26 and Figure 4.27, the predicted velocity gradients are the closest to the true velocity gradients shown in Figure 4.2 and Figure 4.3 respectively. This can be attributed to the more moderate use of adaptive weights, with their values only updated every 125 epochs, as shown in Table 4.1. However, due to the use of the 2D NS equations and the turbulent 3D data used, the values themselves are not accurate.

Case 8 shows how neural networks can fail in the event of overtraining. While Case 8 has the lowest loss of the cases, it is less accurate in predicting velocity gradients than Case 7 or Case 6. This is due to the neural network being optimized to predict the sparse points given rather than the entire flow field. In essence, the neural network becomes good at predicting the 22 points given while becoming worse at predicting the other unknown points.

6. Conclusion

In this work different PINN setups were tested to determine the effectiveness of each one in predicting flow fields given sparse points. It was found that the moderate use of adaptive weights is necessary for the accurate prediction of velocity gradients in flows with heavy turbulence. In less turbulent flows adaptive weights may not be necessary. Additionally, wider and deeper neural networks performed better than narrower and shallower ones, as expected by theory. Training should be limited to as few epochs as possible to prevent overtraining when using sparse points, as neural network may become optimized to only predict these sparse points well.

Due to time and compute constraints, the PINN trained was only able predict velocity gradient effectively, and not the velocities themselves. In practice, this algorithm could be used to find large differences in airflow velocities in a certain region, signifying turbulence. It could not, however, be used to predict airflow velocity itself. Future work should focus on using the 3D NS equations for training, which was not possible in this work due to requiring significant amounts of compute resources. Once fully trained however, a 3D NS PINN should not require significantly more compute to run versus a 2D NS trained PINN.

References

[1] Kikuchi, R., Misaka, T., Obayashi, S., and Inokuchi, H. “Real-Time Estimation of Airflow Vector Based on Lidar Observations for Preview Control.” *Atmospheric Measurement Techniques*, Vol. 13, No. 12, 2020, pp. 6543–6558. https://doi.org/10.5194/amt-13-6543-2020.

[2] Spears, B. K. “Contemporary Machine Learning: A Guide for Practitioners in the Physical Sciences.” 2017. https://doi.org/10.1063/1.5020791.

[3] Shangguan, M., Qiu, J., Yuan, J., Shu, Z., Zhou, L., and Xia, H. “Doppler Wind Lidar From UV to NIR: A Review With Case Study Examples.” *Frontiers in Remote Sensing*, Vol. 2, 2022.

[4] Nijhuis, A. C. P. O., Thobois, L. P., Barbaresco, F., Haan, S. D., Dolfi-Bouteyre, A., Kovalev, D., Krasnov, O. A., Vanhoenacker-Janvier, D., Wilson, R., and Yarovoy, A. G. “Wind Hazard and Turbulence Monitoring at Airports with Lidar, Radar, and Mode-S Downlinks: The UFO Project.” *Bulletin of the American Meteorological Society*, Vol. 99, No. 11, 2018, pp. 2275–2293. https://doi.org/10.1175/BAMS-D-15-00295.1.

[5] Towers, P., and Jones, B. Ll. “Real-Time Wind Field Reconstruction from LiDAR Measurements Using a Dynamic Wind Model and State Estimation.” *Wind Energy*, Vol. 19, No. 1, 2012, pp. 133–150. https://doi.org/10.1002/we.1824.

[6] Bauweraerts, P., and Meyers, J. “Reconstruction of Turbulent Flow Fields from Lidar Measurements Using Large-Eddy Simulation.” *Journal of Fluid Mechanics*, Vol. 906, 2020. https://doi.org/10.1017/jfm.2020.805.

[7] Clifton, A., Boquet, M., Burin Des Roziers, E., Westerhellweg, A., Hofsass, M., Klaas, T., Vogstad, K., Clive, P., Harris, M., Wylie, S., Osler, E., Banta, B., Choukulkar, A., Lundquist, J., and Aitken, M. *Remote Sensing of Complex Flows by Doppler Wind Lidar: Issues and Preliminary Recommendations*. Publication NREL/TP-5000-64634. National Renewable Energy Lab. (NREL), Golden, CO (United States), 2015.

[8] Vasiljević, N., Harris, M., Tegtmeier Pedersen, A., Rolighed Thorsen, G., Pitter, M., Harris, J., Bajpai, K., and Courtney, M. “Wind Sensing with Drone-Mounted Wind Lidars: Proof of Concept.” *Atmospheric Measurement Techniques*, Vol. 13, No. 2, 2020, pp. 521–536. https://doi.org/10.5194/amt-13-521-2020.

[9] Brunton, S. L., Noack, B. R., and Koumoutsakos, P. “Machine Learning for Fluid Mechanics.” *Annual Review of Fluid Mechanics*, Vol. 52, No. 1, 2020, pp. 477–508. https://doi.org/10.1146/annurev-fluid-010719-060214.

[10] Thuerey, N., Weissenow, K., Prantl, L., and Hu, X. “Deep Learning Methods for Reynolds-Averaged Navier-Stokes Simulations of Airfoil Flows.” *AIAA Journal*, Vol. 58, No. 1, 2019, pp. 25–36. https://doi.org/10.2514/1.J058291.

[11] Ganesh, S. What’s The Role Of Weights And Bias In a Neural Network? *Medium*. https://towardsdatascience.com/whats-the-role-of-weights-and-bias-in-a-neural-network-4cf7e9888a0f. Accessed Jul. 29, 2022.

[12] Scherl, I., Strom, B., Shang, J. K., Williams, O., Polagye, B. L., and Brunton, S. L. “Robust Principal Component Analysis for Modal Decomposition of Corrupt Fluid Flows.” *Physical Review Fluids*, Vol. 5, No. 5, 2020, p. 054401. https://doi.org/10.1103/PhysRevFluids.5.054401.

[13] Zhang, J., and Zhao, X. “Three-Dimensional Spatiotemporal Wind Field Reconstruction Based on Physics-Informed Deep Learning.” *Applied Energy*, Vol. 300, 2021, p. 117390. https://doi.org/10.1016/j.apenergy.2021.117390.
[14] Novati, G., de Laroussilhe, H. L., and Koumoutsakos, P. “Automating Turbulence Modelling by Multi-Agent Reinforcement Learning.” *Nature Machine Intelligence*, Vol. 3, No. 1, 2021, pp. 87–96. https://doi.org/10.1038/s42256-020-00272-0.
[15] Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., and Hoyer, S. “Machine Learning–Accelerated Computational Fluid Dynamics.” *Proceedings of the National Academy of Sciences*, Vol. 118, No. 21, 2021, p. e2101784118. https://doi.org/10.1073/pnas.2101784118.
[16] Li, Y., Perlman, E., Wan, M., Yang, Y., Meneveau, C., Burns, R., Chen, S., Szalay, A., and Eyink, G. “A Public Turbulence Database Cluster and Applications to Study Lagrangian Evolution of Velocity Increments in Turbulence.” *Journal of Turbulence*, Vol. 9, 2008, p. N31. https://doi.org/10.1080/14685240802376389.
[17] Orszag, S. A. “Analytical Theories of Turbulence.” *Journal of Fluid Mechanics*, Vol. 41, No. 2, 1970, pp. 363–386. https://doi.org/10.1017/S0022112070000642.
[18] Markidis, S. “The Old and the New: Can Physics-Informed Deep-Learning Replace Traditional Linear Solvers?” *Frontiers in Big Data*, Vol. 4, 2021.
[19] Mishra, S., and Molinaro, R. “Estimates on the Generalization Error of Physics-Informed Neural Networks for Approximating a Class of Inverse Problems for PDEs.” *IMA Journal of Numerical Analysis*, Vol. 42, No. 2, 2022, pp. 981–1022. https://doi.org/10.1093/imanum/drab032.
[20] Gnanasambandam, R., Shen, B., Chung, J., Yue, X., Zhenyu, and Kong. Self-Scalable Tanh (Stan): Faster Convergence and Better Generalization in Physics-Informed Neural Networks. http://arxiv.org/abs/2204.12589. Accessed Jul. 21, 2022.
[21] Dubey, S. R., Singh, S. K., and Chaudhuri, B. B. “Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark.” *Neurocomputing*, Vol. 503, 2022, pp. 92–108. https://doi.org/10.1016/j.neucom.2022.06.111.
[22] Jagtap, A. D., Shin, Y., Kawaguchi, K., and Karniadakis, G. E. “Deep Kronecker Neural Networks: A General Framework for Neural Networks with Adaptive Activation Functions.” *Neurocomputing*, Vol. 468, 2022, pp. 165–180. https://doi.org/10.1016/j.neucom.2021.10.036.
[23] Wang, S., Teng, Y., and Perdikaris, P. “Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks.” *SIAM Journal on Scientific Computing*, Vol. 43, No. 5, 2021, pp. A3055–A3081. https://doi.org/10.1137/20M1318043.
[24] Kingma, D. P., and Ba, J. Adam: A Method for Stochastic Optimization. http://arxiv.org/abs/1412.6980. Accessed Jul. 29, 2022.
[25] Byrd, R. H., Lu, P., Nocedal, J., and Zhu, C. “A Limited Memory Algorithm for Bound Constrained Optimization.” *SIAM Journal on Scientific Computing*, Vol. 16, No. 5, 1995, pp. 1190–1208. https://doi.org/10.1137/0916069.
[26] Haghighat, E., and Juanes, R. “SciANN: A Keras/Tensorflow Wrapper for Scientific Computations and Physics-Informed Deep Learning Using Artificial Neural Networks.” *Computer Methods in Applied Mechanics and Engineering*, Vol. 373, 2020, p. 113552. https://doi.org/10.1016/j.cma.2020.113552.
[27] SciANN Examples. SciANN, Nov 23, 2022.