Prediction of Marine Turbine Parameters using Variational Autoencoders

Filip Marinov

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

20/09/2020

Abstract

The ability to predict the thrust or torque applied to marine turbines by using upstream flow velocity and torque or thrust respectively could improve the lifespan and function of the turbines. This project attempts to develop software, which is capable of such predictions. The approach is split in two stages. The first stage is to use Variational Autoencoders made up of convolutional and fully connected layers to create meaningful latent space representations of the individual parameters (flow velocity, thrust and torque). In the second stage the representations are used in combinations of two to try and predict the third parameter with a decoder neural network. Training for the first stage observes sensible learning over 100 epochs after the units of velocity and thrust are modified so that all parameters' standard deviations have the same order of magnitude as that of torque, O(1), which was the best performing parameter prior to the unit change. For the second stage the results vary. The results when trying to predict torque and thrust with 100 epochs of training are sensible, but velocity prediction was not achieved. The difficulties in predicting velocity could possibly be due to the fact that the standard deviation for velocity is 3 times smaller than those for thrust and torque after the unit change. Another possible explanation is the turbulence in the flow, which would be hard to account for with just two parameters. Future work could explore these ideas further in an attempt to produce a decoder capable of predicting flow velocity from thrust and torque. The GitHub repository used for this project is https://github.com/FpMarinov/TurbinesProject.

Acknowledgements

I would like to acknowledge my supervisor, Dr. Gerardo Aragon Camarasa, for all his help and support throughout this project. I would also like to acknowledge my friends and family for helping me keep a positive mindset throughout the COVID-19 pandemic.

Contents

Chapter 1

Introduction

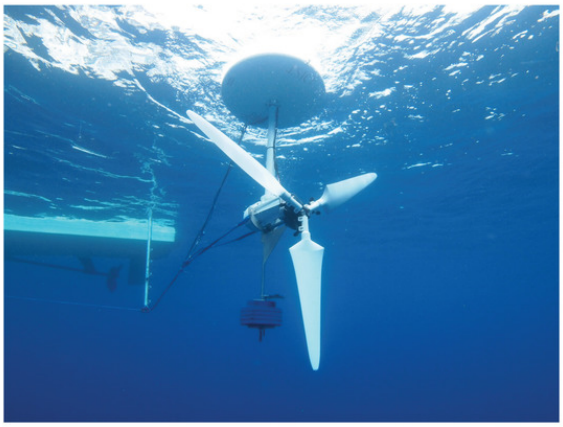

Figure 1.1: A sea turbine taken from https://www.mdpi.com/energies/energies-10-00702/article_deploy/html/images/energies-10-00702-g003-550.jpg.

1.1 Overview & Problem Statement

Marine turbines can be used for the production of electricity by transforming the kinetic energy of flowing water into electric energy [2]. If there are strong enough currents in the water, however, they could damage the turbines, which would lead to financial losses in repairing them [6]. Being able to predict changes in the thrust or torque applied to a marine turbine by using the flow velocity upstream combined with torque or thrust respectively could potentially help with lessening or completely avoiding damage to the turbine in use. The predictions could also help with the estimation of the amount of power that the turbine can produce. Both of these potential applications have great economic benefits.

In this dissertation, an approach is proposed to predict a marine turbine parameter (flow velocity, thrust or torque) by using a decoder neural network on the latent space representations of the other two parameters, produced from the training of Variational Autoencoders.

1.2 Objectives

The aim of this project is in part to use Variational Autoencoders in order to create a neural network, made up of convolutional and fully connected layers, which produces meaningful latent space representations of water flow velocity and the thrust and torque, applied to marine turbines by the flowing water. The other aim of the project is to use the constructed representations to try and predict the values of each of these parameters from those of the remaining two with a decoder neural network, constructed from convolutional and fully connected layers.

1.3 Dissertation Summary

The report structure is as follows. Chapter 1 is the introduction to the project context and aims. Chapter 2 gives the project background and discusses the referenced literature. Chapter 3 discusses the design and implementation of the software used for this project. Chapter 4 discusses the results of the experiments conducted using the developed software and their meaning together with suggestions for further work. Chapter 5 is the conclusion.

Chapter 2

Background & Literature Survey

2.1 What are Marine Turbines

The following discussion is based on [2]. A marine turbine functions like a windmill under flowing water, e.g. the sea. Water currents act on a turbine on contact with it, causing thrust and torque. The torque makes the turbine rotate, creating electricity via a generator. Since air is around 800 less dense than water, marine turbines generate 800 times the amount of energy that a windmill generates, given that the water and wind currents have the same velocity. This makes marine turbines more efficient than windmills. There are further advantages due to the fact that water currents are easier to predict than wind currents in terms of when they appear and their velocity. As discussed in Chapter 1, successfully predicting marine turbine parameters could prove economically beneficial by reducing damage to the turbine (when predicting thrust for example). This would in turn prolong its life cycle and reduce the need for repairs or replacements. This problem has several potential solutions as discussed in the following sections.

2.2 Related Work

In [6] the approach is to calculate the time it takes for a massive perturbation in flow velocity upstream of the turbine to get to the turbine. This is done by using cross-correlation functions, linear wave theory and wave-current interaction theory. The calculation is possible using the results of [6], but whether or not the calculated time would be enough to adequately prepare the turbine for impact is questionable since the prediction needs to be far enough in the future to give time for the preparation, taking into account the time needed to process the turbine data and for the calculations to be done, and for the mechanism for turbine defence to be initiated and completed.

Another potential approach would be to use present water conditions and anticipate future conditions via simulating the fluid dynamics in the future based on the Navier Stokes equations. A numerical technique which tries to solve the equations directly would be unfeasibly costly in terms of time and computational power, but a deep neural network which approximates fluid dynamics by using the Navier Stokes equations as a loss function like in [15] could potentially be viable. Such an approach would be substantially faster than using numerical methods [15]. Whether or not it would

be sufficiently precise, however, is questionable since the water environment the turbine functions in could undergo a variety of conditions and the neural network needs to be trained and tested in all of them in order to ensure the precision of the approximation.

In [1], deep neural networks are used to predict static pressure distributions of turbine blades. In the paper it is established that the prediction accuracy improves when using convolutional layers. The best results are provided from a combination of 4 convolutional layers and 2 fully connected layers. It is observed that the prediction accuracy is proportional to the number of cell activations, which is related to the amount of convolutional layers, i.e. the depth of the network. The paper shows that the turbine blade static pressure distribution can be predicted with accuracy by using convolutional networks.

In [7], deep learning networks, using the LSTM algorithm and Principal Component Analysis, are used to predict the amount of power, generated from a Wave Energy Converter. The results are substantially better than those produced by just using LSTM or different machine learning methods. It is established that high-frequency oscillating waves and long term wave features have a strong effect on the accuracy of the model. The paper displays the advantages of the used methods over other approaches.

2.3 Theoretical Background & Next Steps

Since [6] established a theoretical link between different marine turbine parameters like flow velocity and thrust, it should be possible to train a neural network to approximate this link in a similar way to how the Navier Stokes equations are approximated in [15]. The approximation should be better if it worked with several data points for the input parameters in order to capture their temporal behaviour. In such a case it would prove useful to filter the input and extract a meaningful representation of it in order to reduce the noise in the data and base the predictions on the overall temporal structures that the input parameters exhibit. This could be achieved via using Variational Autoencoders. These are neural networks, which encode (with an Encoder neural network) and then decode (with a Decoder neural network) an input [12]. The Encoder maps the input onto a Gaussian distribution [12]. The distribution is then sampled and the Decoder tries to reconstruct the input from the sample [12]. While training, Variational Autoencoders force their Encoder to come up with meaningful representations of the inputs via using as a term in their loss function the difference (measured in mean squared error for example) of the decoded encoded input (i.e. the output) and the original input [12]. The advantage of using Variational Autoencoders over ordinary Autoencoders (which map inputs onto points in the latent space instead of distributions) is that by mapping inputs onto a distribution in the latent space instead of a single point, they regularise their training [12]. This reduces overfitting by enforcing completeness (sampling random points from the latent space should give meaningful results when decoded) and continuity (sampling two points from the latent space, which are close to each other, should give two points, which are close to each other, when decoded) [12]. To achieve this, Variational Autoencoders add a KL divergence term to the loss, which measures how much the distribution in the latent space, given by the encoded input, differs from a standard Gaussian (one with mean, equal to 0, and variance, equal to 1) [12]. The regularisation process helps to make the latent space more meaningful and would help with potential future experiments, trying to generate new data from the neural networks [12].

The Variational Autoencoder's Encoder and Decoder could be constructed using convolutional net-

works. The advantage of using convolutional networks would be that they are good at capturing patterns in time and space, hence why they are widely used in image recognition [13]. This advantage is a consequence of the way they function. Convolutional neural networks usually take an image, represented as a matrix of numbers (or 3 matrices in the case of RGB images) and then pass a smaller matrix, called a kernel, through the bigger matrix, starting at the top left position, moving right (and eventually down to the leftmost position when they reach the end of the line) [13]. At each stage of the pass, each element in the image matrix, overlapping with the kernel, is multiplied by the corresponding kernel element and the results are summed together with a bias added at the end [13]. The result of this process forms an element of a new matrix [13]. When the element is calculated, the kernel moves to the right by an amount called the stride length (usually equal to 1) [13]. If it is at the end of the image horizontally, it moves to the leftmost position vertically down by one element until it reaches the vertical end as well [13]. This is done to reduce the size of the image, while capturing its essential patterns [13]. In a convolutional neural network, this process is usually repeated several times (in different layers), with the idea being to capture patterns, made up of patterns, made up of patterns, etc. [13]. Sometimes the network pads the image with zeros before it passes the kernel through it in order to get a result of a particular size [13]. Another addition that is sometimes made to the network is to use several different kernels at each stage in order to change the number of channels of the resulting matrix, i.e. producing several convolved matrices, constructed with the different kernels, essentially looking for different patterns [13].

Convolutional neural networks can be used with the turbine parameter data in the following way. The data can be split into sequences of a given size, representing 1 dimensional ”images”. These sequences can then be passed through a 1 dimensional convolutional neural network to be encoded. The resulting distributions can then be sampled. The samples can be decoded by being passed through another 1 dimensional convolutional neural network. This would be similar to what is done in [9], where variational autoencoders, constructed with convolutional networks, are used in order to find meaningful representations of handwritten digit images for classification purposes. The approach that is taken could be modified in order to find meaningful representations of marine turbine parameter sequences for regression purposes. These representations could then be used together for two parameters in order to try and predict the third. For example predicting thrust from the representations of torque and velocity. This is the approach taken in this project.

2.4 Data

The data used for the project was obtained from Dr. Gerardo Aragon Camarasa (Lecturer in Autonomous and Socially Intelligent Robotics, Computing Science, University of Glasgow) via a collaboration with the Energy Systems Research Unit of the University of Strathclyde. It represents measurements of the thrust, torque and flow velocity of marine turbines in a test environment (see Figs. 2.1 and 2.2, for more detail see [6]), measured in SI units. The velocities were measured at 3 distances upstream of the turbine (1D = 1m, 2D = 2m and 3D = 3m, where D is the diameter of the turbine, see Fig. 2.1) with a Laser Doppler Velocimeter (LDV), with the corresponding rotor torque and thrust being measured with an Applied Measurements transducer. The turbine was tested at three depths (0.65m, 0.825m and 1m, see Fig. 2.3), with this project using the data from 1m under water. At each depth the flow velocity was measured at the top, middle and bottom of the turbine rotor (see Fig. 2.3), with this project using the data from the middle.

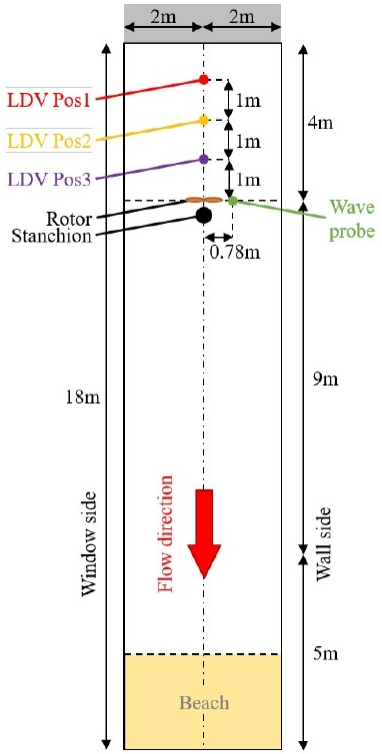

Figure 2.1: Set up of test tank. The image is taken from [6].

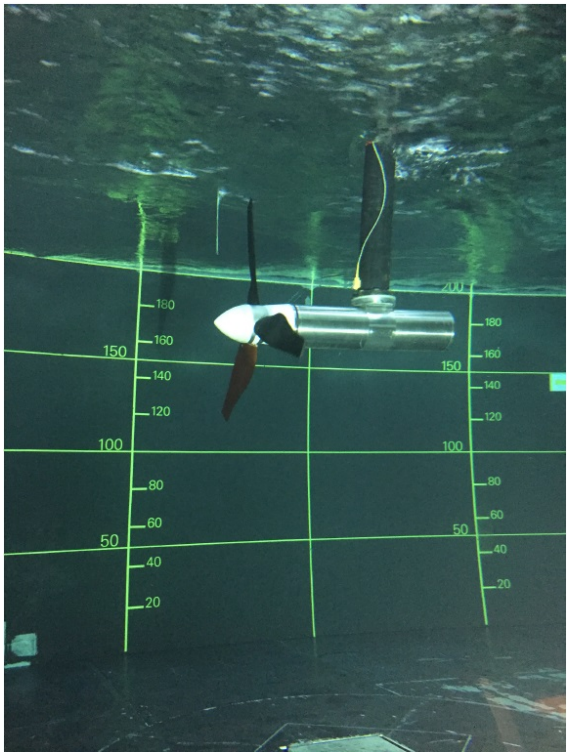

Figure 2.2: Test turbine in tank. The image is taken from [6].

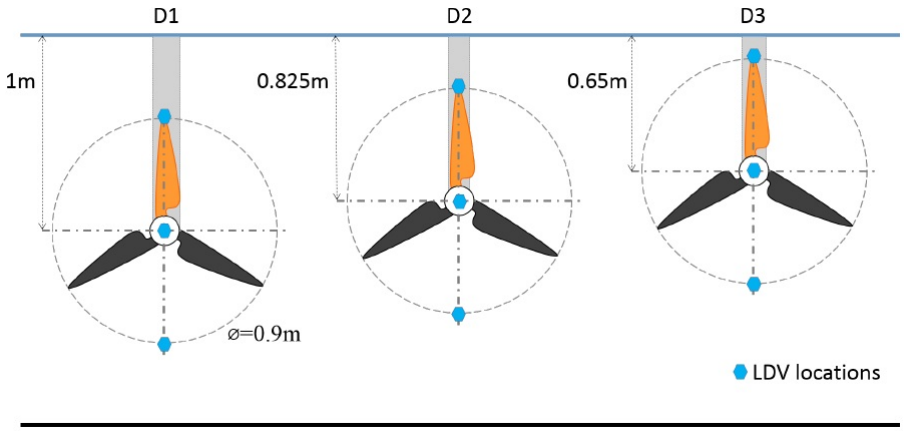

Figure 2.3: Rotor schematic, showing the 3 testing depths: D1, D2 and D3. The blue hexagons show the top, middle and bottom LDV positions. The image is taken from [6].

Chapter 3

Design & Implementation

3.1 System Architecture and Design

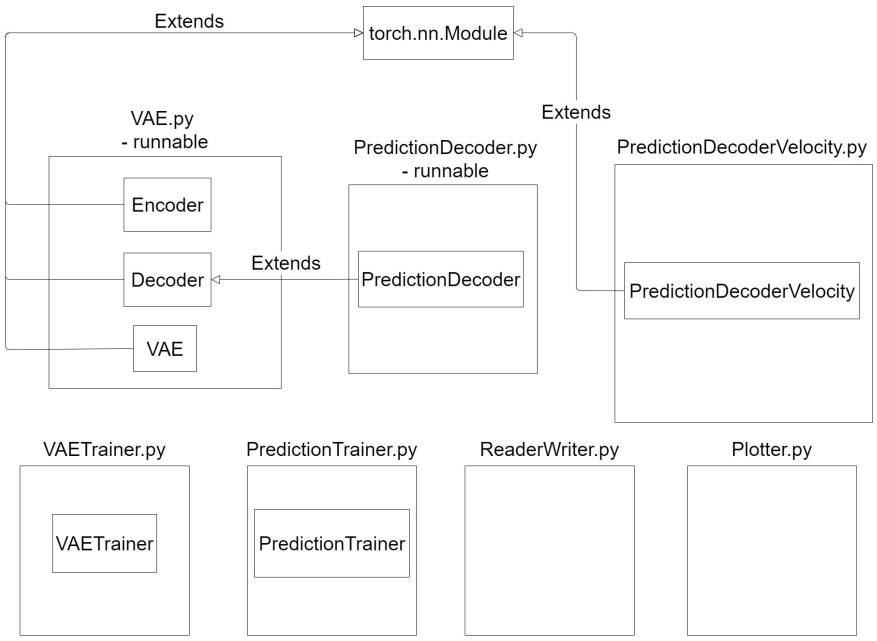

Figure 3.1: Overall architecture diagram. The squares with file names on top represent Python files (”-runnable” in the name means that the file is runnable), the rectangles with class names inside represent classes.

The GitHub repository used for this project is:
https://github.com/FpMarinov/TurbinesProject.
The software in the project is split into 7 Python files: `VAE.py`, `VAETrainer.py`, `PredictionDecoder.py`, `PredictionDecoderVelocity.py`, `PredictionTrainer.py`, `ReaderWriter.py` and `Plotter.py`. Of those, the runnable files are `VAE.py` and `PredictionDecoder.py`, which are used for the variational autoencoder and parameter prediction functionalities respectively. The rationale behind the chosen software structure is to have proper separation

of concerns. There are also several supplementary files. Data.txt is the name of the file in the main directory in which the data is expected to be by the Python files. It is expected in SI units in CSV format with headers and tab as a delimiter, since that was the format the provided data was in. The expected order is flow velocity, thrust, torque. In the case that VAE is not used for training, it expects weights in the main directory with the following names: vae_net_velocity.pth, vae_net_torque.pth and vae_net_thrust.pth. In the case that PredictionDecoder or PredictionDecoderVelocity are not used for training, they expect weights in the main directory with the following names: vae_net_prediction_decoder_thrust.pth, vae_net_prediction_decoder_torque.pth and vae_net_prediction_decoder_velocity.pth. Also, regardless of whether they are used for training or not, PredictionDecoder and PredictionDecoderVelocity require vae_net_velocity.pth, vae_net_torque.pth and vae_net_thrust.pth in order for the encoders to work. VAE, PredictionDecoder and PredictionDecoderVelocity were based on [10]. The reason for this is the fact that the application for the original software (described in [9]) was similar to the desired application for VAE, PredictionDecoder and PredictionDecoderVelocity. Hence, the original framework was adapted into the initial prototypes and used as a starting point for development. VAETrainer and PredictionTrainer were based on [11] for similar reasons. The individual Python file responsibilities are as follows.

VAE

VAE contains the variational autoencoder class, VAE, VAE's encoder class, Encoder, VAE's decoder class, Decoder, and helper functions. The VAE class is responsible for the encoding data into a gaussian distribution, sampling the distribution and then decoding the data.

When the file is run, the behaviour is governed by the changeable parameters:
- `data_type` (string): sets the data type, with which the program will work from among "velocity", "thrust" and "torque". The data is read from Data.txt.
- `mode` (string): decides the mode of action of the program:
(a) `mode` = "train", the variational autoencoder is trained with the chosen data type, saves its weights to a file with a name given by the `weights_path` variable (vae_net_velocity.pth, vae_net_torque.pth or vae_net_thrust.pth), saves the training and validation losses to loss_record.csv, saves the training and validation mean squared errors to mse_loss_record.csv and produces plots of the losses and mean squared errors.
(b) `mode` != "train" (or if `mode` = "train" and after the training is done), the variational autoencoder uses pretrained weights from a file with a name given by the `weights_path` variable to reconstruct the chosen data type and shows a scatter plot, having the original value on the x axis and the reconstructed value on the y axis.
- `epochs` (int): decides the number of epochs the variational autoencoder trains for if it is in train mode.
- `plot_loss_1_epoch_skip` (bool): decides whether to produce additional plots of the training and validation losses and mean squared errors, skipping the first epoch. It should only be used when the chosen number of epochs is 2 or more.
- `plot_loss_50_epoch_skip` (bool): decides whether to produce additional plots of the training and validation losses and mean squared errors, skipping the first 50 epochs. It should only be used when the chosen number of epochs is 50 or more.
- `validation_data_fraction` (float): the fraction of the data which is to be used for validation.

VAE was designed with these changeable variables to provide customizability to the user. The net-

work is meant to work with 3 data types, hence it makes sense for the user to be able to pick the data type as a variable instead of having 3 separate files. The program is meant to be used both for training and for testing after the training is finished, hence the reason for a variable to distinguish between the two cases. The appropriate amount of epochs is something to be established on a case by case basis, hence why the epoch number is left as a changeable variable. The training and validation losses for the first epoch are usually disproportionately large when compared to later losses, hence why the option is given to have a separate loss plot disregarding the first epoch. An option is given to have another extra loss plot disregarding the first 50 epochs in case the user wants to view only the effects of late stage training. The validation data fraction is left as a changeable variable because different data sets have different sizes and in order to ensure divisibility the validation fraction may require changing, also the user may wish to have a greater or lesser fraction for validation. The implemented architecture (see section 3.2) is motivated by the discussion in Chapter 2.

VAETrainer
VAETrainer contains the VAETrainer class which handles the training of the variational autoencoder, VAE, and records the training and validation losses and mean squared errors.

PredictionDecoder
PredictionDecoder contains the prediction decoder class, PredictionDecoder, and helper functions. The PredictionDecoder class is responsible for predicting a parameter (from among velocity, thrust and torque) from the encoded versions of the other two parameters.

When the file is run, the behaviour is governed by the changeable parameters:
- `data_to_predict_type` (string): sets the data type to be predicted by the program from among ”velocity”, ”thrust” and ”torque”. The data is read from Data.txt.
- `mode` (string): decides the mode of action of the program:
(a) `mode` = ”train”, the prediction decoder is trained with the chosen data type, saves its weights to a file with a name given by the `weights_path_decoder` variable (vae_net_prediction_decoder_thrust.pth, vae_net_prediction_decoder_torque.pth or vae_net_prediction_decoder_velocity.pth), saves the training and validation losses to loss_record.csv and produces plots of the losses.
(b) `mode` != ”train” (or if `mode` = ”train” and after the training is done), the prediction decoder uses pretrained weights from a file with a name given by the `weights_path_decoder` variable to predict the chosen data type from the other two and shows a scatter plot, having the original value on the x axis and the reconstructed value on the y axis.
- `epochs` (int): decides the number of epochs the prediction decoder trains for if it is in train mode.
- `plot_loss_1_epoch_skip` (bool): decides whether to produce an additional plot of the training and validation losses skipping the first epoch. It should only be used when the chosen number of epochs is 2 or more.
- `plot_loss_50_epoch_skip` (bool): decides whether to produce an additional plot of the training and validation losses skipping the first 50 epochs. It should only be used when the chosen number of epochs is 50 or more.
- `validation_data_fraction` (float): the fraction of the data which is to be used for validation.

The rationale for the changeable variables in PredictionDecoder is similar to the rationale for the changeable variables in VAE. The implemented architecture (see section 3.2) is almost the same

as that of VAE's Decoder (see section 3.2). The only change is the fact that PredictionDecoder takes four variables as an input (two means and two log variances), instead of one sample (from the Gaussian distribution). During development both decoding using sampling and using the means and log variances directly were tested. The direct approach without sampling resulted in lower losses and hence was chosen as the final approach. Using similar architecture to the one of VAE's Decoder was done under the hypothesis that since such a Decoder could reconstruct a single parameter from its representation, it is plausible that it could reconstruct a single parameter from the representations of the other two, based on the link between the parameters discussed in Chapter 2.

PredictionDecoderVelocity
PredictionDecoderVelocity contains the prediction decoder class for velocity, PredictionDecoderVelocity. It is used when `PredictionDecoder.py` is run with `data_to_predict_type` = "velocity". A separate decoder was constructed for the prediction of velocity in an attempt to improve the results, since velocity was the only parameter exhibiting poor prediction while using PredictionDecoder. The implemented architecture (see section 3.2) is of a similar type to that of PredictionDecoder (see section 3.2). The only change was to make the network less deep by one convolutional layer (trials were done with deeper networks, but resulted in worse losses) and to modify the number of channels in the remaining layers (see section 3.2 for more details).

PredictionTrainer
PredictionTrainer contains the PredictionTrainer class which handles the training of the prediction decoders, PredictionDecoder and PredictionDecoderVelocity, and records the training and validation losses. Also contains the encode_inputs helper function, which encodes inputs.

ReaderWriter
ReaderWriter contains functions for the reading of velocity, thrust and torque data from Data.txt and losses from loss_record.csv and mse_loss_record.csv, and the writing of losses to loss_record.csv and mse_loss_record.csv. Losses are written to files after the training is complete so that the user may have a record of them in case they are needed in the future. The reading capabilities are needed in order to be able to construct data lists with which to train the neural networks.

Plotter
Plotter contains functions for the production of plots, used to visualise data points, training losses, reconstructions and predictions. Also it contains a function for the printing of the mean and standard deviation of a data set. The functionality for the plots and statistics was developed so that the user could view and judge the performance of the neural networks.

3.2 Implementation Details

The neural networks are constructed using PyTorch with Adam as the optimizer, the loss and scatter plots are constructed using matplotlib.pyplot (the rest of the plots are constructed using [5], [17] and Microsoft Paint), file input output is achieved using the csv module. All convolutional layers have stride and padding equal to 1 and kernel size equal to 3. The neural networks' depth and variables

were decided by extensive trial and error using different configurations, with the configurations resulting in consistently lower losses within the first 10 epochs being chosen over the others. The specific implementation details of the neural networks and their trainers are given in the following. For all Encoder/Decoder diagrams: channel sizes are below convolutional layers, data sequence sizes are in the upper left of convolutional layers, 1 dimensional tensor length is below the fully connected layers and batch sizes are not represented.

VAE

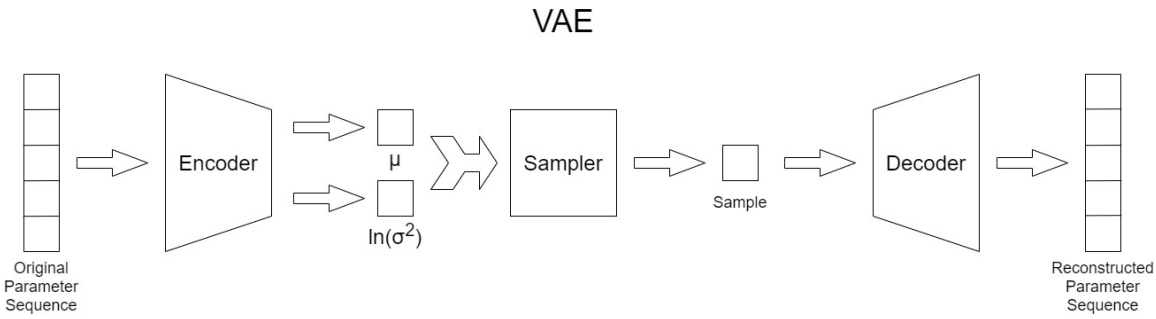

Figure 3.2: Design diagram of VAE. μ is the mean and σ is the standard deviation of the Gaussian distribution.

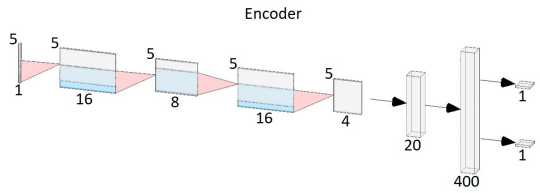

Figure 3.3: Design diagram of VAE's Encoder.

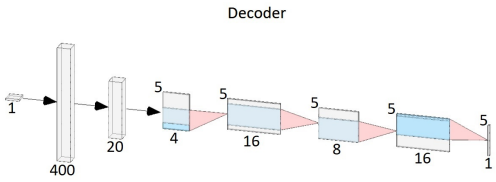

Figure 3.4: Design diagram of VAE's Decoder.

VAE's Encoder takes a tensor with 3 size dimensions: batch dimension (set to 5), channel dimension (set to 1) and data sequence dimension (set to 5). It passes this tensor through 4 convolution layers with a ReLU transformation after every convolution. The convolution layers take the input tensor from 1 channel to 16, to 8 channels, to 16 channels, to 4 channels. After the convolution layers the convolved tensor is flattened, giving it 2 size dimensions: batch dimension (set to 5) and data dimension (set to 20), before passing through a fully connected neural network layer changing its data dimension to 400. After the fully connected layer, the tensor passes through another ReLU transformation. Then the result passes through two separate fully connected neural network layers, changing the data dimension from 400 to 1. One of the layers produces the distribution mean and the other produces the log variance.

After the input has been encoded into a distribution, that distribution is sampled and the sample is passed through the Decoder.

VAE's Decoder takes a tensor with 2 size dimensions: batch dimension (set to 5) and data dimension (set to 1). It passes this tensor through a fully connected neural network layer, changing the data dimension to 400, then it does a ReLU transformation. The tensor then passes through another fully connected neural network layer, changing the data dimension to 20, followed by a ReLu transformation. Then the tensor is reformatted, giving it 3 size dimensions: batch dimension (set to 5), channel dimension (set to 4) and data sequence dimension (set to 5). After this the tensor goes through 4 convolution layers with a ReLU transformation after each one. The convolution layers take the input tensor from 4 channels to 16, to 8 channels, to 16 channels, to 1 channel. At this point the input to the Encoder has been reconstructed.

VAETrainer

VAETrainer trains VAE for the chosen number of epochs by going through each batch in a training data loader, passing it through VAE, calculating the loss (given by the sum of the mean squared error and the KL divergence, divided by the number of data points), doing backpropagation, using the optimizer and recording the losses and mean squared errors. After each epoch the trainer evaluates the model by recording the losses and mean squared errors on the validation data (in the validation data loader). At the end the trainer returns the losses and mean squared errors as lists.

PredictionDecoder

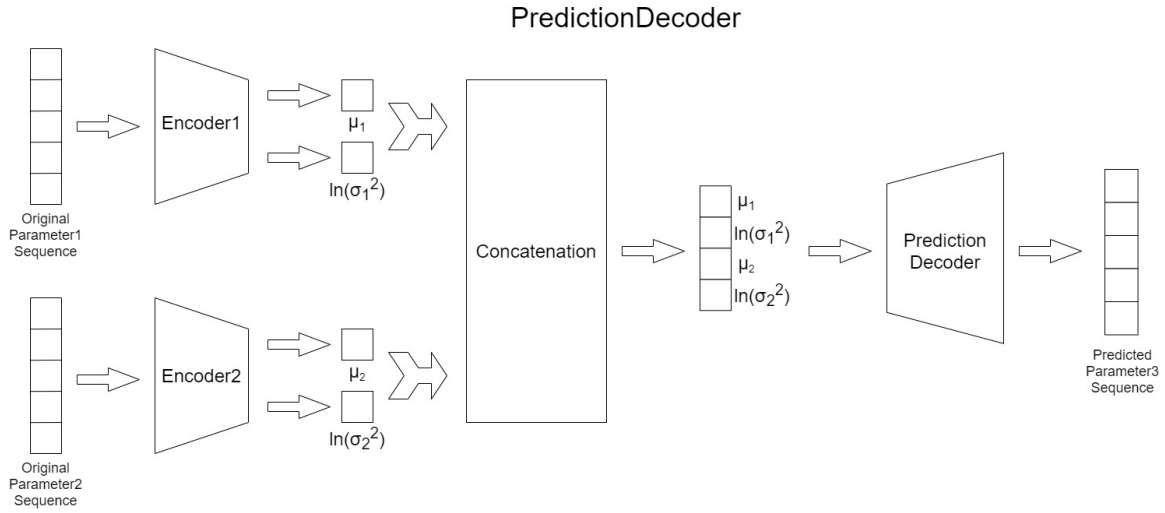

Figure 3.5: Design diagram of how PredictionDecoder is used together with VAE's Encoder for predicting a parameter. μ_i are the means and σ_i are the standard deviations of the Gaussian distributions.

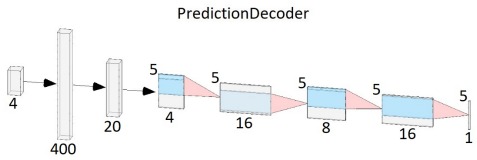

Figure 3.6: Design diagram of PredictionDecoder.

PredictionDecoder inherits from the Decoder in VAE. The only thing it changes is that the initial fully connected network transformation takes 4 parameters (2 means and 2 log variances) instead of 1 (1 sample).

PredictionDecoderVelocity

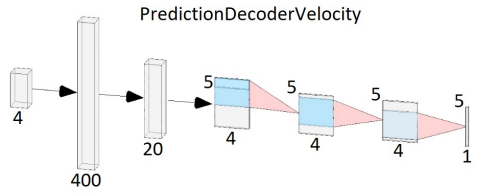

Figure 3.7: Design diagram of PredictionDecoderVelocity.

PredictionDecoderVelocity works just like PredictionDecoder with the only exception being that it has 3 convolution layers instead of 4 and they take the input from 4 channels, to 4 channels, to 4 channels, to 1 channel.

PredictionTrainer

PredictionTrainer works similarly to VAETrainer with several noteworthy differences. It has 3 training and 3 validation data loaders. 2 for the parameters that are to be encoded and 1 for the parameter that is to be predicted. Each batch for the parameters to be encoded passes through their respective pretrained encoders, producing 2 log variances and 2 means for each data sequence in the batch for the parameter to be predicted. The log variances and means are concatenated and then passed through either PredictionDecoder or PredictionDecoderVelocity. The losses are then calculated using the mean squared error as a loss criterion. Afterwards backpropagation is done and the optimizer is used. The loss is then recorded. Evaluation is done after every epoch just like in VAETrainer and the validation loss is recorded. At the end the trainer returns the losses as a list.

Chapter 4

Experiments

4.1 Overview

A series of experiments were conducted with the designed neural networks, using velocity data from all three distances upstream of the turbine (1D = 1m, 2D = 2m and 3D = 3m), together with the corresponding rotor torque and thrust measurements. Throughout this chapter the gathered statistical data is represented with μ for means, σ for standard deviations and MSE for Mean Squared Errors. Means for losses and MSE's are given for the final epoch of training and validation. Means and Standard Deviations for errors measure μ and σ for the signed errors (original data points subtracted from the reconstructions/predictions) measured after the neural networks are run on all the data after training has been completed.

For the experiments in sections 4.2 and 4.4 (using a variational autoencoder to encode and then decode/reconstruct a given parameter) the data was separated into training and validation data, with the validation data being formed of the latter $v\%$ of the whole data for all 3 parameters, where v represents the chosen percentage for the given experiment. Afterwards the data points in both the training and validation data sets were grouped together in sequences of 5. Each sequence was used as both the input and ideal output for the neural network, with which to compare the reconstruction. For the experiments in sections 4.3 and 4.5 (using pretrained VAE encoders for two parameters to produce means and log variances, which are then passed through a specialised decoder to predict the third parameter) the same was done, with the only difference being that the network takes two input sequences (measuring two different parameters) and outputs a third sequence (measuring the third parameter), hence the separation into training and validation data was done for all 3 parameters' data sets. Two data sets were used for the inputs and the third was used as the ideal output, with which to compare the prediction.

The networks were trained for 100 epochs on the training data in all experiments (using the final program versions). In the experiments, described in sections 4.2 and 4.4, for each epoch the average loss (the sum of the mean squared error and the KL divergence divided by the number of data points) and average mean squared error per epoch were recorded for both the training and validation data. In the experiments, described in sections 4.3 and 4.5 only the average loss (mean squared error) per epoch was recorded for the training and validation data.

In all experiments after the training was complete, the network was used on all the data (training

and validation data combined). The resulting signed error's (the original value subtracted from the reconstruction/prediction) mean and standard deviation were recorded. In addition a scatter plot was produced in all experiments, showing the original data point value on the x axis and the reconstructed/predicted data point value on the y axis, together with the straight black line y=x, which represents the ideal reconstruction/prediction. The reason for combining the training and validation data after the training was to observe whether the scatter plots would show meaningful reconstructions/predictions even though the neural network was trained only on the training data.

When the initial versions of the experiments described in section 4.2 were conducted using the prototypes of VAE for 10 epochs, a discrepancy was observed in the reconstruction scatter plots of velocity and thrust, compared to the reconstruction scatter plot of torque (see Figures 4.1, 4.2 and 4.3 below).

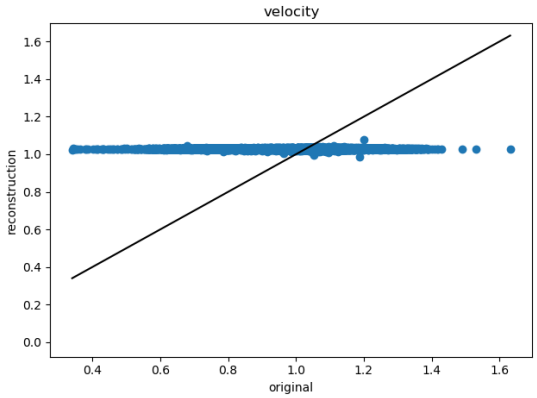

Figure 4.1: Reconstruction of 1D velocity from VAE prototype.

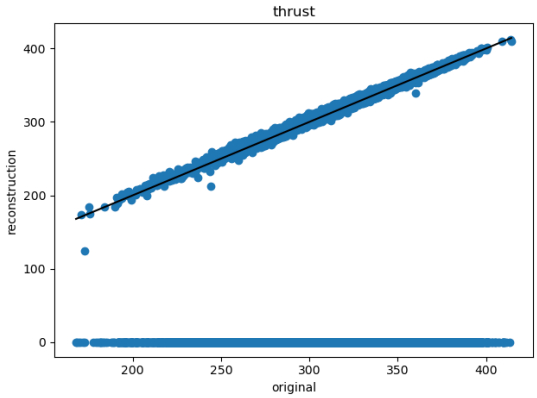

Figure 4.2: Reconstruction of 1D thrust from VAE prototype.

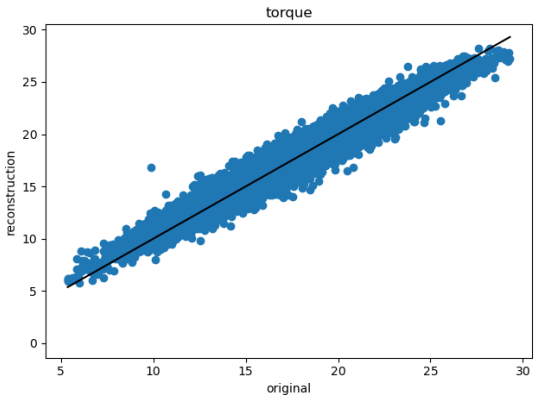

Figure 4.3: Reconstruction of 1D torque from VAE prototype.

An attempt was made to fix this discrepancy by bringing velocity and thrust to the same order of magnitude as torque, which was reconstructing along the desired line y=x. To do this velocity values were multiplied by 10 and thrust values were divided by 10, effectively changing the units of both. The rationale behind this was the fact that for all the data (1D, 2D and 3D) the standard deviations

of thrust, torque and velocity are approximately 30, 3 and 0.1 respectively. Scaling the values by a factor, also scales their mean and hence their standard deviation by the same factor. Bringing the standard deviations to the same order of magnitude ($\sigma_{thrust} \approx 3$, $\sigma_{torque} \approx 3$, $\sigma_{velocity} \approx 1$) was expected to bring the performance of the neural network on the three parameters closer to alignment. The scaling succeeded in fixing the discrepancy as can be observed from the plots in section 4.2. Hence, for all remaining experiments the input values of velocity were multiplied by 10 and those of thrust were divided by 10, which as expected changed the reconstructed/predicted values for those parameters in the same way.

4.2 1D VAE

This was the first experiment to be conducted. The aim was to create a variational autoencoder, that can create meaningful representations of the parameters and reconstruct them accurately. This was done to test the hypothesis that the creation of such a representation and subsequent reconstruction are possible.

The VAE neural network in `VAE.py` was trained and used on the 1D velocity data and the corresponding rotor torque and thrust measurements with $v = 20$.

Thrust

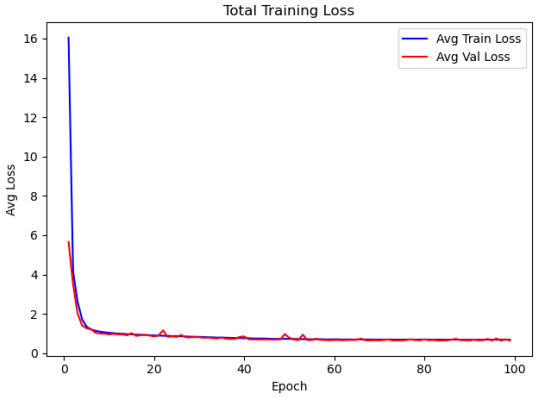

Figure 4.4: Average training loss per epoch for all epochs of training on thrust except the first.

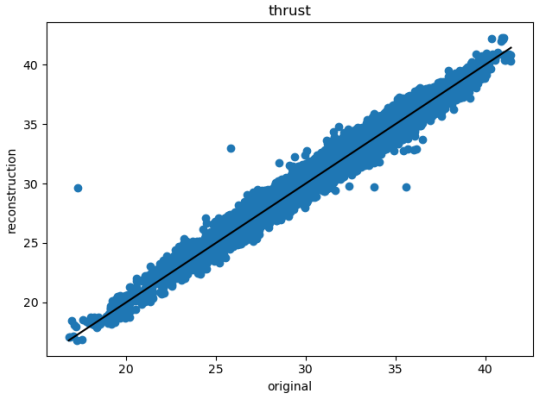

Figure 4.5: Scatter plot of the reconstructed thrust against the original.

Training μ_{Loss}	Validation μ_{Loss}	Training MSE	Validation MSE	μ_{Error}	σ_{Error}
0.69	0.65	0.25	0.22	0.06	0.47

Torque

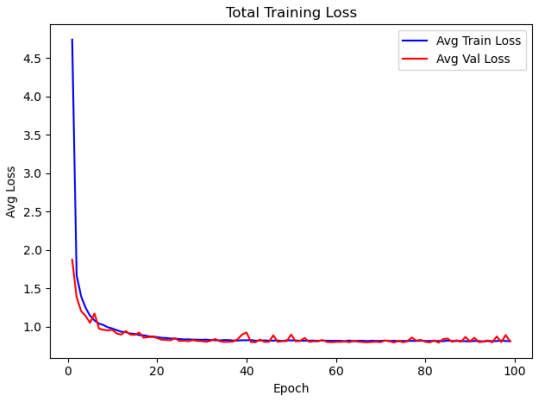

Figure 4.6: Average training loss per epoch for all epochs of training on torque except the first.

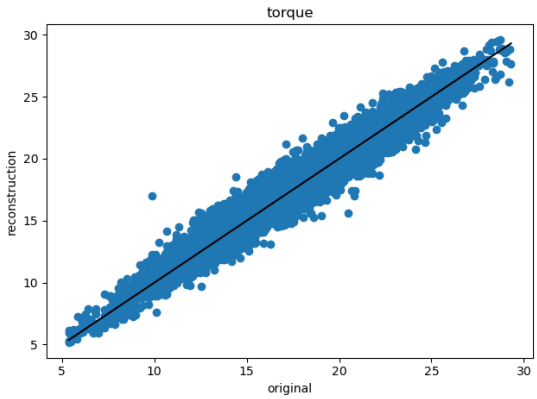

Figure 4.7: Scatter plot of the reconstructed torque against the original.

Training μ_{Loss}	Validation μ_{Loss}	Training MSE	Validation MSE	μ_{Error}	σ_{Error}
0.81	0.81	0.38	0.37	0.09	0.60

Velocity

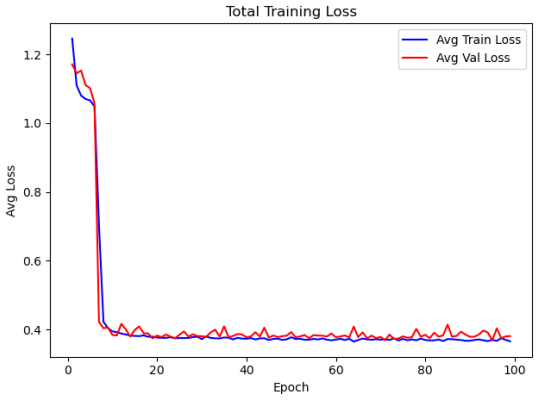

Figure 4.8: Average training loss per epoch for all epochs of training on velocity except the first.

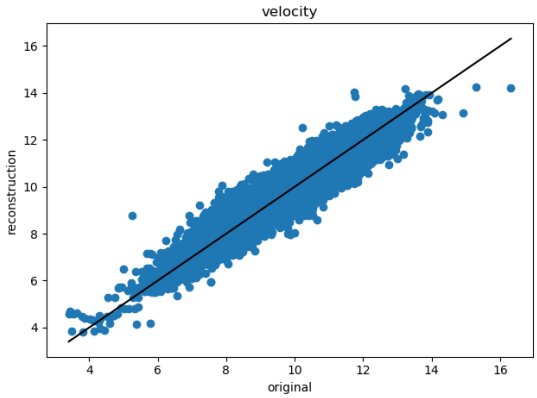

Figure 4.9: Scatter plot of the reconstructed velocity against the original.

Training μ_{Loss}	Validation μ_{Loss}	Training MSE	Validation MSE	μ_{Error}	σ_{Error}
0.37	0.38	0.16	0.16	0.04	0.39

Discussion

It appears that the loss functions cannot decrease further substantially in these experiments with the current network set up. Training observes sensible learning over 100 epochs, although the reconstruction of velocity seems to be doing worse at the end of the distribution (even though it

has the lowest mean squared error), possibly due $\sigma_{velocity}$ being 3 times smaller than σ_{thrust} and σ_{torque} after the unit change.

4.3 1D Prediction using 1D VAE Encoder

This was the second experiment to be conducted. The aim was to predict one parameter from the representations of the other two, constructed by the encoders produced from the previous experiment. This was done to test the hypothesis that a parameter can be determined from the representations of the other two parameters.

The neural networks in `PredictionDecoder.py` and `PredictionDecoderVelocity.py` (using the Encoder from `VAE.py` pretrained on 1D velocity data and the corresponding rotor torque and thrust) were trained and used on all combinations of two parameters from the 1D velocity data, corresponding rotor torque and thrust, trying to predict the third parameter with $v = 20$.

Thrust

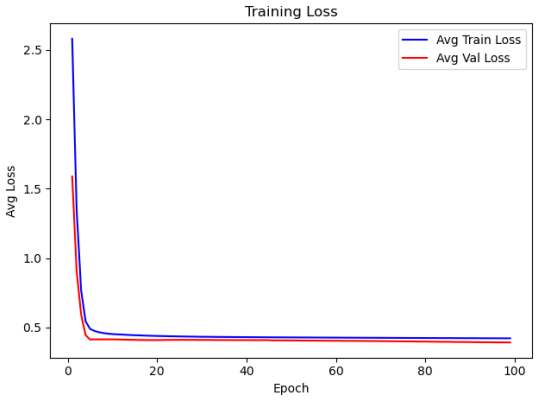

Figure 4.10: Average training loss per epoch for all epochs of training to predict thrust except the first.

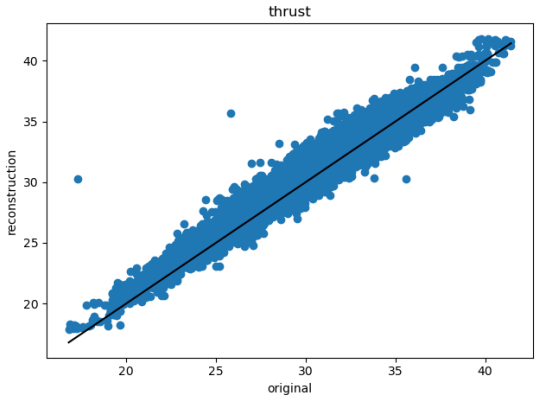

Figure 4.11: Scatter plot of the predicted thrust against the original.

Training μ_{Loss} (MSE)	Validation μ_{Loss} (MSE)	μ_{Error}	σ_{Error}
0.42	0.39	0.18	0.60

Torque

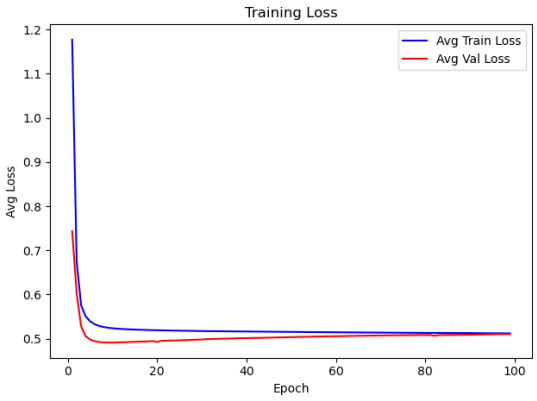

Figure 4.12: Average training loss per epoch for all epochs of training to predict torque except the first.

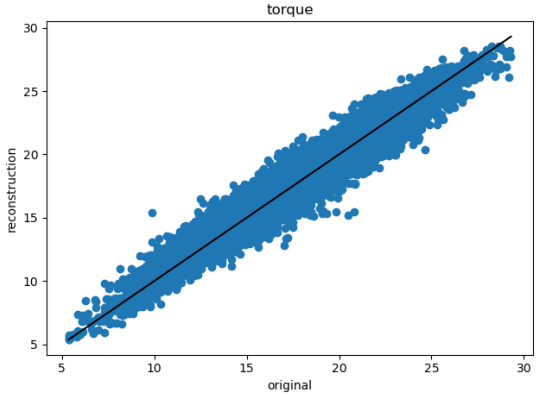

Figure 4.13: Scatter plot of the predicted torque against the original.

Training μ_{Loss} (MSE)	Validation μ_{Loss} (MSE)	μ_{Error}	σ_{Error}
0.51	0.51	0.16	0.70

Velocity

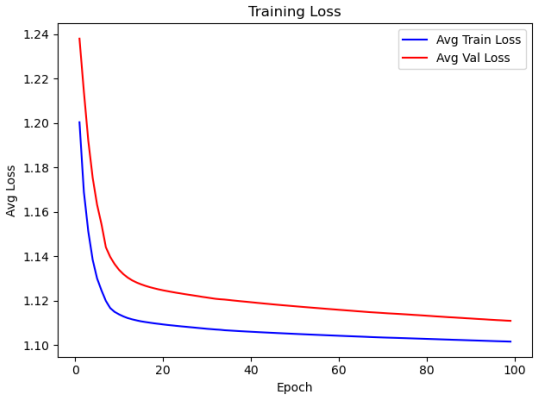

Figure 4.14: Average training loss per epoch for all epochs of training to predict velocity except the first.

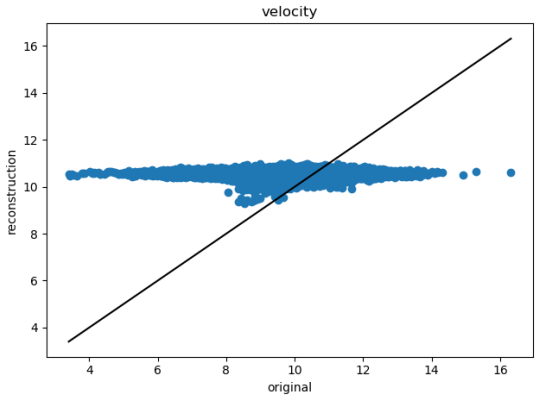

Figure 4.15: Scatter plot of the predicted velocity against the original.

Training μ_{Loss} (MSE)	Validation μ_{Loss} (MSE)	μ_{Error}	σ_{Error}
1.10	1.11	0.26	1.03

Discussion

It appears that the loss functions cannot decrease further substantially in these experiments with the current network set up. Training observes sensible learning over 100 epochs for thrust and torque,

although the prediction for velocity does not appear to have the correct shape for the scatter plot, possibly due $\sigma_{velocity}$ being 3 times smaller than σ_{thrust} and σ_{torque} after the unit change, hence the data points being more focused around the mean equal to 10, where the prediction is the best. Another possible explanation for the difficulty with the prediction could be the turbulence in the water flow, which would make it harder to predict from thrust and torque. Further research can be done with data from a laminar flow or a wider spread in flow velocity to explore these ideas.

When comparing the results with those from section 4.2 it can be seen that the mean squared errors for thrust and torque increased by 0.1 and for velocity it increased by 0.9. This is to be expected since the experiments in section 4.2 use data from the same parameters for reconstruction while the experiments in this section try to predict a given parameter from the other two.

4.4 1D & 3D VAE

This was the third experiment to be conducted. There were two aims for doing it. One was to establish whether the variational autoencoder would become better at reconstruction by training on more varied data, the other was to prepare encoders for the next experiment, described in section 4.5. The hypothesis under which this experiment was conducted was that more varied data would result in more accurate reconstructions.

The neural network in `VAE.py` was trained and used on the 1D and 3D (combined together) velocity data and the corresponding rotor torque and thrust measurements with $v = 25$.

Thrust

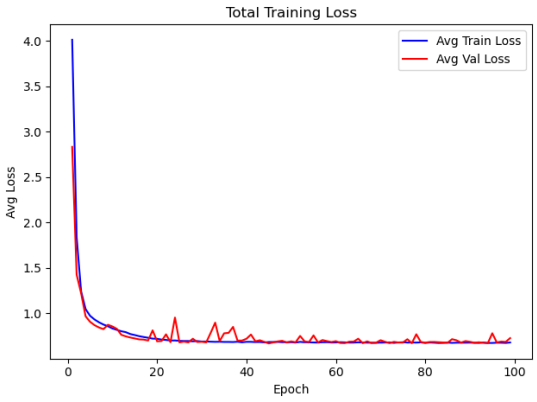

Figure 4.16: Average training loss per epoch for all epochs of training on thrust except the first.

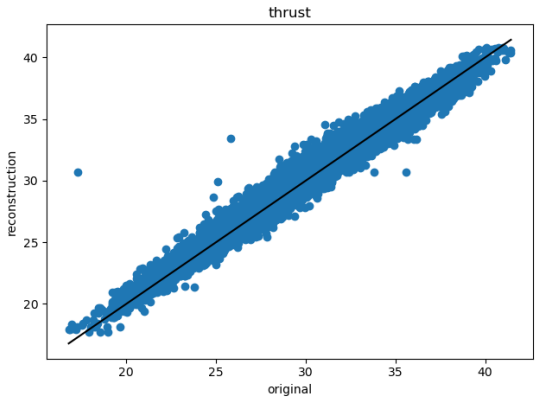

Figure 4.17: Scatter plot of the reconstructed thrust against the original.

Training μ_{Loss}	Validation μ_{Loss}	Training MSE	Validation MSE	μ_{Error}	σ_{Error}
0.68	0.73	0.25	0.27	0.22	0.46

Torque

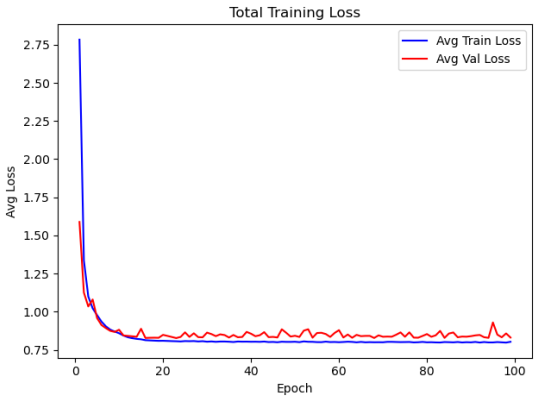

Figure 4.18: Average training loss per epoch for all epochs of training on torque except the first.

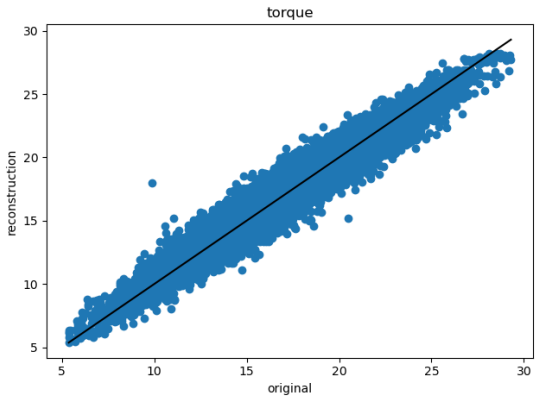

Figure 4.19: Scatter plot of the reconstructed torque against the original.

Training μ_{Loss}	Validation μ_{Loss}	Training MSE	Validation MSE	μ_{Error}	σ_{Error}
0.80	0.83	0.37	0.38	-0.03	0.59

Velocity

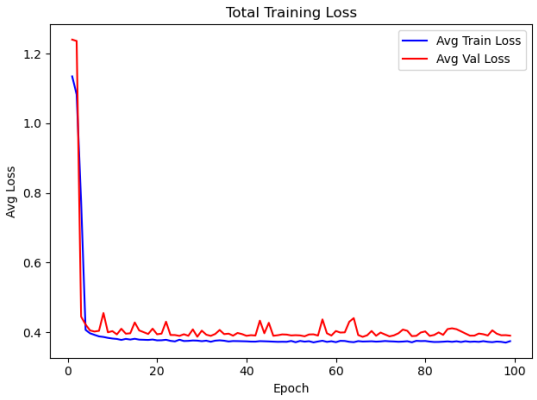

Figure 4.20: Average training loss per epoch for all epochs of training on velocity except the first.

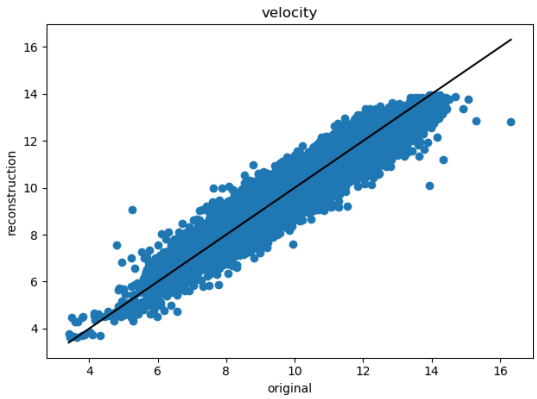

Figure 4.21: Scatter plot of the reconstructed velocity against the original.

Training μ_{Loss}	Validation μ_{Loss}	Training MSE	Validation MSE	μ_{Error}	σ_{Error}
0.37	0.39	0.16	0.16	-0.05	0.39

Discussion

The results for the losses and mean squared errors are similar to those observed in section 4.2. No substantial improvement is gained from using more varied data in this way.

It appears that the loss functions cannot decrease further substantially in these experiments with the current network set up. Training observes sensible learning over 100 epochs, although the reconstruction of velocity seems to be doing worse at the end of the distribution (even though it has the lowest mean squared error), possibly due $\sigma_{velocity}$ being 3 times smaller than σ_{thrust} and σ_{torque} after the unit change.

4.5 2D Prediction using 1D & 3D VAE Encoder

This was the final experiment to be conducted. The aim was to find out if it is possible to predict a 2D parameter from the representations of the other 2 2D parameters using encoders that were trained on data at the other distances (1D and 3D). The hypothesis under which the experiment was conducted was that it is possible to predict parameter data from a given distance from the representations of the other parameters (at the same distance), encoded by encoders trained on other distances.

The Encoder from `VAE.py` was trained on 1D and 3D (combined together) velocity data and the corresponding rotor torque and thrust. Afterwards, The neural networks in `PredictionDecoder.py` and `PredictionDecoderVelocity.py` (using the Encoder from `VAE.py` without modifying the weights further) were trained and used on all combinations of two parameters from the 2D velocity data, corresponding rotor torque and thrust, trying to predict the third 2D parameter with $v = 24$.

Thrust

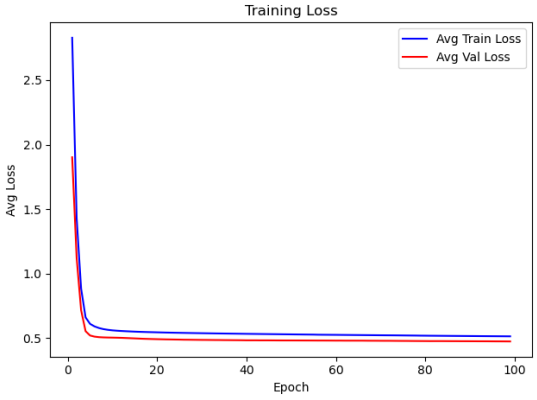

Figure 4.22: Average training loss per epoch for all epochs of training to predict thrust except the first.

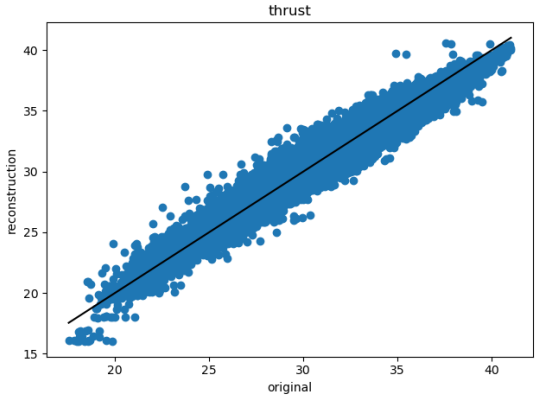

Figure 4.23: Scatter plot of the predicted thrust against the original.

Training μ_{Loss} (MSE)	Validation μ_{Loss} (MSE)	μ_{Error}	σ_{Error}
0.51	0.47	-0.20	0.66

Torque

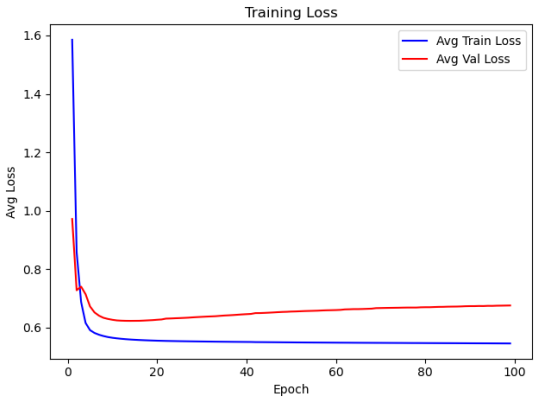

Figure 4.24: Average training loss per epoch for all epochs of training to predict torque except the first.

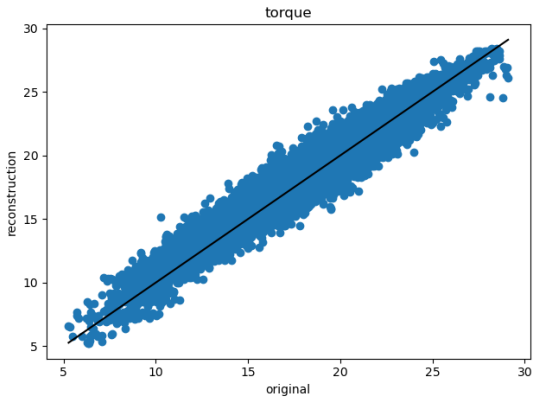

Figure 4.25: Scatter plot of the predicted torque against the original.

Training μ_{Loss} (MSE)	Validation μ_{Loss} (MSE)	μ_{Error}	σ_{Error}
0.56	0.62	0.34	0.72

For torque in this series of experiments, unlike all the other experiments, there appears to be overfitting past the 15th epoch (with validation loss 0.62) as can be seen from Fig.4.24. Because of this the network was retrained for 15 epochs and the prediction shown in Fig.4.25 and the data in the table above are both from the end of that retraining.

Velocity

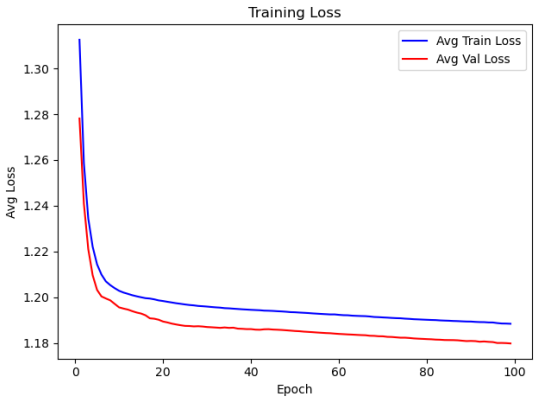

Figure 4.26: Average training loss per epoch for all epochs of training to predict velocity except the first.

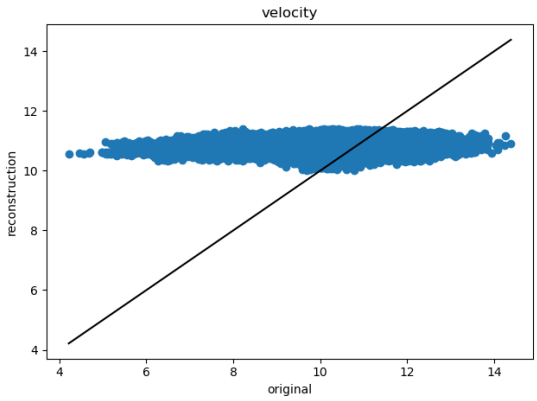

Figure 4.27: Scatter plot of the predicted velocity against the original.

Training μ_{Loss} (MSE)	Validation μ_{Loss} (MSE)	μ_{Error}	σ_{Error}
1.19	1.18	0.25	1.07

Discussion

It appears that the loss functions cannot decrease further substantially in these experiments with the current network set up (there is even overfitting in the case of torque). Training observes sensible learning over 100 epochs (15 for torque), although the prediction for velocity does not appear to have the correct shape for the scatter plot, just like in section 4.3 probably for the same reasons. Hence, the same comments apply here. Further research can be done with data from a laminar flow or a wider spread in flow velocity to explore the proposed ideas.

When comparing the results with those from section 4.4 it can be seen that the mean squared errors for thrust and torque increased by 0.2 and for velocity it increased by 1.0. This is to be expected since the experiments in section 4.4 use data from the same parameters for reconstruction while the experiments in this section try to predict a given parameter from the other two. When comparing the results with those from section 4.3 it can be seen that the mean squared errors increased by 0.1. This is probably due to the fact that the encoders were trained on different data sets than the ones used for the prediction.

Chapter 5

Conclusion

5.1 Overview

In conclusion, most of the aims of this project have been achieved. A variational autoencoder has been created that can successfully construct meaningful representations of marine turbine parameters and reconstructions based on those representations. A decoder has been created that can successfully predict thrust and torque from the representations of the other two parameters. The decoder's current setup cannot successfully predict velocity from the representations of thrust and torque, however. As discussed in Chapter 4, this could be due to the turbulence in the flow velocity and/or the difference in the standard deviations of the parameters even after scaling the data. Turbulence would make velocity hard to predict using only 2 parameters due to the chaotic structure. The smaller standard deviation in flow velocity compared to thrust and torque even after scaling may explain the bad performance of the scatter plot produced from the velocity predictions. A smaller standard deviation would mean that more of the data is concentrated closer to the mean (compared to the better performing parameters), where the scatter plot is most closely aligned with the desired shape along the $y = x$ line.

5.2 Future Work

Future work could attempt to improve the velocity prediction. Further experiments could be done using data from laminar flow and/or data with a greater spread in flow velocity. Apart from that, the constructed variational autoencoder could be used to study decoded random latent space points since the space has been regularised due to the use of distributions in the training process. This may possibly provide a way of predicting the future evolution of the data.

5.3 Lessons Learned

This project has given me the opportunity to learn about and implement deep neural networks. As my background is in Mathematics, Physics and Software Development, I hadn't worked on deep

learning before. I found this experience very different from my previous projects. Apart from the mathematics and statistics in the theory, and the overall design architecture, I found creating a neural network to involve a lot of guesswork and trial and error. After deciding on the type of network I wanted to use, I had to come up with a lot of variations to try out and see which produced the best results. There was no obvious way to ascertain the best option prior to testing, which was something I had very rarely dealt with, coming from a theory heavy background. Even while developing software, my previous experience with testing mostly involved trying to discover bugs in the translation of the design into implemented software, not trying to finish the design itself. This project provided me with the opportunity to learn how to tackle that element of uncertainty and how to systematically do tests, which work to produce an optimal design. Another important lesson I learned was the fact that regardless of the previous successes of my approach in the end I couldn't produce all of the results I had set out to. And there was no way to know whether I could without doing a sufficient number of trials.

Bibliography

[1] Bai, C., Zhou, C.
Pressure Predictions of Turbine Blades with Deep Learning, Turbomachinery Laboratory, College of Engineering, Peking University Beijing, 100871, China. Accessed 20/09/2020.
https://arxiv.org/ftp/arxiv/papers/1806/1806.06940.pdf

[2] Chang, J.
HYDRODYNAMIC MODELING AND FEASIBILITY STUDY OF HARNESSING TIDAL POWER AT THE BAY OF FUNDY , May 2008. Accessed 20/09/2020.
https://web.archive.org/web/20121122141719/...
...http://digitallibrary.usc.edu/assetserver/controller/...
...item/etd-Chang-20080312.pdf

[3] Chintala, S.
DEEP LEARNING WITH PYTORCH: A 60 MINUTE BLITZ, PyTorch, 2017. Accessed 20/09/2020.
https://pytorch.org/tutorials/beginner/deep_learning_60min_...
...blitz.html

[4] Fridman, L.
MIT Deep Learning Basics: Introduction and Overview with TensorFlow, Medium, February 4, 2019. Accessed 20/09/2020.
https://medium.com/tensorflow/mit-deep-learning-basics-...
...introduction-and-overview-with-tensorflow-355bcd26baf0

[5] Lenail, A. *AlexNet style NN-architecture schematics. Accessed 20/09/2020.*
http://alexlenail.me/NN-SVG/AlexNet.html

[6] Martinez, R., Allmark, M., Ordonez-Sanchez, S., Lloyd, C., O'Doherty, T., Germain, G., Gaurier, B., Johnstone, C.
Rotor load prediction using downstream flow measurements, Unpublished Preprint, August 6, 2020.

[7] Ni, C., Ma, X., Wang, J.
Integrated deep learning model for predicting electrical power generation from wave energy converter, National Ocean Technology Center, Tianjin 300112, China and Engineering Department, Lancaster University, Lancaster LA1 4YW, UK. Accessed 20/09/2020.
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=...
...8895237

[8] Nielsen, M.
Neural Networks, December 2019. Accessed 20/09/2020.
http://neuralnetworksanddeeplearning.com/index.html

[9] Pitsillos, N.
MNIST Variational Autoencoder & t-SNE Visualisation, GitHub, May 10, 2020. Accessed 20/09/2020.
https://npitsillos.github.io/posts/2020/05/mnistvae/

[10] Pitsillos, N.
main.py, GitHub. Accessed 20/09/2020.
https://github.com/npitsillos/mnist-vae/blob/master/main.py

[11] Pitsillos, N.
trainer.py, GitHub. Accessed 20/09/2020.
https://github.com/npitsillos/productivity_efficiency/blob/...
...master/torch_trainer/trainer.py

[12] Rocca, J.
Understanding Variational Autoencoders (VAEs), Towards Data Science, Medium, September 24, 2019. Accessed 20/09/2020.
https://towardsdatascience.com/understanding-variational-...
...autoencoders-vaes-f70510919f73

[13] Saha, S.
A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way, Towards Data Science, Medium, December 15, 2018. Accessed 20/09/2020.
https://towardsdatascience.com/a-comprehensive-guide-to-...
...convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[14] Sanderson, G.
Neural Networks, 3Blue1Brown, YouTube, August 1, 2018. Accessed 20/09/2020.
https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_...
...67000Dx_ZCJB-3pi

[15] Wandel, N., Weinmann, M., Klein, R.
Unsupervised Deep Learning of Incompressible Fluid Dynamics, June 15, 2020. Accessed 20/09/2020.
https://arxiv.org/pdf/2006.08762.pdf

[16] Zhang, A., Lipton, Z. C., Li, M., Smola, A. J.
Dive into Deep Learning, 2020. Accessed 20/09/2020.
https://d2l.ai

[17] *diagrams.net Accessed 20/09/2020.*
https://app.diagrams.net/