School of Civil and Mechanical Engineering

Degradation Vector Fields with Uncertainty Considerations

Marco Star

0000-0003-0547-1525

This thesis is presented for the Degree of

Doctor of Philosophy

of

Curtin University

March 2023

Declaration

To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgement has been made.

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.

Signature:

Date: 19/08/2023

Abstract

Machinery prognostics is often described as the problem of estimating the remaining useful life (RUL) of machinery or its components. Deep learning methods have become popular for RUL estimation, given sensor signals that relate to the RUL in some unknown way. However, there are some challenges deep learning models face in the context of machinery prognostics that are often highlighted by the community. Some of these problems include,

- Lack of uncertainty quantification: Deep neural networks are deterministic, and many methods utilised in machinery prognostics do not account for uncertainty in the output RUL estimate; instead, only a point estimate is given.
- Interpretability: Deep neural networks are black box methods where the relationship between the input and output is not clear. This can make it difficult to work with these models or to adjust them based on real word changes in the problem setup.
- Lack of run-to-failure data: Training data is needed to train these networks, meaning the input sensor signals and corresponding RUL target values are needed. However, to get RUL target values, one needs to know the RUL, and this often requires running machines until failure to get this data. This can be expensive if a lot of data is needed, which is the case for deep learning methods.

This thesis mainly addresses the uncertainty quantification and interpretability aspects; however, some work addresses reducing the amount of failure data needed.

Neural differential equations are explored as a way of improving the interpretability of neural networks. This was because they could either be used as sequential models and propagate latent states forward in time or function like a neural network and propagate input data to output data using a differential equation. Both of these applications of neural differential equations could potentially provide improved interpretability of either the neural network results or the neural network's inner workings. The first attempt to apply this was made by replacing networks such as Convolutional Neural Networks (CNNs) with a neural differential equation network that was roughly equivalent and comparing results. Hence, instead of using a CNN to output the RUL, a neural differential equation version of a CNN would be used and the results were compared.

Another avenue was also explored that makes up the bulk of the thesis as it manages to address each of the challenges listed. Deep sequential generative models are explored as probabilistic models that estimate the RUL over time. Specifically, conditional Dynamical Variational Autoencoders (DVAEs) are utilised, as they also define latent space dynamics related to the RUL outputs. Through the generative model, the latent dynamics are related to the RUL (and, therefore, the degradation of the machine), which aids interpretability as the model has effectively learnt dynamics that describe degradation. Moreover, generative models are probabilistic by design and therefore quantify the uncertainty of their outputs. Many generative models used in machinery prognostics are used, like a preprocessing step might be used, to clean the sensor data and reconstruct it with reduced noise or find a lower dimensional latent variable representation of the sensor data. In this thesis, the underlying theory of these sequential generative models is explained and then multiple ways of designing a DVAE to directly estimate the RUL is shown.

Each DVAE has its pros and cons, firstly a “vanilla” DVAE is shown that largely uses standard feedforward networks and some RNNs to estimate the RUL that acts as a baseline model for the rest. Another DVAE uses Neural Stochastic Differential Equations (NSDEs) as a dynamic model for the latent variables and showcases how using different models for the latent dynamics can affect the RUL estimates and interpretability of results. Finally, Kalman-DVAEs are introduced, which show how the degradation dynamics could be formulated in a linear Gaussian latent state space. While the RUL estimation performance may be reduced, the benefit of linear dynamics that describe the degradation mechanics is gained. These models are compared with other state-of-the-art RUL estimation models and evaluated on NASAs turbofan engine dataset (CMAPSS).

This work's novelty includes using neural differential equations as a type of network for RUL estimation and involves a detailed comparison with a Residual Network. The main contribution, however, is the exploration and use of DVAEs for RUL estimation. DVAEs are shown to require conditional variables to have a noncausal relationship with the other variables, and this informs the practical implementation of the model as it requires a sequence-to-sequence approach for RUL estimation. Guided by this requirement and the DVAE framework, the DVAE, NSDE-DVAE and Kalman-DVAE each provide some contribution to exploring this framework. The DVAE serves as a good baseline, and semi-supervised methods are also explored using this architecture. NSDE-DVAEs show how to incorporate NSDEs as the latent dynamics, allowing them to be governed by a continuous model. Kalman-DVAEs show how the DVAE can be simplified to use linear Gaussian state space systems to describe the relationship between latent dynamics and RUL estimates and train this model using the Kalman filter. Part of this novelty comes from the fact that the RUL estimation problem is unique compared to other problems DVAEs may be used for in deep learning applications. For example, RUL trajectories are linear, so the Kalman filter can be used to train the model, whereas other problems tend to have nonlinear target values. Many problems may try to extrapolate certain values for applications such as generating sound or music or extrapolating what will occur in a video. Here the problem is different, and a conditional DVAE is required to output a variable that cannot be observed (the RUL). Hence, the application of DVAEs for the RUL estimation problem was a novel problem, and this thesis explores how one could apply the DVAE to this problem.

Acknowledgements

When I was a kid I watched a movie called "Jimmy Neutron: Boy Genius". The movie's main character was, believe it or not, a boy genius, who had all these cool inventions and in his lab, he always seemed to have some cool experiments running. All the parents in his town also get abducted by aliens, but that's irrelevant to the point I'm trying to make. So at the age of five, I knew what my goal in life would be... to become like Jimmy Neutron. It turns out I could not be a boy genius because, believe it or not, becoming a genius is difficult and requires a lot of studying and hard work, which I was sadly allergic to as a kid. But maybe as an "adult" I can contribute a drop of knowledge to the vast pool of it we have collected over time, which is good enough for me. All this is a convoluted way of saying my first thanks go to my parents, who supported me through all the "studying and hard work" I have done to get to this point. Even if I don't think of myself as a genius, they seem to (don't let them know the truth, dear reader, they'll be crushed). I would also like to thank Chen and Charlie for taking me to the movies to see Jimmy Neutron, for giving me advice, and for generally being interesting people to talk to.

It's rather hard to do a PhD without supervisors, so I would first like to thank Rodney Entwistle for being there initially to guide me through this journey. I would also like to thank Ian Howard, who was the first to bring to my attention during my undergraduate studies that a PhD may be something I would be interested in doing and then supervising me through a large portion of it. Finally, a huge thanks to Kristoffer McKee for being the one to supervise me and reading more drafts than is humanly possible.

I would like to thank the Australian Government Research Training Program (RTP) Scholarship for supporting this research. Thereby giving me the opportunity to pursue my dreams and achieve my goals.

I would also like to thank everyone at Jenike, not just for being flexible with my hours and providing a pleasant work environment, but also for fuelling me with pizza and cake (the companionship wasn't too bad either). A special thanks go to everyone at JJA for all the random conversations and jokes we shared.

Table of Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence

ANFIS Adaptive Neural Fuzzy Inference System

ANN Artificial Neural Network

CMAPSS Commercial Modular Aero-Propulsion System Simulation

CNN Convolutional Neural Network

DBN Deep Belief Network

DVAE Dynamical Variational Autoencoder

ELBO Evidence Lower BOund

EMD Empirical Mode Decomposition

GAN Generative Adversarial Network

GP Gaussian Process

GRU Gated Recurrent Unit

HI Health Indicator

HMM Hidden Markov Model

HPC High-Pressure Compressor

K-DVAE Kalman Dynamical Variational Autoencoder

LGSSM Linear Gaussian State Space Model

LSTM Long Short-Term Memory

ML Machine Learning

MLP Multilayer Perceptron

MSE Mean Squared Error

NODE	Neural Ordinary Differential Equation
NSDE	Neural Stochastic Differential Equation
NSGPR	Non-Stationary Gaussian Process Regression
PCA	Principal Component Analysis
RBM	Restricted Boltzmann Machine
RCNN	Recurrent Convolutional Neural Network
ResNet	Residual Network
RGB	Red, Green and Blue
RMS	Root Mean Squared
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
RSSM	Recurrent State Space Model
RUL	Remaining Useful Life
RVM	Relevance Vector Machine
SDE	Stochastic Differential Equation
SGD	Stochastic Gradient Descent
SRRN	Stochastic Recurrent Neural Network
STFT	Short Time Fourier Transform
VAE	Variational Autoencoder

Chapter 1

Introduction

1.1 Goals of Machinery Prognostics

The goal of machinery prognostics can be defined as trying to estimate the Remaining Useful Life (RUL) of a machine or a machine component. For example, machinery prognostics techniques may be used to find the RUL of a turbofan engine for an aircraft or estimate the degradation of a lubricant used in the bearings of a wind turbine. Estimating the RUL is useful as it allows for informed decision-making, such as when to perform maintenance and knowing the health state of the machine. RUL estimation is generally achieved by modelling the degradation of the equipment and finding patterns, or a relationship, between equipment degradation and its RUL. Generally, the types of models used in machinery prognostics to estimate the RUL can be categorised as follows (T. Xia et al., 2018),

- Physics-based models: uses mathematical models based on the system's physics. For example, Paris's crack law is a differential equation that can be used to estimate the crack length at a given point in time, given certain physical properties and the initial crack length of the component.
- Data-driven models: use data gathered from the component to find patterns or create a model that can estimate the RUL of the machine from the gathered data. Unlike physics-based models, domain knowledge of the degradation characteristics for that component is not required.
- Hybrid models: uses a mixture of data-driven and physics-based models to model the degradation. For example, a differential equation based on the physics of the system could model degradation. At the same time, data-driven methods estimate parameters of that equation that normally require physical tests to estimate.

There has been an increase in available data from industrial plants that have motivated the adoption of data-driven methods, specifically machine learning methods (Diez-Olivan et al., 2019). Due to data availability, this thesis focuses on a specific type of machine learning method known as deep learning, which has become popular due to the rise of available data and its effectiveness when given a large amount of data. Deep learning methods use deep neural networks to estimate the RUL using large amounts of related data. These neural network models excel at finding and modelling a relationship between the input and output variables, e.g. sensor data and RUL values. Due to the effectiveness of neural networks, there has been an increase in research on using deep neural networks in the machinery prognostics domain. However, using deep learning methods for prognostics presents some problems which will be addressed in the next section.

1.2 Problems with Deep Learning Methods in Prognostics

Using deep neural networks in machinery prognostics has shown state-of-the-art results in RUL estimation; however, there are some drawbacks when using deep neural networks.

Firstly, a large problem with utilising deep neural networks in machinery prognostics is that many methods do not account for uncertainty in their RUL estimates. Many deep learning methods for prognostics perform well at estimating the RUL but only output a point estimate of the RUL without accounting for any uncertainties (W. Peng et al., 2019). However, consideration of uncertainties for any machinery prognostics application is important for decision-making and makes the model more useful for reliability and safety applications (Z. Xu & Saleh, 2021). Without the ability to quantify uncertainty, it is difficult to make informed decisions about when to maintain or replace the machine or component. Hence, uncertainty quantification is a critical component of a machinery prognostics model (Sankararaman, 2015).

Secondly, deep neural networks are black-box models, and it isn't easy to interpret how the network achieves its results (Kraus & Feuerriegel, 2019). It can be difficult to see how a network has reached a decision or achieved a certain output and what contributed to its good performance. This can also make it difficult to add expert knowledge to the model as it is not intuitive or obvious how one would give the network expert knowledge it could incorporate into its decision-making. Some cases use deep learning to identify the parameters of known physics-based models for the degradation of certain machines (Nascimento & Viana, 2019; Yucesan & Viana, 2019; Dourado & Viana, 2019). This would create a hybrid model between deep learning and physics-based models. However, these would work as pure data-driven models if no physics-based model was available. Some models aim to learn a Health Indicator (HI) from the sensor data (González-Muñiz et al., 2022; Qin et al., 2021; Ping et al., 2019; She et al., 2020). However, many of these models focus on learning a good HI by assessing metrics such as Monotonicity, Trendability and Robustness rather than RUL estimation. While some do have an analysis of RUL estimation performance, they are often compared only to other HI methods and not to other deep learning methods that directly estimate the RUL. This raises the question of whether it is possible to construct a deep learning model with an interpretable component, such as a HI, and retain state-of-the-art performance for RUL estimation.

Finally, deep learning models require a lot of data to optimise (train); in the machinery prognostics case, this means a lot of run-to-failure data. This means that multiple machines or their components must be run until they fail so that sensor data and the RUL target data can be acquired to train and validate the neural network. The majority of deep learning methods used for machinery prognostics use the sensor and RUL data to train the neural network in a supervised manner. In this case, the sensor signals are used as inputs to the network, which then outputs the RUL estimate. The network's output is then compared to the true RUL gathered from the run-to-failure data and used to train the network. Note that some methods use semi-supervised approaches that aim to reduce the number of target RUL data needed to train the network (Z. Xu & Saleh, 2021). These methods are explored in section 2.2.5. Briefly, semi-supervised methods generally train both an unsupervised model, using the input sensor data as the targets to learn some abstract representation of this data and a supervised model that can use information from the unsupervised model to help estimate the targets. Hence, for machinery prognostics, this could involve training the unsupervised model on the more abundant sensor signals, which outputs a latent variable representing the sensor signals. This latent variable output from the unsupervised model could then be used as an input to the supervised model, which outputs an RUL estimate. Usually, these are trained together, and the supervised model is applied to the data with known corresponding RUL target values. The idea is that the unsupervised stage helps the network get closer to an optimum solution, thereby requiring fewer targets to train the supervised model. Hence, when training the supervised model, it is already close to an optimum solution and requires fewer target data (RUL data) (Listou Ellefsen et al., 2019).

1.3 Deep Generative Models and Latent Dynamics

This section introduces the concept of deep generative models that will play a large role in the majority of the thesis. Generative models learn underlying probability distributions of the training data and play a large role in some of the topics previously mentioned such as, semi-supervised learning and HI construction. For this introduction, an illustrative example is given to aid in understanding what a generative model is and what it can do.

Imagine we have some input data available (X); for example, Red, Green and Blue (RGB) values representing images of various people's faces. A generative model aims to learn[1] how to sample from the distribution, $p(X)$, so that it can sample novel variables, X, that the network did not see during training. Hence, in this example, $p(X)$ would be the probability distribution of all possible RGB values, and the neural network wants to increase the probability that these values take the form of an image (such as a face). Generative models tend to sample from $p(X)$ indirectly by defining a latent space through the distribution, $p(Z)$, and learning a mapping between Z and X. Note, Z is a latent variable that relates to the input data X but doesn't need to have a physical meaning like RGB values do, instead the variable Z functions as an abstract representation of the input data X. This allows one to sample $Z \sim p(Z)$ and "decode" Z using some function $p(X|Z)$ to find samples, X, in the input space. By sampling repeatedly one could sample all over the distribution $p(X)$. Hence, we can indirectly sample from $p(X)$ using $p(X|Z)p(Z) = p(X, Z)$. $p(X, Z)$ will be defined as a generative model due to its ability to generate samples that come from $p(X)$. Note, $p(X, Z)$ is not directly described but is instead the combination of sampling from the latent model, $p(Z)$, and then decoding using $p(X|Z)$. In the case of deep generative models, the distribution $p(Z)$ and/or $p(X|Z)$ are modelled using neural networks, and in this thesis, the term "generative models" will refer to deep generative

[1]The term "learn" in a deep learning context refers to optimising the deep neural networks parameters during the training process; where the optimisation process depends on a loss function defining what "optimal" means and data is used during training to optimise the network.

models unless otherwise stated. Figure 1.1 shows a simplified illustration of a generative model sampling images before and after training; the generative model aims to generate images of “faces”.

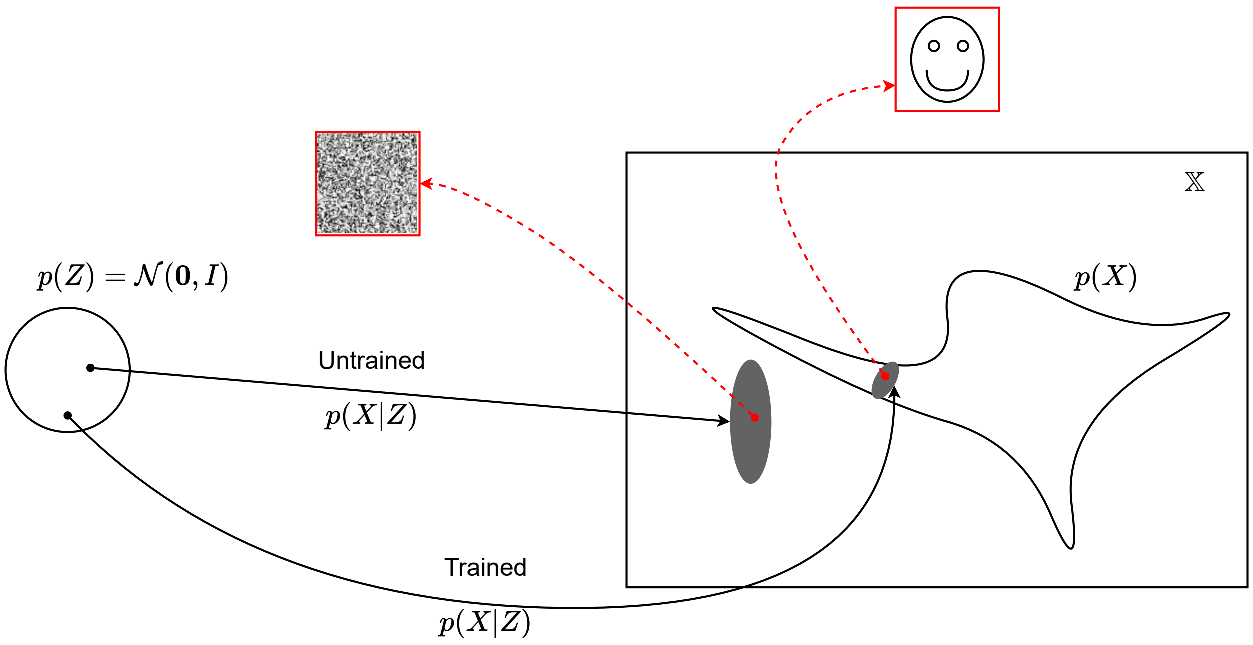


Figure 1.1. Shows a generative model illustration in 2D space for simplicity. The circles and ovals labelled as $p(Z)$ and $p(X|Z)$ are Gaussian distributions in 2D space. $p(Z)$ is a standard Normal distribution representing the prior latent model. $p(X|Z)$ is the decoder and is found by sampling $Z \sim p(Z)$ and using the sample as an input to a neural network that outputs the mean and standard deviation describing the Gaussian distribution $p(X|Z)$. Note that $\mathbb{X}$ represents the space of all possible X values and $p(X)$ is an unknown probability distribution representing the space of X values that make up all possible faces in the given dataset. The red dotted lines point to images that are sampled from the generative model; the untrained model samples random values in the input space $\mathbb{X}$. The trained model samples from a subspace within $p(X)$. Since $p(X)$ contains the combinations of X that represent images of faces, the generative model manages to sample an image of a face. Since it is unlikely to sample an image that exactly matches one of the images in the dataset used to train the network, the sampled image is likely a novel image of a face that is not a part of the original dataset.

Generative models learn probability distributions $p(X|Z)$ and $p(Z)$ and can therefore account for the uncertainties in the variables X and Z; they also find some latent space $p(Z)$ whose variables are related to X through $p(X, Z)$. Hence, they are probabilistic models that effectively learn the probability distribution $p(X)$; usually by sampling from it indirectly as described above. They also have a latent variable related to the inputs X and can therefore be seen as a lower dimensional representation of the input variables. Both of these facts show that generative models can potentially be used to address the “uncertainty quantification” and “black-box method” problems mentioned in the previous section. They can also be trained in an unsupervised manner, making them well-suited to act as the unsupervised model in semi-supervised learning applications. These semi-supervised learning techniques can help pre-train a network using only the input data (such as sensor signals), which can reduce the amount of target data needed to train the RUL estimation network. Another application of semi-supervised learning is to train a network, again using only the input data, and find a latent representation of those input values that essentially acts as a pre-processing data stage before the RUL estimation stage. Hence, generative models address many of the problems mentioned in Section 1.2. The uncertainty quantification is addressed through the probabilistic formulation of the deep generative model; the latent variables address the interpretability of the network outputs, and semi-supervised learning can reduce the amount of run-to-failure data needed to train the model.

To give a concrete example, a simple case of how a generative model may be used to uncover the probabilistic hidden dynamics of a system will be illustrated. Looking at Figure 1.2, the

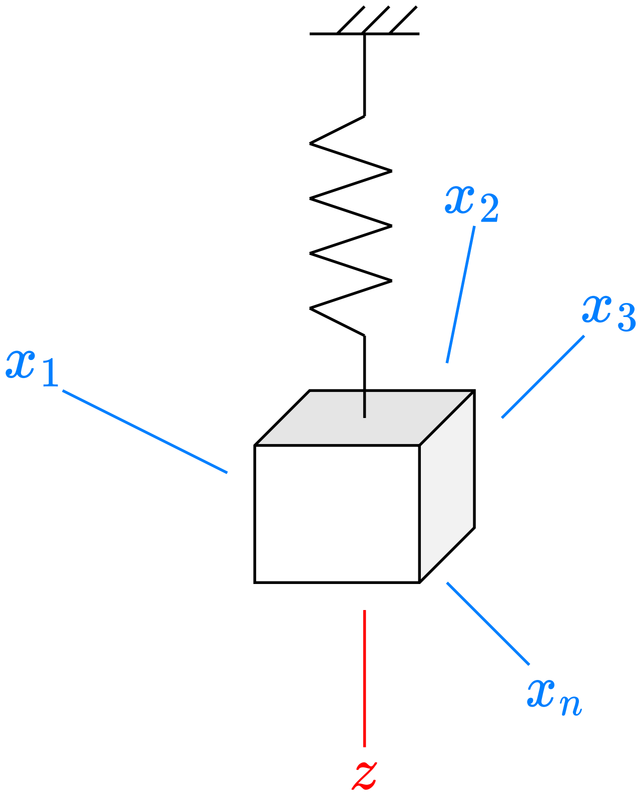


Figure 1.2. A spring-mass system with n sensors that measure the distance between the sensor and the mass; the sensor distances are denoted as $(x_1, ..., x_n)$. The axis labelled z denotes the dimension in which the spring-mass system oscillates. Hence, if z is the axis chosen when analysing the system, the equations of motion would be 1-dimensional.

setup is a simple spring-mass system, and the aim in this example is to discover the equations of motion as it oscillates along the z-axis. One might easily discover the equations of motion about the z-axis (e.g. by using Newton's second law), but what if this was a system we knew nothing about? In this case, measurements are taken that are assumed to be related to the motion of the system; here, they are labelled using x_i, where i is the sensor number and there are n sensors in total. It is easy to imagine, based on the restrictions on the system, that certain positions described using coordinates $X = [x_1, ..., x_n]$ are more likely than others. This would lead to some distribution $p(X)$, which could be sampled to find the most likely coordinates of the spring-mass system. Since $p(X)$ is usually unknown in the complex systems we are interested in, if we want to sample likely coordinates from $p(X)$, the distribution has to be approximated. As previously mentioned, generative models are a way of accomplishing this task. The generative model does this by finding a mapping from some lower dimensional latent space, $p(Z)$, to the input space using a mapping function or likelihood $p(X|Z)$, which is usually a neural network called a decoder. This example is useful because it shows an obvious physical representation of a lower dimensional space that would serve to encode all the input variables. The z-axis may not be what the neural network finds, but this example illustrates the concept. Hence, $p(Z)$ would be the probability of the positions, Z, the spring-mass system could have along the z-axis. A function can then be used to convert the z-axis positions to equivalent $X = [x_1, ..., x_n]$ coordinates, making $p(X|Z)$ a function that transforms Z coordinates to X coordinates. Hence, a generative model aims to come up with a representation for the joint distribution $p(Z, X)$; this is often done in deep learning by defining two neural networks $p(X|Z)$ and $p(Z)$ and using them to sample X that ultimately belongs to the distribution $p(X)$. However, through this example, it can be seen that it may also be possible for one to define probabilistic dynamics of the spring-mass system oscillating latent dynamics describing how Z changes through time. Note this is not strictly true in this example, as we did not consider a temporal component where X and Z are changing through time. There are more details on how to formally construct these dynamical generative models by replacing X with a sequence $x_{1:T}$ and Z with $z_{1:T}$, where T is some arbitrary time point, $T > 1$. These can be found in Section 3.2 in the background chapter; this example was kept simple to introduce the key concepts without getting stuck in formalism. This example shows how generative models create interesting opportunities for machinery prognostics applications. These can be stated as follows,

1. Deep generative models are probabilistic models; therefore, they are deep learning models that quantify uncertainty. Also, note that generative models learn to sample from $p(X)$ which can be any arbitrary distribution; hence, these models can be flexible in their expression of the uncertainties of the input variables X. Expressing the uncertainty of the RUL estimates was one of the main problems identified for deep learning models used in machinery prognostics applications. Using a deep learning model that is probabilistic can therefore potentially help solve this issue if it can be used for RUL estimation.

2. Deep generative models can have encoded latent dynamics with a much lower dimensionality than the input variables X. This could potentially help interpret the network's results by visually showing the latent dynamics which correspond to the degradation of the machine.

These points help deal with two large problems with deep learning applied to machinery prognostics. Generative models could therefore be a potentially useful subsection of deep learning to explore when dealing with machinery prognostics problems.

Generative models have been used successfully in other fields to model dynamic systems using latent dynamics. Many of these methods use the Variational Autoencoder (VAE) (Kingma & Welling, 2014), a generative model trained using variational Bayesian techniques, but extend it to deal with sequential inputs and outputs. A popular application of these sequential generative models is the planning and control of an object using video footage as the input data. In this case, the generative model learns the system's latent dynamics by analysing the video footage and how the images change over time. For example, this could be useful in autonomous vehicles that have cameras and need to use the images they capture to make actions in the world. Another example of these types of models is in speech or music generation; in this case, the network is trained on sound signals and after training, can generate new sound signals. The Recurrent State Space Model (RSSM) and Stochastic Recurrent Neural Network (SRRN) are popular examples of these types of generative models used for these applications (Hafner et al., 2019; Fraccaro et al., 2016). Both use neural networks to describe the dynamics and the decoder from the latent domain, Z, to the input domain, X. However, assumptions and alterations to these types of models can be made, e.g. latent dynamics can be made to be Markovian, i.e. $p(Z) = p(z_t|z_{t-1})$. An interesting practical example of these types of alterations are models which use a network to transform the observations X into augmented observations A. Now in this augmented domain, the generative model could be assumed to be linear and so $p(A|Z)$ and $p(Z)$ could both be linear and if these are Gaussian distributions, this could be solved using the Kalman filter (Fraccaro et al., 2017; Karl et al., 2017; Becker-Ehmck et al., 2019). Since, during training, these models are performing a sequential version of Bayes' theorem to train the model; some works use filtering methods which are algorithms to solve for this sequential Bayes' theorem, such as the particle filter, to optimise these models. Normally the criterion for optimising these models is the Evidence Lower BOund (ELBO). However, this is an indirect way of maximising the likelihood of sampling from $p(X)$. Using filtering algorithms, such as the particle filter, a direct approximation of $p(X)$ can be made and improve the model's performance (Maddison et al., 2017; Naesseth et al., 2018). Differential particle filters can also be found in the literature, where the aim is to learn a particle filter through deep learning that can then be applied to problems such as object tracking and robot localisation (Jonschkowski et al., 2018; Corenflos et al., 2021; Kloss et al., 2021). Using filtering algorithms to train generative models is expanded on further in the background section 3.3.2 and explored in Chapter 7.

1.4 Research Questions

This thesis focuses on both quantifying the uncertainty of the RUL estimates and trying to find some latent dynamics that aid in reducing the black-box nature of deep learning models. Hence, the research questions largely focus on neural network models that use sensor signals as inputs to construct latent dynamics that model degradation and probabilistically model the RUL estimates they output.

1. How could a neural network incorporate dynamic systems to help reduce its black-box nature; specifically for RUL estimation?
2. What would these degradation dynamics look like, i.e. how would they be constructed, and what form would they take? What constraints would need to be placed on them (if any)?
3. How would uncertainty be incorporated into this framework?
4. Can this all be incorporated into one framework? If so, how would it be trained and evaluated?

Throughout the thesis, there is an attempt to answer these questions or at least make enough progress to allow for more specific research question to be asked. The outline of the thesis (after this section) is as follows,

- Chapter 2: The literature review starts to go over general machinery prognostics literature to get an understanding of how RUL estimation problems have been solved. It then specifically looks at deep learning models for RUL estimation found in the literature and any work done on latent dynamic systems or uncertainty quantification.

- Chapter 3: The background goes over many of the more advanced deep learning architectures used in the thesis. It also goes over auxiliary topics such as Bayesian filtering, which helps to understand some of the advanced network architectures used in the thesis.
- Chapter 4: This is the first contribution of the thesis. It uses Neural Ordinary Differential Equations (NODEs) to describe the dynamics of a neural network itself. It does not construct latent dynamics but uses a dynamic system to help understand how a neural network could process information.
- Chapter 5: Shows a relatively simple case of how sequential generative models can be used to address the research questions. The rest of the chapters build on this and look at specific changes that can be made when applying the model while still using the same underlying theory behind deep generative models. Specifically, the sequential generative models are known as Dynamical Variational Autoencoders (DVAEs).
- Chapter 6: Shows another case of using Neural Differential Equations, but here it is used to describe the latent dynamics of the DVAE. Hence, this builds upon the content presented in the previous two chapters.
- Chapter 7: Focuses on DVAEs as the last two chapters did. This chapter simplifies the model and showcases the connections between training deep generative models and state estimation (specifically Kalman Filters). It shows how Linear Gaussian State Space Models (LGSSMs) can be utilised in machinery prognostics to estimate the RUL and model the degradation mechanics through latent dynamics described using LGSSMs.

The main contribution from this thesis can ultimately be described as a framework on how to train and specify neural network models for RUL estimation, which can account for uncertainty in their estimates and have latent dynamics. This is largely done through conditional generative models that deal with sequential data. It should be noted that sequential generative models are nothing new (a good review can be found in Girin et al., 2021); however, they are often not used for direct RUL estimation. Although a handful of works have used generative models for RUL estimation, they are more focused on the HI construction rather than competing with state-of-the-art RUL estimation models Wei et al., 2021. The work in this thesis also makes theoretical justifications for what inputs should be used for each neural network in the generative model and how to construct these models for machinery prognostics tasks specifically. For example, consider the distribution $p(z_{1:T}|x_{1:T})$, this could be factored into a product of distributions $\prod_{t=1}^{T} p(z_t|z_{1:t-1}, x_{1:T})$. If this model is a neural network its inputs would be $z_{1:t-1}$ and $x_{1:T}$ and it would output z_t. The model could potentially be simplified further by making certain assumptions e.g. the Markov assumption so the dependence of the previous $z_{1:t-1}$ becomes a dependence on only the previous variable z_{t-1}. Or the system could be assumed to be causal so future values are not taken into account such that the dependence on $x_{1:T}$ becomes a dependence on $x_{1:t}$. If these assumptions are made then the network inputs become z_{t-1} and $x_{1:t}$ instead of $z_{1:t-1}$ and $x_{1:T}$. The causality assumption is the main assumption that is explored in this thesis as many models in machinery prognostics use this assumption. Hence, these model simplifications are explored as the structure of a lot of these models is assumed, which may lead to models that do not perform to their full potential. An example of how the exploration of these assumptions aids in constructing useful models for machinery prognostics is the model described in Chapter 7. In Chapter 7 deep generative models to state space models and Kalman filtering in an attempt to relate these black box neural network models to theories well-established in other fields (such as control theory). This relation between filtering algorithms and sequential generative models is not new, but the specific setup of the RUL estimation problem in machinery prognostics allows for a novel construction of these types of deep state space models and Kalman filter-based neural networks. Hence, the main contributions in this thesis are how these models are applied to RUL estimation and the simplifications and alterations one can or should make to this model, specifically because it is being applied to the RUL estimation task.

Chapter 2

Literature Review

2.1 Machinery Prognostics Overview

As mentioned in the introduction chapter, there are three broad categories that RUL estimation models can be classified as,

- Physics-based models
- Data-driven models
- Hybrid-models

Data-driven models are the main focus of this thesis, but the physics and hybrid will briefly be looked at so that they may be contrasted and discussed relative to the data-driven models. Each of the model's advantages and disadvantages will be discussed in the following sections.

2.1.1 Physics-Based Methods

As mentioned in the introduction, physics-based models use an understanding of the underlying physics-based principles that govern the degradation mechanism of interest. For example, the crack length could be mathematically modelled through the Paris-Erdogan model (Paris & Erdogan, 1963), and the machine could be said to have reached its RUL at a certain threshold crack length. One of the benefits of a physics-based approach to prognostics is the interpretability of the model, meaning they include variables that have a clear physical interpretation (e.g. crack length). The parameters that a physical model relies on can usually be determined through physical testing or experiments on the equipment or material (Lei et al., 2018). This can be seen as a downside due to the expense of performing experiments on components similar to the original component being analysed for prognostics. However, they do not require the large run-to-failure data sets a machine learning based model would require (Cubillo et al., 2016). Another benefit is that physics-based approaches can be accurate for components with well-known mathematical models. For example, in (Lei et al., 2016), the Paris-Erdogan model is used to model the health of bearings and its parameters are updated using a Particle filer. However, it can become increasingly difficult to understand the physics of degradation for more complex equipment and the addition of noise through environmental variables. Hence, data-driven or hybrid methods must be considered when the physics of degradation becomes too complicated to model.

2.1.2 Hybrid Methods

Hybrid methods can help mitigate some of the downsides of physics-based models. For example, expensive physical testing or experiments may be required to estimate the parameters of a physics-based model. Hybrid models can use gathered data and data-driven methods to estimate these parameters, potentially reducing the labour and expenses involved with physical testing. They can also be used to help improve the overall performance of a prognostics algorithm performance. For example, (Liao & Köttig, 2016) proposed a framework using a physics-based degradation model to estimate the degradation states of lithium-ion batteries while using a data-driven model as a measurement model to convert estimated states from state space to measurement space. Both models were used in a particle filter for state estimation. This resulted in more accurate degradation state estimates and predictions through the use of hybrid methods. (H. Liu et al., 2019) use a similar method where an Adaptive Neural Fuzzy Inference System (ANFIS) is the data-driven model used as a measurement model for a proton exchange membrane fuel cell. A physics-based model was used to model the degradation of the fuel cell, and this state transition model was used along with the ANFIS measurement model in an adaptive unscented Kalman filter. The Kalman filter provided greater accuracy in the degradation state values through its state estimates and predictions.

In (Hagmeyer & Zeiler, 2023), they outline some ways physics-based and data-driven models can be combined to create hybrid models. For example, one could use physics-based models passively or actively. In the passive case, the physics-based model could augment or generate synthetic data to be used in the data-driven prognostics model. In the active case, the outputs of the physics-based model could be used as additional input features for the data-driven prognostics model. Alternatively, in the active case, one can use the physics-based model within the data-driven model as an intermediate step when applying the data-driven model. For example, in (Nascimento & Viana, 2019; Yucesan & Viana, 2019; Dourado & Viana, 2019), physics-based models are used within a Recurrent Neural Network (RNN) cell for computing the degradation of the machine.

However, if the physics of the system is completely unknown due to its complexity, or varying environmental conditions make it difficult to have a consistent physical model, fully data-driven models may be considered.

2.1.3 Data-Driven Methods

Data-driven methods allow for the estimation of RUL using data gathered from the equipment as an input to the model. Estimating the RUL this way requires little to no knowledge of the degradation mechanics, that determine equipment failure. Hence, data-driven methods are considered black-box or grey-box methods, which leads to the downside that they can lack human interpretability when considering how the output was achieved given the inputs (Lei et al., 2018; Baptista et al., 2019). They can make up for this downside through some of the following advantages (Baptista et al., 2019),

- Little to no expert knowledge is required as one does not need to know the details of the degradation process.
- They can be applied more generally to datasets relating to different machinery components.

Unlike physics-based models, understanding the underlying degradation mechanics for specific components is not required. This allows data-driven methods to be used instead of physics-based models if the degradation mechanics have no known physics-based model. Data-driven models generally work by finding patterns between the RUL and the raw input data, assuming the data is dependent on or correlated to the RUL. Hence, they also have the advantage that they can be applied to different machinery components so long as there is enough relevant data available, while a physics-based model might need to find the unknown model parameters for specific machines. This generality makes data-driven models a powerful tool for machinery prognostics applications, especially when considering the wide array of machines and components without detailed mathematical models describing all their possible degradation profiles.

The motivation to use data-driven methods for machinery prognostics has increased as more industrial data is gathered (Diez-Olivan et al., 2019). Data-driven methods can be further categorised as statistical or Artificial Intelligence (AI) methods. This thesis mainly focuses on artificial intelligence methods that use machine learning models to estimate the RUL. Hence, any examples in this review will focus more on AI methods than statistical methods. Traditionally these methods use the following steps to estimate the RUL (Lei et al., 2018).

- Data acquisition
- HI construction
- Health state division
- RUL prediction

This review focuses on the models themselves and how they are applied. Hence, the data acquisition step will not be discussed.

Health Indicators

HIs are variables that give information on the degradation state of the system through a representative variable. The RUL value depends on the system's degradation state; hence, selecting a HI that represents the degradation is an important step when calculating the RUL. For example, when estimating the RUL of lithium-ion batteries, it is common to use battery capacity as a HI with some threshold capacity value determining the point when the battery needs to be replaced. This can allow data-driven methods to fit curves to the data points gathered and extrapolate to the threshold value to estimate the RUL (H. Liu et al., 2019; H. Zhang et al., 2018; G. Zhao et al., 2017; X. Hu et al., 2016). Some HIs do not directly represent a physical or measured quantity, and so no obvious threshold values exist which can define the point of failure and help estimate the RUL. Instead, the HI is used for its correlation to the RUL, and the data-driven method is used to find the RUL by learning the relationship between HIs and the RUL. In this case, multiple HIs can be used as inputs in a data-driven model which finds their relationship to the RUL. These HIs could be the raw sensor data signals acquired from the equipment or derived from them, e.g. Root Mean Squared (RMS), kurtosis, Fourier coefficients, etc.

In traditional machine learning methods, HI construction involves finding hand-crafted features from raw data signals. For example, statistical features such as RMS or kurtosis values are calculated from the signals to indicate the equipment's health. To determine what features are useful for RUL estimation, the features go through a selection process where they are assessed using a selection criteria or metric. Commonly used metrics such as the Pearson correlation coefficient, monotonicity, robustness trendability and autocorrelation are used to determine which features are used as HIs and are unsuitable. (J. Wu et al., 2018) used a threshold Pearson correlation coefficient value to determine what features to keep as HIs depending on their correlation to the machining tool wear. Thresholds for monotonicity and autocorrelation were also used in the study to decide which features would be kept as HIs. (X. Li, Yang, et al., 2019) used a weighted combination of correlation, robustness, trendability and monotonicity metrics for feature selection. (Son et al., 2013) used Principal Component Analysis (PCA) to reduce the multivariate sensor signals to two dimensions. They created a failure zone based on the PCAs coordinates at the machines failure region and constructed a HI based on the Euclidean distance between the current PCA coordinates and the failure points coordinate. The HI represents the distance between the machine's current state and the failure state; once it equals zero, the machine can be said to have failed. Hence, they could then fit a Wiener process to model the HIs progression and extrapolate to this failure threshold to calculate the RUL.

Deep learning generally uses the raw data as inputs, and the neural network learns the features automatically through the operations in each layer of the network. However, deep learning can also create latent variables that can be used as health indicators for other RUL prediction methods. For example, (L. Guo et al., 2017) used a RNN to take raw input data as well as statistical features from the raw data signals as inputs and outputs a one-dimensional HI. An exponential model is fit to the HI and extrapolated to a known threshold to estimate the RUL. (X. Liu et al., 2022) uses a CNN on healthy bearing vibration data to construct a HI, and its trend is fit with an exponential model whose parameters are optimised using the Particle filter. (F. Xu et al., 2020) used a moving window-based stacked autoencoder with an exponential function to create degradation trends with improved monotonicity compared to the input data. There are also cases where networks such as Convolutional Neural Networks (CNNs) are used to learn custom features that are then used as the input to the RUL estimation model. These custom features learnt from the CNN help improve the accuracy of the RUL estimate when compared to statistical features (Mao et al., 2018). Unsupervised methods that use the sensors as input data as well as the targets may also be employed to construct health indicators. Generally, these networks are trained to reconstruct sensor signals from the machine operating in a healthy state. Afterwards, one may use the reconstruction error between the network outputs and the real sensor values to act as a health indicator. This is explored in more detail in section 2.2.5.

The HIs may indicate the state of degradation of the equipment; however, if the machine is healthy, it is difficult to estimate the RUL from near-constant HI values. Hence, the next step in the machinery prognostics process is to split the data into “Health stages”.

Health Stage Division

Health stage division would then help breakdown the stages of degradation, for example, a basic breakdown of stages would be,

- Healthy: prognostics algorithm not needed yet as the machine is not degrading
- Degrading: a fault or anomaly occurred, and the equipment’s health is now degrading, the prognostic algorithm can now make predictions on the degradation trajectory’s trend and RUL
- Failure: the point where the equipment can no longer be used

Some machinery prognostic models use the transition between health stages as a way of modelling the machine’s health and estimating the RUL. For example, when using Hidden Markov Models (HMMs), the degradation stages are often separated into the three stages mentioned above. The transition probability to the failure stage helps estimate the RUL (Du et al., 2017; Kim et al., 2011). (Kumar et al., 2019) noticed that for their application involving finding the RUL of drill bits from cutting tools, two health stages gave the best overall performance for the HMM. This result was achieved by testing the performance of different HMMs with varying health stages (2-8). Deep learning methods can also utilise health stages to improve the performance or interpretability of the method. (M. Xia et al., 2018) used a neural network to classify the monitored data into different health stages by outputting the probability of being in each health stage. Another network could then be used to output the RUL value for each stage resulting in a probability distribution of RUL values. In this case, health stages helped improve the interpretability of the RUL outputs by providing a probabilistic output, while the stages themselves can give an intuitive idea of how close a component is to failing.

Identifying the Health Stages can aid when training the model for RUL prediction. This is because the model can struggle to make predictions when the machine is operating in a healthy mode if the chosen algorithm assumes the machine starts degrading from the start of its life. An example of this is the popular practice for the Commercial Modular Aero-Propulsion System Simulation (CMAPSS) dataset to clip the RUL so that its maximum value is 130 (Heimes, 2008). This was done because it was found that while the true RUL values were greater than 130, the neural network RUL estimation outputted estimates of steadily around 130. After the true RUL decreased past 130, the estimated RUL also decreased. This is because, during the healthy stage, the sensor data does not change much as everything is operating steadily, so inputting these values in the deterministic neural network will always output the same results. Hence, to account for this, the RUL value of 130 indicates the machine is healthy and effectively acts as a “Healthy” health stage. Once the sensor data starts to deviate from its healthy operating mode, the RUL estimations deviate from 130 and the machine starts to degrade. Another example is using a health indicator to determine when the machine deviates from a normal operating mode and, therefore, when degradation starts. For example, statistical HIs, such as RMS or kurtosis, can be calculated using the bearing vibration data, and the degradation process can be identified once those HIs deviate from a normal value or cross some predetermined threshold (Cao et al., 2023; J. Guo et al., 2023). Similarly, in (X. Liu et al., 2022), a CNN based HI is used instead of vibration statistics to determine the healthy and degrading health stages. This process is often applied to bearing prognostics in the literature. Like with CMAPSS, any point before the degradation start point is treated as a constant maximum RUL, and afterwards, the RUL linearly degrades.

RUL prediction

Finally, RUL predictions can be made using a chosen data-driven algorithm. RUL prediction will generally utilise the HI variables when the equipment is in the degradation stage. This can be done by learning the degradation trend of the machine or directly estimating the RUL using the correlation between the HI and RUL. When learning the HI trend during degradation, the RUL can be found by extrapolating this trend to some threshold value and calculating the time difference between the current time and the time the trend hits the threshold. For example, (Saha et al., 2009; Saha & Goebel, 2008) used a Relevance Vector Machine (RVM) as a regression model that is fit to battery capacity health indicator data and then extrapolated to a threshold. The time between the current time point and the time when the threshold is crossed is the RUL of the battery. (H. Liu et al., 2019) also use a threshold-based approach for RUL estimation but uses a semi-empirical model for voltage degradation of a proton exchange membrane fuel cell. They use an Adaptive Unscented Kalman Filter to update the states and the RUL estimates as more data is gathered. Another example involves using a hidden Markov Model to determine the hidden degradation states based on the sensor signals from the machine. Then the probability of transitioning to the final failure state can help determine the estimated RUL (Q. Liu et al., 2015; Cannarile et al., 2017). Most deep learning methods will instead use the data as inputs to the model, which directly outputs the RUL value of the component.

Taking the example of Machine Learning (ML) methods, the following process can be used to estimate the RUL,

- Features can be extracted from the raw data, e.g. wavelet coefficients, statistical features, etc., which act as HIs
- Health stages can be found, and the degradation stage can be isolated, e.g. using some anomaly detection method to find the point where degradation starts
- The ML algorithm uses HIs in the degradation stage as inputs to the model. It then learns a trend which helps estimate the RUL either by directly outputting the RUL or fitting a curve to the degradation trend and extrapolating it until the HI reaches a threshold value

These steps involve calculating hand-crafted features and splitting the data into health stages. This can require expert knowledge of what features help the ML model learn the system's degradation trend and how to split the data into health stages properly. This is one of the downsides of traditional ML data-driven methods, as they require experience and expertise to utilise them due to these “hand crafted” features (W. Zhang et al., 2019). However, deep learning methods utilise deep neural networks (networks with multiple stacked layers), which automatically learn feature representations of the input data. This property allows the network to automatically learn HIs and health stages needed for RUL estimation as the layers of the network extract the required features from the data. This allows for the prediction of the RUL using only the raw input data without requiring expert knowledge regarding the equipment or process. Deep learning methods will be the focus here due to their generality and ability to automate processes that normally require expert knowledge, such as HI construction. Hence, the following sections will focus on presenting some popular deep-learning models used in machinery prognostics.

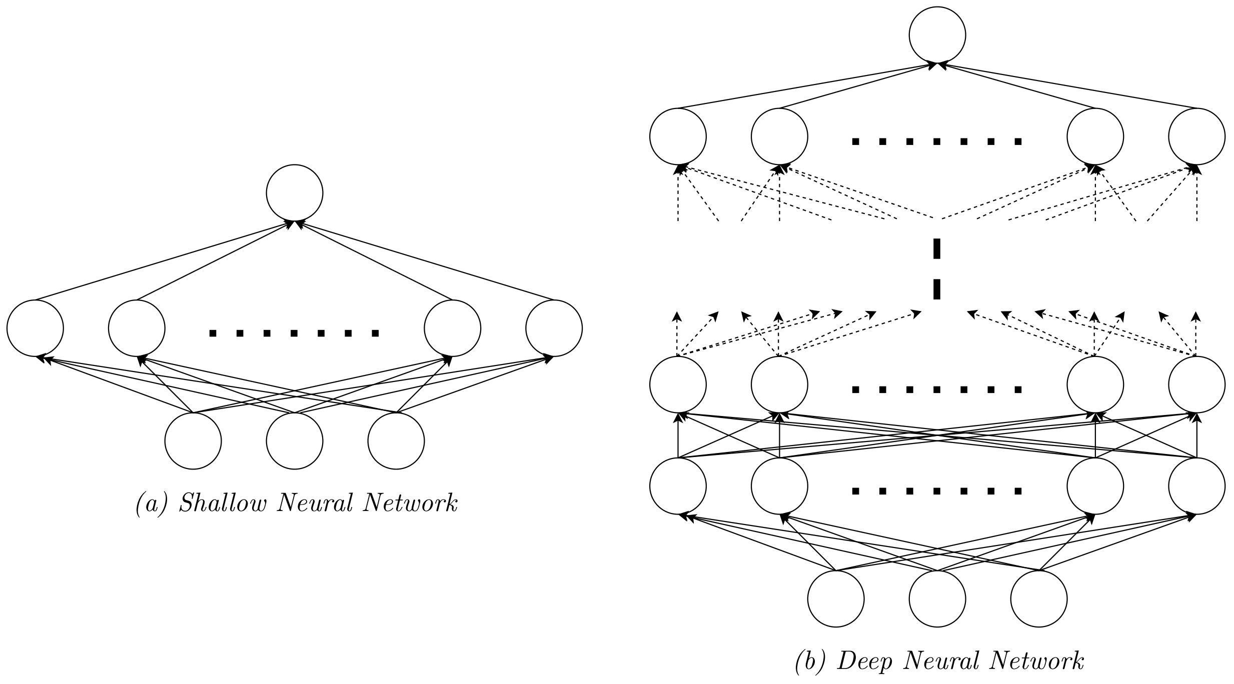
2.2 Deep Learning in Prognostics

As more data is gathered, the effectiveness of deep learning techniques increases making them a popular choice for applications where there is abundant sensor data. This section provides an overview of the different techniques used to estimate the RUL of equipment using deep learning.

2.2.1 Feature Extraction

Often with conventional machine learning techniques, domain expertise is required to take the raw data and construct suitable features for the algorithm to detect patterns between the inputs and desired outputs (Lecun et al., 2015). Deep learning techniques automatically extract features from the raw input data through the use of neural networks, negating the need for careful construction of features that require domain expertise. Neural networks allow for the representation of more complicated nonlinear functions through stacks of more basic nonlinear functions. A deep neural network can be seen as using many network layers or increasing the number of function stacks relative to a shallow neural network to better approximate nonlinear operations. As the inputs go through the network, these stacks can be thought of as extracting features from the data, which eventually gets combined in the end to produce an output. When enough of these operations are stacked together, complicated nonlinear operations can be represented by the network (Hornik et al., 1989), allowing for better approximations of any combination of features needed to produce the desired output. An illustration of a shallow versus deep neural network can be seen in Figure 2.1. Note these simple feedforward networks are often referred to as Multilayer Perceptron (MLP) networks or Artificial Neural Networks (ANNs). Some prognostic procedures mentioned here still use separate feature extraction before inputting the data into the neural network. This can be the case if the raw data alone is insufficient to learn the pattern between the data and the RUL (Y. Wu et al., 2018).

In some simple cases, much like other data-driven methods that do not involve deep neural networks, statistical indicators are extracted and used as inputs to the deep neural network (F. Xu et al., 2020; F. Wang et al., 2019). Sensitivity features that describe the degradation of the component are often extracted for condition monitoring in machinery components. For example, multiple cases can be found where frequency information is extracted using Fourier or Wavelet transforms for condition monitoring applications. In one case (Y. Huang et al., 2010) used the wavelet transform along with further dimensional reduction techniques and a


(a) Shallow Neural Network

(b) Deep Neural Network

Figure 2.1. A diagram showing the difference between a shallow neural network and a deep neural network. The difference simply being the increase in functional stacks for the deep neural network, i.e. the outputs of one stack become the inputs of the next, and this is repeated more times in the deep neural network. The bottom nodes/circles represent the inputs, and the arrows represent the function applied to the inputs and point to the outputs of that function. The top node represents a single final output variable. Note many "deep" neural networks work well with only three layers, and many networks used in this thesis are only three layers deep.

fuzzy c-means clustering method to assess the performance and classify the working conditions of the machinery component. (McKee et al., 2015) used octave band analysis on the frequency information and PCA to determine the most informative components or octave bands in this case. PCA was used to extract features from each chosen octave band and compared with features extracted from healthy pumps using the Mahalanobis distance as a sensitivity parameter to monitor the pump condition. (Lei et al., 2010) used a wavelet transform and empirical mode decomposition on gear vibration signals to extract the fault characteristic information and other general statistical indicators. A gear fault diagnostics algorithm used these parameters as the condition monitoring data. After extracting sensitivity features they can be used to aid in tasks such as diagnostics and prognostics. For example, (Z. Q. Chen et al., 2015) used different statistical and condition monitoring features as input parameters to a convolutional neural network that was then used to identify faults in a gearbox. Wavelet transforms can also compress the signal by representing the signal using wavelet coefficients, which signify the signal's energy. This allows for filtering or denoising the signal by reconstructing the signal using only the high energy coefficients, while the low energy coefficients are assumed to represent the white noise of the original signal (Z. K. Peng & Chu, 2004). Using the resulting filtered signal instead of the noisy raw data as inputs can help improve the overall performance of the deep neural network (Kanarachos et al., 2017). Extracted features can also be used to improve interpretability or lower the dimensionality of the data. An example of this is encoding input values to latent spaces using Restricted Boltzmann Machines (RBMs) through unsupervised learning techniques. (Listou Ellefsen et al., 2019; Ren et al., 2019) both used the RBM to extract features from the input data and then used those latent features in another network to output the RUL. Autoencoders are also used to encode input data and use these latent variables as features for the RUL prediction network (C. Chen et al., 2020). (M. Xia et al., 2018) used an autoencoder to classify the inputs into a discrete number of health stages, determining the probability of being in that stage. Another network could then find the RUL given each of those health stages/features from the autoencoder. The classification into health stages helped with the interpretability of the features as it was directly related to degradation, and probabilities were associated with each health state. (W. Yu et al., 2019) used a bidirectional Long Short-Term Memory (LSTM) network to create a one-dimensional health indicator and curves are fit to this indicator for smoothing. To estimate the RUL, similarity-based techniques are used to compare the current curves to the training curves, and a weighted average of their corresponding RUL values was used as the RUL estimate. (L. Guo et al., 2017) used an RNN on both raw data and extracted statistical features to output a one-dimensional HI, which represented the degradation of the machine. The HI had a range from zero to one, and the RNN was trained so that a HI value of 1 indicated machinery failure. Hence, the HI had an inbuilt threshold value, and in the paper, a simple exponential model was fit to the HI with uncertainty bounds on its parameters. The exponential model could extrapolate the HI to find the time it took for the HI to reach the failure threshold. This model used deep learning to extract a HI, which allowed for a simple model to be used and the incorporation of uncertainty analysis.

2.2.2 Recurrent Neural Network Methods for Prognostics

The data used in machinery prognostics is generally gathered from sensors taking measurements from the equipment at certain time intervals. These sensor signals are time-dependent and are classified as time-series signals. Since these signals are the inputs to the model used for RUL estimation, the RUL output is also dependent on time. Hence, many studies utilise the RNN for RUL estimation due to its feedback architecture being well-suited for analysing time-series signals. A simplified diagram of how the RNN might be used for RUL estimation given sensor data is shown in Figure 2.2.

There are different ways the RNN can be utilised in machinery prognostics, either directly or indirectly estimating the RUL. As mentioned in the feature extraction section, in some cases, statistical features, frequency or time-frequency components may be extracted and used as the inputs to the deep neural network to improve performance. The signals may also be filtered using Fourier, or Wavelet transforms to reconstruct the signal using only the high-energy components found in the frequency or time-frequency transforms (L. Yang et al., 2019). Statistical, frequency and time-frequency features have been used as inputs to the RNN to aid with training the RNN and improving RUL estimation (F. Wang et al., 2019). In some cases, new features are proposed to improve the RNNs performance, (B. Zhang et al., 2019) for example, proposed a waveform entropy feature which helped improve the network performance when used as an input variable to the network. However, deep learning aims to extract features automatically from the raw data and directly output the desired variables. Hence, it was more common to find cases where RNNs were used either on raw data to find the RUL or used as feature extractors themselves while another network or separate technique estimates the RUL. The following paragraph mentions these cases and how they utilise the RNN to estimate the RUL.

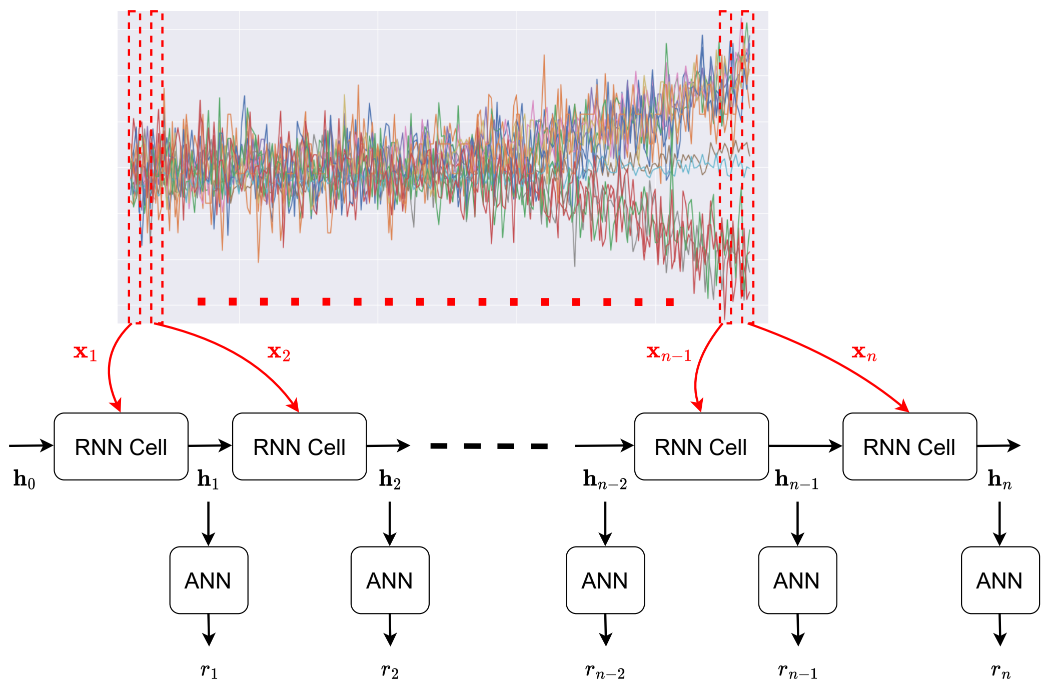


Figure 2.2. This shows an illustration of how an RNN could be used to estimate the RUL. It shows a relatively simple example of taking sensor data at certain time points and using them as inputs to an RNN. The hidden variables go on to be the input variables to an Artificial Neural Network (ANN) whose output is the RUL estimate.

LSTMs are one of the most popular variants of the RNN. Hence, multiple studies use them to estimate the RUL of machinery components using raw sensor data as an input. While many studies use the LSTM on raw data to estimate the RUL, they often differentiate themselves through their methodology or by altering the network to better suit a machinery prognostics context. For example, (W. Peng et al., 2019) used a Bayesian bidirectional LSTM whose parameters are described using Gaussian distributions. This incorporates uncertainty into the LSTM and, therefore, the final RUL output due to being able to sample different parameters from their corresponding distributions and simulate different RUL estimates. Hence, the novelty is not from the network architecture but from how it is used, in this case, to incorporate uncertainty into the RUL estimates. (Aggarwal et al., 2019) also quantifies the uncertainty in their RUL estimates; however, it is done by letting the LSTM output the parameters of a Weibull distribution. The Weibull distribution describes possible RUL values, while the likelihood of the Weibull distribution is used to train the network by evaluating the expression at the target RUL values. (Y. Zhang et al., 2018) inputs multiple possible initial conditions for the battery capacity values into an LSTM and uses Monte Carlo simulation to incorporate uncertainty into the RUL estimation. (Zhou et al., 2019) utilised the LSTM to model the battery capacity trend of a lithium-ion battery; the LSTM can then be used to extrapolate to a threshold value for RUL estimation. (Elsheikh et al., 2019) used a bidirectional handshaking LSTM, which uses the final output of the cell and hidden state for the forward LSTM to initialise the backward LSTM. This improves RUL estimation accuracy when given short observations/sensor data sequences. (J. Wu et al., 2019) also focused on improved RUL estimation accuracy by utilising multiple stacked LSTM networks and optimising the hyperparameters using the grid search method. (C. G. Huang et al., 2019) first used the raw sensor data in a bidirectional LSTM to extract features from the data. These features were then combined with operational condition signals, and the combined signal is the input to another bidirectional LSTM, which produces the RUL output. This method incorporated operating conditions that do not have a natural numeric representation into the network and, therefore, the RUL estimates. (Radaideh et al., 2020) used the LSTM to model different variables monitored for the coolant in a nuclear power plant. This model was used to extrapolate the variables and act as a warning system if any potential accidents could occur due to loss of coolant. While all these studies use the LSTM architecture, they all find novel ways to utilise the architecture to solve another problem in machinery prognostics, e.g. RUL estimation accuracy or uncertainty quantification of the RUL estimates.

An interesting application of RNNs for prognostics is using the data-driven RNN model in conjunction with a physics-based model. (Nascimento & Viana, 2019) used a custom RNN cell that can either work as a standard RNN to model the cumulative degradation of a component or incorporate physics-based differential equation parameters that are optimised during RNN training. Hence, known differential equations that model degradation can have their parameters estimated during RNN training using gathered sensor data. In this case, it was used to model crack length using Paris's law as the physics-based model and the RNN outputs parameters which are normally difficult to estimate accurately. The RNN cell outputs a change in crack length and adds it to the current crack length to model how it changes over time and estimates the RUL life is based on a threshold crack length value. A general diagram of this modified physics-informed RNN cell is shown in Figure 2.3. (Yucesan & Viana, 2019) used a physics-based RNN cell to model the degradation of grease/lubricant in wind turbine bearings. (Dourado & Viana, 2019) modelled corrosion fatigue using a physics-based RNN to estimate the RUL.

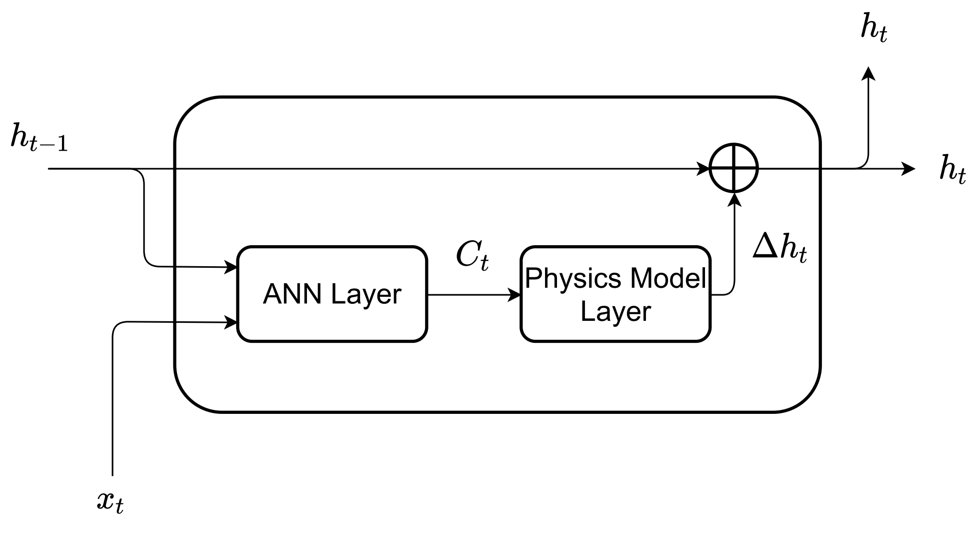


Figure 2.3. A general physics-informed RNN cell. Where x_t represents the inputs to the model, C_t are the coefficients for the physics-based model derived from the neural network structure and input data, h_t is the health indicator the physics-based model outputs and acts as the hidden variable for the RNN.

While RNNs are well suited for tasks involving time-series data, such as sensor signals used in machinery prognostics, they are not the only network used in machinery prognostics. The next section looks at how another popular type of network, CNNs, have been used in machinery prognostics.

2.2.3 Convolutional Neural Network Methods for Prognostics

By themselves, Convolutional Neural Networks are not as common for prognostics compared to the RNN. This is because the RNNs feedback architecture is well suited for analysing time-series data, while the CNNs architecture is more suited for analysing spatial patterns in the data. The CNNs strength in finding spatial patterns makes them widely used in machine vision applications and for analysing image data. Some researchers in prognostics have leveraged this strength by reformatting the sensor signal data to suit the CNN architecture better. The following paragraphs review the CNNs used for prognostics and examples of how data may be reformatted to improve CNN performance when used for RUL estimation.

Many early works using CNN methods for RUL estimation first transform the sensor signals in a grid format using some transform, e.g. Fourier transform. This is comparable to image data which are represented by a grid/s of intensity values, e.g. one grid of grey scale intensities or three grids of red, green and blue intensity values for each corresponding coordinate in the grid. Since CNNs perform well on image data, it is beneficial to restructure the data in a two-dimensional grid format with intensity values at each coordinate, similar to the way image data are formatted. One way of doing this is to extract the time-frequency data from the raw data to create an intensity map from the resulting time-frequency plot. For each time-frequency coordinate, there is an intensity or energy value given by the coefficients from the chosen time-frequency transformation method. This is comparable to image data, where there is a corresponding intensity value for every spacial coordinate. A diagram illustrating the general steps involved in CNN prognostics is shown in Figure 2.4.

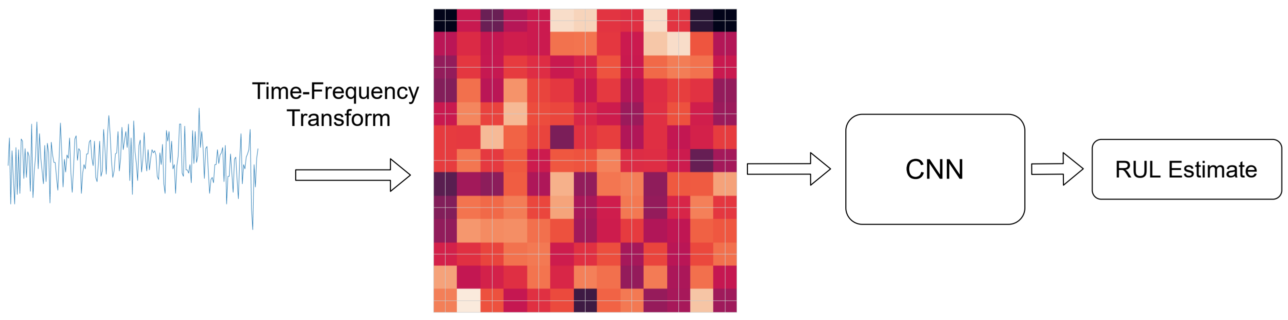


Figure 2.4. The general workflow of the algorithms described in this section. First, the time-series signals (only one shown here) are transformed into a time-frequency spectrogram image where the x-axis is time, the y-axis represents the frequency while the intensity of the colours represents the energy of the signal. This becomes the input to the CNN, which outputs the RUL estimate directly

For example, (Zhu et al., 2019) used the wavelet transform on the input sensor data to transform it into this time-frequency format, and then a CNN was used to estimate the RUL. (X. Li, Zhang, et al., 2019) used a similar method but instead utilised the short-time Fourier transform as the time-frequency transform method. (W. Yang et al., 2020) did not use time-frequency transforms but still applied a transform on the input data to turn it into this "image-like" format by using Empirical Mode Decomposition (EMD). Similarly, (Q. Wang et al., 2019) did not use a time-frequency transform, but instead, a regular Fourier transform to take data into the frequency domain and sectioned the data into bands to construct an $M \times M$ 2D data format. These became the inputs to the CNN used to estimate the RUL. CNNs have also been used without extracting features from the raw data. Instead, time windows are used to create a 2D data format where the data has the dimensions (window length $\times$ feature dimension), which becomes the input to the CNN (X. Li et al., 2018; H. Yang et al., 2019). Hence, to use a CNN effectively, some data augmentation is required, i.e. the input data is transformed into a 2D input which is similar to how image data is structured. This helps improve the performance of the CNN as the data is structured in a way the CNN can handle due to the nature of its architecture. Feature extraction may also be needed, as was the case when wavelet or Fourier transforms were used, depending on how deep the CNN is (which could be limited due to the computing power available) and how much data is available.

Multiple CNNs can be used to split the tasks required for an effective overall prognostics method. (B. Yang et al., 2019) used one CNN on the data to identify if the equipment was degrading. Once the first classifier CNN identifies degradation is occurring, another is used to estimate the RUL. This helps improve performance as the network is more effective at determining the RUL and the degradation trend of equipment when the input data corresponds to the degrading stage of the equipment. So far, all the methods relied on the fact that CNNs are good at identifying spatial patterns in the 2-dimensional grid input data. In the case of prognostics, the grid dimensions are generally a time dimension and a feature dimension, such as the frequency or the raw data dimension, which indicates the component's health. (B. Wang, Lei, Li, et al., 2019) used two 1-dimensional convolution operators acting in each dimension (one in the time dimension and the other in the feature dimension). This enables the CNN to learn the temporal and spatial features separately and improve the RUL estimates.

Combining the CNN and RNN can help provide benefits from both of the networks, i.e. the RNNs ability to deal with time-series data and the CNNs ability to extract spatial features. Some works have used CNNs to extract features and then used those features as an input to an RNN for RUL estimation (Jiang et al., 2019; Kong et al., 2019; R. Zhao et al., 2017; An et al., 2020). The RNN and CNN can also be combined into one network known as Recurrent Convolutional Neural Network (RCNN) which can be used to model the RUL and extract features all in one network (B. Wang, Lei, Yan, et al., 2019).

2.2.4 Attention Mechanisms

Recently Attention mechanisms have become more popular for extracting features from raw data and using these to aid in RUL estimation. Self-Attention is what will be focused on here, as that is what all these methods have been using. Self-Attention takes the input data X and uses linear projections to map them to attention weight matrices Q, K and V. These are then used to calculate the self-attention matrix W_A and the output known as a "head". i.e.

$$Q = XW_Q,\ K = XW_K,\ V = XW_V \tag{2.1}$$

$$W_A = \frac{\exp\left(\frac{QK^T}{\sqrt{d}}\right)}{\sum \exp\left(\frac{QK^T}{\sqrt{d}}\right)} \tag{2.2}$$

$$\text{head} = W_A V \tag{2.3}$$

where W_Q, W_K and W_V are parameters that can be optimised during training, and d is the dimension size of X that helps normalise the attention mechanism for numerical stability. A multi-head attention mechanism calculates multiple heads using different weight matrices W_Q, W_K and W_V, and then concatenates the heads to output multiple extracted features from the input sequence X. This is similar to how CNNs may use multiple filters to scan the input image and therefore output multiple features that have been extracted from that image. The attention matrix is used as a weight matrix to weigh certain variables in a sequence that contribute most to the output. When training the network, W_Q, W_K and W_V are optimised in such a way to achieve this.

Generally, many methods tend to employ a network that can extract temporal information first before applying the attention mechanism. The attention mechanism itself doesn't have a way to understand the importance of the input data ordering, and so to capture the temporal relationships of the input signal a network such as an RNN is used first. (Ragab et al., 2021) incorporated the attention mechanism after an LSTM to weigh the important hidden variables in the sequence and focus on the important time points for decoding sensor sequences which forecast the future sensor signals. (Z. Chen et al., 2021) also used an attention mechanism after an LSTM to extract features from the sensor signals that complemented other handcrafted features, which were concatenated to the features extracted from the attention mechanism. (J. Zhang et al., 2022) applied a bidirectional Gated Recurrent Unit (GRU) and then used an attention mechanism to find the weighted outputs of the hidden variable outputs, which are used to estimate the RUL. Similarly, (Nie et al., 2022) applied a CNN and a bidirectional LSTM followed by an attention mechanism to extract features from the sensor signals gathered from cutting tools. (Qin et al., 2022) uses a temporal CNN to extract the temporal features and an attention mechanism afterwards to determine the important features. (Remadna et al., 2022) applied a CNN on time windowed data and an attention mechanism on its outputs to select important features extracted by the CNN to construct a latent space that acts as a HI. In contrast to these methods, (H. Liu et al., 2021) used the attention mechanism on the sensor signals to identify important features and create a weighted input for the network that estimates the RUL. While in (Cao et al., 2023), attention mechanisms were used on the sensor signals before a GRU was applied and then another attention mechanism was used afterwards. Finally, (D. Xu et al., 2022) proposed a dual-stream attention mechanism that uses as inputs both the sensor data and the difference between the sensor signal and the initial value of the signal. This difference between the signal and its initial value indicates how the machine has deviated from its initial healthy state. The features extracted from the multi-headed attention mechanism are fed into a MLP network to estimate the RUL. Here no network was used to capture the temporal nature of the time-windowed data; instead, the attention mechanisms were simply applied to the time-windowed signals in a noncausal fashion.

It can be seen that many attention-based prognostic methods utilise attention to help extract more useful features from the input sensor signals. This often aids in RUL estimation or HI construction. The most common way these methods utilise attention mechanisms is to find a weighted time-series sequence where the weights highlight the data at important points for estimating the RUL. However, some methods use attention mechanisms to find weighted input features that can then be the input to a network that will extract further features or estimate the RUL. Either way, it can be seen that these methods use attention mechanisms to find weighted features, which can be used to find a weighted average that can act as a feature for RUL estimation. Intuitively the effectiveness of this makes sense as more traditional prognostic methods might use extracted statistical features to calculate a RUL estimate. Many statistical methods rely on calculating some average of the signal e.g. RMS calculates a mean of the squared input values, the variance is the mean of the difference between the signal and the signal's mean. Hence, the attention mechanism acts like a custom weighted average that is optimised for RUL estimation purposes, and so perhaps it is not surprising that it is effective at extracting features for RUL estimation.

2.2.5 Generative models

As mentioned in Section 1.3 a generative model learns to generate samples similar to the input data it was trained on. Sampling from the input data space is done indirectly by sampling from a latent space first and then decoding this latent variable to the input domain. Generative models in prognostics do not generally function in the same way that many of the models described in the previous sections do. In the previous sections, many of the models output the RUL estimate conditioned on the sensor variable inputs. Training the generative model with sensor signal inputs would result in a model that learns how to sample novel sensor signals. Hence, many generative models used in machinery prognostics are utilised for sensor reconstruction and the ability to acquire a latent variable representation of those sensors. This section covers two major areas in which generative models are in machinery prognostics. These two categories use generative models to construct HIs or train in a semi-supervised fashion to reduce the amount of target data needed.

Health Indicators

To construct health indicators using generative models, some works trained a generative model on data from a healthy machine, this meant the model could reconstruct the sensor data from a healthy machine. If the sensor data were from a degrading machine instead of a healthy one, the reconstruction loss would increase. Hence, this reconstruction error could be used as a HI. (Zhai et al., 2021) trained a Variational Autoencoder (VAE) to reconstruct sensor values and used multiple reconstruction loss metrics to construct health indicators for the system. They also conditioned the VAE on the operating condition information; this allowed different future sensor trajectories to be simulated under different planned operating conditions. (González-Muñiz et al., 2022) also used an error value as a HI, but instead of the reconstruction error, they used the error between the latent variables describing a healthy machine and the latent variable from the machine's current status. A Generative Adversarial Network (GAN) can also be employed for HI construction through training on data from healthy machines and then using its discriminator network. The discriminator of the GAN outputs a value from 0 to 1 and represents the probability the data is from the dataset the GAN is trained on. Hence, values close to 1 indicate the data is from the healthy operating region of the machine and as it deviates due to the machine degrading, the values from the discriminator network move from 1 to 0. Therefore, the discriminator output variable can be used as a HI where 1 represents a healthy machine and 0 is close to failure (Que et al., 2019).

Another way of constructing a HI using generative models is through the latent variables constructed during the training of the VAE. This latent variable can be constructed to give it desirable properties for HIs. (Wei et al., 2021) uses a conditional dynamical VAE, which is conditioned on the sensor values and outputs the RUL. The RUL is dependent on the latent variable, and this allows the latent variable to act as a HI due to its relationship with the RUL. Ping et al., 2019 uses a log-Normal distribution instead of the normal distribution, which is commonly used to describe the latent space of a VAE. Compared to the normal distribution, a latent variable which is log-Normally distributed, is better suited to model the asymmetry of the degradation mechanics. (T. Yang et al., 2020) used an LSTM network as the encoder in the VAE framework. This allowed the VAE to capture temporal dependencies and patterns using the LSTM, which is well suited to that task. (Remadna et al., 2022) also uses the latent space of a VAE as the HI. However, instead of an LSTM, they use a CNN along with an attention mechanism as the network used to construct the VAE. Helped extract features from the sensor data while the attention mechanism weights certain parts of the time sequence more than others making it well suited to handle time series data.

Semi-supervised models

Obtaining enough failure data is one of the important challenges to overcome when considering deep learning methods for machinery prognostics (Eker et al., 2012). One way to reduce the amount of failure data required is to use semi-supervised learning. Semi-supervised learning uses unsupervised learning to learn patterns from the input data, such as sensor signals, then utilises this knowledge in various ways to assist a supervised learning stage that uses the target data (RUL data in this case). For example, a VAE could be trained on the sensor signals, and the latent variables from the encoder could be used as the inputs of another network that estimates the RUL. The idea is that the unsupervised portion of the learning can be done on just the sensor signals, which can be easily gathered compared to the RUL target data that requires a machine to be run until failure. Then when training the network for RUL estimation, part of the work was already done during the unsupervised stage, so less RUL data is required to optimise the network.

(Su et al., 2020) uses a VAE to find a latent variable representation of the input sensor data. They apply an LSTM on those latent variables during the supervised stage to estimate the RUL. (K. Yu et al., 2021) has a similar technique but uses a temporal-CNN to estimate the RUL from the latent variables during the supervised stage. Other methods continue to train the encoder during the supervised stage. (T. Wang et al., 2021) trains a VAE in an unsupervised manner as the previous methods did. However, the previous methods used the latent variable outputs from the VAE's encoder and did not train the VAE further during the supervised stage. Here, the encoder of the VAE was further trained during the supervised stage; the unsupervised stage acts as a pretraining stage, while the supervised stage further tweaks the encoder network for the RUL estimation task.

Some methods involve VAEs which account for the data's temporal nature and are specifically designed to deal with sequential data. (Martin & Droguett, 2022) creates a temporal VAE (tVAE), which can be trained to find latent space dynamics and sequences of latent variables. These latent variables can be used as inputs for the supervised portion of the training to train a network to estimate the RUL. (Verstraete et al., 2019) also constructs a VAE method designed to learn latent dynamics and is better suited to deal with sequential data than the standard VAE. They train the VAE using adversarial techniques and, like the other methods, use the latent variables to train the RUL estimation network during the supervised training phase.

2.2.6 Uncertainty Considerations in Deep Learning Methods for Prognostics

One of the major pitfalls of using deep learning methods for prognostics is the lack of a natural way of incorporating and quantifying the uncertainty of RUL estimates. Generally, deep learning methods used in prognostics usually rely on some extra step or modification that enables uncertainty to be incorporated into the estimated RUL. This can be contrasted to statistical methods, which incorporate uncertainty representation and quantification as a part of the model itself. Uncertainty quantification is often neglected in deep learning methods for machinery prognostics (W. Peng et al., 2019) hence, estimating uncertainties for RUL prediction is generally seen as a challenge and a research focus for machinery prognostics (Lei et al., 2018). This section of the review aims to first briefly look at the different methods used for uncertainty quantification in more traditional machinery prognostics techniques and then review how different deep learning methods have quantified the uncertainty in their RUL estimates.

Uncertainty in Traditional Data-Driven Methods

Bayesian methods are popular for uncertainty quantification in more traditional data-driven methods used in machinery prognostics. Due to the lack of multiple test cases, the frequentist interpretation is generally unsuitable for machinery prognostics. Instead, data-driven condition-based monitoring techniques focus on analysing the states or sensor values at any point in time for a single piece of equipment. Hence, the uncertainty of the RUL estimate depends on the health indicator's uncertainties. As more measurements arrive, the uncertainty of RUL estimates would therefore change and need to be updated. Bayesian approaches can be used for these uncertainty updates and therefore are well suited for machinery prognostics (Sankararaman, 2015). There are many cases where Bayesian techniques are used for state estimation and uncertainty quantification in machinery prognostics methods (such as the Kalman or Particle filter). These methods may use some mathematical model (e.g. a machine learning model) to model the dynamics (state transition model) of a HI and a particle filter for state estimation when new data is gathered. The particle filter can then be used along with the model to extrapolate and propagate the uncertainties forward in time to a threshold value that will define the RUL. Table 2.1 summarises different ways Bayesian filters, such as the particle and Kalman filter, have been used to estimate the RUL uncertainty.

Uncertainty in Deep Learning Methods

Deep learning methods that quantify uncertainty do not tend to model a HI trajectory and then represent uncertainty through forecasting uncertainty as the methods mentioned in the previous section did. Many methods stick to the strength of the deep learning model, being able to estimate the RUL from the network's output directly. Hence, these methods tend to try techniques such as having the network output parameters for a distribution that describes the current RUL, or accounting for uncertainty in the neural network model parameters. However, there are cases, such as in (L. Guo et al., 2017; Z. Xu et al., 2021), where the neural network does output a HI and then applies a probabilistic curve fitting technique to the HIs to quantify uncertainty.

One method of incorporating uncertainty could be through the sensor or measurement uncertainties. If measurement uncertainties are known through a probability distribution, this distribution can be sampled, and multiple RUL estimates can be found using Monte Carlo simulation. (Y. Zhang et al., 2018) used Monte Carlo simulation along with an RNN to simulate battery capacity trajectories for a Li-ion battery. These trajectories can be used to extrapolate to some threshold battery capacity value, and the RUL is estimated from that. In many cases,

Table 2.1: Traditional methods which estimate RUL uncertainty using state transition models and Bayesian filters

Author Reference	State Transition Model	Filter	Description
(Saha & Goebel, 2008)	RVM	Particle Filter	Used the model to propagate the PF weights to a threshold and find the RUL distribution
(Saha et al., 2009)	RVM	Particle Filter	Compared different PFs used with the state transition model to estimate RUL distributions
(Y. Hu & Luo, 2012)	RVM	Particle Filter	RVM is used when the dynamics of the machinery degradation is unknown, and the state is updated with a PF
(Cheng et al., 2018)	ANFIS	Particle Filter	Used ANFIS as the model and PF to estimate the RUL distribution
(Cheng et al., 2019)	ANFIS	Particle Filter	Used a novel particle resampling method to improve PF state estimation and extrapolate particles to the threshold
(Bai & Wang, 2016)	Neural Network	Particle Filter	Updates the weights of the network model instead of the HI states. Different weights can be sampled, and the models extrapolate the HI trajectory to find the RUL distribution
(Tse et al., 2019)	Linear State Space model	Kalman Filter	The normal distribution of the Kalman filter is sampled from, and the model propagates the states to the threshold value

however, the uncertainty of the sensor inputs is not known, and we do not have a physical health indicator with a known threshold, like the battery capacity. Some methods train the neural networks to act as state transition and measurement likelihood models to get around this. The state transition can describe the dynamics for some latent HI, and the likelihood model describes the relationship between the RUL and this latent HI. (Deutsch et al., 2017), used a Deep Belief Network (DBN) as a state transition and measurement function for a particle filter which updates the probability distribution for the RUL estimate as new measurements are gathered. Similarly to Monte Carlo methods, ensembles of neural networks can be trained with different criteria in mind. Bagging or weighted RUL estimates can be made, representing the uncertainties of the RUL prediction (M. Xu et al., 2020). Generative models can also simulate possible outcomes and ultimately estimate the uncertainty of the RUL. Many generative models designed to output a HI were discussed in Section 2.2.5. Each of these would output a probability distribution of the latent variable and sample different possible outcomes from the distribution. After sampling, these could be decoded, synthetic sensor values could be created, and various HI values could be calculated by comparing the generated sensor values to the current ones. Such as in (Zhai et al., 2021) where the HI trajectories for different operating conditions could be simulated based on a VAE trained on healthy sensor data.

A popular sampling-based method for quantifying uncertainty in deep learning models is the Monte Carlo Dropout method (Gal & Ghahramani, 2016). This method has been used in many instances within the prognostics literature and is relatively simple to implement as it incorporates uncertainty through the dropout mechanism in a neural network. When training a neural network, dropout can be used to prevent overfitting; after training, dropout is normally switched off, making the network fully deterministic. Monte Carlo dropout leaves the dropout mechanism in place even after training, allowing multiple output samples to be generated by evaluating the network multiple times. This has been used in different prognostic applications such as quantifying the RUL uncertainty of bearings (Cao et al., 2023). It is also useful in predictive maintenance if the goal is to ultimately construct a policy for when to maintain machines. The models that estimate the RUL play an important role in designing a maintenance policy as these rely on the uncertainties of the RUL estimates. Hence, Monte Carlo dropout has been used to allow powerful deep learning models such as CNNs to express uncertainty and improve the RUL prediction accuracy and, therefore, the overall performance of the maintenance policy itself (Mitici et al., 2023; Lee & Mitici, 2023).

There are non-sampling-based methods that instead let the network output a known distribution type, e.g. Gaussian, and use that distribution to approximate the RUL distribution. This can be applied to deep learning methods by letting the network output the parameters of a distribution describing the RUL uncertainty. The distribution can be used to find the likelihood that the target RUL value falls in this distribution which becomes the loss function for training the network. Variational inference methods might restrict the model by only being able to represent a specific simplified distribution when compared to Monte Carlo methods. However, they have reduced computational costs as they only need to output a small number of parameters instead of doing computations on a large number of samples. While generative models may use variational inference (such as the VAE), not all methods use a generative model to estimate a probability distribution. The models discussed here are discriminative models that describe a conditional distribution $p(RUL|sensors)$ as opposed to the generative model, which models a joint distribution $p(Z, X)$. (Rigamonti et al., 2018) trained a type of RNN known as the Echo State Network to output the mean and variance of a Gaussian distribution which is used to describe the uncertainty of the RUL estimate. Similarly, (B. Wang, Lei, Yan, et al., 2019) let the output of their Recurrent Convolutional neural network be the mean and variance of a Gaussian distribution to describe the RUL. Instead of using a latent Gaussian distribution approach, (Aggarwal et al., 2019) used a more expressive distribution which is better suited for the prognostics task to directly describe the RUL uncertainty. They used an RNN which outputs the parameters for a Weibull distribution whose likelihood is the loss function.

Bayesian neural networks are another model type that can represent uncertainty in the RUL estimate. Instead of outputting a known distribution, Bayesian neural networks represent their parameters (weights and biases) using Gaussian distributions. During training, the parameters can be updated using Bayes' rule as they are represented by distributions and are updated due to new measurements. (W. Peng et al., 2019) focused on altering popular networks used in machinery prognostics (e.g. LSTM and CNNs), turning them into Bayesian networks to incorporate uncertainty into their estimates.

2.3 Conclusions

This literature review section has focused on the different deep learning methods used for machinery prognostics as well as how deep learning methods have incorporated uncertainty into their RUL estimates. This review focused on deep learning models used in machinery prognostics. To that end, some network architectures were explored due to their use in machinery prognostics, namely the RNN and CNN. However, most of the methods described in those sections tended to estimate a point RUL estimate instead of accounting for the uncertainty in those estimates. Many of the models described were discriminative models that focused on neural networks that described a conditional relationship between the RUL and sensor inputs. Generative models differ in the sense they describe a conditional distribution between the network inputs and a latent variable defined during training. As a result, the generative models discussed could often not directly estimate the RUL. Instead, they would be used to construct HIs either through their latent variables or by reconstructing the input sensor data and comparing it to the available sensor data. Another use of generative models was in semi-supervised learning. Here, the generative model was often used as the unsupervised model in semi-supervised training to aid the training of the supervised model that estimated the RUL. However, while generative models are probabilistic by design, many methods did not directly look at probabilistic RUL estimation that could compete with the current state-of-the-art methods. Hence, the review also focused on uncertainty analysis in deep learning methods. It first stated that in traditional data-driven methods that do not use deep neural networks, many methods use Bayesian filters such as the Kalman or Particle filter to quantify the uncertainty of the RUL. Some deep learning methods use this strategy where a deep neural network learns the dynamics of a known HI and simulates many HI trajectories to a threshold to estimate the RUL distribution. However, this HI is not always known, and various options are available if this is true. Bayesian neural networks were an option that made the neural network itself probabilistic by defining the parameters of the network using probability distributions instead of point values. One could also construct a neural network that could define a HI deterministically but use a probabilistic curve fitting technique to extrapolate to a known threshold. Similarly, HI could be constructed using generative models, and various HI trajectories could be simulated to a threshold. Finally, neural networks that output the parameters to describe a distribution, such as a Weibull distribution, were also identified as a way deep learning methods in machinery prognostics have quantified the uncertainty of the RUL. However, few methods combine state-of-the-art performance on RUL estimation tasks while providing interpretable results, e.g. by constructing HIs.

This thesis will focus on exploring deep learning methods whose primary purpose is to estimate the uncertainty of the RUL estimates and improve the neural network's interpretability. Both uncertainty and reducing the black-box nature of these deep neural networks are identified as areas where deep learning models for machinery prognostics could improve. Unlike the HI models presented in this review, this thesis does not specifically focus on the HI aspect of the networks. Instead, the primary focus is on the RUL estimation while making the network more interpretable or the theory more accessible so that the practitioner can adjust and utilise the model, knowing how the outputs may be affected. However, first, it is important to give the necessary technical background for introducing these methods in this thesis.

Chapter 3

Background

This section will introduce the theory required for the main portion of the thesis. Some of the more basic aspects of deep learning are assumed to be known and won't be covered. For example, standard feedforward or MLP networks and optimising deep neural networks with methods such as Stochastic Gradient Descent (SGD) or ADAM won't be covered. It is assumed the reader is familiar with the basics of what neural networks are and the training procedure required to optimise and use that network for inference. Hence, in this background, the Variational Autoencoder will be covered simply as a way of covering generative models through a concrete example and because the theory is expanded to the sequential case in the Dynamic Variational Autoencoder section. Neural Differential Equations are a more advanced type of network architecture that is used throughout the thesis and hence will be covered here. Finally, some Bayesian filtering theory will be covered due to its similarity to the variational autoencoder models previously mentioned and it's use in some of the later sections. Bayesian filters also allow us to work with a principled way to develop the framework used to construct these variational autoencoder models and expose some of the logic behind the design decisions made during their construction.

3.1 Variational Autoencoders

Variational Autoencoders (VAEs) are a generative deep learning method that aims to sample from the original data distribution $p(X)$, where X is the input data. However, $p(X)$ is generally unknown. To get around this, generative models represent the probability distribution $p(X, Z)$ where Z is a latent variable. This works by separating the generative model into its component parts, $p(X, Z) = p(X|Z)p(Z)$, hence, sampling the latent variable $Z^{(i)} \sim p(Z)$ then "decoding" using $X \sim p(X|Z^{(i)})$ would indirectly sample from the original data distribution $p(X)$. The VAE uses an auto-encoding architecture to find a valid decoding distribution $p(X|Z)$ by training a probabilistic auto-encoder. This means an encoder, $p(Z|X)$, is utilised in the training of the VAE to learn an encoding that connects the latent variable Z to the input domain X. A problem arises when using Bayes' theorem, it can be shown that,

$$p(Z|X) = \frac{p(X|Z)p(Z)}{p(X)}. \tag{3.1}$$

Using the total law of probability and noticing $p(X)$ is a normalising constant it can be shown,

$$p(X) = \int p(X|Z)p(Z)dZ. \tag{3.2}$$

As originally stated, $p(X)$ is unknown and now it can be seen via Bayes Theorem that it can be found through an integral, however, it is often intractable (too difficult to solve directly), hence, $p(Z|X)$ cannot found directly. This creates a "Catch-22" where we need $p(Z|X)$ to find $p(X)$ and sample from it, but $p(X)$ is needed to find $p(Z|X)$. The VAE deals with this problem by using variational methods to approximate $p(Z|X)$ with a simpler distribution $q(Z|X)$ (e.g. a Gaussian distribution) and trains a neural network to find $q(Z|X)$. The KL-divergence between $q(Z|X)$ and $p(Z|X)$ is used as a criterion to encourage the distributions to match each other. This criterion cannot directly be solved and an estimate known as the Evidence Lower BOund (ELBO) (also known as the Variational Objective) is used which indirectly optimises the criterion (Kingma & Welling, 2014) and is stated as,

$$\mathcal{L}(\theta_X, \theta_Z, \phi, X) = -\mathbb{E}_{q_\phi(Z|X)}\left[\log p_{\theta_X}(X|Z)\right] + D_{KL}\left(q_\phi(Z|X)||p_{\theta_Z}(Z)\right). \tag{3.3}$$

Where D_{KL} is the KL-divergence operator that provides a metric of how "close" the distributions $q_\phi(Z|X)$ and $p_{\theta_Z}(Z)$ are. Mathematically the KL-divergence operator can be stated as,

$$D_{KL}\left(q(X)||p(X)\right) = \int q(X)\log\frac{q(X)}{p(X)}dX. \tag{3.4}$$

The ELBO loss can alternatively be derived by minimising, $-\log\, p(X)$, as we previously established $p(X)$ and $p(Z|X)$ are connected and both optimisation criteria lead to the above approximation (Eqn. 3.3). Note minimising $-\log\, p(X)$ was the main objective of the generative model, as it is effectively the same as maximising the likelihood that a sample X belongs to the data distribution $p(X)$. Hence, minimising the ELBO is an indirect way of solving the integral in Eqn. 3.2. A VAE that is trained by minimising the ELBO is therefore ultimately trying to maximise the probability that X is sampled from the distribution $p(X)$.

From the theory so far it can be seen that there are three main components to the ELBO,

1. Decoder network: $p_{\theta_X}(X|Z)$
2. Prior: $p_{\theta_Z}(Z)$, for a VAE this is usually defined as a standard normal distribution $\mathcal{N}(0, I)$
3. Encoder network: $q_\phi(Z|X)$

The first part of the ELBO loss function involving the expectation on $p_{\theta_Z}(X|Z)$ helps optimise the network parameters to improve reconstruction accuracy; this allows latent variables to be mapped back to the input space accurately. The second part involving the KL-divergence optimises the network parameters so that the inference distribution $q_\phi(Z|X)$ closely matches the prior distribution describing the latent space $p_{\theta_Z}(Z)$. When training a generative model, the goal is generally to generate new inputs, X, not seen during training. Hence, by constraining the latent space to the prior ($p_{\theta_Z}(Z)$) during training, at test time, the prior can be sampled from and decoded to generate new inputs, i.e. sample latent variable $Z \sim p_{\theta_Z}(Z)$ then decode using $X \sim p_{\theta_Z}(X|Z)$. This second loss term also helps generalise the model by forcing the latent variables to follow a specific structure defined by the prior. Note, prior and decoder represent a joint distribution $p_{\theta_Z}(Z, X) = p_{\theta_Z}(Z)p_{\theta_Z}(X|Z)$. Since they are used for generating new data this joint distribution is referred to as the generative model, while $q_\phi(Z|X)$ is known as the inference model. Hence, the aim of the VAE can be seen as attempting to learn a joint distribution $p_{\theta_Z}(Z, X)$ to act as a generative model that can sample from $p(X)$. The workings of the VAE training and generative model are shown more intuitively using the sketch shown in Figure 3.1.

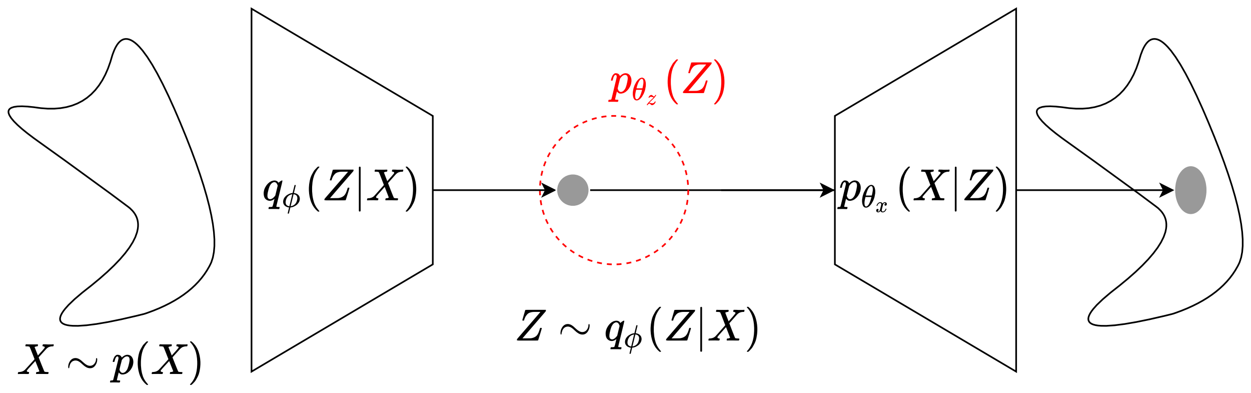


Figure 3.1. The VAE during training can be used to encode to a section of the prior distribution, during testing, the prior distribution can be sampled from (i.e. sample from the entire latent space) and this approximates a local Gaussian distribution in the input space $p(X)$ which can be sampled from. To sample from a larger area of $p(X)$ it can be seen that this can be done by sampling multiple latent variables via $Z \sim p(Z)$ and decoding them all to form local Gaussian approximations covering the distribution $p(X)$.

3.2 Dynamical Variational Autoencoders

The VAE is generally used to generate static data X, e.g. generating new images similar to the input images used to train the VAE. To generate sequential data that involve some underlying dynamics, Dynamical Variational Autoencoders (DVAEs) can be used instead to capture the underlying dynamic structure and is better suited to generate sequential data. An in-depth review of DVAEs from which this section borrows a lot of the terminology can be found in (Girin et al., 2021). DVAEs essentially replace the static input and latent variables (X and Z) of the VAE and replace them with sequences $X = x_{1:T}$ and $Z = z_{1:T}$. Hence, using the same structure as the VAE, the generative and inference models would be,

$$\text{Generative model: } p_\theta(X, Z) = p_\theta(x_{1:T}, z_{1:T}) \tag{3.5}$$

$$\text{Inference model: } q_\phi(Z|X) = q_\phi(z_{1:T}|x_{1:T}). \tag{3.6}$$

This isn't helpful as we may not always want to generate sequences of length T. To gain more flexibility the transition distribution describing the dynamics of the sequence would be helpful to generate custom length sequences. Hence, Eqns. 3.5 and 3.6 can be broken down as,

$$\text{Generative model: } p_\theta(x_{1:T}, z_{1:T}) = \prod_{t=1}^{T} p_{\theta_x}(x_t|x_{1:t-1}, z_{1:t}) p_{\theta_z}(z_t|x_{1:t-1}, z_{1:t-1}) \tag{3.7}$$

$$\text{Inference model: } q_\phi(z_{1:T}|x_{1:T}) = \prod_{t=1}^{T} q_\phi(z_t|x_{1:T}, z_{1:t-1}). \tag{3.8}$$

Notice, from the generative model, the decoder and prior are found in the product (decoder: $p_{\theta_x}(x_t|x_{1:t-1}, z_{1:t})$, prior: $p_{\theta_z}(z_t|x_{1:t-1}, z_{1:t-1})$). Analogous to the VAE, where the inference model is optimised to encode to a prior latent space $p(Z)$, for the DVAE case, the inference model encodes to match some prior transition dynamics $p_{\theta_z}(z_t|x_{1:t-1}, z_{1:t-1})$. An important note is that the inference model is dependent on $x_{1:T}$ for all previous times t, as the model can not be factorised further to remove this dependency. Hence, even if the generative model is causal (current time values are only dependent on previous time values), the inference model is non-causal, (future values are needed to estimate the current time values). Since the inference model is only used during training, this is not a problem as the entire training data is available, therefore, the future values are available.

To train the DVAE, the new generative and inference models can be substituted into Eqn 3.3, which results in the new loss,

$$\begin{aligned}\mathcal{L}(\theta_x, \theta_z, \phi, x_{1:T}) = &-\sum_{t=1}^{T} \mathbb{E}_{q_\phi(z_{1:t}|x_{1:T})} \left[\log p_{\theta_x}(x_t|x_{1:t-1}, z_{1:t})\right] \\ &+ \sum_{t=1}^{T} \mathbb{E}_{q_\phi(z_{1:t-1}|x_{1:T})} \left[D_{KL}\left(q_\phi(z_t|z_{1:t-1}, x_{1:T}) || p_{\theta_z}(z_t|x_{1:t-1}, z_{1:t-1})\right)\right]. \end{aligned} \tag{3.9}$$

Note a full derivation for a DVAE which estimates the RUL can be found later in Chapter 5 in Eqn. 5.12.

3.2.1 DVAE Structure and Implementation

This section will cover some common assumptions and implementations that can be utilised to construct a DVAE in practice. The first practical challenge is dealing with sequential dependencies such as $x_{1:T}$, i.e. how can a sequence of variables such as $x_{1:T}$ be represented as an input to a neural network. Most DVAEs use RNNs to achieve this, bypassing the sequence into an RNN, where the internal state is a representation of the sequence so far. Basically if x_t is the input to the RNN cell and h_t is the internal state generated, then h_t represents $x_{1:t}$ as illustrated in Figure 3.2. Note, here t denotes the current time point which is arbitrary in these examples and can be any value between 1 and T, as the total sequence is denoted as $x_{1:T}$.

Another consideration when implementing DVAEs in a practical setting is that simplifying assumptions can be applied. This is because the full general form of the DVAE stated in Eqns

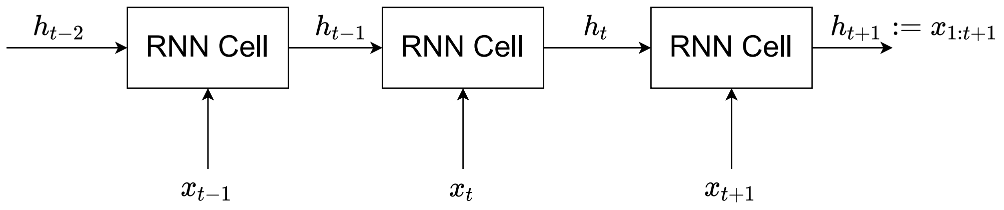


Figure 3.2. An illustration of an RNN being used to represent a sequence of inputs which can be used in a DVAE model

3.7 and 3.8 are not always necessary. For example, the Markov assumption can be made, which means the current variable will only depend on the previous time variable (instead of the entire previous sequence) e.g. $p(z_t|z_{1:t-1}, x_{1:t}) = p(z_t|z_{t-1})$.

It is helpful to show a quick example that illustrates how to design and implement a DVAE given certain assumptions. Consider the following generative and inference model,

$$\text{Decoder: } p_{\theta_x}(x_t|x_{1:t-1}, z_{1:t}) = p_{\theta_x}(x_t|x_{1:t-1}, z_t) \tag{3.10}$$

$$\text{Prior: } p_{\theta_z}(z_t|x_{1:t-1}, z_{1:t-1}) = p_{\theta_z}(z_t|x_{1:t-1}, z_{t-1}) \tag{3.11}$$

$$\text{Encoder: } q_\phi(z_t|x_{1:T}, z_{1:t-1}) = q_\phi(z_t|x_{1:T}, z_{t-1}) \tag{3.12}$$

Recall the encoder is noncausal and therefore requires the entire sequence $x_{1:T}$ as an input, while the other models only require variables related to the current or previous time points.

Figure 3.3 is a diagram showing the variables in the models and their dependencies. The stochastic variables in the DVAE (such as z_t) are achieved by making the neural network output a mean and log-variance, which describe a Gaussian distribution. Hence, the generative and inference models can be represented by neural networks as follows,

$$p_{\theta_x}(x_t|x_{1:t-1}, z_t) := f_{\theta_x}(h_t, z_t) = [\mu_{x_t}, \log\,\sigma^2_{x_t}] \tag{3.13}$$

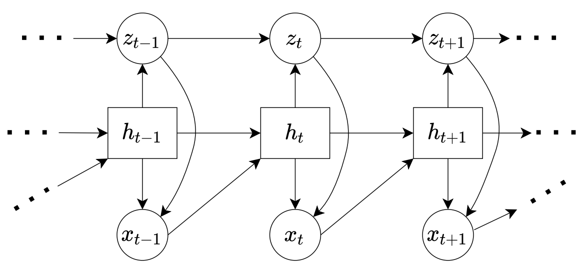
$$p_{\theta_z}(z_t|x_{1:t-1}, z_{t-1}) := f_{\theta_z}(h_t, z_{t-1}) = [\mu_{z_t}, \log\,\sigma^2_{z_t}] \tag{3.14}$$

$$q_\phi(z_t|x_{1:T}, z_{t-1}) := f_\phi(\overleftarrow{h}_t, z_{t-1}) = [\mu_{(z_t)_{inf}}, \log\,\sigma^2_{(z_t)_{inf}}] \tag{3.15}$$

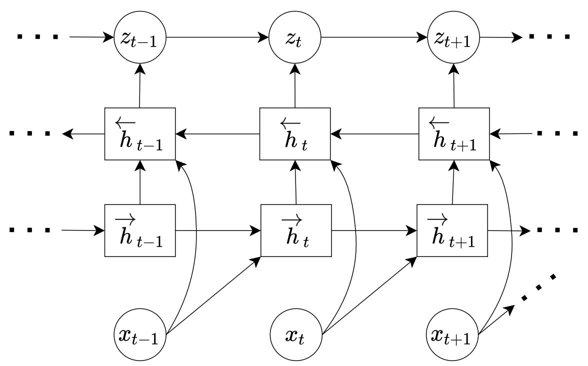
where, f_{θ_x}, f_{θ_z} and f_ϕ are all neural networks and the RNN hidden variables, $h_t := x_{1:t-1}$ and $\overleftarrow{h}_t := x_{1:T}$ as shown in Figure 3.3. The neural networks all output a mean and log-variance (μ and $\log \sigma^2$) which describe a Gaussian distribution. For example, $[\mu_{z_t}, \sigma^2_{z_t}]$ describes a Gaussian distribution that is associated with the latent variable z_t; hence, that variable could be sampled from its distribution, i.e. $z_t \sim \mathcal{N}(\mu_{z_t}, \sigma^2_{z_t})$. Note that log variance is used, as the variance or standard deviation would need to be a positive value; hence, the network output would need to be restricted to achieve this. By using log variance the network's output does not need to be restricted but the standard deviation and variance can easily be retrieved from the log-variance output if needed. Along with these networks describing the stochastic variables, a RNN is used to represent $x_{1:t-1}$ as previously discussed,

$$x_{1:t-1} := \text{RNNCell}(x_{t-1}, h_{t-1}) = h_t \tag{3.16}$$

where, RNNCell could be any kind of RNN cell, for example, a LSTM or Gated Recurrent Unit (GRU) cell. Note that unlike in Figure 3.2 where $x_{1:t} := h_t$ here the variables are shifted down a single time point as shown in Figure 3.3 so that $x_{1:t-1} := h_t$.



(a) Generative model



z_{t-1}
z_t
z_{t+1}
$\overleftarrow{h}_{t-1}$
$\overleftarrow{h}_t$
$\overleftarrow{h}_{t+1}$
$\overrightarrow{h}_{t-1}$
$\overrightarrow{h}_t$
$\overrightarrow{h}_{t+1}$
x_{t-1}
x_t
x_{t+1}

(b) Inference model

Figure 3.3. In the example DVAE implementation, 3.3a is the generative model described by Eqns. 3.10 & 3.11 while 3.3b is the inference model described by Eqn. 3.12. Note the variables inside circles are stochastic variables while the rectangles are deterministic

3.3 Bayesian Filtering

Bayesian filtering is a technique used to find the conditional distribution $p(x_{1:t}|y_{1:t})$, where x_t and y_t represent the states and measurements of a system modelled using a set of related Stochastic Differential Equations (SDEs). In practice, the dynamic process and measurement models are often described using discrete-time representations. Regardless of the representation, Bayesian filtering requires a model that describes the dynamics of the state (x_t) and a measurement model that relates the state to the observation (y_t). In the case of a continuous-time representation, an example of a system that could be filtered using filtering theory is described by the following SDEs,

$$dX_t = f(X_t)dt + g(X_t)dW_t \tag{3.17}$$
$$dY_t = h(X_t)dt + k(X_t)dB_t. \tag{3.18}$$

A filter in this context is an algorithm that outputs the conditional distribution $p(X_{1:t}|Y_{1:t})$ or $p(X_t|Y_{1:t})$ for any time point t in the sequence. For the discrete case which is more common in practice as measurements are sampled at discrete points, an example of a model would be,

$$x_t = f(x_{t-1}) + w_t \tag{3.19}$$
$$y_t = g(x_t) + v_t. \tag{3.20}$$

Where, $w_t \sim \mathcal{N}(0, Q_t)$ and $v_t \sim \mathcal{N}(0, R_t)$ are discrete noise samples. Here a filtering algorithm would find $p(x_{1:t}|y_{1:t})$ or $p(x_t|y_{1:t})$ for any given time point t in the sequence. An example of a Bayesian filter that works on a discrete system is the Kalman filter (if the dynamics and measurement model are linear and the states are described with Gaussian distributions) or the Particle filter (works in nonlinear, non-Gaussian cases).

More generally Bayesian filters apply Bayes' theorem to sequential random variables i.e.

$$p(x_{1:t}|y_{1:t}) = \frac{p(y_{1:t}, x_{1:t})}{p(x_{1:t})}. \tag{3.21}$$

Note that $p(y_{1:t}, x_{1:t})$ is a generative model and given enough training data, a DVAE can be trained to learn this generative model if it is unknown. Also, note the denominator $p(x_{1:t})$ is the marginal likelihood; the negative marginal log-likelihood is the variational objective when optimising DVAEs and is indirectly optimised using the ELBO (Eqn. 3.9). Hence, filters and DVAEs are related and are made up of the same components, however, in the DVAE case the generative model is unknown and is being learnt through the use of training data.

3.3.1 Kalman Filter

The Kalman Filter is a linear filtering technique that uses a dynamic and measurement model that can update the states of a system based on new measurements. The mathematical form of the state space model that a Kalman filter can be applied to generally takes the following form (Simon, 2006),

$$\mathbf{z}_k = F_k \mathbf{z}_{k-1} + B_k \mathbf{u}_k + \mathbf{w}_k \tag{3.22a}$$

$$\mathbf{x}_k = H_k \mathbf{z}_k + \mathbf{v}_k \tag{3.22b}$$

where $\mathbf{w}_k$ and $\mathbf{v}_k$ are noise processes that are modelled as white, zero mean Gaussian distributions with covariance matrices Q_k and R_k respectively. Hence, $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, Q_k)$ and $\mathbf{v}_k \sim \mathcal{N}(\mathbf{0}, R_k)$. The dynamic model described using matrix F_k can propagate the previous state $\mathbf{z}_{k-1}$ to $\mathbf{z}_k$, and if there are control inputs u_k, such as external forces acting on the system, these can be included using matrix B_k. However, propagating the state $\mathbf{z}_{k-1}$ to $\mathbf{z}_k$ is not perfect due to noise in the system, modelled using $\mathbf{w}_k$ in this case, which can add up as the state is propagated throughout time. Kalman filters reduce the uncertainty of the states $\mathbf{z}_k$ by using observations to update $\mathbf{z}_k$ based on the observation and its uncertainty. To do this effectively, a measurement model is used H_k which brings the state $\mathbf{z}_k$ to the observation domain (e.g. if the states are the temperature of a chemical process but the sensor uses volts, the matrix H_k would convert temperature to equivalent sensor voltage). There are some important points to be aware of when using a Kalman filter,

1. A linear system is assumed because if $\mathbf{z}_k$ and $\mathbf{x}_k$ are described by Gaussian distributions, then linear transformations would ensure they remain Gaussian distributions after the transformation.

2. Gaussian distributions are fully described by their mean and covariance (or variance in the 1D case) so the Kalman filter works by updating these statistics which fully describe an updated Gaussian distribution.

3. Markov assumption is used, meaning the current variable is dependent on the previous variable, e.g. $p(\mathbf{z}_t|\mathbf{z}_{1:t-1}) = p(\mathbf{z}_t|\mathbf{z}_{t-1})$

4. Measurements have conditional independence meaning current measurements do not depend on the previous ones, e.g. $p(\mathbf{x}_t|\mathbf{x}_{1:t-1}) = p(\mathbf{x}_t)$

The Kalman filter equations can be stated as,

- Update Mean and Covariance using Dynamics

$$\mathbf{z}_k^- = F_k \mathbf{z}_{k-1}^+ + B_k \mathbf{u}_k \tag{3.23a}$$
$$P_k^- = F_k P_{k-1}^+ F_k^T + Q_k \tag{3.23b}$$
$$S_k = H_k P_k^- H_k^T + R_k \tag{3.23c}$$
$$\boldsymbol{\mu}_k = H_k \mathbf{z}_k^- \tag{3.23d}$$

- Measurement Update

$$K_k = P_k^- H_k^T S_k^{-1} \tag{3.24a}$$
$$\mathbf{z}_k^+ = \mathbf{z}_k^- + K_k(\mathbf{x}_k - \boldsymbol{\mu}_k) \tag{3.24b}$$
$$P_k^+ = (I - K_k H_k) P_k^- \tag{3.24c}$$

where the $+$ exponent was used to define the posterior estimate (after measurement update) and $-$ denoted the prior variables (after dynamics) e.g. $\mathbf{z}_k^-$ was the mean of the prior after $\mathbf{z}_{k-1}^+$ was put through the dynamics and $\mathbf{z}_k^+$ was the mean of the posterior distribution after the measurement update. There are many ways to derive the Kalman filter; a useful derivation involves using Bayes rule, which shows the Kalman filter equations effectively apply Bayes rule to update the prior distribution $p(\mathbf{z}_{t-1}|\mathbf{x}_{1:t-1})$ to the posterior distribution $p(\mathbf{z}_{t-1}|\mathbf{x}_{1:t})$ given a new measurement $\mathbf{x}_t$. This Bayesian probabilistic framework will be useful in a deep learning context, where certain architectures, such as VAEs and DVAEs model probability distributions using neural networks. Writing the Kalman filter in probabilistic notation using Bayes' rule (Särkkä, 2013) would give,

$$p(\mathbf{z}_t|\mathbf{x}_{1:t}) = \frac{p(\mathbf{x}_{1:t}|\mathbf{z}_t)p(\mathbf{z}_t)}{p(\mathbf{x}_{1:t})} \tag{3.25}$$
$$p(\mathbf{z}_t|\mathbf{x}_{1:t}) = \frac{1}{\eta_t} p(\mathbf{x}_t|\mathbf{z}_t, \mathbf{x}_{1:t-1}) p(\mathbf{z}_t|\mathbf{x}_{1:t-1}) \tag{3.26}$$
$$= \frac{1}{\eta_t} p(\mathbf{x}_t|\mathbf{z}_t) p(\mathbf{z}_t|\mathbf{x}_{1:t-1}) \tag{3.27}$$

Where, $\eta_t = \int p(\mathbf{x}_t|\mathbf{z}_t)p(\mathbf{z}_t|\mathbf{x}_{1:t-1})d\mathbf{z}_t$ is a normalisation constant. Note that conditional independence of measurements was assumed in Eqn. 3.27 so that the current measurement $\mathbf{x}_t$ does not depend on the state and measurement histories, and hence, $p(\mathbf{x}_t|\mathbf{z}_t, \mathbf{x}_{1:t-1}) = p(\mathbf{x}_t|\mathbf{z}_t)$. Hence, using this Bayesian framework, it can be seen that two distributions are prominent in the formulation,

- $p(\mathbf{x}_t|\mathbf{z}_t)$: measurement model which takes the state $\mathbf{z}_t$ to the measurement domain.
- $p(\mathbf{z}_t|\mathbf{x}_{1:t-1})$: transition model which propagates the state $\mathbf{z}_{t-1}$ to the current time variable $\mathbf{z}_t$.

The transition model can be expanded further to show how $\mathbf{z}_{t-1}$ is transitioned to $\mathbf{z}_t$.

$$p(\mathbf{z}_t|\mathbf{x}_{1:t-1}) = \int p(\mathbf{z}_t|\mathbf{z}_{t-1}\mathbf{x}_{1:t-1})p(\mathbf{z}_{t-1}|\mathbf{x}_{1:t-1})d\mathbf{z}_{t-1} \tag{3.28}$$

$$p(\mathbf{z}_t|\mathbf{x}_{1:t-1}) = \int p(\mathbf{z}_t|\mathbf{z}_{t-1})p(\mathbf{z}_{t-1}|\mathbf{x}_{1:t-1})d\mathbf{z}_{t-1} \tag{3.29}$$

Where the Markov assumption is used in Eq. 3.29 so that $p(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_{1:t-1}) = p(\mathbf{z}_t|\mathbf{z}_{t-1})$. Again two distributions are prominent in the expanded transition model,

- $p(\mathbf{z}_t|\mathbf{z}_{t-1})$: the dynamics that takes the previous state $\mathbf{z}_{t-1}$ to the current state $\mathbf{z}_t$.
- $p(\mathbf{z}_{t-1}|\mathbf{x}_{1:t-1})$: the previous filtered distribution for state $\mathbf{z}_{t-1}$.

If the probability distributions described in this Bayesian formulation are all Gaussian distributions, then the Kalman filtering equations perform the Bayesian update described in Eq. 3.27.

A note on Kalman Filters and DVAEs

One way of deriving the ELBO for the DVAE is by minimising the negative log-likelihood of the marginalised distribution $p(\mathbf{x}_{1:T})$. This is the equivalent of maximising the likelihood that any sampled input data $\mathbf{x}_{1:t}$ comes from the original data distribution $p(\mathbf{x}_{1:T})$. With the VAE and DVAE, equations 3.3 and 3.9 were used respectively as an indirect way of achieving this objective. In those cases, the data distribution was not known and could not be directly solved due to an intractable integral. Hence, $p(Z|X)$ was not known, and $q_\phi(Z|X)$ was used to approximate it. The state space model can also act as a generative model by using the transition and measurement model (prior and decoder), resulting in the generative model,

$$p(\mathbf{x}_t, \mathbf{z}_t) = p(\mathbf{x}_t|\mathbf{z}_t)p(\mathbf{z}_t|\mathbf{x}_{1:t-1}) \tag{3.30}$$

However, to train this generative model, the negative marginal log-likelihood, $-\text{log}p(\mathbf{x}_{1:T})$, can directly be minimised as the Kalman filter has an analytical solution for it.

$$p(\mathbf{x}_t) = \mathcal{N}(H_t\mathbf{z}_t^-, H_tP_t^-H_t^T + R_t) \tag{3.31}$$

So the negative log-likelihood becomes,

$$\begin{aligned} -\text{log}\ p(\mathbf{x}_{1:T}) &= -\text{log}\prod_{t=1}^{T} p(\mathbf{x}_t|\mathbf{x}_{t-1}) \\ L &= -\sum_{t=1}^{T} \text{log}\ p(\mathbf{x}_t) \end{aligned} \tag{3.32}$$

Recall that the Kalman filter assumes independent measurements; hence, $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ became $p(\mathbf{x}_t)$. Hence, optimising any parameters by minimising Eqn. 3.32 would result in a generative model which could be used as a filter, given new measurements.

This fact is used in some works, such as creating an Extended Kalman Filter using neural networks to model the transfer and measurement functions (A. H. Li et al., 2021). Another example is Kalman filters which work on pseudo-observations in a latent space where the linear Kalman filter can be used but then applying Normalizing Flows to the marginal likelihood to allow more complex distributions to be modelled (de Bézenac et al., 2020). These examples could use the Kalman filter to analytically solve for $p(\mathbf{x}_t)$ and directly optimise the neural networks using Eqn. 3.32. This can lead to more accurate estimates, and tighter uncertainty bounds as the encoder is no longer being approximated like it is by $q(Z|X)$ in the VAE. Since the Kalman filter deals with linear Gaussian models, some works use the Particle filter algorithm to estimate the marginal likelihood $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ and use this in a negative log-likelihood objective function (Maddison et al., 2017; Naesseth et al., 2018). This way nonlinear models, such as neural networks, can be used to define the state dynamics and measurement functions and the output is not restricted to a Gaussian distribution.

3.3.2 Particle Filters

This background section mainly shows the similarity between Sequential Monte Carlo techniques (otherwise known as particle filters) and DVAEs. Note, in section 5.1.1, a derivation of the ELBO loss function for a conditional DVAE uses importance sampling techniques[1], and while filtering itself is not used in the thesis, the generative models used for particle filtering are. Particle Filters (PF) use Monte Carlo techniques to estimate $p(z_{1:t}|x_{1:t})$ as often in the general nonlinear, non-Gaussian case since this posterior is difficult or impossible to find analytically. Monte Carlo methods are numerical methods that allow one to calculate integrals such as,

$$\mathbb{E}_{p(z|x_{1:T})}[f(z)|x_{1:T}] = \int f(z)p(z|x_{1:T})dz. \tag{3.33}$$

This is done by sampling N samples from $p(z|x_{1:T})$ and estimating the integral using a sum,

$$\mathbb{E}_{p(z|x_{1:T})}[f(z)|x_{1:T}] \approx \frac{1}{N}\sum_{i=1}^{N} f(z^{(i)}). \tag{3.34}$$

However, it is often difficult to sample from the posterior distribution or even the marginal distribution, $p(x_{1:T})$ if training the generative model is the ultimate goal. To solve this problem, importance sampling is often used.

Importance Sampling

Importance sampling (IS) uses a distribution that is easy to sample from to generate samples from a more complex distribution. This distribution is usually of the same form as the complex

[1]The inference model q_ϕ in the DVAE can be thought of as an importance distribution and this theory is used in research to improve VAE training (Maddison et al., 2017; Naesseth et al., 2018)

distribution, i.e. has the same variables and conditional variables, e.g. $q(z|x_{1:T})$ which is defined as a Gaussian distribution to sample from $p(z|x_{1:T})$, which may not be Gaussian. This can be implemented using,

$$\begin{aligned}
\mathbb{E}_{p(z|x_{1:T})}[f(z)p(z|x_{1:T})] &= \int f(z)p(z|x_{1:T})\frac{q(z|x_{1:T})}{q(z|x_{1:T})}dz && (3.35a)\\
&= \int f(z)\frac{p(z|x_{1:T})}{q(z|x_{1:T})}q(z|x_{1:T})dz && (3.35b)\\
&= \mathbb{E}_{q(z|x_{1:T})}\left[f(z)\frac{p(z|x_{1:T})}{q(z|x_{1:T})}\right] && (3.35c)\\
&\approx \frac{1}{N}\sum_{i=1}^{N} f(z^{(i)})\frac{p(z^{(i)}|x_{1:T})}{q(z^{(i)}|x_{1:T})} && (3.35d)\\
&\approx \sum_{i=1}^{N} w^{(i)} f(z^{(i)}) && (3.35e)
\end{aligned}$$

where now $z^{(i)} \sim q(z|x_{1:T})$, instead of sampling from $p(z|x_{1:T})$ and $w^{(i)} := \frac{1}{N}\frac{p(z^{(i)}|x_{1:T})}{q(z^{(i)}|x_{1:T})}$. By using Bayes' Theorem, we can also get rid of the dependence on knowing $p(z|x_{1:T})$ and instead let the weights be equal to (Särkkä, 2013),

$$\hat{w}^{(i)} = \frac{p(x_{1:T}|z^{(i)})p(z^{(i)})}{q(z^{(i)}|x_{1:T})} \tag{3.36}$$

$$w^{(i)} = \frac{\hat{w}^{(i)}}{\sum_{j=1}^{N}\hat{w}^{(j)}}. \tag{3.37}$$

Where $\hat{w}^{(i)}$ is the unnormalised weight and $w^{(i)}$ is the normalised weights. Hence, the marginal likelihood can be estimated using the normalisation constant $p(x_{1:T}) \approx \sum_{j=1}^{N}\hat{w}^{(j)}$.

Particle Filtering Algorithm

To find the filtering distribution for sequential variables, i.e. $p(z_{1:T}|x_{1:t})$, Sequential Importance Sampling (SIS) can be used. Hence, in the sequential case, the variable z can be replaced with $z_{1:T}$ and use a similar argument as with Eqns. 3.35, the weights in the sequential case can be stated as,

$$\hat{w}_T^{(i)} = \frac{p(x_{1:T}|z_{1:T}^{(i)})p(z_{1:T}^{(i)})}{q(z_{1:T}^{(i)}|x_{1:T})} = \frac{p(x_{1:T}, z_{1:T}^{(i)})}{q(z_{1:T}^{(i)}|x_{1:T})} \tag{3.38}$$

$$\hat{w}_T^{(i)} = \frac{p(x_T|x_{1:T-1}, z_{1:T}^{(i)})p(z_T^{(i)}|z_{1:T-1}^{(i)}, x_{1:T-1})}{q(z_T^{(i)}|z_{1:T-1}^{(i)}, x_{1:T})}\frac{p(x_{1:T-1}, z_{1:T-1}^{(i)})}{q(z_{1:T-1}^{(i)}|x_{1:T})} \tag{3.39}$$

$$\hat{w}_T^{(i)} = \frac{p(x_T|x_{1:T-1}, z_{1:T}^{(i)})p(z_T^{(i)}|z_{1:T-1}^{(i)}, x_{1:T-1})}{q(z_T^{(i)}|z_{1:T-1}^{(i)}, x_{1:T})}\hat{w}_{T-1}^{(i)} \tag{3.40}$$

Using this result, the following recursion can be applied,

$$\hat{w}_t^{(i)} = \frac{p(x_t|x_{1:t-1}, z_{1:t}^{(i)})p(z_t^{(i)}|z_{1:t-1}^{(i)}, x_{1:t-1})}{q(z_t^{(i)}|z_{1:t-1}^{(i)}, x_{1:T})}\hat{w}_{t-1}^{(i)} \tag{3.41}$$

The normalised weights can then be found,

$$w_t^{(i)} = \frac{\hat{w}_t^{(i)}}{\sum_{i=1}^{N} \hat{w}_t^{(i)}} \tag{3.42}$$

Hence, the marginal log-likelihood can be found through the normalised weights,

$$\begin{align} p(x_{1:T}) &= \prod_{t=1}^{T} p(x_t|x_{1:t-1}) \tag{3.43} \\ &= \prod_{t=1}^{T} \sum_{j=1}^{N} \hat{w}_t^{(j)} \tag{3.44} \\ \log p(x_{1:T}) &= \sum_{t=1}^{T} \log p(x_t|x_{1:t-1}) \tag{3.45} \\ &= \sum_{t=1}^{T} \log \sum_{j=1}^{N} \hat{w}_t^{(j)} \tag{3.46} \end{align}$$

The particle filter algorithm is, therefore,

1. Initialise the prior $p(z_0)$ and set the initial weights (usually chosen to all be uniform, i.e. $w_0^{(i)} = \frac{1}{N}, \ \forall i \in [1, N]$
2. Draw samples $z_t^{(i)}$ from $q(z_t^{(i)}|z_{1:t-1}^{(i)}, x_{1:T})$ i.e. $z_t^{(i)} \sim q(z_t^{(i)}|z_{1:t-1}^{(i)}, x_{1:T}), \ \forall i \in [1, N]$
3. Calculate new weights according to Eqn. 3.41 and normalise using Eqn. 3.42

To apply these equations in practice, notice that the following distributions/models are necessary.

- Measurement model (decoder): $p(x_t|x_{1:t-1}, z_{1:t}^{(i)})$
- Transition/Dynamic model (prior): $p(z_t^{(i)}|z_{1:t-1}^{(i)}, x_{1:t-1})$
- Proposal/Inference model (encoder): $q(z_t^{(i)}|z_{1:t-1}^{(i)}, x_{1:T})$

Notice, in brackets, the terminology for the equivalent network in a VAE/DVAE context is stated. IS, and in the sequential case SIS, has been applied as an alternate way of estimating the marginal log-likelihood directly, instead of indirectly through the ELBO (Maddison et al., 2017; Burda et al., 2016; Ishizone et al., 2020). These applications show how Bayesian filters and DVAEs are related; they use the same components but have different end goals. Filters attempt to find $p(z_{1:T}|x_{1:T})$ to estimate $z_{1:T}$ in light of new data, while DVAEs want to sample from $p(x_{1:T})$ to generate novel samples and do this by learning the generative model $p(z_{1:T}, x_{1:T})$ and require the importance distribution (inference model) q_ϕ to indirectly sample from $p(x_{1:T})$.

3.4 Neural Differential Equations

3.4.1 Neural Ordinary Differential Equations

Neural Ordinary Differential Equations (NODEs) (R. T. Q. Chen et al., 2018) essentially use a neural network to describe a first order differential equation (or a series of first order differential equations). The equation describing NODEs is given by equation 3.47,

$$\frac{d\mathbf{x}(t)}{dt} = f_\theta(\mathbf{x}(t), t). \tag{3.47}$$

where $\mathbf{x}_t$ is the input data at index t and θ are the parameters of the neural network f_θ. The following integral would need to be solved for the trajectory of states over the index variable t,

$$\mathbf{x}(T) = \mathbf{x}(0) + \int_0^T f_\theta(\mathbf{x}(t), t)dt \tag{3.48}$$

This can be solved numerically with a number of ODE solver algorithms, for example Euler's method,

$$\mathbf{x}_T = \mathbf{x}_0 + \Delta t \cdot f_\theta(\mathbf{x}_t, t) \tag{3.49}$$

The inputs are transformed to the output variables using a numerical differential equation solver and the network f_θ acts as the function describing the first order ODE $\frac{d\mathbf{x}}{dt}$. An application of this is that the network f_θ could describe another larger network and t would be the index of the layers of the network. In this case the variables are propagating through the network "layers" using the smaller network f_θ that describes an ODE. By choosing the number of steps in the ODE solver or the tolerance for an adaptive ODE solver the amount of "layers" or functional evaluations are controlled. t could also be time itself and the NODE could try model a time-series signal and output trajectories much like a regular ODE might be used.

Figure 3.4 shows a simple illustrated example of how NODEs take input variables and transform them to an output variable to describe a larger neural network.

3.4.2 Neural Stochastic Differential Equations

The latent dynamics used in the DVAE method highlighted in this work will utilise Neural Stochastic Differential Equations (NSDEs); hence, this section will give a brief background on them. A general stochastic differential equation takes the following form,

$$dX_t = f(X_t, t)dt + g(X_t, t)dW_t \tag{3.50}$$

where f is the drift function describing the transition dynamics, g is the diffusion governing the amount of noise in the process, and dW_t is the Brownian motion term commonly used in SDEs. NSDEs (X. Li et al., 2020) essentially use neural networks to describe the drift and diffusion

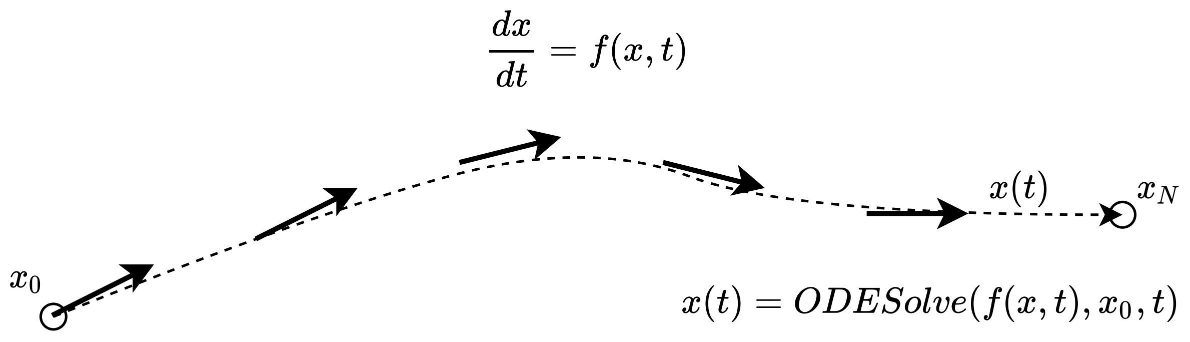


Figure 3.4. A visual example of how NODE propagates the input variable x_0 through the network, which is described by an ODE. Each solid arrow is an evaluation of the network $\frac{dx}{dt} = f(x,t)$ which is used to find the next step or next network hidden layer variable, where t is the depth of the network. Since, there are 5 arrows shown this is equivalent to a network with 5 hidden layers. The dotted path labelled $x(t)$ is the trajectory the variables take through the NODE and x_N is the final output of the NODE.

functions, which can then be used in a numerical SDE solver to propagate the desired variable (X_t) through time. Hence, the neural networks describe a vector field in which trajectories pass through instead of directly outputting the variables X_t at each time point like an RNN would.

A common way of implementing a NSDE for time series analysis is using an RNN encoder to find the initial latent condition (Z_0) needed for the SDE solver, then propagating through time to find the sequence $Z_{0:T}$. Figure 3.5 shows an example of a NSDE being used to forecast an input time series sequence $x_{t_0:T}$ to the time $T+s$. By using the RNN to "scan" the input data, an initial condition Z_0 can be found, and the NSDE can extrapolate by simply using the time period $[t_0, T+s]$ in the numerical SDE solver. The networks $f(Z_t,t)$ and $g(Z_t,t)$ are trained to give good reconstructions when Z_t is put through a decoder to give the reconstruction $\hat{x_t}$, while the RNN is trained to give a different initial condition Z_0 based on different sequences of $x_{t_0:T}$ given.

A problem with both the NSDE and NODE is that the implementation for time series data involves using this RNN encoder to output some unique latent initial condition z_0 given the input data sequence $x_{0:T}$. Hence, if we get x_{T+1} then we have to scan the entire sequence again and include this new value ($x_{0:T+1}$) and then use the neural differential equation and a differential equation solver to find the new sequence of latent variables. This can make an online implementation difficult if we require a quick way to update our latent states based on an incoming stream of observations. In the next section, we will look at the Kalman filter which is a state estimation or Bayesian filtering technique that can update states based on new observations or data given.

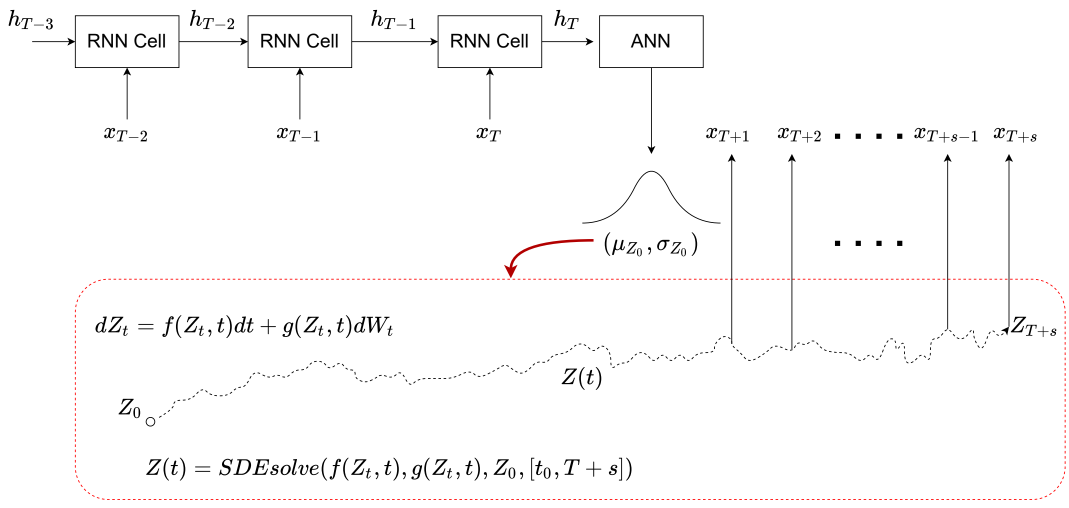


Figure 3.5. NSDE trained to reconstruct input data x_t by learning latent dynamics which represent the dynamics of x_t

Chapter 4

Neural Ordinary Differential Equations for RUL Estimation

This chapter focuses on showing how neural networks can be represented as dynamic systems to try to reduce some of the mystery behind their inner workings. The majority of the content in this chapter is a slightly modified version of “Remaining Useful Life Estimation Using Neural Ordinary Differential Equations” published in the International Journal of Prognostics and Health Management and has been used here with the permission of the copyright holder (Star & McKee, 2021). As such, the model was compared to state-of-the-art models at the time of writing that paper, and future chapters offer more current state-of-the-art results. However, this chapter’s focus is on the validity of using NODEs in machinery prognostics and hence, helps build the foundation for future work within the thesis. Hence, it focuses less on state-of-the-art performance and is more of an exploration of how NODEs can be utilised in machinery prognostics and compares them to similar networks that do not use the differential equation framework.

The NODE and ResNet are compared by constructing both neural networks and assessing their performance for RUL estimation on a turbofan engine dataset. In the original paper (R. T. Q. Chen et al., 2018), NODE was introduced by first considering the Residual Network (ResNet) (He et al., 2016) as an Euler discretization of an ODE. NODE can be considered a continuous and generalised version of ResNet. ResNet was first introduced as an architecture that improved image recognition results. Hence, for the first application of NODE to machinery prognostics, it is fitting that the prognostics method chosen focuses on using CNNs for time-frequency spectral images. This will allow for the comparison between the ResNet CNN architecture and a NODE version of the CNN, which will be referred to as NODE-CNN. Note that for this method, the NODE does not model the dynamics of degradation but instead describes how the input data propagates through the neural network to produce an output (the RUL estimate in this case).

4.1 Data

To compare the ResNet and NODEs, they will both be trained on a prognostics dataset to estimate the RUL of machinery. This section will explain the contents of the dataset. Additionally, it covers how the data was prepared to become the inputs of the neural networks and how the RUL targets were prepared for the supervised learning task.

4.1.1 CMAPSS dataset

The data used here is NASA's Commercial Modular Aero-Propulsion System Simulation (CMAPSS) dataset (Saxena et al., 2008). This dataset was produced through simulations of sensor readings for aircraft gas turbine engines. The RUL values and, therefore, the time of failure is known for each simulated engine and was found when the simulated variables reached a threshold determined by a failure criteria. There are four separate datasets that vary in difficulty concerning RUL estimation; a summary of each dataset is given in Table 4.1. The possible failure modes that could occur in this dataset set are either the High-Pressure Compressor (HPC) or fan degrading below a certain threshold. Note that the HI and threshold are unknown and must be inferred through the sensor data; this is the challenge presented by the CMAPSS dataset. Each dataset includes training and testing trajectories (time-series data) which contain the sensor values and operating conditions of different units/engines. The training set is simulated until failure; hence, the final time point is the time of failure (i.e. RUL = 0 at the final time). The testing dataset contains trajectories until a random time point, and the RUL at that time point is supplied for each trajectory. The trained network can be evaluated on this testing set, and the estimated RUL can be compared with the true RUL value given in the testing dataset.

Table 4.1: Different datasets found in CMAPSS: states how many different engines are simulated, their operating conditions and what types of failures they can experience (Fault Modes)

Dataset	Train Trajectories	Test Trajectories	Conditions	Fault Modes
FD001	100	100	1 (sea level)	1 (HPC Degradation)
FD002	260	259	6	1 (HPC Degradation)
FD003	100	100	1 (sea level)	2 (HPC Degradation, Fan Degradation)
FD004	248	249	6	2 (HPC Degradation, Fan Degradation)

Each dataset consists of multiple time series for both the training and testing datasets. There are 26 columns of data for each unit. The contents of each column are specified in Table 4.2.

Table 4.2: Information contained in each column for the training and testing datasets

columns	
1	unit number
2	time (cycles)
3	operational setting 1
4	operational setting 2
5	operational setting 3
6-26	sensor measurements (1-21)

Using this dataset, the two networks specified in this section will be trained on this data and compared based on how they perform on the test data. Specifics on evaluating the performance of the network on the test data will be mentioned in the Results section.

4.1.2 Preparing the Data

The first step in many prognostics methods is normalising the time series data. This is often done by taking the means and standard deviations of the sensor signals for each sensor in the training dataset and applying equation 4.1,

$$\hat{\mathbf{x}}_d^{(n)} = \frac{\mathbf{x}_d^{(n)} - \mu_d}{\sigma_d}. \tag{4.1}$$

Where $\hat{\mathbf{x}}_d^{(n)}$ are the normalised sensor values for unit number n and sensor d over the lifetime of the unit. Similarly, μ_d denotes the means of each sensor, σ_d denotes the standard deviations and $\mathbf{x}_d^{(n)}$ are the raw sensor values.

Normalisation is often carried out this way, where the mean and standard deviation are calculated based on a particular sensor signal. However, with multiple operating conditions, such as in the datasets FD002 and FD004 (see Table 4.1), it is often more useful to normalise the sensor data based on a particular operating condition (Pasa et al., 2019). Hence, for these datasets, the normalisation of the data can be done by applying equation 4.2,

$$\hat{\mathbf{x}}_d^{(n)} = \sum_{c=0}^{N_c} \delta_c \odot \frac{\mathbf{x}_d^{(n)} - \mu_d^{(c)}}{\sigma_d^{(c)}}. \tag{4.2}$$

In this case, $\mu_d^{(c)}$ and $\sigma_d^{(c)}$ are the mean and standard deviation of sensor d data found in operating condition c, N_c denotes the total number of operating conditions (six for FD002 and FD004) and $\delta_c = 1$ when the data at that time point is in the operating condition c otherwise $\delta_c = 0$. A clustering algorithm can be used to find which operating condition the data belongs to, and this will allow for the calculation of the means and standard deviation of each sensor for each operating condition. In this case, the K-means clustering algorithm was used for simplicity.

Some studies take time-series signals and use time-frequency methods such as wavelet transforms or short-time Fourier transforms to convert to a 2D time-frequency spectral plot (Zhu et al., 2019; X. Li et al., 2018). For this experiment, a similar process to (Zhu et al., 2019) was used, which involved the following general steps,

1. Relevant sensor signals are taken from the machinery sensor data.
2. A short-time Fourier transform is applied to each time-series sensor signal (i.e. output tensor size = [amount of sensors, time, frequency]).
3. Bilinear interpolation is used to change the size of each time-frequency signal to a 30×30 format (i.e. output tensor size = [amount of sensors, 30, 30]).

For this specific experiment, the details of these steps are given are described in the following paragraphs and a general diagram of the data are shown in Figure 4.1. Firstly the relevant sensor signals refer to the fact that not all the sensor values change over time. Some of the data given are not sensor values but only numerical representations of the machine's condition, such as operating mode. Therefore, some of the data stays constant throughout time and can cause problems for any time-series analysis. To avoid these types of problems, any sensor that remained constant throughout the lifetime of the machine was removed from the analysis. Secondly, once these signals are chosen, a short-time Fourier transform is performed on them to produce time-frequency representations of the signals. Short-time Fourier transforms use a time window to split the data into different sections. The Fourier transform is applied to each section to find the frequency information for each time section, giving the signal a time-frequency representation. Lastly, these time-frequency representations have a size based on the time window size chosen. To batch the input data and have consistent output sizes for the CNN, the inputs should be the same size; however, each sensor signal has a varying length. To achieve same-size inputs, bilinear interpolation will be used to convert the time-frequency signals to a 30 × 30 size "image-like" format.

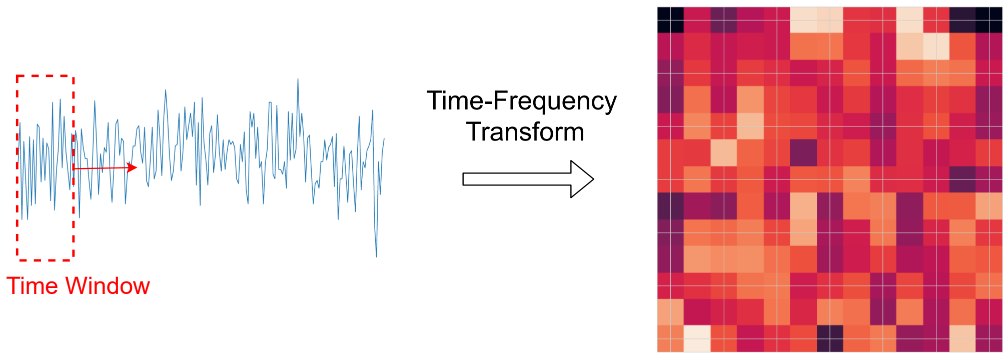


Figure 4.1. A single time-series signal which is transformed to the time-frequency domain using the STFT. The frequency intensities from the Fourier coefficients is shown in the image that results from the time-frequency transform

The testing and training datasets were prepared similarly but had one notable difference. For the training data preparation, the sensor signals were broken up into a certain amount of smaller signals (the specific amount depended on a chosen hyperparameter that will be called *split* here). The percentage of the signal used is calculated using equation 4.3,

$$\%split_i = \frac{i}{split + 1}, \text{for } i = 1, ..., split \tag{4.3}$$

Hence, if $split = a$ and let L be the total length of the signal and $L_i = L \times \%split_i$, then the signals of a certain unit used for training the network would be $\mathbf{x}_{0:L_i}$ where $\mathbf{x}_{0:L_i} = [x_{0:L_1}, ..., x_{0:L_a}]$. If, like with the test dataset, the entire training signals were used, then the RUL would always equal zero as the entire training signal represents the entire life of the engine. This would not robustly train the network, and hence the signals were split to give the network different cases exposing it to multiple RUL values. Also note the denominator of equation 4.3 is $split + 1$, hence, 100% of the signal is not used when training meaning the network does not train on RUL = 0 cases, as this is not useful. The testing dataset has multiple signal lengths and RUL values based on those signals meaning this process was not necessary.

4.2 Methodology

This section states the details of the experiments performed on the data using the different network architectures. The general process used to estimate the RUL will be stated as well as the specifics of what hyperparameters were used to construct the network. Both the ResNet and NODE-CNNs have similar processes for estimating the RUL; therefore, a general overview is given below, which is relevant to both architectures. Note that this is expected considering ResNets are utilised here as a discrete version of the NODE.

1. A separate CNN is used to downsample (reduce the dimensions of) the prepared input data described in the previous section.
2. The ResNet or NODE is applied to extract features from the downsampled input data.
3. The final output from the ResNet/NODE is flattened to a vector and the input to a standard feedforward MLP network. The output of this MLP is the RUL estimate of the machinery.

Bayesian optimisation tools were used to optimise the hyperparameters. The software package Ax which uses the package BoTorch as a backend (Balandat et al., 2019) was used to tune the hyperparameters using Bayesian optimisation. Bayesian optimisation uses a curve-fitting method known as Gaussian process regression to optimise the hyperparameters based on some criterion (Frazier, 2018). It takes a vector of hyperparameters as inputs and outputs some criterion or loss. In the deep learning case, a set of hyperparameters is used to train the neural network; after training, the criterion can be calculated on a validation dataset. Hence, the input hyperparameters and output criterion represent a point and Gaussian Process Regression can be used to fit a curve given these points. The goal of Bayesian optimisation is to find the vector of input hyperparameters that minimises the criterion which is the equivalent of finding the minimum of the curve. Each new criterion calculated for a set of input hyperparameters is treated as a data point describing the curve and Bayesian methods are used to update the curve based on this new data.

The criterion used here for hyperparameter optimisation is the normalised score function (Eq. 4.4). The score function was designed to result in higher loss values for RUL estimates that are larger than the actual RUL (Saxena et al., 2008). This is done to encourage more conservative estimates so that failures do not occur before the prediction from the estimated RUL.

$$s = \begin{cases} \frac{1}{N}\sum\limits_{i=1}^{N} \exp\left[\frac{-(\hat{y}_i - y_i)}{13}\right] - 1, & \text{if } \hat{y}_i \leq y_i \\ \frac{1}{N}\sum\limits_{i=1}^{N} \exp\left[\frac{(\hat{y}_i - y_i)}{10}\right] - 1, & \text{if } \hat{y}_i > y_i \end{cases} \tag{4.4}$$

Where $\hat{y}_i$ is the estimated RUL of unit i from the network, y_i is the actual RUL of unit i and N is the total amount of units in the dataset. Notice here an average score is used (referred to as normalised score); hence this involves calculating the score found in (Saxena et al., 2008) and then dividing by the total amount of units N.

The criterion is evaluated on a validation set, separate from the dataset the network is trained on when tuning the hyperparameters. Here the validation set was acquired from the original training data, which is further split into a training dataset and a validation dataset. An 80%/20% (training/validation) split was used on the original training data in this particular case. The new training set is used to optimise the network during the hyperparameter optimisation. In contrast, the validation set acts as a testing dataset that the network did not see during training. Therefore, the validation loss represents a testing loss that can be used to optimise the hyperparameters. Note that the testing dataset is left alone and untouched by any optimisation process to avoid overfitting.

This hyperparameter optimisation method was chosen as it is more computationally efficient than other optimisation methods such as random search, which randomly tests different hyperparameter combinations. Bayesian optimisation reduces the amount of time required to find well-performing hyperparameters compared to other methods, which may find more optimal hyperparameters but require a longer time to search for them. The hyperparameters which are being tuned are,

- Learning rate
- Batch size
- Time window for STFT
- split (see Eq. (4.3))
- Weight decay (L2 regularisation on the network parameters)
- Training epochs

Once the hyperparameters are chosen, the networks could be trained on the full training dataset. The chosen hyperparameters for the networks in each dataset are stated in Appendix A. The hyperparameters table in the Appendix also states other hyperparameters such as convolution kernel size and the number of channels the CNN has. Both NODE and ResNet have a convolution kernel size of 3 and a channel size of 64. Further research could fine-tune the kernel and channel size for this application; as it is, the values here are chosen as they are common values for these variables to take in other image-based tasks.

The network itself was trained using the Mean Squared Error (MSE) loss function (Eq. (4.5)).

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y_i} - y_i)^2. \tag{4.5}$$

4.2.1 ResNet

For the ResNet experiment, the time-frequency data was the input to an initial set of downsampling layers to give the data a certain amount of channels that will be used in the ResNet. The output of the downsampling layers became the inputs to the ResNet blocks. Several ResNet blocks were stacked, so the outputs of one block were the inputs of the other. Finally, the output of the final ResNet block was flattened and passed through a feedforward network which would output the RUL estimate. Table 4.3 shows the basic layout of the ResNet architecture used as well as the size of the output tensors of each layer.

Table 4.3: Architecture of the ResNet

Layers	**Layer Output Size**
Downsampling Layers	
Conv2D	(bs,channels,28,28)
ResNet block	(bs,channels,14,14)
ResNet block	(bs,channels,7,7)
ResNet Layers	
ResNet block $\times n$	(bs,channels,7,7)
Feedforward Layers	
BatchNorm	(bs, channels, 7, 7)
ReLU	(bs, channels, 7, 7)
AvgPooling	(bs, channels, 1, 1)
Flatten	(bs, channels)
Linear	(bs, 816)
ReLU	(bs, 816)
Linear	(bs, 200)
ReLU	(bs, 200)
Linear	(bs, 1)

Figure 4.2 is a diagram showing the layout of the ResNet used to estimate the RUL using the turbofan engine data.

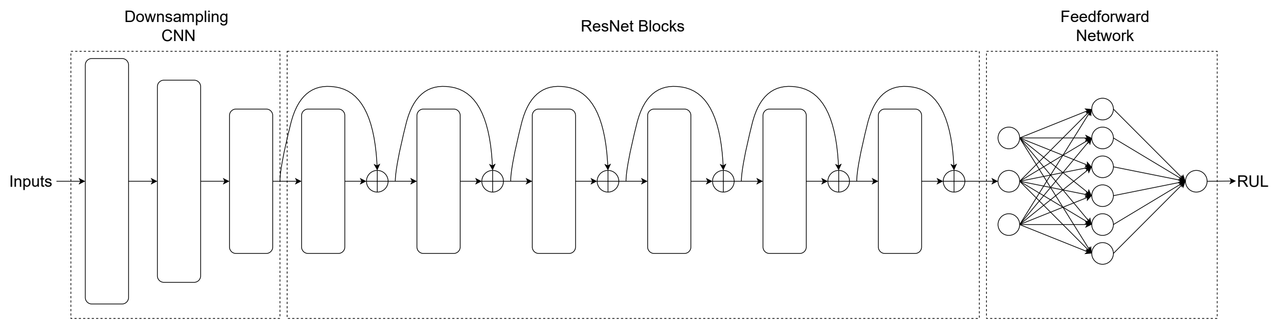


Figure 4.2. An illustration of the ResNet architecture layout used in this experiment. Note each rectangular block is a CNN, and the $\bigoplus$ is an element-wise addition operator

Note the formula describing the operation of each ResNet block can be stated as,

$$\mathbf{x}_{k+1} = \mathbf{x}_k + F_k(\mathbf{x}_k). \tag{4.6}$$

Where $\mathbf{x}_k$ is the input to the k^{th} network or the output of the $(k-1)^{th}$ ResNet block and $F_k(\mathbf{x}_k)$ is the output of the k^{th} network and is added to x_k to become the output of the ResNet block. The addition of the network output and input shown in Equation 4.6 is what the $\bigoplus$ symbolises in Figure 4.2.

4.2.2 NODE-CNN

Much like the ResNet described previously, the NODE-CNN layout involves a downsampling layer that reduces the dimensions of the input data and extract features. Figure 4.3, illustrates the NODE-CNN layout. The main difference between the ResNet and the NODE-CNN network layout is that the ResNet Blocks described in the previous section are replaced with NODE-CNN blocks. These blocks are effectively CNN networks whose output is used in a numerical ODE solver and which propagates the hidden state forward. Finally, like the ResNet, a feedforward network is used to transform the output of the previous CNN network into a single value which will be the RUL estimate. The NODE-CNN layout is shown in Table 4.4.

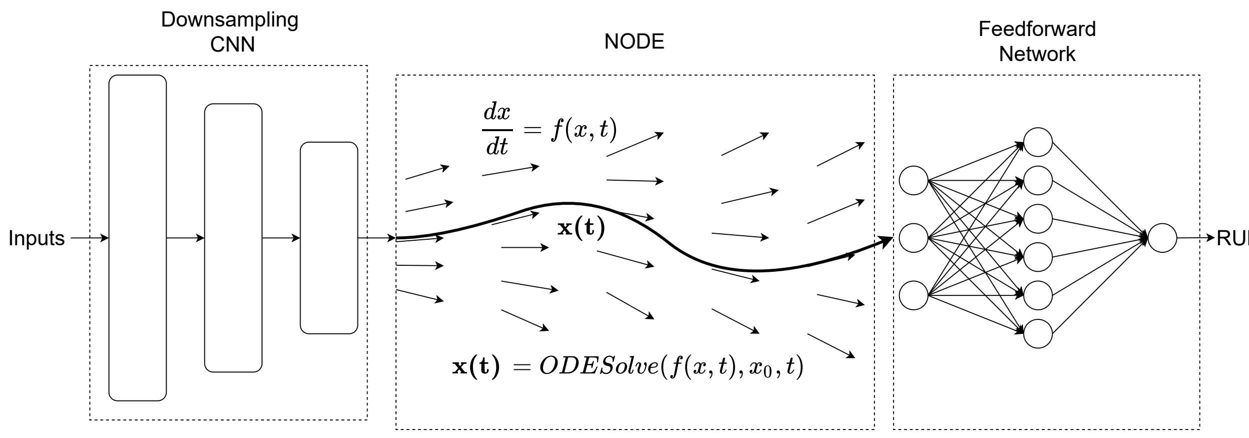


Figure 4.3. An illustration of the NODE-CNN architecture used for this experiment. Note that $f(x,t)$ is the CNN that describes the gradient at any point (x,t) and is used in the numerical ODE solver to propagate the trajectory forward. In this case t is not time but the depth of the NODE network.

The NODE-CNN is similar to the ResNet, however, the ResNet blocks can be seen as using Euler's method to solve for the layer output. For this NODE-CNN the Runge-Kutta method is used instead which will improve how the trajectory of the differential equation is propagated through the network layer space. The reason for using Runge-Kutta instead of an adaptive solver is due to the more reliable performance of using a fixed amount of steps (the solver will not stop halfway through due to the stiffness of the ODE). It is also used because of the size of the problem and the networks involved which caused large computation times when using adaptive solvers. Hence, to train the network reliably and in a reasonable amount of time the Runge-Kutta (RK) solver was used.

4.3 Results

To compare the performance of the models the Root Mean Squared Error (RMSE) between the RUL estimates for the testing dataset and the actual RUL values are used as a performance indicator. The RMSE can be calculated as shown in Eqn. 4.7,

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(R\hat{U}L_i - RUL_i)^2}. \tag{4.7}$$

Table 4.4: Architecture of the NODE-CNN

Layers	**Layer Output Size**
Downsampling Layers	
Conv2D	(bs, channels, 28, 28)
BatchNorm	(bs, channels, 28, 28)
ReLU	(bs, channels, 28, 28)
Conv2D	(bs, channels, 14, 14)
BatchNorm	(bs, channels, 14, 14)
ReLU	(bs, channels, 14, 14)
Conv2D	(bs, channels, 7, 7)
CNN-NODE	
BatchNorm	(bs, channels, 7, 7)
ReLU	(bs, channels, 7, 7)
Conv2D	(bs, channels, 7, 7)
BatchNorm	(bs, channels, 7, 7)
ReLU	(bs, channels, 7, 7)
Conv2D	(bs, channels, 7, 7)
BatchNorm	(bs, channels, 7, 7)
Feedforward Layers	
BatchNorm	(bs, channels, 7, 7)
ReLU	(bs, channels, 7, 7)
AvgPooling	(bs, channels, 1, 1)
Flatten	(bs, channels)
Linear	(bs, 816)
ReLU	(bs, 816)
Linear	(bs, 200)
ReLU	(bs, 200)
Linear	(bs, 1)

Where $R\hat{U}L_i$ is the RUL estimate for unit i, RUL_i is the actual RUL for unit i and N is the total number of units. The RMSE is commonly used in many similar works when dealing with the CMAPSS dataset allowing for direct comparison between different methods. RMSE values acquired for each of the CMAPSS testing datasets were calculated using Eq. 4.7 (Table 4.5). Table 4.5 shows the RMSE scores, which are calculated using equation 4.7 acquired for each of the CMAPSS datasets. Each unit in the testing dataset underwent the same data-cleaning process as the training set. Note that the means and standard deviations of the training set were used for normalising the testing dataset. As the full sensor trajectories are not available for the testing dataset, the mean and standard deviation of an entire trajectory wouldn't normalise the data the same way it was for the training sensor signals. The signals resulting from that process were inputs to the neural network, and the estimated RUL output was compared with the actual RUL that was given as a part of the dataset.

The other testing performance criterion used was the score originally introduced in (Saxena et al., 2008). A normalised version of this score is shown in Eqn. 4.4 which was used for hyperparameter tuning, however, other prognostics methods use the non-normalised version

Table 4.5: *RMSE performance on each of the testing CMAPSS datasets with maximum RUL value of 130*

Method	FD001	FD002	FD003	FD004
ResNet (This work)	12.81	16.92	14.68	18.25
NODE-CNN (This work)	13.65	**14.30**	12.65	**15.06**
CNN (Pasa et al., 2019)	15.00	17.50	14.80	17.40
LSTM (Pasa et al., 2019)	16.50	18.10	15.90	17.20
MLP (Pasa et al., 2019)	15.10	18.00	14.30	16.60
CNN (X. Li et al., 2018)	12.61	22.36	12.64	23.31
LSTM (Listou Ellefsen et al., 2019)	**12.56**	22.73	**12.10**	22.66

(Eqn. 4.8) and it will therefore be used for comparison here.

$$s = \begin{cases} \sum_{i=1}^{N} \exp\left[\frac{-(\hat{y}_i - y_i)}{13}\right] - 1, & \text{if } \hat{y}_i \leq y_i \\ \sum_{i=1}^{N} \exp\left[\frac{(\hat{y}_i - y_i)}{10}\right] - 1, & \text{if } \hat{y}_i > y_i \end{cases} \tag{4.8}$$

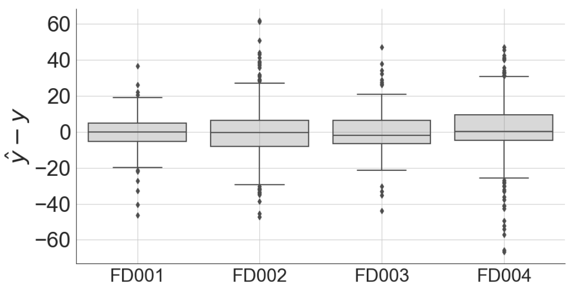
Table 4.6 shows the (non-normalised) scores compared with the other prognostic methods.

Table 4.6: *comparison of the scores (Eq. (4.8)) on each of the testing CMAPSS datasets with a maximum RUL value of 130*

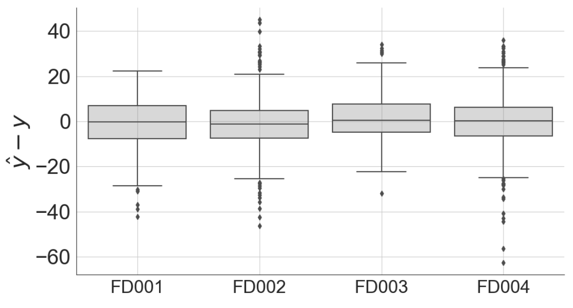
Method	FD001	FD002	FD003	FD004
ResNet (This work)	250	2250	428	1842
NODE-CNN (This work)	235	**886**	270	**947**
CNN (Pasa et al., 2019)	369	1757	332	1678
LSTM (Pasa et al., 2019)	444	942	718	1487
MLP (Pasa et al., 2019)	411	1113	1091	2755
CNN (X. Li et al., 2018)	273	10412	284	12466
LSTM (Listou Ellefsen et al., 2019)	**231**	3366	**251**	2840

Figure 4.4 shows the box and whisker plots that were generated using the difference between estimated and target RUL values for each unit in the testing datasets. The figure shows the errors spread between the estimated RUL and the true RUL values over all the units in each testing dataset. Note that the negative values correspond to an estimate less than the true RUL, while positive values indicate a larger RUL estimate.

Finally, the effect of the ODE solver step size was tested for the NODE-CNN. Figure 4.5 shows how the RMSE on each testing dataset alters by adjusting the number of steps in the numerical ODE solver. In this case, the variable associated with "time", describes the depth of the NODE network. The network depth is controlled by choosing an interval ($t \in [0, 1]$) and splitting the interval into an even number of steps. Hence, increasing the number of steps increases the resolution, which in standard ODEs would increase the accuracy at the expense of computational time. For this test, the NODE-CNN was retrained with 10 steps using 50 epochs, so there were more step options to test before reaching zero steps when reducing the number of steps. Note the newly trained networks do not have optimal RMSE values, but this experiment is only concerned with the relative differences in RMSE as the number of steps changes.

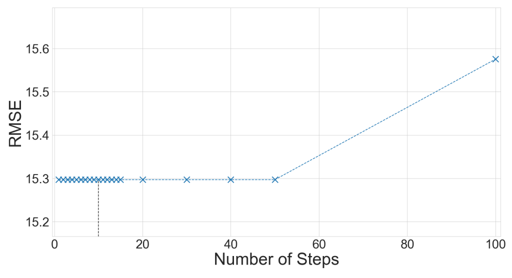


(a) ResNet

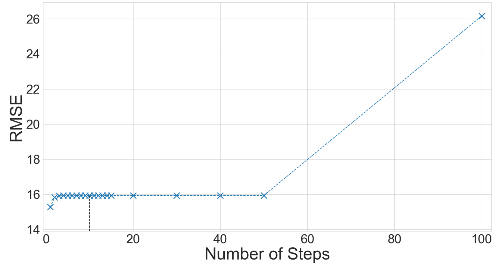


(b) NODE-CNN

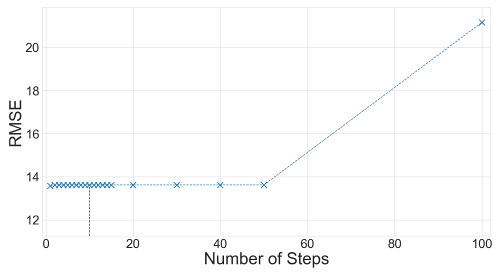
Figure 4.4. Box and whisker plot for the ResNet (4.4a) and NODE-CNN (4.4b) RUL differences ($\hat{y}$ = estimated RUL and y = true RUL) over all the units for each testing dataset



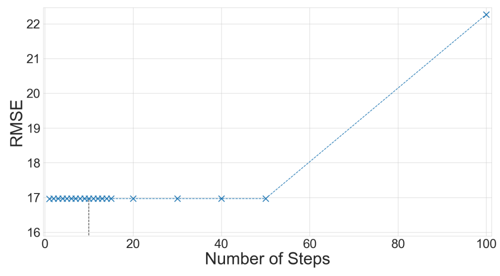
(a) FD001



(b) FD002



(c) FD003



(d) FD004

Figure 4.5. Graphs showing testing performance as measured by the RMSE metric versus the number of steps used for the numerical integration method (4th order Runge-Kutta). Here the network was retrained using 10 steps in the ODE solver otherwise the same hyperparameters were used, but 50 epochs are used for faster training as the relative performance is of interest here. During testing, the number of steps was altered and the RMSE was recalculated for the testing dataset as shown on the plot. The vertical dotted line indicates the number of steps the network was trained on.

4.4 Discussion

The ResNet and NODE-CNN showed good performance when compared to the other methods in Tables 4.5 and 4.6. In particular, ResNet and NODE-CNN performed especially well on the FD002 and FD004 datasets. This is most likely because these datasets have multiple operating conditions, and it was shown in (Pasa et al., 2019) that performing operating condition-based normalisation (Eq. (4.2)) improves performance in this case. Hence, ResNet and NODE-CNN outperformed the other methods on the FD002 and FD004 datasets which did not use operating condition-based normalisation. (only the methods from (Pasa et al., 2019) used this normalisation technique). When comparing the performance of NODE-CNN on datasets FD002 and FD004 to (Pasa et al., 2019), it can be seen that the NODE-CNN outperformed their regular (non-ResNet) CNN. The improved performance could be because ResNet and NODE outperform regular CNNs in image recognition tasks (R. T. Q. Chen et al., 2018; He et al., 2016), hence, the architecture of NODE-CNN may be due to an improvement to the simpler CNN architecture used in (Pasa et al., 2019). The datasets with only one operating condition (FD001 and FD003) also performed relatively well compared to state-of-the-art techniques. However, the best results for these datasets were achieved by (Listou Ellefsen et al., 2019), who used a Restricted Boltzmann Machine to extract degradation features and input them into an LSTM, which estimated the RUL. It should also be noted that (Listou Ellefsen et al., 2019) did not employ the same operating condition-based normalisation as was done in this study. Hence, it may retain state-of-the-art performance for the FD002 and FD004 datasets if the sensor signals are normalised this way. The multi-scale CNN (X. Li et al., 2018) also outperformed the NODE-CNN and ResNet in terms of the RMSE measure for FD001 and FD003, but NODE-CNN and ResNet had better scores. This could be because both ResNet and NODE-CNN had their hyperparameters tuned to minimise the normalised score criterion and hence would produce more conservative RUL estimates but not necessarily better RMSEs. However, a more detailed analysis would be required to state this as fact. While not completely outperforming all these techniques, the aim of showing that NODEs and ResNets could compete with state-of-the-art techniques has been achieved.

Using the intuition of an ODE modelling a dynamic system, it may be expected that changing the time step size used in the numerical ODE solver would affect the accuracy of the NODE-CNN. Altering this step size would normally provide a trade-off between accuracy and computation time. (Queiruga et al., 2020) mentioned that the solution accuracy of NODEs does not change with step size as a standard ODE would. For their tests, they used NODEs to model the dynamics of a mechanical system; while here, NODE-CNN was used to model some unknown and more abstract dynamics of the hidden variables in a neural network. Hence, their accuracy refers to the error between the "true" trajectory and the NODEs trajectory at each time-point. In contrast, the error discussed here is between the network's final estimates and some known RUL values. Their results showed that NODEs using 4th order Runge-Kutta did achieve some increased accuracy with increased resolution (decreasing time step) but not significant compared to what is expected by a numerical integrator ODE solver. Figure 4.5 shows the number of steps did not affect the testing loss on the final outputs except in extreme cases (number of steps altered by a factor of 10 from the original training value) before the error increased. This could be because the number of steps corresponds to the depth of the NODE-CNN and not necessarily an increase in resolution (and therefore accuracy). This suggests one must be careful when designing the network and specifying the problem if a relationship between step size and solution accuracy is to be achieved. Here the network was used as a standard black-box, feedforward network, so it did not benefit from the accuracy-time trade-off. (Queiruga et al., 2020) also mention how NODEs can be altered to achieve this property by describing multiple networks for portions of the trajectory. Hence, suppose one wants to reduce the number of steps in the ODE solver. In that case, a certain number of networks are removed, and the leftover networks are used to describe the dynamics during those periods in the trajectories propagation. This reduces the accuracy of the results but also reduces the time required to compute those results.

4.5 Conclusions

In this chapter, we have shown that NODEs can be used to construct neural networks and therefore describe the inner workings of the network using ODEs. It can be seen through the results that the NODE-CNN had comparable performance to the other RUL estimation models shown in the results. The score values for the NODE beat the ResNet performance and had better scores than the other models for FD002 and FD004, while it was relatively close to the best result for FD001 and FD003. As mentioned in the discussion, its good performance on the FD002 and FD004 datasets could be due to the lack of operating condition normalisation for some of the other models. The chapter aimed to show that NODEs could be a potential avenue to explore in the field of machinery prognostics. This could potentially be used to leverage the knowledge of differential equations to alter these inner workings. For example, instead of describing the network using a NODE, a neural stochastic differential equation (SDE) can be used instead, thereby accounting for uncertainty in the network (X. Li et al., 2020). These types of extensions mean knowledge of differential equations becomes a key factor in developing these types of networks. Knowledge in these areas already exists within the machinery prognostics community, as statistical methods are a popular methodology in data-driven machinery prognostics (Si et al., 2011). SDEs themselves can be found throughout these statistical methods. Hence, showing that NODE can perform just as well as other network types could explore ways to use or augment differential equations to apply deep learning for machinery prognostics.

Chapter 5

Dynamical Variational Autoencoders for Remaining Useful Life Estimation

Much of the background section focused on generative models and how they can be extended to deal with sequential data. This chapter aims to show how this theory can be used to construct a Dynamical Variational Autoencoder (DVAE) for RUL estimation. In later chapters, DVAE will be further explored using different architectures and training methods to alter the generative models and how they are applied. This chapter walks through the construction of what will be referred to as a "standard-DVAE" for RUL estimation. It is called a standard-DVAE as the neural network architectures do not differ much from what is seen in other literature for applications other than machinery prognostics (Hafner et al., 2019; Fraccaro et al., 2016; Girin et al., 2021). Many of those models use RNNs to encode sequential information and Multilayered Perceptron (MLP) networks as encoders, decoders and prior networks. The main difference in this work is that a conditional DVAE is necessary when applying DVAEs for RUL estimation. This is because we train the network in a supervised manner, unlike many generative modelling applications that use unsupervised learning. For example, in (Hafner et al., 2019), the DVAE is trained to create a latent representation of the pixel dynamics of videos. Hence, the input pixels of the videos are also the target outputs, and during training, one wants to reconstruct them. During testing, the network might want to extrapolate how the video may play out; this is done through the learnt latent dynamics that can be used to extrapolate the latent variables and then decode to find the pixels of the possible future video. Here, the DVAE for RUL estimation is not being applied in this unsupervised manner. Instead, the DVAE will attempt to directly estimate the RUL using the sensor data and latent variables the model learns to construct during training. Hence, the latent variables here are more like auxiliary variables whose dynamics the model learns during training, and these variables aid the model in its RUL estimation. Hence, unlike the other DVAE models, the input training data is not also the target data.

5.1 Methodology

Ultimately the goal of a data-driven probabilistic RUL estimation model is to find $p(r_{1:T}|x_{1:T})$, i.e. the probability of the RUL sequence $r_{1:T}$ given the sensor data $x_{1:T}$. Taking the deep generative model approach, we want to maximise the probability that the DVAE generates any sample, $r_{1:T}$, that comes from the distribution, $p(r_{1:T}|x_{1:T})$. This is similar to how in the background section, the goal of a VAE was to sample X that is from the unknown distribution $p(X)$ and for the DVAE, the goal was adjusted for sequential variables, where the sample $x_{1:T}$ should come from the unknown distribution $p(x_{1:T})$. Minimising the ELBO loss function (Eqn. 3.9 in the DVAE case) was the criteria used to train three separate models, the inference, prior and decoder, which ultimately resulted in a generative model that outputs samples $x_{1:T}$ that are very likely to belong to $p(x_{1:T})$. In this chapter, instead of deriving the ELBO to maximise the likelihood that the generative model output belongs to the distribution $p(x_{1:T})$ instead we want to increase the likelihood a sample $r_{1:T}$ comes from the conditional distribution $p(r_{1:T}|x_{1:T})$. This will create a conditional DVAE that can be used for RUL estimation much like other neural network models such as in Chapter 4, however, unlike those models, this is a probabilistic model, which includes latent variables that aid the model in RUL estimation.

5.1.1 Deriving the ELBO Loss Function

To derive an ELBO loss for the conditional DVAE, we can start with the negative log-likelihood of the distribution $p(r_{1:T}|x_{1:T})$. If this criterion is minimised, then it would be the same as maximising the likelihood samples $r_{1:T}$ belong to $p(r_{1:T}|x_{1:T})$.

$$\begin{aligned} L &= -\log p(r_{1:T}|x_{1:T}) && (5.1) \\ &= -\log \int p(r_{1:T}, z_{1:T}|x_{1:T}) dz_{1:T} && (5.2) \end{aligned}$$

Note, $z_{1:T}$ is an arbitrary latent sequence we introduced to construct the generative model. One way to derive the ELBO is to use an importance distribution $q_\phi(z_{1:T}|r_{1:T}, x_{1:T}) = \prod_{t=1}^{T} q_\phi(z_t|z_{1:t-1}, r_{1:T}, x_{1:T})$ and use the same techniques found in importance sampling to approximate the more complicated distribution $p(r_{1:T}|x_{1:T})$ (see section 3.3.2 in the Background chapter). Normally $q_\phi(z_t|z_{1:t-1}, r_{1:T}, x_{1:T})$ would be a simple distribution such as a Gaussian distribution. Note any subscripts, such as ϕ used here, represent the parameters of a neural network; hence, q_ϕ is a neural network.

$$\begin{aligned} L &= -\log \int p(r_{1:T}, z_{1:T}|x_{1:T}) \frac{q_\phi(z_{1:T}|r_{1:T}, x_{1:T})}{q_\phi(z_{1:T}|r_{1:T}, x_{1:T})} dz_{1:T} && (5.3) \\ &= -\log \int \frac{p(r_{1:T}, z_{1:T}|x_{1:T})}{q_\phi(z_{1:T}|r_{1:T}, x_{1:T})} q_\phi(z_{1:T}|r_{1:T}, x_{1:T}) dz_{1:T} && (5.4) \\ &= -\log \mathbb{E}_{q_\phi(z_{1:T}|r_{1:T}, x_{1:T})} \left[\frac{p(r_{1:T}, z_{1:T}|x_{1:T})}{q_\phi(z_{1:T}|r_{1:T}, x_{1:T})} \right]. && (5.5) \end{aligned}$$

At this point, the expression is still too difficult to evaluate; normally, when deriving the ELBO for a VAE at this step, Jensen's inequality is used, $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$, where f is a convex function. Hence, this results in,

$$L \leq -\mathbb{E}_{q_\phi(z_{1:T}|r_{1:T},x_{1:T})}\left[\log \frac{p(r_{1:T}, z_{1:T}|x_{1:T})}{q_\phi(z_{1:T}|r_{1:T}, x_{1:T})}\right] \tag{5.6}$$

$$\mathcal{L} = -\mathbb{E}_{q_\phi(z_{1:T}|r_{1:T},x_{1:T})}\left[\log \prod_{t=1}^{T} \frac{p_{\theta_r}(r_t|r_{1:t-1}, z_{1:t}, x_{1:T})p_{\theta_z}(z_t|z_{1:t-1}, r_{1:t-1}, x_{1:T})}{q_\phi(z_t|z_{1:t-1}, r_{1:T}, x_{1:T})}\right] \tag{5.7}$$

$$= -\mathbb{E}_{q_\phi(z_{1:T}|r_{1:T},x_{1:T})}\left[\sum_{t=1}^{T} \log \frac{p_{\theta_r}(r_t|r_{1:t-1}, z_{1:t}, x_{1:T})p_{\theta_z}(z_t|z_{1:t-1}, r_{1:t-1}, x_{1:T})}{q_\phi(z_t|z_{1:t-1}, r_{1:T}, x_{1:T})}\right] \tag{5.8}$$

$$= -\sum_{t=1}^{T} \mathbb{E}_{q_\phi(z_{1:t}|r_{1:T},x_{1:T})}\left[\log \frac{p_{\theta_r}(r_t|r_{1:t-1}, z_{1:t}, x_{1:T})p_{\theta_z}(z_t|z_{1:t-1}, r_{1:t-1}, x_{1:T})}{q_\phi(z_t|z_{1:t-1}, r_{1:T}, x_{1:T})}\right]. \tag{5.9}$$

Note, the last line uses the fact that expectations are a linear operator, i.e. $\mathbb{E}[f(x) + g(x)] = \mathbb{E}[f(x)] + \mathbb{E}[g(x)]$, and that expectations can cascade (Girin et al., 2021) i.e.,

$$\mathbb{E}_{q_\phi(z_{1:T}|r_{1:T},x_{1:T})}[f(z)] = \mathbb{E}_{q_\phi(z_1|r_{1:T},x_{1:T})}\left[\mathbb{E}_{q_\phi(z_2|z_1,r_{1:T},x_{1:T})}\left[\dots \mathbb{E}_{q_\phi(z_T|z_{T-1},r_{1:T},x_{1:T})}[f(z)]\dots\right]\right].$$

Expanding the expression further yields,

$$\mathcal{L} = -\sum_{t=1}^{T} \mathbb{E}_{q_\phi(z_{1:t}|r_{1:T},x_{1:T})}\left[\log p_{\theta_r}(r_t|r_{1:t-1}, z_{1:t}, x_{1:T})\right] - \sum_{t=1}^{T} \mathbb{E}_{q_\phi(z_{1:t}|r_{1:T},x_{1:T})}\left[\log \frac{p_{\theta_z}(z_t|z_{1:t-1}, r_{1:t-1}, x_{1:T})}{q_\phi(z_t|z_{1:t-1}, r_{1:T}, x_{1:T})}\right] \tag{5.10}$$

$$= -\sum_{t=1}^{T} \mathbb{E}_{q_\phi(z_{1:t}|r_{1:T},x_{1:T})}\left[\log p_{\theta_r}(r_t|r_{1:t-1}, z_{1:t}, x_{1:T})\right] + \sum_{t=1}^{T} \mathbb{E}_{q_\phi(z_{1:t-1}|r_{1:T},x_{1:T})}\left[\mathbb{E}_{q_\phi(z_t|z_{1:t-1},r_{1:T},x_{1:T})}\left[\log \frac{q_\phi(z_t|z_{1:t-1}, r_{1:T}, x_{1:T})}{p_{\theta_z}(z_t|z_{1:t-1}, r_{1:t-1}, x_{1:T})}\right]\right]. \tag{5.11}$$

Note,

$$\mathbb{E}_{q_\phi(z_t|z_{1:t-1},r_{1:T},x_{1:T})}\left[\log \frac{q_\phi(z_t|z_{1:t-1}, r_{1:T}, x_{1:T})}{p_{\theta_z}(z_t|z_{1:t-1}, r_{1:t-1}, x_{1:T})}\right] = D_{KL}\left(q_\phi(z_t|z_{1:t-1}, r_{1:T}, x_{1:T})||p_{\theta_z}(z_t|z_{1:t-1}, r_{1:t-1}, x_{1:T})\right),$$

which therefore gives the final expression for the DVAE ELBO loss function,

$$\mathcal{L} = -\sum_{t=1}^{T} \mathbb{E}_{q_\phi(z_{1:t}|r_{1:T},x_{1:T})}\left[\log p_{\theta_r}(r_t|r_{1:t-1}, z_{1:t}, x_{1:T})\right] + \sum_{t=1}^{T} \mathbb{E}_{q_\phi(z_{1:t-1}|r_{1:T},x_{1:T})}\left[D_{KL}\left(q_\phi(z_t|z_{1:t-1}, r_{1:T}, x_{1:T})||p_{\theta_z}(z_t|z_{1:t-1}, r_{1:t-1}, x_{1:T})\right)\right]. \tag{5.12}$$

Intuitively this is the same as the ELBO for the non-conditional DVAE (Eqn. 3.9); however, here, it is conditioned on the conditional variable $x_{1:T}$. Note, $x_{1:T}$ could not be simplified any further (unless causality is assumed); this conditional DVAE for RUL estimation should therefore be a noncausal model if no further assumptions are made. This noncausal model is implemented in practice through sequence-to-sequence models, e.g. inputting a time-windowed sequence into a bi-directional RNN which outputs another sequence of the same size. Hence, the entire available sequence isn't processed in one go; instead, the model uses all the variables within a certain time window of size T, this way the model is not trained to require an entire run-to-failure sequence to estimate the RUL (as that would be pointless). For reference later in the text, note that the ELBO can be broken down into two terms, the negative log-likelihood term (Eqn. 5.13) and the KL-divergence term (Eqn. 5.14),

$$NLL = -\sum_{t=1}^{T} \mathbb{E}_{q_\phi(z_{1:t}|r_{1:T},x_{1:T})} \left[\log p_{\theta_r}(r_t|r_{1:t-1}, z_{1:t}, x_{1:T})\right] \tag{5.13}$$

$$KLD = \sum_{t=1}^{T} \mathbb{E}_{q_\phi(z_{1:t-1}|r_{1:T},x_{1:T})} \left[D_{KL}\left(q_\phi(z_t|z_{1:t-1}, r_{1:T}, x_{1:T})||p_{\theta_z}(z_t|z_{1:t-1}, r_{1:t-1}, x_{1:T})\right)\right]. \tag{5.14}$$

It may not be clear from these terms how one should train and implement a DVAE. Hence, the next sections will go over the practical steps to implement the DVAEs for RUL estimation. However, it is important to realise that a benefit of deriving this loss is that we started with the basic goal of a machinery prognostics model, i.e. minimise $-\log p(r_{1:T}|x_{1:T})$, and ended up knowing what form the different models must take to achieve that goal (q_ϕ, p_{θ_r} and p_{θ_z}). This derivation process has been vital in identifying how to construct the neural network models for estimating the RUL. For example, the noncausal nature of the models was made clear from the derivation of the ELBO. Any assumptions made that simplify the model can be explicitly made and compared to the models presented in this derivation. The rest of this thesis continues presenting different realisations of DVAE models; keep in mind this derivation has presented a general framework that is used for constructing each of them.

5.1.2 General Overview of Model Training and Testing

From the ELBO criterion derived and shown in Eqn. 5.12, it can be seen the three networks needed to construct the generative model are,

- Inference model: $q_\phi(z_{t-1}|z_{1:t-2}, r_{1:T}, x_{1:T})$
- Prior/Transition model: $p_{\theta_z}(z_t|z_{1:t-1}, r_{1:t-1}, x_{1:T})$
- Decoder/Measurement model: $p_{\theta_r}(r_t|r_{1:t-1}, z_{1:t}, x_{1:T})$

Note, from now on, the Prior and Decoder models will be known as the Transition and Measurement models respectively. This change is partly because of their relation to state space models and Bayesian filtering, as stated in the Background section 3.3.1 and also to be more consistent with the terminology used in later chapters, such as chapter 7, where this relation is explored in more depth. For the work in this chapter, it was found empirically that the best performance came from assuming the previous RUL does not impact the transition or measurement model outputs. Each of the model's outputs describes a Gaussian distribution as is common in deep generative models. Note that generative models use sampling to represent the distribution they are describing, $p(r_{1:T}|x_{1:T})$ in this case; hence, the generative model can represent non-Gaussian distributions through repeated sampling even if the transition and measurement models output a local Gaussian distribution (Figure 1.1 in the Introduction showed an illustration of this concept). To make the models represent Gaussian distributions the neural networks output a mean and log-variance (that was converted to a standard deviation); these statistics fully describe a Gaussian distribution. Another assumption used was the Markov assumption for the prior model latent variables, so instead of being dependent on $z_{1:t-1}$, it was only dependent on the previous latent variable z_{t-1}. This also means the decoder was dependent on z_t instead of $z_{1:t}$. The Markov assumption was mainly used for simplicity when implementing the model in code. Hence, the models for the DVAE used in this chapter can be described using,

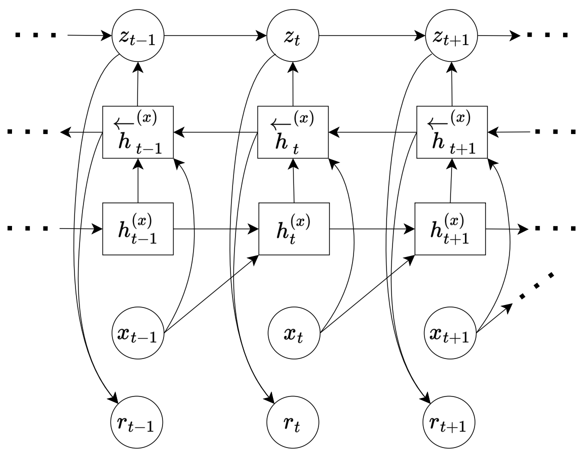
- Inference model: $q_\phi(z_t|z_{1:t-1}, r_{1:T}, x_{1:T}) = \mathcal{N}(\mu_t^{(\hat{z})}, \sigma_t^{(\hat{z})})$
- Transition model: $p_{\theta_z}(z_t|z_{t-1}, x_{1:T}) = \mathcal{N}(\mu_t^{(z)}, \sigma_t^{(z)})$
- Measurement model: $p_{\theta_r}(r_t|z_t, x_{1:T}) = \mathcal{N}(\mu_t^{(r)}, \sigma_t^{(r)})$

where $(\mu_{\hat{z}}, \log\sigma_{\hat{z}}^2)$, $(\mu_z, \log\sigma_z^2)$ and $(\mu_r, \log\sigma_r^2)$ were the outputs of the inference, transition and measurement networks respectively. Note the other models not explicitly stated here are the encoder models, which encode and represent $x_{1:T}$ and $r_{1:T}$. Here bi-directional RNNs Gated Recurrent Unit (GRU) RNNs are used; the forward pass captures $h_t := x_{1:t-1}$ and the backwards pass captures $\overleftarrow{h}_t := x_{1:T}$. Using these models and the ELBO loss in Eqn. 5.12, the conditional DVAE can be trained for RUL estimation. To apply the model and calculate the ELBO during training, the following steps are applied,

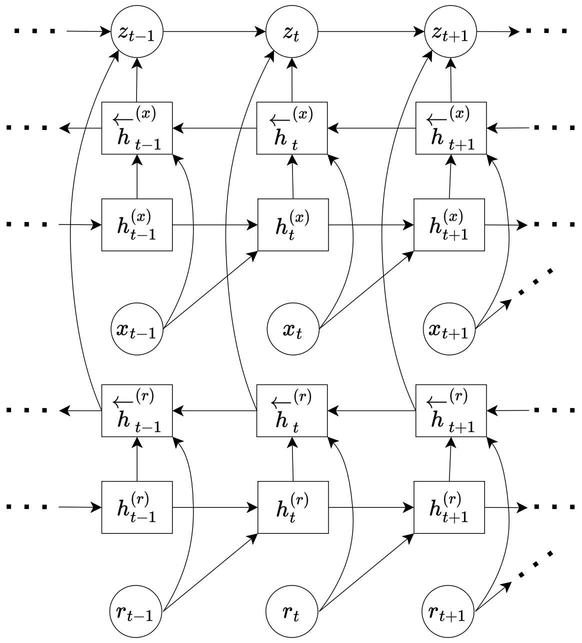
1. Encode the data, $r_{1:T}$ and $x_{1:T}$, with two separate bidirectional RNNs.
2. Initialise a latent variable (z_0) and use the encoded variables $\overleftarrow{h}_t^{(x)} := x_{1:T}$ and $\overleftarrow{h}_t^{(r)} := r_{1:T}$ as inputs in the inference model (q_ϕ) to recursively generate latent variable distributions which ultimately gives $\mathcal{N}(\mu_{1:T}^{(\hat{z})}, \sigma_{1:T}^{(\hat{z})})$.
3. Sample $z_{1:T}$ from this distribution and using the previous initial latent variable construct a sequence $z_{0:T-1}$.
4. Use the transition model on each of these to predict the next time point, i.e. $z_{t-1} \rightarrow p_{\theta_z}(z_t|z_{t-1}, x_{1:T}) = \mathcal{N}(\mu_t^{(z)}, \sigma_t^{(z)})$. Doing this for each time point results in the transition latent variable distribution $\mathcal{N}(\mu_{1:T}^{(z)}, \sigma_{1:T}^{(z)})$
5. Using the transition and inference distributions, $p_{\theta_z} = \mathcal{N}(\mu_{1:T}^{(z)}, \sigma_{1:T}^{(z)})$ and $q_\phi = \mathcal{N}(\mu_{1:T}^{(\hat{z})}, \sigma_{1:T}^{(\hat{z})})$, the KL-divergence term in the ELBO can be calculated, Eqn. 5.14
6. The sampled latent sequence from the inference distribution, $z_{1:T}$, and the encoded sequence $\overleftarrow{h}_t^{(x)} := x_{1:T}$, $\forall\ t$, can then be decoded using $p_{\theta_r}(r_t|z_t, x_{1:T}) = \mathcal{N}(\mu_t^{(r)}, \sigma_t^{(r)})$ at each time point to find $\mathcal{N}(\mu_{1:T}^{(r)}, \sigma_{1:T}^{(r)})$.
7. The negative log-likelihood term from the ELBO, Eqn. 5.13, can be calculated using the measurement model outputs.

8. With both the negative log-likelihood and KL-divergence terms calculated, the ELBO loss can be found and used to optimise the neural network during training.

Note, since z_0 is a dummy variable (only time points 1 to T are relevant), it can be initialised as zero, or a simple MLP network could be used to output an initial variable z_0. Here an MLP was used to initialise z_0 using the sensor values at the first time-point, i.e. $f_{\theta_{z_0}}(x_1) = z_0$, where θ_{z_0} are the parameters of the neural network that can be optimised during training. A graphical representation of the inference and generate models is shown in Figure 5.1 as an aid.



(a) Generative model


z_{t-1}
z_t
z_{t+1}
$\overleftarrow{h}^{(x)}_{t-1}$
$\overleftarrow{h}^{(x)}_{t}$
$\overleftarrow{h}^{(x)}_{t+1}$
$h^{(x)}_{t-1}$
$h^{(x)}_{t}$
$h^{(x)}_{t+1}$
x_{t-1}
x_t
x_{t+1}
$\overleftarrow{h}^{(r)}_{t-1}$
$\overleftarrow{h}^{(r)}_{t}$
$\overleftarrow{h}^{(r)}_{t+1}$
$h^{(r)}_{t-1}$
$h^{(r)}_{t}$
$h^{(r)}_{t+1}$
r_{t-1}
r_t
r_{t+1}

(b) Inference model

Figure 5.1. A graphical representation of the conditional DVAE for RUL estimation, 3.3a is the generative model described by Eqns. 3.10 & 3.11 while 3.3b is the inference model described by Eqn. 3.12.

The steps for applying the model after training are more simple as it essentially involves applying the generative model,

1. Encode the sensors, $x_{1:T}$, with the bidirectional RNN.

2. Initialise a latent variable z_0

3. Use the transition model to generate the next latent variable, $p_{\theta_z}(z_t|z_{t-1}, x_{1:T}) = \mathcal{N}(\mu_t^{(z)}, \sigma_t^{(z)})$. i.e. starting from z_0 generate the distribution for z_1, sample z_1 from that distribution and use that to generate z_2 and so on.

4. From the last step, we end up with a sequence of sampled latent variables $z_{1:T}$; use the samples at each time point in the measurement model, $p_{\theta_r}(r_t|z_t, x_{1:T}) = \mathcal{N}(\mu_t^{(r)}, \sigma_t^{(r)})$ to generate the distributions of the RUL estimates.

5. RUL estimates can sampled from the distribution, $r_t \sim p_{\theta_r}(r_t|z_t, x_{1:T})$

6. Monte Carlo simulation can be done by repeating steps 3-5, N times, for some "large" value N (although all N calculations are done in parallel in this work).

In this work $N = 100$ when testing the DVAE[1].

5.2 Experiments

5.2.1 Data

The data used to test this conditional DVAE for RUL estimation is the CMAPSS dataset described in section 4.1.1.

5.2.2 Preparing the Data

The sensors are normalised using operating-condition-based normalisation as mentioned in section 4.1.2, Eqn 4.2. Sensors that are constant values throughout the lifetime of the units are also dropped. Normalising and dropping sensors that do not aid RUL estimation are common practices employed for the experiments throughout the thesis. The maximum RUL was also set to 130 as before to represent the healthy portion of the unit's life.

The model must utilise the inputs $x_{1:T}$ and $r_{1:T}$ in a noncausal manner. To do this, a sequence-to-sequence approach was taken where the data was broken down into time-windowed portions using a sliding time window of size T. This sliding time window applied to the sensor data is illustrated in Figure 5.2. The different time-windowed data slices were combined into batches when training the model. Hence the size of the input tensors was, (batch size, T, number of sensors). Therefore, the model required a time window of data points, $x_{1:T}$, before estimating a time window of RUL estimates $r_{1:T}$. During the testing phase, the model is required to evaluate the entire sequence of sensors for each testing unit, each of which may have a variable sequence length. To apply this sequence-to-sequence model in this setting, a sliding time window goes over the test units sensor sequence and uses the generative model on each time window to find a sequence of RUL estimates for each time window. These estimates are then put back together into a single sequence, where the average RUL value is taken for the time points with overlapping time windows. An illustration of this is shown in Figure 5.3.

5.2.3 Experiments using DVAEs

Supervised Learning

In this experiment, the generative model is trained in a supervised manner. The inference, transition and measurement models are trained using the time windowed RUL sequences ($r_{1:T}$)

[1]The code implementing the DVAE and for the experiments mentioned in this chapter can be found at https://github.com/StarMarco/DVAE_torch or contact the author for more information

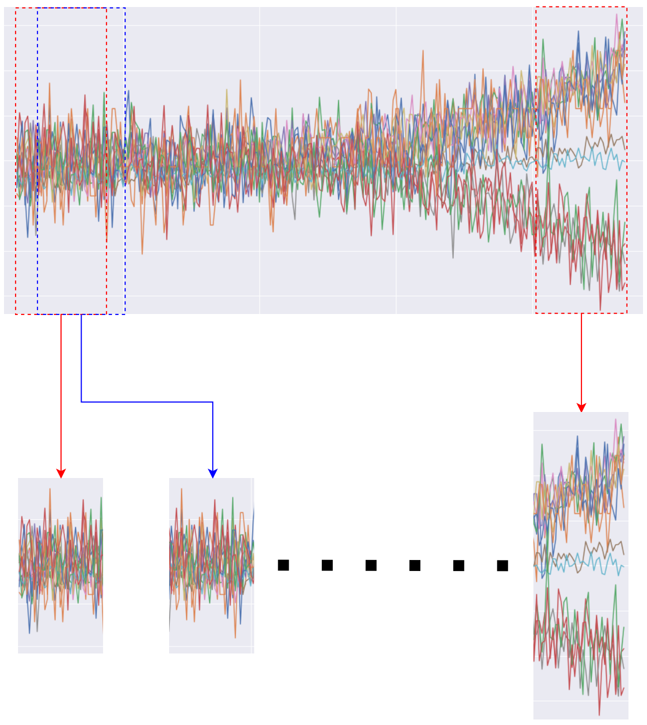


Figure 5.2. An illustration of a sliding time window applied to the sensor signals of a single unit in the CMAPSS dataset

as target values and the sensor sequences ($x_{1:T}$) as the input data. After training, the generative model estimates the RUL sequences from the sensor sequences for each unit in the testing dataset. Note that the generative model refers to the use of the transition and measurement model to propagate states z_t through time and convert to the measurement space r_t to find RUL estimates. Figure 5.1a illustrates this graphically in the previous section. Similarly to the previous chapter 4, the purpose of this experiment is to test the performance of this model for RUL prediction so that it can be compared to other state-of-the-art models. Section 5.1.2 in the methodology portion of this chapter explained the steps of training and testing the DVAE. Hence, this experiment applies those principles to train and test the DVAE using all the data available in the training and testing datasets. In contrast, the next section introduces the semi-supervised experiments which will artificially remove some of the target RUL data to simulate a lack of run-to-failure training data available for the practitioner.

Semi-Supervised Learning

The aim of this experiment is simply to see how DVAEs could potentially be used for semi-supervised learning to reduce the amount of target data needed to train the supervised DVAE. Hence, the results are mainly compared with the supervised DVAE from the previous experiment as a baseline. Although the results from this experiment may encourage further experimentation using DVAEs for semi-supervised learning in machinery prognostics applications. However, here it can be seen as a supplementary experiment to the supervised experiment, and the results do aid some of the arguments made in the supervised experiment's discussion. Namely, one of the competing methods uses semi-supervised learning techniques to improve their performance, and the results here help for a fair analysis of the results Listou Ellefsen

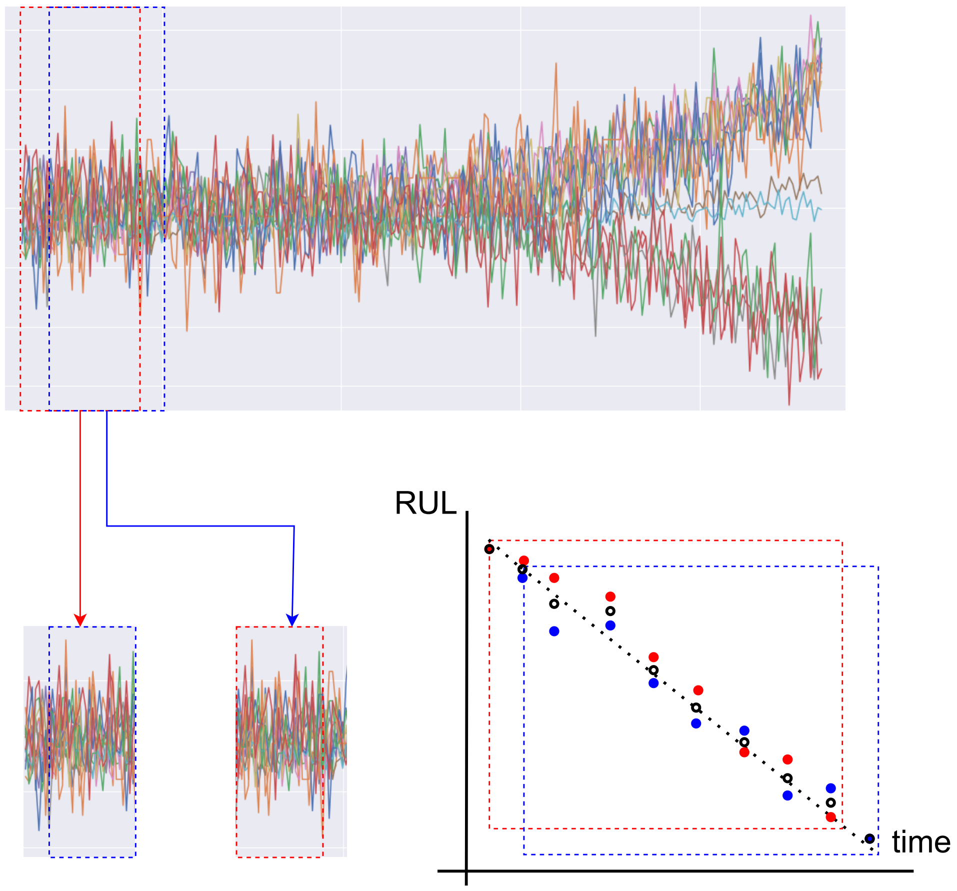


Figure 5.3. The overlapping time windows of sensor data each have a corresponding RUL estimate sequence. Some of the RUL estimates from different time windows have the same corresponding time points; in order to plot a single RUL estimate trajectory, the RUL estimates are averaged at the same time points. This is shown for the red and blue time windowed data. The red and blue points are the corresponding RUL estimates (made up for illustrative purposes) and the black circles are the average RUL estimates between the red and blue RUL estimates.

et al., 2019.

In this experiment, another DVAE is utilised as an unsupervised model which is trained to reconstruct the input sensor values. Typically many semi-supervised methods train a generative model in an unsupervised manner using the sensor data while a supervised model uses the outputs of that unsupervised model to estimate the RUL. For example, a VAE could be used to process the data, and then the latent variable output of the VAEs encoder could be used as an input to an MLP network that estimates the RUL. However, this still means the RUL estimate is a point estimate without consideration for its uncertainty. An interesting aspect of the DVAE is that it is a generative model; hence, a DVAE can be used to reconstruct the sensors and find the latent variable trajectories in an unsupervised manner, and then another DVAE could be constructed for probabilistic RUL estimation. The benefit of this is that, unlike the VAE, the DVAE is well-suited for sequential data. Hence, the DVAE should be better at the unsupervised processing of the sensor signals. During the supervised stage, the DVAE provides the benefit of capturing the uncertainties of the RUL estimates it produces by repeatedly sampling from the generative model. As the point of semi-supervised learning is for the unsupervised model to capture information from the input data to aid the supervised model in learning the targets, the DVAEs ability to handle sequential data should aid in this application.

Generally, semi-supervised methods in machinery prognostics define an unsupervised network and a supervised network (both DVAEs in this case) and train both simultaneously. Hence, here a DVAE is defined to reconstruct the sensor data, and the ELBO loss (Eqn. 3.9) is the unsupervised loss, while a conditional DVAE is used as the supervised network and the conditional ELBO (5.12) is the supervised loss. The total loss is taken as the addition of the supervised and unsupervised loss i.e.

$$L_{\text{Total}} = \mathcal{L}_{\text{unsupervised}} + \mathcal{L}_{\text{supervised}} \tag{5.15}$$

Where $\mathcal{L}_{\text{unsupervised}}$ is defined by Eqn. 3.9 and $\mathcal{L}_{\text{supervised}}$ is defined by Eqn. 5.12. However, to find a supervised and unsupervised loss, we must split the training data into sensor data with labelled RUL target pairs and sensors without target RUL data. Note that the training data has all the RUL target values available, so this experiment needs to manually discard some of these to test the semi-supervised method. The specifics of this data preparation for the semi-supervised experiment are as follows,

- The training data can be described as, $(\{x_t^{(n)}\}_{t=1}^T, \{r_t^{(n)}\}_{t=1}^T)_{n=1}^N$, where T is the time window, and N are the amount of time windowed batches for all the units in the dataset, e.g. FD001.
- Some of the targets will be "deleted" by replacing them with -1 so that the algorithm knows to ignore these values when calculating the supervised loss.
- Hence, this results in the following dataset, $(\{x_t^{(k)}\}_{t=1}^T, \{r_t^{(k)}\}_{t=1}^T)_{k=1}^K$, $(\{x_t^{(m)*}\}_{t=1}^T)_{m=1}^M$, where $M + K = N$ and K is the amount of input-target pairs left and M is the amount of time windowed sensor batches without targets (practically the targets were replaced with -1).
- During training, all the sensor data can be used in the unsupervised phase. However, the sensor data batches with corresponding batches of -1 targets are dropped during the supervised phase, so only $(\{x_t^{(k)}\}_{t=1}^T, \{r_t^{(k)}\}_{t=1}^T)_{k=1}^K$, are used as inputs to the supervised network, while all $(\{x_t^{(n)*}\}_{t=1}^T)_{n=1}^N$ is used in the unsupervised stage.

Note that the targets chosen to be dropped are randomly chosen over all the data points in the dataset regardless of the unit it belongs to or its corresponding time point. This was mainly done to stay consistent with other works in the literature that test semi-supervised methods. The different percentages of input-target pairs kept for training the semi-supervised model are 1%, 5%, 10%, 50% and 100% (i.e. the percentage of supervised data = $\frac{K}{N} \times 100$). Hence, this gives us a variety of tests that can assess the model's performance on various amounts of run-to-failure data availability.

An unsupervised model is used in the semi-supervised experiment to learn information from the unlabelled data and assist the supervised model. In this case, the unsupervised model will be a DVAE (but not conditional like the RUL estimation supervised DVAE), and it can reconstruct the sensor values and generate latent variables $z_{1:T}$ to aid the supervised model. The latent variable outputs from the unsupervised DVAEs inference model will be used along with the original sensor values as inputs to the supervised DVAE that estimates the RUL. This supervised DVAE is essentially the same type of model described in the supervised experiment; however, instead of only having the sensor sequences $x_{1:T}$ as conditional inputs, both the unsupervised model latent variables and sensors are used as conditional inputs, $[x_{1:T}, z_{1:T}]$.

5.2.4 Network Design

Supervised Experiment

A bi-directional GRU encodes the sensors for each time window, $x_{1:T}$. This means each hidden variable represents the past, current and future variables, $\overleftarrow{h}_t^{(x)} := x_{1:T}$, which makes this a noncausal model. For the inference model that is used during the training of the DVAE, the measurements/known RUL target values are also encoded by a separate bi-directional GRU to capture the time windowed target values $r_{1:T}$. A GRU RNN is used for the inference model where each hidden variable represents the latent variable $h_t = z_{1:t}$ and the inputs to the RNN are the encoded values for $x_{1:T}$. Note all the hidden variables of these RNNs are initialised as zero, i.e. $h_0 = 0$. Both the transition and measurement models are modelled using MLP networks. Hence, as a summary, the models are listed below,

- Encoder for $x_{1:T}$: $BiDirectionalGRU(x_t, h_t^{(x)}, \overleftarrow{h}_t^{(x)}) = \overleftarrow{h}_{t-1}^{(x)}$
- Encoder for $r_{1:T}$: $BiDirectionalGRU(r_t, h_t^{(r)}, \overleftarrow{h}_t^{(r)}) = \overleftarrow{h}_{t-1}^{(r)}$
- Inference model, $q_\phi(z_t|z_{1:t-1}, r_{1:T}, x_{1:T})$: $MLP\Big(GRU(h_{t-1}, \overleftarrow{h}_t^{(r)}, \overleftarrow{h}_t^{(x)})\Big)$ $= MLP(h_t) = \Big[\mu_{(z_t)_{inf}}, \log\sigma^2_{(z_t)_{inf}}\Big]$
- Transition model, $p_{\theta_z}(z_t|z_{t-1}, x_{1:T})$: $MLP(z_{t-1}, \overleftarrow{h}_t^{(x)}) = \big[\mu_{z_t}, \log\sigma^2_{z_t}\big]$
- Measurement model, $p_{\theta_r}(r_t|z_t, x_{1:T})$: $MLP(z_{t-1}, \overleftarrow{h}_t^{(x)}) = \big[\mu_{r_t}, \log\sigma^2_{r_t}\big]$

The background section 3.2.1 notation is used here for the mean and standard deviation. Note the networks with multiple inputs concatenate these inputs, so the network has a single input variable comprising multiple concatenated variables. This was all implemented using PyTorch as the deep learning software (Paszke et al., 2017).

Semi-Supervised Experiment

In the semi-supervised experiment, a DVAE is used to learn the sensor signal trajectories in an unsupervised manner. This unsupervised DVAE has the following model forms,

- Encoder for $x_{1:T}$: $BiDirectionalGRU(x_t, h_t^{(x)}, \overleftarrow{h}_t^{(x)}) = \overleftarrow{h}_{t-1}^{(x)}$, Note, the forward pass of the RNN produces $h_t^{(x)} := x_{1:t-1}$
- Inference model: $q_\phi(z_t|z_{1:t-1}, x_{1:T})$: $MLP\Big(GRU(h_{t-1}, \overleftarrow{h}_t^{(x)})\Big)$ $= MLP(h_t) = \Big[\mu_{(z_t)_{inf}}, \log\sigma^2_{(z_t)_{inf}}\Big]$
- Transition model: $p_{\theta_z}(z_t|z_{t-1}, x_{1:t-1})$: $MLP(z_{t-1}, h_t^{(x)}) = \big[\mu_{z_t}, \log\sigma^2_{z_t}\big]$
- Measurement model: $p_{\theta_x}(x_t|z_t, x_{1:t-1})$: $MLP(z_{t-1}, h_t^{(x)}) = \big[\mu_{r_t}, \log\sigma^2_{r_t}\big]$

Hence, this unsupervised generative model is a causal model like the DVAEs mentioned in the background section 3.2. Note, as this DVAE is a generative model, it aims to learn how to sample from the distribution $p(x_{1:T})$, i.e. distribution of possible sensor values.

The supervised DVAE has the same form as the conditional DVAE used in the supervised experiment. The only difference is that the conditional inputs to this supervised model are the sensor variables and latent variables from the unsupervised inference model. This matches the methodology of other semi-supervised models that train a VAE and use the encoder output (inference model output in this context) as the input to the supervised RUL estimation model (Martin & Droguett, 2022; Costa & Sánchez, 2021).

5.3 Results

5.3.1 Supervised

This section compares the supervised DVAE with other state-of-the-art RUL estimation methods. Many of the best-performing RUL estimation methods in the machinery prognostics literature are not probabilistic. An exception here is another model that performs well on the dataset, known as Deep Learning Non-stationary Gaussian Process Regression (DL-NSGPR) (Z. Xu et al., 2021). It uses Non-Stationary Gaussian Process Regression (NSGPR) after a stage where a deep neural network estimates point RUL estimates. The NSGPR helps smooth the estimates, deal with noise in the point estimates, and quantify uncertainty. The rest of the models use various neural networks such as CNNs on spectrogram images of the sensor data, created using time-frequency transforms (X. Li et al., 2018), A semi-supervised method that preprocesses the data using a Restricted Boltzmann Machine (RBM) and then uses an LSTM to predict the RUL (Listou Ellefsen et al., 2019), a Temporal Deep Degradation Network (TDDN), which uses CNNs and Attention mechanisms and finally, a Dual-Stream Self-Attention Neural Network (DS-SANN), which uses Attention mechanisms to estimate the RUL from the sensor signals. Table 5.1 compares the performance of the different models using the RMSE metric found in equation 5.16.

$$RMSE = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2} \tag{5.16}$$

where N is the total amount of RUL targets in the testing dataset, $\hat{y}_n$ is the RUL estimate (or the mean estimate in the case of probabilistic models) and y_n is the corresponding true RUL value.

Table 5.2 shows the comparison between the DVAE and other selected model's score values. The expression for calculating the score can be stated as (Saxena et al., 2008),

$$s = \begin{cases} \sum_{i=1}^{N} \exp\left[\frac{-(\hat{r}_i - r_i)}{13}\right] - 1, & \text{if } \hat{r}_i \leq r_i \\ \sum_{i=1}^{N} \exp\left[\frac{(\hat{r}_i - r_i)}{10}\right] - 1, & \text{if } \hat{r}_i > r_i \end{cases} \tag{5.17}$$

Table 5.1: RMSE performance on each of the testing CMAPSS datasets

Method	FD001	FD002	FD003	FD004
DVAE (This work)	8.63	**10.29**	**7.28**	9.61
DL-NSGPR (Z. Xu et al., 2021)	**7.4**	11.8	7.5	**8.3**
CNN (X. Li et al., 2018)	12.61	22.36	12.64	23.31
RBM-LSTM (Listou Ellefsen et al., 2019)	12.56	22.73	12.10	22.66
TDDN (Qin et al., 2022)	9.47	10.93	9.17	11.16
DS-SANN (D. Xu et al., 2022)	11.64	13.34	12.28	14.98

Note, to be consistent with other models reporting scores only the final RUL estimate is used to calculate the score of each testing unit. Hence, N in the context of the score is equal to the number of units in each dataset. The score function is used to penalise RUL estimates larger than the true value so that conservative results would result in a lower value. Note not all the methods from the RMSE performance table (Table 5.1) are in this table as not all of those methods listed the scores of their model.

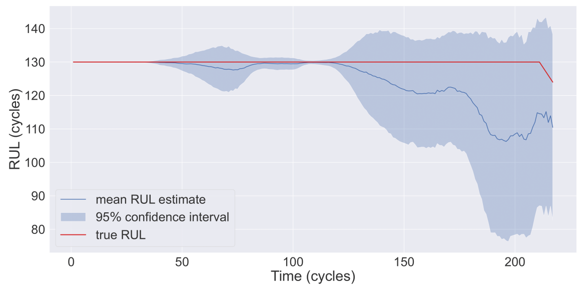
Table 5.2: Comparison of the scores (Eq. (5.17)) on each of the testing CMAPSS datasets

Method	FD001	FD002	FD003	FD004
DVAE (This work)	376	1791	533	1403
CNN (X. Li et al., 2018)	273	10412	284	12466
LSTM (Listou Ellefsen et al., 2019)	231	3366	251	2840
TDDN (Qin et al., 2022)	214	**562**	**217**	**998**
DS-SANN (D. Xu et al., 2022)	**210**	867	286	1150

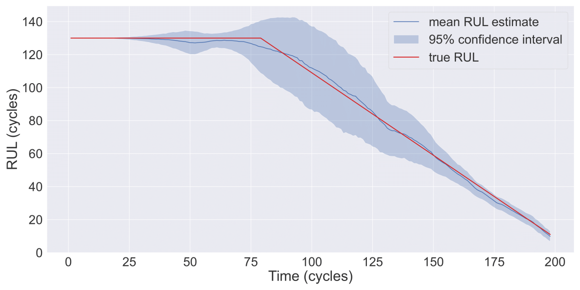
Figure 5.4 shows the RUL plots over the available operating time for certain testing units in FD001. Units 12 and 41 are chosen as they represent the longest and the shortest amount of time a unit is healthy respectively (a RUL value of 130 is taken as "healthy"). Units 35 and 100 were randomly chosen to showcase some results that aren't extreme cases, like units 12 and 41.

Note that the RUL estimation trends for units 12 and 41 shown in Figures 5.4a and 5.4c deviate from the true RUL trend at certain points. Unit 12's RUL estimate trend starts to deviate after cycle 100, while unit 41's trend stays constant until around cycle 40 despite the true RUL beginning to degrade near the start. This could be due to the fact not all the units begin to degrade when RUL equals 130. Hence, unit 12 may actually start to degrade after cycle 100, which is reflected in the estimated RUL trend. Unit 41 could transition from a healthy state to a degrading one at around cycle 40, and this could be why the DVAE keeps estimating a RUL of 130 until cycle 40. While this is not explored in this thesis, a way of testing this is to find out when the unit transitions from a healthy state to a degrading one and begin the linearly decaying RUL trend at that point instead of setting it as 130 for every unit. This is similar to certain methods in bearing prognostics that use calculated vibration statistics such as RMS or kurtosis to determine when a fault occurs and the degradation process begins (Cao et al., 2023; J. Guo et al., 2023).

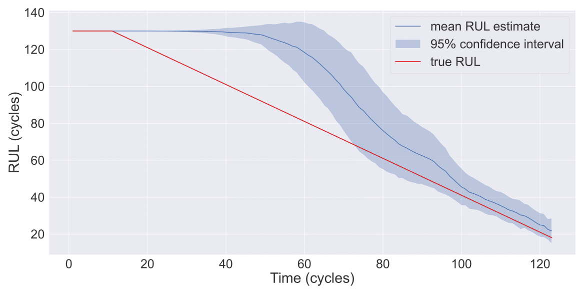
The dimensionality of the latent variables was chosen as 2. Hence, this creates an opportunity to plot latent phase-space plots with each latent variable represented by an axis. In Figure



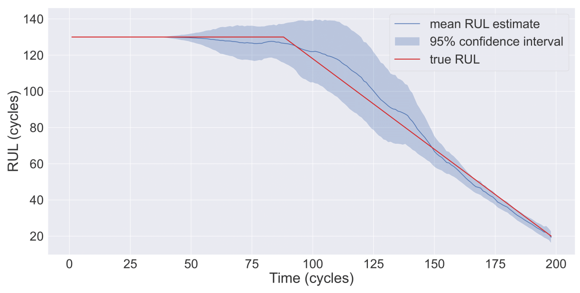
(a) RUL estimates vs lifetime of unit 12



(b) RUL estimates vs lifetime of unit 35



(c) RUL estimates vs lifetime of unit 41



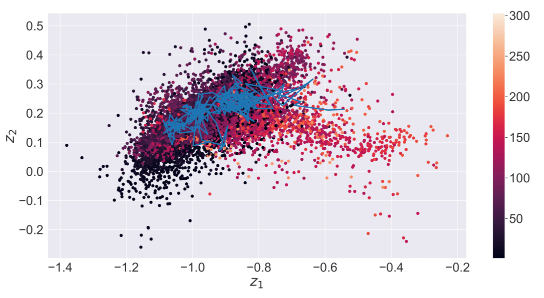
(d) RUL estimates vs lifetime of unit 100

Figure 5.4. The RUL estimates over the known lifetime of the testing units for a selected subset of units in the FD001 test dataset.

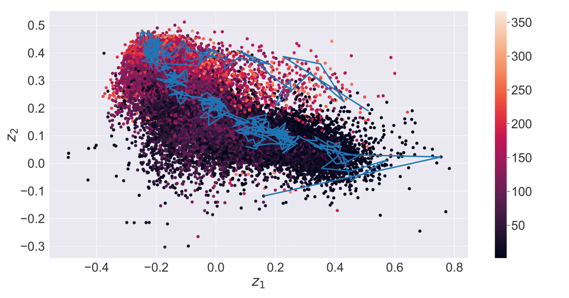
5.5, the latent phase-space trajectories for each unit are plotted as a scatter plot with a colour gradient showing their evolution in time. All the unit's latent trajectories are plotted for each dataset in CMAPSS to show a general trend in the latent space as the machines degrade. Note not all the units in the testing datasets get close to degrading; some units, like unit 12 in FD001, stay healthy for the majority of the time and barely degrade at all during the time captured in the test. Hence, there will be more points closer to the healthy regions where the trajectories start from, and fewer points as the trajectories evolve from the machine degradation.

The final RUL estimates can give a useful visual indicator of the model's overall performance. For example, Figure 5.6 shows the final RUL estimates versus the True RUL for that unit in the FD001 dataset as a scatter plot. This helps visualise how well the model performs as the machine approaches the failure point. For example, from the figure, it is easy to see as the machine gets closer to the failure point of RUL = 0, the model performs better as the points are less scattered from the line that denotes the accurate values.

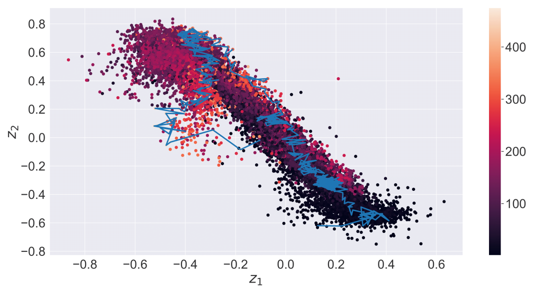
Finally, another use for the final RUL values is that it indicates how well the uncertainty captures the true RUL for each unit. Figure 5.7 shows the RUL estimates along with 95% confidence intervals as error bars. Hence, at a glance, it can be seen if the true RUL values are captured within the error bars that denote the model's confidence. It can also be seen that for larger final RUL values, the confidence intervals also seem to be large, while for the smaller RUL values the error bars are smaller. This helps confirm the statement from the last plot (Figure 5.6) that as the machine approaches failure, the model becomes more confident in its RUL estimates.



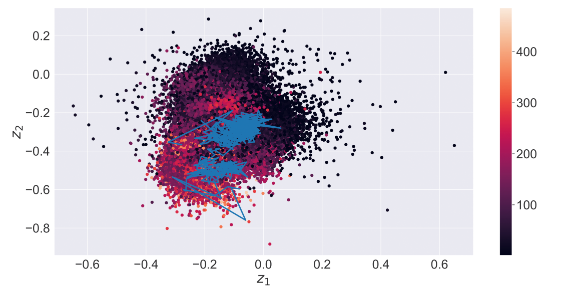
(a) Latent state space trajectories for FD001



(b) Latent state space trajectories for FD002



(c) Latent state space trajectories for FD003



(d) Latent state space trajectories for FD004

Figure 5.5. A plot of the latent state space trajectories for the test units in each dataset. The colour bar represents the time i.e. the darker colours represent earlier time points, and the lighter ones show later time points. The longest trajectory is plotted as a blue line to show how a unit may develop from a healthy to near-failure condition.

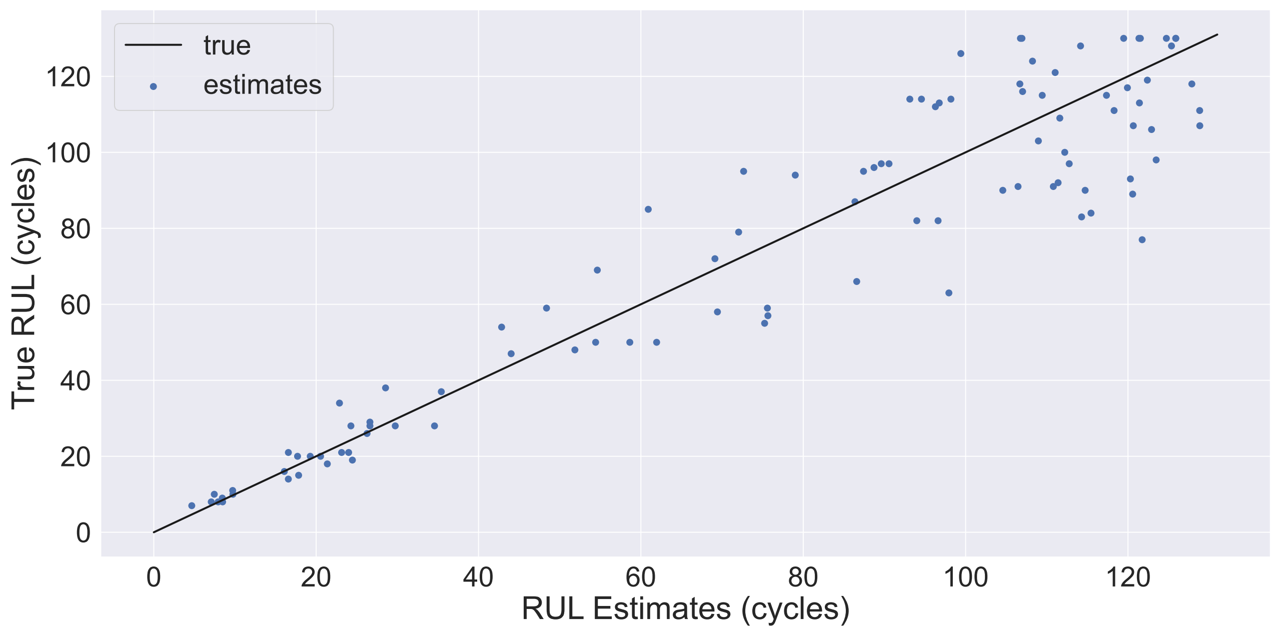


Figure 5.6. The final RUL estimates for the test units in the FD001 dataset. These were plotted as a scatter plot and compared with the true RUL values plotted as a line

5.3.2 Semi-Supervised

Comparing different approaches for semi-supervised learning is difficult due to the different evaluation setups used within the field (Krokotsch et al., 2022). In Table 5.3, we have listed

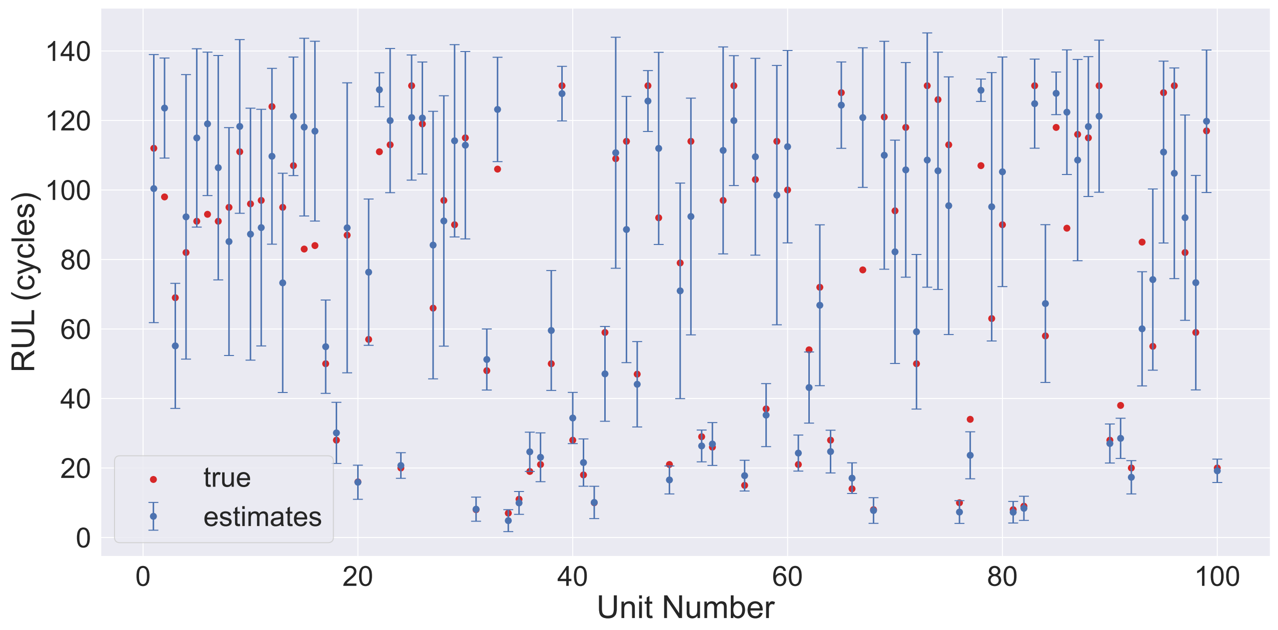


Figure 5.7. The final RUL mean estimates and True RUL values plotted as a scatter plot vs the unit index for each unit in the FD001 test dataset. The 95% confidence intervals are also shown for the final RUL estimate.

some of the RMSE results for the different percentages of labelled data points randomly chosen in each of the CMAPSS datasets.

Table 5.3: Semi-supervised RMSE performance on each of the testing CMAPSS datasets for different percentages of available data during RUL estimation training.

Dataset	1%	5%	10%	20%	50%	100%
FD001	14.88	9.13	9.35	8.68	8.78	8.53
FD002	10.86	10.29	10.20	10.01	9.96	10.15
FD003	12.99	9.47	8.52	8.43	7.88	7.33
FD004	15.73	9.72	9.69	9.53	9.96	9.77

Similarly, the scores are presented in Table 5.4 and show how the score metric changes with a given percentage of RUL labels.

Table 5.4: Semi-supervised NASA score function performance on each of the testing CMAPSS datasets for different percentages of available data during RUL estimation training.

Dataset	1%	5%	10%	20%	50%	100%
FD001	842	265	256	349	245	234
FD002	1853	1057	783	788	800	839
FD003	880	671	291	467	270	314
FD004	2505	1203	1032	1183	1113	1060

While semi-supervised learning is not the focus of the thesis, these results may go show the potential effectiveness of using DVAE methods for semi-supervised RUL estimation. Semi-supervised methods improve the performance of the RUL estimation models if there is little run-to-failure data available. Hence, the DVAEs capability of dealing with sequential data could aid in reducing the expense of requiring a lot of run-to-failure data to train these deep-learning models.

5.4 Discussion

Regarding RMSE performance, the DVAE performs better than all the models that do not account for uncertainty. The only model whose performance is better for some of the datasets is DL-NSGPR, which is also a relatively recent model that also accounts for uncertainties in the RUL estimate. However, DVAE also has a latent space that can potentially act as a HI and give some intuition behind the RUL estimates. DVAEs are also not restricted to Gaussian distributions to represent the RUL estimates like DL-NSGPR is. Using the DVAEs generative model and sampling multiple $r_{1:T}$ estimates can represent any arbitrary probability distribution using those samples. Another strength of this DVAE model is that it provides an underlying theoretical framework. Looking at the theory behind DVAEs, others can use the same theory while finding different ways to achieve the underlying mathematical structure of the model. For example, instead of using a bidirectional RNN to encode $x_{1:T}$ into a hidden variable, perhaps another network architecture could be used, such as a Transformer (Vaswani et al., 2017). Or perhaps use different assumptions than the one used here, e.g. use transition model $p_{\theta_z}(z_t|z_{1:t-1}, x_{1:T})$ instead of $p_{\theta_z}(z_t|z_{t-1}, x_{1:T})$. A useful aspect of using these sequential conditional generative modelling techniques, such as DVAEs, is that there can be some flexibility in how each model is constructed (e.g. what network architectures are used) while still guided by the overarching framework. Hence, the main benefit of this work is the DVAE framework and understanding that noncausal models are required for the generative model to learn how to sample from the underlying distribution $p(r_{1:T}|x_{1:T})$. The network architectures stated here are less important and were only chosen to test the model's effectiveness using relatively simple network architectures used in other DVAE models (Girin et al., 2021).

The structure of the DVAE can be compared to the other methods listed in Table 5.1. The TDDN Qin et al., 2022 used 1D convolutional layers and an attention mechanism to extract features and uses them as inputs to an MLP network that estimates the RUL. Similarly, DS-SANN D. Xu et al., 2022 uses attention mechanisms and an MLP network to estimate the RUL. The CNN method X. Li et al., 2018 uses the short-time Fourier transform on the sensors to construct a spectrogram image and applies the CNN to estimate the RUL based on the image inputs. In these cases, time windows were used on the input sensors, but instead of a sequence-to-sequence model, the RUL of the final time point was estimated given a sequence of sensor inputs. The DVAE manages to outperform these methods in terms of the RMSE and captures the uncertainty of the RUL. Part of this may be due to the noncausal assumption. Note that the method with comparable performance is the DL-NSGPR Z. Xu et al., 2021, which uses NSGPR on the MLP network outputs to account for uncertainty. An MLP network would not achieve relatively accurate RUL estimates on its own, e.g. Pasa et al., 2019 report the RUL estimation results of MLP applied in a sequence-to-sequence fashion and achieve an RMSE of 15.1 on FD001, 18.0 on FD002, 14.3 on FD003 and 16.6 on FD004 (with operating condition normalisation hence, their FD002 and FD004 results may be better than some of the methods listed in Table 5.1). However, applying NSGPR on top of that seems to improve the results. Note that one could look at Gaussian Process as a Stochastic Differential Equation, and Gaussian Process Regression applies a Kalman filter and smoother based on observed data Sarkka et al., 2013. Note Bayesian smoothing is a noncausal operation and relies on future values to estimate current ones and also estimates its parameters using the likelihood of the data like the DVAE does through minimising Eqn. 5.1 (although it is done indirectly through the ELBO in the DVAE case). There appear to be some similarities between these methods as they both use noncausal models to estimate the RUL and optimise their parameters using a marginal likelihood. Hence, this may be why they give similar RMSE results, as they have a similar noncausal structure and are optimised on the same underlying loss function.

However, the model's final RUL estimate scores in each dataset does not produce state-of-the-art score results in the supervised case. Note that one of the methods used for comparison, (Listou Ellefsen et al., 2019), uses semi-supervised learning as a part of their model. Hence, when looking at the semi-supervised scores, one can see that they become more comparable. However, for the rest of the score results it can be seen that the DVAE does not perform as well as the others. While the supervised DVAE may lack accuracy for the final mean RUL estimates, it does account for uncertainty in its estimates and can therefore give confidence bounds. The score is a measure that rewards more conservative models; the DVAE however gives uncertainty for its estimates which the better scoring models do not. Hence, this can help with the interpretability of the RUL estimates and also allow for more conservative estimates if need be. Figure 5.7 shows that many of the actual testing RUL values were captured within the confidence bounds. Therefore, while the model may lack the same capabilities as other models in terms of score it does gain a confidence interval that can effectively capture the actual RUL value of the machine.

In (Wei et al., 2021), a conditional DVAE is also used to estimate the RUL, but the model does not use a noncausal model to represent the sensor sequences, instead it assumes a causal model. They also assume the decoder or measurement model, p_{θ_r}, is only dependent on the current latent variable, $p_{\theta_r}(r_t|z_t)$, whereas we assumed the model to be $p_{\theta_r}(r_t|z_t, x_{1:T})$. It should be noted that if we also assume the decoder to be $p_{\theta_r}(r_t|z_t)$, while keeping the rest of the models the same, the RMSE values suffer slightly but the latent variables follow a more obvious degradation trend. Hence, since (Wei et al., 2021) was focused on creating a health indicator, this was a good assumption to make. However, this paper showcases that by starting with the underlying theory and being aware of the assumptions, one can know the trade-offs made during the model's construction.

With regards to the semi-supervised experiment, the results showcase how, even with little labelled data available, the trained DVAE was still able to obtain accurate probabilistic RUL estimates. Although note, the experiment done here was more a proof of concept than a more detailed investigation as was done with the supervised methods throughout the thesis. It can also be noted that even at around 5% available labels, the semi-supervised model's performance has already started to plateau. Although, further work could be done to reduce the variability of results as some of the results get slightly worse after more labels are added and then get better again after adding more labels e.g. see FD001 in Table 5.3 as the results oscillate from better to worse as more labels are added. Perhaps a less random test could be done where instead of randomly choosing the targets to drop each time, a set list of targets is consistently dropped for each experiment or a few units are chosen and all target data from those units are dropped. It should be noted that the results in the tables are averages over ten different training runs to attempt to avoid the impacts of randomness. However, the results indicate that more may need to be done to effectively eliminate randomness from this experiment. As previously mentioned, comparison of semi-supervised models throughout the literature can be difficult due to the different evaluation setups they can have (Krokotsch et al., 2022). Hence, the supervised DVAE model can serve as a baseline here, where it is easy to see that the semi-supervised models benefited from the unsupervised stage in the 100% case. It can also be seen that the RMSE performances of the semi-supervised model at the other label percentage cases were similar to the supervised models except at 1% where they had a noticeable drop in performance (if not in RMSE then in score).

5.5 Conclusions

This chapter has shown the effectiveness of using a conditional sequential generative model, specifically the DVAE, for estimating the RUL sequences given the sensor signals. Conditional sequential generative models train neural networks to sample from some underlying distribution $p(r_{1:T}|x_{1:T})$. Hence, by constructing a DVAE and choosing the RUL as our input variables, $r_{1:T}$, and the sensor signals as the conditional variable $x_{1:T}$, we were able to train the DVAE to sample from the underlying distribution of RUL estimate sequences given the sensor signals. Most deep learning machinery prognostics methods aim to estimate the RUL given the sensor signals; however, by using a generative model the neural networks were trained to learn how to sample from a probability distribution instead of giving point estimates. The effectiveness of using this generative model approach is shown in the results, where the DVAE was capable of outperforming most of the state-of-the-art methods except on the FD001 and FD004 datasets, where DL-NSGPR did better.

The other benefit of using DVAEs was to learn the latent sequences that could act as trajectories that indicate the degradation of the machine. This gives it an advantage over many of the state-of-the-art methods presented as they do not also produce a HI trajectory. Although the HIs produced by the DVAE were not the main focus here and could potentially be improved beyond simply giving a visual intuition of the machinery degradation. Further work could be done to improve these latent dynamics, such as making the trajectories smoother or perhaps having linear dynamics in the latent space. Improving these latent dynamics may be a potential way to improve the performance of the DVAE model or maybe a way of opening up possibilities of understanding the mechanisms behind the machinery's degradation. Another research avenue to explore is improving the semi-supervised method presented here. Note both the inference model from the unsupervised model and the transition model from the supervised model could potentially have the same structure i.e. $q(z_t|z_{1:t-1}, x_{1:T})$ and $p(z_t|z_{1:t-1}, x_{1:T})$. Hence, if these are both modelled by the same network, the parameters of the network can be optimised during the unsupervised phase which could pre-train the network so that it is closer to an optimal model when targets are available and supervised learning is done. This sort of pretraining of model parameters is a known method in the semi-supervised literature known as unsupervised preprocessing (van Engelen & Hoos, 2020). Because both the unsupervised and supervised models are DVAEs and have a similar structure it allows for this type of pretraining to be implemented relatively easily.

Since the DVAE can capture the uncertainty behind our RUL estimates, the next chapters can look at improving these latent dynamics. This could be done by having latent dynamics that aren't expressed only by a latent 1-step transition function as was done here by using an

MLP network $p_{\theta_z}(z_t|z_{t-1}, x_{1:T})$ to transition from z_{t-1} to z_t by some fixed time interval. Hence, the next chapter will use Neural Differential Equations to describe continuous latent dynamics. Using differential equations may also aid in understanding the dynamics through the lens of differential equations that are relatively well understood and are often used to describe dynamic systems.

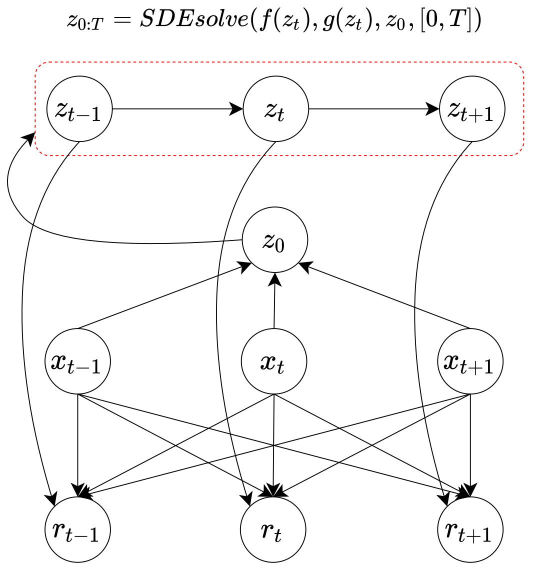
Chapter 6

Dynamical Variational Autoencoders with Neural Stochastic Differential Equations

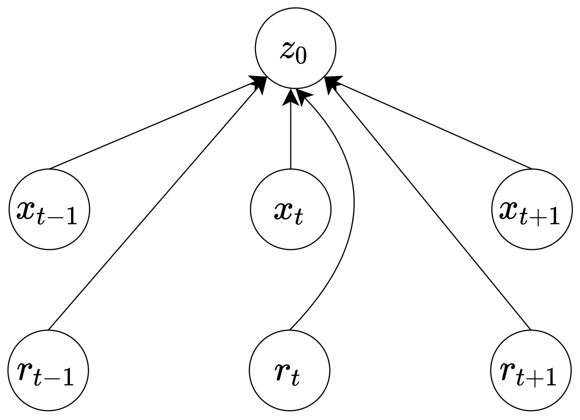
In this section, a conditional DVAE is used to estimate the RUL, conditioned on the sensor input data much like in Chapter 5. However, in this chapter, a different implementation of the DVAE is explored. This chapter uses NSDEs as the transition model to express continuous dynamics instead of the discrete ones used in the standard DVAE for RUL estimation. Hence, here the transition works by using a neural network to represent, $\frac{dz_t}{dt}$, and using the differential equation to propagate the latent states forward in time. Many statistical methods in machinery prognostics use SDEs to model the degradation of the machine (Si et al., 2011). Using NSDEs to model the latent dynamics, which in a DVAEs case represent the degradation of the machine, could complement the statistical approach to machinery prognostics. The fundamentals however remain the same; a DVAE is trained to sample from some underlying distribution, which in the machinery prognostics case, is $p(r_{1:T}|x_{1:T})$. This chapter explores how changing the transition models dynamics to these NSDE dynamics can alter the DVAEs latent dynamics and performance.

6.1 Methodology

The model used for estimating the RUL is based on the DVAE. Much like the DVAE, noncausal transition, measurement and inference models are required. Hence, this will be a sequence-to-sequence model that uses a sliding time window as the DVAE did in Chapter 5. The difference here is that the transition model will be an NSDE and so it needs an initial condition to generate a sequence of latent variables using the Neural Differential Equation. This will be done by encoding the sequences in a noncausal manner as with the DVAE's inference model but the output will be a single initial latent variable z_0, i.e. $q_\phi(z_0|x_{1:T}, r_{1:T})$. Additionally instead of a transition model, $p_{\theta_z}(z_t|z_{t-1}, x_{1:T})$, we will use a initial condition model $p_{\theta_{z_0}}(z_0|x_{1:T})$ along with a transition model described by the NSDE, $p(z_t|z_{t-1})$. Hence, the noncausal conditions are captured in the initial latent variables. Figure 6.1 shows the generative and inference models for the NSDE-DVAE. The diagrams show the variable dependencies and how to generate the variables needed to estimate the RUL.



(a) Generative model



(b) Inference model

Figure 6.1. The general diagram of the NSDE-DVAE for RUL estimation, where 6.1a is the generative model and 6.1b is the inference model. Note only times $t-1$, t and $t+1$ are shown for convenience

From Figure 6.1 the following forms can be inferred for the generative and inference models,

$$\text{Generative: } p_\theta(r_{1:T}, z_{1:T}|x_{1:T}) = \prod_{t=2}^{T} p_{\theta_r}(r_t|z_t, x_{1:T}) p_{\theta_z}(z_t|z_{t-1}) p_{\theta_{z_1}}(z_1|x_{1:T}) \tag{6.1}$$

$$\text{Inference: } q_\phi(z_{1:T}|r_{1:T}, x_{1:T}) = \prod_{t=2}^{T} p_{\theta_z}(z_t|z_{t-1}) q_\phi(z_1|r_{1:T}, x_{1:T}). \tag{6.2}$$

The generative model comprises the decoder and the prior (as is the case with the general DVAE model described in the background section). Notice that the noncausal sequences are accounted for through the model that defines the initial latent variable, z_1. Both the inference and transition model use the same transition dynamics $p_{\theta_z}(z_t|z_{t-1})$, but use different networks to define the initial condition. Hence, the ELBO formula for this DVAE would be,

$$\begin{aligned}\mathcal{L}(\theta_r, \theta_{z_1}, \phi, x_{1:T}, r_{1:T}) = &-\sum_{t=1}^{T} \mathbb{E}_{q_\phi(z_{1:t}|r_{1:T}, x_{1:T})} \left[\log p_{\theta_r}(r_t|z_t, x_{1:T})\right] \\ &+ \beta \sum_{t=1}^{T} \mathbb{E}_{q_\phi(z_{1:t-1}|r_{1:T}, x_{1:T})} \left[D_{KL}\left(q_\phi(z_1|r_{1:T}, x_{1:T}) || p_{\theta_{z_1}}(z_1|x_{1:T})\right)\right].\end{aligned} \tag{6.3}$$

Note here the hyperparameter β is used to help regularise the loss function as is done in β-VAEs (Higgins et al., 2017). Setting this value for some of the datasets helped improve the overall performance.

As before we aim to define a conditional DVAE that is trained to sample from the underlying conditional distribution $p(r_{1:T}|x_{1:T})$.[1].

6.2 Experiments

The supervised experiment will be performed where the network is given all the training sensors and targets and aims to estimate the RUL of the testing datasets given the sensors. Hence, like the DVAE chapter the supervised experiment is performed but no semi-supervised experiments are done here.

6.2.1 Data

The data used to test the DVAE method which will be described in this section is the CMAPSS dataset previously mentioned in section 4.1.

6.2.2 Preparing the Data

The data is also normalised using operating condition-based normalisation as shown in Eqn. 4.2. Like the DVAE from the previous chapter, the sensor signals from each unit are split into time windows of size T through the use of a sliding time window. Given a batch size N; each batch of training data, therefore, contains N signals of size T. Like the previous methods, the maximum RUL value is 130. The model is tested like the DVAE where a sliding time window goes over the testing unit sensor values and the corresponding RUL sequences are generated. As was shown in Figure 5.3 the overlapping RUL estimates for each time window are averaged for each of the RUL estimates with the same corresponding time point.

6.2.3 Network Design

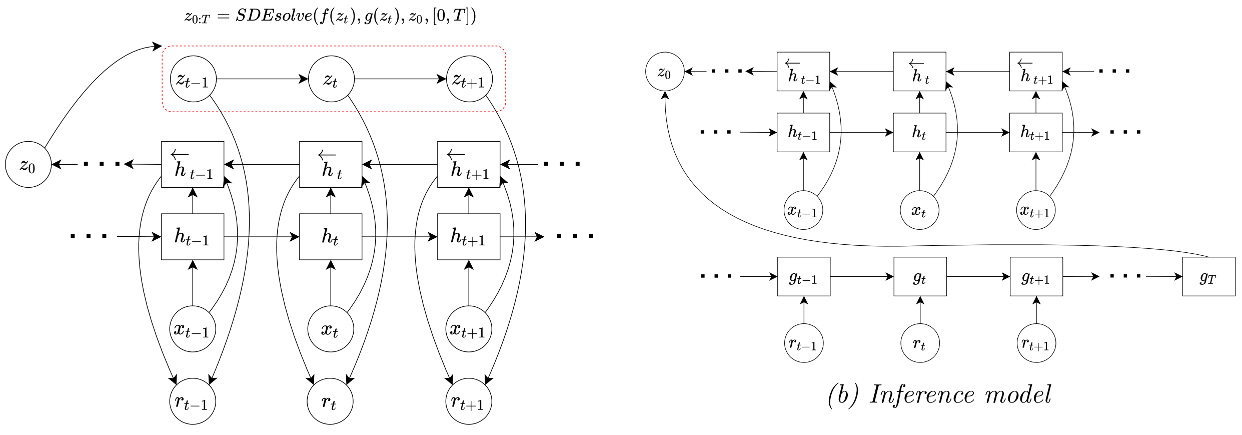
The specific implementation details of the NSDE-DVAE for RUL estimation used in this paper are addressed in this section. Looking at the dependencies for the generative and inference models in Eqns 6.1 and 6.2, it can be seen the following problems must be addressed.

- The variable $x_{1:T}$ needs to be expressed as a variable for the decoder, prior and inference models.
- The transition $p_{\theta_z}(z_t|z_{t-1})$ needs to be modelled.
- $r_{1:T}$ needs to be expressed for the inference model.

[1]The code implementing the NSDE-DVAE can be found at https://github.com/StarMarco/NSDE_DVAE or contact the author for more information

Figure 6.2 showcases a more detailed version of the NSDE-DVAE. From the figure, the following can be seen in regard to the implementation details,

- Expressing $x_{1:T}$ is achieved the same way as in the DVAE. A Bidirectional GRU is used to capture the entire sequence within the time window.
- As mentioned previously, the NSDE is used to model the transition $p_{\theta_z}(z_t|z_{t-1})$.
- For the inference model, $r_{1:T}$ is needed. This is done by using a single forward RNN and using the last hidden variable of that RNN (as it would represent $r_{1:T}$). Since $r_{1:T}$ is only needed as an input to generate z_1 in the inference model, there was no need to use a bidirectional GRU to find a representation of $r_{1:T}$ at each time point, t. Hence, this (along with $x_{1:T}$) is then used to infer the initial value of the NSDE z_1.



(a) Generative model

(b) Inference model

Figure 6.2. The detailed diagram NSDE-DVAE for RUL estimation where 6.2a is the generative model and 6.2b is the inference model (note during training, the inference models estimate for z_0 is used in the SDE solver to generate the sequence $z_{1:T}$)

The NSDE drift function uses a state-space model formulation to create a 2^{nd} order drift function i.e.

$$(\dot{z}_1)_t = (z_2)_t \tag{6.4}$$

$$(\dot{z}_2)_t = f_{\theta_z}\left((z_1)_t, (z_2)_t\right) \tag{6.5}$$

where f_{θ_z} is a standard feed-forward neural network. This allows for the variables z_1 and z_2 to be related to each other. For example, if z_1 indicated the machinery's degradation z_2 would be the velocity at which it degrades, which would be useful information when trying to estimate the RUL. The diffusion is modelled as being proportional to the current state and optimises the constant variable ($\boldsymbol{\theta}_d$) during training. This can be expressed as,

$$g\left((z_1)_t, (z_2)_t\right) = \boldsymbol{\theta}_d \mathbf{z}_t \tag{6.6}$$

where, $\mathbf{z}_t = [(z_1)_t, (z_2)_t]$ and $\boldsymbol{\theta}_d$ is the trainable parameter.

Another important note about the NSDE used for the dynamics is that an augmented variable is used to make the dynamics more expressive (Dupont et al., 2019). Augmented Neural

Differential equations use an extra augmented dimension concatenated to the latent variables. This essentially gives the latent dynamics another dimension they could use if needed, allowing the latent variables to have trajectories they wouldn't normally be able to have if restricted to their dimensions. In this work, the augmented dimension a_t is kept constant with respect to time by setting $da_t/dt = 0$ in the drift function. Figure 6.3 illustrates the idea behind using an augmented neural differential equation with a constant augmented variable. Normally, the trajectory of a differential equation depends on the initial condition, and, the augmented dimension allows for trajectories to start with the same initial latent condition $((z_1)_0, (z_2)_0)$ but have different trajectories as time goes on due to different initial conditions when considering the entire space (z_1, z_2, a). The intuition behind the idea was that if the latent trajectories represent degradation, then when the machine is healthy the model should allow the latent variables to be similar or the same (to represent a healthy machine). However, as time goes on, different machines may degrade differently or start degrading at different times. The augmented dimension helps capture this difference between the machines and the different possible future trajectories the degradation may take.

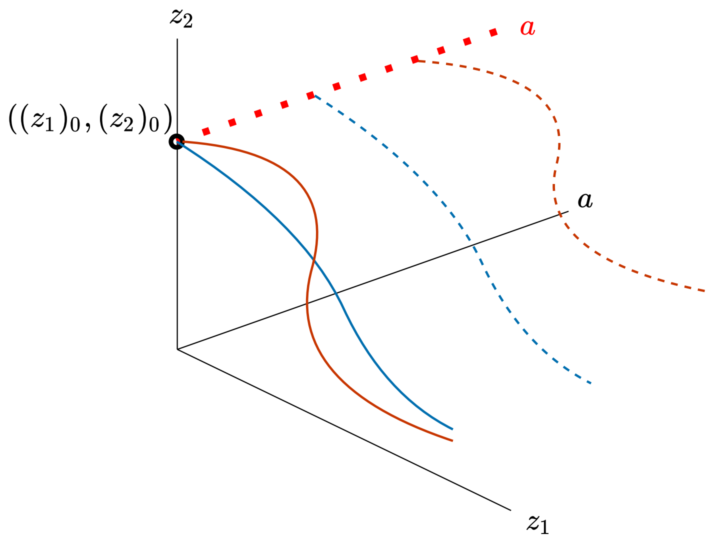


Figure 6.3. An illustration of how augmented dimensions allow for different latent variable trajectories despite having the same initial condition in the latent space (z_1, z_2). The solid line trajectories are projections of the dotted line trajectories in the 3D space (z_1, z_2, a) to the 2D latent space (z_1, z_2). Each dotted line trajectory exists on a constant line $a =$ constant, which is meant to represent different unit trajectories.

Hyperparameters such as the time window size and the value of β in the ELBO loss shown in Eqn 6.3 were found using Bayesian optimisation. Bayesian optimisation was implemented using the PyTorch package Ax, which utilises the BoTorch library (Balandat et al., 2019) to implement Bayesian optimisation for hyperparameter tuning. The resulting hyperparameters can be found in Appendix C. Specific architecture details (such as sizes of the network hidden layers) are given in Appendix D.

6.3 Results

In this section, the performance of the NSDE-DVAE is evaluated on each of the four CMAPSS datasets and compared to other existing methods. The NSDE-DVAE is compared to various other data-driven methods, as shown in Table 6.1, which reports the RMSE for each dataset in CMAPSS. The RMSE is calculated using equation 5.16 from section 5.3.1. Another common metric used for the CMAPSS dataset is the score function (Saxena et al., 2008). The scores of the NSDE-DVAE and other works are shown in Table 6.2.

The NSDE-DVAE was compared with the same models the DVAE was compared with in Table 5.1 and the same applies to the score results. As a reminder, not all models reported score values and are therefore missing in Table 6.2

Table 6.1: RMSE performance on each of the testing CMAPSS datasets with maximum RUL value of 130.

Method	FD001	FD002	FD003	FD004
NSDE-DVAE (This work)	8.75	**10.20**	8.30	8.92
DL-NSGPR (Z. Xu et al., 2021)	**7.4**	11.80	**7.5**	**8.3**
CNN (X. Li et al., 2018)	12.61	22.36	12.64	23.31
RBM-LSTM (Listou Ellefsen et al., 2019)	12.56	22.73	12.10	22.66
TDDN (Qin et al., 2022)	9.47	10.93	9.17	11.16
DS-SANN (D. Xu et al., 2022)	11.64	13.34	12.28	14.98

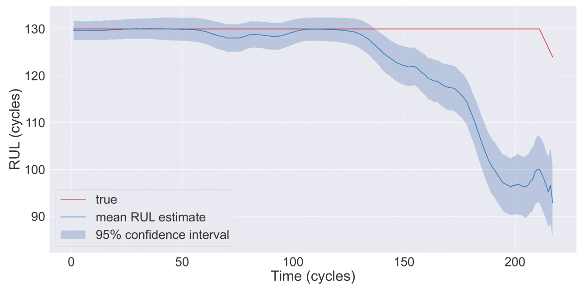
Table 6.2: comparison of the scores (Eq. (5.17)) on each of the testing CMAPSS datasets

Method	FD001	FD002	FD003	FD004
NSDE-DVAE (This work)	468	1165	420	1025
CNN (X. Li et al., 2018)	273	10412	284	12466
LSTM (Listou Ellefsen et al., 2019)	231	3366	251	2840
TDDN (Qin et al., 2022)	214	**562**	**217**	**998**
DS-SANN (D. Xu et al., 2022)	**210**	867	286	1150

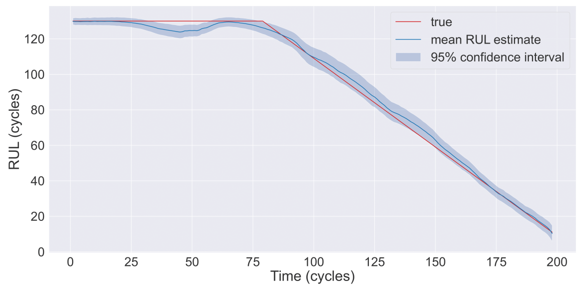
As with chapter 5, plots of the RUL over the testing unit lifetime are given. Again for comparison the results for units 12, 35, 41 and 100 are plotted in Figure 6.4. As with the results from the DVAE shown in Figure 5.4, it can be seen that the RUL estimates for units 12 and 41 deviate from the true RUL. The same reasoning for this phenomenon can be used here; however, note that the DVAE did a better job capturing the uncertainty when the estimates deviate from the true RUL trajectory, which will be discussed further in section 6.4.

Similarly, the latent state space plots for each of the testing units are plotted for each of the datasets in CMAPSS as shown in Figure 6.5. These are plotted as scatter plots and the colour of the points indicates the time point as indicated on the colour bar.

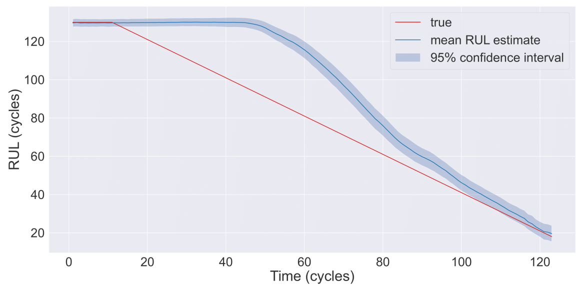
Like the DVAE chapter, the plot of the final RUL estimates versus the True final RULs for each testing unit are plotted in Figure 6.6. It can be seen the closer the testing unit is to failure, the less scattered the points are, indicating that the model becomes more accurate the closer the machine gets to failure. This is to be expected and is a good sanity check for the model's performance.



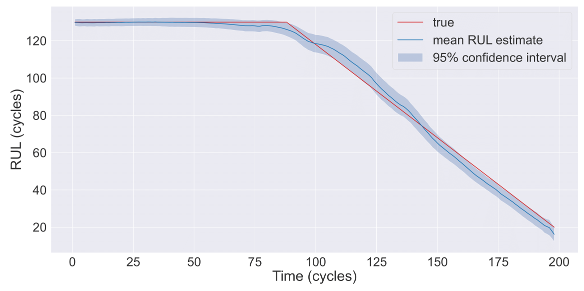
(a) RUL estimates vs lifetime of unit 12



(b) RUL estimates vs lifetime of unit 35

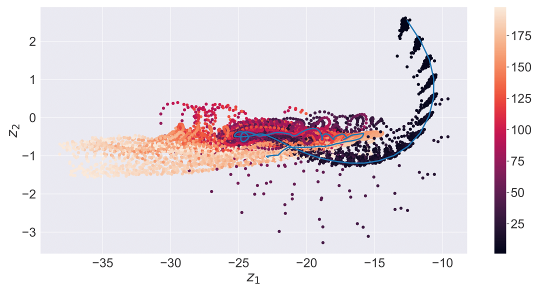


(c) RUL estimates vs lifetime of unit 41

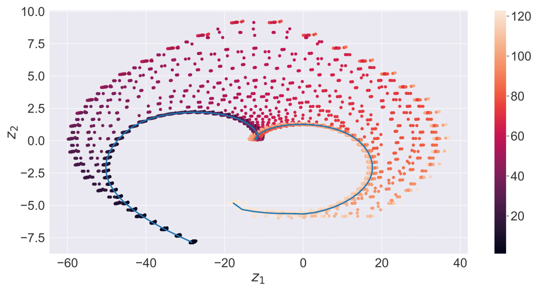


(d) RUL estimates vs lifetime of unit 100

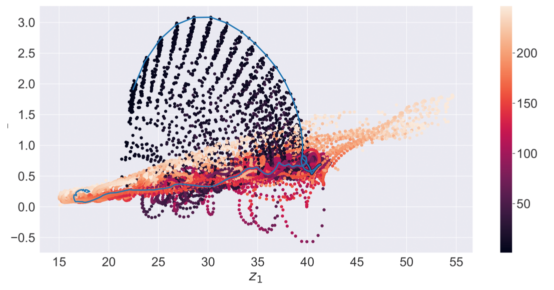
Figure 6.4. The RUL estimates over the known lifetime of the testing units for a selected subset of units in the FD001 test dataset.



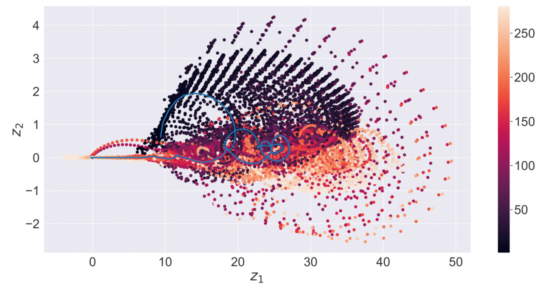
(a) Latent state space trajectories for FD001



(b) Latent state space trajectories for FD002



(c) Latent state space trajectories for FD003



(d) Latent state space trajectories for FD004

Figure 6.5. A plot of the mean latent state space trajectories for the test units in each dataset. The colour bar represents the time i.e. the darker colours are earlier times and the lighter points represent later time points. The longest trajectory is plotted as a blue line to show how a unit may develop from a healthy to near-failure condition.

The plot of the final RUL estimates, along with an error bar showing the 95% confidence interval (assuming the output is a Gaussian), is shown here for each of the units in FD001 dataset (Figure 6.7.

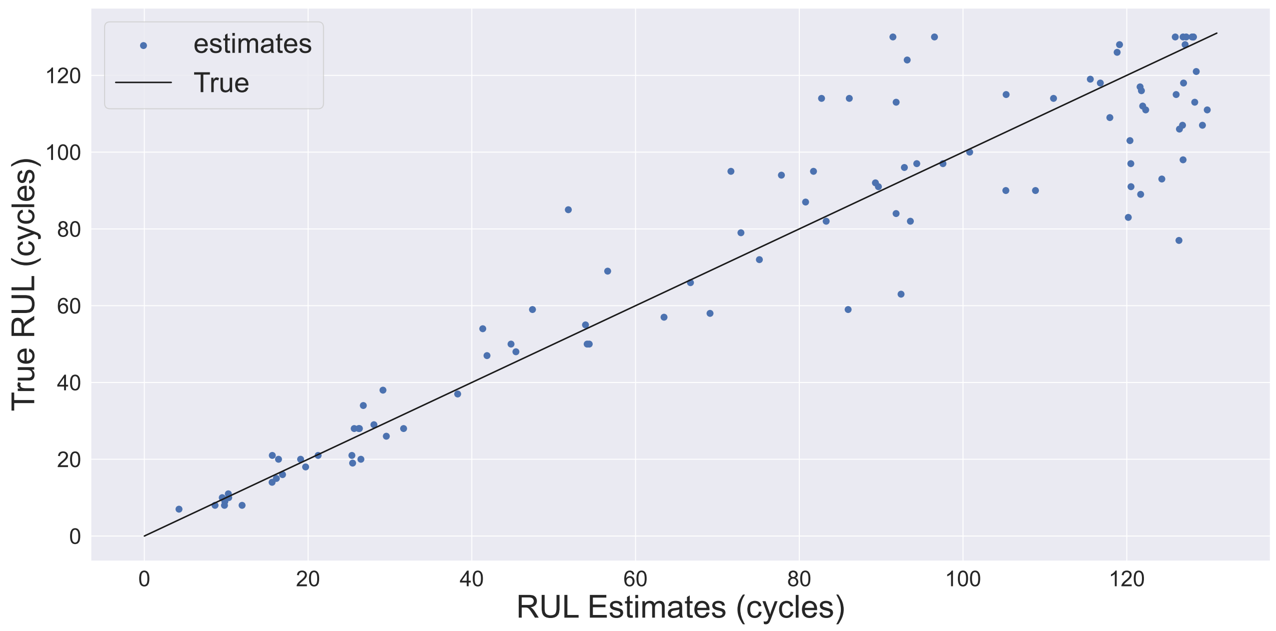


Figure 6.6. A plot of the RUL vs the True RUL for each of the testing units final time point. A line with gradient 1 is also plotted to represent the points where RUL = Actual RUL. Hence, the more accurate the points are, the closer they match the line.

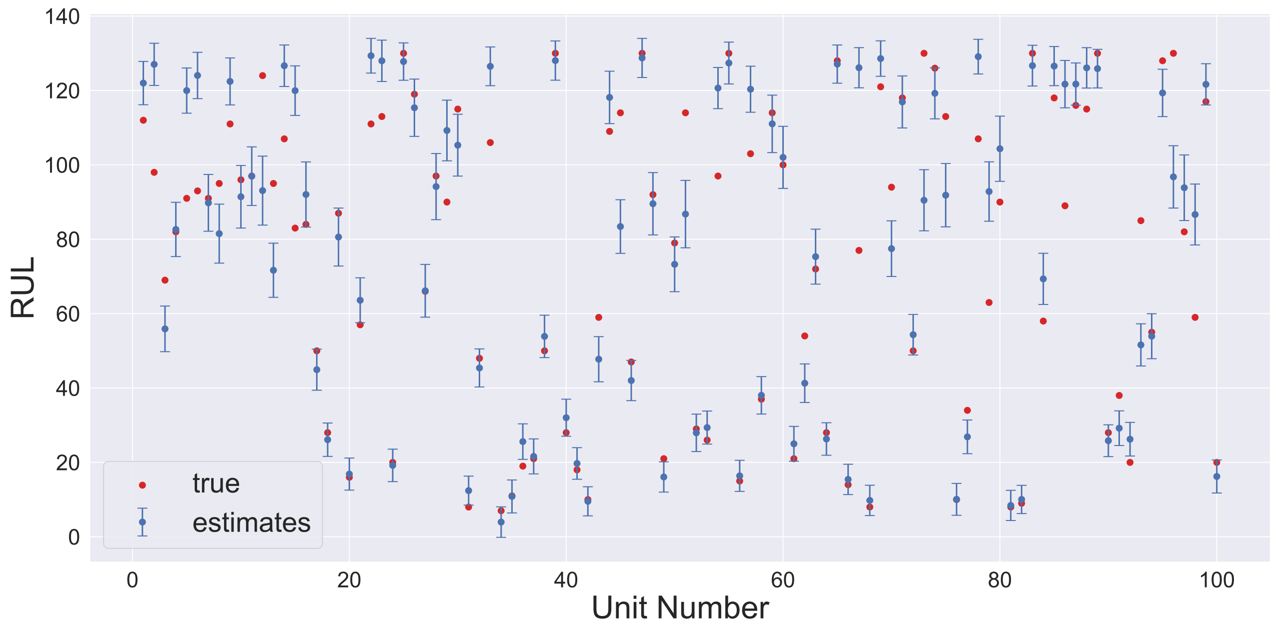


Figure 6.7. The final RUL mean estimates and True RUL values plotted as a scatter plot vs the unit index for each of the units in the FD001 test dataset. The 95% confidence intervals are also shown for the final RUL estimate.

It can be seen from Tables 6.1 and 6.2 that the NSDE-DVAE can be comparable to the state-of-the-art methods and performs exceptionally well on the CMAPSS dataset. Given the theory and the effectiveness of DVAEs on sequential data, it is not too surprising that it performs well when analysing a sequential stream of sensor data. However, another recent method, Deep Learning Non-Stationary Gaussian Process Regression (DL-NSGPR) (Z. Xu et al., 2021) achieves state-of-the-art performance on most of the datasets (except FD002). It has the advantage of using a relatively simple feed-forward deep neural network, it then fits a non-stationary Gaussian Process on top of the network outputs (estimated RUL). A benefit of the DVAE methodology is the theory behind how the model is structured, which makes it relatively simple to expand or change the model while retaining the structure described in Eqns 3.7 and 3.8, i.e. it functions as a framework instead of a single specific architecture giving it flexibility in its potential uses. For example, the DVAEs found in the literature generally use RNNs for the dynamics model. However, other models (such as NSDEs) can be used while still retaining the general structure of the DVAE (or changing the structure slightly using simplifying assumptions).

6.4 Discussion

The NSDE-DVAE performs well on the CMAPSS dataset, as might be expected due to its similarity to the DVAE presented in Chapter 5. However, DVAE outperforms NSDE-DVAE on all the datasets. Since the biggest difference between these models is the latent transition model used, this would probably be the reason for the change in results. The difference between the NSDE transition model and the MLP model used in DVAE is that the NSDE largely relies on its initial condition z_0 and in this implementation, its form was restricted. Here we tried to create a more interpretable model by constructing a 2^{nd} order drift function shown in Eqns. 6.4 and 6.5. However, using these different latent dynamics caused a drop in performance compared to the DVAE used in Chapter 5. Table 6.3 shows the RMSE results for the DVAE and NSDE-DVAE for ease of comparison. Note that both perform similarly except for the FD003 and

Table 6.3: Comparing RMSE performance between DVAE and NSDE-DVAE on each of the testing CMAPSS datasets

Method	FD001	FD002	FD003	FD004
DVAE	**8.63**	10.29	**7.28**	9.61
NSDE-DVAE	8.75	**10.20**	8.30	**8.92**

FD004 datasets. This drop in performance in the FD003 dataset meant the NSDE-DVAE model no longer had state-of-the-art results for that dataset. However, it still outperformed most of the models regarding RMSE performance and remained state-of-the-art for FD002 (not counting the previous DVAE method from Chapter 5). The DVAE also seems to have captured uncertainties better than NSDE-DVAE. From Figure 6.7, it can be seen that multiple units are outside of the 95% confidence interval. Because of the model's accuracy, plenty of the true RUL values are within the bounds; however, on the more inaccurate estimates, NSDE-DVAE does not seem to capture the uncertainties well. Consider Figure 6.7 where most units have the final true RUL captured by the model's confidence bounds. A specific example of this can be seen in Figures 5.4a and 6.4a, where both mean RUL estimate trajectories for unit 12 diverge from the true RUL as the model thinks the machine has started to degrade (RUL estimates deviate from the healthy RUL value that is set to 130). However, the DVAEs uncertainty manages to capture this increase in uncertainty and the bounds show that the machine could still be in the healthy operating region while NSDE-DVAEs uncertainty bounds do not. Since the only major difference between the two DVAEs are the latent dynamic models, this suggests that the type of model used can significantly affect how the uncertainties are quantified. The scores between DVAE and NSDE-DVAE are stated in Table 6.4 for ease of comparison, Note that NSDE-DVAE manages to obtain better scores than DVAE. Hence, it was able to obtain

Table 6.4: Comparing scores between DVAE and NSDE-DVAE on each of the testing CMAPSS datasets

Method	FD001	FD002	FD003	FD004
DVAE	**376**	1791	533	1403
NSDE-DVAE	468	**1165**	**420**	**1025**

more conservative final RUL estimates for the majority of the datasets. However, the DVAE had better uncertainty quantification which may be more beneficial for a practitioner when implementing a machinery prognostics method. Hence, the DVAE may be the better model overall in terms of performance. A useful aspect of the NSDE-DVAE is that it has continuous dynamics and can account for different time steps. Hence, one could apply the NSDE-DVAE to time series data whose data is not evenly spaced. Although the RNNs used as encoders would also need to be replaced with an architecture that could account for different time steps between the data points (e.g. attention mechanisms could be used in this way). This means for specific scenarios where the data is not evenly spaced in time, it may be worth using the NSDE-DVAE instead of the standard DVAE presented in Chapter 5.

A downside of the model is that it overfits the time window it was trained on. If the time window increased or decreased when testing the model, i.e. $T = 39$ or $T = 41$, the performance was slightly worse (depending on how much the window changed). Hence, the sequence-to-sequence model was the best choice when considering the time windows impact on the performance and the noncausal requirements of the DVAE models.

- It is slower than an RNN method. However, this could be due to the implementation of neural differential equations that are not as optimised as some of the larger deep learning packages available.
- Sensitive to hyperparameter selection. It was found when testing different hyperparameters that the model's performance could change drastically with the change of hyperparameters.
- The noncausal model requires a whole time window of input data before RUL can be estimated. The model overfits this time window. The NSDE-DVAE also has to be operated in a sequence-to-sequence fashion which may not be suitable for traditional online settings where the RUL needs to be updated for each new data point as it is sampled in real-time.

However, these initial results in terms of performance seem promising, and future work will aim to fix or reduce the impact of these problems. Again it is worth mentioning that the (DL-NSGPR) (Z. Xu et al., 2021), which also achieved similar performance, has the benefit of employing only a simple feedforward network to output the RUL and fits a Gaussian Process (GP) to the outputs of the trained network. This allowed for uncertainty quantification of the RUL through the GP, however, note that DVAEs can represent arbitrary distributions through Monte Carlo sampling. By repeatedly sampling latent sequences and ultimately sampling multiple RUL trajectories using the generative model, one can represent the RUL distribution through those samples. However, as previously mentioned this particular NSDE-DVAE model was not able to capture uncertainties like the standard DVAE was able to. Perhaps by altering the form of the latent dynamics, this could be improved.

Another interesting component of the DVAE is the latent dynamics it uses to generate the data (RUL in this case). Because of the connection between the RUL and the latent dynamics, in a sense, the latent dynamics capture the degradation of the machine as the RUL, and latent variables are connected through the generative model $p(r_{1:T}, z_{1:T}|x_{1:T})$. From the latent dynamics seen in 6.5, it can be seen the NSDE latent dynamics were capable of learning dynamics that could start at a “healthy point” and head towards a failure point as they degrade. However, the dynamics look complex as an auxiliary latent dimension was used hence, the latent dynamics are 3-dimensional; however, the third dimension was kept constant, and the plots are projected into 2-dimensional space (as illustrated by 6.3). It was hypothesised that this would allow the NSDE to have a common starting point for the latent trajectories and allow for more flexible dynamics where trajectories could cross in the 2D space. This seems to have worked, as is best seen in the FD002 plot, which has a common starting cluster but different trajectories diverge over time. However, the form of the latent dynamics is important as well. The initial reason why neural differential equations were used here was to attempt to learn an underlying differential equation that could describe the latent dynamics of the system’s degradation. If a valid differential equation describing the degradation of the machine could be found, other potential avenues for research could be explored such as analysing these dynamics to learn more about the machinery’s degradation. For example, the stability of the dynamic system could be analysed, or perhaps a control law could be derived that reduces the chance of the machine quickly degrading. Many physics-based or statistical methods in machinery prognostics either use an ODE or SDE to model the degradation of a machine. Hence, the NSDE-DVAE could be used to connect these methodologies with deep learning methods by using those differential equations in the NSDE-DVAE latent dynamic models and testing its performance. Another possible benefit of using NSDEs or NODEs relies on the fact that multiple works mention how to regularise the dynamics and cite the large literature available for understanding and analysing the properties of ODEs and SDEs. For example, in this chapter, augmented dimensions were used based on the works of, Dupont et al., 2019, who use the augmented dimensions to get around the fact ODE trajectories can’t cross. Other works from, Finlay et al., 2020, use properties of the ODE to stabilise the dynamics and they do this by adding a simple regularising term in the loss function. Hence, deep learning techniques often suffer from their black-box nature and can be difficult to work with as a result; NSDE-DVAEs attempt to clarify the inner workings of the network while also providing a method that quantifies the uncertainty of the RUL estimates.

6.5 Conclusions

This chapter explored the DVAE methodology further by exploring a different model describing its latent dynamics. NSDEs allowed the latent dynamics to be described using a continuous SDE formulation instead of the discrete MLP formulation used for the DVAE in the previous chapter. This has the benefit of using a well-known model in the dynamics setting, namely differential equations. The chapter did show that simply applying the NSDE as a latent model does not improve the performance of the DVAE. While overall performance was not poor and still did well against many state-of-the-art models; the extra complication required to model the NSDE latent dynamics did not significantly improve the results when compared with the standard DVAE. However, there are still some potential benefits of this model.

The NSDE-DVAE (or one could even construct a NODE-DVAE) has latent dynamics described by differential equations. Differential equations are well-studied mathematical objects, and future avenues for improving this model's latent dynamics may be clearer than the previous DVAE. As mentioned in the previous section, various works use the properties of differential equations to find ways to stabilise or improve the trajectories from these neural differential equations. While in this particular case, the latent dynamics don't seem to be an improvement over the DVAE in terms of how they impact performance, the NSDE formulation of the dynamics is flexible and can be altered and experimented with further as a result.

An interesting extension to this work would be to look at existing SDEs or ODEs that describe physics or statistics-based degradation models and apply those as the latent dynamic model in the DVAE like NSDEs were here. Some of the unknown parameters of the SDEs or ODEs could be learnt through the deep learning framework, but this would give principled latent dynamics that could more closely match a physical HI. If the model is an ODE that is not normally probabilistic, the DVAE formulation would turn it into one and allow for various MC simulations to be done and predict various possible outcomes for the machine's degradation. Hence, it may be useful to use the DVAE framework along with physics-informed networks such as the ones found in (Nascimento & Viana, 2019; Yucesan & Viana, 2019; Dourado & Viana, 2019).

Chapter 7

Kalman Dynamical Variational Autoencoders

When training deep generative models such as the VAE, sampling from the marginal distribution (see Eqn. 3.2) is often the objective. The generative models presented thus far use the ELBO as an objective function to indirectly maximise the likelihood that sampled input values from the generative model come from this marginal distribution. When adding conditional variables the marginal likelihood becomes a conditional distribution as shown in the DVAE chapter (from Eqn. 5.1 onwards derives a conditional DVAE). Hence, when constructing the DVAE model, the RULs were chosen to be the generative model inputs, or observed variables, and the DVAE was conditioned on the sensor signals. By choosing the RULs as inputs and sensors as conditional variables it resulted in the model optimising the DVAE to sample from the marginal likelihood, $p(r_{1:T}|x_{1:T})$, (the likelihood of sampling the correct RULs given a sequence of sensor values). Hence, the trained model is a probabilistic model that estimates the RUL given the sensors and this could be trained using the theory from VAEs/DVAEs as was done in Chapters 5 and 6. The models so far have used neural networks to describe the transition dynamics for the latent variables, p_{θ_z}, as well as the likelihood, p_{θ_r} and an inference model q_ϕ was used to help optimise these models during network training. However, we don't know if the latent space requires nonlinear functions to describe the transition between latent states or the relationship between latent states $z_{1:T}$ and the RUL $x_{1:T}$, i.e. it is not known if p_{θ_z} and p_{θ_r} need to be nonlinear. We do know that the relationship between sensors and the RUL is unknown and probably nonlinear, but the rest could potentially be linear, this linearity between the latent states and RUL values are tested in this chapter.

One way of expressing transition and measurement models is by using a Linear Gaussian State Space Model (LGSSM),

$$\mathbf{z}_t = F_t\mathbf{z}_{t-1} + \mathbf{u}_t + \mathbf{w}_t \tag{7.1}$$

$$\mathbf{x}_t = H_t\mathbf{z}_t + \mathbf{d}_t + \mathbf{v}_t \tag{7.2}$$

where, $\mathbf{u}_t$ and d_t are control variables, $\mathbf{w}_t$ and $\mathbf{v}_t$ are noise samples generated from $\mathcal{N}(\mathbf{0}, Q_t)$ and $\mathcal{N}(\mathbf{0}, R_t)$ respectively and the other variables follow the Kalman filter notation from section 3.3.1. If we use deep learning methods to optimise the parameters of this generative model we would not need to use an inference model and the ELBO loss function. Recall from section 3.3.1, specifically Eqns. 3.31 and 3.32, to find the marginal likelihood we can use the Kalman filter algorithm. Note, in that section, the control variables were not considered and hence the marginal likelihood is stated as $p(\mathbf{x}_{1:T})$ but here we have control variables so it would be $p(\mathbf{x}_{1:T}|\mathbf{u}_{1:T})$ which becomes $p(r_{1:T}|\mathbf{u}_{1:T})$ in the prognostics case.

Hence, this chapter explores the simplification of the DVAE model in the prognostics case by modelling the generative model as a LGSSM and using the Kalman filter to train the model. The latent variables do not necessarily need to have nonlinear relationships between the RUL and themselves and so this chapter tests the validity of that assumption. The Kalman filter also allows for a direct calculation of the marginal likelihood instead of indirectly optimising it using the ELBO loss function as is normally done in VAE and DVAE applications. This also means an inference model, q_ϕ is not needed to optimise the model during training, as Kalman filtering only requires a transition and measurement model in the form of an LGSSM.

7.1 Methodology

From the background section, we know that a generative model is a model whose goal is to sample from the data distribution $p(X)$. In the sequential case, one can simply substitute $X = \mathbf{x}_{1:T}$ and the distribution becomes $p(\mathbf{x}_{1:T})$. Generative models generally end up using the negative log-likelihood as a criterion for optimisation, $-\log p(\mathbf{x}_{1:T})$. The question remains, how does one create a RUL estimation model using these types of generative models? A relatively simple way is to use the Kalman filter and state space model framework. As mentioned in section 3.3.1, the Kalman filter also uses the same marginal negative log-likelihood loss function to optimise any unknown parameters in the state space model, specifically, this is shown in Eqns 3.31 and 3.32. Hence, the Kalman filter could be used to optimise the state space model used for the latent dynamics ($\mathbf{z}_{1:T}$) and the inputs ($\mathbf{x}_{1:T}$) instead of using the ELBO to indirectly optimise the neural networks. Another benefit of viewing the DVAE like this is that we view it from a state space modelling and filtering framework which are both well-known concepts with a rich literature.

To create a DVAE that estimates the RUL it is useful to start from the loss function we desire and then work up to a model. Hence, the loss function should maximise the probability of estimating the RUL sequence ($r_{1:T}$) given the sensor values $\mathbf{x}_{1:T}$,

$$-\log p(r_{1:T}|\mathbf{x}_{1:T}) = -\log \int p_\theta(r_{1:T}, \mathbf{z}_{1:T}|\mathbf{x}_{1:T}) d\mathbf{z}_{1:T}, \tag{7.3}$$

where $p(r_{1:T}, \mathbf{z}_{1:T}|\mathbf{x}_{1:T})$ would be a conditional generative model. For this conditional generative model, the RUL sequence ($r_{1:T}$) are the input variables that are related to the latent sequence $\mathbf{z}_{1:T}$, however unlike the DVAE in the background section, this one is conditioned on the sensor sequence $\mathbf{x}_{1:T}$. The generative model can be broken down into a prior and decoder network or using the state space model terminology, the state dynamics and the observation likelihood,

$$p_\theta(r_{1:T}, \mathbf{z}_{1:T}|\mathbf{x}_{1:T}) = \prod_{t=1}^{T} p_{\theta_r}(r_t|r_{1:t-1}, \mathbf{z}_{1:t}, \mathbf{x}_{1:T}) p_{\theta_z}(\mathbf{z}_t|\mathbf{z}_{1:t-1}, r_{1:t-1}, \mathbf{x}_{1:T}). \tag{7.4}$$

Where the decoder/observation likelihood and the prior/state dynamics are,

- Prior/State Dynamics: $p_{\theta_z}(\mathbf{z}_t|\mathbf{z}_{1:t-1}, r_{1:t-1}, \mathbf{x}_{1:T})$.

- Decoder/Observation likelihood: $p_{\theta_r}(r_t|r_{1:t-1}, \mathbf{z}_{1:t}, \mathbf{x}_{1:T})$.

Notice that the conditional variable, $x_{1:T}$, cannot be broken down further and so the model is a noncausal model as future values impact the current one. In the Kalman filter framework, these distributions are modelled as Gaussian distributions and the dynamics and measurement transitions are linear. The framework also makes some assumptions such as the Markov assumption for the state dynamics and the observations are independent of each other (Särkkä, 2013). Hence, the models can be updated by applying these assumptions,

- Prior/State Dynamics: $p_{\theta_z}(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t)$.
- Decoder/Observation likelihood: $p_{\theta_r}(r_t|\mathbf{z}_t, d_t)$.

Where here we have included control variables $\mathbf{u}_t$ and d_t in the state space models and so the transition and measurement models are dependent on them as well. However the DVAEs used so far do not have "control" variables, however, one can see by the above formulation of p_{θ_z} and p_{θ_r} that they can also act as conditional variables, hence, one could make both $\mathbf{u}_t$ and d_t represent $x_{1:T}$. Hence, it is proposed that a neural network encodes $\mathbf{x}_{1:T}$ to represent the control variable $\mathbf{u}_t$ and d_t. This would result in the same DVAE structure used in Chapter 5 but with linear Gaussian assumptions for the transition and measurement models. This linearity is valid as the RUL trajectory is a linear degrading trajectory and the latent space is arbitrarily created by the generative model to sample from that space and relate it to the input space. Here it is hypothesised that the only potentially nonlinear relationship in the machinery prognostics problem is the relationship between the sensors and the RUL or the sensors and the degradation HI (such as z_t). Hence, this is dealt with by using a neural network to bring $\mathbf{x}_{1:T}$ into the linear state space in the form of control variables $\mathbf{u}_t$ and d_t that can then be used in the LGSSM model.

For this paper, the formulation of the state space model used for Kalman filtering will be in the following form (Särkkä & Garcia-Fernandez, 2021),

$$\mathbf{z}_t = F_t \mathbf{z}_{t-1} + \mathbf{u}_t + \mathbf{w}_t, \tag{7.5a}$$
$$r_t = H_t \mathbf{z}_t + d_t + v_t. \tag{7.5b}$$

This differs from Eqn. 3.22 as it adds another known control variable d_t in the observation model (Eqn. 7.5b) and replaces $B_t\mathbf{u}_t$ with $\mathbf{u}_t$. Note that replacing $B_t\mathbf{u}_t$ means that the neural network that outputs $\mathbf{u}_t$ will already account for any transform a matrix such as B_t would apply to $\mathbf{u}_t$ so there is no need to add B_t for this application. To be clear one can use the formulation, $B_t\mathbf{u}_t$, if the application requires B_t to be known. To encode the entire sensor signal $\mathbf{x}_{1:T}$ into $\mathbf{u}_t$ and d_t a bidirectional RNN can be used (a diagram is shown in Figure 7.1); specifically a bidirectional Gated Recurrent Unit (GRU) network is used in the experiments. The matrices describing the dynamics such as F_t, H_t, Q_t and R_t, can be optimised during the neural network training.

A benefit of this formulation is the indirect optimisation of the model using the ELBO as shown in Eqn. 3.9 is not required. Instead, the model can be optimised by directly finding the observation's negative log-likelihood, $-\log p(r_{1:T}|\mathbf{x}_{1:T})$, as discussed in Section 3.3.1. When applying the model after training, one no longer has known input RUL values (r_t) available and

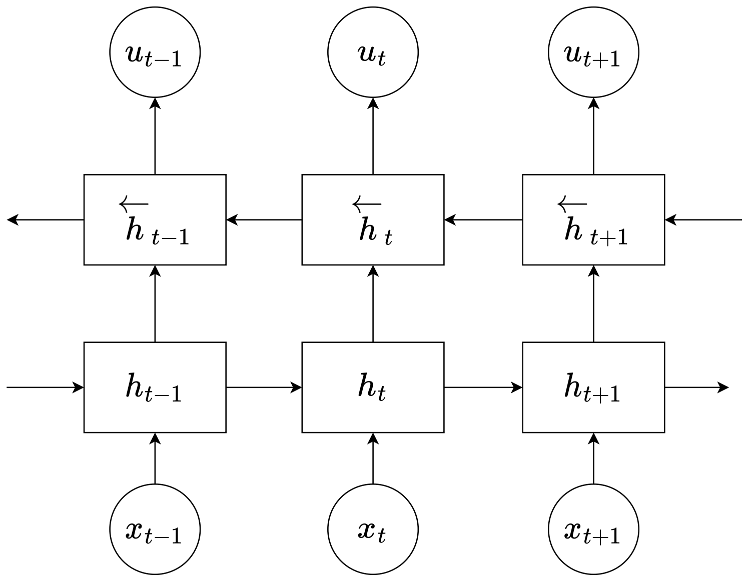


Figure 7.1. Block diagram of the bidirectional RNN used for calculating the control variables $u_{1:T}$ given a sensor sequence $x_{1:T}$

so the Kalman filter cannot be applied. If the RUL values are not known, the learnt generative model is used to propagate the state values (z) through time and find likely observations which have been defined as the RUL (r) in this generative model. In this case, the generative model has a convenient state-space form allowing one to forecast the state and RUL values using Eqn. 3.23. Since the RUL values are unknown the measurement update step, shown in Eqn. 3.24, is simply ignored and Eqn. 3.23 is applied repeatedly to forecast the variables throughout time. Due to the combination of the DVAE generative modelling framework and the Kalman filter used for training, this model will be referred to as the Kalman Dynamical Variational Autoencoder (K-DVAE) throughout the text[1].

7.2 Experiments

7.2.1 Data

The data used to test the DVAE method which will be described in this section is the CMAPSS dataset previously mentioned in section 4.1.

7.2.2 Preparing the Data

Operating-condition-based normalisation is used on the sensor signals and the maximum RUL was set to 130 as was done in the previous experiments. As with the DVAE and NSDE-DVAE, the data is prepared as it was in section 6.2.2 by splitting the training trajectories into time windows so it is easy to do the calculations in parallel during training. The time window size

[1]The code for the K-DVAE and the experiments in this section can be found at https://github.com/StarMarco/Kalman_DVAE or contact the author

was set to 40 cycles ($T = 40$). The training dataset was further split into a training and validation dataset. This was done by splitting the original training dataset, where 80% of the dataset became the new training dataset and the remaining 20% of the data became the validation dataset (an 80/20 split).

7.2.3 Network Design

There are a number of decisions that can be made when designing the K-DVAE model to tackle a machinery prognostics problem. Firstly, the sequence of sensor values $x_{1:T}$ needs to be represented or encoded into a value $\mathbf{u}_t$. As mentioned previously this will be done via. a bidirectional RNN; in this specific case a bidirectional GRU is used. This stays consistent with the way DVAEs handle these non-causal types of models where future values can impact current and past values. Note that a bidirectional RNN is not strictly necessary and future works may want to look at other ways of representing $x_{1:T}$ through different types of neural network architectures.

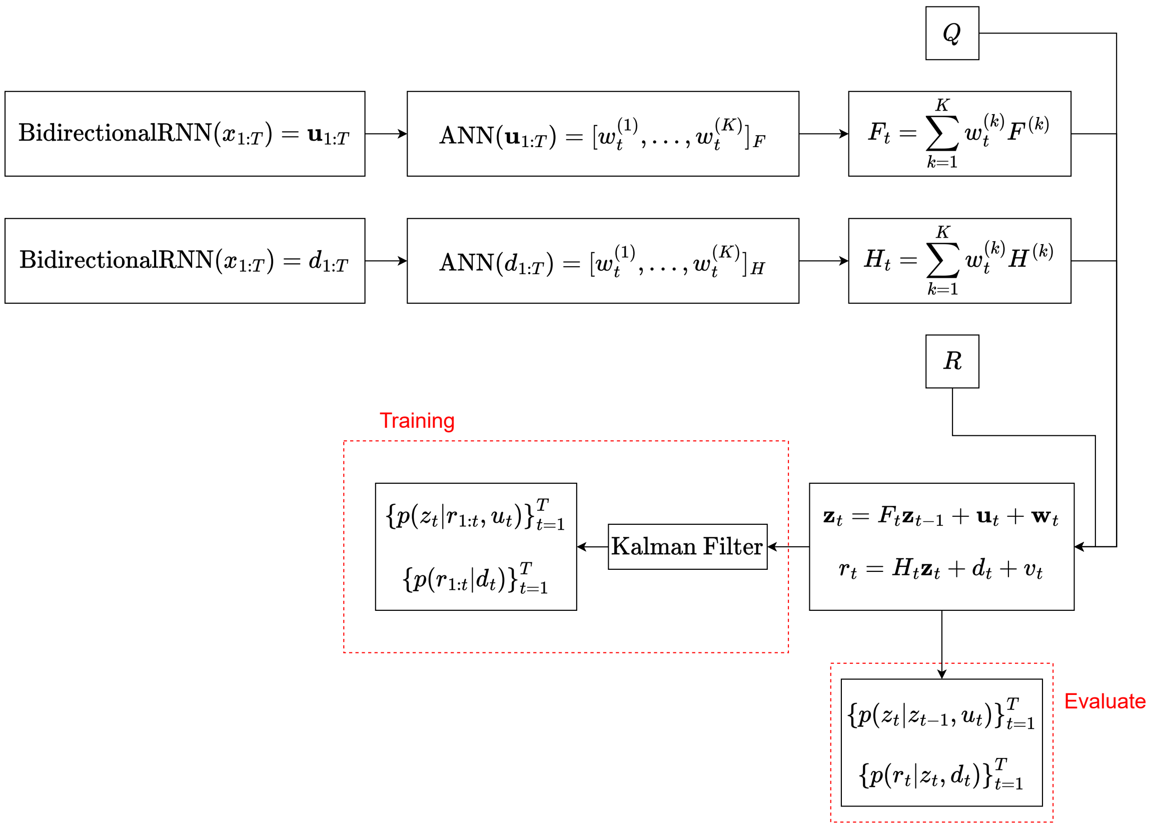
Another design decision required for the K-DVAE implementation is how the state-space model parameters will be constructed and optimised. For example, one can simply randomly initialise each of the matrices as constants F, H, Q and R, and let them be optimised during the neural networks training phase through backpropagation. However, in this problem, the RUL has been split into a healthy and degrading phase; in the healthy phase the RUL remains at a maximum value of 130 and after that it degrades until it reaches zero. Hence, one linear model would not capture this behaviour well, so weighted or interpolated matrices are found Fraccaro et al., 2017. This works by having K constant global matrices and then having a neural network output weights which sum to one to create a weighted average of the K global matrices. For example for the state transition matrix,

$$F_t = \sum_{k=1}^{K} \alpha^{(k)}(\mathbf{u}_t) F^{(k)}, \tag{7.6}$$

where $\alpha(\mathbf{u}_t)$ is a neural network that outputs a vector of normalised weights $[\alpha_1, ..., \alpha_K]$ and $\alpha^{(k)}(\mathbf{u}_t)$ is shorthand for selecting the k^{th} element of that vector. The weight neural network is implemented using a standard feedforward network and using $\mathbf{u}_t := \mathbf{x}_{1:T}$ as an input. The weights must be positive and hence we assume the network outputs the log weights; an exponential function can then be used to turn them into positive weight values. These values are then normalised and that is considered the output of $\alpha(\mathbf{u}_t)$. Hence, the function $\alpha(\cdot)$ applies,

$$\alpha(\mathbf{u}_t) := \frac{\exp\left(f_\theta(\mathbf{u}_t)\right)}{\sum_{k=1}^{K} \exp\left(f_\theta(\mathbf{u}_t)\right)^{(k)}}, \tag{7.7}$$

where $\exp\left(f_\theta(\mathbf{u}_t)\right)^{(k)}$ is the k^{th} element in the unnormalised weight vector and f_θ is the neural network. The weighted average matrices are used for both the state transition and measurement matrices F and H. The noise matrices (Q and R) however, are kept constant throughout time but are also optimised during training. Using these weighted average transition and measurement matrices allows for the model dynamics to change over time as the machine transitions from a healthy to a degrading operating mode. A simplified diagram is shown in Figure 7.2, the diagram shows how the input time windowed sensor signals are used to train and evaluate the K-DVAE.



Q
$\text{BidirectionalRNN}(x_{1:T}) = \mathbf{u}_{1:T}$
$\text{ANN}(\mathbf{u}_{1:T}) = [w_t^{(1)}, \ldots, w_t^{(K)}]_F$
$F_t = \sum_{k=1}^{K} w_t^{(k)} F^{(k)}$
$\text{BidirectionalRNN}(x_{1:T}) = d_{1:T}$
$\text{ANN}(d_{1:T}) = [w_t^{(1)}, \ldots, w_t^{(K)}]_H$
$H_t = \sum_{k=1}^{K} w_t^{(k)} H^{(k)}$
R
Training
$\{p(z_t|r_{1:t}, u_t)\}_{t=1}^{T}$
$\{p(r_{1:t}|d_t)\}_{t=1}^{T}$
Kalman Filter
$\mathbf{z}_t = F_t\mathbf{z}_{t-1} + \mathbf{u}_t + \mathbf{w}_t$
$r_t = H_t\mathbf{z}_t + d_t + v_t$
$\{p(z_t|z_{t-1}, u_t)\}_{t=1}^{T}$
$\{p(r_t|z_t, d_t)\}_{t=1}^{T}$
Evaluate

Figure 7.2. A block diagram showing how the sensor data is used to find the necessary outputs for training and testing the K-DVAE. Both $\mathbf{u_t}$ and d_t represent the entire sensor sequence $x_{1:T}$. Q and R are both constants that can be optimised during the training phase. Also note, for simplicity, the output weights from the Artificial Neural Network (ANN) block; include the normalisation of the weights in that step.

7.2.4 Model Training

To train the model the Kalman filter algorithm is applied to find the conditional data negative log-likelihood,

$$L_{nll} = -\log\, p(r_{1:T}|x_{1:T}). \tag{7.8}$$

The details on this are mentioned in the background section 3.3.1, specifically Eqns. 3.31 and 3.32. However, some additional terms are added to this to help regularise the model and improve its forecasting ability when no measurements are given to the filter. The latent dynamics have a chance to become unstable during training causing numerical errors as the values become too large. Hence, to help keep the dynamics stable the model is regularised using the eigenvalues of the latent dynamics matrix F_t. For the dynamics to be stable, the real values of the eigenvalues for F_t should be between 0 and 1. It was found that simply adding the sum of the real parts of these eigenvalues, and multiplying by a constant (β) to scale the impact of this regularisation term, helped with stability during training. This regularisation term will be denoted L_{reg}. When testing the model, there are no RUL values available and so the Kalman filter cannot be used. Instead, the model performance relies on the accuracy of the state space model dynamics learnt during training to forecast RUL estimates. During training, the loss function derived from the Kalman filter does not incorporate long-term predictions due to the constant updates from RUL measurements. Hence, a method known as “replay overshooting” is used to add another term to the loss function that aids in training a model for long-term predictions (A. H. Li et al., 2021). This is done by applying a smoother after the filter step and finding new latent mean and covariance initial conditions, $\mathbf{z}_0^{(s)}$, $P_0^{(s)}$. These initial conditions can be used, along with the state space model, to propagate the variables forward in time and estimate the RUL distributions, $\mathcal{N}(r_t, S_t)$. This distribution and the target RUL can be used to find the likelihood term that can be added to the total loss, along with a constant scaling term (α) to determine how much this forecasting step should impact the model training. The forecasting loss term derived from replay overshooting will be denoted as L_{RO}. The total loss can be stated as,

$$L = (1 - \alpha) \cdot L_{nll} + \alpha \cdot L_{RO} + \beta \cdot L_{reg} \tag{7.9}$$

7.3 Results

The Kalman-DVAE is compared with various other methods that perform well on the CMAPSS dataset. These include the Temporal Deep Degradation Net (TDDN) (Qin et al., 2022), Restricted Boltzmann Machine (RBM) and LSTM combination (Listou Ellefsen et al., 2019), CNN applied to spectrograms of the data using Short Time Fourier Transform (STFT) (X. Li et al., 2018) and the Dual-Stream Self-Attention Neural Network (D. Xu et al., 2022). To compare these models the Root Mean Squared Error (RMSE) metric was used as shown in equation 5.16 from section 5.3.1.

Table 7.1: RMSE performance on each of the testing CMAPSS datasets with a maximum RUL value of 130.

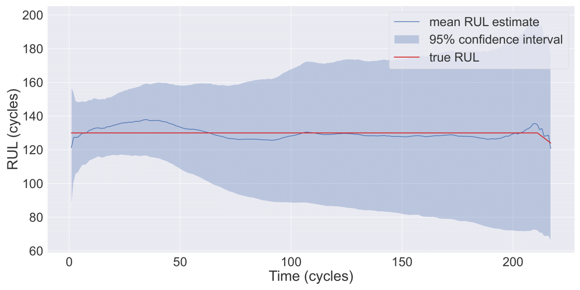
Method	FD001	FD002	FD003	FD004
Kalman-DVAE (This work)	10.67	11.39	10.13	10.18
DL-NSGPR (Z. Xu et al., 2021)	**7.4**	11.80	**7.5**	**8.3**
CNN (X. Li et al., 2018)	12.61	22.36	12.64	23.31
RBM-LSTM (Listou Ellefsen et al., 2019)	12.56	22.73	12.10	22.66
TDDN (Qin et al., 2022)	9.47	**10.93**	9.17	11.16
DS-SANN (D. Xu et al., 2022)	11.64	13.34	12.28	14.98

The scores are reported in Table 7.2. Note that some methods do not report an score in their work and therefore have no score stated in this table.

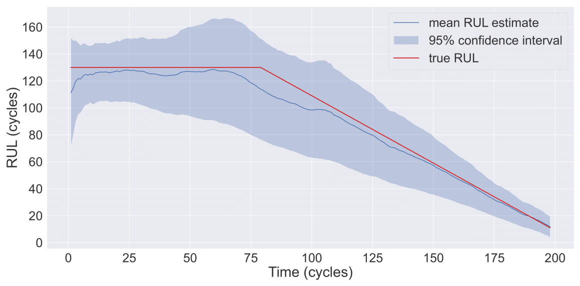
Table 7.2: Comparison of the scores (Eq. (5.17)) on each of the testing CMAPSS datasets

Method	FD001	FD002	FD003	FD004
Kalman-DVAE (This work)	388	983	470	1328
CNN (X. Li et al., 2018)	273	10412	284	12466
LSTM (Listou Ellefsen et al., 2019)	231	3366	251	2840
TDDN (Qin et al., 2022)	214	**562**	**217**	**998**
DS-SANN (D. Xu et al., 2022)	**210**	867	286	1150

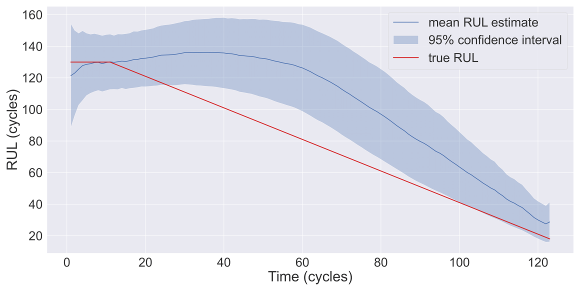
To visualise some of the K-DVAE results, a selection of units from the FD001 dataset was chosen as examples. Figure 7.3 shows the RUL estimate trajectories plotted over the known lifetime of the test units (the time when the sensor values are known).



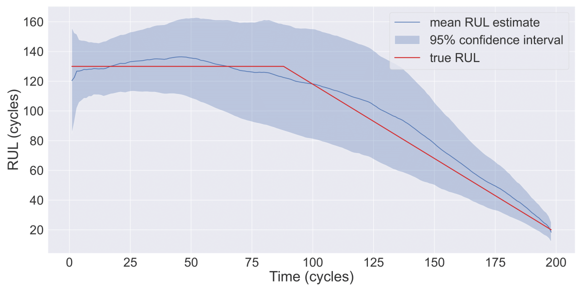
(a) RUL estimates vs lifetime of unit 12



(b) RUL estimates vs lifetime of unit 35



(c) RUL estimates vs lifetime of unit 41



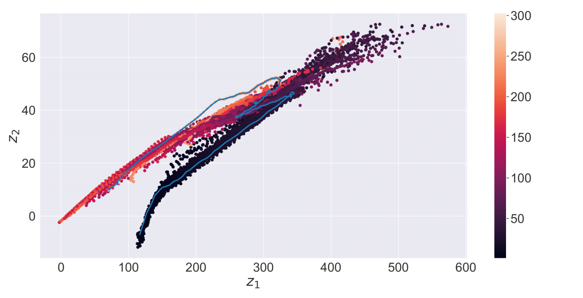
(d) RUL estimates vs lifetime of unit 100

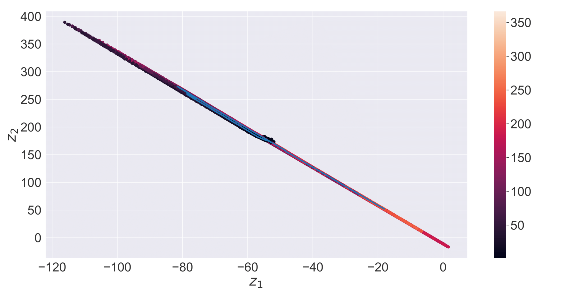
Figure 7.3. The RUL estimates over the known lifetime of the testing units for a selected subset of units in the FD001 test dataset.

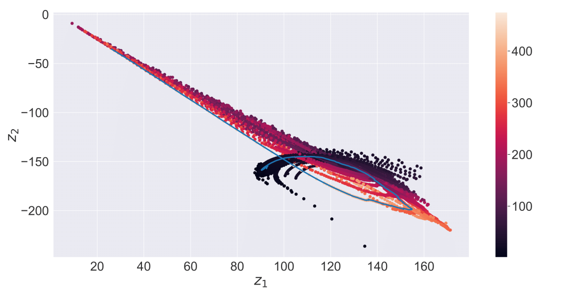
The latent variables were made two-dimensional; this means they can be plotted in state space, where the x-axis is the first latent variable and the y-axis is the second one. Figure 7.4 shows the plot of all the latent state space trajectories for every unit in each test dataset in CMAPSS (FD00X, where X = 1, 2, 3 and 4). They are plotted as scatter plots to show the general trajectory of all the units to see if all of them follow a particular trajectory that could be considered indicative of the machinery's degradation.

Figure 7.5, is a scatter plot that shows the final true RUL value along with the output RUL estimate from the K-DVAE model for each of the units in the FD001 test dataset. Ideally, they should all fall on a line with slope 1 if the estimates were 100% accurate, as the estimate RUL would equal the actual RUL. This line of slope 1 is plotted as well for convenience. Notice that the RUL estimates are more accurate as the machine reaches the failure point, which is what one may intuitively expect, as it becomes more obvious when a machine will fail the closer it is to the failure point. This can be seen from the plot as the estimates become less scattered and follow the line closer as the true RUL values become smaller.

Figure 7.6, shows the final RUL estimates and actual RUL values as a scatter plot over all the unit indexes for the test units in the FD001 dataset. It also plots the uncertainty bounds showing the 95% confidence interval of the Gaussian distribution. This plot gives an idea of how well the model performed for each unit. We can also see that the confidence bounds are generally larger for the units that have larger RUL values and so the machine is further from the failure point. For the test units that are closer to failure, the confidence bounds are smaller and the estimates are more accurate as the machine is closer to failure.


(a) *latent state space plot of all units for FD001*


(b) *latent state space plot of all units for FD002*


(c) *latent state space plot of all units for FD003*

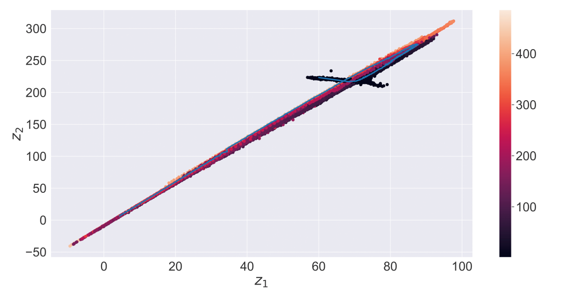

(d) *latent state space plot of all units for FD004*

Figure 7.4. The latent variable state space trajectories over the known lifetime of the testing units for all the CMAPSS datasets. The colour bar indicates the time, hence, darker colours are early in the lifetime of the machine while lighter colours are later periods of the machine's lifetime. The longest-time trajectory is plotted as a blue line, to give an indication of what a trajectory from a healthy status to a near failure status may look like.

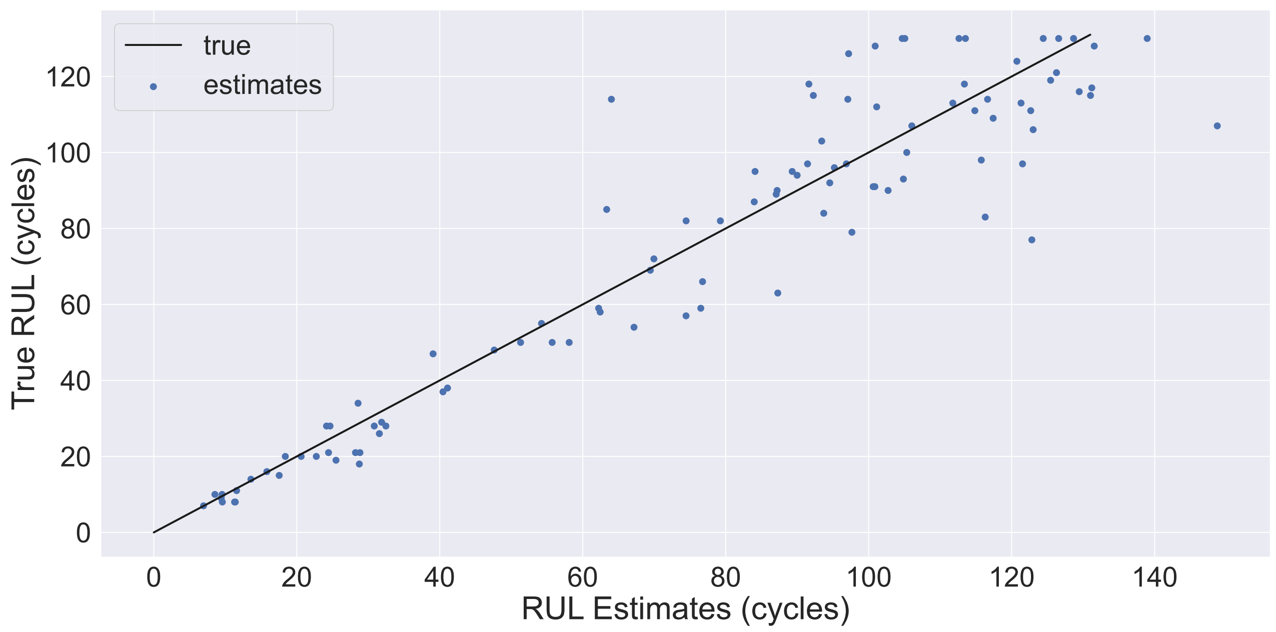


Figure 7.5. The final RUL estimates for the test units in the FD001 dataset. These were plotted as a scatter plot and compared with the true RUL values plotted as a line

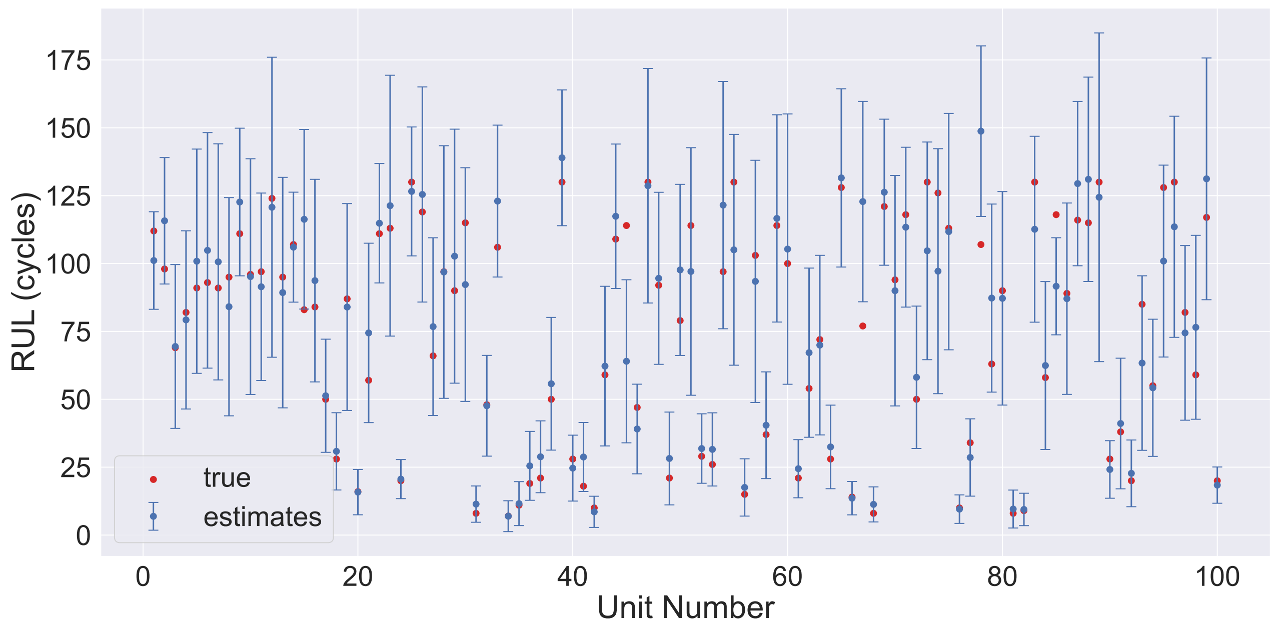


Figure 7.6. The final RUL mean estimates and True RUL values plotted as a scatter plot vs the unit index for each of the units in the FD001 test dataset. The 95% confidence intervals are also shown for the final RUL estimate.

7.4 Discussion

In terms of RMSE, the K-DVAE still manages to perform well when compared with some of the other models. While the state-of-the-art models manage to outperform it the K-DVAE does have some benefits that may be considered worth the trade-off. One benefit is that some of these models have no uncertainty quantification while the K-DVAE does. The way it quantifies uncertainty is also relatively simple compared to the other DVAE models that were introduced in this thesis. In this model, the contribution deep neural networks make is encoding the sensors using the bidirectional RNN. After encoding those variables are used as "control variables" (to use control theory terminology) in an LGSSM whose measurement space describes the RUL of the machine. Hence, the model shows that the main contribution that deep neural networks give in these DVAE methods to achieve good RUL predictions is by capturing the information from the sensor signals in a noncausal manner. But there are methods that outperform the K-DVAE and do quantify uncertainty. In terms of RMSE performance (Z. Xu et al., 2021) remains state-of-the-art for most of the results. Unlike the other DVAE methods presented the K-DVAE does not hold the advantage of being able to represent distributions other than Gaussian distributions. Because of the restrictions made by setting the transition and measurement models as LGSSMs, the outputs are restricted as Gaussian distributions. Hence, both (Z. Xu et al., 2021) and the K-DVAE models are restricted to Gaussian distribution representations of the RUL estimate distributions. However, the K-DVAE has the potential advantage of having latent linear dynamics that are related to the RUL estimates. Hence, there is an interpretable latent space that could potentially be used to simulate various possible RUL trajectories given different possible sensor values. Another possibility due to its simple LGSSM dynamics is that one could find a control law that could reduce the rate of degradation of the machine. The latent space provides a type of HI that can be used to interpret the behaviour of the machine's degradation and mathematically model it. Note that, unlike other HIs in the machinery prognostics literature, these HIs do not necessarily find a HI that could be extrapolated to a threshold or perform well in metrics such as monotonicity and trendability. However, they are HIs in the sense that they are related to the RUL by a known model (the measurement model), and unlike other HI methods that need to fit a curve to the HI outputs to model their dynamics, the DVAE has HI dynamics that are trainable during the optimisation process. This is leveraged in this work by making the dynamics conform to an LGSSM. Hence, the DVAE formulation allows for one to achieve some level of customisation in the model by setting restrictions to the transition model; although as this chapter shows that can come at the cost of RUL estimation performance.

In section 5.4, the structure of the DVAE was compared to the models presented in Table 5.1, which are the same models presented in the results here in Table 7.1. Notably, the DVAE and DL-NSGPR were said to be similar as they are both noncausal models that are optimised using the marginal likelihood as a loss function. This can be further expanded upon here as it was mentioned NSGPR uses the same likelihood as a loss function and that it can be viewed as a formulation of Kalman filtering and smoothing Sarkka et al., 2013. Here we do use a Kalman filter and smoother to train the model, and an LGSSM is used as the transition and measurement models, further increasing the similarities between the DL-NSGPR method. However, from this, the differences between the two models also become more clear. The main difference is that in the K-DVAE the sensors are encoded using a neural network and then function as "control variables" in the LGSSM. For DL-NSGPR a neural network also processes the sensor signals; however, these then act as the observations for NSGPR, not the control variables. The K-DVAE cannot use the Kalman filter after training because it uses the true RUL values as observations; hence, when evaluating the model, the LGSSM is applied over the time period of interest without any filtering. DL-NSGPR uses NSGPR even on test cases to smooth the RUL predictions of the neural network that act as observations for NSGPR. The benefit of the K-DVAE approach is that it has latent variables that could be utilised to aid in understanding the degradation behaviour. Furthermore, while not true of the K-DVAE, the more general DVAE and NSDE-DVAE are able to model non-Gaussian distributions as they use Monte Carlo sampling to represent arbitrary distributions, while NSGPR cannot. Although note, DL-NSGPR has the benefit of being relatively simple to implement as it trains an MLP network to estimate the RUL which is easier to train and computationally less expensive.

The results of the K-DVAE are compared with the other DVAE models in terms of the RMSE in Table 7.3 and score in Table 7.4. These are shown for a quick reference for convenience. Note, the other DVAEs largely outperform the K-DVAE as one might expect given the restrictions on the K-DVAE. However, the K-DVAE does manage to achieve a better score on the FD002 dataset.

Table 7.3: Comparing RMSE performance between DVAE and NSDE-DVAE on each of the testing CMAPSS datasets

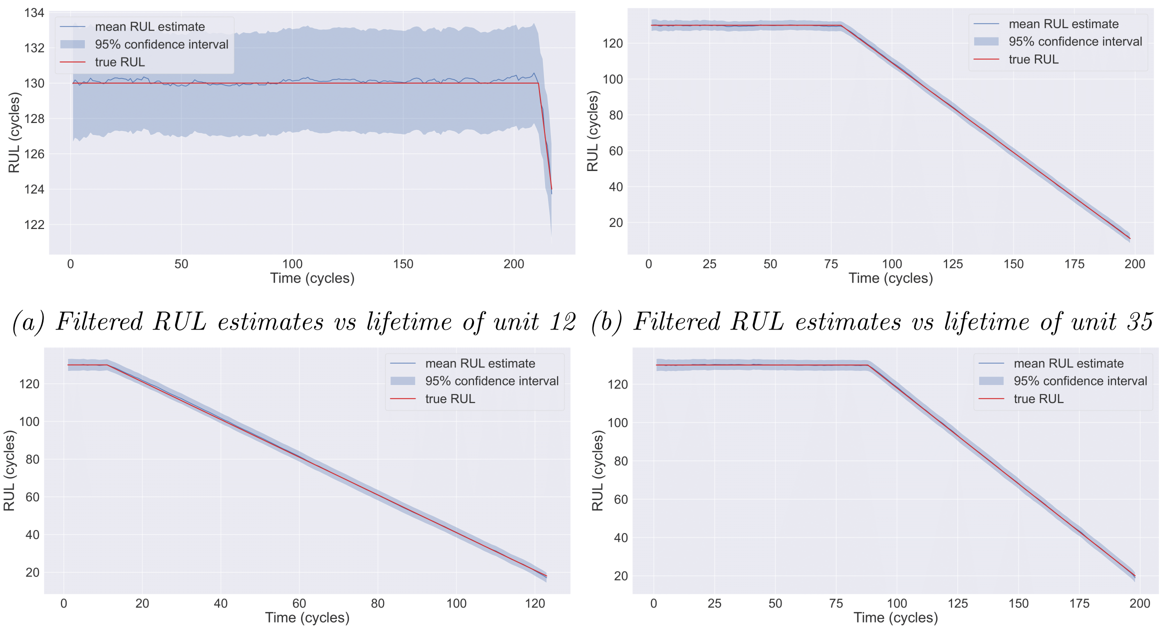
Method	FD001	FD002	FD003	FD004
DVAE	**8.63**	10.29	**7.28**	9.61
NSDE-DVAE	8.75	**10.20**	8.30	**8.92**
Kalman-DVAE	10.67	11.39	10.13	10.18

Note that the K-DVAE performance is improved if we could observe the RUL as the filtering step could be applied to update the estimate based on the observation. If the practitioner has other

Table 7.4: Comparing scores between DVAE and NSDE-DVAE on each of the testing CMAPSS datasets

Method	FD001	FD002	FD003	FD004
DVAE	**376**	1791	533	1403
NSDE-DVAE	468	1165	**420**	**1025**
Kalman-DVAE	388	**983**	470	1328

variables that could be observed, for example, by inspecting the machine, and was related to the RUL or served the same purpose as the RUL (e.g. a physical health indicator), then this model's performance could easily be improved. Since this K-DVAE framework already incorporates a filtering step during training, if we had a variable related to the RUL that the model was trained on, then it could naturally incorporate that observation in its estimates by applying the Kalman filtering step. A similar concept was explored in (Que et al., 2019), where they used a generative model to construct a HI. The HI could be observed through inspections, and its trend was estimated by fitting an exponential curve to it and the states updated with each observation using a particle filter. Given a similar set-up where a HI can be observed through inspection, the K-DVAE model's performance could be drastically improved. Figure 7.7 shows how the RUL trajectories for the selected testing units in Section 7.3 would look like if the RUL values were known and the Kalman filtering algorithm could be applied. While the actual RUL cannot be known, if there is a case, such as in Que et al., where the RUL can be estimated periodically by inspection of the machine, we can see how the RUL estimate trajectories could be updated to achieve better RUL estimates. Hence, this could be an interesting research



(a) Filtered RUL estimates vs lifetime of unit 12 *(b) Filtered RUL estimates vs lifetime of unit 35*

(c) Filtered RUL estimates vs lifetime of unit 41 *(d) Filtered RUL estimates vs lifetime of unit 100*

Figure 7.7. The filtered RUL estimates over the known lifetime of the testing units for a selected subset of units in the FD001 test dataset. This was partly done as a sanity check for the Kalman filtering algorithm, but it also shows how useful it could be if an observable metric could be used instead of the RUL. Even if it could only be observed occasionally, the filtering algorithm would be able to update the estimate and achieve much more accurate results compared to having no direct observations as shown in Figure 7.3.

direction focusing on a combination of estimating RUL from sensor signals and less frequent inspections that allow for this RUL estimate to be updated. Alternatively, if the practitioner is aware of another metric or HI that can replace the RUL in a specific case, then this model could still be used. If that HI is easily observable or could be observed through regular inspection, then the K-DVAE would be an ideal model for this scenario, as the HI could be estimated from sensor values when an inspection cannot be performed, and an updated estimate can be found after the inspection.

Further work could be done to explore the different types of encoder networks that could represent the sensor sequences $x_{1:T}$. Here the bidirectional GRU was used; however, different noncausal networks could be tested to find different ways of representing the sensor sequences. In this work, we have focused on giving a simple example of how these sequential generative models could be used to estimate the RUL. However, more complicated generative models could be used instead, and assumptions could be relaxed. For example, the Kalman filter was used here as it is relatively simple to deal with, and it was assumed due to the linearity of the RUL targets, it would be suitable for the RUL estimation task. This could be further explored, and different models could be tested, such as models whose variables are not Normally distributed and/or nonlinear models. If this is done, note that the Kalman filter training method would no longer be appropriate. However, one could still use a particle filter, although training the neural networks can be complicated due to the resampling step not being differentiable (Jonschkowski et al., 2018). Works have tried to remedy this by finding ways to create differentiable resampling (Corenflos et al., 2021; Lai et al., 2021). Note that technically, we do not need a filtering method to train the generative model. Instead, the variational objective is shown in Eqn. 3.9 could be used. However, there have been works showing it can be more effective to train using the marginal likelihood achieved from filtering as the variational objective is indirectly optimising the marginal likelihood while filtering algorithms directly return a marginal likelihood (Maddison et al., 2017; Naesseth et al., 2018; A. H. Li et al., 2021). Hence, there are many different ways generative models can be constructed and trained; the effectiveness of these different model constructions could be tested in future works, and it may give an idea of what assumptions can be made for constructing generative models for machinery prognostics tasks.

7.5 Conclusions

This work's aim was to create a probabilistic model for RUL estimation using deep sequential generative models which was accomplished using the Kalman-DVAE. The Kalman-DVAE framework takes the RUL estimation problem from machinery prognostics and formulates it as an LGSSM whose behaviour and mechanics are relatively well understood. By using this LGSSM framework the deep learning part of the framework became a parameter estimation problem which is common in these types of LGSSM problems.

The model performance is in line with the state-of-the-art models found in the literature, but it can also quantify uncertainty in its RUL estimates which is vital for machinery prognostics tasks. This work also provides a framework for constructing and training these types of generative models and highlights the different ways one may construct and train DVAEs. The background section attempts to make the theory behind DVAEs and generative models clear which will hopefully aid further research in this area as there are not many works that are using sequential generative models for machinery prognostics. If generative models are used, it is usually the VAE used in a static case as described in Section 3.1 and not adjusted for sequential variables as seen in Section 3.2.

Note that, the Kalman filter framework is not strictly necessary, however, it is useful if one wants to develop relatively easy latent dynamics. However, other training procedures can be used to allow for more complex nonlinear models, such as the ELBO loss for the DVAE shown in Eqn. 3.9. Another option is to use particle filters to train the nonlinear model, which would be analogous to training with the Kalman filter but applied to a nonlinear setting. Using particle filters to train DVAE models has been explored previously in the literature, however, one of the reasons they were avoided here was because the resampling step is not differentiable and hence cannot be optimised by backpropagation (Maddison et al., 2017; Naesseth et al., 2018). Some works try to get around this and have even created fully differentiable particle filters (Corenflos et al., 2021; Lai et al., 2021), but all this adds an extra complication to the model. This was one of the reasons why this work started by introducing this concept through a relatively simple Kalman filter training scheme. These nonlinear models and particle filter training schemes could therefore be a potential research direction and future work that builds on this K-DVAE model.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This thesis aimed to find ways in which deep learning methods for machinery prognostics could quantify the uncertainty of their RUL estimates and improve the interpretability of the results. We looked at various methods to achieve these aims, such as Neural Differential Equations and VAEs. When applying Neural DEs, the idea was to help either improve the interpretability of the neural network by implementing the structure of the network itself as a Neural DE (such as in Chapter 4) or by using Neural DEs to construct latent dynamics (such as in Chapter 6). The idea in Chapter 4 was to show that neural networks such as ResNets could be replaced with NODEs that could be easier to customise. If an ODE describes the underlying computations of the neural network, the ODEs properties could be analysed, and during training, desirable properties could be encouraged. For example, this is done in works where the performance of the network is regularised to have smoother trajectories by using the properties of ODEs (Finlay et al., 2020). By replacing the NODE with an NSDE, uncertainty could be accounted for in the network itself, similar to how Bayesian Neural Networks can be made by adding dropout mechanisms to a network. However, ultimately DVAEs ended up performing quite well, and this caused a shift in the thesis towards the variational framework for machinery prognostics applications. Part of the exploration of DVAEs for machinery prognostics reintroduced Neural DEs into the picture. This time Neural DEs were utilised to construct latent dynamics instead of defining the neural network structure itself. The DVAE sections of the thesis explored how DVAEs could be used to estimate the RUL. DVAEs are generative models that aim to train neural network models to sample from an unknown underlying distribution. The key for using DVAEs in machinery prognostics was to find a way to make this underlying distribution represent, $p(r_{1:T}|x_{1:T})$ (the probability of the RUL sequence given a sensor sequence). Hence, Chapter 5 showed how this could be done, while the next chapters showed how altering the architecture could affect the outputs of the DVAE.

8.1.1 Research Questions

DVAEs became a key component in answering the research questions presented in this thesis. These questions can now be looked at, and the answers for each can be summarised here.

1. How could a neural network incorporate dynamic systems to help reduce its black-box nature?

A part of the DVAE framework involved learning a transition model, p_{θ_z}, that described the transition of the latent state, z_t. This transition model could be defined using different models; the standard DVAE used an MLP network, NSDE-DVAEs used NSDEs to describe the latent dynamics, and Kalman-DVAEs used an LGSSM. This could aid the user to visualise the degradation of the machine or even potentially give some control over the degradation due to an understanding of the dynamics, although this point was not explored in this thesis but could be potential future work. DVAEs are trained to generate trajectories from an underlying probability distribution $p(r_{1:T}|x_{1:T})$. Hence, $p(r_{1:T}|x_{1:T})$ can be expressed by simulating multiple trajectories, and the uncertainty can be quantified. By combining the transition model (p_{θ_z}) and measurement model (p_{θ_r}), the DVAE captures this underlying probability distribution while also producing latent dynamics that relate to the RUL. Many state estimation problems, such as the ones that utilise Kalman or Particle filters, also have a similar setup in terms of mathematical underpinnings and dynamic models. By using these underpinnings, deep generative models such as DVAEs could be understood from that perspective. The Kalman-DVAE method was created because of this understanding, and this reduces some of the complexity or black-box nature of deep learning models for machinery prognostics. In the Kalman-DVAE approach, the parameters of an LGSSM system are optimised using the Kalman Filtering algorithm, which is not strictly a deep learning approach. The main use of deep learning in that method is to encode the sensors into a lower dimensional variable, u_t, that acts as a "control" variable in the LGSSM. Hence, the machinery prognostics problem was reduced to identifying the parameters of an LGSSM using Kalman Filtering (the network parameters of the sensor encoder are also optimised this way). This method is hopefully a step towards reducing the black-box nature of deep learning methods in machinery prognostics applications by using methods such as system identification of an LGSSM using a Kalman filtering algorithm.

2. What would these degradation dynamics look like?

Using the different DVAE methods presented in this thesis, the properties of the degradation dynamics could be altered based on the architecture. The standard DVAE used in Chapter 5, the latent dynamics that indicate degradation of the machine are discrete nonlinear dynamics. In Chapter 6, continuous nonlinear latent dynamics are applied using NSDEs. While in Chapter 7, discrete linear dynamics and measurement models are used through the identification of an LGSSM model. The dynamics are somewhat customisable depending on the transition and measurement models chosen. Although trade-offs were noted depending on the models used, e.g. the standard DVAE, which is fully nonlinear, performed better than the LGSSM model that was linear but had simpler latent dynamics.

3. How could uncertainty be incorporated into this framework?

The DVAE incorporates uncertainty naturally due to the fact it is a generative model, and the primary goal of a generative model is to learn how to sample from some underlying probability distribution. By defining a DVAE that, when trained, has an underlying probability distribution of $p(r_{1:T}|x_{1:T})$, this helps quantify the uncertainty of RUL estimates. The DVAE can now generate multiple RUL estimate trajectories that can help represent $p(r_{1:T}|x_{1:T})$ through Monte Carlo simulation. Hence, if the machinery's RUL distribution, $p(r_{1:T}|x_{1:T})$, is complicated, the DVAE can represent it by sampling multiple trajectories. The exception to this is the Kalman-DVAE, as it uses an LGSSM model to represent the RUL and therefore, $p(r_{1:T}|x_{1:T})$, would be represented as a Gaussian distribution.

4. Can this all be incorporated into one framework? If so, how would it be trained and evaluated?

This could all be incorporated into the DVAE framework. Stepping back and looking at the bigger picture, this was done by utilising deep generative models that aim to learn an underlying probability distribution. The DVAE is a generative model that could be utilised to learn this desired underlying distribution, however, note that theoretically, it is not strictly necessary to use the DVAE framework. Future work could explore other generative models, such as GANs or Normalizing Flows. However, in this thesis, the DVAE was used as its underlying mathematics is connected to concepts such as Bayesian filters (e.g. Particle and Kalman Filtering) and they use latent dynamics in their architecture that could represent the degradation of the machine. The DVAE models are generally trained using the ELBO loss function, as shown in Eqn. 3.9, however, in special cases such as the Kalman-DVAE where an LGSSM is used, Kalman filtering can also train the model. Particle filters could also train the DVAE model as $p(r_{1:T}|x_{1:T})$, which can be found using this algorithm for nonlinear and non-Gaussian distributed models, however, that was not explored in this thesis. However, note that the sensor signals, $x_{1:T}$, had to be noncausal inputs for the models. This was handled using sliding time windows, where T was the time window size, and the sensors were noncausal inputs for the model within each time window. Hence, the sequence-to-sequence model could output a sequence of RUL estimates, $r_{1:T}$, given a sequence of sensor variables, $x_{1:T}$. During testing, the DVAE no longer required the inference model. Instead, the generative model, which includes the transition and measurement models, was applied using the time window of sensor signal inputs to generate the RUL estimate and latent variable sequences for each time window. The Kalman-DVAE didn't require an inference model at all, instead using the Kalman filter algorithm during training. It was also evaluated using the generative model, like the standard DVAE, therefore not using the Kalman filtering algorithm during this phase.

8.1.2 Novelties and Key Contributions

In Chapter 4, the main novelty was using the NODE for RUL estimation and comparing it to the ResNet. The comparison showed how NODE could be used to estimate the RUL while remaining competitive with other network architectures. The potential benefits of using NODE were also listed, such as using SDEs instead of ODEs to propagate the variables and account for uncertainty. However, the main contributions from the thesis came from the subsequent chapters that focus on the various DVAE implementations.

In Chapter 5, it was shown that the construction of a conditional DVAE required noncausal conditional variable inputs. This led to a sequence-to-sequence model implementation of the DVAE when used to estimate the RUL conditioned on sensor signals. RUL uncertainty quantification in deep learning methods generally utilises Monte Carlo techniques as seen in section

2.2.6. However, many use Monte Carlo Dropout or Bayesian neural networks to alter the network itself to make it probabilistic instead of deterministic. Here we showed how variational methods may be used to learn how to sample from an underlying probability distribution and quantify the RUL uncertainty. Note the DVAE is still a Monte Carlo method as we require multiple samples to express the marginal distribution $p(r_{1:T}|x_{1:T})$ by repeatedly sampling from the distributions outputted from the transition and measurement neural network models. While these methods exist in the deep learning literature showing how they can be applied to machinery prognostics was not previously done. It also showed it could achieve state-of-the-art performance while still accounting for uncertainty in the RUL estimates.

Chapter 6 showed how NSDEs could be applied to model the latent dynamics of the DVAE. It utilised augmented variables to allow the latent dynamics to have the same initial condition to represent a healthy unit while being able to deviate its trajectory as different machines exhibit different degradation behaviour at different times. The NSDE-DVAE utilised continuous latent dynamics instead of discrete dynamics that assume a constant time step between data points. Although to utilise this fully, one must also change the encoder's architecture to also work with non-constant time steps.

Finally, Chapter 7 showed how to create a DVAE with linear Gaussian State Space latent dynamics and use the Kalman filter to find the marginal likelihood loss. The main novelty in this chapter was to realise that the RUL trajectories are linear and so a Kalman filter could be used to find the marginal likelihood loss instead of indirectly estimating it using the ELBO loss. This also meant the latent dynamics could be relatively simple and easier to work with compared to the neural networks used in the DVAE and NSDE-DVAE latent dynamic models.

In summary, the key contributions were constructing a model that could perform well on the RUL estimation task while accounting for uncertainty. The noncausal nature of the model was also mathematically justified when deriving the DVAEs ELBO loss in section 5.1.1 and this led to the sequence-to-sequence implementation of the model. Finally, various latent dynamic models were tested for the DVAE such as the NSDE and linear state space models which have the potential to be expanded upon in future work. For example, the Kalman-DVAE's latent dynamics are in the form of a linear state-space model which is easier to interpret and work with. In that chapter, this was used to stabilise the dynamics during training. This could be expanded upon by perhaps finding a control law that minimises the degradation of the machine based on the linear state space model degradation dynamics.

8.1.3 Limitations

A potential limitation of this work is the computational complexity. Unlike Monte Carlo dropout, where a dropout mechanism is added to the network and this accounts for uncertainty, the DVAE implements multiple networks to achieve uncertainty quantification (inference, transition and measurement models as well as the bidirectional RNNs that encoded the sensors). The Kalman-DVAE helps alleviate this as it technically only requires networks to encode the sensors, but practically, it performs better if the matrices in the linear state space model are also modelled using neural networks, as outlined in Chapter 7. However, this also gives the DVAE the ability to create latent dynamics that can aid in describing the machine's degradation; hence, the user is presented with a trade-off. Note that this was a downside of NODEs when compared to ResNets in Chapter 4, as evaluating the ODE can be more computationally expensive than ResNet when using solvers such as Runge-Kutta.

Like all deep learning models, a practical limitation of these models is that data is required to train these models for effective RUL estimation. Semi-supervised learning could aid with that problem, but some failure data is still needed. This model could not take cases outside of the domain it was trained on and learn how to predict the RUL of other machines based on what it was trained on. Unlike general laws of physics that could model certain types of degradation processes for a variety of components and machines (e.g. crack length given certain properties of a material) the latent dynamics are not general laws that apply to the same degradation process in other machines. This may be explored in future work by trying to match the latent dynamics of different machines with the same degradation process, but this would require more data and was not the scope of this thesis.

A limitation of the Kalman-DVAE is that it assumes the variables are all described using Gaussian distributions. This may not be a valid assumption for the RUL in many cases, and this may need to be altered to enable the use of non-Gaussian distributions. However, this would mean the primary benefit of using the Kalman filter to calculate the marginal likelihood loss would not be possible. One could use a nonlinear filter such as an Unscented Kalman filter or Particle filter to aid with this limitation, but this would increase the computational complexity. Note that using nonlinear filters such as the Extended Kalman filter for estimating the marginal likelihood loss has been successfully used in other applications (A. H. Li et al., 2021).

Some of these may be overcome with more research, e.g., one could find a way to apply the latent dynamics to different types of machines with the same degradation process to estimate their RUL. But many are also limitations that must be worked around, such as the increased computational complexity compared to Monte Carlo dropout or the fact the Kalman filtering algorithm requires Gaussian distributed variables. With the benefits and limitations stated, we can now go on to discuss potential future work that can expand the work presented in the thesis.

8.2 Future Work

Many different avenues could be explored as future work when considering the content introduced in this thesis. Both theoretical and practical extensions can be made to the methods presented here.

- **Neural Differential Equations describing larger neural networks**
  - **Neural SDEs as networks**: In Chapter 4, a NODE is used to describe the computations of a larger neural network. Instead of a NODE, an NSDE could be used. This could be analogous to using a Bayesian Neural Network over an MLP network, as the computations within the network itself incorporate randomness and allow for multiple possible outputs. Using this as a network for machinery prognostics that uses sensor signals as inputs and outputs the RUL estimate, much like the NODE

and ResNet was used in Chapter 4, but it would be probabilistic and could therefore quantify uncertainty.

- **DVAE Structure**
  - **Trying different encoders for sensors**: In this thesis, different models were used for the DVAEs transition, measurement and inference models; while more experiments could try different network architectures for these models, one could also change the encoder network architecture. In each DVAE method presented, the encoder used to find a noncausal sequence representation has been a bidirectional RNN, as is common in the DVAE literature (Girin et al., 2021). However, other architectures could be used to find a noncausal representation of the sensors, such as the Transformer (Vaswani et al., 2017). As noted in the Kalman-DVAE method, the encoding of the sensors is an important aspect of these DVAEs. The Kalman-DVAE could still get acceptable performance with the simpler LGSSM used to describe the transition and measurement models, with the only nonlinear component coming from the nonlinear sensor encoding. Hence, looking at better network architectures for encoding the sensors may be a more practical avenue to explore for improving these DVAE models for machinery prognostics applications.
  - **Altering DVAE dynamics to incorporate physics-based methods**: Physics-inspired networks are used to perform RUL estimation in works such as (Nascimento & Viana, 2019; Yucesan & Viana, 2019; Dourado & Viana, 2019). Future work could look at incorporating these types of physics-inspired networks in the DVAEs transition and measurement models. This could help the DVAE incorporate expert knowledge into the model and potentially reduce the amount of data needed to train the model.
  - **Altering DVAE dynamics to incorporate statistical methods**: Many statistical methods in machinery prognostics involve modelling an SDE to describe the degradation behaviour of the machine. The DVAE or, more specifically, the NSDE-DVAE, could be altered to incorporate the structure of these SDEs and model the degradation with SDEs that are well suited for the task. This could be another way of incorporating expert knowledge or bridging the gap between Deep Learning and statistical methods in machinery prognostics.
  - **Improving the dynamics properties**: The latent dynamics of the DVAE could be improved by utilising the properties of the dynamic system. For example, the LGSSM dynamics were regularised during training using the eigenvalues of the transition matrices. By understanding how eigenvalues can alter the stability of the dynamic system, this knowledge could be used to improve the latent dynamics of the model in the Kalman-DVAE case. The same approach could be made for any dynamic system used in the DVAE. Similar approaches were made for NODEs; through the understanding of ODEs, NODEs could be altered, and their performance could be improved, such as in (Dupont et al., 2019; Finlay et al., 2020). Hence, by using existing knowledge of the properties of dynamic systems, the DVAE framework used here for RUL estimation could potentially be improved.
- **DVAE Methods**
  - **Semi-supervised methods**: Semi-supervised methods can be expanded in more detail. Generative models are already utilised in machinery prognostics for semi-

supervised learning and reducing the amount of training data needed for deep learning methods. Many of these methods do not utilise DVAEs but instead use generative models more suited to non-sequential data, e.g. VAEs. One potential avenue to explore is that if a DVAE is trained to generate sensor signals, our inference model would have the structure $q_\phi(z_t|z_{t-1}, x_{1:T})$. The DVAE we used for RUL estimation has this same structure, i.e. $p_{\theta_z}(z_t|z_{t-1}, x_{1:T})$. Hence, perhaps we could pre-train the transition model p_{θ_z} by training a DVAE in an unsupervised manner on the sensor signals, then use that inference model as p_{θ_z} for training a DVAE to estimate the RUL. This way, the parameters of the network were already partially optimised during the unsupervised training, meaning it may require less run-to-failure data to train the DVAE for RUL estimation.

  - **Identifying troublesome sensors**: The model could potentially be altered to identify exactly what sensor variables are capturing the primary cause of the machinery degradation. For example, if attention mechanisms are used instead of a bi-directional RNN for sensor encoding, perhaps the attention weights along the sensor dimension could help identify which sensors most contribute to the machine's degradation. This could reduce the black-box nature of Deep Learning methods and keep the benefits of the DVAE presented in the thesis (uncertainty quantification and interpretable latent dynamics).

- **Control Theory and Machinery Prognostics**

  - **Kalman-DVAE control laws**: The Kalman-DVAE defines an LGSSM that models the RUL and degradation (latent) trajectories. LGSSMs are often used in control theory and a relatively easy to work with. Perhaps one could design a control law that considers the machine's degradation and can alter its operation to reduce the degradation experienced by the machine to an extent.
  - **Nonlinear control with DVAE**: One could take a network like the NSDE-DVAE or DVAE, which have nonlinear latent dynamics and design control laws using these dynamics. Hence, the Kalman-DVAE may offer a simpler method to test out various control laws, but ultimately, the DVAE and NSDE-DVAE have superior performance when it comes to RUL estimation. DVAEs have already been used in control theory applications such as in (Hafner et al., 2019), where reinforcement learning is used to plan motion from motion captured in a video dataset. This would need to be extended to the noncausal, conditional DVAE case for the models presented in this thesis.

These are some of the major areas which could be explored further to expand upon the work done in this thesis.

Appendix A

NODE-CNN and ResNet Hyperparameters

ResNet Hyperparameters

This appendix contains the hyperparameters used for training the ResNet on different datasets. Table A.1 states the values of these hyperparameters.

Table A.1: Hyperparameters used to train the ResNet and affect its architecture. Hyperparameters for each C-MAPSS dataset are listed

Hyperparameter	FD001	FD002	FD003	FD004
Batch size (bs)	586	213	538	348
Learning rate	3.39×10^{-5}	6.97×10^{-3}	9.79×10^{-3}	2.21×10^{-5}
Weight Decay (L2)	2.10×10^{-4}	7.54×10^{-4}	4.31×10^{-4}	2.17×10^{-4}
Time Window	28	10	12	15
Data split (*split* in Eq. (4.3))	184	197	64	123
Training epochs	99	57	59	62
Convolution Kernel size	3	3	3	3
Convolution channel size	64	64	64	64
Amount of ResNet blocks (n)	6	6	6	6

NODE-CNN Hyperparameters

This appendix contains the hyperparameters used for training the NODE-CNN on different datasets. Table A.2 states the values of these hyperparameters.

Table A.2: Hyperparameters used to train the NODE-CNN and affect its architecture. Hyperparameters for each C-MAPSS dataset are listed

Hyperparameter	FD001	FD002	FD003	FD004
Batch size (bs)	206	580	207	482
Learning rate	9.22×10^{-3}	5.52×10^{-4}	1.11×10^{-3}	5.46×10^{-3}
Weight Decay (L2)	1.14×10^{-4}	1.08×10^{-9}	3.26×10^{-9}	2.75×10^{-8}
Time Window	30	10	38	15
Data split (*split* in Eq. (4.3))	103	70	96	75
Training epochs	136	149	65	148
Convolution Kernel size	3	3	3	3
Convolution channel size	64	64	64	64
steps for RK solver	5	5	5	5

Appendix B

DVAE Experiment Hyperparameters

Table B.1 shows the hyperparameters used both in the supervised and semi-supervised DVAE experiments for all the CMAPSS datasets.

Table B.1: Hyperparameters for the Kalman-DVAE for all datasets

Hyperparameters	Values
Training epochs	200
Batch size (bs)	250
Learning rate	1×10^{-3}
L2 regularisation constant (weight decay)	1×10^{-5}
Time Window	40
Training/Valid split	80/20

Appendix C

NSDE-DVAE Hyperparameters

Table C.1 states the hyperparameters used when training the neural networks (the time window is also used when evaluating the dataset). They were all trained for 100 epochs and with a weight decay of 1×10^{-5}.

Table C.1: Hyperparameters used for each dataset when training the neural networks

Dataset	learning rate	time window (T)	batch size	β
FD001	0.001	40	150	1
FD002	0.001	40	150	1
FD003	0.001	44	200	0.48
FD004	0.00041	47	167	0.76

Appendix D

NSDE-DVAE Network Details

The DVAE consisted of multiple networks including Multi-Layered Perceptron (MLP) networks and Gated Recurrent Networks (GRU).

The RNNs capturing sensor signals ($x_{1:t}$) and RUL ($r_{1:t}$) time series:

Both are GRUs with sizes,

- Input size: RUL GRU = 1, sensor GRU = amount of sensor signals chosen
- Hidden/GRU output size: 50

The backwards GRU for the inference model to represent $x_{1:T}$ has sizes,

- Input size: output of the sensor GRU + amount of sensors: 50 + number of sensors
- Hidden/GRU output size: 50

Encoder:

A MLP with sizes,

- Input size: includes both hidden dimension sizes of $50 + 50 = 100$
- Hidden size: 150
- Output size: number of latent dimension + number of augmented dimensions (times 2 as mean and log-variance is the output): $2 \times (2 + 1) = 6$

Decoder:

A MLP with sizes,

- Input size: number of latent dimension + number of augmented dimensions (times 2 as mean and log-variance is the output): $2 \times (2 + 1) = 6$
- Hidden size: 150
- Output size: 1×2 (as both mean and log-variance are the outputs)

NSDE:

Drift function is a MLP with sizes,

- Input size: number of latent dimension + number of augmented dimensions: $2 + 1 = 3$
- Hidden size: 150
- Output size: 1 (state space hence this network is the output for only $\dot{z}_2$ in the 2-dimensional latent space case)

Diffusion is a constant parameter ($\boldsymbol{\theta}_d$) that can be optimised during training.

Appendix E

Kalman-DVAE Hyperparameters

Table E.1 shows the hyperparameters for the Kalman DVAE training on all the CMAPSS datasets.

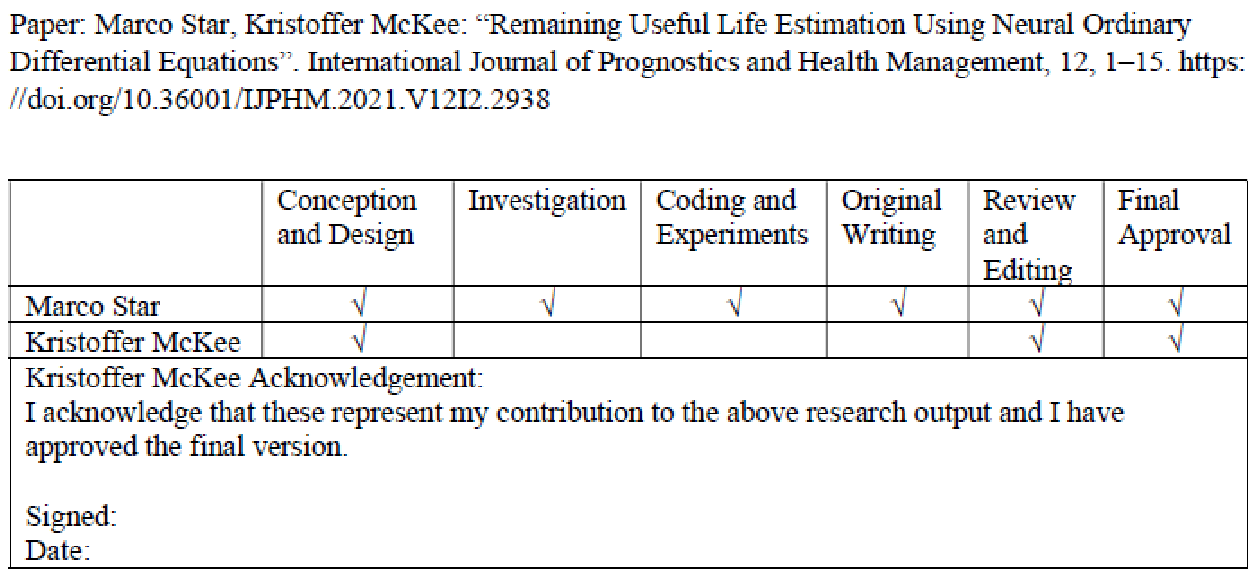
Table E.1: Hyperparameters for the Kalman-DVAE for all datasets

Hyperparameters	Values
Training epochs	100
Batch size (bs)	150
Learning rate	1×10^{-3}
Time Window	40
Replay Overshoot (α)	0.6
Regularisation Constant (β)	1×10^{-4}
Amount of Global Matrices (K see Eqn. 7.6)	2 (FD001/2) or 3 (FD003/4)
Training/Valid split	80/20

Note that "Amount of Global Matrices" K changes values from 2 to 3 based on whether the dataset needs 2 or 3 different "mathematical models". In FD001 and FD002, there is a healthy and degrading stage; hence, the two global matrices mean both dynamics of the healthy and degradation stage can be captured. For FD003 and FD004, there is also a healthy stage, but it has two potential failure modes. Hence, $K = 3$ is used, so one matrix describes the healthy dynamics as before, and the other two describe each failure mode.

Appendix F

Authorship Statement



Paper: Marco Star, Kristoffer McKee: "Remaining Useful Life Estimation Using Neural Ordinary Differential Equations". International Journal of Prognostics and Health Management, 12, 1–15. https://doi.org/10.36001/IJPHM.2021.V12I2.2938

	Conception and Design	Investigation	Coding and Experiments	Original Writing	Review and Editing	Final Approval
Marco Star	√	√	√	√	√	√
Kristoffer McKee	√				√	√

Kristoffer McKee Acknowledgement:
I acknowledge that these represent my contribution to the above research output and I have approved the final version.

Signed:
Date:

Bibliography

Xia, T., Dong, Y., Xiao, L., Du, S., Pan, E., & Xi, L. (2018). Recent advances in prognostics and health management for advanced manufacturing paradigms. *Reliability Engineering & System Safety*, *178*, 255–268. https://doi.org/https://doi.org/10.1016/j.ress.2018.06.021

Diez-Olivan, A., Del Ser, J., Galar, D., & Sierra, B. (2019). Data fusion and machine learning for industrial prognosis: Trends and perspectives towards Industry 4.0. *Information Fusion*, *50*, 92–111. https://doi.org/10.1016/j.inffus.2018.10.005

Peng, W., Ye, Z.-S., & Chen, N. (2019). Bayesian Deep Learning based Health Prognostics Towards Prognostics Uncertainty. *IEEE Transactions on Industrial Electronics*, 1–1. https://doi.org/10.1109/TIE.2019.2907440

Xu, Z., & Saleh, J. H. (2021). Machine learning for reliability engineering and safety applications: Review of current status and future opportunities. *Reliability Engineering and System Safety*, *211*, 107530. https://doi.org/10.1016/j.ress.2021.107530

Sankararaman, S. (2015). Significance, interpretation, and quantification of uncertainty in prognostics and remaining useful life prediction. *Mechanical Systems and Signal Processing*, *52-53*(1), 228–247. https://doi.org/10.1016/j.ymssp.2014.05.029

Kraus, M., & Feuerriegel, S. (2019). Forecasting remaining useful life: Interpretable deep learning approach via variational Bayesian inferences. *Decision Support Systems*, *125*(July), 113100. https://doi.org/10.1016/j.dss.2019.113100

Nascimento, R. G., & Viana, F. A. C. (2019). Fleet Prognosis with Physics-informed Recurrent Neural Networks. http://arxiv.org/abs/1901.05512

Yucesan, Y. A., & Viana, F. A. C. (2019). Wind Turbine Main Bearing Fatigue Life Estimation with Physics-informed Neural Networks. *Phm 2019*, *11*(1), 1–14. https://doi.org/10.36001/PHMCONF.2019.V11I1.807

Dourado, A., & Viana, F. A. C. (2019). Physics-Informed Neural Networks for Corrosion-Fatigue Prognosis. *Phm 2019*, *11*(1), 1–12.

González-Muñiz, A., Díaz, I., Cuadrado, A. A., & García-Pérez, D. (2022). Health indicator for machine condition monitoring built in the latent space of a deep autoencoder. *Reliability Engineering & System Safety*, *224*, 108482. https://doi.org/10.1016/J.RESS.2022.108482

Qin, Y., Zhou, J., & Chen, D. (2021). Unsupervised health indicator construction by a novel degradation-trend-constrained variational autoencoder and its applications. *IEEE/ASME Transactions on Mechatronics*. https://doi.org/10.1109/TMECH.2021.3098737

Ping, G., Chen, J., Pan, T., & Pan, J. (2019). Degradation feature extraction using multi-source monitoring data via logarithmic normal distribution based variational auto-encoder. *Computers in Industry*, *109*, 72–82. https://doi.org/10.1016/j.compind.2019.04.013

She, D., Jia, M., & Pecht, M. G. (2020). Sparse auto-encoder with regularization method for health indicator construction and remaining useful life prediction of rolling bearing.

Measurement Science and Technology, *31*, 105005. https://doi.org/10.1088/1361-6501/ab8c0f

Listou Ellefsen, A., Bjørlykhaug, E., Æsøy, V., Ushakov, S., & Zhang, H. (2019). Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture. *Reliability Engineering and System Safety*, *183*, 240–251. https://doi.org/10.1016/j.ress.2018.11.027

Kingma, D. P., & Welling, M. (2014). Auto-encoding variational bayes. *CoRR*, *abs/1312.6114*.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., & Davidson, J. (2019). Learning latent dynamics for planning from pixels. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (pp. 2555–2565). PMLR. https://proceedings.mlr.press/v97/hafner19a.html

Fraccaro, M., Sønderby, S. K., Paquet, U., & Winther, O. (2016). Sequential neural models with stochastic layers. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems*. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2016/file/208e43f0e45c4c78cafadb83d2888cb6-Paper.pdf

Fraccaro, M., Kamronn, S., Paquet, U., & Winther, O. (2017). A disentangled recognition and nonlinear dynamics model for unsupervised learning. *Advances in Neural Information Processing Systems*, *2017-December*, 3602–3611. http://arxiv.org/abs/1710.05741

Karl, M., Soelch, M., Bayer, J., Smagt, P. V. D., & Group, V. (2017). Deep Variational Bayes Filters : Unsupervised Learning of State Space Models from Raw Data. (2), 1–13.

Becker-Ehmck, P., Peters, J., & Van Der Smagt, P. (2019). Switching linear dynamics for variational bayes filtering. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (pp. 553–562). PMLR. https://proceedings.mlr.press/v97/becker-ehmck19a.html

Maddison, C. J., Lawson, D., Tucker, G., Heess, N., Norouzi, M., Mnih, A., Doucet, A., & Teh, Y. W. (2017). Filtering variational objectives. https://doi.org/10.48550/arxiv.1705.09279

Naesseth, C., Linderman, S., Ranganath, R., & Blei, D. (2018). Variational sequential monte carlo. In A. Storkey & F. Perez-Cruz (Eds.), *Proceedings of the twenty-first international conference on artificial intelligence and statistics* (pp. 968–977). PMLR. https://proceedings.mlr.press/v84/naesseth18a.html

Jonschkowski, R., Rastogi, D., & Brock, O. (2018). Differentiable particle filters: End-to-end learning with algorithmic priors. *Robotics: Science and Systems XIV*. https://doi.org/10.15607/RSS.2018.XIV.001

Corenflos, A., Thornton, J., Deligiannidis, G., & Doucet, A. (2021). Differentiable particle filtering via entropy-regularized optimal transport. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (pp. 2100–2111). PMLR. https://proceedings.mlr.press/v139/corenflos21a.html

Kloss, A., Martius, G., & Bohg, J. (2021). How to train your differentiable filter. *Autonomous Robots*, *45*, 561–578. https://doi.org/10.1007/s10514-021-09990-9

Girin, L., Leglaive, S., Bie, X., Diard, J., Hueber, T., & Alameda-Pineda, X. (2021). Dynamical Variational Autoencoders: A Comprehensive Review. *Foundations and Trends® in Machine Learning*, *15*(1-2), 1–175. https://doi.org/10.1561/2200000089

Wei, Y., Wu, D., & Terpenny, J. (2021). Learning the health index of complex systems using dynamic conditional variational autoencoders. *Reliability Engineering & System Safety*, *216*, 108004. https://doi.org/10.1016/J.RESS.2021.108004

Paris, P. C., & Erdogan, F. (1963). A critical analysis of crack propagation laws. *Journal of Basic Engineering*, *85*(4), 528–533. https://doi.org/10.1115/1.3656900

Lei, Y., Li, N., Guo, L., Li, N., Yan, T., & Lin, J. (2018). Machinery health prognostics: A systematic review from data acquisition to RUL prediction. https://doi.org/10.1016/j.ymssp.2017.11.016

Cubillo, A., Perinpanayagam, S., & Esperon-Miguez, M. (2016). A review of physics-based models in prognostics: Application to gears and bearings of rotating machinery. *Advances in Mechanical Engineering*, *8*. https://doi.org/10.1177/1687814016664660/ASSET/IMAGES/LARGE/10.1177_1687814016664660-FIG9.JPEG

Lei, Y., Li, N., Gontarz, S., Lin, J., Radkowski, S., & Dybala, J. (2016). A model-based method for remaining useful life prediction of machinery. *IEEE Transactions on Reliability*, *65*, 1314–1326. https://doi.org/10.1109/TR.2016.2570568

Liao, L., & Köttig, F. (2016). A hybrid framework combining data-driven and model-based methods for system remaining useful life prediction. *Applied Soft Computing Journal*, *44*, 191–199. https://doi.org/10.1016/j.asoc.2016.03.013

Liu, H., Chen, J., Hissel, D., & Su, H. (2019). Remaining useful life estimation for proton exchange membrane fuel cells using a hybrid method. *Applied Energy*, *237*(January), 910–919. https://doi.org/10.1016/j.apenergy.2019.01.023

Hagmeyer, S., & Zeiler, P. (2023). A comparative study on methods for fusing data-driven and physics-based models for hybrid remaining useful life prediction of air filters. *IEEE Access*. https://doi.org/10.1109/ACCESS.2023.3265722

Baptista, M., Henriques, E. M. P., de Medeiros, I. P. P., Malere, J. P. P., Nascimento, C. L. L., & Prendinger, H. (2019). Remaining useful life estimation in aeronautics: Combining data-driven and Kalman filtering. *Reliability Engineering and System Safety*, *184*(February 2018), 228–239. https://doi.org/10.1016/j.ress.2018.01.017

Zhang, H., Miao, Q., Zhang, X., & Liu, Z. (2018). An improved unscented particle filter approach for lithium-ion battery remaining useful life prediction. *Microelectronics Reliability*, *81*(24), 288–298. https://doi.org/10.1016/j.microrel.2017.12.036

Zhao, G., Zhang, G., Liu, Y., Zhang, B., & Hu, C. (2017). Lithium-ion battery remaining useful life prediction with Deep Belief Network and Relevance Vector Machine. *2017 IEEE International Conference on Prognostics and Health Management, ICPHM 2017*, 7–13. https://doi.org/10.1109/ICPHM.2017.7998298

Hu, X., Jiang, J., Cao, D., & Egardt, B. (2016). Battery health prognosis for electric vehicles using sample entropy and sparse Bayesian predictive modeling. *IEEE Transactions on Industrial Electronics*, *63*(4), 2645–2656. https://doi.org/10.1109/TIE.2015.2461523

Wu, J., Su, Y., Cheng, Y., Shao, X., Deng, C., & Liu, C. (2018). Multi-sensor information fusion for remaining useful life prediction of machining tools by adaptive network based fuzzy inference system. *Applied Soft Computing Journal*, *68*, 13–23. https://doi.org/10.1016/j.asoc.2018.03.043

Li, X., Yang, X., Yang, Y., Bennett, I., & Mba, D. (2019). An intelligent diagnostic and prognostic framework for large-scale rotating machinery in the presence of scarce failure data. *Structural Health Monitoring*. https://doi.org/10.1177/1475921719884019

Son, K. L., Fouladirad, M., Barros, A., Levrat, E., & Iung, B. (2013). Remaining useful life estimation based on stochastic deterioration models: A comparative study. *Reliability Engineering and System Safety*, *112*, 165–175. https://doi.org/10.1016/j.ress.2012.11.022

Guo, L., Li, N., Jia, F., Lei, Y., & Lin, J. (2017). A recurrent neural network based health indicator for remaining useful life prediction of bearings. *Neurocomputing*, *240*, 98–109. https://doi.org/10.1016/j.neucom.2017.02.045

Liu, X., Chen, G., Cheng, Z., Wei, X., & Wang, H. (2022). Convolution neural network based particle filtering for remaining useful life prediction of rolling bearing. *Advances in Mechanical Engineering*, *14*, 168781322211006. https://doi.org/10.1177/16878132221100631

Xu, F., Yang, F., Fan, X., Huang, Z., & Tsui, K. L. (2020). Extracting degradation trends for roller bearings by using a moving-average stacked auto-encoder and a novel exponential function. *Measurement: Journal of the International Measurement Confederation*, *152*. https://doi.org/10.1016/j.measurement.2019.107371

Mao, W., He, J., Tang, J., & Li, Y. (2018). Predicting remaining useful life of rolling bearings based on deep feature representation and long short-term memory neural network. *Advances in Mechanical Engineering*, *10*(12), 168781401881718. https://doi.org/10.1177/1687814018817184

Du, Y., Wu, T., & Makis, V. (2017). Parameter estimation and remaining useful life prediction of lubricating oil with HMM. *Wear*, *376-377*, 1227–1233. https://doi.org/10.1016/j.wear.2016.11.047

Kim, M. J., Jiang, R., Makis, V., & Lee, C. G. (2011). Optimal Bayesian fault prediction scheme for a partially observable system subject to random failure. *European Journal of Operational Research*, *214*(2), 331–339. https://doi.org/10.1016/j.ejor.2011.04.023

Kumar, A., Chinnam, R. B., & Tseng, F. (2019). An HMM and polynomial regression based approach for remaining useful life and health state estimation of cutting tools. *Computers and Industrial Engineering*, *128*(May 2018), 1008–1014. https://doi.org/10.1016/j.cie.2018.05.017

Xia, M., Li, T., Shu, T., Wan, J., De silva, C. W., & Wang, Z. (2018). A Two-Stage Approach for the Remaining Useful Life Prediction of Bearings using Deep Neural Networks. *IEEE Transactions on Industrial Informatics*, *PP*(100), 1. https://doi.org/10.1109/TII.2018.2868687

Heimes, F. O. (2008). Recurrent neural networks for remaining useful life estimation. *2008 International Conference on Prognostics and Health Management*, 1–6. https://doi.org/10.1109/PHM.2008.4711422

Cao, L., Zhang, H., Meng, Z., & Wang, X. (2023). A parallel gru with dual-stage attention mechanism model integrating uncertainty quantification for probabilistic rul prediction of wind turbine bearings. *Reliability Engineering & System Safety*, *235*, 109197. https://doi.org/https://doi.org/10.1016/j.ress.2023.109197

Guo, J., Wang, J., Wang, Z., Gong, Y., Qi, J., Wang, G., & Tang, C. (2023). A cnn-bilstm-bootstrap integrated method for remaining useful life prediction of rolling bearings. *Quality and Reliability Engineering International*. https://doi.org/10.1002/QRE.3314

Saha, B., Goebel, K., Poll, S., & Christophersen, J. (2009). Prognostics methods for battery health monitoring using a Bayesian framework. *IEEE Transactions on Instrumentation and Measurement*, *58*(2), 291–296. https://doi.org/10.1109/TIM.2008.2005965

Saha, B., & Goebel, K. (2008). Uncertainty management for diagnostics and prognostics of batteries using Bayesian techniques. *IEEE Aerospace Conference Proceedings*, 1–8. https://doi.org/10.1109/AERO.2008.4526631

Liu, Q., Dong, M., Lv, W., Geng, X., & Li, Y. (2015). A novel method using adaptive hidden semi-Markov model for multi-sensor monitoring equipment health prognosis. *Mechanical Systems and Signal Processing*, *64-65*, 217–232. https://doi.org/10.1016/j.ymssp.2015.03.029

Cannarile, F., Compare, M., Rossi, E., & Zio, E. (2017). A fuzzy expectation maximization based method for estimating the parameters of a multi-state degradation model from imprecise maintenance outcomes. *Annals of Nuclear Energy*, *110*, 739–752. https://doi.org/10.1016/j.anucene.2017.07.017

Zhang, W., Yang, D., & Wang, H. (2019). Data-Driven Methods for Predictive Maintenance of Industrial Equipment: A Survey. *IEEE Systems Journal*, *13*(3), 2213–2227. https://doi.org/10.1109/JSYST.2019.2905565

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*. https://doi.org/10.1016/0893-6080(89)90020-8

Wu, Y., Yuan, M., Dong, S., Lin, L., & Liu, Y. (2018). Remaining useful life estimation of engineered systems using vanilla LSTM neural networks. *Neurocomputing*, *275*, 167–179. https://doi.org/10.1016/j.neucom.2017.05.063

Wang, F., Han, Q., Liu, X., Deng, G., Yu, X., & Li, H. (2019). Remaining Life Prediction Method for Rolling Bearing Based on the Long Short-Term Memory Network. *Neural Processing Letters*, *50*(3), 2437–2454. https://doi.org/10.1007/s11063-019-10016-w

Huang, Y., Liu, C., Zha, X. F., & Li, Y. (2010). A lean model for performance assessment of machinery using second generation wavelet packet transform and Fisher criterion. *Expert Systems with Applications*, *37*(5), 3815–3822. https://doi.org/10.1016/j.eswa.2009.11.038

McKee, K. K., Forbes, G. L., Mazhar, I., Entwistle, R., Hodkiewicz, M., & Howard, I. (2015). A vibration cavitation sensitivity parameter based on spectral and statistical methods. *Expert Systems with Applications*, *42*(1), 67–78. https://doi.org/10.1016/j.eswa.2014.07.029

Lei, Y., Zuo, M. J., He, Z., & Zi, Y. (2010). A multidimensional hybrid intelligent method for gear fault diagnosis. *Expert Systems with Applications*, *37*(2), 1419–1430. https://doi.org/10.1016/j.eswa.2009.06.060

Chen, Z. Q., Li, C., & Sanchez, R. V. (2015). Gearbox Fault Identification and Classification with Convolutional Neural Networks. *Shock and Vibration*, *2015*. https://doi.org/10.1155/2015/390134

Peng, Z. K., & Chu, F. L. (2004). Application of the wavelet transform in machine condition monitoring and fault diagnostics: A review with bibliography. *Mechanical Systems and Signal Processing*, *18*(2), 199–221. https://doi.org/10.1016/S0888-3270(03)00075-X

Kanarachos, S., Christopoulos, S. R. G., Chroneos, A., & Fitzpatrick, M. E. (2017). Detecting anomalies in time series data via a deep learning algorithm combining wavelets, neural networks and Hilbert transform. *Expert Systems with Applications*, *85*, 292–304. https://doi.org/10.1016/j.eswa.2017.04.028

Ren, L., Cheng, X., Wang, X., Cui, J., & Zhang, L. (2019). Multi-scale Dense Gate Recurrent Unit Networks for bearing remaining useful life prediction. *Future Generation Computer Systems*, *94*, 601–609. https://doi.org/10.1016/j.future.2018.12.009

Chen, C., Liu, Y., Wang, S., Sun, X., Di Cairano-Gilfedder, C., Titmus, S., & Syntetos, A. A. (2020). Predictive maintenance using cox proportional hazard deep learning. *Advanced Engineering Informatics*, *44*(December 2019), 101054. https://doi.org/10.1016/j.aei.2020.101054

Yu, W., Kim, I. Y., & Mechefske, C. (2019). Remaining useful life estimation using a bidirectional recurrent neural network based autoencoder scheme. *Mechanical Systems and Signal Processing*, *129*, 764–780. https://doi.org/10.1016/J.YMSSP.2019.05.005

Yang, L., Wang, F., Zhang, J., & Ren, W. (2019). Remaining useful life prediction of ultrasonic motor based on Elman neural network with improved particle swarm optimization. *Mea-*

surement: Journal of the International Measurement Confederation, *143*, 27–38. https://doi.org/10.1016/j.measurement.2019.05.013

Zhang, B., Zhang, S., & Li, W. (2019). Bearing performance degradation assessment using long short-term memory recurrent network. *Computers in Industry*, *106*, 14–29. https://doi.org/10.1016/j.compind.2018.12.016

Aggarwal, K., Atan, O., Farahat, A. K., Zhang, C., Ristovski, K., & Gupta, C. (2019). Two birds with one network: Unifying failure event prediction and time-to-failure modeling. *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018*, 1308–1317. https://doi.org/10.1109/BigData.2018.8622431

Zhang, Y., Xiong, R., He, H., & Pecht, M. G. (2018). Long short-term memory recurrent neural network for remaining useful life prediction of lithium-ion batteries. *IEEE Transactions on Vehicular Technology*, *67*(7), 5695–5705. https://doi.org/10.1109/TVT.2018.2805189

Zhou, Y., Huang, Y., Pang, J., & Wang, K. (2019). Remaining useful life prediction for supercapacitor based on long short-term memory neural network. *Journal of Power Sources*, *440*(March), 227149. https://doi.org/10.1016/j.jpowsour.2019.227149

Elsheikh, A., Yacout, S., & Ouali, M. S. (2019). Bidirectional handshaking LSTM for remaining useful life prediction. *Neurocomputing*, *323*, 148–156. https://doi.org/10.1016/j.neucom.2018.09.076

Wu, J., Hu, K., Cheng, Y., Zhu, H., Shao, X., & Wang, Y. (2019). Data-driven remaining useful life prediction via multiple sensor signals and deep long short-term memory neural network. *ISA Transactions*, *97*, 241–250. https://doi.org/10.1016/j.isatra.2019.07.004

Huang, C. G., Huang, H. Z., & Li, Y. F. (2019). A Bidirectional LSTM Prognostics Method Under Multiple Operational Conditions. https://doi.org/10.1109/TIE.2019.2891463

Radaideh, M. I., Pigg, C., Kozlowski, T., Deng, Y., & Qu, A. (2020). Neural-based time series forecasting of loss of coolant accidents in nuclear power plants. *Expert Systems with Applications*, *160*, 113699. https://doi.org/10.1016/j.eswa.2020.113699

Zhu, J., Chen, N., & Peng, W. (2019). Estimation of Bearing Remaining Useful Life Based on Multiscale Convolutional Neural Network. *IEEE Transactions on Industrial Electronics*, *66*(4), 3208–3216. https://doi.org/10.1109/TIE.2018.2844856

Li, X., Zhang, W., & Ding, Q. (2019). Deep learning-based remaining useful life estimation of bearings using multi-scale feature extraction. *Reliability Engineering and System Safety*, *182*, 208–218. https://doi.org/10.1016/j.ress.2018.11.011

Yang, W., Yao, Q., Ye, K., & Xu, C. Z. (2020). Empirical Mode Decomposition and Temporal Convolutional Networks for Remaining Useful Life Estimation. *International Journal of Parallel Programming*, *48*(1), 61–79. https://doi.org/10.1007/s10766-019-00650-1

Wang, Q., Zhao, B., Ma, H., Chang, J., & Mao, G. (2019). A method for rapidly evaluating reliability and predicting remaining useful life using two-dimensional convolutional neural network with signal conversion. *Journal of Mechanical Science and Technology*, *33*(6), 2561–2571. https://doi.org/10.1007/s12206-019-0504-x

Li, X., Ding, Q., & Sun, J. Q. (2018). Remaining useful life estimation in prognostics using deep convolution neural networks. *Reliability Engineering and System Safety*, *172*(December 2017), 1–11. https://doi.org/10.1016/j.ress.2017.11.021

Yang, H., Zhao, F., Jiang, G., Sun, Z., & Mei, X. (2019). A Novel Deep Learning Approach for Machinery Prognostics Based on Time Windows. *Applied Sciences*, *9*(22), 4813. https://doi.org/10.3390/app9224813

Yang, B., Liu, R., & Zio, E. (2019). Remaining useful life prediction based on a double-convolutional neural network architecture. *IEEE Transactions on Industrial Electronics*, *66*(12), 9521–9530. https://doi.org/10.1109/TIE.2019.2924605

Wang, B., Lei, Y., Li, N., & Yan, T. (2019). Deep separable convolutional network for remaining useful life prediction of machinery. *Mechanical Systems and Signal Processing*, *134*, 106330. https://doi.org/10.1016/j.ymssp.2019.106330

Jiang, J.-R., Lee, J.-E., & Zeng, Y.-M. (2019). Time Series Multiple Channel Convolutional Neural Network with Attention-Based Long Short-Term Memory for Predicting Bearing Remaining Useful Life. *Sensors*, *20*(1), 166. https://doi.org/10.3390/s20010166

Kong, Z., Cui, Y., Xia, Z., & Lv, H. (2019). Convolution and Long Short-Term Memory Hybrid Deep Neural Networks for Remaining Useful Life Prognostics. *Applied Sciences*, *9*(19), 4156. https://doi.org/10.3390/app9194156

Zhao, R., Yan, R., Wang, J., & Mao, K. (2017). Learning to monitor machine health with convolutional Bi-directional LSTM networks. *Sensors (Switzerland)*, *17*(2). https://doi.org/10.3390/s17020273

An, Q., Tao, Z., Xu, X., El Mansori, M., & Chen, M. (2020). A data-driven model for milling tool remaining useful life prediction with convolutional and stacked LSTM network. *Measurement: Journal of the International Measurement Confederation*, *154*, 107461. https://doi.org/10.1016/j.measurement.2019.107461

Wang, B., Lei, Y., Yan, T., Li, N., & Guo, L. (2019). Recurrent convolutional neural network : A new framework for remaining useful life prediction of machinery. *Neurocomputing*, *379*(40), 117–129. https://doi.org/10.1016/j.neucom.2019.10.064

Ragab, M., Chen, Z., Wu, M., Kwoh, C.-K., Yan, R., & Li, X. (2021). Attention-based sequence to sequence model for machine remaining useful life prediction. *Neurocomputing*, *466*, 58–68. https://doi.org/10.1016/j.neucom.2021.09.022

Chen, Z., Wu, M., Zhao, R., Guretno, F., Yan, R., & Li, X. (2021). Machine remaining useful life prediction via an attention-based deep learning approach. *IEEE Transactions on Industrial Electronics*, *68*, 2521–2531. https://doi.org/10.1109/TIE.2020.2972443

Zhang, J., Jiang, Y., Wu, S., Li, X., Luo, H., & Yin, S. (2022). Prediction of remaining useful life based on bidirectional gated recurrent unit with temporal self-attention mechanism. *Reliability Engineering & System Safety*, *221*, 108297. https://doi.org/10.1016/j.ress.2021.108297

Nie, L., Zhang, L., Xu, S., Cai, W., & Yang, H. (2022). Remaining useful life prediction of milling cutters based on cnn-bilstm and attention mechanism. *Symmetry 2022, Vol. 14, Page 2243*, *14*, 2243. https://doi.org/10.3390/SYM14112243

Qin, Y., Cai, N., Gao, C., Zhang, Y., Cheng, Y., & Chen, X. (2022). Remaining useful life prediction using temporal deep degradation network for complex machinery with attention-based feature extraction. *ArXiv*. https://doi.org/10.48550/arxiv.2202.10916

Remadna, I., Terrissa, L. S., Masry, Z. A., & Zerhouni, N. (2022). Rul prediction using a fusion of attention-based convolutional variational autoencoder and ensemble learning classifier. *IEEE Transactions on Reliability*. https://doi.org/10.1109/TR.2022.3190639

Liu, H., Liu, Z., Jia, W., & Lin, X. (2021). Remaining useful life prediction using a novel feature-attention-based end-to-end approach. *IEEE Transactions on Industrial Informatics*, *17*, 1197–1207. https://doi.org/10.1109/TII.2020.2983760

Xu, D., Qiu, H., Gao, L., Yang, Z., & Wang, D. (2022). A novel dual-stream self-attention neural network for remaining useful life estimation of mechanical systems. *Reliability Engineering & System Safety*, *222*, 108444. https://doi.org/10.1016/j.ress.2022.108444

Zhai, S., Gehring, B., & Reinhart, G. (2021). Enabling predictive maintenance integrated production scheduling by operation-specific health prognostics with generative deep learning. *Journal of Manufacturing Systems*. https://doi.org/10.1016/J.JMSY.2021.02.006

Que, Z. J., Xiong, Y., & Xu, Z. G. (2019). A semi-supervised approach for steam turbine health prognostics based on gan and pf. *IEEE International Conference on Industrial Engineering and Engineering Management*, 1476–1480. https://doi.org/10.1109/IEEM44572.2019.8978717

Yang, T., Wang, J., Chen, L., Cui, Z., Qi, J., & Wang, R. (2020). Machinery health prognostics of dust removal fan data through deep neural networks. In H. Ning & F. Shi (Eds.), *Cyberspace data and intelligence, and cyber-living, syndrome, and health* (pp. 27–37). Springer Singapore. https://doi.org/10.1007/978-981-33-4336-8_3

Eker, O. F., Camci, F., & Jennions, I. K. (2012). *Major Challenges in Prognostics: Study on Benchmarking Prognostics Datasets* (tech. rep.). PHM Society. http://dspace.lib.cranfield.ac.uk/handle/1826/9994

Su, C., Li, L., & Wen, Z. (2020). Remaining useful life prediction via a variational autoencoder and a time-window-based sequence neural network. *Quality and Reliability Engineering International*, *36*, 1639–1656. https://doi.org/10.1002/QRE.2651

Yu, K., Wang, D., & Li, H. (2021). A prediction model for remaining useful life of turbofan engines by fusing broad learning system and temporal convolutional network. *2021 International Conference on Information, Cybernetics, and Computational Social Systems, ICCSS 2021*, 137–142. https://doi.org/10.1109/ICCSS53909.2021.9722026

Wang, T., Guo, D., & Sun, X. (2021). A new paralleled semi-supervised deep learning method for remaining useful life prediction. In F. Sun, H. Liu, & B. Fang (Eds.), *Cognitive systems and signal processing* (pp. 219–226). Springer Singapore.

Martin, G. S., & Droguett, E. L. (2022). Temporal variational auto-encoders for semi-supervised remaining useful life and fault diagnosis. *IEEE Access*, *10*, 55112–55125. https://doi.org/10.1109/ACCESS.2022.3174860

Verstraete, D., Droguett, E., & Modarres, M. (2019). A Deep Adversarial Approach Based on Multi-Sensor Fusion for Semi-Supervised Remaining Useful Life Prognostics. *Sensors*, *20*(1), 176. https://doi.org/10.3390/s20010176

Hu, Y., & Luo, P. (2012). Performance Data Prognostics Based on Relevance Vector Machine and Particle Filter. *Ephm*, *33*, 349–354. https://doi.org/10.3303/CET1333059

Cheng, F., Qu, L., & Qiao, W. (2018). Fault prognosis and remaining useful life prediction of wind turbine gearboxes using current signal analysis. *IEEE Transactions on Sustainable Energy*, *9*(1), 157–167. https://doi.org/10.1109/TSTE.2017.2719626

Cheng, F., Qu, L., Qiao, W., & Hao, L. (2019). Enhanced Particle Filtering for Bearing Remaining Useful Life Prediction of Wind Turbine Drivetrain Gearboxes. *IEEE Transactions on Industrial Electronics*, *66*(6), 4738–4748. https://doi.org/10.1109/TIE.2018.2866057

Bai, G., & Wang, P. (2016). Prognostics Using an Adaptive Self-Cognizant Dynamic System Approach. *IEEE Transactions on Reliability*, *65*(3), 1427–1437. https://doi.org/10.1109/TR.2016.2570542

Tse, Y. L., Cholette, M. E., & Tse, P. W. (2019). A multi-sensor approach to remaining useful life estimation for a slurry pump. *Measurement: Journal of the International Measurement Confederation*, *139*, 140–151. https://doi.org/10.1016/j.measurement.2019.02.079

Xu, Z., Guo, Y., & Saleh, J. (2021). Accurate Remaining Useful Life Prediction With Uncertainty Quantification: A Deep Learning and Nonstationary Gaussian Process Approach. *IEEE Transactions on Reliability*. https://doi.org/10.1109/TR.2021.3124944

Deutsch, J., He, M., & He, D. (2017). Remaining Useful Life Prediction of Hybrid Ceramic Bearings Using an Integrated Deep Learning and Particle Filter Approach. *Applied Sciences*, *7*(7), 649. https://doi.org/10.3390/app7070649

Xu, M., Baraldi, P., Al-Dahidi, S., & Zio, E. (2020). Fault prognostics by an ensemble of Echo State Networks in presence of event based measurements. *Engineering Applications of Artificial Intelligence*, *87*(October 2019), 103346. https://doi.org/10.1016/j.engappai.2019.103346

Gal, Y., & Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd international conference on machine learning* (pp. 1050–1059). PMLR. https://proceedings.mlr.press/v48/gal16.html

Mitici, M., de Pater, I., Barros, A., & Zeng, Z. (2023). Dynamic predictive maintenance for multiple components using data-driven probabilistic rul prognostics: The case of turbofan engines. *Reliability Engineering & System Safety*, *234*, 109199. https://doi.org/https://doi.org/10.1016/j.ress.2023.109199

Lee, J., & Mitici, M. (2023). Deep reinforcement learning for predictive aircraft maintenance using probabilistic remaining-useful-life prognostics. *Reliability Engineering & System Safety*, *230*, 108908. https://doi.org/https://doi.org/10.1016/j.ress.2022.108908

Rigamonti, M., Baraldi, P., Zio, E., Roychoudhury, I., Goebel, K., & Poll, S. (2018). Ensemble of optimized echo state networks for remaining useful life prediction. *Neurocomputing*. https://doi.org/10.1016/j.neucom.2017.11.062

Simon, D. (2006). *Optimal state estimation: Kalman, h infinity, and nonlinear approaches*. Wiley. https://books.google.com.au/books?id=UiMVoP%5C_7TZkC

Särkkä, S. (2013). *Bayesian filtering and smoothing*. Cambridge University Press. https://doi.org/10.1017/CBO9781139344203

Li, A. H., Wu, P., & Kennedy, M. (2021). Replay overshooting: Learning stochastic latent dynamics with the extended kalman filter. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, *2021-May*, 852–858. https://doi.org/10.1109/ICRA48506.2021.9560811

de Bézenac, E., Rangapuram, S. S., Benidis, K., Bohlke-Schneider, M., Kurle, R., Stella, L., Hasson, H., Gallinari, P., & Januschowski, T. (2020). Normalizing kalman filters for multivariate time series analysis. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (pp. 2995–3007). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2020/file/1f47cef5e38c952f94c5d61726027439-Paper.pdf

Burda, Y., Grosse, R. B., & Salakhutdinov, R. (2016). Importance weighted autoencoders. *CoRR*, *abs/1509.00519*.

Ishizone, T., Higuchi, T., & Nakamura, K. (2020). Ensemble kalman variational objectives: Nonlinear latent trajectory inference with a hybrid of variational inference and ensemble kalman filter. https://doi.org/10.48550/arxiv.2010.08729

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018). Neural Ordinary Differential Equations. (NeurIPS). http://arxiv.org/abs/1806.07366

Li, X., Wong, T.-K. L., Chen, R. T. Q., & Duvenaud, D. (2020). Scalable gradients for stochastic differential equations. In S. Chiappa & R. Calandra (Eds.), *Proceedings of the twenty third international conference on artificial intelligence and statistics* (pp. 3870–3882). PMLR. https://proceedings.mlr.press/v108/li20i.html

Star, M., & McKee, K. (2021). Remaining useful life estimation using neural ordinary differential equations. *International Journal of Prognostics and Health Management*, *12*, 1–15. https://doi.org/10.36001/IJPHM.2021.V12I2.2938

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Saxena, A., Goebel, K., Simon, D., & Eklund, N. (2008). Damage propagation modeling for aircraft engine run-to-failure simulation. *2008 International Conference on Prognostics and Health Management*, *41*, 1–9. https://doi.org/10.1109/PHM.2008.4711414

Pasa, G., Medeiros, I., & Yoneyama, T. (2019). Operating Condition-Invariant Neural Network-based Prognostics Methods applied on Turbofan Aircraft Engines. *Annual Conference of the PHM Society*, *11*(1). https://doi.org/https://doi.org/10.36001/phmconf.2019.v11i1.786

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., & Bakshy, E. (2019). BoTorch: Programmable Bayesian Optimization in PyTorch. http://arxiv.org/abs/1910.06403

Frazier, P. I. (2018). A tutorial on bayesian optimization.

Queiruga, A. F., Erichson, N. B., Taylor, D., & Mahoney, M. W. (2020). Continuous-in-depth neural networks. http://arxiv.org/abs/2008.02389

Si, X. S., Wang, W., Hu, C. H., & Zhou, D. H. (2011). Remaining useful life estimation - A review on the statistical data driven approaches. *European Journal of Operational Research*, *213*(1), 1–14. https://doi.org/10.1016/j.ejor.2010.11.018

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., Facebook, Z. D., Research, A. I., Lin, Z., Desmaison, A., Antiga, L., Srl, O., & Lerer, A. (2017). *Automatic differentiation in PyTorch* (tech. rep.).

Costa, N., & Sánchez, L. (2021). Remaining useful life estimation using a recurrent variational autoencoder. In H. Sanjurjo González, I. Pastor López, P. García Bringas, H. Quintián, & E. Corchado (Eds.), *Hybrid artificial intelligent systems* (pp. 53–64). Springer International Publishing.

Krokotsch, T., Knaak, M., & Gühmann, C. (2022). Improving semi-supervised learning for remaining useful lifetime estimation through self-supervision. *International Journal of Prognostics and Health Management*, *13*. https://doi.org/10.36001/IJPHM.2022.V13I1.3096

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 6000–6010.

Sarkka, S., Solin, A., & Hartikainen, J. (2013). Spatiotemporal learning via infinite-dimensional bayesian filtering and smoothing: A look at gaussian process regression through kalman filtering. *IEEE Signal Processing Magazine*, *30*, 51–61. https://doi.org/10.1109/MSP.2013.2246292

van Engelen, J. E., & Hoos, H. H. (2020). A survey on semi-supervised learning. *Machine Learning*, *109*, 373–440. https://doi.org/10.1007/S10994-019-05855-6/FIGURES/5

Higgins, I., Matthey, L., Pal, A., Burgess, C. P., Glorot, X., Botvinick, M. M., Mohamed, S., & Lerchner, A. (2017). Beta-vae: Learning basic visual concepts with a constrained variational framework. *ICLR*, 1–22.

Dupont, E., Doucet, A., & Teh, Y. W. (2019). Augmented neural odes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (pp. 1–20). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2019/file/21be9a4bd4f81549a9d1d241981cec3c-Paper.pdf

Finlay, C., Jacobsen, J., Nurbekyan, L., & Oberman, A. M. (2020). How to train your neural ODE: The world of Jacobian and Kinetic regularization. *CoRR*, *abs/2002.02798*. https://arxiv.org/abs/2002.02798

Särkkä, S., & Garcia-Fernandez, A. F. (2021). Temporal parallelization of bayesian smoothers. *IEEE Transactions on Automatic Control*, *66*(1), 299–306. https://doi.org/10.1109/TAC.2020.2976316

Lai, J., Domke, J., & Sheldon, D. (2021). Variational marginal particle filters. *Proceedings of The International Conference on Artificial Intelligence and Statistics (AISTATS)*. https://par.nsf.gov/biblio/10359646

Every reasonable effort has been made to acknowledge the owners of copyright material. I would be pleased to hear from any copyright owner who has been omitted or incorrectly acknowledged