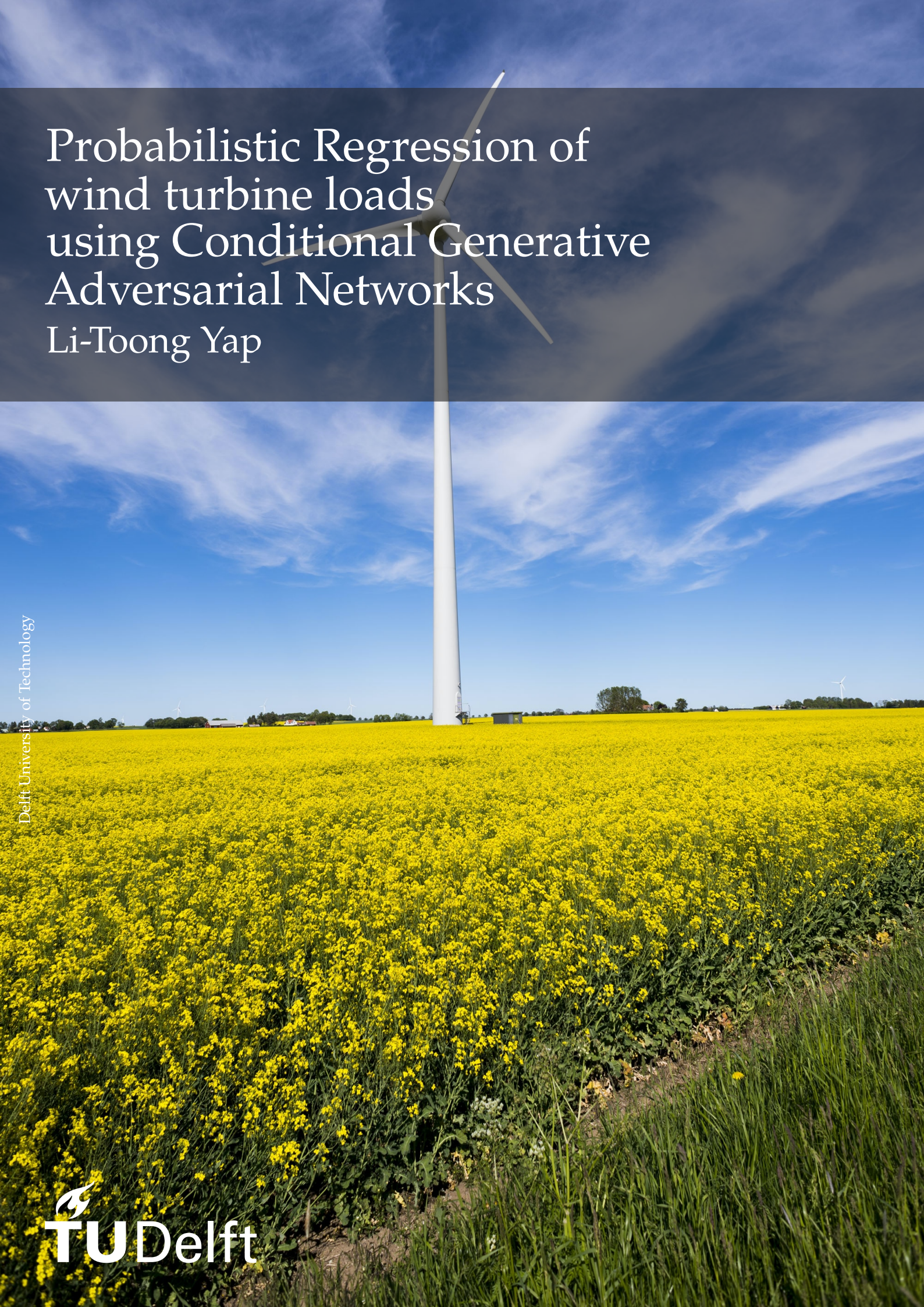
Probabilistic Regression of wind turbine loads using Conditional Generative Adversarial Networks

Li-Toong Yap



Delft University of Technology

TU Delft

Probabilistic Regression of wind turbine loads using Conditional Generative Adversarial Networks

by

Li-Toong Yap

Student number: 5226325

Dr R.P. Dwight,	Supervisor, chair
Deepali Singh,	Co-Supervisor
Dr Eliz-Mari Lourens	Examiner
Dr Wei Yu	Examiner
Thesis Duration:	August, 2022 - September, 2023
Faculty:	Faculty of Aerospace Engineering, Delft

TU Delft

Abstract

Site analysis to determine the loads experienced by wind turbines based on site-specific environmental conditions is typically done using either coupled aero-servo-elastic simulations for onshore wind turbines or coupled aero-servo-hydroelastic simulations in the case of offshore wind turbines. These simulations become computationally expensive when multiple load cases are needed to be taken into account, together with the numerous possible combinations of turbulence inflow patterns that result in the same mean inflow conditions. Probabilistic surrogate models offer a cheap alternative to these expensive simulations for predicting load statistics. This thesis explores a type of neural network called Conditional Generative Adversarial Networks (CGANs) as a potential candidate for such a surrogate model. Originally developed for image generation, CGANs have seen success in other applications. However, most applications of GANs to date are high-dimensional, with relatively low research focused on low-dimensional problems such as wind turbine load statistics. Multiple experiments are conducted using various multimodal and heteroscedastic datasets to assess its ability to model such characteristics accurately. The conditional log-likelihood and Wasserstein-1 distances were used as metrics. The results show that CGANs can indeed model such low-dimensional datasets. Finally, the CGANs are trained on data from simulations of onshore and offshore wind turbines in OpenFAST and compared with predictions from Mixture Density Networks (MDNs). The results from CGANs are comparable to MDNs, showing its potential as another alternative surrogate method, although more research needs to be performed.

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
ANN	Artificial Neural Networks
CGAN	Conditional Generative Adverserial Network
CVAE	Conditional Variational Autoencoders
GAN	Generative Adverserial Network
GPR	Gaussian Process Regression
IPM	Integral Probability Metrics
JSD	Jensen-Shannon Divergence
KL	Kullback-Leiber
MLE	Maximum Likelihood Estimation
NREL	National Renewable Energy Laboratory
RKL	Reverse Kullback-Leiber
PCE	Polynomial Chaos Expansion
pdf	Probability Density Function
VAE	Variational Autoencoders

Symbols

Symbol	Definition
D	Discriminator/Critc
D^*	Optimal Discriminator/Critc
G	Generator
I	Identity matrix
$\mathcal{L}$	Loss function
l_h	Number of hidden layers in main network
$l_{h,I}$	Number of hidden layers in initial network
m	input dimensionality/Wöhler coefficient
p	output dimensionality
p_d	pdf of the real distribution
p_g	pdf of the generated distribution
p_z	pdf of the latent distribution
$\mathbf{u}^{(n)}$	output of neural network layer n
u_h	Number of units per hidden layer in main network
$u_{h,I}$	Number of units per hidden layer in initial network
$\mathbf{w}$	vector of weights in linear regression
$\mathbf{W^{(i)}}$	Matrix of weights between layer i and $i+1$ of a neural network
$\mathbf{x}$	Input vector
$\mathbf{y}$	Output vector
$\mathbf{z}$	Latent vector
X, Y	Random Variables
ϵ	Noise term
θ	Neural network parameters

Symbol	Definition
μ	Mean
π_k	Weight of mixture k
σ_n^2	Noise variance
$\sigma(x)$	Sigmoid function

1

Introduction

1.1. Background

Site analysis is performed before the installation of wind turbines to determine the loads experienced by the wind turbine based on site-specific environmental conditions. This is typically done using coupled aero-servo-elastic simulations or aero-servo-hydroelastic simulations in the case of offshore wind turbines. However, these coupled simulations are computationally expensive, especially in the case of offshore wind turbine siting and a large number of load cases required.

To expedite this process, an alternative is to use surrogate models, which are low-cost computational models that approximate the full-order models by mapping the inputs to the outputs of the original model. The low cost of running surrogate models makes them ideal for preliminary analysis, where some accuracy can be traded for computational time. One way to design a surrogate model for wind turbine site-specific loads involves a deterministic approach: given a training set $\left\{(\mathbf{x}^i, y^i)\right\}_{i=1}^{N_{\text{train}}}$, $N_{\text{train}} \in \mathbb{R}$, the deterministic surrogate model maps a set of m-dimensional input features $\mathbf{x} \in \mathbb{R}^m$ to a set of 1-dimensional outputs $y \in \mathbb{R}$.

However, real-world datasets have inherent uncertainty that is either unknown or inexpressible and cannot be be reduced with sufficient training data. This is also known as aleatoric uncertainty, which results in irreducible error in surrogate modelling and violates the standard assumption of a deterministic relationship between a set of inputs and outputs.

A deterministic approach would only be able to infer the most likely load responses and assume a Gaussian error. On the other hand, in a probabilistic approach, the inputs and outputs are modelled as random variables $X \in \mathbb{R}^m$ and $Y \in \mathbb{R}$ respectively, allowing the uncertainty of the input to be propagated through the surrogate model and appear as uncertainty in the output, allowing the complete probability density function (pdf) of the response to be modelled.

In the context of surrogates for wind turbine loads, a set of inflow conditions averaged over 10 minutes could be obtained from any variations in the time domain, leading to multiple possible load histories to which the averaged inflow conditions could be mapped. In wind turbine analysis tools such as OpenFAST, this stochasticity is simulated via random number generators initialised by random seeds, which outputs a frozen turbulence field for the analysis tool.

These load histories can provide valuable insights about the mean response and any variations, potentially saving costs on construction and maintenance since safety factors factored into wind turbine design, such as structural strength, can be reduced. This would reduce material costs and increase the operational lifespan of wind turbines.

With that in mind, this thesis aims to present a methodology for probabilistic load prediction using conditional generative adversarial networks (CGANs), a type of deep neural network. Rather than explicitly model the probability density function of the output, CGANs produce samples from a distribution that is defined implicitly by the choice of parameters of the neural network, thereby avoiding the need to make assumptions about the probability density of the output. In the following sections, there will be a literature review on existing methods used for modelling wind turbine loads and the usage of CGANs for regression problems, followed by the research questions to be answered in this thesis and the theory behind CGANs.

1.2. Motivation

In this thesis, the usage of CGANs for probabilistic regression modelling is investigated to model arbitrary distributions which can be non-parametric, with an application towards wind turbine loads. Basic regression involves modelling a relationship between an input variable $\mathbf{x}$ and an output variable y. A common regression model used is that of a deterministic function $h(\mathbf{x})$ with additive Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ such that

$$y = h(\mathbf{x}) + \epsilon \tag{1.1}$$

An example of an application of this basic regression model is linear regression, where $h(x) = \mathbf{w}^T\mathbf{x}$, $\mathbf{w}$ being the vector of weights. Linear regression allows for the prediction of a real-valued output given an input. However, this output is deterministic. It would be more interesting to model $p(y|\mathbf{x})$ to gain insights into its statistical properties for more informed decision-making, which is the goal of probabilistic regression.

One popular probabilistic regression model is Gaussian Process Regression (GPR). GPRs can be extended to multiple dimensions, but only the 1-D case is considered here. In the 1-D case, GPRs follow the basic regression model defined in Equation 1.1, except that the function $h(\mathbf{x})$ is now modelled as a random variable. A Gaussian Process (GP) is any collection of random variables such that any finite subset of the variables is jointly normally distributed. A GP then defines a distribution over the function space $h(\mathbf{x})$ belongs to, such that

$$\begin{aligned} h(\mathbf{x}) &\sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')) \\ m(\mathbf{x}) &= \mathbb{E}[h(\mathbf{x})] \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(h(\mathbf{x}) - m(\mathbf{x}))(h(\mathbf{x}') - m(\mathbf{x}'))] \end{aligned} \tag{1.2}$$

where $m(\mathbf{x})$ and $k(\mathbf{x}, \mathbf{x}')$ are the mean function and covariance function respectively.

The effectiveness of GPR relies on the choice of both functions, which in turn requires some knowledge of the nature of the dataset that is often not accessible or accurate. For instance, an infinitely-differentiable and smooth covariance function would imply the same as the underlying function. The distributions of y and $h(\mathbf{x}')$ are also modelled as Gaussian, a strong assumption for real datasets. Furthermore, the additive noise is assumed to be homoscedastic, which further limits its modelling abilities. To remedy the shortcomings of GPR, an extension called chained GPRs is proposed by Saul et al. [1], where a second GP is used for the now input-dependent noise variance, allowing for heteroscedastic regression according to the model

$$\begin{aligned} y &= h(\mathbf{x}) + \epsilon(\mathbf{x}) \\ \epsilon &\sim \mathcal{N}(0, \sigma_n^2(\mathbf{x})) \end{aligned} \tag{1.3}$$

The drawback with chained GPRs is that, unlike the original GPR model, the chained GPR does not have a closed-form solution, and methods that approximate the posterior distribution have to be employed [1].

Another probabilistic regression model is Mixture Density Networks (MDN). MDNs are Gaussian mixture models combined with neural networks [2], where the target conditional distribution is modelled as a mixture of Gaussians, where the mixture components are a function of $\mathbf{x}$:

$$p(y|\mathbf{x}) = \sum_{k=1}^{K} \pi_k(\mathbf{x}) \mathcal{N}(y|\mu_k(\mathbf{x}), \sigma_k^2(\mathbf{x})) \tag{1.4}$$

The mixture coefficients $\{\pi_k(\mathbf{x})\}_{k=1}^K$, means $\{\mu_k(\mathbf{x})\}_{k=1}^K$ and variances $\left\{\sigma_k^2(\mathbf{x})\right\}_{k=1}^K$ are the outputs of the neural network that then define the model distribution. Besides modelling heteroscedasticity, MDNs can produce multimodal distributions, which is an advantage over GPRs since real datasets are generally multimodal, making using a unimodal distribution like a Gaussian an inaccurate option.

However, the use of Gaussian mixtures still limits the type of possible model distributions since the predicted distribution will be expressed as a sum of Gaussians. CGANs offer a possible alternative for approximating a more general class of distributions. Previous work on CGANs focused on high-dimensional y, such as image generation ($y \in \mathbb{R}^{784}$ for a 28×28 image), and relatively little work has been done on the usage of CGANs for low-dimensional y (1 in the context of this thesis).

1.3. Literature review

1.3.1. Applications in wind energy

Of the various deterministic data-driven models used, artificial neural networks (ANNs) have been used in Schröder et al., Dimitrov and Shaler et al. [3–5]. Shaler et al. [5] in particular investigated the performance of inverse distance weighting, radial basis functions, ANNs, Kriging with a partial least squares dimension reduction and regularised minimal-energy tensor-product b-splines for load surrogates of wind turbines in an array, showing that inverse distance weighting and ANNs gave the best average numerical R^2 performance across all output statistics. However, these methods do not model variability due to stochastic inflow conditions.

On the other hand, several approaches have been taken to tackle the probabilistic nature of wind turbine loads. In Zhu et al. [6], a joint polynomial chaos expansion (PCE)-generalised λ-distribution algorithm is introduced to model the pdf of the response using a λ-distribution with calibrated parameters. λ-distributions are used to model output response, with its parameters' dependence on the input being modelled by PCEs. This model is able to predict the pdf for a simple test case of a fixed-bottom wind turbine.

In addition, GPR has been evaluated in Teixeira et al., Dimitrov et al., Avendaño-Valencia et al., Gasparis et al. and Okpokparoro et al. [7–11]. GPR has been evaluated against other methods such as PCE, importance sampling, nearest-neighbour interpolation, and quadratic response surface in Dimitrov et al. [8], concluding that the GPR gave more accurate results at a higher computational cost. A similar study is performed in Gasparis et al. [10] but against linear regression and artificial neural nets instead, showing that the GPR outperforms both models in terms of the normalised root mean square error based on a given small amount of training samples. Slot et al. [12] performs a detailed study on both GPR and PCE, noting that GPR has a higher accuracy per invested simulation than PCE. However, they also note that the accuracy of GPR relies on the number of seeds used per training sample. In particular, they proposed that at least 4 seeds per training sample be used to obtain a sufficiently high accuracy.

Despite the positive results of standard GPR, the heteroscedasticity of the inherent noise is not considered in this framework. This is the focus in Singh et al. [13], which investigates the usage of heteroscedastic-GPR (H-GPR), showing that H-GPR shows an overall considerable improvement over GPR in predicting the conditional distribution of the average and maximum loads, despite challenges faced in the prediction of the standard deviation in loads. However, this method has only been applied to small datasets but is not scalable to higher dimensions. In addition, GPR also assumes that the conditional distribution has a Gaussian shape, which may not necessarily be true.

On the other hand, neural networks based on a probabilistic framework have yet to be explored as much for load emulation purposes. Deep neural networks have the advantage of scaling well with data dimensionality and the ability to model non-linear functions by defining the parameters of the neural network. An approach known as generative modelling uses this to implicitly model an unknown probability distribution by generating samples that follow an arbitrary distribution induced by the parameters of the neural network. By changing the parameters, the distribution of the generated samples can be adjusted to match that of the real distribution closely. This is useful in applications where it is more beneficial to generate samples than to know the numerical value of the density that defines the distribution [14].

Variable autoencoders (VAE) and generative adversarial networks (GAN) are examples of generative models, and such generative models have been applied to probabilistic predictions in the wind energy field successfully, among other applications. In Mylonas et al. [15], a variational autoencoder (VAE), conditioned on wind-field measurement techniques, is trained to model the probability distribution of the accumulated fatigue on the root cross-section of a turbine blade. The results from Mylonas et al. [15] demonstrate the conditional VAE's (CVAE) ability to accurately capture the shape of the fatigue distribution over the cross-section and in cases where training data is scarce. Besides wind turbine loads, VAEs have also been applied to other engineering problems. In Wang et al. [16], VAEs are used as feature extractors for downstream classification tasks in the context of planetary gear fault classification. Yoon et al. [17] applied VAEs for feature learning applied to semi-supervised remaining useful life prediction.

1.3.2. CGANs

Besides VAEs, generative adversarial networks (GAN) [18] are also an example of a deep generative model. GANs consist of a discriminator and a generator, where the generator's role is to create samples that the discriminator will evaluate whether the sample comes from a real dataset or not. Ultimately, the goal is to create a generator that produces samples from a distribution that closely matches a target distribution. The discriminator and generator are complex functions that typically do not have a closed form; hence they are usually modelled via neural networks. The aim is to optimise the neural networks to minimise a given objective function, which can be done using standard neural network training methods. Using neural networks also makes GANs scalable to higher dimensions due to the parallelalisable nature of stochastic gradient descent in neural network training.

In the original formulation, there is no control over the type of samples being generated. This is not very useful, especially in the case of wind turbine siting, since the loads need to be conditioned on specific inflow conditions. It would be more desirable for the generator to generate samples from the distribution $p_g(Y|\mathbf{x}^i)$ instead of $p_g(Y)$, where p_g is the generator's induced distribution. This is the motivation behind conditional GANs (CGANs) [19]. In practice, CGANs only require a slight modification to the standard GAN formulation to generate samples based on a given input.

As GANs aim to generate samples from a distribution that closely matches a target distribution, utilising a difference measure to quantify the difference between the two would be natural. One of interest to GANs is called f-divergences [20], a family of difference measures between probability distributions. In the standard GANs formulation, it can be shown that training a generator against an optimal discriminator amounts to minimising the Jensen-Shannon divergence [18], an f-divergence, between the generator distribution and the actual distribution. It follows that by modifying the objective function of the standard GANs as given in [18], it is possible to design GANs that minimise different f-divergences, which can lead to different results. [20] proposes such a formulation called f-GAN that allows for the minimisation of any f-divergence. This formulation is useful when considering the characteristics of the f-divergences when minimised. For instance, the Kullback-Leiber (KL) divergence and the reverse Kullback-Leiber (RKL) are f-divergences known to have mean-seeking and mode-seeking properties respectively [21]. [20] demonstrates that using GANs with different f-divergence objectives gives results that are similar to directly minimising the respective divergences, noting that the f-GAN with the KL and RKL divergences as their objective produce results that demonstrate their mean-seeking and mode-seeking behaviour.

Besides f-divergences, other difference measures can be used for GAN training. One such family is the Integral Probability Metrics (IPMs) [22]. Within this family of difference measures, one that is of interest is the Wasserstein-1 distance, which is used to define an alternative GAN model that minimises the Wasserstein-1 distance known as Wasserstein GANs (WGANs) [14]. Wasserstein GANs have gained interest in the following years as it has been claimed to have improved training performance over the original GAN. One disadvantage of using standard GANs is that the training of the generator can be challenging due to vanishing gradients as the discriminator becomes better, thereby leading to the phenomenon known as mode collapse. On the other hand, WGANs can provide more useful gradient information to the generator during training, making training more stable and preventing mode collapse [14].

GANs have been mainly utilised in applications where the dimensionality of the output, p, is much larger than that of the input with much success [23]. One example is image generation that mimics a given collection of images [18], such as the MNIST dataset [24]. On the other hand, GANs usage in applications where $m > p$ is relatively uncommon. Nonetheless, there has been research into applications for such problems. As a basic overview, [23, 25, 26] investigate the usage of GANs for regression problems with both synthetic and real-world datasets where $m > p$. Both [23] and [25] note that GANs excel at modelling complex noise distributions compared to methods like Gaussian Processes (GPs) and encompass various regression models using a single formulation and implementation, showing its viability as a probabilistic regression tool. However, it is also noted that to fully judge the usefulness of GANs for probabilistic regression, more applications to datasets, particularly those from the real world, are required. It is also noted that GANs also come with disadvantages: they require a lot of parameter tuning. and they are also dependent on the amount of training data available to capture any complex noise distributions. GANs also do not have an explicit expression for the generated distribution, which makes accurate comparisons of the model distributions difficult. Nevertheless, GANs have been successfully applied to real-world datasets such as [27], which uses GANs for climate

predictions as well as [28] for data-driven electric vehicle charging profile generation. [29] looks at scenario generation for wind power using WGANs, while [30] looks at using WGANs for wind power scenario generation for multiple wind farms. Both applications have been shown to outperform existing methods.

There has been success in applying GANs to various engineering fields, especially in the renewable energy industry. However, to our knowledge, GANs of any form have yet to be applied to load estimation for single wind turbines. In particular, CGANs allow us to estimate the probability distribution of loads given specific inputs, such as environmental conditions. While GANs are usually applied to problems with very high dimensions, such as image generation, load emulation is a relatively low-dimensional problem, which CGANs may or may not be successful at tackling. Given the promising results of GANs and its variants when applied to other engineering fields, it would be worth investigating the usage of CGANs as a probabilistic regression tool for predicting wind turbine load statistics.

1.4. Research Objective and Questions

The main research question to be answered in this thesis is:

> **Should GANs be used as a probabilistic surrogate model for the load emulation of wind turbines?**

To be able to answer this question, a set of sub-questions are crafted to guide the process.

1. **How well do CGANs work for low dimensional problems in general, especially where the dimensionality of the input is much larger than the output?**
2. **How does the choice of neural network architectures and training objectives affect the training process of CGANs to approximate distributions?**
3. **How well do CGANs compare to alternative probabilistic models like Mixture Density Networks for load emulation?**
4. **What are the challenges faced when using a CGAN model for load emulation?**

1.4.1. Research Objective

The research objective for this thesis is to investigate the usage of CGANs for probabilistic regression when applied to predictions of 10-minute load statistics under given environmental conditions. This is to be achieved by performing a few sub-goals.

The performance of CGANs for low-dimensional problems first has to be investigated since GANs are typically used for high-dimensional problems where the input dimensionality is larger than the output dimensionality. There is no guarantee that the success of GANs for high-dimensional problems will translate to problems of lower dimensions. CGANs will be trained on synthetic datasets to evaluate its performance before moving to real-world datasets.

Besides investigating the performance of CGANs for low-dimensional problems, research into various types of CGANs will be done. Depending on the formulation used for the cost function, different types of CGANs can have different advantages and disadvantages over one another. Investigating different types of CGANs will help determine which is most suited for the actual problem of load prediction.

Finally, the CGAN models will be applied to real-world datasets. The results from the various CGAN models will be evaluated against the probabilistic models already in literature, to determine any advantages or disadvantages over existing methods. The work in this thesis is based on the code of Oskarsson [25][1], together with some of the datasets, namely the heteroscedastic and wmix dataset. Real-world datasets consist of OpenFAST simulations on NREL's 10-MW reference wind turbine for both onshore and offshore wind conditions [31].

[1] Available at `https://github.com/joeloskarsson/CGAN-regression`

2

Theory

This section will cover the basics of neural networks, followed by an overview of generative adversarial networks and its variants. Ways of evaluating the performance of generative adversarial networks will also be discussed.

2.1. Neural Networks

A diagram of a neural network is shown in Figure 2.1. It consists of a set of layers, each containing a number of computational units called artificial neurons. The first layer of a neural network is called the input layer, the last layer is called the output layer, and the layers in-between are called hidden layers. Each unit in a layer has weighted connections to all units in the following layer, allowing the inputs from previous layers to change as they pass through the network.

The output of a layer i is a vector $\mathbf{u}^{(i)} \in \mathbb{R}^{m_i}$, where m_i is the number of units in layer i. Then, the input layer $\mathbf{u}^{(0)}$ corresponds to the input $\mathbf{x}$, and the output layer $\mathbf{u}^n$ corresponds to the prediction $\hat{\mathbf{y}}$. Thus, the dimensionality of $\mathbf{x}$ and $\hat{\mathbf{y}}$ also corresponds to the number of units in the input and output layer i.e. $\mathbf{x} \in \mathbb{R}^{m_0}$ and $\hat{\mathbf{y}} \in \mathbb{R}^{m_n}$. The weights of the connections between two layers are stored as a matrix $\mathbf{W}^{(i)} \in \mathbb{R}^{m_{i+1} \times m_i}$, where $w^{(i)}_{j,k}$ is the weighted connection between unit k in layer i and unit j in layer $i+1$:

$$\mathbf{W}^{(i)} = \begin{bmatrix} w^{(i)}_{1,1} & w^{(i)}_{1,2} & \dots & w^{(i)}_{1,m_i} \\ w^{(i)}_{2,1} & w^{(i)}_{2,2} & \dots & w^{(i)}_{2,m_i} \\ \vdots & \vdots & & \vdots \\ w^{(i)}_{m_{i+1},1} & w^{(i)}_{m_{i+1},2} & \dots & w^{(i)}_{m_{i+1},m_i} \end{bmatrix} \tag{2.1}$$

In addition, there is a bias vector $\mathbf{b}^{(i)} \in \mathbb{R}^{m_i}$ associated with each $\mathbf{u}^{(i)}$ which acts as a constant offset. The computation of each $\mathbf{u}^i$ is then

$$\mathbf{u}^{(i)} = g(\mathbf{W}^{(i-1)}\mathbf{u}^{(i-1)} + \mathbf{b}^{(i-1)}) \tag{2.2}$$

where $g : \mathbb{R}^{m_i} \to \mathbb{R}^{m_i}$ is a non-linear function, referred to as the activation function. The values of $\mathbf{u}^{(i)}$ are passed on to layer $i+1$ for the computation of $\mathbf{u}^{(i+1)}$ and so on for subsequent layers until the output layer is reached. This basic model is also known as a feedforward model since the input flows through the function approximated by the hidden layers and produces an output without any feedback connections that pass the output back into the model [32].

2.1.1. Activation functions

Activation functions are non-linear functions applied entry-wise over the layer output $\mathbf{u}^{(i)}$ that allow the neural network to approximate complex functions which might not have a closed form. The most common activation function is the Rectified Linear Unit (ReLU) activation function, defined as

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{otherwise} \end{cases} \tag{2.3}$$

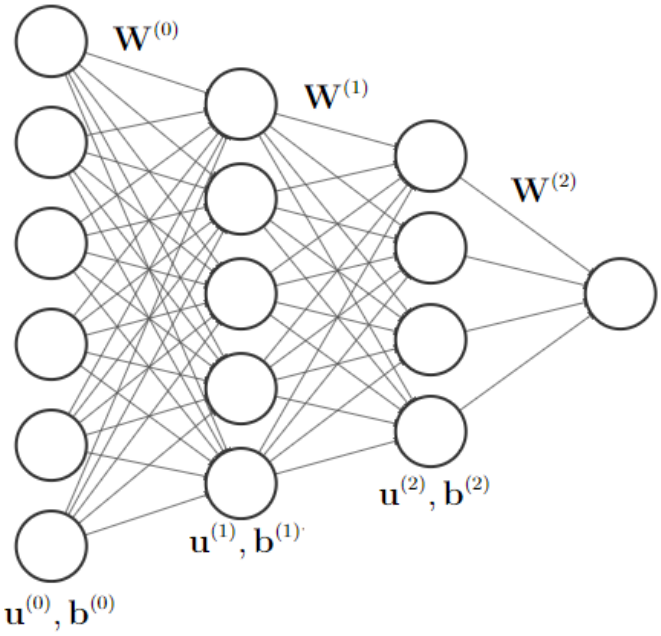


Figure 2.1: An neural network with 2 hidden layers

This function is non-differentiable at $x = 0$; however, this is not an issue for practical computational purposes since the chances of x being exactly equal to 0 is very small. In addition, a suitable derivative can be defined in the implementation [32].

ReLU units have large and consistent gradients for $x > 0$ but zero gradients otherwise. Zero gradients can be an issue when using gradient-based training methods, leading to potential problems such as slow learning or non-activation of units [33]. As a result, a variant of ReLU called LeakyReLU is proposed by [33], defined as

$$\text{LeakyReLU}(x) = \begin{cases} \alpha x & \text{if } x \leq 0 \\ x & \text{otherwise} \end{cases} \tag{2.4}$$

where α is a tunable parameter (this is set to 0.01 in the original implementation). The gradient for $x \leq 0$ is then the value of α.

Besides ReLU and its variants, another activation function is the exponential linear unit (ELU) activation [34]. This takes the form of

$$\text{ELU}(x) = \begin{cases} \alpha(\exp(x) - 1) & \text{if } x \leq 0 \\ x & \text{otherwise} \end{cases} \tag{2.5}$$

where $\alpha > 0$ is also a tunable parameter for the ELU activation. ELUs push the mean output values toward zero during training, speeding it up. ELUs have been shown in [34] to give better generalisations and faster training than networks with LeakyReLU or ReLU activations. Section 3.3.3 will discuss the activation functions used for this thesis.

2.1.2. Network Training

By optimising the trainable parameters in the neural network θ, which consists of all the weights and biases, it learns the underlying structure of a given problem and provides suitable predictions. This is done by minimising a loss function $\mathcal{L}(\theta)$ with respect to θ. The loss function measures how well a network performs on a given training set, given as [32]:

$$\begin{aligned} \mathcal{L}(\theta) &= \mathbb{E}_{(\mathbf{x},y)\sim\hat{p}_{data}}[L(f(\mathbf{x};\theta), y)] \\ &\approx \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \left(L(f(\mathbf{x^i};\theta), y^i) \right) \end{aligned} \tag{2.6}$$

where $\{(\mathbf{x}^i, y^i)\}_{i=1}^{N_{\text{train}}}$ are training pairs from the empirical dataset and $f(\mathbf{x}^i; \theta)$ is the model prediction given $\mathbf{x}^i$. L is a per-example loss function, such as the negative conditional log-likelihood $-\log p(y|\mathbf{x}; \theta)$ or squared error $[y - f(\mathbf{x}; \theta)]^2$. In particular, the gradients of the model weights w.r.t. the loss function $\nabla_\theta \mathcal{L}(\theta)$ are of interest since they determine how much θ should be adjusted to minimise the loss function.

In optimizing convex functions, reaching a global minimum is straightforward since any local minimum is guaranteed to be a global minimum. Even if there is a flat region at the bottom instead of a single global minimum, it suffices to find a solution at any point in that region[32]. On the other hand, the non-linearity of the neural networks due to the activation functions makes most loss functions non-convex [32]. Such functions will have multiple local minima, and finding a global minimum is usually impossible [25]. However, it suffices to find a suitable local optimum, so long as the local optimum has a low cost in comparison with the global optimum [32].The optimisation process is usually done via gradient descent [35], where the model parameters are updated using $\nabla_\theta \mathcal{L}(\theta)$ according to

$$\theta \leftarrow \theta - \gamma \nabla_\theta \mathcal{L}(\theta) \tag{2.7}$$

and γ is an adjustable learning rate hyperparameter for the optimiser. $\nabla_\theta \mathcal{L}(\theta)$ is calculated via the backpropagation algorithm [32], which can be done in a computationally efficient manner for neural networks by keeping track of the flow of information from the start to end of the neural network.

In practice, an extension of the gradient descent known as stochastic gradient descent is used. The motivation for stochastic gradient descent comes from the problem that large training sets are required for models to generalise well while also being computationally expensive to use. For the loss function as shown in Equation 2.6, computing the gradients for the gradient descent algorithm would require

$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \nabla_\theta \left(L(f(\mathbf{x^i}; \theta), y^i) \right) \tag{2.8}$$

which becomes computationally expensive (increases by $O(N_{\text{train}})$) as the number of training samples increase. The idea behind stochastic gradient descent is that the gradient of the loss function is an expectation as shown in Equation 2.8, and it can be approximated by considering a small batch of samples drawn uniformly from the original training set. In this way, a model can be trained on a large dataset while using only a small subset of the dataset for updates. In practice, a training dataset is split into subsets known as batches. Then, the training and stochastic gradient descent is performed on each batch. After every epoch, defined as a single pass through all the batches of the dataset, the dataset is reshuffled and re-split into batches to prevent overfitting, and the training process repeats.

2.1.3. Optimisers

Optimisation of θ via stochastic gradient descent has proven effective for training neural networks; however, it requires choosing an optimiser and an appropriate learning rate γ from Equation 2.7. Too high a learning rate and the loss function may not decrease on every iteration or not converge. On the other hand, if the learning rate is too low, the loss function will be slow to converge. To make the selection of learning rate easier, two approaches have been developed and incorporated into modern optimisers: momentum and adaptive learning rates.

Momentum is a mechanism inspired by the corresponding physics concept. In its basic form, the parameters are treated as a unit mass, and the training process involves moving the parameters at some velocity $\mathbf{v}$ (and momentum, since momentum is mass times velocity) through the parameter space [32]. The velocity is set to an exponentially decaying average of the negative gradient [32], which is used in the update of the model parameters:

$$\begin{aligned} \mathbf{v} &\leftarrow \mu \mathbf{v} - \gamma \nabla_\theta \mathcal{L}(\theta) \\ \theta &\leftarrow \theta + \mathbf{v} \end{aligned} \tag{2.9}$$

where $\mu \in [0, 1)$ is a hyperparameter that controls the decay rate of previous gradients' contributions. This term is analogous to friction since it controls the velocity and prevents it from "overshooting".

In the momentum mechanism, γ is fixed for all parameters. Adaptive learning rates let γ adapt throughout the learning process and allow individual learning rates for each model parameter. Modern optimisers make use of adaptive learning rates along with momentum, of which the most notable are AdaGrad [36], RMSProp [37], and Adam [38].

The AdaGrad algorithm adapts the learning rate of each model parameter based on the history of the gradients of said parameter. It does so by scaling the learning rate γ inversely proportional to the sum of the squared-gradients of previous training steps [32] i.e. $\frac{\lambda}{r}$, $r = \sqrt{g_1^2 + g_2^2 + ... + g_{S-1}^2}$ at training step S, where g_S is the gradient of the model parameter that step. The idea is that parameters with the largest partial derivatives of the loss (receive big updates) will have a corresponding rapid decrease in their effective learning rate and vice-versa.

The learning rate in AdaGrad always decreases over time, making it desirable for convergence in theory. However, in practice, it can cause problems for neural network training since the normalisation factor is continuously increasing, which may excessively decrease the learning rate before a local optimum is reached [32]. The RMSProp algorithm is a modified version of AdaGrad by replacing the gradient accumulation with an exponentially weighted moving average [32], such that the learning rate depends more on the most recent gradients. RMSProp and AdaGrad do not have the momentum mechanism in the original formulations, although they can be easily incorporated [32]. The Adam optimiser builds on the adaptive learning rate from RMSProp, combining it with momentum and with various modifications [32].

The optimisers' performance largely depends on the problem at hand and the network used. The choice of optimiser for this thesis will be discussed in Section 3.3.3.

2.1.4. Regularisation

When training machine learning models, it can be that the model only fits to the training data. While this would give a low loss value based on the training set, it would not perform very well on unseen data since it has yet to discover any underlying structure of the dataset. This is known as model overfitting, and it can occur for various reasons, such as small dataset sizes, large model complexities, or noisy datasets.

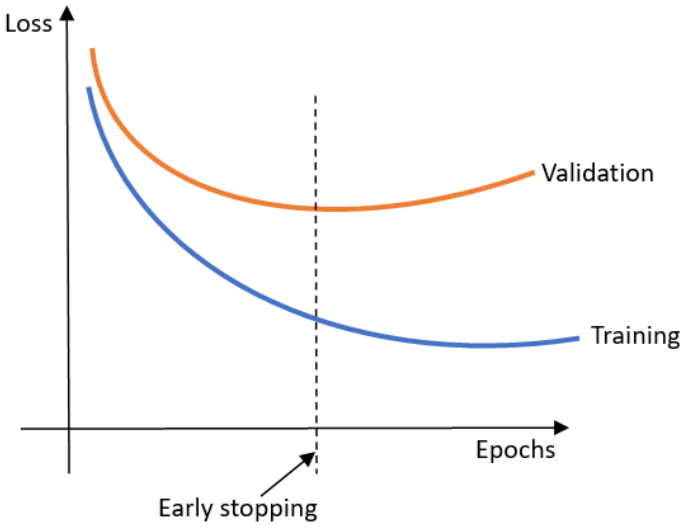


Figure 2.2: Early stopping

To prevent overfitting, regularisation techniques have to be implemented to limit the representational capacity of the model. The regularisation method implemented in this thesis is early stopping [32], illustrated in Figure 2.2. Neural networks are trained iteratively, causing the network to overfit slowly. Thus, overfitting can be avoided if the training is stopped early enough. To determine a good stopping point, a metric is required to gauge the ability of the model to generalise to unseen data over the training process. This is done by defining a separate set of data called the validation data to calculate the validation error. The validation dataset comes from the original training set but is not used for training the model. As the training progresses, the validation error decreases until the validation error starts to increase as the model begins to overfit. A good model can be obtained by stopping the training before this point. In practice, early stopping can be implemented by training a model for a fixed number of

epochs and saving the model from the epoch with the lowest validation error [32].

2.2. Generative Adversarial Networks

2.2.1. Background

Probabilistic machine learning models can be split into prescribed and implicit models [39]. Prescribed probabilistic models define an explicit likelihood function for the distribution of an observed random variable Y. If the distribution is defined by a pdf $p(Y;\theta)$, then the explicit likelihood function is

$$L(Y) = \prod_{i=1}^{n} p(y_i;\theta) \tag{2.10}$$

where y_i is a sample drawn from Y.

On the other hand, implicit models like GANs do not specify any likelihood function. Instead, they use a latent variable $\mathbf{z} \sim p_z(\mathbf{z})$ that is transformed into samples via a function $G(\mathbf{z};\theta) : \mathbb{R}^m \to \mathbb{R}^p$ where $m, p \in \mathbb{R}$. The parameters that define G implicitly define a distribution p_g that aims to approximate the actual distribution. Learning in implicit models is more complicated due to lacking a likelihood function to serve as a basis for an optimisation problem. At the same time, implicit models have more flexibility in the type of distributions learned since p_g does not require a closed form. This makes them of interest for problems where the true data distribution is complex or even intractable, and access to samples from the distribution is more important than knowing the true distribution [14]. Implicit generative models have been on the rise in the past few years, the most notable being ChatGPT [1] and Stable Diffusion [40].

A widely-used approach for learning in prescribed models is maximum likelihood estimation (MLE) [35]. MLE aims to find a set of model parameters $\hat{\theta}$ that maximises the likelihood function. Formally, the problem can be written as

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \prod_{i=1}^{n} p(y_i;\theta) \tag{2.11}$$

This approach does not work with implicit learning models since the likelihood function itself is unknown to begin with. Thus, alternative learning approaches need to be used. Rather than defining a likelihood function, one could instead draw samples from the model for evaluation against samples of the actual distribution under a suitable metric, typically a probability distance measure. The distances between the two distributions can then be used to guide the training process.

The idea of implicit generative models can be extended to conditional distributions, where the generator function G now takes an extra conditioning variable $\mathbf{x}$ and $p_g(Y|\mathbf{x})$ is now modelled instead. This creates extra complications in the training since the conditional distributions over the space of conditional inputs now have to be modelled, rather than just the marginals. In the following sections, the training of CGANs, together with their challenges, will be discussed.

2.2.2. Standard GANs

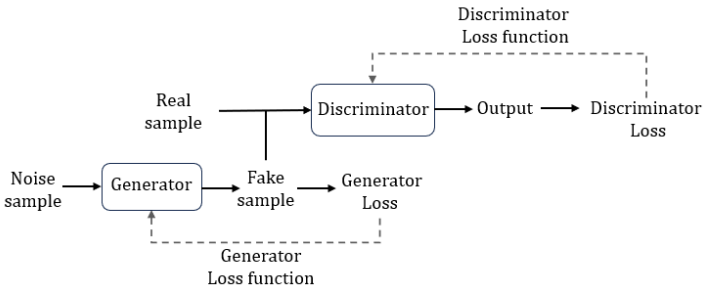


Figure 2.3: Structure of GAN training

The training process of a standard GAN [18] is shown in Figure 2.3. It consists of two parts: the discriminator D and the generator G. G is a function that takes a sample of noise $\mathbf{z} \sim p_z(\mathbf{z})$, $\mathbf{z} \in \mathbb{R}^n$ and

[1] Available at `https://chat.openai.com/chat`

outputs a generated sample $\widehat{\mathbf{y}} = G(\mathbf{z})$, $\widehat{\mathbf{y}} \in \mathbb{R}^p$. The noise distribution p_z, sometimes referred to as the latent distribution, is any distribution that can be freely sampled from. Typically, a normal distribution or uniform distribution is used, although other types of distributions have been proposed to improve the diversity of samples, the accuracy of samples, and the speed of training, such as the Bernoulli distribution [41] or mixtures of t-distributions [42].

D is a function that accepts a given dataset containing both $\widehat{\mathbf{y}}$ and real samples $\mathbf{y} \sim p_d(y)$, $\mathbf{y} \in \mathbb{R}^p$ from a dataset and estimates the probability of the given samples coming from p_d. In practice, the dataset for training the discriminator is split into real and fake samples, and the discriminator is trained on each set separately. For this thesis, $\mathbf{y}$ is a scalar output i.e. $p = 1$ and the training set consists of N_{train} pairs of $\left\{(\mathbf{x}^i, y^i)\right\}_{i=1}^{N_{\text{train}}}$ where $\mathbf{x} \in \mathrm{R}^m$ and y is used in place of $\mathbf{y}$.

By having the generator and discriminator work against one another, this can be seen as a competitive game where the generator is trying to fool the discriminator while the discriminator is trying not to be fooled. This intuition is formally defined in the GAN training objective given as [18]

$$\begin{aligned} &\min_G \max_D V(D, G) \\ V(D, G) &= \mathbb{E}_{y \sim p_d(y)}[\ln(D(y))] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})}[\ln(1 - D(G(\mathbf{z})))] \\ &= \mathbb{E}_{y \sim p_d(y)}[\ln(D(y))] + \mathbb{E}_{y \sim p_g(y)}[\ln(1 - D(y))] \end{aligned} \tag{2.12}$$

where p_g is the distribution induced by the choice of p_z and G. D is assumed to include a sigmoid layer at the output, restricting it to (0,1) i.e. $D = \sigma(\tilde{D})$ where the sigmoid function is defined as $\sigma(x) = (1 + \exp(-x)))^{-1}$ and $\tilde{D}$ is the output of D before the sigmoid is applied. This can be interpreted as the estimated probability of any given sample being real.

Finding a closed form for D and G is impossible in practice due to the large space of functions that they belong to [25]. By modelling D and G as neural networks with parameters θ_D and θ_G respectively, the search is restricted to the class of functions parameterised by θ_D and θ_G, with the bonus that standard neural network training algorithms can be used. Equation 2.12 can be split into separate training objectives for D and G as

$$\max_{\theta_D} \mathbb{E}_{y \sim p_d(y)}[\ln(D(y; \theta_D))] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})}[\ln(1 - D(G(\mathbf{z}; \theta_G); \theta_D))] \tag{2.13}$$

$$\min_{\theta_G} \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})}[\ln(1 - D(G(\mathbf{z}; \theta_G)))] \tag{2.14}$$

and the loss functions for each network training are

$$\mathcal{L}_D(\theta_D) = -\frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \ln(D(y^i; \theta_D)) - \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \ln(1 - D(G(\mathbf{z}^i; \theta_G); \theta_D)) \tag{2.15}$$

$$\mathcal{L}_G(\theta_G) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \ln(1 - D(G(\mathbf{z}^i; \theta_G); \theta_D)) \tag{2.16}$$

where the maximisation problem for D has been turned into a minimisation problem through multiplying the objective with -1. Minimisation of Equation 2.15 is done via gradient descent followed by minimisation of Equation 2.16 using the same data samples [18]. As with standard neural network training, stochastic batching is also usually applied during training, where the N_{train} samples are split into batches of N_{batch} samples each. The averaging in Equation 2.15 and 2.16 is then over each batch instead of the entire training set.

2.2.3. Conditional GANs

The conditional GAN (CGAN) formulation is an extension of the standard GAN in Section 2.2.2. It has a similar structure, except that both the generator and discriminator now accept an additional conditioning input $\mathbf{x} \in \mathbb{R}^m$ as shown in Figure 2.4. In practice, the conditioning is done by concatenating $\mathbf{x}$ to the input of both neural networks (the noise sample for the generator and the real/fake samples for the discriminator) [19].

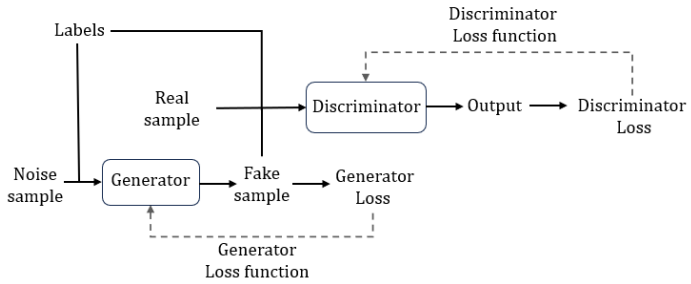


Figure 2.4: Structure of CGAN training

The corresponding training objective is then a modified version of Equation 2.17, given as [25]

$$\begin{aligned} &\min_G \max_D V_c(D,G) \\ V_c(D,G) &= \mathbb{E}_{\mathbf{x}\sim p_d(\mathbf{x})}\left[\mathbb{E}_{y\sim p_d(y|\mathbf{x})}[\ln(D(y|\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\ln(1-D(G(\mathbf{z}|\mathbf{x})|\mathbf{x}))]\right] \\ &= \mathbb{E}_{\mathbf{x}\sim p_d(\mathbf{x})}\left[\mathbb{E}_{y\sim p_d(y|\mathbf{x})}[\ln(D(y|\mathbf{x}))] + \mathbb{E}_{y\sim p_g(y|\mathbf{x})}[\ln(1-D(y|\mathbf{x}))]\right] \end{aligned} \tag{2.17}$$

The generator now learns a distribution $p_g(Y|\mathbf{x})$ that approximates the true distribution $p_d(Y|\mathbf{x})$. Furthermore, the training objective now has an additional expectation over $\mathbf{x}$ since there can be multiple y for a given $\mathbf{x}$ (for instance, multiple images of cats). The resulting loss functions for D and G simplifies to

$$\mathcal{L}_D(\theta_D) = -\frac{1}{N_{\text{train}}}\sum_{i=1}^{N_{\text{train}}}\ln(D(y^i|\mathbf{x}^i;\theta_D)) - \frac{1}{N_{\text{train}}}\sum_{i=1}^{N_{\text{train}}}\ln(1-D(G(\mathbf{z}^i|\mathbf{x}^i;\theta_G)|\mathbf{x}^i;\theta_D)) \tag{2.18}$$

$$\mathcal{L}_G(\theta_G) = \frac{1}{N_{\text{train}}}\sum_{i=1}^{N_{\text{train}}}\ln(1-D(G(\mathbf{z}^i|\mathbf{x}^i;\theta_G)|\mathbf{x}^i;\theta_D)) \tag{2.19}$$

where the expectation over $\mathbf{x}$ is done for a single sample in this thesis since $\mathbf{x}$ is continuous and there is only one y for every $\mathbf{x}^i$.

2.2.4. GAN training optimum

In theoretical GAN analysis, it is of interest to determine the form that $D(y), y \in \mathbb{R}$ should take to maximise Equation 2.12 for a fixed G. To do so, Equation 2.12 can be rewritten

$$V(D,G) = \int_y \Big(p_d(y)\ln(D(y)) + p_g(y)\ln(1-D(y))\Big)dy \tag{2.20}$$

By differentiating the term in the brackets (it is safe to do so since all values of y are considered) w.r.t. $D(y)$, it can be shown that for any G, the optimal D, denoted D^*, to maximise Equation 2.12 has the form

$$D^*(y) = \frac{p_d(y)}{p_g(y)+p_d(y)} \tag{2.21}$$

Thus, for an optimal generator that has $p_g = p_d$, $D^* = 0.5$. By inserting into D^* into Equation 2.20, it follows that $\min_G V(D^*,G) = 2D_{JS}(p_d||p_g) - 2\ln(2)$, where D_{JS} is the Jensen-Shannon divergence (JSD) given by

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}\left(p\middle\|\frac{p+q}{q}\right) + \frac{1}{2}D_{KL}\left(q\middle\|\frac{p+q}{q}\right) \tag{2.22}$$

$$D_{KL}(p||q) = \int_{\mathcal{U}} p(u)\ln\frac{p(u)}{q(u)}du \tag{2.23}$$

and D_{KL} is the Kullback-Leiber (KL) divergence. Both divergences serve as a measure of the differences between two probability distributions.

Therefore, minimising $V(D^*, G)$ w.r.t. θ_G allows for the interpretation of the generator training as changing p_g to match p_d such that the minimisation of the JSD is achieved. The JSD is bounded, having a lower bound of 0 in the case of $p_g = p_d$, which would be the case of an optimal generator.

A similar analysis can be done for the case of conditional GANs by expansion of Equation 2.17

$$\begin{aligned} V_c(D, G) &= \int_{\mathbf{x}} p_d(\mathbf{x}) \left[\int_y [p_d(y|\mathbf{x}) \ln (D(y|\mathbf{x}))] dy + \int_y [p_g(y|\mathbf{x}) \ln (1 - D(y|\mathbf{x}))] dy \right] d\mathbf{x} \\ &= \int_{\mathbf{x}} \int_y p_d(\mathbf{x}) p_d(y|\mathbf{x}) \ln (D(y|\mathbf{x})) + p_d(\mathbf{x}) p_g(y|\mathbf{x}) \ln (1 - D(y|\mathbf{x})) dy d\mathbf{x} \\ &= \int_{\mathbf{x}} \int_y p_d(\mathbf{x}, y) \ln (D(y|\mathbf{x})) + p_g(\mathbf{x}, y) \ln (1 - D(y|\mathbf{x})) dy d\mathbf{x} \end{aligned} \tag{2.24}$$

for a training pair of ($\mathbf{x}$,y), where the joint distributions are considered; $p_g(\mathbf{x}, y) = p_d(\mathbf{x}) p_g(y|\mathbf{x})$ since the generator does not estimate a distribution over $\mathbf{x}$ [25]. The resulting D^* for the CGAN is then

$$D^*(y|\mathbf{x}) = \frac{p_d(\mathbf{x}, y)}{p_g(\mathbf{x}, y) + p_d(\mathbf{x}, y)} = \frac{p_d(y|\mathbf{x})}{p_g(y|\mathbf{x}) + p_d(y|\mathbf{x})} \tag{2.25}$$

which has a similar form as the original GANs, except that D^* is now a function of the conditional distributions $p_g(y|\mathbf{x})$ and $p_d(y|\mathbf{x})$ instead.

Despite the training objective of GANs and CGANs leading to optimisation problems with suitable optima, the standard formulation has been shown to have some problems during training in practice. Gradient-related issues during training make the GAN training process unstable. GANs are also susceptible to mode-collapse, a form of training failure where there is a lack of diversity in samples generated. These problems have led to research efforts to develop variations of GANs designed to tackle such problems. More details will be discussed in the following sections.

2.2.5. Common problems in GAN training

Vanishing gradients

Early in the training process, the discriminator will usually outperform the generator [18]. Thus, one might consider first training D to optimality to improve the approximation of the loss function to the JSD, which matches the analysis of the GAN training optimum discussed previously. However, in practice, the updates to G deteriorate as D improves; The authors of [18] proposed that this was due to the saturation of Equation 2.14 early in the training process. For a close to-optimal discriminator, $D(G(\mathbf{z}; \theta_G); \theta_D)$ will be approximately 0 (low probability the generated samples come from the actual distribution). Training the generator using gradient-based methods at this value becomes slow since Equation 2.14 will have a flat loss surface and generate near-zero gradients. Hence, in the original paper [18], it is suggested that in practice, the generator training objective in Equation 2.14 be modified to

$$\max_{\theta_G} \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\ln (D(G(\mathbf{z}; \theta_G); \theta_D))] \tag{2.26}$$

$$\mathcal{L}_G(\theta_G) = -\frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \ln (D(G(\mathbf{z}^i|\mathbf{x}^i; \theta_G)|\mathbf{x}^i; \theta_D)) \tag{2.27}$$

i.e. maximise the log-probability of the discriminator being mistaken rather than minimise the log-probability of the discriminator being correct. While Equation 2.26 now does not suffer from the small gradient problem, the problems of unstable training are not completely resolved as noted in [43]. In particular, [43] proposes that if the supports of p_g and p_d are disjoint or lie on low-dimensional manifolds in high-dimensional space (which has shown to be typically the case for real-world datasets), there exists a D^* between them, leading to unreliable training of G.

Mode collapse

Another problem in GAN training is that of mode collapse. The generator's objective is to produce a large variety of plausible samples that the discriminator cannot distinguish from the real samples. However, if the generator creates an especially plausible sample, it can decide to produce only that

sample if the discriminator does not learn to reject it. As each iteration of the generator optimises for the discriminator, and if the discriminator does not manage to learn its way out of this trap, the generator ends up producing a limited range of samples.

Despite the phenomenon being known, the reason behind it is yet to be determined. In [43], the authors propose that mode collapse in standard GANs with Equation 2.26 (as is used in practice) stems from the nature of the divergence being minimised. For a fixed D^*,

$$\nabla_{\theta_G}\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[-\ln D^*(G(\mathbf{z};\theta_G))] = \nabla_{\theta_G}[D_{KL}(p_g||p_d) - 2D_{JS}(p_g||p_d)] \tag{2.28}$$

where Equation 2.26 has been multiplied with -1 for minimisation purposes. As D_{JS} is bounded and symmetric, the first term $D_{KL}(p_g||p_d)$ is of importance. In contrast with $D_{KL}(p_d||p_g)$, this form of the D_{KL}, also known as the reverse KL (RKL), has a moment matching property [35]. It suggests that in practice, stabilised GANs can produce good-looking samples while experiencing extensive mode collapse.

On the other hand, another proposal by [44] suggests that the alternate gradient update method commonly used in GAN training converges to a bad local equilibrium, and the assumptions of the minimisation of the RKL divergence in GAN training do not hold in the case of the alternate gradient update method. Regardless of the reasons for mode collapse, it is a prevalent failure mode in traditional GAN training besides vanishing gradients that has spurred research in various GAN architectures to avoid it.

2.2.6. Other GAN models

As mentioned, research has been conducted into various GAN architectures. This section covers two such GAN architectures, namely f-GANs and Wasserstein GANs (WGANs). Both GANs are also used in this thesis.

f-GANs

f-GANs are a framework of GANs proposed by [20] that aims to train GANs via a method termed as variational divergence minimisation. f-GANs focus on minimising f-divergences, a family of difference measures between probability distributions. Intuitively, minimising a statistical divergence should result in a model that approximates the actual distribution well.

For any convex, lower-semicontinuous function $f : \mathbb{R}_+ \to \mathbb{R}$ such that $f(1) = 0$, the corresponding f-divergence D_f is defined as

$$D_f(P||Q) = \int_{\mathcal{U}} q(u) f\left(\frac{p(u)}{q(u)}\right) du \tag{2.29}$$

where P and Q are distributions that have continuous pdfs p and q with respect to a base measure du defined on the domain $\mathcal{U}$ [20]. The KL divergence defined in Equation 2.23 is an example of an f-divergence, with

$$f_{KL}(u) = u \ln(u) \tag{2.30}$$

The JSD is also another f-divergence with

$$f_{JSD}(u) = -\frac{u+1}{2}\ln\left(\frac{1+u}{2}\right) + \frac{u}{2}\ln(u) \tag{2.31}$$

As shown by Goodfellow et al. [18], the standard GAN is thus minimising a particular f-GANs. Furthermore, by changing the GAN training objective $V(D, G)$ appropriately, it is possible to design GANs that minimise various f-divergences. Accordingly, this GAN formulation is termed f-GAN by [20]. The general GAN/CGAN formulation for minimising any D_f is done in [20, 25]; however, the relevant details are reproduced here for clarity purposes.

For any convex, lower-semicontinuous function, there is a convex conjugate f^* defined as [45]

$$f^*(t) = \sup_{u\in\text{dom}_f} \{ut - f(u)\} \tag{2.32}$$

This function f^* is also convex, lower-semicontinuous and has the property that $f^{**} = f$, thus f can be written as

$$f(u) = \sup_{t\in\text{dom}_{f^*}} \{tu - f^*(t)\} \tag{2.33}$$

Equation 2.33 can be interpreted as $f(u)$ able to be represented by a set of point-wise supremum of linear functions with slope t and intercept $f^*(t)$.

Applying Equation 2.33 to Equation 2.29, Nowozin et al. [20] show a connection between the f-divergence and the GAN training objective via the result

$$\begin{aligned} D_f(p_d||p_g) &\geq \sup_{T\in\mathcal{T}} \left(\mathbb{E}_{y\sim p_d(\mathbf{y})}\left[T(y)\right] + \mathbb{E}_{y\sim p_g(y)}\left[-f^*(T(y))\right]\right) \\ &= \sup_{\tilde{D}\in\mathcal{D}} \left(\mathbb{E}_{y\sim p_d(y)}\left[g_f(\tilde{D}(y))\right] + \mathbb{E}_{y\sim p_g(y)}\left[-f^*(g_f(\tilde{D}(y)))\right]\right) \end{aligned} \tag{2.34}$$

where $\mathcal{T}$ is an arbitrary class of functions $T : \mathcal{U} \to \mathbb{R}$ and T is expressed as $T = g_f(\tilde{D}(y)$ in the context of neural networks being used to approximate the functions. The discriminator is then $g_f(\tilde{D}(y))$, where g_f is the final activation layer of the discriminator and $\tilde{D}$ all previous layers. The training objective for minimising D_f is [20]:

$$\begin{aligned} &\min_G \max_{\tilde{D}} V_f(\tilde{D}, G) \\ V_f(\tilde{D}, G) &= \mathbb{E}_{y\sim p_d(y)}[g_f(\tilde{D}(y))] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[-f^*(g_f(\tilde{D}(G(\mathbf{z}))))] \\ &= \mathbb{E}_{y\sim p_d(y)}[g_f(\tilde{D}(y))] + \mathbb{E}_{\mathbf{y}\sim p_g(y)}[-f^*(g_f(\tilde{D}(y)))] \end{aligned} \tag{2.35}$$

A list of f^* and g_f for minimising various divergences are provided in [20]. As noted by [20], there is some freedom in the choice of g_f as long as it only takes values in the domain of f^* i.e. $g_f : \mathbb{R} \to \text{dom}_{f^*}$.

As before, the f-GAN framework can be extended to CGANs [25] to obtain the CGAN training objective. The CGAN training objective in the f-GAN framework is then

$$V_{c,f}(\tilde{D}, G) = \mathbb{E}_{\mathbf{x}\sim p_d(\mathbf{x})}\left[\mathbb{E}_{y\sim p_d(y|\mathbf{x})}[g_f(\tilde{D}(y|\mathbf{x}))] + \mathbb{E}_{y\sim p_g(y|\mathbf{x})}[-f^*(g_f(\tilde{D}(y|\mathbf{x})))]\right] \tag{2.36}$$

As mentioned in Section 2.2.5, usage of an alternate training objective for the standard GANs has proven beneficial. This also applies to the f-GAN framework as noted by [20]. As such, rather than minimise $\mathbb{E}_{y\sim p_g(\mathbf{y})}[-f^*(g_f(\tilde{D}(y)))]$ or $\mathbb{E}_{\mathbf{x}\sim p_d(\mathbf{x})}[\mathbb{E}_{y\sim p_g(y|\mathbf{x})}[-f^*(g_f(\tilde{D}(y|\mathbf{x})))]]$ in the case of the CGAN training objective, the generator objective for the f-GAN should be

$$\max_G \mathbb{E}_{y\sim p_g(y)}\left[g_f(\tilde{D}(y))\right] \tag{2.37}$$

or in the case of the f-CGAN:

$$\max_G \mathbb{E}_{\mathbf{x}\sim p_d(\mathbf{x})}\left[\mathbb{E}_{\mathbf{y}\sim p_g(\mathbf{y}|\mathbf{x})}\left[g_f(\tilde{D}(\mathbf{y}|\mathbf{x}))\right]\right] \tag{2.38}$$

Wasserstein-GANs

Wasserstein GANs (WGAN) [14] fall under another family of difference measures besides f-divergences known as Integral Probability Measures (IPMs). For any class of functions $\mathcal{W}$, the IPM between two distributions P and Q is defined as

$$\gamma_{\mathcal{W}}(P, Q) = \sup_{w\in\mathcal{W}} \left|\mathbb{E}_{\mathbf{y}\sim p(y)}[w(\mathbf{y})] - \mathbb{E}_{\mathbf{y}\sim q(y)}[w(\mathbf{y})]\right| \tag{2.39}$$

The choice of $\mathcal{W}$ determines the IPM used. Choosing $\mathcal{W}$ to be the class of 1-Lipschitz[2] functions lead to the dual form of the Wasserstein-1 distance. The Wasserstein-1 distance is defined as

$$W_1(P, Q) = \inf_{\gamma\in\Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma}\left[\|x - y\|\right] \tag{2.40}$$

where $\Pi(P, Q)$ is the set of all distributions $\gamma(x, y)$ defined on a compact metric set whose marginals are P and Q respectively. While intractable in its original form, the usage of the Kantorovich-Rubinstein duality [46] allows Equation 2.40 to be rewritten as

$$W_1(P, Q) = \sup_{w\in\|f\|_L\leq K} \frac{1}{K}\left(\mathbb{E}_{\mathbf{y}\sim p(y)}[w(\mathbf{y})] - \mathbb{E}_{\mathbf{y}\sim q(y)}[w(\mathbf{y})]\right) \tag{2.41}$$

[2]A function is K-Lipschitz if $\forall x \forall y$, $|w(y) - w(x)| \leq K\|y - x\|_2$

thus allowing for the Wasserstein-1 distance to be calculated up to a multiplicative constant. The authors of [14] provide the argument that optimising the Wasserstein-1 distance via gradient descent between two distributions that are disjoint (as is conjectured to be the case for real datasets) is possible due to the resulting continuity while the same is not true for loss functions resulting from other distance measures. At the same time, the WGAN has been shown to overcome problems that plague regular GANs, such as mode collapse, training instability, and convergence. Furthermore, as the loss function approximates the Wasserstein-1 distance, the sample quality is correlated with the critic loss function.

Based on Equation 2.41, the training objective of the Wasserstein GAN is given as

$$\begin{aligned} &\min_G \max_D V_W(D,G) \\ V_W(D,G) &= \mathbb{E}_{\mathbf{y}\sim p_d(\mathbf{y})}[D(\mathbf{y})] - \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[D(G(\mathbf{z}))] \\ &= \mathbb{E}_{\mathbf{y}\sim p_d(\mathbf{y})}[D(\mathbf{y})] - \mathbb{E}_{\mathbf{y}\sim p_g(\mathbf{z})}[D(\mathbf{y}))] \end{aligned} \tag{2.42}$$

with the requirement that D has to be K-Lipschitz. Unlike the standard GAN, the discriminator in a WGAN does not have a final sigmoid layer. Thus, its output is not a probability but a scalar score that can be interpreted as an evaluation of the sample quality (how real or fake it is). Thus, the discriminator in the WGAN is commonly called the critic instead.

The conditional variant of the WGAN (WCGAN) is

$$\begin{aligned} &\min_G \max_D V_{c,W}(D,G) \\ V_{c,W}(D,G) &= \mathbb{E}_{\mathbf{x}\sim p_d(\mathbf{x})}\left[\mathbb{E}_{y\sim p_d(y|\mathbf{x})}[D(y|\mathbf{x})] - \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[D(G(\mathbf{z}|\mathbf{x})|\mathbf{x})]\right] \end{aligned} \tag{2.43}$$

Going back to the smoothness of the Wasserstein-1 distance, this property allows for the continued training of the critic till optimality [14] since the more the critic is trained, the more reliable the gradient of the Wasserstein-1 distance as it is almost differentiable everywhere. A side benefit is that mode collapse is also avoided due to the lack of vanishing gradients from the Wasserstein-1 distance. This allows the critic to avoid being stuck in local optima and discern the outputs the generator stabilises on. Typically, the critic is trained 5 times for each generator iteration, although the number of training iterations per epoch for the critic is a hyperparameter that can be adjusted accordingly. The approximation of the Wasserstein-1 distance also improves as the quality of the critic improves, as discussed previously.

One difficulty with WGANs is how the Lipschitz constraint should be enforced. As an initial solution, the authors of [14] proposed to clamp the weights of the critic neural network to a fixed interval while noting that better methods could be utilised. Hence, there has been research into other methods of enforcing the Lipschitz constraint, most notably [47], where they propose adding a gradient penalty (GP) to the training objective to directly constrain the gradients of the critic's output with respect to its input:

$$\begin{aligned} &\min_G \max_D V_{W,GP}(D,G) \\ V_{W,GP}(D,G) &= \mathbb{E}_{y\sim p_d(y)}[D(y)] - \mathbb{E}_{y\sim p_g(y)}[D(y))] + \lambda \mathbb{E}_{\hat{y}\sim p(\hat{y})}\left[(||\nabla_{\hat{y}} D(\hat{y})||_2 - 1)^2\right] \end{aligned} \tag{2.44}$$

where λ is another hyperparameter that constrains the critic to be K-Lipschitz. $\hat{y}$ is defined as a linear combination of real and generated samples $\hat{y} = ty + (1-t)G(\mathbf{z})$, $t \sim \mathcal{U}(0,1)$. The justification for this constraint comes from the proposition in [47] that while a differentiable function is 1-Lipschitz i.f.f. it has gradient norms of at most 1 everywhere, enforcing the Lipschitz constraint everywhere is intractable. However, it proves sufficient to enforce it on straight lines with gradient norm 1 connecting coupled points from p_g and p_d. This form of Lipchitz constraint, also known as WGAN-GP, has been shown to provide better sample quality and training speed. The penalty term in Equation 2.44 can be extended to WCGANs in Equation 2.43 as

$$\mathbb{E}_{\mathbf{x}\sim p_d(\mathbf{x})}\left[\lambda \mathbb{E}_{\hat{y}\sim p(\hat{y}|\mathbf{x})}\left[(||\nabla_{\hat{y}} D(\hat{y}|\mathbf{x})||_2 - 1)^2\right]\right] \tag{2.45}$$

Another variant of the Lipschitz constraint called WGAN-LP, also proposed by Gulrajani et al. [47] as a "one-sided penalty" and further investigated by Petzka et al. [48], uses the form

$$\begin{aligned} &\min_G \max_D V_{W,LP}(D,G) \\ V_{W,LP}(D,G) &= \mathbb{E}_{y\sim p_d(y)}[D(y)] - \mathbb{E}_{\mathbf{y}\sim p_g(y)}\left[D(y))\right] + \lambda \mathbb{E}_{\hat{y}\sim p(\hat{y})}\left[\left(\max\left\{0, ||\nabla_{\hat{y}} D(\hat{y})||_2 - 1\right\}\right)^2\right] \end{aligned} \tag{2.46}$$

with the conditional variant similar to the WCGAN-GP formulation described above. In Petzka et al. [48], they suggest that the original WGAN-GP formulation, while true, holds only for an optimal critic that is differentiable and that y and $G(\mathbf{z})$ are sampled from the optimal coupling π^* i.e. $(y, G(\mathbf{z})) \sim \pi^*$, which may not be the case in practice. Hence, this penalty, which penalises gradient norms greater than one in contrast to the original formulation that encourages the gradient norm to go towards one (termed as two-sided penalty in [47]), offers a less strict regularisation method. The WGAN-LP method has been shown to be less sensitive to the values of λ while providing more stable training.

2.3. Evaluation

For a CGAN, given a data distribution p_d and a model distribution p_g, the goal is to ensure that the conditional distributions $p_g(\mathbf{y}|\mathbf{x})$ is as close to $p_d(\mathbf{y}|\mathbf{x})$ as possible. However, in an implicit probabilistic model like CGANs, p_g is usually unknown. In addition, p_d is also unknown, which creates further complications in evaluating its performance. While the actual form of p_d is known for a synthetic dataset, the problem of finding a representation for p_g remains. Various methods exist, such as Fréchet Inception Distances (FID) [49] for evaluating GANs; however, those methods focus on high-dimensional data such as images.

One evaluation method for datasets in this thesis is calculating the log-likelihood of the generated dataset. Another option is calculating the Wasserstein distance between the real and generated datasets. The following sections will discuss how these metrics are calculated.

2.3.1. Log-likelihood

The log-likelihood estimates how close p_g is to p_d. A high log-likelihood suggests that the test data comes from p_g, which in turn implies that p_d is close to p_g. The log-likelihood is also connected to the KL divergence; maximising the log-likelihood is equivalent to minimising $D_{KL}(p_d(y|\mathbf{x})||g_g(y|\mathbf{x}))$.

Log-likelhood estimation has been used as an evaluation metric by Goodfellow et al. [18], and this is also applied to probabilistic regression with CGANs in Aggarwal et al. [23]. In the case of a CGAN, the conditional log-likelihood is used, although it is still referred to as the log-likelihood. For this thesis, the average log-likelihood over N_{test} samples is calculated as

$$\begin{aligned} LL &= \frac{1}{N_{\text{test}}} \log \prod_{i=1}^{N_{\text{test}}} p_g(y^i|\mathbf{x}^i) \\ &= \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \log(p_g(y^i|\mathbf{x}^i)) \end{aligned} \tag{2.47}$$

on a test set $\left\{(\mathbf{x}^i, y^i)\right\}_{i=1}^{N_{\text{test}}}$ from p_d, and p_g can be estimated via kernel density estimates (KDE) [18]. KDE makes use of kernel functions as a basis for an approximation of a distribution of unknown form, in this case the conditional distribution of p_g:

$$p_g(y|\mathbf{x}) \approx \frac{1}{N_{kde}} \sum_{i=1}^{N_{kde}} K(\hat{y}^i, y; h) \tag{2.48}$$

where $\left\{(\mathbf{x}^i, \hat{y}^i)\right\}_{i=1}^{N_{kde}}$ is a set of generated samples, and h is a tunable hyperparameter, also known as the bandwidth of the kernel K. There are many choices for K; the Gaussian kernel [35] is used in this thesis:

$$K(\hat{y}^i, y; h) = \frac{1}{\sqrt{2\pi h^2}} \exp\left(-\frac{\|y - \hat{y}^i\|^2}{2h^2}\right) \tag{2.49}$$

However, caution should be exercised when using log-likelihood estimates to evaluate GANs, according to Theis et al. [50]. For instance, a model can overfit to training data and produce poor samples; then it will have a high log-likelihood on training data but poor log-likelihood on unseen test data. At the same time, the higher the dimensionality of the dataset, the higher the log-likelihood will be compared to lower-dimensional data. Since this thesis focuses on a 1-dimensional output, the negative issues with using log-likelihood estimates are less severe. The log-likelihood estimate for this thesis is not used as an absolute metric but rather as an initial evaluation of the model's performance over the training process and a comparison between different models.

2.3.2. Wasserstein Distance

The Wasserstein distance is a distance metric for two different probability distributions. It is symmetric, non-negative, and satisfies the triangle inequality, making it a true distance metric. Formally, the Wasserstein-p distance for p_d and p_g defined on a compact metric space with p-moments is

$$W_p(p_d, p_g) = \left(\inf_{\gamma \in \Pi(p_d, p_g)} \mathbb{E}_{(y,\hat{y})\sim\gamma} \left[\|y - \hat{y}\|^p \right] \right)^{1/p} \tag{2.50}$$

where $\Pi(p_d, p_g)$ is the set of all couplings of p_d and p_g. In the case of 1-D distributions, the Wasserstein distance also has the form

$$W_p(p_d, p_g) = \left(\int_0^1 \left| F_d^{-1}(t) - F_g^{-1}(t) \right|^p dt \right)^{1/p} \tag{2.51}$$

where F_d^{-1} and F_g^{-1} are the quantile functions of p_d and p_g respectively. The Wasserstein distance is linked to the optimal transport problem [46] where the goal is to transform one distribution into another via a transport plan with a minimum cost; this is the optimal plan. The Wasserstein-1 distance is then the optimal cost if the moving cost is the Euclidean distance between two points from each distribution.

3

Methodology

This chapter contains the methodology employed for this thesis. The first part introduces OpenFAST, the multiphysics tool used to simulate the aero-hydro-elastic simulation of single wind turbines. Following that, the simulation details for producing the dataset used in the training of the CGAN are discussed.

Next, the third part discusses the CGAN implementation for this thesis. Different CGAN models were implemented using PyTorch, each with its own training objective and described in Section 3.3. Network architectures used in this thesis are also discussed in this section, along with the surrogate model training details. The evaluation metrics used for evaluating the performances of the CGAN models on all the datasets are discussed in Section 3.4. Finally, the details of the datasets used in this thesis are presented in Section 3.5.

3.1. OpenFast

3.1.1. Introduction

OpenFAST [51] is a multiphysics tool designed to simulate the coupled dynamic response of single wind turbines subjected to aero-hydro-servo-elastic loads. The engineering models in OpenFAST are based on analytical theory from the fundamental laws of physics, with various simplifications and assumptions made. It is also supported with computational and empirical data where necessary [51]. Various wind turbine configurations subject to various operating conditions can be simulated in OpenFAST [51]. The architecture of OpenFAST is shown in Figure 3.1.

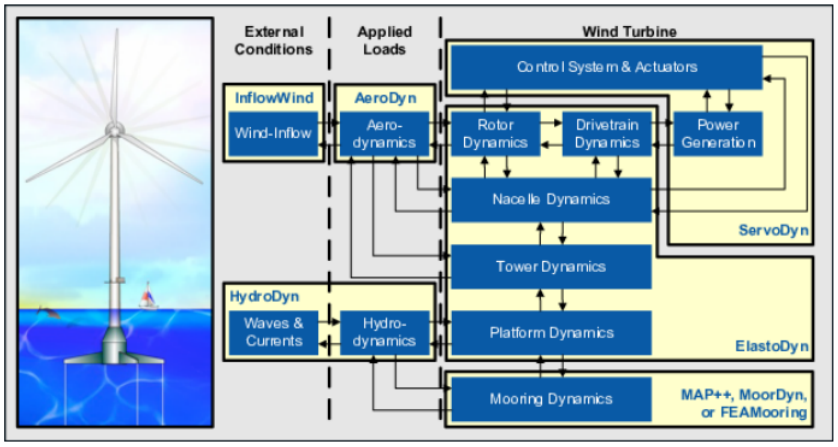


Figure 3.1: OpenFAST schematic [51]

3.1.2. OpenFAST modules

At the heart of OpenFAST is a code that couples various modules, each of which handles different aspects of the simulation, ranging from the generation of the stochastic environmental conditions such

as waves, currents, and full-field flows to modelling the aerodynamic loads experienced by the blades and the hydrodynamic loads experienced by an offshore wind turbine substructure, to name a few outputs. The relevant OpenFAST modules used for the simulations used in this thesis are discussed in this section.

Turbsim
Turbsim uses a statistical model to simulate the time series of three-dimensional wind speed vectors in a two-dimensional vertical rectangular grid fixed in space [51]. The Turbsim code generates randomised coherent turbulent structures that are superimposed on the more random background turbulent fields. The result is a full-field flow with bursts of coherent turbulence that reflect the observations made of actual flows [52].

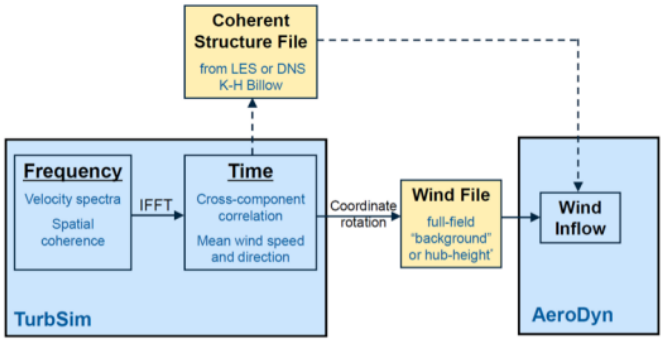


Figure 3.2: Turbsim schematic [52]

InflowWind
This module takes as input the coordinates of various points and returns the undisturbed wind-inflow velocities at these positions. The output of Turbsim can be used as an input to InflowWind, interpolating its generated fields in both time and space to obtain local wind speeds via Taylor's frozen turbulence hypothesis [51].

AeroDyn
Aerodyn is responsible for calculating the aerodynamic loading on the blades and tower based on the principle of actuator lines, where the 3D flow around a body is approximated via the 2D flow at cross sections. The 2D aerodynamic loads and moments are lumped at nodes distributed across the length of the blade and tower and are calculated as distributed loads per unit length. To obtain the total 3D aerodynamic loads, the 2D distributed loads are integrated along the length. The actuator line approximation places limits on the types of models investigated and requires treatment to ensure its accuracy [51].

AeroDyn consists of six submodules: rotor wake/induction, blade airfoil aerodynamics, tower influence on the flow local to the blade nodes, tower drag, aeroacoustics and buoyancy on the blades, hub, nacelle, and tower for floating wind turbines. It assumes the turbine geometry consists of a one, two, or three-bladed rotor on a single tower that is straight and vertical. When coupled to OpenFAST, AeroDyn receives the instantaneous structural position, orientation, and velocities of the relevant nodes, together with the local freestream velocities at the nodes, and returns the aerodynamic loads to OpenFAST, which can be used by other modules such as ElastoDyn for aeroelastic calculations.

AeroDyn also offers various wake models for the calculation of the influence of the wake. The options include blade momentum element theory (BEMT), dynamic blade momentum element theory (DBEMT), or cOnvecting LAgrangian Filaments (OLAF). The blade element momentum model is based on the quasi-steady BEM theory, where the induction changes instantaneously with loading changes. The DBEMT model accounts for induction dynamics as a result of transient conditions. Alongside the BEM models are correction factors; Glauert's empirical correction for replacing the linear momentum balance at high axial induction factors is included. Other 3D corrections available include Prandtl's tip-loss, Prandtl's hub-loss, and Pitt and Peters's skewed-wake model. The OLAF model is a free-vortex wake model for calculating the aerodynamic forces on moving horizontal axis wind turbines [53]. It

does so by modelling the blade using a lifting-line representation, characterised by a distribution of bound circulation - the spatial and temporal evolution of the bound circulation results in vorticity being shed in the wake. OLAF solves for the wake in a time-accurate manner, allowing the vortices to stretch, convect and diffuse [53]. In this study, the BEMT model is used since OLAF usage leads to long computational times. The methodology can however be extended to simulations performed with OLAF.

For the airfoil aerodynamics, there is the option of steady or unsteady aerodynamics. In the steady model, static airfoil data, such as lift, drag, pitching moments, and pressure coefficients versus angle of attack, is used to calculate the nodal loads. The unsteady model allows for modelling flow hysteresis, including unsteady attached flow, trailing-edge flow separation, dynamic stall, and flow reattachment. The unsteady aerodynamic models available in AeroDyn are the original Beddoes-Leishman model and extensions to the model made by González and Minnema/Pierce.

The tower's influence on the fluid local to the nodes on the blades is determined via a potential-flow model where the tower is modelled as a cylinder. Corrections such as the Bak correction or the tower shadow model can be used if needed. Finally, the tower's aerodynamic load depends on the tower diameter, drag coeffcient, and the local relative fluid velocity between the undisturbed freestream and structure at each tower node.

HydroDyn
HydroDyn is responsible for calculating the hydrodynamic loads on a structure given the position, orientation, velocities, and accelerations of the structures through potential-flow theory solution, strip-theory solution, or a combination of both. HydroDyn can generate waves internally for finite depth using first-order or first plus second-order wave theory [51]. Alternatively, externally-generated wave kinematics can be used within HydroDyn. The waves generated within HydroDyn can be periodic with the option of specifying the phase or irregular and long-crested or short-crested waves. HydroDyn offers the option of using the JONSWAP or white noise spectrum as the wave kinematic model for the generation of irregular waves. Other options available for wave generation include the wave height, wave direction, and the kinematic model for incident wave stretching.

Besides waves, currents can also be specified, together with options for floating structures. For this thesis, any offshore wind turbine simulations are done with no currents present in the simulations. Settings related to floating structures are also not relevant as the wind turbine is fixed to the ground/seabed.

ElastoDyn
ElastoDyn uses inputs such as the aerodynamic loads from AeroDyn or the hydrodynamic loads from HydroDyn for the calculation of the time variation of the degrees of freedom (DoF) of the system. ElastoDyn uses the Euler-Bernoulli beam model, where members are modelled as straight and isotropic structures; therefore only bending effects are modelled without torsional coupling effects. In ElastoDyn, the DoFs of the turbine can be specified, such as the blade flapwise modes, edgewise blade modes, fore-aft tower bending modes, and side-to-side tower bending modes. For floating platforms, additional DoFs can be enabled. The turbine configurations, initial conditions, and structural properties of the turbine, such as mass and inertia of the blade, tower, hub, and nacelle, are also specified here. Other input data that may be of interest include the turbine drivetrain and rotor-teeter settings. ElastoDyn can output quantities such as but not limited to: blade root bending moments, tower-top bending moments and tower-base bending moments.

3.2. Complex engineering model

3.2.1. Model Setup

Simulations in OpenFAST are performed on the IEA-10MW offshore reference wind turbine [54] for 900 seconds, with the first 300 seconds discarded to remove any transient results. Turbsim was used to generate the inflow turbulence using a grid resolution of 40 points on a $1.16D$x$1.16D$ domain, D being the rotor diameter in this case. Details of the input features for both simulations are discussed in Section 3.2.2.

While the reference wind turbine is primarily designed as an offshore wind turbine, it is also simulated as an onshore wind turbine to evaluate any differences between the training of onshore and offshore conditions. The aerodynamic loads of the onshore simulations are done with AeroDyn using

the BEMT model with the Pitt/Peters skewed-wake correction model and Prandtl's tip/hub loss model. The default value was used for the constant in the Pitt/Peters skewed wake model. In addition, the tangential induction was factored in the BEM calculations together with the drag terms for both the axial and tangential induction calculations. As for the airfoil aerodynamics model, the Minnema/Pierce variant of the Beddoes-Leishman unsteady model was used. The tower's influence on the flow around the blades and the tower aerodynamics are also included, with the tower length discretised into 10 nodes.

For the offshore simulations, the wind turbine is also subjected to hydrodynamic loads, which are calculated via HydroDyn. Irregular first-order waves are generated using the JONSWAP Spectrum wave kinematic model, with no stretching incident waves. No wave directional spreading function was used. The offshore simulations also used the same settings as the onshore counterpart in AeroDyn.

ElastoDyn was used to calculate the wind turbine dynamics and the loads experienced by the tower. All simulations are performed with single precision to save time while preserving the accuracy of the results.

3.2.2. Input features

Onshore

3 features are used for the onshore model: the wind speed U, power-law exponent α, and turbulence intensity TI. These parameters are shown to have the highest impact on the load response in previous studies [8]. The variable bounds are also based on [8] and listed in Table 3.1 below. The scatter plots of the relevant variables are shown in Figure 3.3. R and z are the rotor and hub radius respectively, defined in [54]. The samples are drawn from a 3D Sobol sequence to ensure an even spread of points in the sampling space [55]. The turbulence random seed used by Turbsim is not included as a training variable to ensure that every sample drawn from the sample space is associated with a unique seed without any repetitions.

Offshore

The wind turbine is now subjected to aerodynamic and hydrodynamic loading. As such, there are 3 additional input features along the parameters in the onshore simulations. The extra parameters are: wave height Hs, wave period Tp and wave direction $Wdir$, totaling 6 input features. The scatter plots of the variables are shown in Figure 3.3. The values for Hs and Tp are obtained from data provided by the SIMAR point 4038006 from the Spanish Port Authority [56]. The aerodynamic variables and Turbsim solution files remain the same as in the onshore simulations. As with the turbulence seeds, the wave seeds are not included in the model training, making each input sample drawn from the sample space associated with a unique turbulence and wave seed.

3.2.3. Outputs

Of interest in this thesis are the tower base fore-aft bending moments, tower top fore-aft bending moments, blade root flapwise and edgewise bending moments. For each load channel, the mean, maximum and standard deviation of the 10-minutes load history are calculated. The 10-minute fatigue is also calculated using short-term equivalent loads (ST_DEL). ST_DEL converts the irregular time series of the original load history to a regular one with constant amplitude and frequency that produces the same amount of fatigue damage [55]. For a 1 Hz ST_DEL over a 10-minute period, it can be estimated via [8]

$$\text{ST\_DEL} = \left(\sum \frac{n_i S_i^m}{N_{\text{ref}}} \right)^{1/m} \tag{3.1}$$

Table 3.2: Wöhler coefficients for different load channels

Load channel	m
Tower top/base moments	3.5
Blade edgewise moments	8
Blade flapwise moments	10

Table 3.1: Input variables, together with their bounds, for both onshore and offshore simulations

Variable	Lower bound	Upper bound	Sampling
Wind Speed (U) [m/s]	4	25	Uniform, Sobol
Turbulence Intensity (TI) [%]	2.5	$\frac{18}{U}\left(6.8+0.75U+3\left(\frac{10}{U}\right)^2\right)$	Uniform, Sobol
Power-law exponent (α) [-]	$0.15-0.23\left(\frac{U_{\max}}{U}\right)(1-0.4\log\frac{R}{z})$	$0.22+0.4\left(\frac{R}{z}\right)\left(\frac{U_{\max}}{U}\right)$	Uniform, Sobol
Wave Significant Height (Hs) [m]	0	6	Site, pseudo-random
Wave Time period (Tp) [s]	1	21	Site, pseudo-random
Wave direction ($Wdir$) [°]	-180	180	Uniform, pseudo-random
Turbulence Random Seed [-]	-50000	50000	Uniform, pseudo-random
Wave Random Seed 1 [-]	-50000	50000	Uniform, pseudo-random
Wave Random Seed 2 [-]	-50000	50000	Uniform, pseudo-random

where N_{ref} is a reference number of cycles (600 for a 1 Hz ST_DEL over a 10-minute period). S_i and n_i are the i-th load range and number of cycles observed for the i-th load range respectively. Both are estimated from the original 10-minute load time series via a rainfall counting algorithm. m is the material-specific Wöhler coefficient obtained from the S-N curve of said material with the form $K = N \cdot S^m$, where K is the material-specific Wöhler constant. The values of the Wöhler coefficient for the wind turbine are given in Table 3.2 [55].

3.3. CGANs

3.3.1. Network architectures

The first architecture is the basic feedforward architecture, termed Feedforward. It is made up of l_h hidden Linear layers with u_h units each. For the generator, the feature vector $\mathbf{x}$ is concatenated with the latent vector $\mathbf{z}$ and passed through the network, outputting the generated sample $\hat{\mathbf{y}}$. Conversely, $\mathbf{x}$ is concatenated with y for the discriminator, where y is either the real data or the generated data in this case. The output is then $\tilde{D}(y)$ from the traditional GAN and f-GAN framework in Section 2.2.2 and 2.2.6 respectively. The structure of the network is shown in Figure 3.4. Activations were applied after every layer except for the last.

The second network architecture is called Double-Input. This network architecture is also implemented in [23] and [25]. The idea behind Double-Input is to use some initial layers to learn useful features for each network input separately [25]. The structure of the Double-Input architecture is shown in Figure 3.5. $\mathbf{x}$ and $\mathbf{z}$ (or y for the discriminator) are first separately passed through an initial network, as shown in Figure 3.5. The initial neural network has a Feedforward architecture, consisting of $l_{h,I}$ hidden Linear layers with $u_{h,I}$ units each. The outputs of both networks are then concatentated together before being passed through the main neural network, which is also a Feedforward architecture consisting of l_h hidden Linear layers with u_h units each. For the initial and main neural networks, activations were applied after every layer except for the last layer of the main neural network. The Double-Input architecture can be seen as a general case of the Feedforward architecture, where the Feedforward architecture has $l_{h.I}$ zero for the generator and discriminator.

Alongside the Double-Input architecture, another architecture called Noise-Injection, also taken from [25], is used. As with [25], this architecture is only for the generator. The structure is shown in Figure 3.6. It is similar to the Feedforward architecture in Figure 3.4 except $\mathbf{z}$ is also concatenated to the hidden representation at each layer. Thus, each layer takes as input the output of the previous

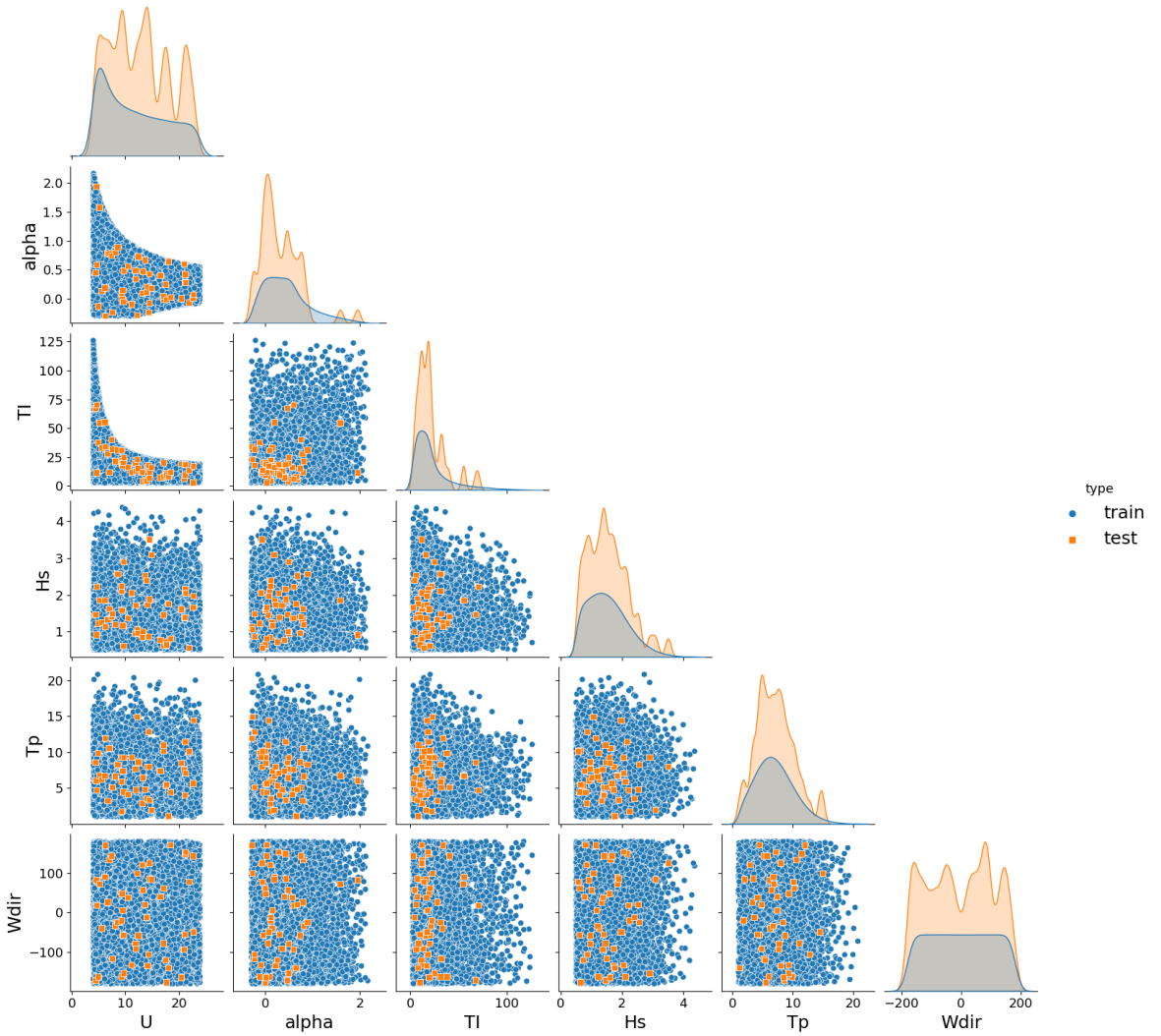


Figure 3.3: Pairplot of input features for the Aero/Aerohydro dataset, showing both training and test samples

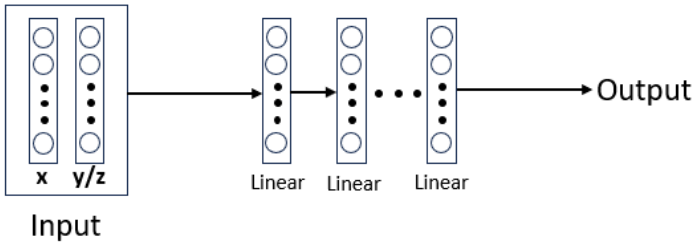


Figure 3.4: Feedforward architecture

layer concatenated with the noise vector. The idea behind the Noise-Injection architecture is to let the generator learn how early the noise should be included in the network. If the data distribution can be approximated well enough with additive Gaussian noise, the noise can be included only at the final layer. On the other hand, if the noise has to undergo a complex transformation, it can be included starting for the first layer [25].

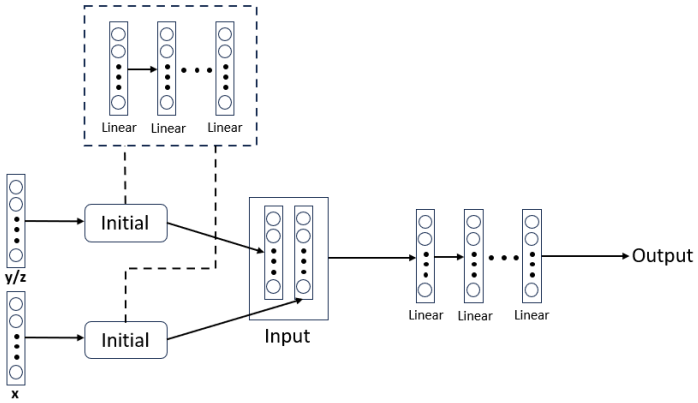


Figure 3.5: Double-Input architecture. Schematic replicated from [25]

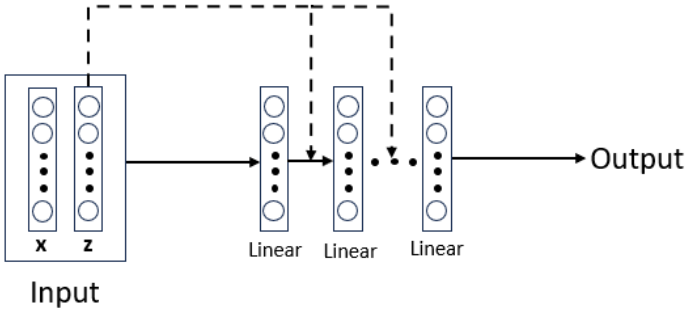


Figure 3.6: Noise-Injection architecture for the generator. Schematic replicated from [25]

3.3.2. Loss functions

Given the training set $\{(\mathbf{x}^i, y^i)\}_{i=1}^{N_{\text{train}}}$, $N_{\text{train}} \in \mathbb{R}$, $\mathbf{x} \in \mathbb{R}^m$, $y \in \mathbb{R}$, various loss functions were also implemented corresponding to the different types of CGANs implemented. Both f-GANs and WCGAN-GP are implemented, and the loss functions for each GAN will be discussed below.

f-CGAN

The objective function based on Equation 2.36 is approximated as

$$\begin{aligned} V_{c,f}(\tilde{D}, G) &= \mathbb{E}_{\mathbf{x}\sim p_d(\mathbf{x})}\left[\mathbb{E}_{y\sim p_d(y|\mathbf{x})}[g_f(\tilde{D}(y|\mathbf{x}))] + \mathbb{E}_{y\sim p_g(y|\mathbf{x})}[-f^*(g_f(\tilde{D}(y|\mathbf{x})))]\right] \\ &\approx \frac{1}{N_{\text{train}}}\sum_{i=1}^{N_{\text{train}}} g_f(\tilde{D}(y^i|\mathbf{x}^i;\theta_D)) - \frac{1}{N_{\text{train}}}\sum_{i=1}^{N_{\text{train}}} f^*(g_f(\tilde{D}(G(\mathbf{z}^i|\mathbf{x}^i;\theta_G)|\mathbf{x}^i;\theta_D))) \end{aligned} \tag{3.2}$$

Batching was applied by splitting the N_{train} into batches of size N_{batch} each. For the discriminator, the loss function with batching is

$$\mathcal{L}_D(\theta_D) = -\frac{1}{N_{\text{batch}}}\sum_{i=1}^{N_{\text{batch}}} g_f(\tilde{D}(y^i|\mathbf{x}^i;\theta_D)) - \frac{1}{N_{\text{batch}}}\sum_{i=1}^{N_{\text{batch}}} f^*(g_f(\tilde{D}(G(\mathbf{z}^i|\mathbf{x}^i;\theta_G)|\mathbf{x}^i;\theta_D))) \tag{3.3}$$

where the objective function in Equation 3.2 is negated since it has to be maximised for the discriminator.

For the CGAN generator loss, the alternative formulation in Section 2.2.6 and suggested by [20] was used. Using the same reasoning as the discriminator, the resulting loss function for the generator is

$$\mathcal{L}_G(\theta_G) = -\frac{1}{N_{\text{batch}}}\sum_{i=1}^{N_{\text{batch}}} g_f(\tilde{D}(G(\mathbf{z}^i|\mathbf{x}^i;\theta_G)|\mathbf{x}^i;\theta_D)) \tag{3.4}$$

With the loss functions for the discriminator and generator defined, the only thing left to do is to define f^* and g_f. As mentioned in Section 2.2.6, the only difference between different f-GANs lies in the choice

of f^* and g_f, which correspond to the minimisation of different f-divergences. Different f^* and g_f are given in [20], of which some were selected based on the results of [25] and implemented. Table 3.3 lists the different divergences used.

Table 3.3: f-divergences used for the f-GAN models

Divergence	$g_f(v)$	$f^*(g_f(v))$
Standard	$\log(\sigma(v))$	$-\log(1-\sigma(v))$
KL	v	$\exp(v-1)$
Reverse KL (RKL)	$-\exp(v)$	$-1-v$
Pearson χ^2	v	$v(\frac{1}{4}v+1)$

WCGAN

For the WCGAN, both the gradient penalty (GP) and Lipschitz penalty (LP) variants were implemented. Starting from the formulation defined in Section 2.2.6 and using the same approach as the f-GANs, the critic loss function with batching applied is

$$\mathcal{L}_{D,GP}(\theta_D) = -\frac{1}{N_{\text{batch}}}\sum_{i=1}^{N_{\text{batch}}}\Big\{D(y^i|\mathbf{x}^i;\theta_D) - D(G(\mathbf{z}|\mathbf{x}^i;\theta_G)|\mathbf{x}^i;\theta_D) + \lambda\left(\left\|\nabla_{\hat{\mathbf{y}}}D(\hat{\mathbf{y}}^i|\mathbf{x}^i)\right\|_2 - 1\right)^2\Big\} \tag{3.5}$$

where $\hat{\mathbf{y}}^i = t\mathbf{y}^i + (1-t)G(\mathbf{z})$, $t \sim \mathcal{U}(0,1)$. For each batch, one sample of $\mathbf{z}$ is drawn from the distribution p_z for the calculation of $\hat{\mathbf{y}}^i$. Similarly, the loss function for the WCGAN-LP is:

$$\mathcal{L}_{D,LP}(\theta_D) = -\frac{1}{N_{\text{batch}}}\sum_{i=1}^{N_{\text{batch}}}\Big\{D(y^i|\mathbf{x}^i;\theta_D) - D(G(\mathbf{z}|\mathbf{x}^i;\theta_G)|\mathbf{x}^i;\theta_D) + \lambda\left(\max\left\{0, \left\|\nabla_{\hat{\mathbf{y}}}D(\hat{\mathbf{y}}^i|\mathbf{x}^i)\right\|_2 - 1\right\}\right)^2\Big\} \tag{3.6}$$

For the experiments in this thesis, the WCGAN-LP was used as the default loss function for the WCGAN unless otherwise specified.

3.3.3. Training

In the training of the f-GANs, a single gradient descent step was taken for both the generator and discriminator. As for the WCGANs, the critic was trained 5 times per generator training step following the recommendations in Gulrajani et al. [47], using a different batch for each critic iteration. The WCGAN also uses an additional hyperparameter: the gradient penalty coefficient λ. As suggested by Gulrajani et al. [47], the default value is 10 and has been used successfully in other applications. However, upon further testing, a lower value was found to give better results for this thesis. Hence, in this thesis, the default value of λ was set to 0.02 after trying values in $\{1, 0.2, 0.02\}$.

In training the f-GANs and WCGANs, a batch size $N_{\text{batch}} = 200$ was used. Both the Adam and RMSProp optimisers have been used in the literature of GAN training, and Adam has been recommended for the training of WGAN-GP [47]. However, after initial testing, RMSProp was used as the default optimiser for training all GAN models since it showed more stable performance compared to Adam. In addition, all hidden layers used the ReLU activation unless otherwise specified.

The sampling of the noise vector $\mathbf{z}$ for the training process was done within the training loop. Unless specified otherwise, the noise was sampled from a Gaussian distribution, as it has been proven useful in existing literature [19]. The noise dimensionality was varied depending on the dataset being used, which will be discussed in the later sections.

Training of the f-GANs used the 'medium Noise-Injection' architecture from [25], the details of which are described in Table 3.4. This architecture is given the name "Base" in this thesis.

As for the WCGANs, they are trained using 3 different architectures for the generator and discriminator, which are given in Table 3.5.

Table 3.4: Network architecture for f-gan

	Generator		Discriminator		
	Architecture	l_h x u_h	Architecture	$l_{h,I}$ x $u_{h,I}$	l_h x u_h
Base	Noise-Injection	6 x 64	Double-Input	2 x 64	6 x 64

Table 3.5: Network architecture for WCGAN

	Generator		Discriminator		
	Architecture	l_h x u_h	Architecture	$l_{h,I}$ x $u_{h,I}$	l_h x u_h
WGAN-A1	Noise-Injection	6 x 64	Double-Input	2 x 64	6 x 64
WGAN-A2	Feedforward	4 x 64	Feedforward	-	4 x 64
WGAN-A3	Feedforward	7 x 64	Feedforward	-	7 x 64

3.4. Evaluation metrics

3.4.1. Log-Likelihood

Log-likelihood was used for the qualitative evaluation of the CGANs in all experiments. The log-likelihood was evaluated on the training and validation sets using KDE as an estimate. For each $\mathbf{x}^i$ in the dataset, 200 samples were generated, from which the log-likelihood $\log(p_g(y|\mathbf{x}))$ was estimated as discussed in Section 2.3.1. The KDE estimate requires specifying the scale parameter; it was selected by testing a set of values spaced logarithmically between 0.001 and 0.7 and choosing the value which maximised the log-likelihood. As the log-likelihood estimation involves various sources of randomness, such as samples being drawn and the seed used in the initialisation of the training of the model, the log-likelihood was evaluated across 10 separate runs with the best performing model chosen.

3.4.2. Conditional Wasserstein-1 Distance

The conditional Wasserstein-1 (1-W) distance was also used as a qualitative metric for the conditional distributions generated by the CGAN. The objective was to determine areas of the joint distribution $p_d(\mathbf{x}, y)$ that the CGAN is both strong and weak at predicting. The conditional Wasserstein-1 distance was calculated on the test dataset, which consists of 50 test points, each of which has 300 samples. The conditional Wasserstein-1 distances are also normalised with the standard deviation of the real p.d.fs:

$$d^i_{W_1}\left(p^i_d, p^i_g\right) = \frac{W_1\left(p^i_d, p^i_g\right)}{\sigma\left(p^i_d\right)} \tag{3.7}$$

where p^i_d, p^i_g is the p.d.f of the real and generated data at the i-th test point respectively. The calculation of the Wasserstein-1 distance on each of the conditional p.d.fs was done via the use of the Python Optimal Transport (POT) library [57]. Like the log-likelihood, the distances at each test point were evaluated across 10 separate runs.

3.5. Datasets

This section will discuss the different datasets used in the experiments. Synthetic datasets are generated to demonstrate the capabilities of the CGAN models, together with real-world datasets.

3.5.1. Synthetic datasets

Three types of synthetic datasets with different properties are generated. Two datasets originate from [25], with the only difference being the number of samples used for training. For clarity purposes, a self-contained description of the details as described by [25] is presented here. The synthetic datasets are discussed below.

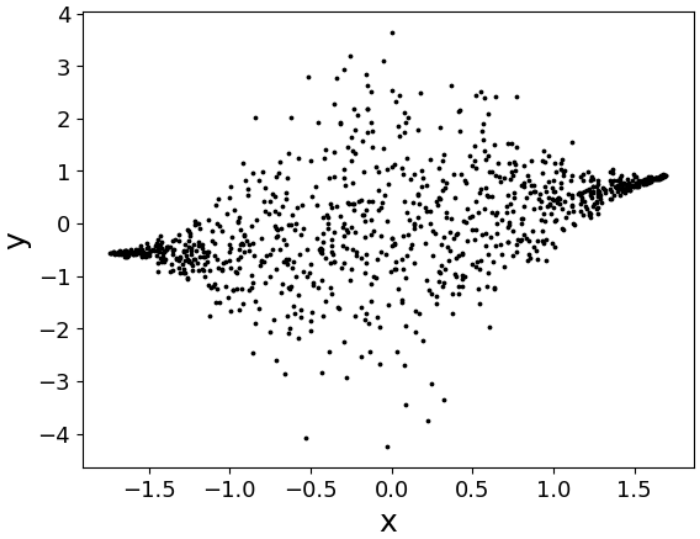


Figure 3.7: Generated test set for the heteroscedastic dataset. All values in this plot are normalised.

Heteroscedastic dataset
This dataset is a heteroscedastic Gaussian dataset borrowed from [25], where the variance is a non-linear function of $x \sim \mathcal{U}(0, 1)$. For this dataset, x and y are one-dimensional. The goal of this dataset is to evaluate the CGAN's ability to model heteroscedasticity. The details of this dataset are given by:

$$\begin{aligned} \mu(x) &= x^2 + 0.5 \\ \sigma(x) &= \sin^2(\pi x) + 0.01 \\ \epsilon &= \mathcal{N}(0, \sigma^2) \\ y &= \mu(x) + \epsilon \end{aligned} \tag{3.8}$$

2000 samples were generated, 1000 of which were used for training and the rest for log-likelihood validation. For the evaluation of the conditional 1-W distances, 50 test points were randomly chosen, with 200 samples generated for each test point.

1D Gaussian Mixtures (1D-GM)
This is a Gaussian Mixtures (GM) dataset consisting of k Gaussian distributions, each of which has a mean μ_i and variance σ_i^2 that are are functions of $x \sim \mathcal{U}(0, 1)$, as well as a weight π_i. The subscript i represents the index of the mixture component. For this dataset, x and y are 1-dimensional. The parameters associated with each component are given as

Table 3.6: Mixture details

Mixture k_i	μ_i	σ_i	π_i
k_1	$x + 1$	0.2	0.1
k_2	$2x + 2.5$	$0.3x + 0.2$	0.4
k_3	$4x + 5$	$0.7x + 0.1$	0.2
k_4	$1.8x + 10$	$0.8 - 0.5x$	0.3

The output y is then generated by

$$\begin{aligned} c &\sim \text{Categorical}(\pi_1, \ldots, \pi_i) \\ y &\sim \mathcal{N}(\mu_c, \sigma_c^2) \end{aligned} \tag{3.9}$$

where Categorical(π_1,...,π_i) is the distribution of a discrete random variable that takes the value i with probability π_i.

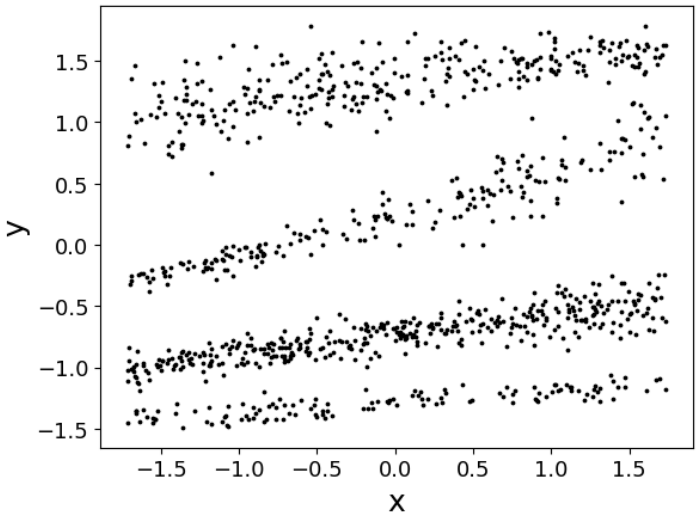


Figure 3.8: Generated test set for the GM dataset. All values in this plot are normalised.

This dataset serves as a baseline for evaluating the performance of CGANs on capturing multiple modes in the actual distribution besides heteroscedasticity. For this dataset, 5000 samples were generated, of which 3500 were used for the training process. For the evaluation of the conditional 1-W distances, 50 test points were randomly chosen, with 200 samples generated for each test point.

wmix6

The wmix6 dataset is borrowed from [25]. It comprises a mixture of Weibull distributions in contrast to the Gaussian Mixtures dataset. The Weibull distribution is defined as

$$\text{Weibull}(y;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{y}{\lambda}\right)^{k-1} e^{-(y/\lambda)^k} & \text{if } y \geq 0 \\ 0 & \text{if } y < 0 \end{cases} \tag{3.10}$$

where λ and k are the scale parameter and shape parameter respectively. By controlling the two parameters, distributions of various types can be created, leading to more complex conditional distributions than simple Gaussians [25].

Each component in the mixture is associated with 3 parameters: a shape factor k_i, offset o_i, and weight π_i. Each parameter is sampled from the uniform distribution $\mathcal{U}(0,1)$. These parameters make up the conditioning vector $\mathbf{x}$, defined as

$$\mathbf{x} = [o_1, o_2, \ldots o_{n_{comp}}, \bar{k}_1, \bar{k}_2, \ldots \bar{k}_{n_{comp}}, \hat{\pi}_1, \hat{\pi}_2, \ldots \hat{\pi}_{n_{comp}}] \in \mathbb{R}^{3n_{comp}} \tag{3.11}$$

The wmix6 dataset then has $\mathbf{x} \in \mathbb{R}^6$, and has its number of components $n_{comp} = 2$. The output $y \in \mathbb{R}$ is then generated via:

$$\begin{aligned} \pi_i &= \frac{\hat{\pi}_i}{\sum_{j=1}^{n_{comp}} \hat{\pi}_j}, \quad i = 1 \ldots n_{comp} \\ c &\sim \text{Categorical}(\pi_1, \ldots, \pi_{n_{comp}}) \\ \tilde{y} &\sim \text{Weibull}(1, (1 + \bar{k}_c)) \\ y &= \tilde{y} + 5o_c \end{aligned} \tag{3.12}$$

As mentioned, the motivation behind using the Weibull as a basis for the mixture is to generate complex conditional distributions. As the number of components increases, the complexity of the distribution increases. A good model should be able to properly estimate such a complex distribution, making such a dataset a good estimate of the performance of the CGAN model for real datasets. 6600 samples were generated, of which 3300 were used for training. For the evaluation of the conditional 1-W distances, 50 test points were randomly chosen, with 300 samples generated for each test point.

Aero_MDN
The Aero_MDN dataset is a dataset consisting of the predictions of the simulation results for the IEA-10MW offshore reference wind turbine [54] subjected to aerodynamic forces over 10 minutes using a Mixture Density Network (MDN)[2] based on [55]. MDNs are a probabilistic regression method that combines Gaussian mixture models with artificial neural networks, using the neural networks to predict the mixture parameters of the model instead of the output directly. This has been demonstrated to be a better surrogate model compared to previous methods such as chained Gaussian processes [13].

While it is only a rough approximation of the actual dataset, additional data can be generated for initial estimations of training size effects without having to perform expensive simulations. The dataset is normalised against the training data of the simulation results, the details of which are in Section 3.5.2. The validation dataset consists of 3400 points, and the test set for the conditional Wasserstein distance evaluation uses the same conditional inputs as the Aero dataset. For each test point, 300 samples from the MDN are generated, giving a total of 15000 samples.

3.5.2. Aero
The Aero dataset consists of the results of the onshore simulations performed in Section 3.2.1, available at [31]. The names of the load channels of interest are: the tower base fore-aft moments (TwrBsMyt), tower top fore-aft moments (TwrBrMyn), blade root edgewise bending moment (RootMxb), and blade root flapwise bending moment (RootMyb). The mean, maximum, standard deviation (stddev), and short-term DEL (ST_DEL) of each load channel are calculated from the simulations in Section 3.2.1.

The dataset comprises 7500 points and is split as 80% and 20% into training and validation sets. For testing, a different dataset is used to evaluate the prediction accuracy of the conditional p.d.f. This dataset consists of 50 pseudo-randomly-selected points within the range of the parameters in the training set given in Table 3.1. At each test point, 300 full-order simulations are done in OpenFAST to obtain a reference pdf, while varying the turbulence and wave random seeds for each simulation as is done for the training set. This results in a total of 15000 OpenFAST simulations. All data was normalised against the training data.

3.5.3. Aerohydro
The Aerohydro dataset is the dataset containing the results of the offshore simulations performed in Section 3.2.1, also available at [31]. It comprises 7500 points and is split as 80% and 20% into training and validation sets. The names of the load channels of interest are the same as in the Aero dataset. The test points for the Aerohydro dataset are the same as that of the Aero dataset, but with the three extra input features Hs, Tp and $Wdir$ in Table 3.1. The number of samples per test point is the same as the Aero dataset. All data is normalised against the training data of the Aerohydro dataset.

4

Experiments

In this section, different experiments were performed with the various CGAN models. In the first experiment, multiple CGAN models were trained on synthetic datasets of varying complexity. In addition, using the WCGAN as a base CGAN training objective, various neural network architectures were also trained on the same synthetic datasets. These two experiments help to gain some understanding for answering research question 1 and research question 2. The CGAN models are also evaluated on real datasets, which will help to answer research questions 3 and 4.

For the log-likelihood evaluation using the KDE, the scale parameter was restricted to be between 0.001 and 0.7 for all experiments. Early stopping is also applied as described in Section 2.1.4 for all experiments. Unless otherwise specified, the RMSProp optimiser is used for all experiments.

4.1. Different CGANs

Different CGAN models were trained on various datasets and evaluated based on the log-likelihood and conditional Waserstein-1 distances as discussed in Section 3.4. This experiment aims to evaluate the performances of the various models and identify any strengths and drawbacks of each model.

4.1.1. Experiment setup

Table 4.1: Summary of common hyperparameters used for the f-gans in Section 4.1

Hyperparameter	Value
Epochs	3000
Discriminator learning rate	0.0001
Generator learning rate	0.0001
Batch size	200
Generator optimiser	RMSProp
Discriminator optimiser	RMSProp
Architecture	'Base'
Activation function (hidden layers)	ReLU

The f-GANs listed in Table 3.4 and the WCGAN were trained on the heteroscedastic, 1D-GM, and wmix6 datasets. Through this experiment, the performance of each type of GAN can be observed for datasets with differing properties, such as heteroscedasticity and multimodality, as illustrated in Section 3.5. The specific hyperparameters used for each dataset are discussed below. For all three datasets, the f-GANs were trained for 3000 epochs each, and the generator and discriminator learning rates were both set to 0.0001. The learning rate was determined by trying values in $\{0.001, 0.0001, 0.00005\}$. All f-GANs used the 'Base' architecture as described in Table 3.4.

The WCGAN used the WCGAN-A1 architecture from Table 3.5. The WCGAN was trained for 3000 epochs on the heteroscedastic dataset and 5000 epochs on the other two datasets. The learn-

Table 4.2: Summary of common hyperparameters used for WCGANs in Section 4.1. *3000 epochs used for heteroscedastic dataset, 5000 epochs used for wmix6 and 1D-GM datasets

Hyperparameter	Value
Epochs	3000/5000*
Critic learning rate	0.0001
Generator learning rate	0.0002
Batch size	200
Generator optimiser	RMSProp
Critic optimiser	RMSProp
Architecture	'WCGAN-A1'
Activation function (hidden layers)	ReLU
Gradient penalty coefficient	0.02

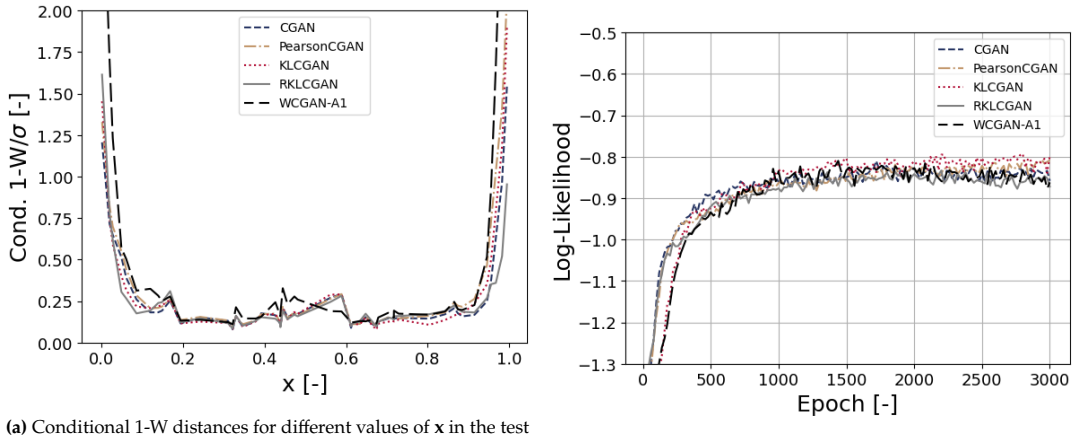
Table 4.3: Noise dimensionality used for each dataset for all CGAN models. The noise distribution used was a multivariate Gaussian

Dataset	Noise Dimensionality
Hetereoscedastic	5
1D-GM	30
wmix6	30

ing rates for the critic and generator were set at 0.0001 and 0.0002 respectively, following the two-timescale update rule (TTUR) in Heusel et al. [49]. The critic learning rate was determined from trying values in $\{0.001, 0.0001, 0.00005\}$ while the generator learning rate was chosen from values in $\{0.002, 0.0002, 0.00004\}$. All other relevant hyperparameters are given in Section 3.3.3. Finally, the noise dimensionality used for the heteroscedastic, 1D-GM, and wmix6 datasets is 5, 30, and 30 respectively.

4.1.2. Results

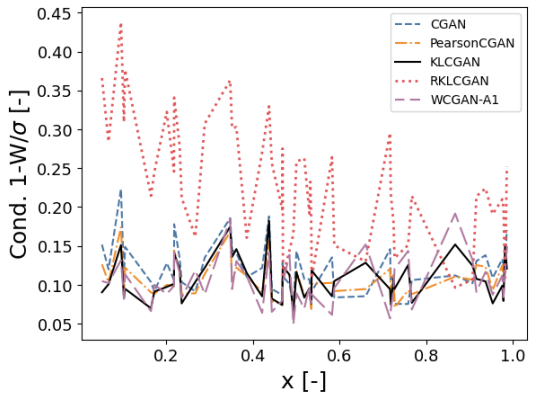
Figures 4.1-4.3 show the results from the best performing models for all 3 datasets. The conditional 1-W distances for the heteroscedastic and 1D-GM datasets are plotted against the raw values of the 50 test points. As for the wmix6 dataset, the conditional 1-W distances are plotted against the index number of the 50 test points sorted in no particular order instead. Figure B.1-B.3 shows plots of samples from the models. For the wmix6 dataset, KDE plots of samples conditioned on two different **x** are shown. 3000 samples are used for the KDE visualisation of every model at each **x**.



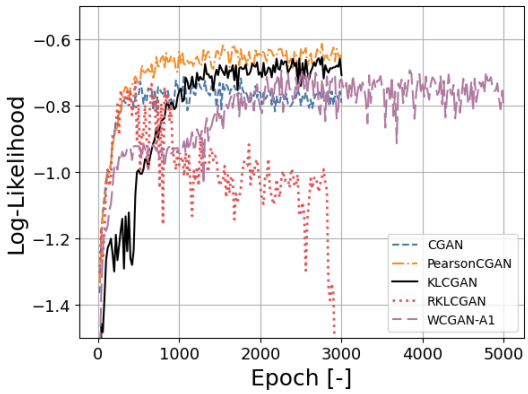
(a) Conditional 1-W distances for different values of **x** in the test dataset

(b) Validation log-likelihood against training epochs

Figure 4.1: Results for models trained on the heteroscedasatic dataset

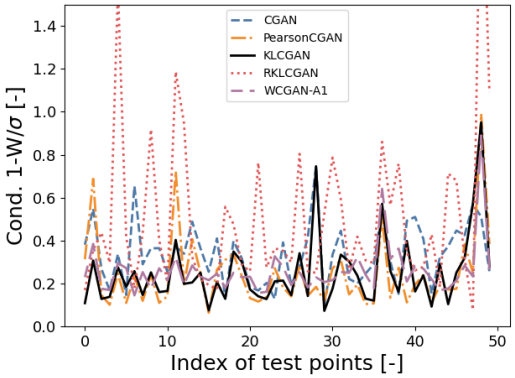


(a) Conditional 1-W distances for different values of **x** in the test dataset

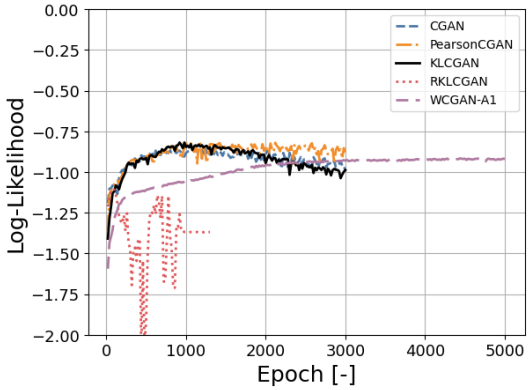


(b) Validation log-likelihood against training epochs

Figure 4.2: Results for models trained on the 1D-GM dataset



(a) Conditional 1-W distances for the 50 points in the test dataset. The index of the test points are numbered from 0-49



(b) Validation log-likelihood against training epochs

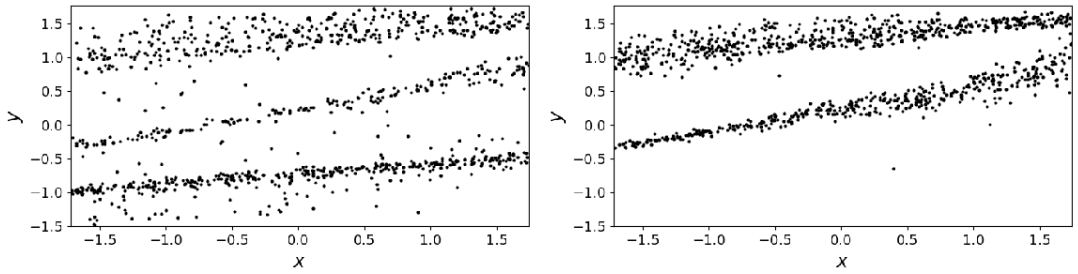
Figure 4.3: Results for models trained on the wmix6 dataset

4.1.3. Discussion

For the heteroscedastic dataset, all models display similar performances based on the log-likelihood. Of the various models, the KLCGAN has the highest log-likelihood, which is to be expected since maximising the log-likelihood is the equivalent of minimising D_{KL}. The conditional 1-W distances also indicate that most conditional distributions are well-approximated, as evidenced by the plots in Figure B.1. There are minor differences observable for the points towards the edges of the training domain, which is also reflected in the 1-W distances increasing towards 0 and 1. Although the WCGAN's objective is to minimise the 1-W distance rather than the conditional distances, it is interesting to note that other models do produce similar distances or even better distances in some areas of the distribution, especially towards the tails. Despite that, the heteroscedasticity of the distribution appears to have been captured by all the models.

On the 1D-GM dataset, the Pearson χ^2 and KL CGAN have the best performance with respect to the validation log-likelihood, although the Pearson χ^2 variant achieves a higher log-likelihood than the KLCGAN. Despite that, both models appear to have no noticeable differences in the samples generated in Figure B.2. Furthermore, while the standard CGAN appears to perform worse than the WCGAN on the validation log-likelihood, the samples generated by both models have noticeable differences, with the samples CGAN having less variance along all 4 modes than the WCGAN. Despite the differences, the conditional 1-W distances indicate similar performances across all 4 models, although the Pearson χ^2 CGAN appears to provide the most consistent performance. On the other hand, the RKLCGAN has the worst performance, as shown by the conditional 1-W distance and log-likelihood. Looking at the samples of the RKLCGAN, only 3 distinct modes are visible, which is a sign of mode collapse starting to

occur. Figure 4.4 shows the samples generated at two different epochs in the training progress. Towards the end of the training, two of the four modes are not replicated by the RKLCGAN model.



(a) Epoch: 440 **(b)** Epoch: 3000

Figure 4.4: Training progress of the RKLCGAN on the 1D-GM dataset

Next is the wmix6 dataset. Except for the RKLCGAN, all three other f-GANs produce similar log-likelihoods. While the KLCGAN does have the highest maximum log-likelihood of the three models, the log-likelihood on the validation set drops slightly with the number of epochs, indicating some slight overfitting to the training set. The conditional 1-W distances for the three models are similar, with no clear indication of which model performs best. The KDE plots in Figure B.3 show that all three models can capture the multimodality at different conditional $\mathbf{x}$. The WCGAN also performs equally well on the conditional 1-W distances and log-likelihood. As with the 1D-GM dataset, the RKLCGAN has the worst performance. During the training of the RKLCGAN, it was noticed that the process was unstable, with the model crashing prematurely. A further investigation showed that just before the model crashed, the final layer of the discriminator produced very large outputs, which, when combined with the corresponding g_f from Table 3.3, resulted in the instability. While this behaviour was not present during the training of the KLCGAN, which also uses exponentials in the loss functions, it is worth noting that the instability in the training of the RKLCGAN could be a potential problem for the KLCGAN.

4.2. Different Architectures

This experiment investigates the use of different architectures for the same datasets as Section 4.1. This experiment aims to understand how the network architecture affects the performance of WCGANs. As the WCGAN has already been trained on the A1 architecture and the results shown in Section 4.1, the primary focus is how well other architectures perform relative to the A1 architecture.

4.2.1. Experiment setup

Table 4.4: Summary of common hyperparameters used in Section 4.2

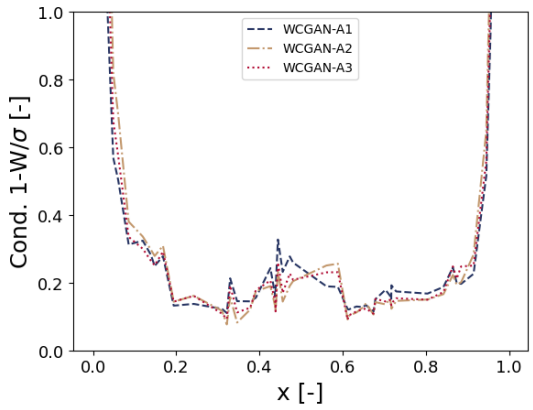
Hyperparameter	Value
Epochs	8000
Critic learning rate	0.0001
Generator learning rate	0.0002
Batch size	200
Generator optimiser	RMSProp
Critic optimiser	RMSProp
Activation function (hidden layers)	ReLU
Gradient penalty coefficient	0.02

This experiment uses the architectures from Table 3.5. As mentioned, the two architectures WCGAN-A2 and WCGAN-A3 were trained and evaluated on the heteroscedastic, 1D-GM, and wmix6 datasets. For all three datasets, the learning rates of the critic and generator were set to 0.0001 and 0.0002 for the two architectures, just as with the WCGAN-A1 model in Section 4.1. The A2 and A3 architectures were trained on 8000 epochs for the 1D-GM and wmix6 datasets. As with Section 4.1, the noise

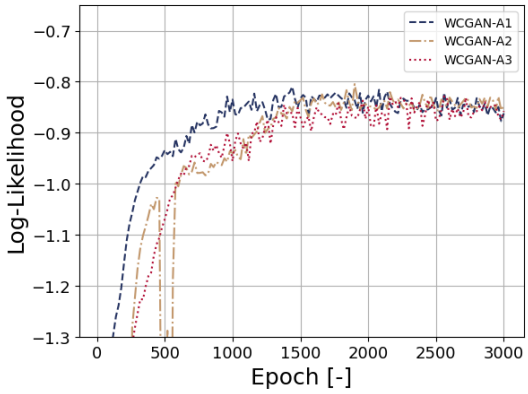
dimensionality used for each dataset in this experiment is the same, given in Table 4.3. Other relevant training hyperparameters are also in Section 3.3.3.

4.2.2. Results

The conditional 1-W distances and the log-likelihoods are shown in Figures 4.5-4.7. As with Section 4.1, the conditional 1-W distances for the heteroscedastic and 1D-GM datasets are plotted against the raw values of the 50 test points, while the conditional 1-W distances are plotted against the index number of the 50 test points for the wmix6 dataset, sorted in no particular order instead. Samples from the two architectures are also shown in Figure B.1-B.3.

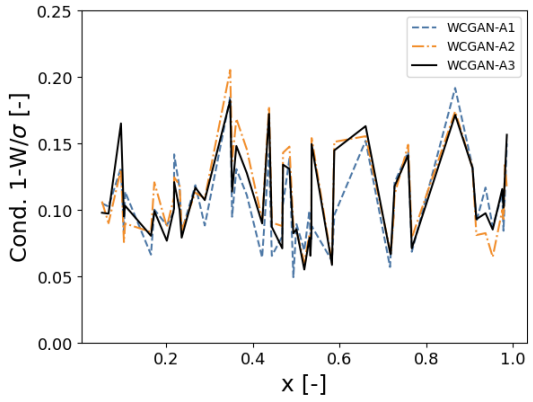


(a) Conditional 1-W distances for different values of **x** in the test dataset

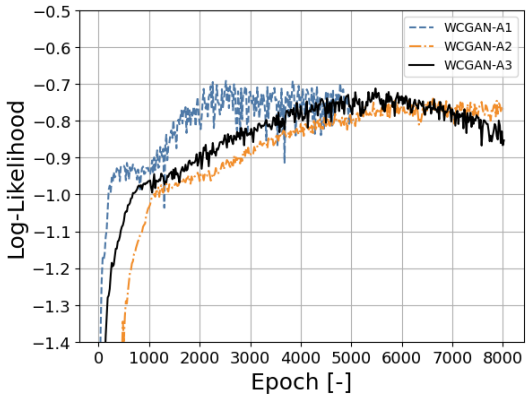


(b) Validation log-likelihood against training epochs

Figure 4.5: Results for the WCGAN models trained on the heteroscedastic dataset



(a) Conditional 1-W distances for different values of **x** in the test dataset



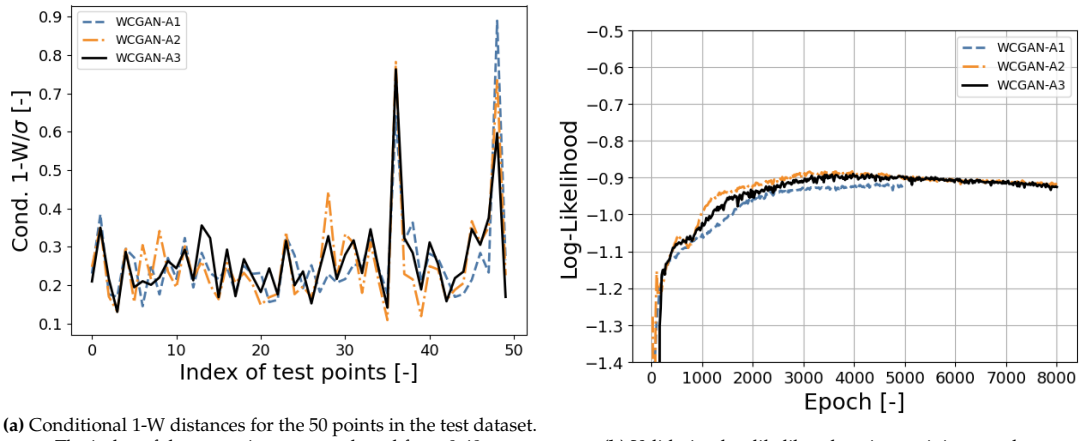
(b) Validation log-likelihood against training epochs

Figure 4.6: Results for the WCGAN models trained on the 1D-GM dataset

4.2.3. Discussion

On the heteroscedastic dataset, all three architectures achieve similar results. Looking at the scatter plots, the only noticeable differences appear to be at the tails of the distribution, which is also reflected in the conditional 1-W distances. For a simple dataset like the heteroscedastic dataset, the choice of architecture appears to play a minor role in the CGAN performance.

Moving onto the 1D-GM dataset, there appears to be slight overfitting for the WCGAN-A3 model compared to the WCGAN-A2 model, which is expected considering that the A3 architecture uses more hidden layers, thus increasing the capacity of the model. The slight overfitting appears to have a slight effect on the samples generated in part of the distribution, although the 4 modes are still present. The

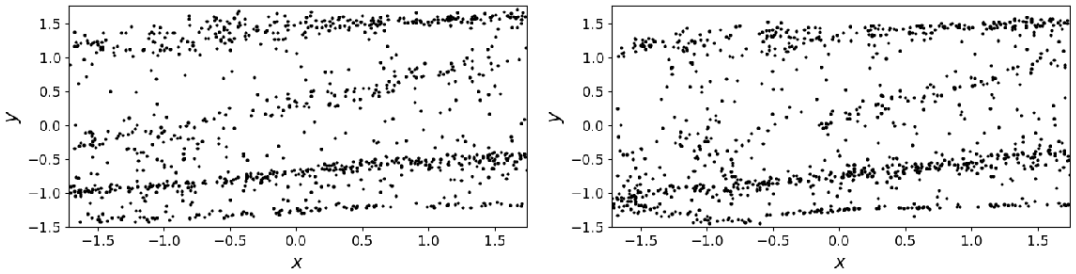


(a) Conditional 1-W distances for the 50 points in the test dataset. The index of the test points are numbered from 0-49

(b) Validation log-likelihood against training epochs

Figure 4.7: Results for the WCGAN models trained on the wmix6 dataset

maximum log-likelihoods achieved for all three models and the conditional 1-W distances on the 1D-GM datasets are similar, resulting in similar-looking scatter plots of the samples generated by each model.



(a) Epoch: 5000

(b) Epoch: 7999

Figure 4.8: Training progress of the WCGAN-A3 on the 1D-GM dataset

As for the wmix6 dataset, both the WCGAN-A2 and WCGAN-A3 models achieve slightly better log-likelihoods, although the conditional 1-W distances are largely the same as that of the WCGAN-A1 model. Looking at the KDE plots, the multimodality of the first actual condtiional distribution is better captured by both the A2 and A3 architecture than the A1 architecture. However, it is only a slight difference, as evidenced by the similar conditional 1-W distances. This suggests that while the choice of network architectures has an impact on the performance of the CGAN, it is not necessarily the deciding factor in its performance.

4.3. Aero_MDN experiment

The goal of this experiment was to investigate the effect of the sample size of the Aero/Aerohydro dataset using a placeholder dataset which approximates the actual datasets. The standard deviation (stddev) and 10-minute damage equivalent loads (DEL) of the tower base fore-aft moments (TwrBsMyt) were investigated for this experiment.

4.3.1. Experiment setup

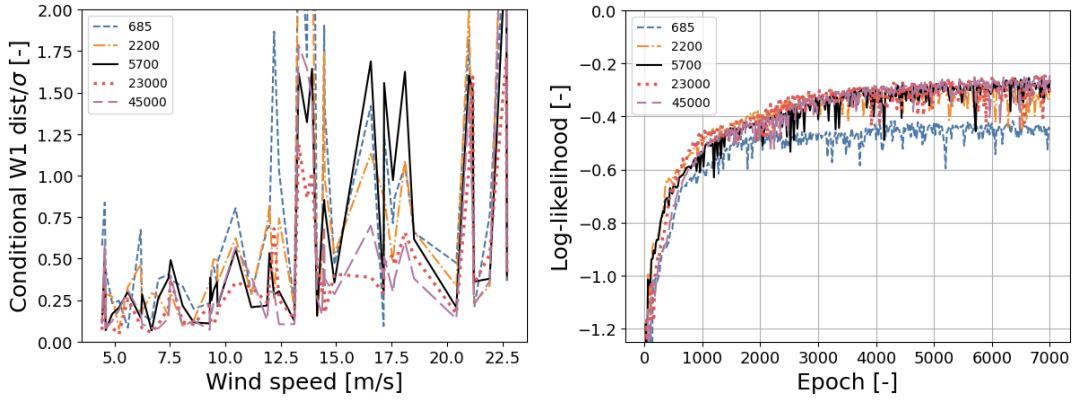
For this experiment, the WCGAN-A2 architecture was trained using five different training sizes $N_{\text{train}} = \{685, 2200, 5700, 23000, 45000\}$ of the Aero_MDN dataset. Noise dimensionality was fixed to 30, with the other relevant hyperparameters for the WCGAN given in Table 4.5. To get a better comparison of all runs, the same validation and test set for the log-likelihood estimation and conditional 1-W distance was used for all runs.

Table 4.5: Summary of common hyperparameters used in Section 4.3

Hyperparameter	Value
Epochs	7000
Critic learning rate	0.0001
Generator learning rate	0.0002
Batch size	200
Generator optimiser	RMSProp
Critic optimiser	RMSProp
Architecture	'WCGAN-A2'
Activation function (hidden layers)	ReLU
Gradient penalty coefficient	0.02

4.3.2. Results

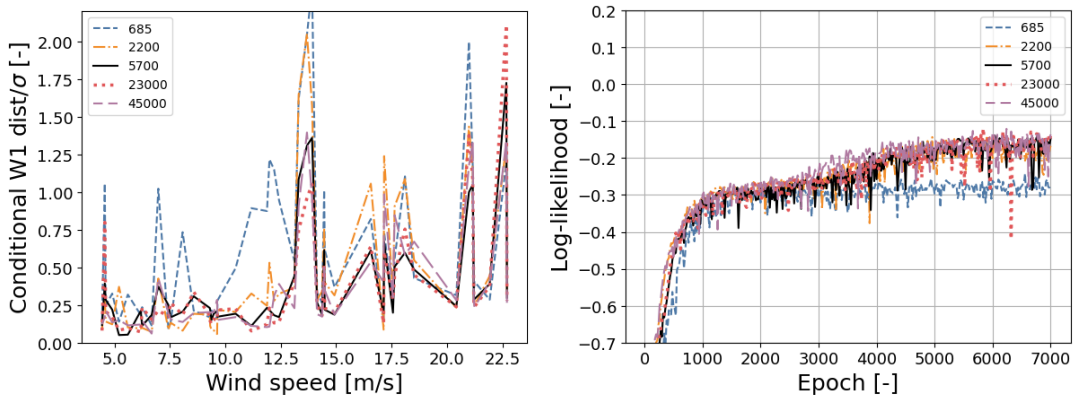
The results for the conditional 1-W distances and log-likelihoods of each sample size are shown in Figures 4.9 and 4.10. The conditional 1-W distances are plotted against the test points visualised using the wind speeds in ascending order. In addition, the maximum log-likelihoods achieved for each load channel are shown in Figure 4.11.



(a) Conditional 1-W distances for different wind speeds

(b) Validation log-likelihood against training epochs

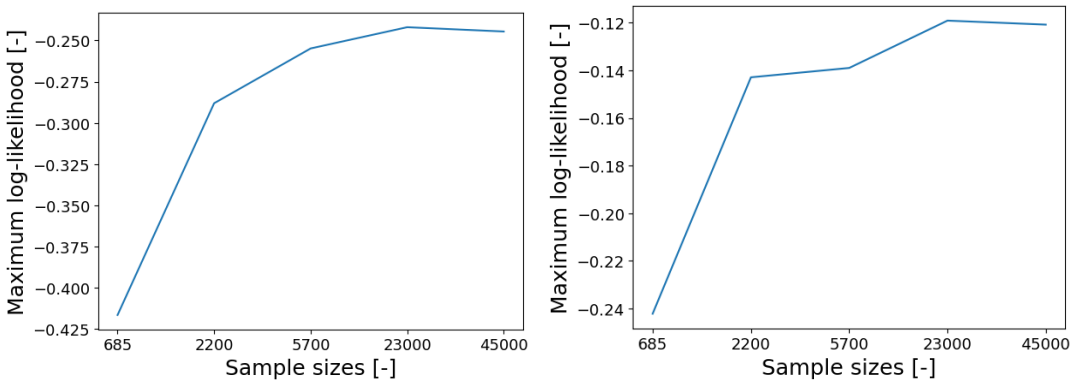
Figure 4.9: Results from models trained on the TwrBsMyt_stddev load channel using different dataset sizes. The same test/validation dataset was used for each dataset size.



(a) Conditional 1-W distances for different wind speeds

(b) Validation log-likelihood against training epochs

Figure 4.10: Results from models trained on the TwrBsMyt_ST_DEL load channel using different dataset sizes. The same test/validation dataset was used for each dataset size.



(a) TwrBsMyt_stddev **(b)** TwrBsMyt_ST_DEL

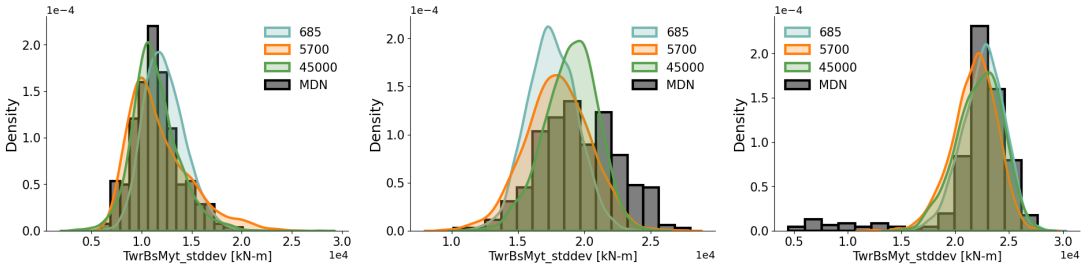
Figure 4.11: Convergence of validation log-likelihood

4.3.3. Discussion

Increasing the sample size for training appears to give an overall increase in the log-likelihood on the validation dataset, which is to be expected since the CGAN is able to learn from more data. Based on the convergence plots in Figure 4.11, there appear to be diminishing returns on the increase in log-likelihood going past 23000 training samples. Furthermore, the conditional 1-W distances for both load channels improve with increasing training size. The improvement is however largely dependent on the load channel and conditional input. Figures 4.12 and 4.13 illustrate the differences in pdf based on training size for both load channels at the medium wind speed condition; the pdf for the TwrBsMyt_ST_DEL load channel at 5700 training samples fits the reference data better than the pdf from 685 training samples. At the same time, increasing the number of training samples does not allow the model to replicate the left tail of the reference pdf for the TwrBsMyt_stddev load channel at the high wind speed condition. Despite that, increasing the training size does appear to provide an overall increase in accuracy.

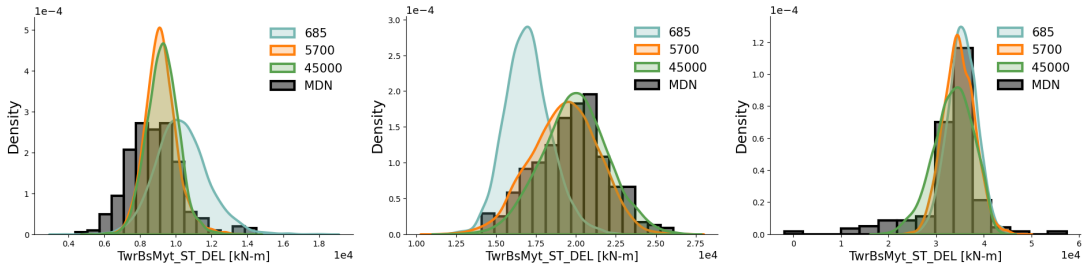
Table 4.6: Values of input features for the pdfs in Figure

Wind condition	U [m/s]	TI [-]	α [-]
Low wind speed	7.0	12.0	0.79
Medium wind speed	12.0	12.9	0.35
High wind speed	21.2	18.4	0.28



(a) Low wind speed **(b)** Medium wind speed **(c)** High wind speed

Figure 4.12: Predicted and reference (MDN) conditional pdf for the TwrBsMyt_stddev load channel in the Aero_MDN dataset



(a) Low wind speed **(b)** Medium wind speed **(c)** High wind speed

Figure 4.13: Predicted and reference (MDN) conditional pdf for the TwrBsMyt_ST_DEL load channel in the Aero_MDN dataset

4.4. Real Datasets

Moving on to real datasets, the performance of the f-GANs and WCGANs on both the Aero and Aerohydro dataset were investigated. 5 load channels were selected for this purpose: the tower base fore-aft moments standard deviation (TwrBsMyt_stddev), the tower base fore-aft moments 10-minute DEL (TwrMsMyt_ST_DEL), the tower top fore-aft moments 10-minute DEL (YawBrMyn_ST_DEL), the blade root flapwise moment 10-minute DEL (RootMyb1_ST_DEL) and the blade root edgewise moment 10-minute DEL (RootMxb1_ST_DEL).

4.4.1. Experiment Setup

The Pearson χ^2 CGAN and the WCGAN were considered for both the Aero and Aerohydro dataset. The Pearson χ^2 CGAN used the 'Base' architecture and was trained for 3000 epochs. The learning rate of the discriminator and generator for the Pearson χ^2 CGAN was set to 0.0001 after using the same grid search for the learning rates in Section 4.1. The WCGAN was trained on architectures A1 to A3, and the learning rates were set to 0.0002 and 0.0001 for the generator and critic respectively. All three WCGAN models were trained for 30000 epochs to ensure convergence. For both datasets, the noise dimensionality was set to 30.

Table 4.7: Summary of hyperparameters for the Pearson χ^2 CGAN used in Section 4.4

Hyperparameter	Value
Epochs	3000
Critic learning rate	0.0001
Generator learning rate	0.0001
Batch size	200
Generator optimiser	RMSProp
Discriminator optimiser	RMSProp
Architecture	Base
Activation function (hidden layers)	ReLU

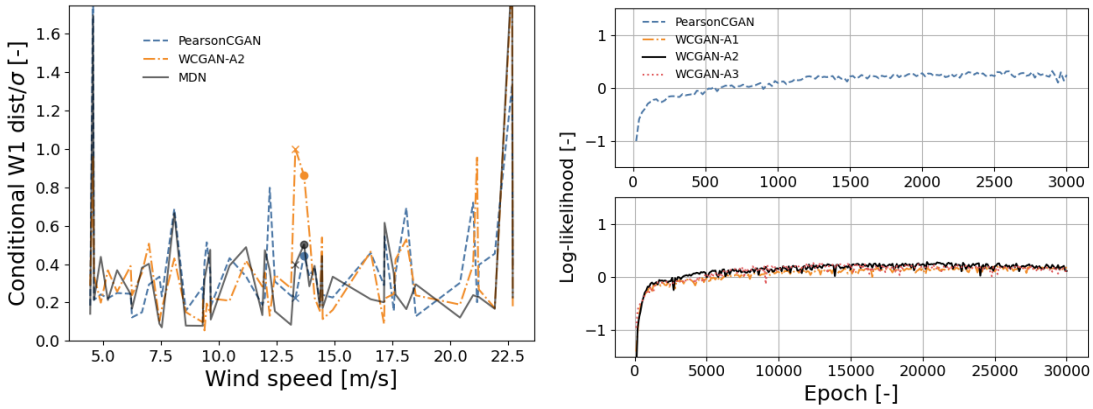
Table 4.8: Summary of common hyperparameters for the WCGANs used in Section 4.4

Hyperparameter	Value
Epochs	30000
Critic learning rate	0.0001
Generator learning rate	0.0002
Batch size	200
Generator optimiser	RMSProp
Critic optimiser	RMSProp
Activation function (hidden layers)	ReLU
Gradient penalty coefficient	0.02

4.4.2. Results

The results for the tower base and tower top fore-aft moments 10-minute DEL of the Aero dataset are shown in Figures 4.14-4.16, and Figures 4.18-4.20 for the Aerohydro dataset. As with Section 4.3, the conditional 1-W distances are plotted against the wind speeds of the test points. The rest of the results are shown in Appendix C and D. Log-likelihood plots of the Pearson χ^2 CGAN are shown separately from the other plots for clarity. For the same reason, only the conditional 1-W distances and KDE plots of the samples from the Pearson χ^2 CGAN, WCGAN-A2 and the MDN dataset are shown. The same test points in Table 4.6 were used for the KDE plots.

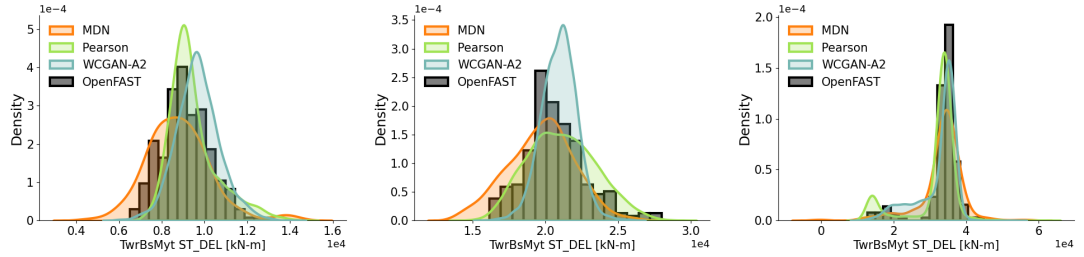
Aero



(a) Conditional 1-W distances for different wind speeds

(b) Validation Log-likelihood against training epochs

Figure 4.14: Results from models trained on the TwrBsMyt_ST_DEL load channel in the Aero dataset

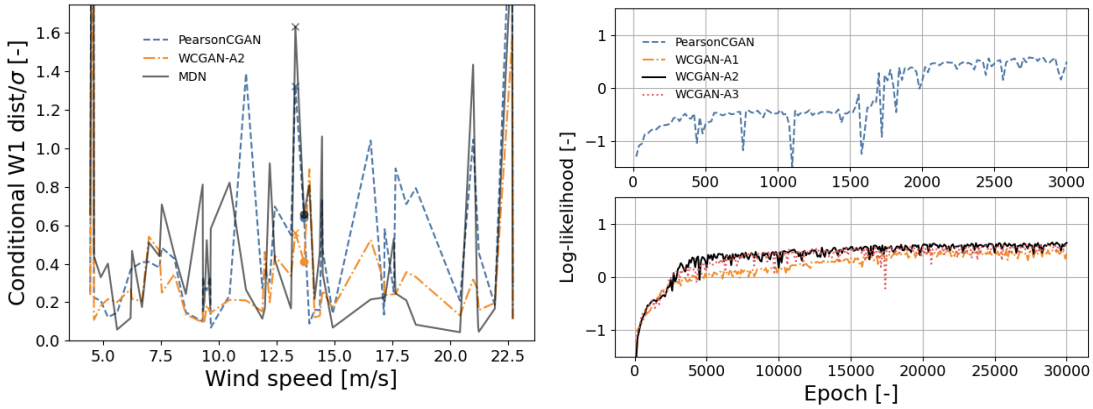


(a) Low wind speed

(b) Medium wind speed

(c) High wind speed

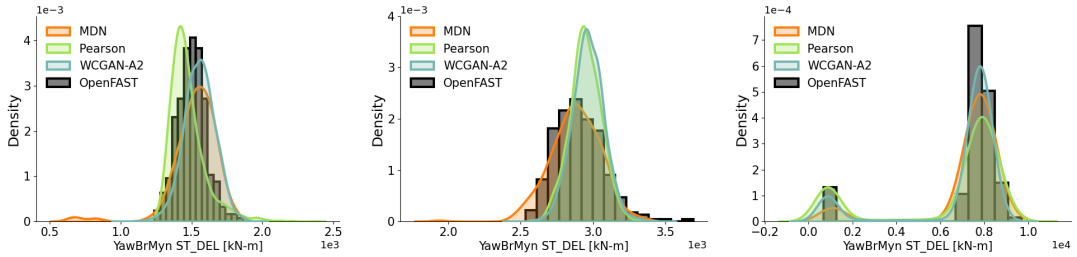
Figure 4.15: Predicted and reference (OpenFAST) conditional pdf for the TwrBsMyt_ST_DEL load channel in the Aero dataset



(a) Conditional 1-W distances for different wind speeds

(b) Validation log-likelihood against training epochs

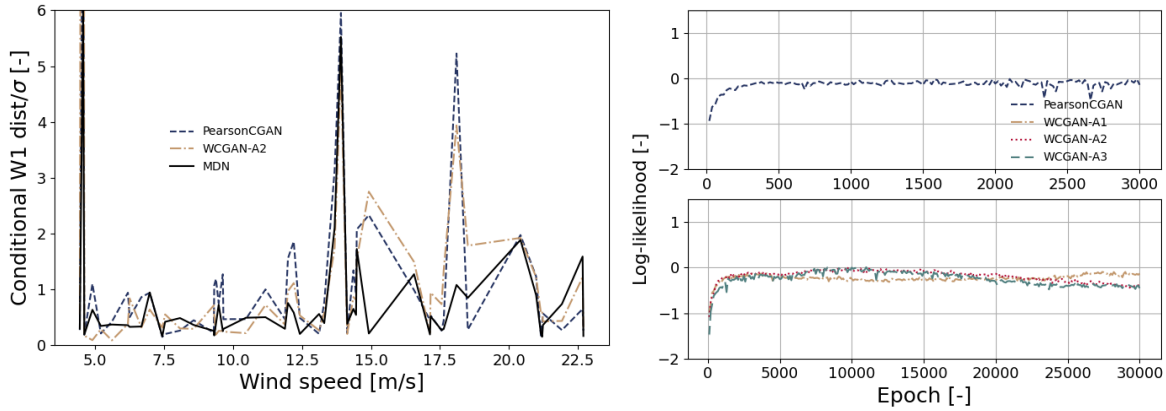
Figure 4.16: Results from models trained on the YawBrMyn_ST_DEL load channel in the Aero dataset



(a) Low wind speed **(b)** Medium wind speed **(c)** High wind speed

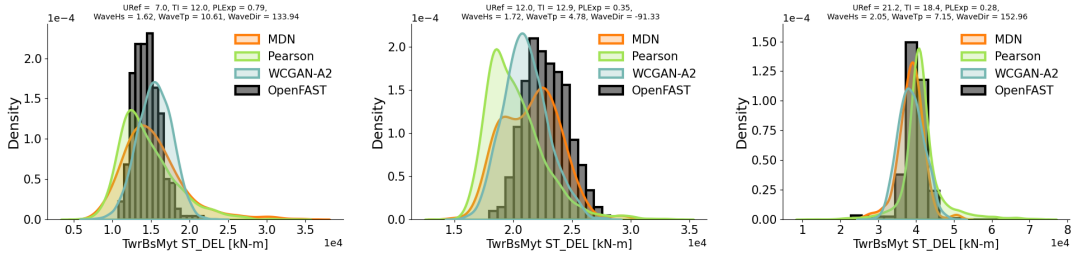
Figure 4.17: Predicted and reference (OpenFAST) conditional pdf for the YawBrMyn_ST_DEL load channel in the Aero dataset

Aerohydro



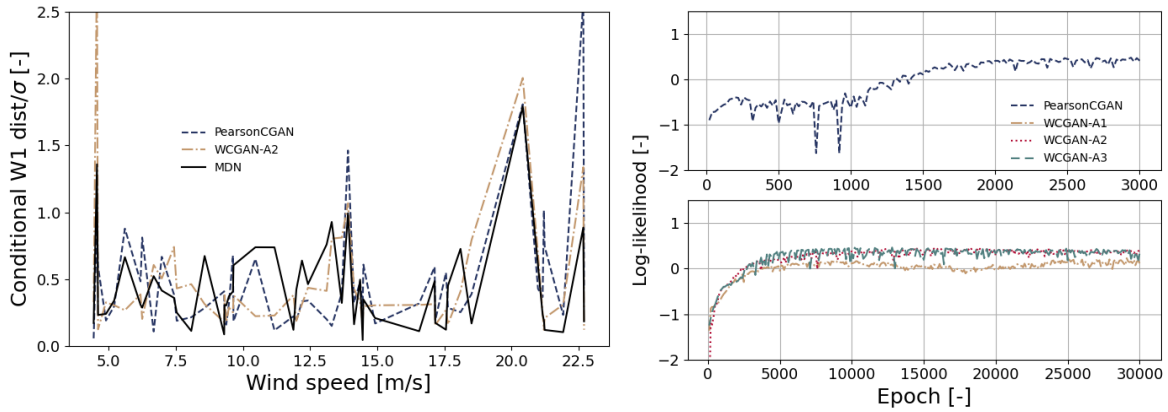
(a) Conditional 1-W distances for different wind speeds **(b)** Validation log-likelihood against training epochs

Figure 4.18: Results from models trained on the TwrBsMyt_ST_DEL load channel in the Aerohydro dataset



(a) Low wind speed **(b)** Medium wind speed **(c)** High wind speed

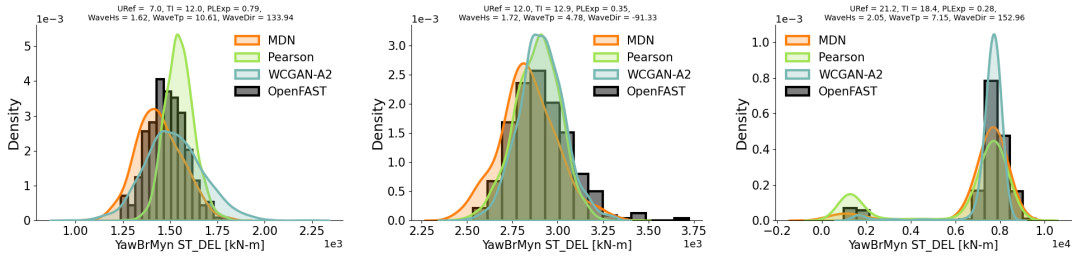
Figure 4.19: Predicted and reference (OpenFAST) conditional pdf for the TwrBsMyt_ST_DEL load channel in the Aerohydro dataset



(a) Conditional 1-W distances for different wind speeds

(b) Validation log-likelihood against training epochs

Figure 4.20: Results from models trained on the YawBrMyn_ST_DEL load channel in the Aerohydro dataset



(a) Low wind speed

(b) Medium wind speed

(c) High wind speed

Figure 4.21: Predicted and reference (OpenFAST) conditional pdf for the YawBrMyn_ST_DEL load channel in the Aerohydro dataset

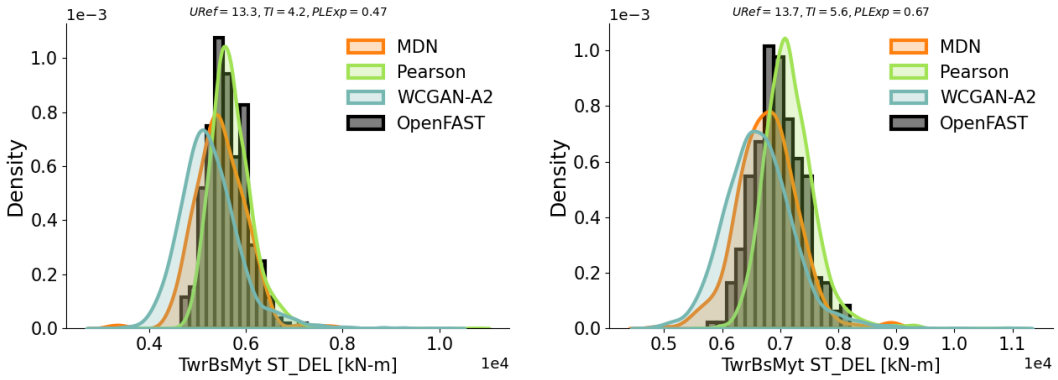
4.4.3. Discussion

Both Pearson χ^2 and WCGANs managed to achieve similar log-likelihoods. In the case of the WCGANs, usage of the A1 architecture gives slightly worse log-likelihoods compared to the other two architectures for all load channels. Looking at the conditional 1-W distances, similar to the MDNs, both models do not manage to capture the conditional distributions at the test points with low and high wind speed conditions for both the Aero and Aerohydro set. One possible reason for the poor performance in both wind conditions is a large change in conditional distributions with increasing wind speeds, which the models may not accurately capture. At low wind speeds, there is potentially some resonance due to the controller, which can cause an unexpected wind turbine response. However, these regions are not the most critical since fatigue is driven by larger turbulent disturbances [55].

Besides the test points at low and high wind speeds, the predicted conditional 1-W distances depend on both the model and load channel. For instance, in the Aero dataset, the WCGAN has the worst distance for the TwrBsMyt_ST_DEL channel while the opposite is true for the test points marked with a cross and circle in Figure 4.14 and 4.16. There is no clear explanation for this, although it highlights the fact that there is no single best model for all load channels.

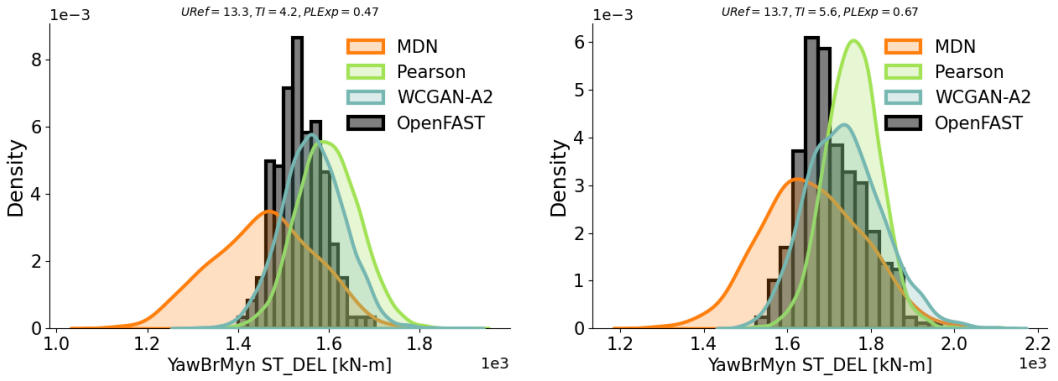
For some of the load channels investigated in both datasets, the log-likelihood history of the Pearson χ^2 CGAN has spikes, corresponding to large spikes in the generator and discriminator losses. The spikes in the loss function caused some setbacks in the training, but the Pearson χ^2 CGAN managed to recover from it. Looking at the KDE plots, the Pearson χ^2 CGAN manages to capture some of the conditional distributions in specific load channels compared to both WCGANs and MDNs such as in Figure C.6, but produces worse results in other load channels e.g. Figure C.3 and C.4. In comparison, the WCGAN experiences more training stability while producing similar results.

Looking at both datasets, both types of CGANs do not appear to perform better than the MDN baseline overall. In some cases, the MDN baseline outperforms both CGAN models e.g. Figure 4.16. This would indicate that the actual conditional distributions are best modelled by MDNs, which explains



(a) Test point 1 **(b)** Test point 2

Figure 4.22: Predicted and reference (OpenFAST) conditional pdf at two test points for the TwrBsMyt_ST_DEL load channel in the Aero dataset. Test point 1 and 2 are also marked by a cross and circle respectively in Figures 4.14a.



(a) Test point 1 **(b)** Test point 2

Figure 4.23: Predicted and reference (OpenFAST) conditional pdf at two test points for the YawBrMyn_ST_DEL load channel in the Aero dataset. Test point 1 and 2 are also marked by a cross and circle respectively in Figures 4.16a.

its superior performance. The shortcomings of both CGANs are, however, most likely related to the training process rather than the implicit generative approach, given that the models only perform worse in some areas of the feature space and can achieve similar conditional 1-W distances as the MDNs in other areas.

5
Discussion

This chapter contains an overview of the work done in this thesis. The results of each experiment have been discussed in the relevant sections in Chapter 4. A general discussion of the results is presented below.

5.1. Results

Various experiments were performed to assess the suitability of the CGAN model for low-dimensional probabilistic regression with the purpose of applying it to the probabilistic regression of wind turbine loads. Multiple CGAN models exist, the main difference being the loss function used. These models are introduced in Section 2.2, namely the f-GAN and WCGAN models. The f-GANs were shown to be an extension of the standard GAN model first introduced by Goodfellow et al. [18]. In the original formulation and further elaborated on by Arjovsky et al. [43], it was noted that the standard GAN formulation has its limitations in the training process. For this thesis, the CGANs performed well on the heteroscedastic and 1D-GM datasets. However, it did not perform as well on more complicated datasets, including real-world datasets. Thus, while the standard CGAN model could serve as a starting point when applying CGANs to other applications, it is most likely that other CGAN variants will outperform it.

Using the f-GAN framework, other types of CGANs that minimise different f-divergences could be formulated. As mentioned, the only difference between the f-GANs lies in the loss functions, leading to straightforward implementations. Some of the f-GANs specified in Nowozin et al. [20] were tested on datasets in this thesis, namely the KLCGAN, RKLCGAN, and Pearson χ^2 CGAN alongside the standard CGANs. These f-GANs have shown varying performances in this thesis, depending on the dataset on which the models were trained. Across all f-GANs, there were spikes in the loss functions during training, especially on the real-world datasets. This could be due to the nature of the CGAN training dynamics between the generator and discriminator and the type of loss function used. These spikes would lead to setbacks in training, although the f-GAN could recover from it. Nowozin et al. [20] suggests that gradient clipping could be helpful in training of the f-GANs to mitigate these issues.

All f-GAN models tested on the heteroscedastic dataset have shown equivalent performance, with the generated samples being mostly indistinguishable from the real dataset. However, on the 1D-GM and wmix6 datasets, the RKLCGAN was unstable in training, with mode dropping observed on the 1D-GM dataset. This was attributed to the high outputs of the discriminator combined with its loss function. This suggests that the RKLCGAN should not be used in its current form for CGAN training. While the KLCGAN did not exhibit similar problems, its loss function is similar to that of the RKLCGAN. Thus, the KLCGAN could also be subject to similar issues on other datasets, although this was not observed in this thesis. The KLCGAN however did not manage to perform as well as other models for the real-world datasets to be considered for model selection.

As mentioned above, the training of f-GANs corresponds to the minimisation of its respective f-divergence. For instance, the KL and RKL are divergences that models can use to match the data distribution [35]. In the case of the KL divergence, minimisation of the KL divergence leads to maximisation of the log-likelihood. While the KLCGAN did give the best log-likelihood on the

heteroscedastic and wmix6 dataset, it did not manage to do the same for the 1D-GM dataset. This suggests that the underlying f-divergence being minimised should be one of many factors when considering the performances of CGANs. Based on all the experiments performed, the Pearson χ^2 CGAN appears to have the most stable performance. Why this is the case is unclear and worth investigating why.

Besides f-GANs, the WCGAN was also investigated. The WCGAN is based on the minimisation of the Wasserstein-1 distance, and it has been shown to reduce mode collapse issues and introduce more stability in training compared to the standard GANs. When trained on the synthetic datasets, the WCGAN models' log-likelihoods are slightly lower compared to other models, even though the conditional 1-W distances produced by the WCGAN are similar. The samples produced by the WCGAN models have small but noticeable differences from the f-GANs, although this is attributed to the network architectures being used.

Finally, the CGANs were evaluated on their ability to model wind turbine load statistics. Both aerodynamic and aero-hydrodynamic loading were considered. The Pearson χ^2 CGAN and WCGAN were chosen for this experiment and compared with the results from an MDN model. The results indicate that both models face similar difficulties as the MDNs in predicting certain areas of the feature space. The load channel being predicted also plays a important role in the performance of both CGAN models. Overall, both models show comparable performance with the MDNs, demonstrating its potential.

5.2. Method

5.2.1. Datasets

The synthetic datasets in this thesis test the CGAN's ability to model heteroscedasticity and multimodality. While they have shown to be largely successful, these datasets focus mainly on generating a one-dimensional $\mathbf{y}$ given an m-dimensional $\mathbf{x}$. Since CGANs are typically used in high-dimensional problems such as image generation. One could expect CGANs to perform better in regression problems where $\mathbf{y} \in \mathbb{R}^p, p > 1$. In the context of real-world datasets, this would mean predicting multiple load channels for a given averaged wind condition $\mathbf{x}$. More research has to be done to determine if this is a feasible approach.

5.2.2. Training

In this thesis, the hyperparameters were tuned via a grid search over different values. This has been done for each model to ensure that no model suffers badly due to poor hyperparameter choices. That said, the space of hyperparameters to tune is large, some of which are model-specific such as the gradient penalty coefficient in WCGANs. Exhaustively searching this space is infeasible; hence, choices must be made about which hyperparameters to focus on. These choices are made based on experiments done in existing literature together with initial experiments. The learning rate and the gradient penalty coefficient were identified as important hyperparameters to tune. In existing literature, usages of different optimisers have also led to different results, depending on the nature of the problem. The choice of optimisers and the specific hyperparameters were not investigated in detail for this thesis. Doing so could lead to improved sample quality and faster convergence.

In addition, the number of critic training iterations per generator iteration is another hyperparameter that was not explored in detail for the WCGANs. The idea behind having multiple critic training iterations is that a well-trained critic would be able to provide more information for the generator training without worrying about vanishing gradients, thus improving its sample quality as the number of epochs increases. In initial experiments, it was found that 5 critic training iterations were sufficient; however, this could change subject to additional hyperparameters.

Lastly, the choice of network architectures explored in this thesis is also limited to basic feedforward architectures or its variants. Future work could explore other types of architectures that have shown to be successful in other work, such as convolutional neural networks (CNN) [58]. While CNNs are not applicable to the datasets investigated in this thesis, GANs that use CNNs have been successfully applied to the generation of time series data. One possibility of future work is to generate the time history of the wind turbine loads instead, from which the statistics could be modelled.

5.2.3. Evaluation

As mentioned, the CGAN is an example of an implicit generative model. Unlike explicit generative models, there is no expressible log-likelihood function. Hence, an approximation has to be performed. In this thesis, KDE has been used to assess the models as well as for model selection. This approach has been shown to be problematic for high-dimensional $\mathbf{y}$ by Theis et al. [50]. As the focus of this thesis is on low-dimensional $\mathbf{y}$, this was deemed to be fine. Two parameters are important for getting good log-likelihood estimates via KDE: the scale parameter h and the number of samples drawn from the model. h is crucial for accurate log-likelihood estimates; too high a value can lead to log-likelihoods far from the actual value [25]. In literature where KDE estimates are used [18, 20], h is determined using validation data, which is also done in this thesis.

The number of samples used for the KDE estimate also has an impact on the approximation, especially in the conditional case. Intuitively, one would expect this to be true since the KDE estimate involves averaging. However, it has also been noted by Theis et al. [50] that for a fixed kernel scale, the improvements from drawing more samples saturate. Hence, drawing an arbitrarily large number of samples for accurate log-likelihood estimates is not an option. There is also the computational aspect to consider. In the unconditional case, a large number of samples can be drawn easily. For the conditional case, each x^i in $\mathbf{x}$ induces its own conditional distribution, which in turn has to be estimated via KDE. This means generating new samples for every validation/test point, requiring large amounts of time and memory if many samples and test points are required. In Aggarwal et al. [23], 100 samples were used for the KDE-estimation of conditional distributions, while Oskarsson [25] used 1000. For this thesis, taking into account the available computational resources, 200 samples were used. For future experiments, one could try to identify a suitable value from a given range to determine when improvements start to saturate.

Alongside the KDE estimates, the conditional Wasserstein(1-W) distance, normalised by the standard deviation of the data distribution, is used. This was used instead of the more conventional 1-W distance since the specific conditional distributions were of interest. The Wasserstein distance is a distance metric that aims to quantify the amount of work required to transform one distribution into another. As such, comparing the conditional 1-W distance provides a secondary metric alongside the log-likelihood to evaluate the performance of each model. One advantage of the 1-W distance over the KDE estimate of the log-likelihood is that it is not affected by any mismatch in support between the model distribution and data distribution. These are areas where the output has a high probability under the actual distribution but zero to low probability under the model distribution (which is usually the case in the initial training process). In the case of the KDE estimate, the log-likelihood estimate would be $-\infty$ [25], and measures such as setting a lower limit on the kernel scale have to be employed. The conditional 1-W distances, combined with the visualisations via KDE plots, have shown to be more informative of the accuracy of the model distributions, thus indicating its usefulness for future work.

6

Conclusions and recommendations

6.1. Conclusions

The work in Chapter 4 has demonstrated the ability of CGANs to capture complex distributions with varying levels of accuracy between different models. Regarding its practical application, the potential for predicting wind turbine load statistics exists. However, there is no clear advantage of using CGANs over other probabilistic regression methods such as MDNs.

The usage of CGANs for probabilistic regression problems with the goal of applying it to wind turbine load statistics has been explored in this thesis. The research objective in this thesis was:

> **Should GANs be used as a probabilistic surrogate model for the load emulation of wind turbines?**

A series of subquestions were crafted in order to guide the exploration process. Through the experiments conducted in this thesis, answers to each research question are found, which are discussed below.

How well do CGANs work for low dimensional problems in general, especially where the dimensionality of the input is much larger than the output?
CGANs have shown that they can learn various types of low-dimensional complex distributions through various toy datasets. The CGANs can model properties such as multimodality and heteroscedasticity, although the individual performance is dependent on CGAN model. Each CGAN performs similarly on simple datasets; the differences between each model appear when training on more complicated datasets such as real-world datasets. It should be noted that the problems investigated in this thesis involved datasets that have multidimensional $\mathbf{x}$ and one-dimensional $\mathbf{y}$. More research with datasets with different dimensionality of $\mathbf{x}$ and $\mathbf{y}$ should be done for future work.

How does the choice of training objectives and neural network architectures affect the training process of CGANs to approximate distributions?
Among all the CGAN training objectives used, the Pearson χ^2 CGANs and the WCGANs have stood out as the best-performing CGANs using the conditional 1-W distances and the log-likelihood estimation as metrics. These two models also have a relatively stable training process compared to the rest of the models.

The noise-injection architecture was used alongside the basic feedforward architecture in the experiments. The noise-injection architecture involves hidden layers that depend on both noise and the input from previous layers. On the WCGAN training objective, the noise-injection and feedforward neural network produces results with minor differences. This suggests that the architecture choice is of little importance, although more architectures should be investigated in the future to confirm this. .

How well do CGANs compare to alternative probabilistic models like Mixture Density Networks for load emulation?
CGANs have achieved varying results on real-world datasets, depending on the dataset type and the load channel investigated in that dataset. Overall, the CGAN performed on par with the MDNs;

MDNs gave more consistent conditional 1-W distances on the test datasets, which could be due to the conditional distributions being accurately modelled with Gaussians. More work would be needed to make CGANs perform better than MDNs on the real-world datasets in question.

What are the challenges faced when using a CGAN model for load emulation?
As with machine learning models, hyperparameter tuning is a key challenge in training a CGAN model. While the importance of some is obvious, such as learning rates, there are other hyperparameters whose importance may be less obvious, such as the gradient penalty coefficient for WCGANs. Each model also comes with its own specific hyperparameters that may be important for low-dimensional problems. Even after hyperparameter tuning, other factors, such as network architecture/size, could impact the results. Hence, it is essential to identify potentially important factors early in the initial training process that would yield the best possible results.

6.2. Recommendations

Throughout this thesis, it was noticed that further studies need to be performed to make CGANs a reliable tool for probabilistic regression in low dimensions. Different neural network architectures centered around basic feedforward networks have been covered briefly in this thesis. However, CGANs have also shown exceptional performance in literature with other neural network architectures, such as CNNs. Subsequently, there have been CGAN applications to problems where such neural networks are suitable, such as CNNs for time series data regression. Considering more types of neural networks together with different kinds of data, such as time histories of wind turbine loads, could be a possible future study to perform. In addition, considering that MDNs work well on the real-world datasets in this thesis, one might consider using an MDN as a generator in a CGAN framework to improve the generated results further.

References

[1] A.D. Saul et al. "Chained Gaussian Processes". In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Ed. by Arthur Gretton and Christian C. Robert. Vol. 51. Proceedings of Machine Learning Research. Cadiz, Spain: PMLR, 2016, pp. 1431–1440. URL: https://proceedings.mlr.press/v51/saul16.html.

[2] C. M. Bishop. *Mixture density networks*. English. Tech. rep. Aston University, 1994.

[3] L. Schröder et al. "Wind turbine site-specific load estimation using artificial neural networks calibrated by means of high-fidelity load simulations". In: *Journal of Physics: Conference Series* 1037 (June 2018), p. 062027. DOI: 10.1088/1742-6596/1037/6/062027.

[4] N. Dimitrov. "Surrogate models for parameterized representation of wake-induced loads in wind farms". In: *Wind Energy* 22.10 (2019), pp. 1371–1389. DOI: https://doi.org/10.1002/we.2362.

[5] K. Shaler, J. Jasa, and G.E. Barter. "Efficient Loads Surrogates for Waked Turbines in an Array". In: *Journal of Physics: Conference Series* 2265 (2022), p. 032095. DOI: 10.1088/1742-6596/2265/3/032095.

[6] X Zhu and B Sudret. "Emulation of Stochastic Simulators Using Generalized Lambda Models". In: *SIAM/ASA Journal on Uncertainty Quantification* 9.4 (2021), pp. 1345–1380. DOI: 10.1137/20m1337302.

[7] R. Teixeira et al. "Analysis of the design of experiments of offshore wind turbine fatigue reliability design with Kriging surfaces". In: *Procedia Structural Integrity* 5 (2017), pp. 951–958.

[8] N. Dimitrov et al. "From wind to loads: wind turbine site-specific load estimation with surrogate models trained on high-fidelity load databases". In: *Wind Energy Science* 3 (Oct. 2018), pp. 767–790. DOI: 10.5194/wes-3-767-2018.

[9] L.D. Avendaño-Valencia, I. Abdallah, and E. Chatzi. "Virtual fatigue diagnostics of wake-affected wind turbine via Gaussian Process Regression". In: *Renewable Energy* 170 (2021), pp. 539–561. ISSN: 0960-1481. DOI: https://doi.org/10.1016/j.renene.2021.02.003.

[10] G. Gasparis, A. Lio, and F. Meng. "Surrogate Models for Wind Turbine Electrical Power and Fatigue Loads in Wind Farm". In: *Energies* 13 (Dec. 2020). DOI: 10.3390/en13236360.

[11] S. Okpokparoro and S. Sriramula. "Uncertainty modeling in reliability analysis of floating wind turbine support structures". In: *Renewable Energy* 165 (2021), pp. 88–108. ISSN: 0960-1481. DOI: https://doi.org/10.1016/j.renene.2020.10.068.

[12] R.M.M. Slot et al. "Surrogate model uncertainty in wind turbine reliability assessment". In: *Renewable Energy* 151 (2020), pp. 1150–1162. ISSN: 0960-1481. DOI: https://doi.org/10.1016/j.renene.2019.11.101.

[13] D. Singh et al. "Probabilistic surrogate modeling of offshore wind-turbine loads with chained Gaussian processes". In: *Journal of Physics: Conference Series* 2265.3 (2022), p. 032070. DOI: 10.1088/1742-6596/2265/3/032070.

[14] M. Arjovsky, S. Chintala, and L. Bottou. *Wasserstein GAN*. 2017. DOI: 10.48550/ARXIV.1701.07875.

[15] C. Mylonas, I. Abdallah, and E. Chatzi. "Conditional variational autoencoders for probabilistic wind turbine blade fatigue estimation using Supervisory, Control, and Data Acquisition data". In: *Wind Energy* 24 (Oct. 2021). DOI: 10.1002/we.2621.

[16] Y.R Wang et al. "Planetary gearbox fault feature learning using conditional variational neural networks under noise environment". In: *Knowledge-Based Systems* 163 (2019), pp. 438–449. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2018.09.005.

[17] A.S. Yoon et al. *Semi-supervised Learning with Deep Generative Models for Asset Failure Prediction*. 2017. arXiv: 1709.00845 [cs.LG].

[18] I. Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*. Vol. 27. 2014, pp. 2672–2680.

[19] M. Mirza and S. Osindero. *Conditional Generative Adversarial Nets*. DOI: 10.48550/ARXIV.1411.1784.

[20] S. Nowozin, B. Cseke, and R. Tomioka. “f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization”. In: *Advances in Neural Information Processing Systems*. Vol. 29. 2016. DOI: 10.48550/arXiv.1606.00709.

[21] T. Minka. *Divergence Measures and Message Passing*. Tech. rep. MSR-TR-2005-173. 2005, p. 17.

[22] B.K. Sriperumbudur et al. *On integral probability metrics, ϕ-divergences and binary classification*. 2009. DOI: 10.48550/ARXIV.0901.2698.

[23] K. Aggarwal et al. *Benchmarking Regression Methods: A comparison with CGAN*. 2019. DOI: 10.48550/ARXIV.1905.12868.

[24] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[25] J. Oskarsson. “Probabilistic Regression using Conditional Generative Adversarial Networks”. MA thesis. Linköping University, The Division of Statistics and Machine Learning, 2020, p. 111.

[26] T.R.F. Phillips et al. “Multi-Output Regression with Generative Adversarial Networks (MOR-GANs)”. In: *Applied Sciences* 12.18 (2022). ISSN: 2076-3417. DOI: 10.3390/app12189209.

[27] D.J. Gagne et al. “Machine Learning for Stochastic Parameterization: Generative Adversarial Networks in the Lorenz ’96 Model”. In: *Journal of Advances in Modelling Earth Systems* 12.3 (2020). DOI: 10.1029/2019ms001896.

[28] P. Zhixin et al. “Data-Driven EV Load Profiles Generation Using a Variational Auto-Encoder”. In: *Energies* 12.5 (2019). ISSN: 1996-1073. DOI: 10.3390/en12050849.

[29] C. Jiang et al. “Scenario Generation for Wind Power Using Improved Generative Adversarial Networks”. In: *IEEE Access* 6 (2018), pp. 62193–62203. DOI: 10.1109/ACCESS.2018.2875936.

[30] Y. Zhang et al. “Typical wind power scenario generation for multiple wind farms using conditional improved Wasserstein generative adversarial network”. In: *International Journal of Electrical Power & Energy Systems* 114 (2020), p. 105388. ISSN: 0142-0615. DOI: 10.1016/j.ijepes.2019.105388.

[31] D. Singh. *Training and validation datasets for training probabilistic machine learning models on NREL’s 10-MW reference wind turbine*. 2023. DOI: 10.4121/21939995.v1. URL: https://data.4tu.nl/articles/dataset/Training_and_validation_datasets_for_training_probabilistic_machine_learning_models_on_NREL_s_10-MW_reference_wind_turbine/21939995/1.

[32] I.J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[33] A.L. Maas, A.Y. Hannun, and A.Y. Ng. “Rectifier Nonlinearities Improve Neural Network Acoustic Models”. In: *Proceedings of the International Conference on Machine Learning*. Atlanta, Georgia, 2013.

[34] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2016.

[35] C.M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[36] J. Duchi, E. Hazan, and Y. Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In: *Journal of Machine Learning Research* 12.61 (2011), pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html.

[37] T. Tieleman and G. Hinton. “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[38] D.P. Kingma and J Ba. *Adam: A Method for Stochastic Optimization*. 2017.

[39] S. Mohamed and B. Lakshminarayanan. *Learning in Implicit Generative Models*. 2017.

[40] R. Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2021. arXiv: 2112.10752 [cs.CV].

[41] A. Brock, J. Donahue, and K. Simonyan. *Large Scale GAN Training for High Fidelity Natural Image Synthesis*. 2019. arXiv: 1809.11096 [cs.LG].

[42] J. Sun et al. "Generative adversarial networks with mixture of t-distributions noise for diverse image generation". In: *Neural Networks* 122 (2020), pp. 374–381. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2019.11.003.

[43] M. Arjovsky and L. Bottou. *Towards Principled Methods for Training Generative Adversarial Networks*. 2017. DOI: 10.48550/ARXIV.1701.04862.

[44] N. Kodali et al. *On Convergence and Stability of GANs*. 2017. arXiv: 1705.07215 [cs.AI].

[45] J.B. Hiriart-Urruty and C. Lemaréchal. *Fundamentals of Convex Analysis*. Springer, 2012.

[46] V. Cédric. *Optimal transport: Old and New*. Grundlehren der mathematischen Wissenschaften. Springer, 2009.

[47] I. Gulrajani et al. *Improved Training of Wasserstein GANs*. 2017. arXiv: 1704.00028 [cs.LG].

[48] H. Petzka, A. Fischer, and D. Lukovnikov. "On the regularization of Wasserstein GANs". In: Sept. 2017.

[49] M. Heusel et al. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6629–6640. ISBN: 9781510860964.

[50] L. Theis, A. van den Oord, and M. Bethge. "A note on the evaluation of generative models". In: *International Conference on Learning Representations*. 2016. URL: http://arxiv.org/abs/1511.01844.

[51] National Renewable Energy Laboratory (NREL). *OpenFAST documentation*. Version 3.2.0. 2022.

[52] N.D. Kelley and B.J. Jonkman. *Overview of the TurbSim Stochastic Inflow Turbulence Simulator*. Tech. rep. NREL/TP-5000-41137. National Renewable Energy Laboratory, 2007.

[53] S.. Kelley, E. Brandlard, and A. Platt. *OLAF User's Guide and Theory Manual*. Tech. rep. NREL/TP-5000-75959. National Renewable Energy Laboratory, 2020.

[54] P. Bortolotti et al. *EA Wind Task 37 on Systems Engineering in Wind Energy WP2.1 Reference Wind Turbines*. Tech. rep. NREL/TP-5000-73492. Colorado: National Renewable Energy Laboratory, 2019.

[55] D. Singh, R. Dwight, and A. Vire. "Probabilistic Surrogate Modeling of Site-Specific Loads on Onshore and Offshore Wind Turbines Using Mixture Density Networks". In: *SSRN Electronic Journal* (Feb. 2023). DOI: 10.2139/ssrn.4354826.

[56] F. Vigara et al. *Design Basis*. Version 2.0. 2020. DOI: 10.5281/zenodo.4518828. URL: https://doi.org/10.5281/zenodo.4518828.

[57] R. Flamary et al. "POT: Python Optimal Transport". In: *Journal of Machine Learning Research* 22.78 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-451.html.

[58] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

A

Further experiments

This appendix includes additional experiments performed with the WCGAN. The purpose of these experiments is to provide some guidance for the practical application of WCGANs. The sections below describe experiments performed on the TwrBsMyt_ST_DEL channel using the WCGAN-A2 architecture, together with the results.

A.1. Different noise dimensionality

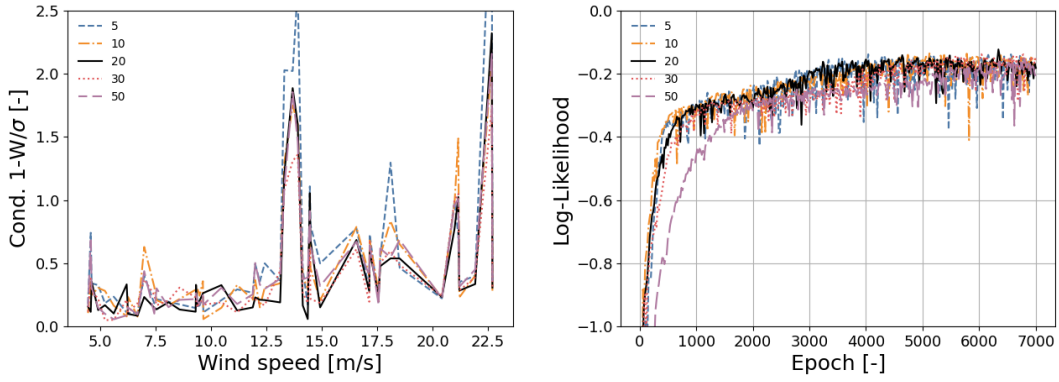
Different values of the noise dimensionality can be used for the input to the WCGAN generator. To better understand how the dimensionality affects results, different values of noise dimensionality were used in the training.

A.1.1. Experimental setup

The WCGAN was trained for each noise dimensionality in $\{5, 10, 20, 30, 50\}$. For each noise dimensionality, the WCGAN was trained for 7000 epochs with a learning rate of 0.0002 and 0.0001 for the generator and critic respectively, using the A2 architecture from Table 3.5. All other relevant training parameters are as given in Section 3.3.3. The trained models were evaluated on the likelihoods and conditional 1-W distances like the experiments in Chapter 4.

A.1.2. Results

The conditional 1-W distances and the log-likelihoods are shown in Figure A.1. KDE plots for the three test points in Table 4.6 are shown in Figure A.3. Only the KDE plots for noise dimensionalities of $\{5, 20, 30\}$ are shown for clarity.



(a) Conditional 1-W distances for different wind speeds

(b) Validation log-likelihood against training epochs

Figure A.1: Results for different noise dimensionalities

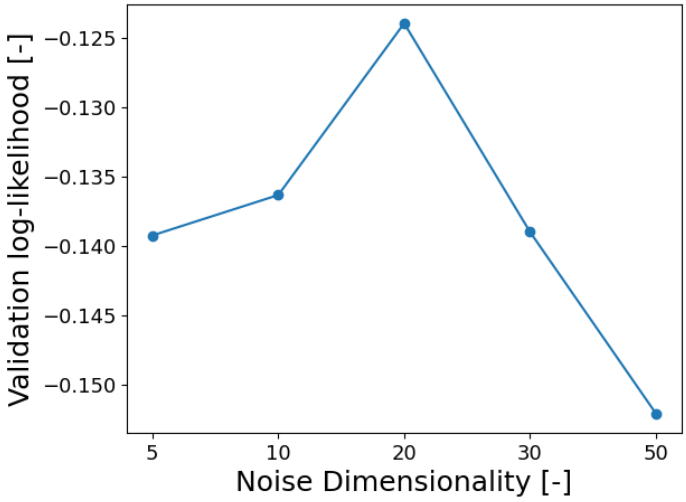
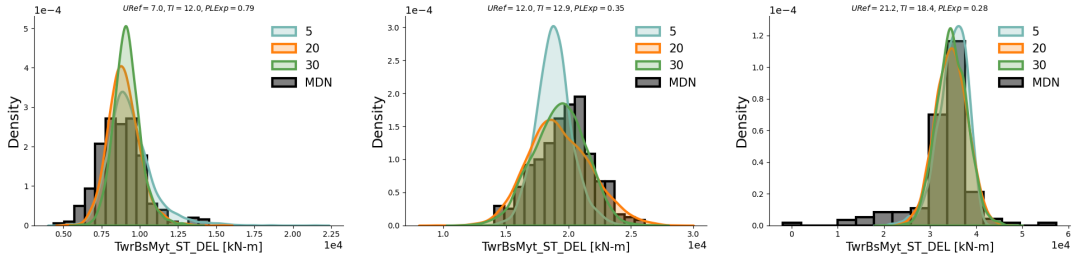


Figure A.2: Maximum validation log-likelihood obtained vs noise dimensionality



(a) Low wind speed **(b)** Medium wind speed **(c)** High wind speed

Figure A.3: Predicted and reference (MDN) conditional pdf for TwrBsMyt_ST_DEL at each test point using different noise dimensionalities

A.1.3. Discussion

The maximum log-likelihoods obtained in Figure A.2 suggest an optimal noise dimensionality value. The log-likelihood decreases going past a noise dimension of 20, but this could be due to the bandwidth value choice for the log-likelihood estimation. The KDE plots of the test points, together with the conditional 1-W plots, do suggest that better results are obtained with increasing noise dimensionality. However, the amount of improvement is likely to be dependent on the dataset. Hence, for real-world datasets where the data distribution is unknown, there appears to be no drawback to setting a high initial value for the noise dimension.

A.2. Different noise distributions

Different distributions can be used for the noise input to the WCGAN generator. To understand how this can affect the performance of WCGANs, it was trained using different noise distributions.

A.2.1. Experiment setup

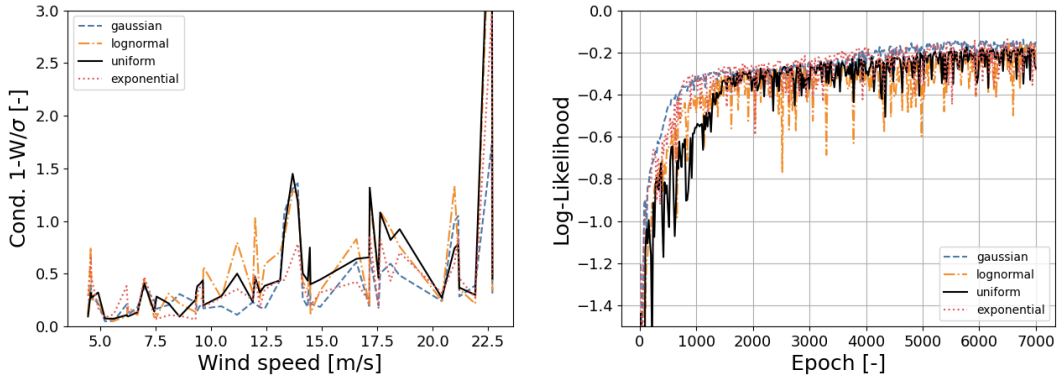
The different noise distributions investigated are:

- A standard Gaussian $\mathcal{N}(0, I)$
- A uniform distribution $\mathcal{U}[0, 1]$
- An exponential distribution with rate parameter 1
- A lognormal distribution LogNormal$(0, I)$

For each distribution, the WCGAN-A2 architecture from Table 3.5 was trained for 7000 epochs, using a learning rate of 0.0002 and 0.0001 for the generator and discriminator for each noise distribution. All other relevant training parameters are as given in Section 3.3.3.

A.2.2. Results

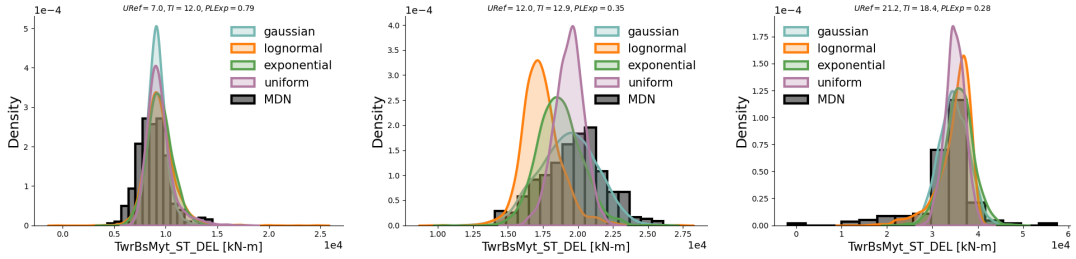
Figures A.4 and A.5 show the impact of the noise distributions.



(a) Conditional 1-W distances for different wind speeds

(b) Validation log-likelihood against training epochs

Figure A.4: Results for different noise distributions



(a) Low wind speed

(b) Medium wind speed

(c) High wind speed

Figure A.5: Predicted and reference (MDN) conditional pdf for TwrBsMyt_ST_DEL at each test point using different noise distributions

A.2.3. Discussion

The Aero_MDN dataset is generated from an MDN, which uses Gaussian mixtures to approximate the data distribution. If the noise distribution should imply something about the underlying distribution, then using a Gaussian noise distribution is expected to produce the best results. Other distributions appear to give similar results based on the conditional 1-W distances, with the exception of the uniform distribution.

For real-world datasets, where not much is known about it, usage of a normal distribution as the latent distribution could serve as an initial option. The choice of noise distributions has also been investigated in [41], showing that different noise distributions lead to different levels of performance in terms of convergence and sample quality. As such, any future work should also consider the choice of the noise distribution as a hyperparameter.

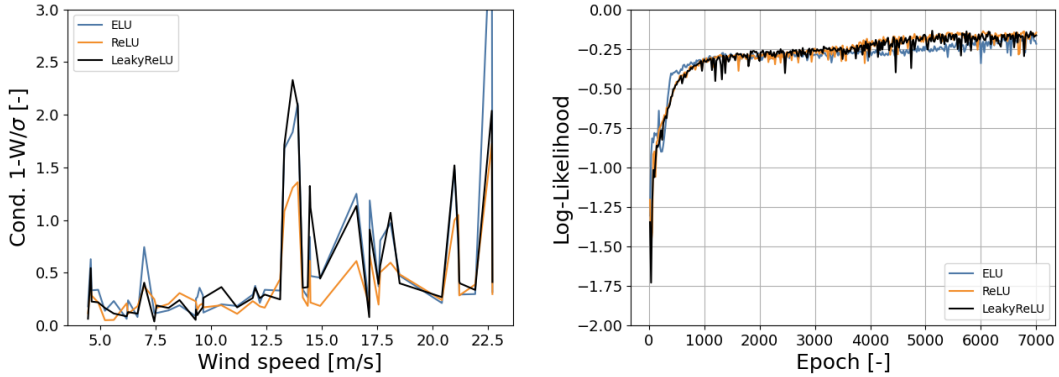
A.3. Different activation functions

This section investigates the use of different activation functions for WCGANs.

A.3.1. Experiment setup

The same experimental setup as in Section A.2 was used on the TwrBsMyt_ST_DEL load channel of the Aero dataset. The noise dimensionality was set to 30, and the ELU, ReLU, and LeakyReLU activations were used for all layers in both the generator and critic.

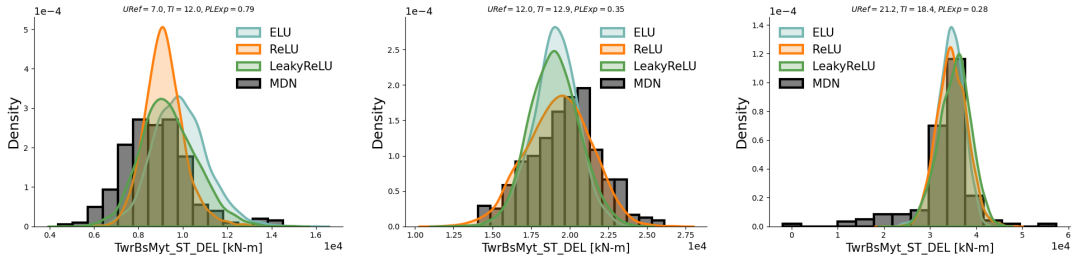
A.3.2. Results



(a) Conditional 1-W distances for different wind speeds

(b) Validation log-likelihood against training epochs

Figure A.6: Results for different activation functions



(a) Low wind speed

(b) Medium wind speed

(c) High wind speed

Figure A.7: Predicted and reference (MDN) conditional pdf for TwrBsMyt_ST_DEL at each test point using different activation functions

A.3.3. Discussion

Based on the conditional 1-W distances, the ReLU activation function stands out as the best performing activation function. With regards to the KDE visualisations, the ReLU activation function appears to have the best fit for the medium and high wind speed conditions, while the opposite is true for the low wind speed condition. As mentioned in Section 4.4, the low wind speeds are not the most critical. Thus, even though only one load channel is investigated in this section, using ReLU for the activation function should be a good starting choice for activation functions in WCGANs.

A.4. Different regularisation value for WCGAN-GP/one-sided vs two-sided

This section discusses the impact of the gradient penalty coefficients λ in WCGAN-GP, together with the use of one-sided vs two-sided penalties.

A.4.1. Experiment Setup

The same experimental setup in Section A.2 was used. Three values of λ were investigated: $\{0.02, 0.5, 2\}$, and for each value, both one-sided and two-sided gradient penalties are utilised.

A.4.2. Results

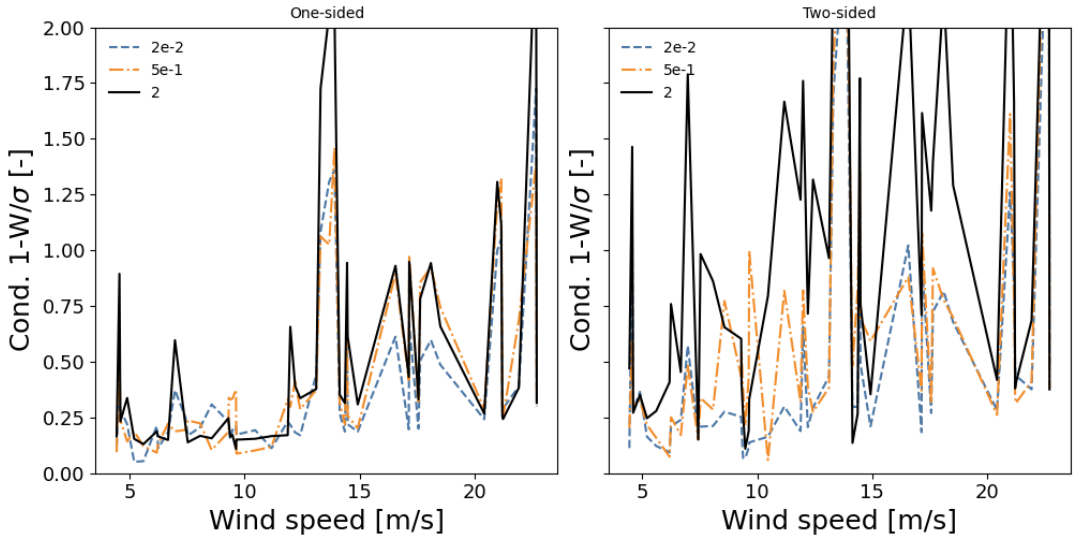


Figure A.8: Conditional 1-W distances for different wind speeds using different λ for both one-sided and two-sided gradient penalties

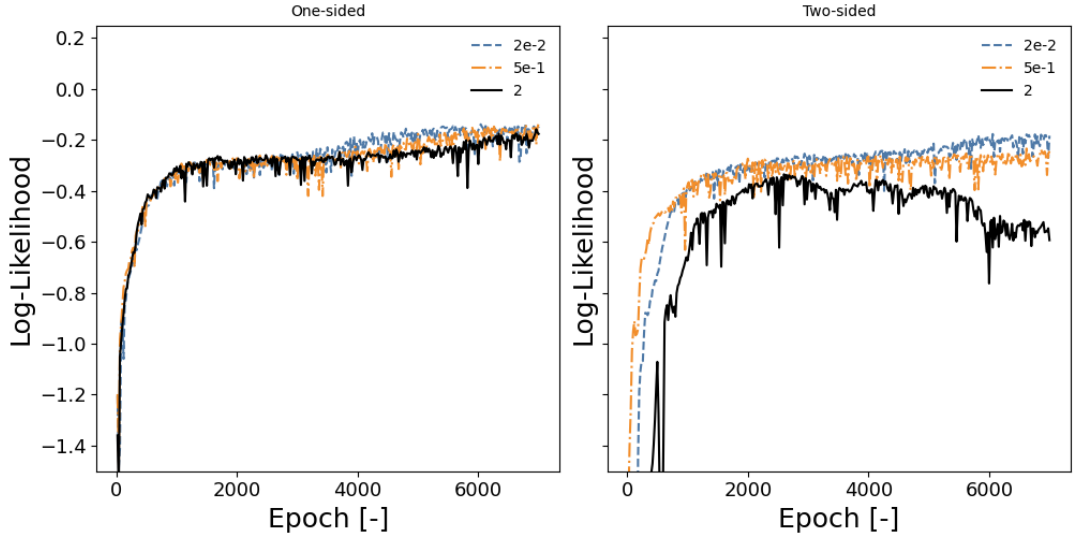


Figure A.9: Validation log-likelihood against training epochs for different λ using both one-sided and two-sided gradient penalties

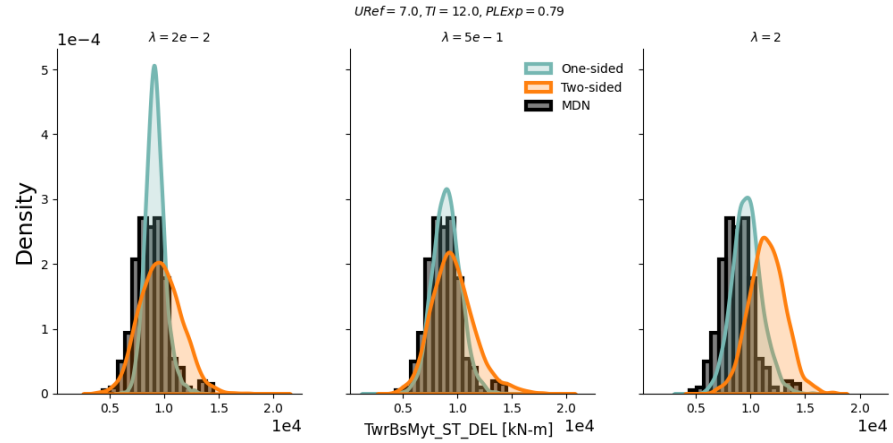


Figure A.10: KDE plots for low wind speed condition with different values of λ

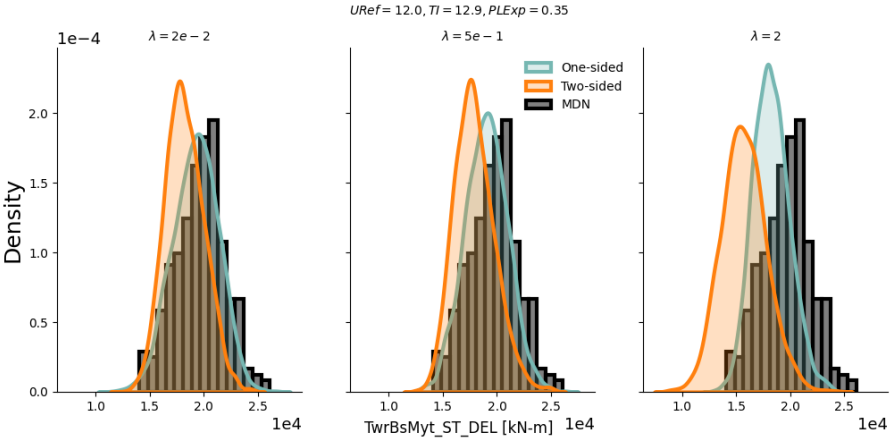


Figure A.11: KDE plots for medium wind speed condition with different values of λ

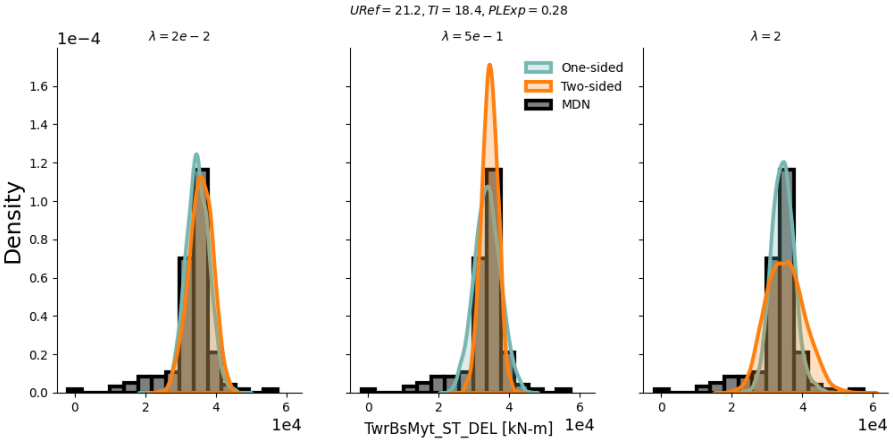


Figure A.12: KDE plots for high wind speed condition with different values of λ

A.4.3. Discussion

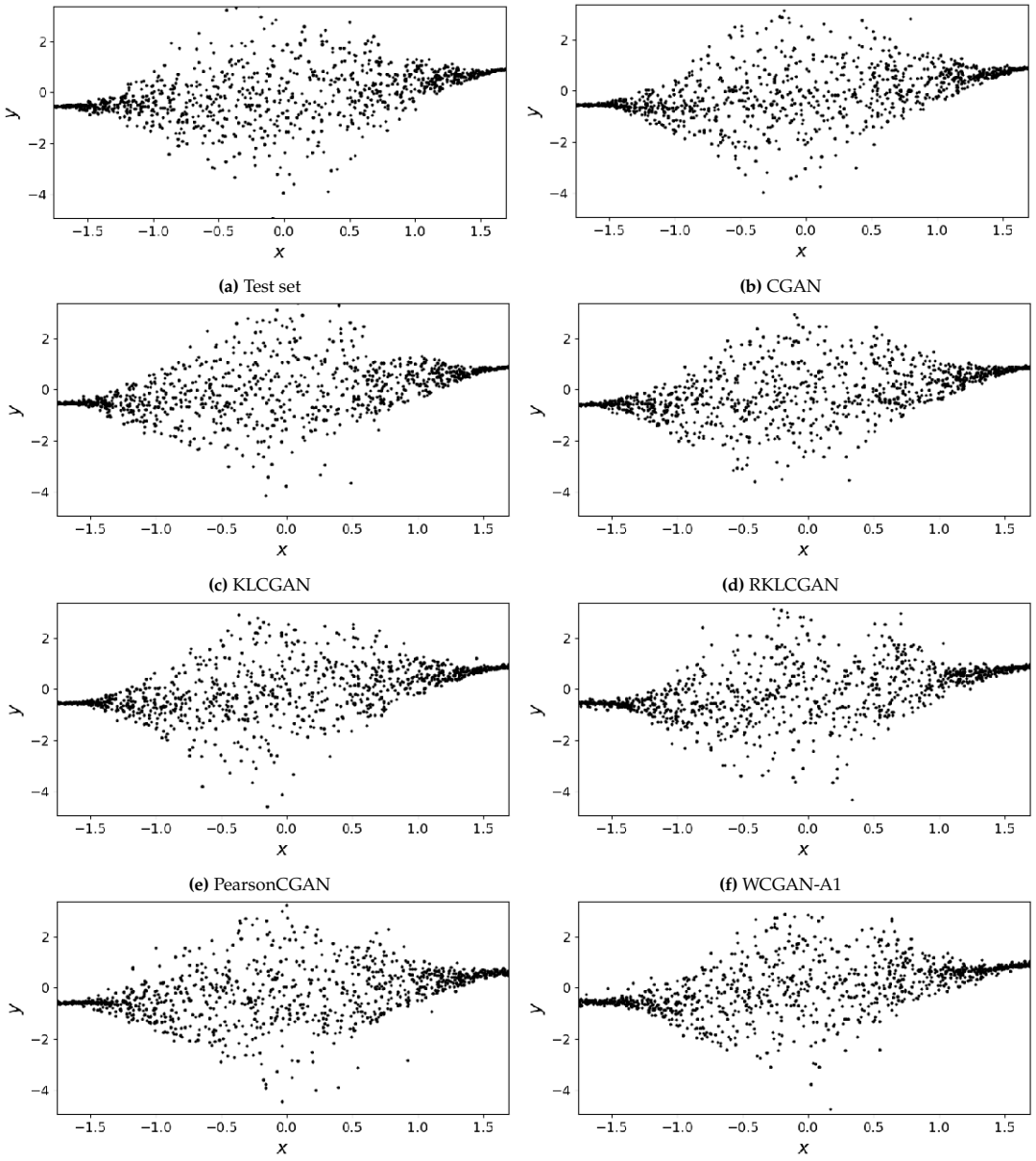
λ appears to have a huge impact on the results, especially for the two-sided penalty. Overall, the results of the log-likelihood and conditional 1-W distances appear to be more consistent between the different gradient penalties when using the one-sided penalty compared to the two-sided penalty. Using the two-sided penalty results in overfitting of the log-likelihood and worse conditional 1-W distances with

increasing values of λ. This is in line with the observation made in [48]. The KDE visualisations also show that the distributions obtained from the one-sided penalty are similar between different λ and more accurate.

B

Samples from models trained on synthetic datasets

This chapter contains scatter plots as well as KDE plots from different models used in the experiments on synthetic datasets. The test set is also plotted for each dataset for comparison.



(a) Test set

(b) CGAN

(c) KLCGAN

(d) RKLCGAN

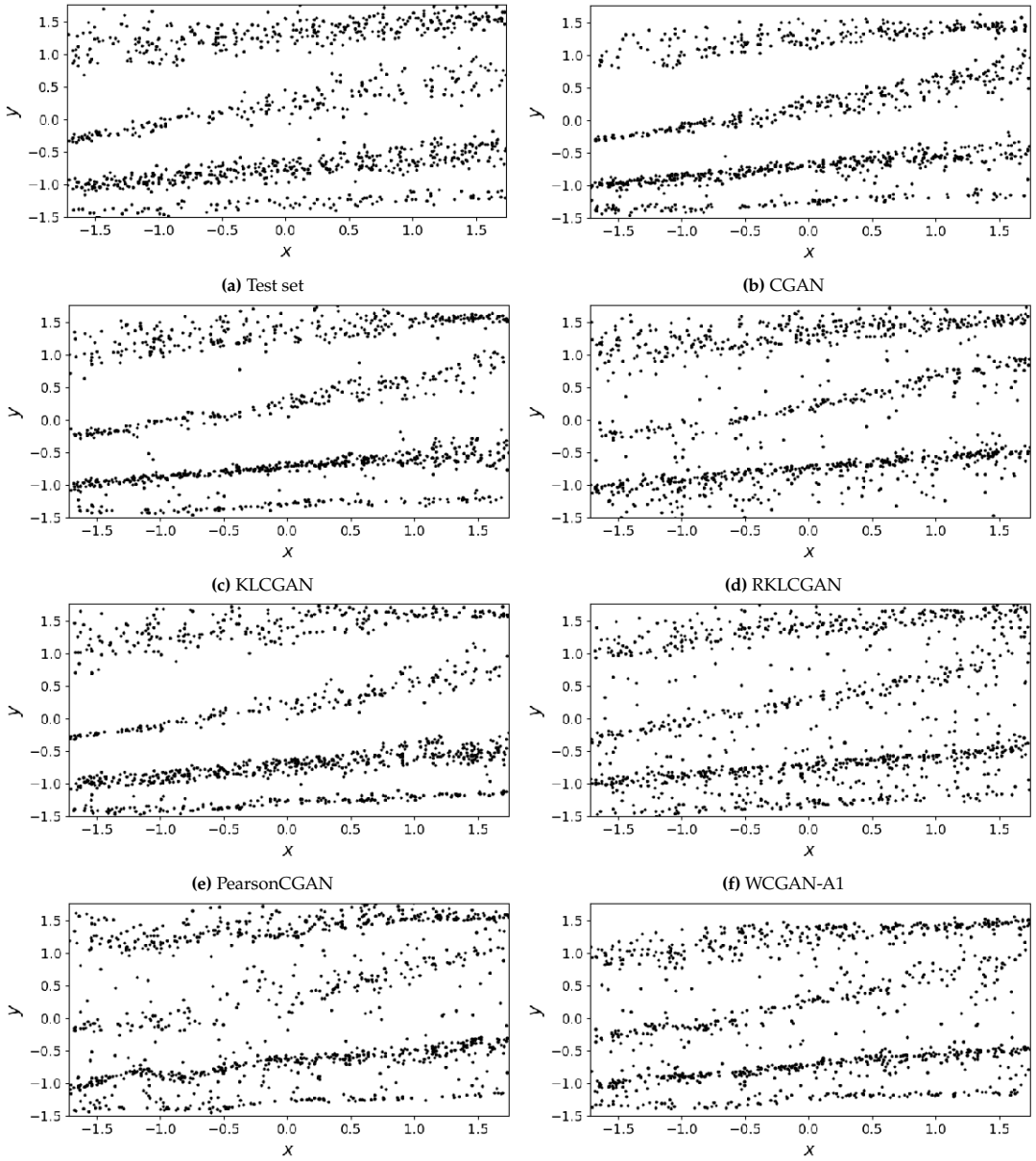
(e) PearsonCGAN

(f) WCGAN-A1

(g) WCGAN-A2

(h) WCGAN-A3

Figure B.1: Samples from models trained on the Heteroscedastic dataset



(a) Test set

(b) CGAN

(c) KLCGAN

(d) RKLCGAN

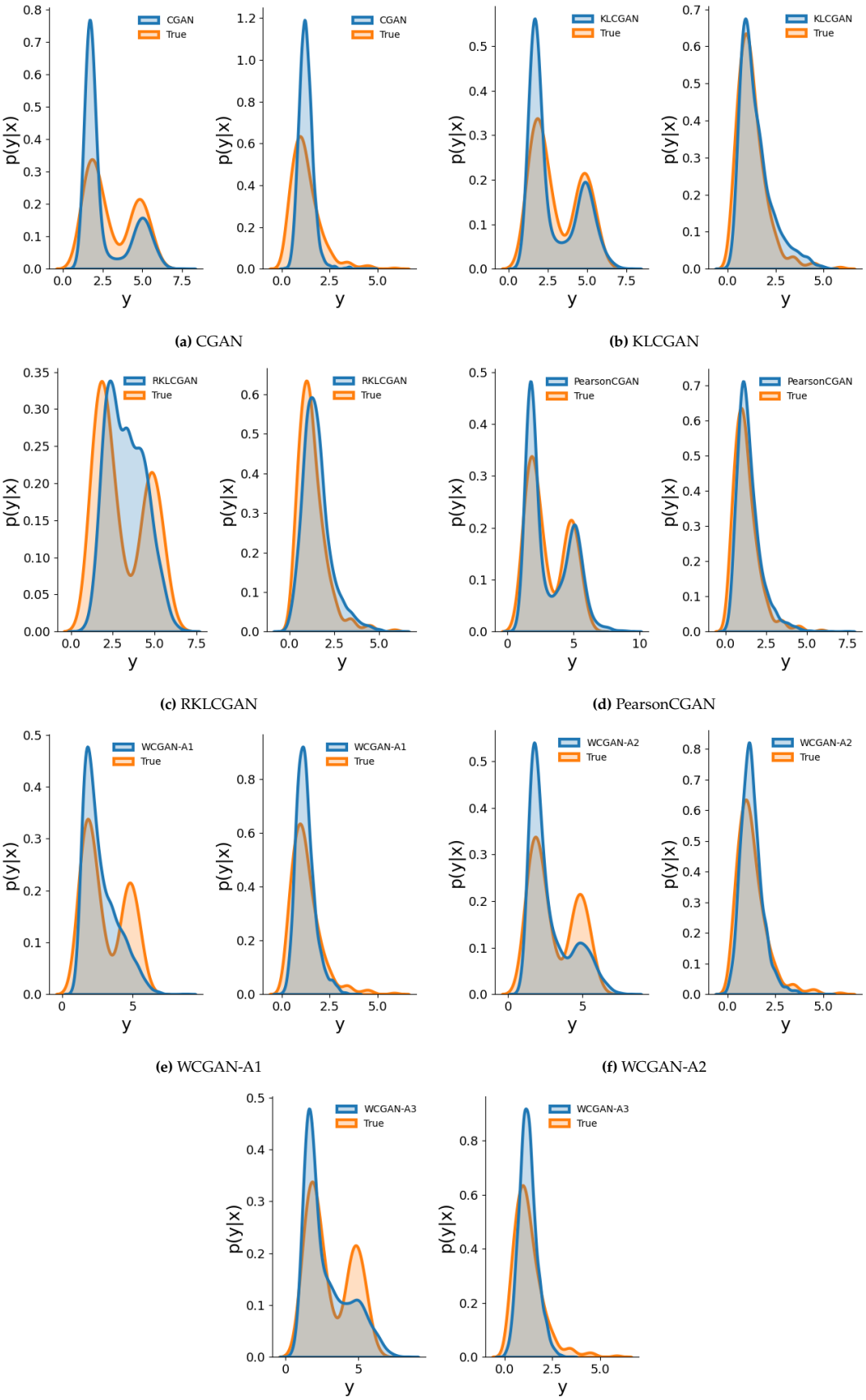
(e) PearsonCGAN

(f) WCGAN-A1

(g) WCGAN-A2

(h) WCGAN-A3

Figure B.2: Samples from models trained on the 1D-GM dataset



(a) CGAN **(b)** KLCGAN

(c) RKLCGAN **(d)** PearsonCGAN

(e) WCGAN-A1 **(f)** WCGAN-A2

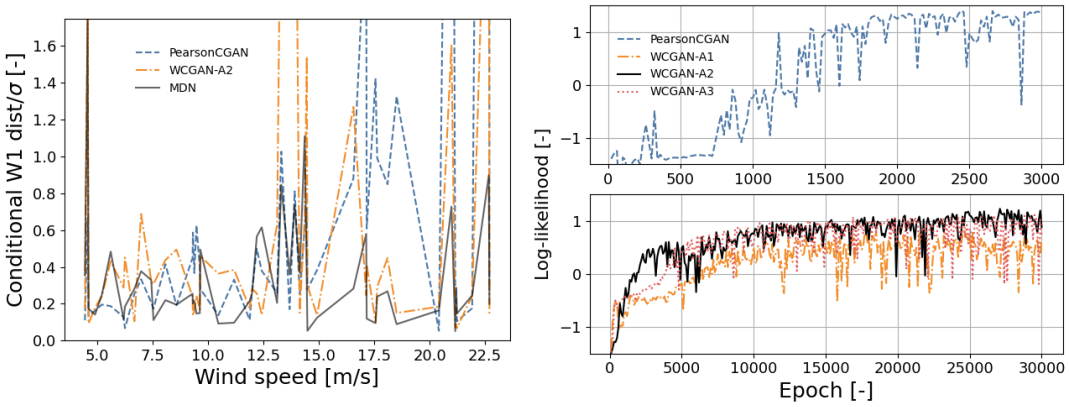
(g) WCGAN-A3

Figure B.3: KDE plots for models trained on the wmix6 dataset. Two separate conditional inputs are used for each model: $x = [0.20, 0.80, 0.93, 0.56, 0.64, 0.90]$ and $x = [0.12, 0.01, 0.57, 0.65, 0.08, 0.86]$

C

Further results for Aero dataset

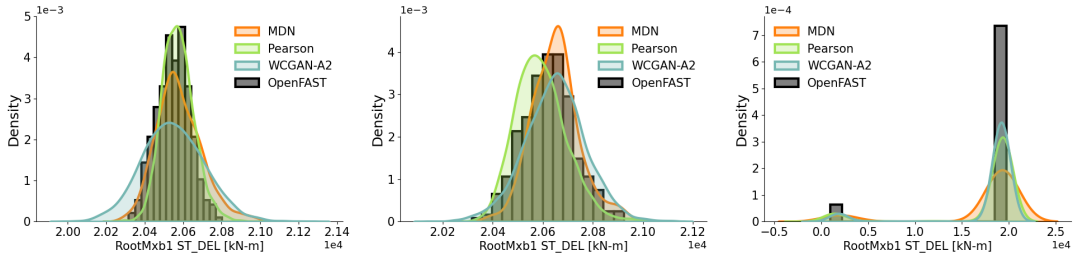
This chapter contains the rest of the results from the experiment on the Aero dataset in Section 4.4.



(a) Conditional 1-W distances for different wind speeds

(b) Validation log-likelihood against training epochs

Figure C.1: Results from models trained on the RootMxb1_ST_DEL load channel in the Aero dataset

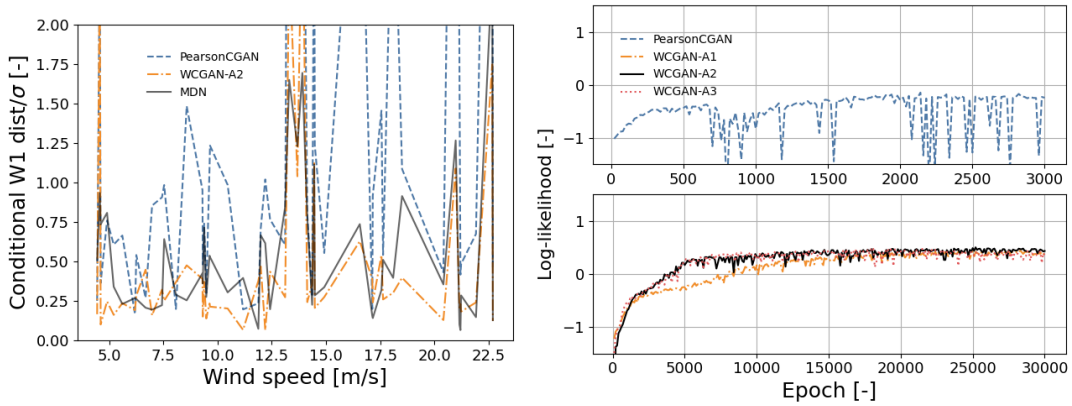


(a) Low wind speed

(b) Medium wind speed

(c) High wind speed

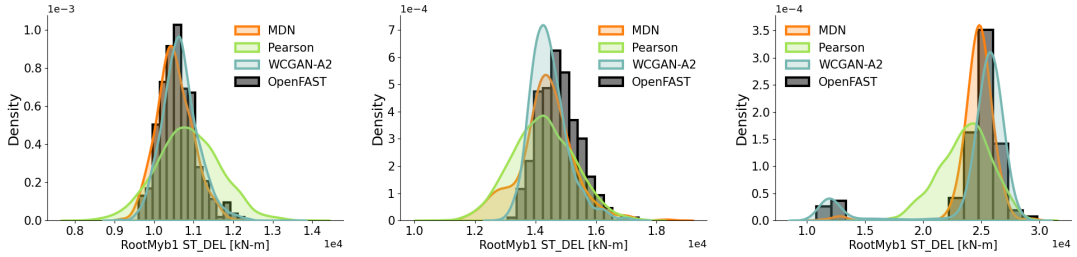
Figure C.2: Predicted and reference (OpenFAST) conditional pdf for the RootMxb1_ST_DEL load channel in the Aero dataset



(a) Conditional 1-W distances for different wind speeds

(b) Validation log-likelihood against training epochs

Figure C.3: Results from models trained on the RootMyb1_ST_DEL load channel in the Aero dataset

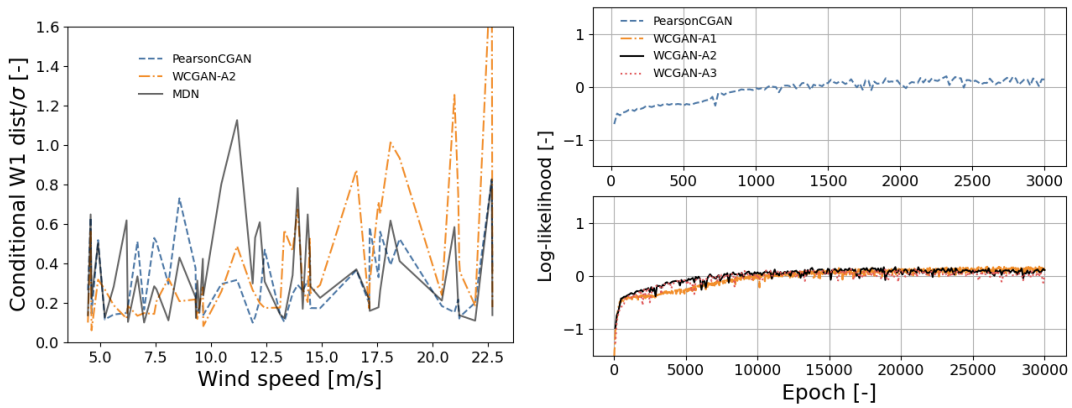


(a) Low wind speed

(b) Medium wind speed

(c) High wind speed

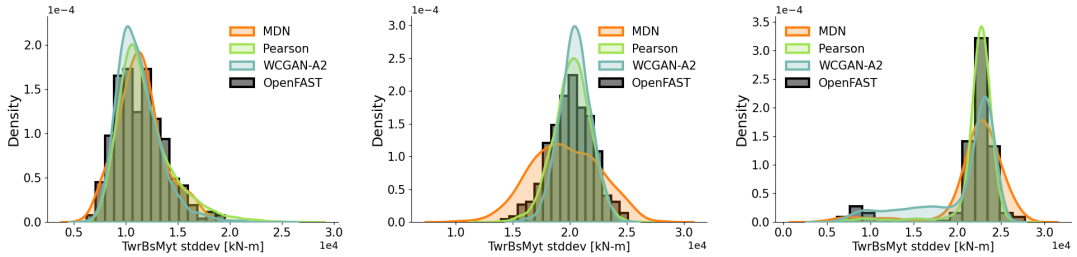
Figure C.4: Predicted and reference (OpenFAST) conditional pdf for the RootMyb1_ST_DEL load channel in the Aero dataset



(a) Conditional 1-W distances for different wind speeds

(b) Validation log-likelihood against training epochs

Figure C.5: Results from models trained on the TwrBsMyt_stddev load channel in the Aero dataset



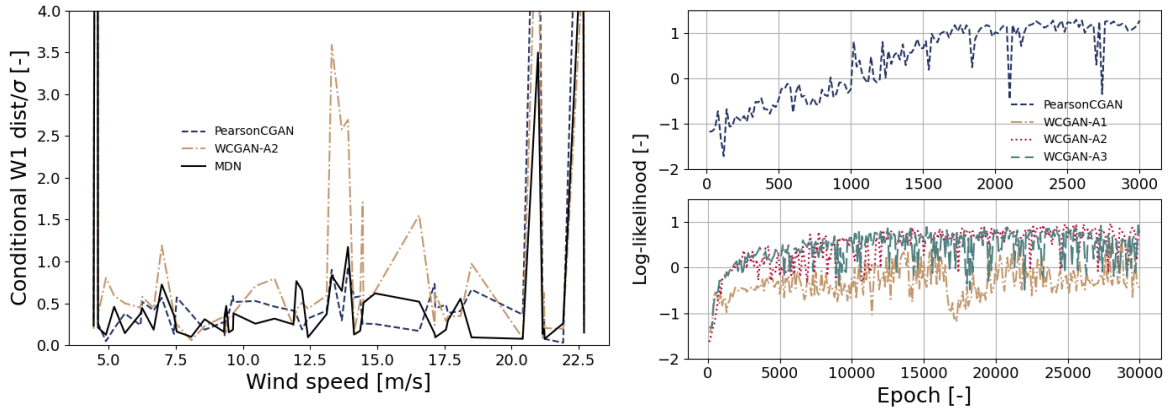
(a) Low wind speed **(b)** Medium wind speed **(c)** High wind speed

Figure C.6: Predicted and reference (OpenFAST) conditional pdf for the TwrBsMyt_stddev load channel in the Aero dataset

D

Further results for AeroHydro dataset

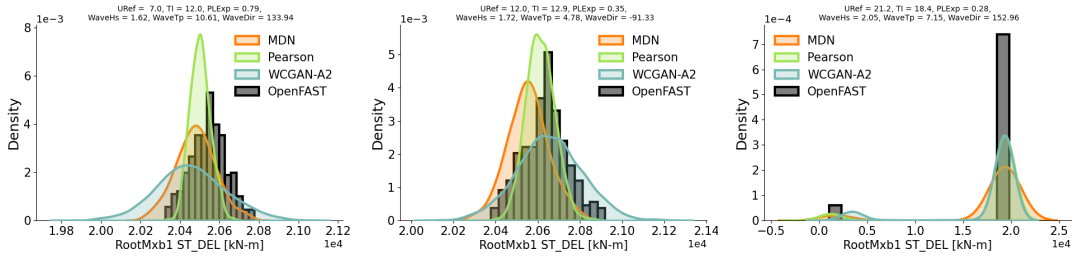
This chapter contains the rest of the results from the experiment on the AeroHydro dataset in Section 4.4.



(a) Conditional 1-W distances for different wind speeds

(b) Validation log-likelihood against training epochs

Figure D.1: Results from models trained on the RootMxb1_ST_DEL load channel in the Aerohydro dataset

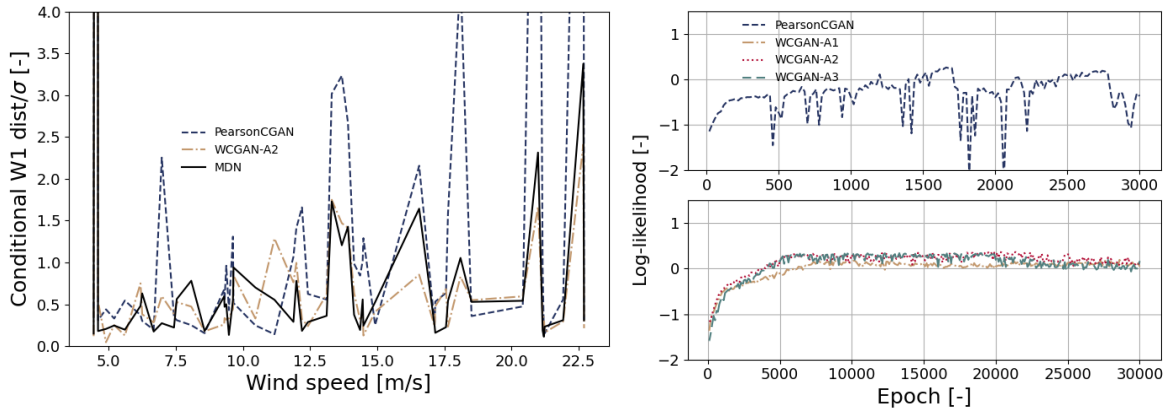


(a) Low wind speed

(b) Medium wind speed

(c) High wind speed

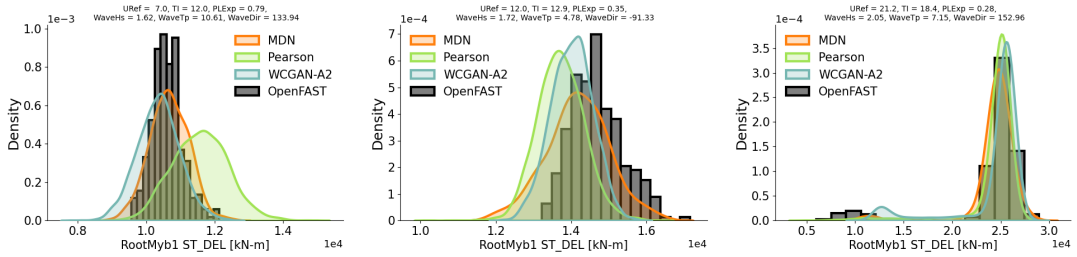
Figure D.2: Predicted and reference (OpenFAST) conditional pdf for the RootMxb1_ST_DEL load channel in the Aerohydro dataset



(a) Conditional 1-W distances for different wind speeds

(b) Validation log-likelihood against training epochs

Figure D.3: Results from models trained on the RootMyb1_ST_DEL load channel in the Aerohydro dataset

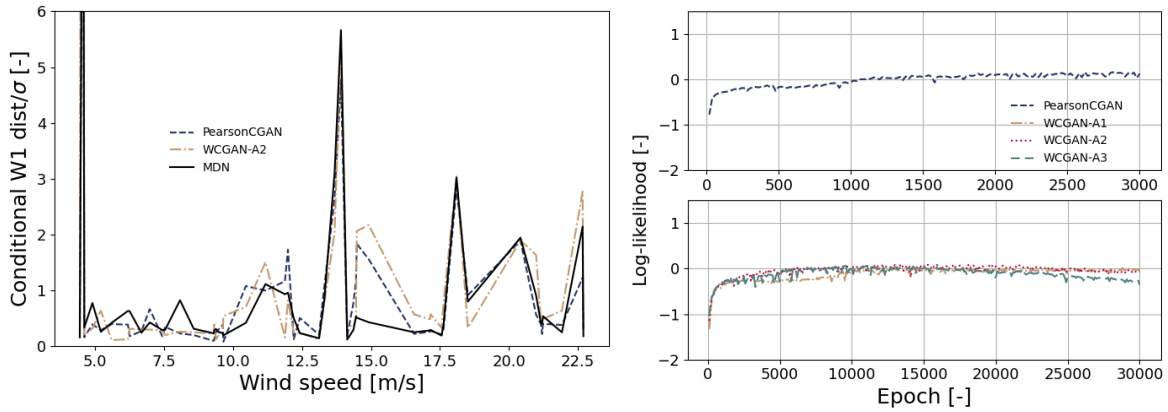


(a) Low wind speed

(b) Medium wind speed

(c) High wind speed

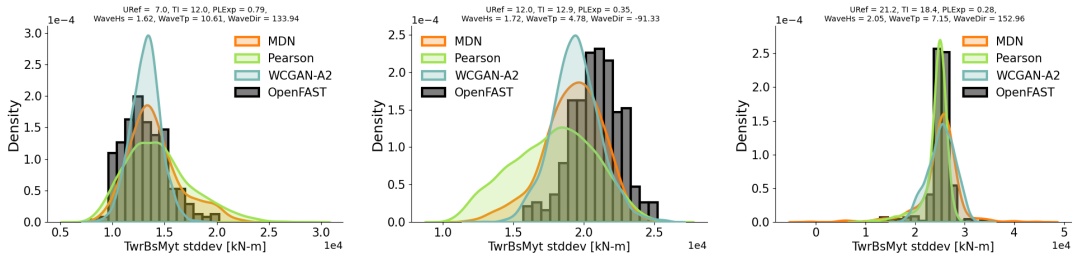
Figure D.4: Predicted and reference (OpenFAST) conditional pdf for the RootMyb1_ST_DEL load channel in the Aerohydro dataset



(a) Conditional 1-W distances for different wind speeds

(b) Validation log-likelihood against training epochs

Figure D.5: Results from models trained on the TwrBsMyt_stddev load channel in the Aerohydro dataset



(a) Low wind speed **(b)** Medium wind speed **(c)** High wind speed

Figure D.6: Predicted and reference (OpenFAST) conditional pdf for the TwrBsMyt_stddev load channel in the Aerohydro dataset