Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

R&D Project

Generative Models for the Analysis of Dynamical Systems with Applications

Alan Preciado Grijalva

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr. Paul G. Ploeger
Dr. Rodrigo Iza-Teran

August 2020

https://doi.org/10.18418/opus-6041

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work.

Date	Alan Preciado Grijalva

Abstract

High-dimensional and multi-variate data from dynamical systems such as turbulent flows and wind turbines can be analyzed with deep learning due to its capacity to learn representations in lower-dimensional manifolds. Two challenges of interest arise from data generated from these systems, namely, how to anticipate wind turbine failures and how to better understand air flow through car ventilation systems.

There are deep neural network architectures that can project data into a lower-dimensional space with the goal of identifying and understanding patterns that are not distinguishable in the original dimensional space. Learning data representations in lower dimensions via non-linear mappings allows one to perform data compression, data clustering (for anomaly detection), data reconstruction and synthetic data generation.

In this work, we explore the potential that variational autoencoders (VAE) have to learn low-dimensional data representations in order to tackle the problems posed by the two dynamical systems mentioned above. A VAE is a neural network architecture that combines the mechanisms of the standard autoencoder and variational bayes. The goal here is to train a neural network to minimize a loss function defined by a reconstruction term together with a variational term defined as a Kulback-Leibler (KL) divergence.

The report discusses the results obtained for the two different data domains: wind turbine time series and turbulence data from computational fluid dynamics (CFD) simulations.

We report on the reconstruction, clustering and unsupervised anomaly detection of wind turbine multi-variate time series data using a variant of a VAE called Variational Recurrent Autoencoder (VRAE). We trained a VRAE to cluster normal and abnormal wind turbine series (two class problem) as well as normal and multiple abnormal series (multi-class problem). We found that the model is capable of distinguishing between normal and abnormal cases by reducing the dimensionality of the input data and projecting it to two dimensions using techniques such as Principal Component Analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE). A set of anomaly scoring methods is applied on top of these latent vectors in order to compute unsupervised clustering. We have achieved an accuracy of up to 96% with the $KMeans++$ algorithm.

We also report the data reconstruction and generation results of two dimensional turbulence slices corresponding to CFD simulation of a HVAC air duct. For this, we have trained a Convolutional Variational Autoencoder (CVAE). We have found that the model is capable of reconstructing laminar flows up to a certain degree of resolution as well generating synthetic turbulence data from the learned latent distribution.

Acknowledgements

I want to thank Prof. Dr. Jochen Garcke for accepting me to work in his group at Fraunhofer SCAI. I thank Dr. Victor Rodrigo Iza Teran for his technical guidance on the wind turbines project. Special thanks to Dr. Ivan Lecei for providing the wind turbine data that was used in this project. Thanks to M. Sc. Christian for his technical guidance on the turbulence project and providing the first data preprocessing steps. I also want to thank Prof. Dr. Paul Ploger for providing useful observations that made this work better. Lastly, thanks to Fraunhofer SCAI for providing the necessary equipment to perform the computations.

Contents

List of Figures

List of Tables

1

Introduction

Dynamical systems theory studies the state evolution of a system in time. A state at time t corresponds to the instantaneous description of the system in terms of its physical variables. In this theory, such a description is sufficient to predict the future states of the system without having to look at prior ones.

Examples of dynamical systems are the Lorentz oscillator and Atwood's machine. Turbulence phenomena can also be analyzed from the dynamical systems perspective [28]. A principal characteristic of most dynamical systems is that they can not easily be predicted or diagnosed. The nature of most phenomena is chaotic, this means that the state evolution of a system is driven by factors that are difficult to quantify and put into equations. To some degree, equations that govern the behavior of these systems can be written, but very often these equations are too complex and depend on specific state observations, causing the accuracy of long-term predictions to become untractable.

In the case of turbulence, Computational Fluid Dynamics (CFD) is the field that mathematically models such phenomena. Here, turbulence is defined as fluid motion characterized by chaotic changes in pressure and velocity. The most common turbulence models in CFD are Large Eddy Simulation (LES) and Direct Numerical Simulation (DNS).

Progress in computational capacities has allowed the resolution of the data generated by CFD models to increase. Higher resolution resolved fluids means that the volumes of data generated are bigger. Therefore, the post-processing (e.g. storage) and analysis of this data with novel data-driven techniques is of general interest.

Another type of dynamical system that produces vast amounts of data are wind turbines [10]. Here, sensors monitor different physical variables that give a precise diagnosis of the system, for example, the acceleration of the turbine blades as a function of time. It is of particular interest in engineering to study the freezing of the blades, this because harsh weather conditions can affect the optimal performance of the turbines. Similarly to CFD data, it is of great interest to analyze the data collected from these systems in the form of high-dimensional, multi-variate time series.

To analyze and gain meaningful insights from the data generated by systems such as turbulent flows and wind turbines, certain data processing tasks are crucial, these include data **compression**, **clustering**, **reconstruction** and **generation**.

Data compression, for example, consists of reducing the dimensionality of input data in order to represent it in a lower-dimensional space by encoding information using less bits compared to the original

representation. The main objective of compression is to be able to reconstruct original data with a high degree of resolution. There have been encoding-decoding frameworks that have shown that it is possible to reconstruct a high-resolution flow field from a massively under-resolved turbulent flow field using machine learning techniques [6].

Data clustering, in contrast, makes use of this encoded representation in order to visually identify patterns/grouping in two or three dimensions. This is very important since numerous physical spatio-temporal insights can be derived from this; structure (clustering) in low-dimensional spaces means that high-dimensional data has (un)correlated attributes. From this, one can perform anomaly detection of time series as reported in [25].

With data and novel machine learning models at hand, it is reasonable to explore its potential to tackle the problem domains mentioned above (i.e. turbulent flow and wind turbine data). In this work we focus in particular deep learning implementations.

Deep learning methods have been improving the benchmarks in fields like computer vision, robotic control and natural language processing. The main reasons behind this, are improvements in neural networks architectures, greater computational power and data availability [21].

In the case of data compression, deep neural networks have proven to be efficient methods for non-linear dimensionality reduction. *Hinton et al. (2006)* [13] propose an architecture based on Restricted Boltzmann Machines (RBM) capable of encoding and decoding data. Such autoencoding procedure allowed the authors to learn structures in lower-dimensions more effectively than the classical Principal Component Analysis (PCA).

In more detail, the architecture that *Hinton et al. (2006)* introduced is known as autoencoder. The autoencoder is a multi-layered bottleneck neural network that encodes data into lower dimensions in a self-supervised way. The loss penalty is a reconstruction term that compares the input to the reconstructed output. An autoencoder can be defined either as a Convolutional Neural Network (CNN) or Recurrent Neural Network (RNN) architecture, this depends on the task at hand (e.g. image processing or time series analysis).

The capabilities of autoencoders were expanded further by *Kingma et al. (2013)* [19] by introducing variational inference into the architecture. These changes allowed autoencoders to learn not only single point mappings but complete latent data distributions making them generative models. The proposed autoencoder architecture is called Variational Autoencoder (VAE) and it has the capacity to learn disentangled latent data distributions that provide a precise statistical representation of the data.

VAEs have achieved state-of-the-art results in benchmark datasets in semi-supervised learning in image generation [15], clustering and anomaly detection [25] [26] [34], and data reconstruction [32]. The main reason for this is that the latent space learned here is continuous (probability distributions), thereby allowing straightforward random sampling and interpolation. The next chapter contains a more in-depth discussion of the state of the art using this model.

Based on these recent success achieved by VAEs, we have taken them as the core model for the research presented in this report. This report is an exploration of several VAE architectures applied to our two data domains: CFD turbulence data and wind turbine time series data. Next we describe them more in

detail and the problems trying to be solved.

For the case of wind turbines, the general goal is to build an adaptive controller using **predictive maintenance**. Such scheme should ideally be an intelligent and integrated one. The scheme is an iterative cycle that consists of three modules: **1)** data collection sensor module, **2)** predictive maintenance module and **3)** a controller module. In this work, we will focus in module two: predictive maintenance by performing anomaly detection of wind turbine multi-variate time series.

Figure 1.1: 3D rendered diagram of a wind turbine.

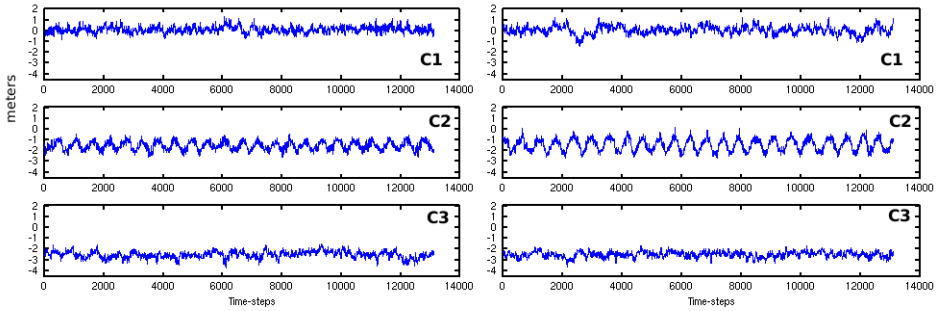


Figure 1.2: Wind turbine sensor data collection.

Figure 1.2 shows an example of sensors monitoring the state of a wind turbine as a function of time. Each sensor monitors a particular physical variable, for example, the acceleration along the edge of a blade. Under optimal conditions, wind turbines are capable of outputing the expected energy, however, it is often the case that weather conditions such as ice accumulation on the blades cause malfunctions. An early identification of this ice mass is a core problem indeed, and several anomaly detection and fault analysis approaches have been explored [7] [34]. In this context, anomaly detection is framed as identifying (predicting) when is it that a wind turbine is operating sub-optimally versus operating optimally.

Here, we report on the implementation of an unsupervised learning method for anomaly detection of

wind turbine simulation data. The pipeline consists of two main steps: **(1)** learning abstract time series data representations using a Variational Recurrent Autoencoder (VRAE) and **(2)** perform anomaly scoring (classification) using these learned representations. In the first module, a VRAE processes multi-variate time series inputs to reduce their dimensionality. This abstract representation is projected into two dimensions via PCA and t-SNE in order to visually evalute it. In the second module, we perform clustering methods (k-means++, DBSCAN, etc.) on top of the projected latent vectors to compute a classification accuracy based on the predicted classes (1 for "normal operating condition" and 2 for "abnormal operating condition").

Great emphasis has been put in module one; unsupervised representation learning of time series, since the core goal here is to cluster data adequately in lower dimensions by fine-tunning the neural network architecture. This procedure is similar to the one reported by *Pereira et al (2018)* [25] [26], one of the main differences is that we are using our own simulated data (no benchmark dataset).

For the case of turbulence data, we analyze the extracted slices from a heating, ventilation, and air conditioning (HVAC) duct that are produced from numerical simulations by our research group. The figure below shows a temporal evolution of a typical turbulent flow. As we can see, each turbulence slide contains spatial and temporal information of the physics of the flow (such as velocity, velocity gradients and pressure). The relevant part of the solution of the flow depends on boundary conditions and main simulation parameters (viscosity, time step, total kinetic energy, Reynolds number, etc.).

CFD simulation data is vast due to the high resolution needed to model a flow reallistically. Our specific goal in this domain is to perform compression and reconstruction on turbulent flow data. This helps reducing resources for storage and handling of this data, moreover, this could also help understanding airflow of HVAC systems to localize noise sources by decomposing it in more interpretable modes. Also, we will explore the feasibility of our model to generate synthetic turbulent flow slices based on the learned probability distribution of the data. For these tasks, we will be training a Convolutional Variational Autoencoder (CVAE) on turbulent velocity fields in order to improve reconstruction and generation performance. We are using convolutions given that the problem is being framed as 2D slices of nxm pixel images.

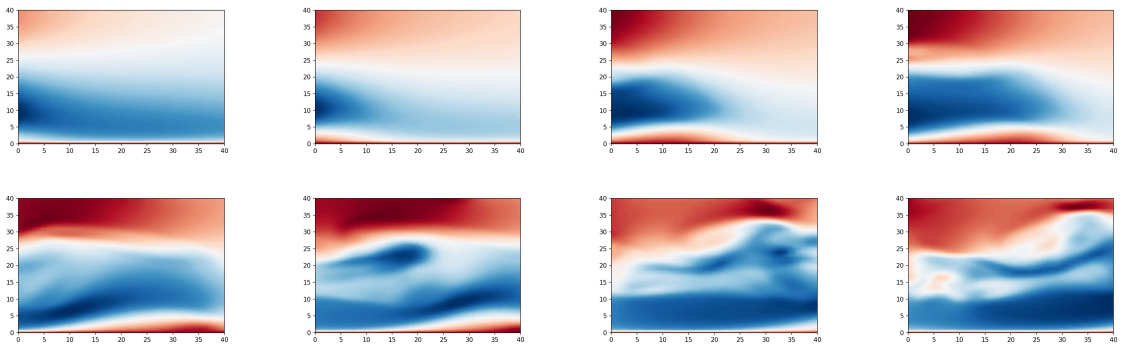


Figure 1.3: Temporal evolution of a turbulent flow (from left to right, upper to bottom rows).

The following subsections highlight as bullet lists the main points discussed above.

1.1 Motivation

- Develop a module for robust predictive maintenance of wind turbines by implementing an anomaly detection pipeline that implements a Variational Recurrent Autoencoder to do unsupervised representation learning of multi-variate time series
- Explore and understand the abstract learned representations of wind turbine time series by evaluating their clustering behavior in lower dimensions
- Study the reconstruction of time series and clustering of multiple time series classes
- Evalute unsupervised clustering with simulation data (no benchmark datasets) using state of the art neural network architectures
- For CFD data, it is important to characterize turbulence through decomposition into more interpretable modes by performing compression and reconstruction
- Compression-reconstruction of high-resolution flow data can save storage space given the vast amount of data generated by CFD methods
- Explore the capacities of convolutional variational autoencoders to reconstruct with high-resolution 2D turbulence slides
- Synthetic turbulence data generation by training a convolutional variational autoencoder is still a novel challenge being researched and has the potential to be an alternative to other methods that are computationally more expensive
- Investigate the benefits of incorporating more physical knowledge into the loss function of convolutional variational autoencoders in order to frame the problem as physics-informed machine learning

1.2 Challenges and Difficulties

- Understanding at a conceptual and practical level how to train a VAE in order to learn unsupervised representations of time series and turbulence data
- Understand and overcome the challenges imposed in the pre-processing of time series data in order to learn efficient representations such as: number of time steps per sample, number of features, data normalization methods, defining what constitutes ”normal” versus ”abnormal” cases in time series
- Fine-tune the hyperparameters of the model in order to cluster time series given two classes and multiple-classes
- Interpreting the latent space of our projected time series data and the model parameters that give the best clsutering such the dimensionality of these latent vectors

- A wind turbine time series dataset must have an adequate of "anomalous" cases (abnormal) and "non-anomalous" cases (normal), but this depends on the data available after simulations
- Training a VAE on turbulent data requires important data pre-processing steps similar to images, in this case, the physical nature of the data must always be taken into account, therefore, cropping might not make sense since we are loosing important information
- Improve the reconstruction performance of turbulence data at a high resolution is a challenge due to the level of detail this data contains, also, a method to quantify reconstruction is desirable
- A physics informed machine learning approach would consist in creating a customized loss function
- There is not a lot work published on turbulence reconstruction-generation
- The chaotic nature of flow data: there is possibly no adequate or interpretable latent space

1.3 Problem Statement

In this project the problem of compression, reconstruction, clustering and generation of multi-variate time series and turbulence data using variational autoencoders is explored. The systems under consideration are wind turbine time series and turbulent flow sections across HVAC ducts.

For wind turbine systems, we will explore the capacities of variational recurrent autoencoders to learn unsupervised representations of simulated time series. The goal is to cluster time series efficiently in order to implement clustering methods on top of these learned representations. This procedure would ideally be a robust time series anomaly detection by identifying normal versus abnormal cases in the context of ice mass accumulation in the blades of the turbines.

In the case of turbulence data, we will implement a convolutional variational autoencoder to perform compression and reconstruction of flows. We will also inquire in the capabilities of the model to cluster flows in lower dimensions and also analyze the synthetic data generated by the model given the learned latent probability distribution. The data derives from simulations of turbulent HVAC systems.

1.4 Sections outlook

This report is structure as follows: **Chapter 2** contains a discussion about previous work in the wind turbines and turbulence domains from a deep learning approach. It contains a description of the neural network architectures studied in this report, namely, Variational Autoencoders. The chapter also discusses the state of the art results on anomaly detection using Variational Autoencoders on time series benchmark datasets as well as the most up-to-date generation and high-resolution reconstruction results on 2D turbluence data. **Chapter 3** contains a detailed description of the datasets we worked with and also the main pre-processing data steps we implemented. **Chapter 4** contains the results obtained on time series compression, clustering and anomaly detection using Variational Autoencoders. Similarly, we detailed the reconstruction and generation results obtained on a HVAC duct turbluence dataset. **Chapter 5** summarizes with the contributions, lessons learned and future work.

2

State of the Art

This section covers related work in the field of high-resolution turbulent flows reconstruction using machine learning techniques and previous work on unsupervised anomaly detection of time series via clustering. First, we provide a discussion of the backbone model in our research; the variational autoencoder. Secondly, we introduce the specific models used for wind turbines; variational recurrent autoencoder and for tubulence data; convolutional variational autoencoder. Third, we describe work that has been done using machine learning to reconstruct DNS turbulence data and proof-of-concept implementations of models to generate synthetic turbulence data. Thereafter, we present a discussion on time series clustering on a benchmark dataset using variational recurrent autoencoders.

2.1 Autoencoder

An autoencoder is a neural network that aims to reconstruct given input data in a self-supervised way. It learns to *encode* and *decode* its inputs by using an encoder and a decoder neural network. The encoder maps input data $\mathbf{x} \in \mathbb{R}^{d_x}$ into a latent representation $\mathbf{z} \in \mathbb{R}^{d_z}$ and the decoder reconstructs latent representations into the input dimensions. This network is trained with a loss function that compares inputs $\mathbf{x}$ vs reconstructions $\hat{\mathbf{x}}$ and tries to make them as equal as possible.

Generally, autoencoders encode data in lower dimensions and have proven to be better dimensionality reduction methods than several standard methods. The figure below shows a schematic diagram of a standard multilayered autoencoder architecture. From left to right, the first layers takes in original data, the second layer projects data into five dimensions, the third layer acts as a bottleneck layer projecting into three dimensional latent representations. The next part corresponds to the decoder which performs the inverse operations as the encoder in order to reconstruct the data.

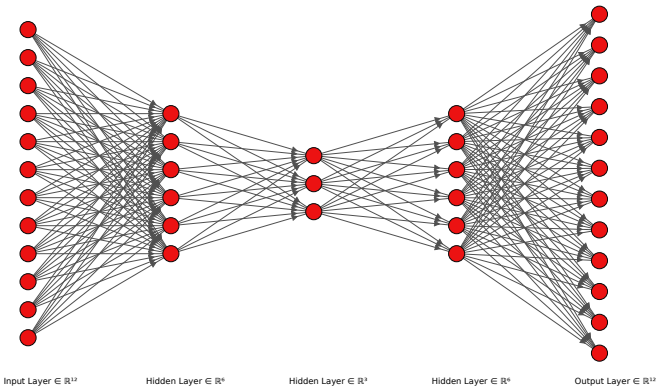


Figure 2.1: Standard autoencoder neural network architecture.

Introduced first by *Hinton et. al (2006)* [13], autoencoders proved to be better at compressing images better than PCA. The architecture presented by *Hinton et. al (2006)* used Restricted Boltzmann Machines (RBM) in its layers to encode-decode image data. The figure below shows the proposed architecture. The first part consists of pre-training the RBMs separately. The unrolling part connects these trained parts to form the autoencoder. Lastly, the autoencoder is fine-tuned using backpropagation.

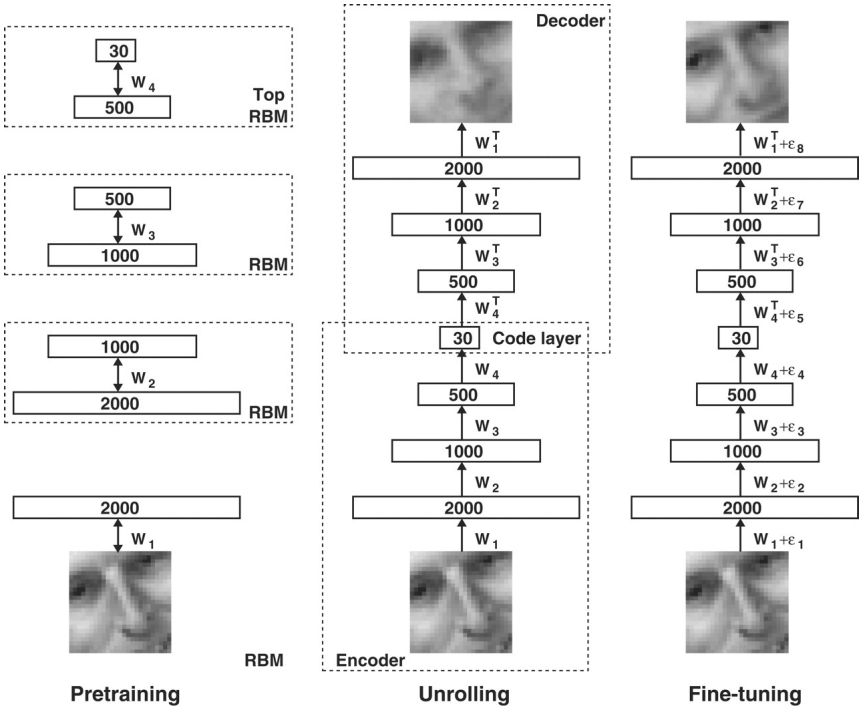


Figure 2.2: Multi-layered autencoder architecture using Restricted Boltzmann Machines. Image obtained from *Hinton et. al, 2006* [13, p. 505].

2.2 Variational Autoencoder

Variational autoencoders (VAEs) [*Kingma et. al (2013)* [19]] are neural networks that make use of Bayesian variational inference in order to learn low-dimensional latent and structured representations of high-dimensional data. Variational Bayes methods are used to give an analytical approximation to the **posterior probability of unobserved variables**, thus allowing statistical inference over such variables to be performed. They are also used to provide lower bounds of the marginal likelihood of the observed data, this helps in model selection. VAEs thus, are an extension of autoencoders, the added capability is that they also learn distributions in the latent space (the bottleneck layer of autoencoders). It follows that VAEs are generative models since the decoder can sample randomly from the learned latent distributions and decode them.

More formally, let $\mathbf{x} \in \mathbb{R}^{d_x}$ be the input data vector of a VAE. This vector is then mapped into a latent vector $\mathbf{z} \in \mathbb{R}^{d_x}$ $(d_z \leq d_x)$ by the encoder network $q_\psi(\mathbf{z}|\mathbf{x})$, then, the decoder network $p_\phi(\mathbf{x}|\mathbf{z})$ takes this latent vector to create a reconstruction $\hat{\mathbf{x}}$. When training VAEs, we attempt to make the reconstruction $\hat{\mathbf{x}}$ as close as the original input $\mathbf{x}$ plus the constraint that the learned latent distribution (whose vectors are of dimensionality d_z) must be appropriately regularized. The loss function introduced by *Kingma et. al (2013)* used to train VAEs is the following one

$$\mathcal{L}(\psi, \phi) = \mathbb{E}_{x \sim p_{data}(x)} \left[-\mathbb{E}_{z \sim q_\psi(z|x)}[log p_\phi(x|z)] + D_{KL}(q_\psi(z|x) \| p(z)) \right] \tag{2.1}$$

The first term in the above equation is the reconstruction term that aims to train the decoder network $p_\phi(\mathbf{x}|\mathbf{z})$ to match target versus reconstructions by maximizing the log-likelihood of the data. The second term, on the other hand, is the Kullback-Leibler (KL) divergence ($D_{KL}(p\|q)$) defined as a measure of distance between probability distributions. $D_{KL}(p\|q) = \int plog\left(\frac{p}{q}\right)$ where q is the encoder network $q_\psi(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{z})$ is a prior distribution.

The figure below shows a schematic diagram of a standard VAE that processes MNIST images. The first part is the encoder network $q_\psi(\mathbf{z}|\mathbf{x})$ which attempts to map input vectors into low-dimensional latent vectors while at the same time learning a latent distribution capable of capturing the statistics of the dataset. This latent distribution is parametrized by the mean μ and standard deviation σ of the bottleneck layer. The second part is the decoder which reconstructs the inputs from the latent vectors.

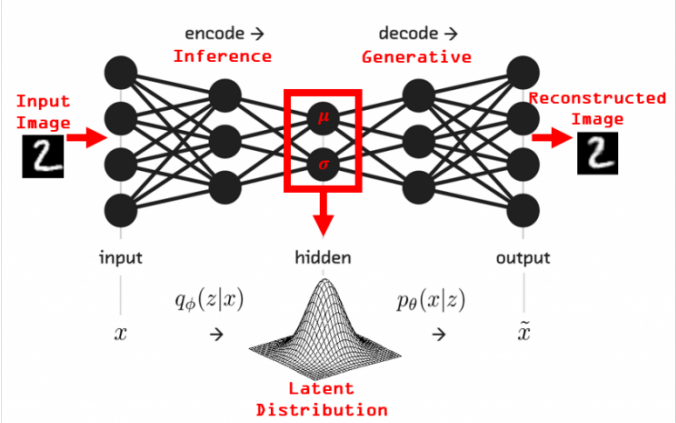


Figure 2.3: Variational autoencoder standard architecture. Image obtained from Sergios Karagiannakos-AI Summer [17].

A main assumption when training VAEs is that the prior distribution $p(\mathbf{z})$ that the encoder tries to approximate is a standard normal distribution ($\mathcal{N}(\mu, \sigma^2)$). Note that the latent vectors $\mathbf{z}$ are random variables distributed according to the prior $p(\mathbf{z})$ [19]. Since a true posterior $p_\psi(\mathbf{z}|\mathbf{x})$ is intractable, it has to be parametrized by the encoder $q_\psi(\mathbf{z}|\mathbf{x})$ which is a Gaussian distribution whose mean and standard deviation are computed in the bottleneck layer. In order to computer these parameters, *Kingma et. al (2013)* [19] introduced the re-parametrization trick (see figure below). The idea of doing this is to avoid differentiation through a stochastic node by rewriting the parametrized Gaussian as $\mathcal{N}(\mu, \sigma^2) = \mu + \sigma\mathcal{N}(0, 1)$. Modelling the latent variables in this way allows the KL divergence to be integrated analytically.

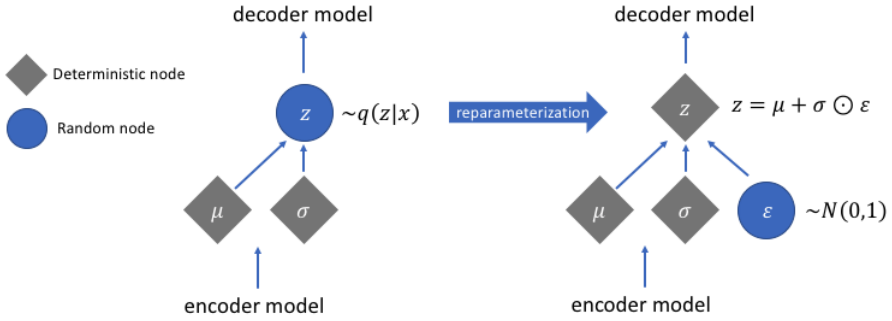


Figure 2.4: Reparametrization trick used when training VAEs. Image obtained from [16].

The role of the KL divergence in the loss function is to prioritize latent distributions that match the prior. The figure below shows the effect of both terms in the loss function when projecting MNIST data in the two dimensional plane. The left-most figure shows the latent space trained only with reconstruction loss, as we can see, it is capable of splitting data based on their digit class, however, there are spaces

on the plane that have no meaning if we look at it from a probabilistic perspective. The middle image shows the latent space only with the KL regularization term, note that the latent space has a normal-like distribution but no distinction between digits. The right-most image is the combination of both loss terms and shows that the model is indeed being able to disentangle digit classes while learning structured normal distributions.

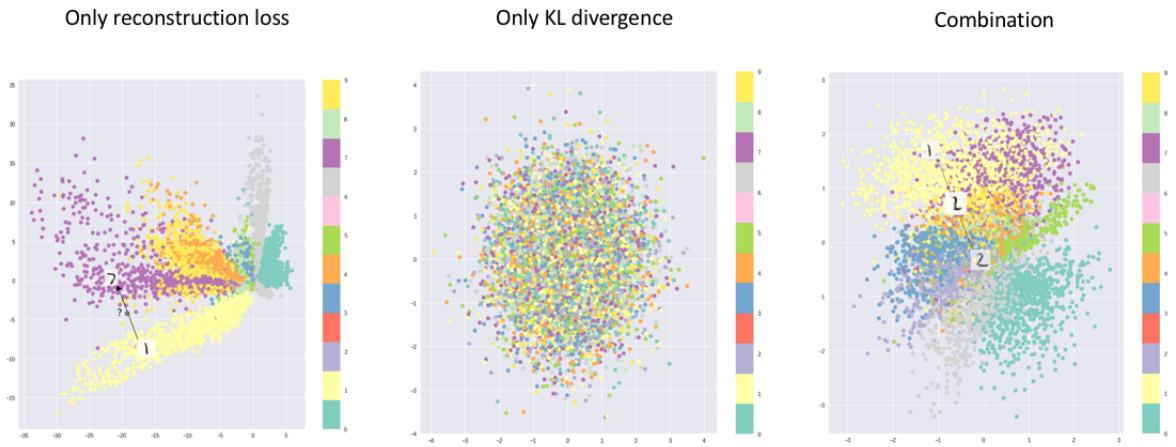


Figure 2.5: Effect of loss terms in latent space regularization of the MNIST VAE. Image obtained from [16].

VAEs have had several improvements recently such is the case of β-VAE [12], which is a model capable of learning interpretable and disentangled representations of data. VAEs stand out against generative adversarial networks (GANs) [8] in that they are easier to implement and more robust during training. Moreover, VAEs have achieved several state of the art results in image processing and language processing applications.

2.2.1 Variational Recurrent Autoencoder

Recurrent Neural Networks (RNNs) are models capable of capturing time dependencies in data like time series [14]. It is their gated mechanisms that allow them to learn long-term temporal relationships. An example of the success of RNNs is in machine translation, where encoder-decoder architectures have set the state of the art [2]. It has also been shown that the strengths of RNNs can be combined with VAEs in order to create the Variational Recurrent Autoencoder [*Fabius et. al (2015)* [5]]. This model allows to map time sequences to latent representations and enables optimized and large scale unsupervised variational learning on time sequences. This model at its core is a VAE that implements RNNs in its layers. It contains an encoder-decoder scheme where both are separate networks that have a set of recurrent connections such that the state h_{t+1} is calculated on the previous state h_t and on the data x_{t+1} of corresponding time step. The distribution over the latent random variable $\mathbf{z}$ is obtained from the **last** state of the RNN, h_{end} in the following manner

$$
\begin{aligned}
h_{t+1} &= tanh(W_{enc}^T h_t + W_{in}^T x_{t+1} + b_{enc}) \\
\mu_z &= W_\mu^T h_{end} + b_\mu \\
log(\sigma_z) &= W_\sigma^T h_{end} + b_\sigma
\end{aligned}
\tag{2.2}
$$

Here, W_{enc} and W_{in} are the weight matrices that correspond to the gates in the RNN architecture. W_μ and W_σ are two fully connected layers that allow to compute the mean and standard deviation in the bottleneck layer (variational layer).

Using the reparametrization trick described above, a latent vector **z** is sampled from this encoding and is fed into the decoder RNN. Similarly, the states in the decoder are updated as in the encoder by doing:

$$
\begin{aligned}
h_0 &= tanh(W_z^T z + b_z) \\
h_{t+1} &= tanh(W_{dec}^T h_t + W_x^T x_t + b_{dec}) \\
x_t &= sigmoid(W_{out}^T h_t + b_{out})
\end{aligned}
\tag{2.3}
$$

Finally, W_{dec}^T and W_z^T correspond to the weight matrices that compose the autoencoder and x_t is the reconstructed term corresponding to the sampled latent vector **z**.

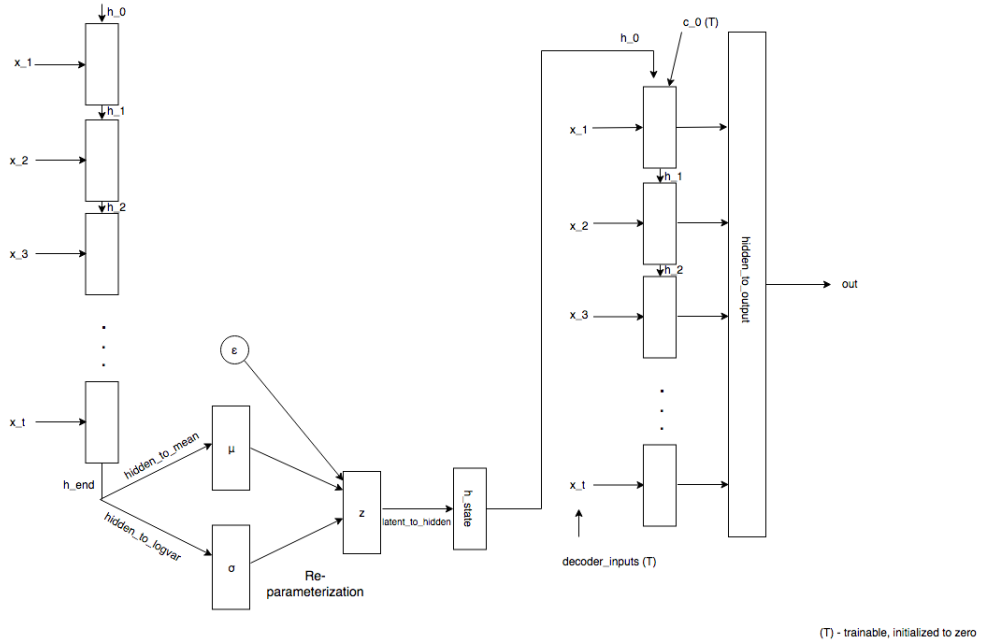


Figure 2.6: Variational Recurrent Autoencoder architecture. Image obtained from Python Awesome website [3].

Figure 2.6 shows a standard diagram of a VRAE. On the left side, the encoder takes t data time

steps and processes them through RNN cells computing hidden states h_t. Then, the VRAE takes the last hidden state h_{end} and computes its mean μ and standard deviation σ using two fully connected layers. These two parameters are used to compute the latent vector $\mathbf{z}$ and finally this vector is the input to the RNN decoder which outputs the reconstruction of the time series.

Fabius et. al (2015)[5] trained a VRAE on song fragments in order to learn a latent space capable of generating synthetic music. The figure below (right image) shows the organized data points (songf fragments) in the latent space. Each encoded datapoint is visualized at the location of the resulting two-dimensional mean (x,y) of the encoding.

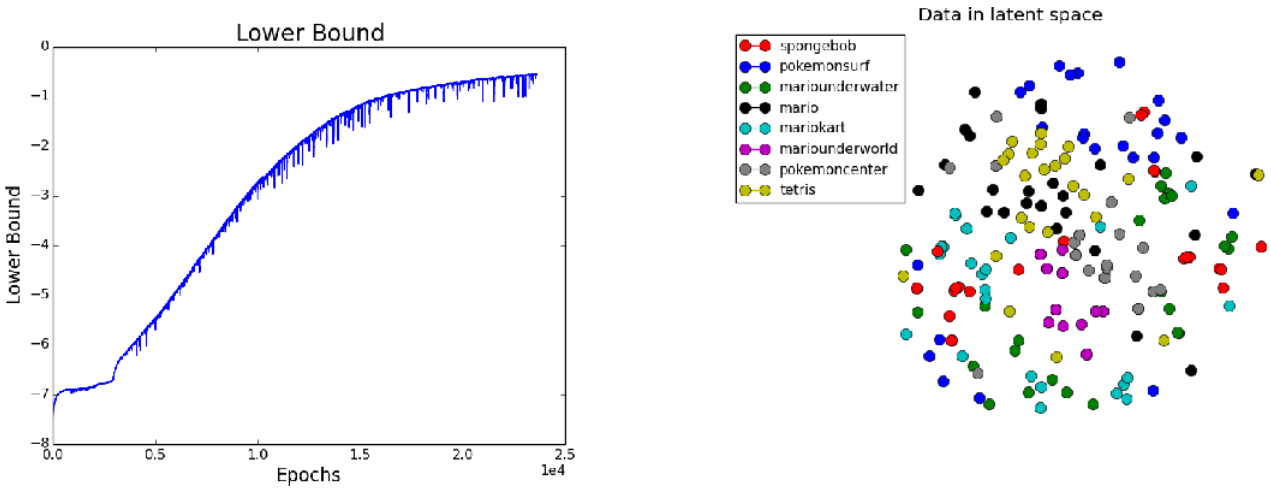


Figure 2.7: Learned representations of song fragments (see labels) using a VRAE. The latent vectors are projected in two dimensions to evaluate clustering. Image obtained from *Fabius et. al (2015)* [5, p. 3].

2.2.2 Convolutional Variational Autoencoder

Another type of neural network that has had vast success in tasks in areas related to computer vision and image processing are convolutional neural networks (CNNs). The combination of machine learning algorithms with the properties of convolutional filters (weight sharing and local invariance) make them very suitable to perform multiple tasks of interest in image-like data. CNNs hold state of the art in applications like image classification [20] and image segmentation [11].

Similarly to RNNs, CNNs can be combined with a VAE in order to form a Convolutional Variational Autoencoder (CVAE). This architecture is thus capable of learning image representations in low-dimensional spaces and generating synthetic images.

In the case of turbulent flows, we have framed the problem of dealing with turbulence data as images. In this manner, the channels of a turblence slice correspond to the temporal physical values that each slice carries (pressure, velocity). The figure below shows a CVAE that handles 2D turbulence slices. Initially (top row) the CNN encoder takes 2D slices as inputs and processes them through a series of convolutions and downsampling operations. The dimensionality of the latent vectors is a main parameter that is tuned.

After that, variational layer (bottom row, right part) reshapes the latent vectors in order to feed them through dense fully connected layers to compute the parameters of the latent distribution (mu and sigma). Lastly (bottom row), the CNN decoder implements the same transposed convolutions and upsampling operations and outputs the reconstructions.

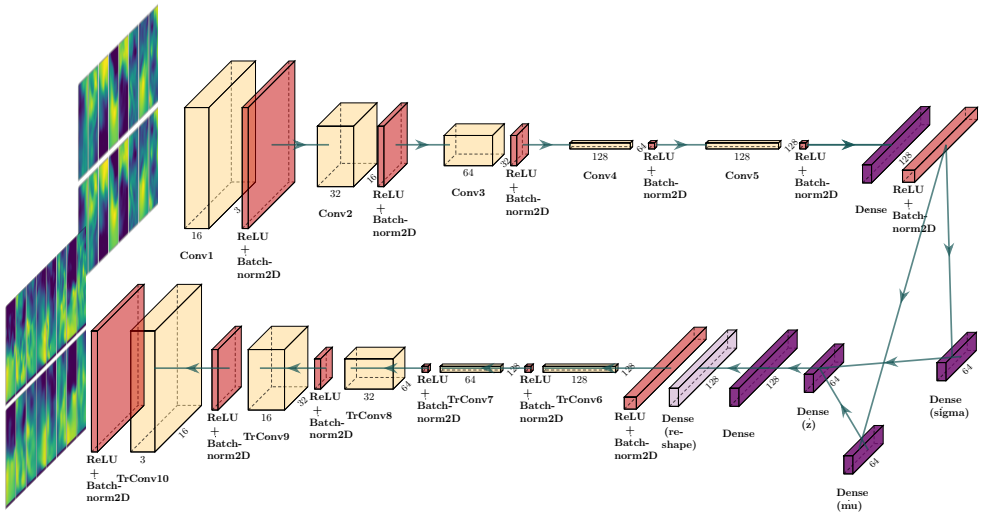


Figure 2.8: Convolutional Variational Autoencoder architecture.

2.3 Unsupervised Representation Learning of Time Series with Deep Neural Networks

Data compression methods like PCA and kernel-PCA have been efficient in many use cases and also straightforward to implement [23]. However, a numerous amount of datasets show non-linearities that can not always be captured by these methods. This motivated the development of other techniques like t-SNE by *van Der Maaten et. al (2008)* [29]. Just like PCA, t-SNE is also used for visualizing large datasets, it uses a random walk on neighborhood graphs to reveal structure at different scales. These methods of dimension-reduction are a key tool to gain insights into any clustering behavior that the data may present.

In the case of multi-variate high-dimensional time series VAEs, work has been conducted by *Pereira et. al (2019)* [25] on anomaly detection using VRAEs. The authors demonstrated state-of-the-art accuracy for the detection of anomalies in the ECG500 dataset. The high-dimensional data was first compressed and then a clustering algorithm was applied to this transformed data. This unsupervised learning scheme achieved an accuracy exceeding 90%.

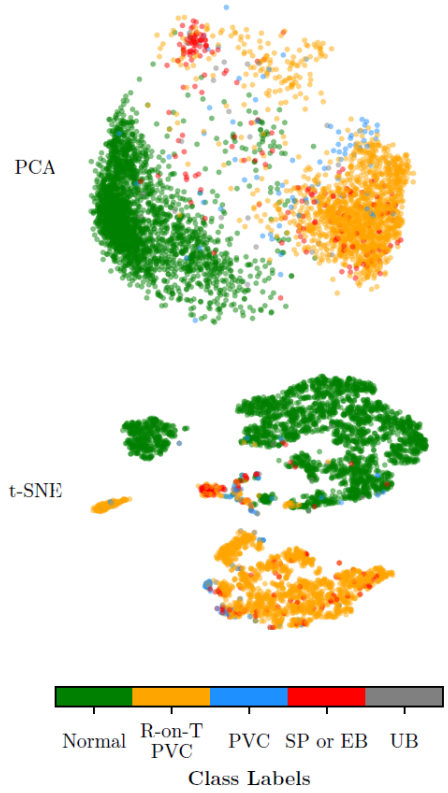


Figure 2.9: 2D projection of the compressed ECG500 time series using a VRAE. PCA and t-SNE are compared. Image reproduced from *Pereira et. al, 2018* [25, p. 5].

Moreover, *Zhao et. al (2018)* were able to extract the relationship between time series variables obtained from the monitoring of wind turbine systems in [34]. This group worked with an autoencoder network based on Restricted Boltzmann Machines as well. Their method successfully implemented an early warning of faulty components and also deduced the physical location of such components.

2.4 Turbulent Flow Compression, Reconstruction and Generation using Machine Learning

The most recent related work on reconstruction and generation of turbulent flows can be found in *Xing (2018)* [30] and *Fukami (2019)* [6]. *Fukami et al. (2019)* [6] developed a deep learning method to generate time-dependent turbulent inflow data, their neural network uses a combination of convolutions with a multi-layered perceptron in an autoencoding manner to reconstruct the presented DNS data (see figure below). One of the main results obtained here was that the machine learning turbulence generator was capable of accumulating turbulence statistics at a much lower computational cost than the corresponding driver simulation. A main drawback of this implementation is that it is very sensitive to changes in network parameters (hidden layers, units) and a better network structure is needed to achieve robustness.

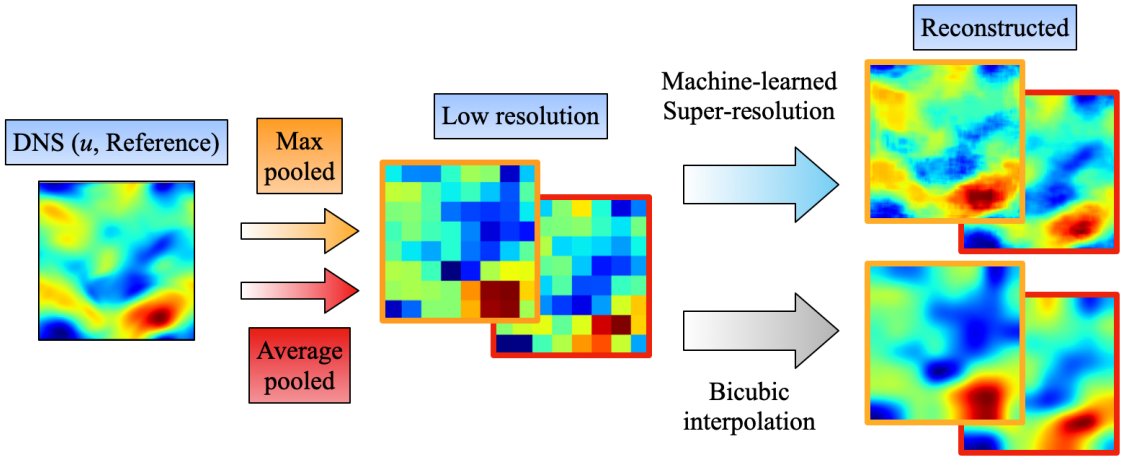


Figure 2.10: Super-resolution machine learning pipeline to reconstruct DNS flow data. The output compares the machine learning reconstruction versus bicubic interpolation. Image reproduced from *Fukami et. al, 2019* [6, p. 3].

For generation of turbulent flows, *Grogan (2017)* gave a proof of concept for the viability of VAEs to learn the characteristics of turbulent flows in [9]. In his report, Grogan used a 3D convolutional VAE and trained it on a standard database of homogeneous isotropic turbulence. The author was able to reconstruct a non-trivial turbulent vector field. The main deficit of his work is that his results are under-fitted resulting in a low-quality reconstruction.

Xing (2018) presented an improvement on the reconstruction results of Grogan as reported in [30]. One of the main modifications is that this author worked with 2D instead of 3D data to avoid overfitting. He trained 800 2D velocity fields of size 1024x1024 and conducted a more extensive analysis of reconstruction, generation and lossy compression. Potential improvements to his work are desirable as he trained with data from a single simulation, only used small 32x32 velocity fields and restricted himself to 2D slices.

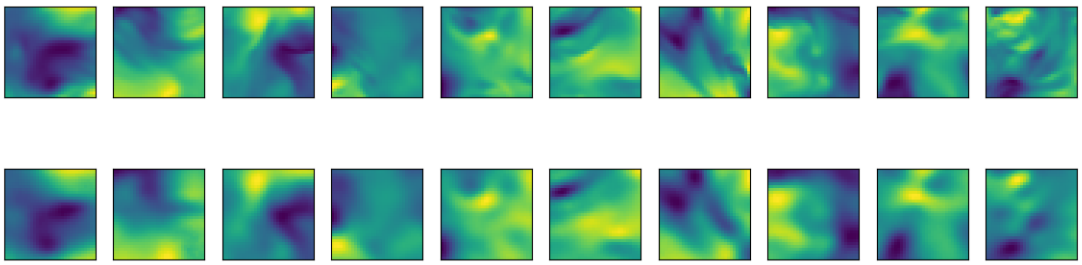


Figure 2.11: 2D reconstructed flow data using a VAE. The top row are the samples and the bottom row are corresponding reconstructions. Image reproduced from *Xing et. al, 2018* [30, p. 17].

2.5 Limitations of previous work

One of the main motivations to make use of VAEs in both problem domains is that the related literature has reported results only using benchmark datasets. In our case, we will use simulation data of systems of our own interest.

For wind turbine anomaly detection, the potential of VRAEs for this data will be researched. *Pereira et. al (2019)* [25] show that VRAEs can efficiently learn representations of uni-variate high-dimensional and labelled data. In this case, multi-variate and non-labelled data together with more time steps than the standard ECG500 datasets will be used. These technicalities suggest that data compression should be performed in a more gradual way by introducing another hidden layer to the network initially.

In *Zhao et.al (2018)* [34], the team used SCADA data for learning. This data is different to the one used in monitoring systems. The difference is that the analyzed quantity is either the time series or the frequencies of multiple sensors. SCADA data contains operating parameters of the gear box, generator speed and power. These quantities are rather meta-data. The aim here is a finer analysis using the time series from tachometers, more in the sense of *Gantsala et.al (2018)* [7] but using the time series and not the frequencies.

In addition, *Zhao et.al (2018)* analyze the data gathered from a 1.5 MW plant (also smaller than the plants analyzed here). In this research there is a controlled data set as given by simulations where parameters can be varied and the effects can be identified. Depending on the results from the simulations, real data from the company Weidmueller could be analyzed later.

In the case of turbulent flows, more work is needed using VAEs for different types of flows. *Xing et. al (2018)* [30] offer one of the most detailed reports as yet on turbulence using VAEs. However, it focuses on a particular type of flow. This is not sufficient since the potential that VAEs can bring to the analysis of the HVAC systems described here has not yet been explored. The HVAC system contains several challenging flow features to investigate, such as the flow around an obstacle and pressure driven flow separation.

3

Methodology

This chapter contains a description of the wind turbine and turbulence datasets used in this project as well as a section that describes several relevant data pre-processing steps. It also includes sections that detail the proposed models for both wind turbines and turbulence data separately explaning the use of VAEs for unsupervised representation learning.

3.1 Wind turbine time series dataset

This dataset is composed by time series generated from a simulation of a wind turbine system. It contains simulations of the turbine operating with and without ice accumulated in the rotorblades.

In the setup of this simulation, we divide the rotorblades in three sections: one covers the first half of the blade and the other two sections divide the second half into two extra halves again. We model different masses at each region. The convention to refer to the region and amount of mass for each simulation is $x - y - z$. x corresponds to mass increase in the first zone, y on the second zone and z on the third zone.

The dataset contains only simulations of time series with mass places in one region at a time (no combination of masses in multiple regions) that have the following configuration:

$$\begin{aligned} zone1 &\rightarrow x_{mass} - 0 - 0 \\ zone2 &\rightarrow 0 - y_{mass} - 0 \\ zone3 &\rightarrow 0 - 0 - z_{mass} \end{aligned} \tag{3.1}$$

Considering that this dataset is used to train neural network models, we have balanced it to have both "normal" and "abnormal" time series cases. A normal time series corresponds to the configuration $0 - 0 - 0$ (no ice accumulation) and the abnormal configuration corresponds to any of the three in eq. 3.1.

We have arranged the dataset so that it has approximately the same number of normal and abnormal cases. It contains a total of 25 simulations (14 normal cases and 11 abnormal) cases. Each simulation corresponds to a time series consisting of 47,000 time steps. These long sequences can, thus, be splitted into smaller chunks of 200-1000 time steps to use them as input data to train a neural network model.

These simulations have been generated using a wind turbine simulation software called FAST (v8). A total of 27 variables have been taken into account and their description is found in *appendix A.0.1.*

The figure below shows a normalized time series corresponding to one simulation. Here, we plot six different variables out of the total of 27. This data corresponds to 1 sample with 6 features and 47,000 time steps.

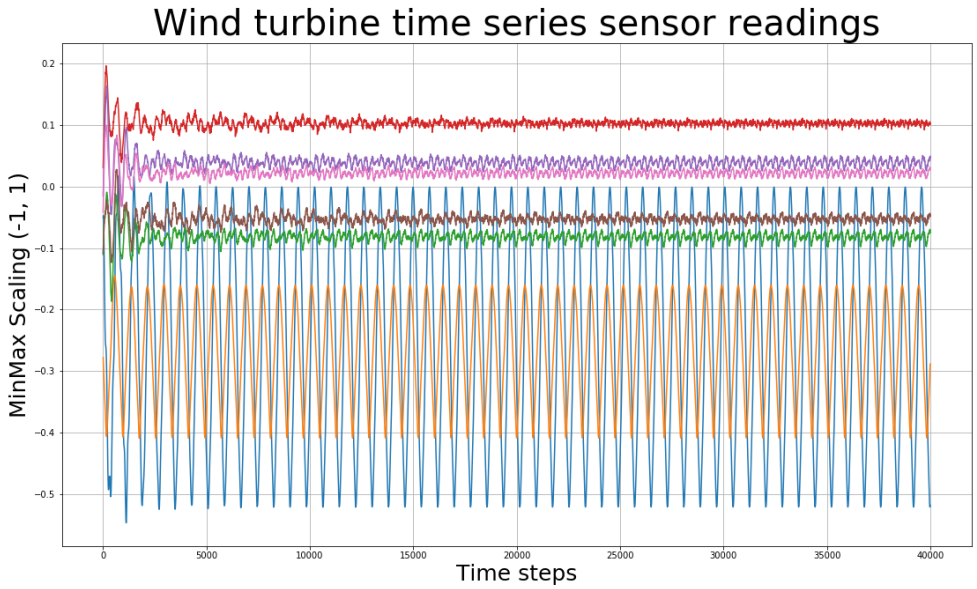


Figure 3.1: Wind turbine normalized time series sample.

3.2 Wind turbine time series data pre-processing

In this section we describe the main data pre-processing steps prior to input it into a neural network model.

We read the data from individual h5 files and grab only up to 10,000 time steps (out of the 47,000). In some of these time series wind speed is constant after some threshold and also they take some time to reach a stable state, that is why most of the relevant information is contained in between the 2000 - 10,000 time steps. For this project, we are only reading the weight configurations shown in equation 3.1 and not any combination of weights in two or three zones.

As a next step we normalize our time series using MinMaxScaler to a given range (-1 to 1). This is a crucial pre-processing step since there are sensor channels (features) that have higher amplitudes than others, this can affect the learning performance of a neural network model. After this we filter the features we work with and narrow them down to only six features, namely the accelerations in flapwise and edgewise components for three blades (see table below).

We make sure that the data is in the format *(samples, timesteps, features)* which is the required input for an RNN in Pytorch. Next we cut each time series into smaller samples, such that one time series can actually be regarded as having more observations. As the simulations are set up such that we have

Table 3.1: Wind turbine time series filtered features from simulation.

Wind turbine simulation parameters	
Parameter	Description
Spn1ALxb1	Blade 1 local flapwise acceleration (absolute) of span station 1
Spn1ALyb1	Blade 1 local edgewise acceleration (absolute) of span station 1
Spn1ALxb2	Blade 2 local flapwise acceleration (absolute) of span station 1
Spn1ALyb2	Blade 2 local edgewise acceleration (absolute) of span station 1
Spn1ALxb3	Blade 3 local flapwise acceleration (absolute) of span station 1
Spn1ALyb3	Blade 3 local edgewise acceleration (absolute) of span station 1

roughly 12 rotations per minute, we should see one rotation every 5 seconds, hence taking a length of 500 or 600 time steps could serve as a first initialization. This procedure generate approximately 1250 chunks that are then split into training and validation sets.

After these steps the data is ready to be fed into a neural network. These pre-processing steps ensure a robust and reliable procedure in order to learn efficient representations.

3.3 Wind turbine time series proposed model for anomaly detection

The proposed model for ice detection in the rotorblades consists of two fundamental steps: representation learning and anomaly detection. Both steps are merely unsupervised. We are basing this approach based on the one proposed by [25] using the ECG500 time series benchmark dataset.

Representation Learning

The model that we use for this task is the VRAE (introduced in chapter 2). More formally, let $\chi = [\mathbf{x}^{(n)}]_{n=1}^{N}$ be the time series dataset composed of N sequences, with each sequence having a length T, $x^{(n)} = [x_1^{(n)}, x_2^{(n)}, ..., x_T^{(n)}]$, and each datapoint $x_t^{(n)}$ is a d_x dimensional vector (number of features).

The encoder of the VRAE takes each time series $x^{(n)}$ and it is parametrized by a long short-term memory (LSTM) layer that at each time step computes a hidden state h_t^{enc}. The last hidden state h_T^{enc} is thus an abstract representation that represents the whole given sequence $\mathbf{x}$. Similarly to [25][26], the prior distribution $p(\mathbf{z})$ is a multi-variate normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The parameters that approximate the posterior distribution $q_\psi(z|x)$, μ_z and Σ_z, are obtained by taking mean and standard deviation from this last hidden state by using two fully connected layers with a SoftPlus activation. According to [25], using a SoftPlus activation ensures that variance is non-negative . The latent variables $\mathbf{z}$ are sampled from the parametrized posterior $q_\psi(z|x)$ via μ_z and Σ_z by using the re-parametrization trick discussed already in chapter 2 by doing

$$z = \mu_z + \sigma_z \odot \epsilon \tag{3.2}$$

Where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is gaussian noise and $\odot$ corresponds to element-wise product.

The decoder of the VRAE is another LSTM network that takes as input the latent vector $\mathbf{z}$ from the approximate posterior and outputs at each time step t the parameters that reconstruct the input variable $\mathbf{x}$. Similar to the encoding distribution, the decoding distribution $p_\phi(x|z)$ is defined as a

multi-variate Gaussian distribution. The loss function is the VAE loss function introduced in chapter 2 and the training procedure follows subsequently following gradient update and stochastic gradient descent.

Anomaly detection

We perform anomaly scoring using the learned low-dimensional time series representations provided by the VRAE model. Following the procedure of [25], the model is mapping sequences $\mathbf{x}$ into a lower-dimensional space and we then project them into two dimensions using PCA and t-SNE in order to evaluate grouping in specific regions. This makes it more feasible for a clustering method to detect normal vs anomalous (abnormal) cases. Anomaly detection consists, therefore, in detecting if a latent representations is normal or abnormal. In this work, we have implemented this detection using clustering algorithms.

Clustering algorithms give a numerical label to each latent representation, framing the problem as a two-class or multiple-class classification problem. We are taking this approach based on the fact that the model is capable of learning representations that tend to group in lower-dimensions given a balanced normal and abnormal percentage of training data, also assuming there is statistical difference between cases. We have applied three different clustering methods in the representations: *k-means ++* [1], density-based spatial clustering (*DBSCAN*) [4] and *hierarchical clustering* [27]. These methods are set to find two (or more) clusters given the number of classes (normal and abnormal). The output is then matched to the ground truth labels given corresponding to the actual class each representation belongs to. With this, we can compute a classification accuracy.

3.4 Turbulence dataset, data pre-processing and proposed model

This dataset is composed of extracted slices/cubes from a HVAC duct CFD simulation. We store the 2D slices as arrays of dimensions $41 \times 41 \times 2000$ ($coordinate_x$, $coordinate_y$, $timestep$). Each slice represents a two dimensional temporal snapshot of the turbulent flow carrying physical information at that particular time. Similarly, the extracted 3D cubes are stored in arrays of dimensions $21 \times 21 \times 21 \times 2000$ ($coordinate_x$, $coordinate_y$, $coordinate_z$, $timestep$).

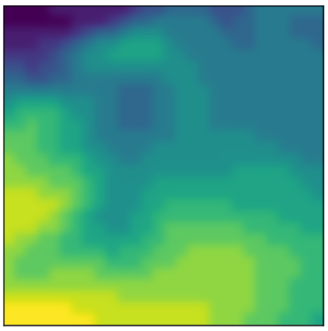


Figure 3.2: Two dimensional slice extracted from a HVAC duct simulation.

The information extracted from the simulation is based on flow variable (velocity U, static pressure

p), the vector component (x, y, z) for U, the scalar component for p as well as the orientation of the flow based on the normal direction of the slice. The figure above shows an example of an extracted 2D slice. We visualize an individual slice in the form of a heatmap.

We frame the problem of feeding turbulent data into a neural network model as if they were images. This is based on the parallelism that exists between both data types and the results obtained by [30] [9] using a the John Hopkins Turbulence benchmark database. In contrast to images, we don't perform any operation like cropping or rotations for augmenation, we rather keep the original dimensionality of the data in order to maintain all the physical information possible. The only operation performed is a normalization step to fix a range in the values of the data. The figure below shows a pre-processed set of turbulence slices. This data is the input into our neural network model.

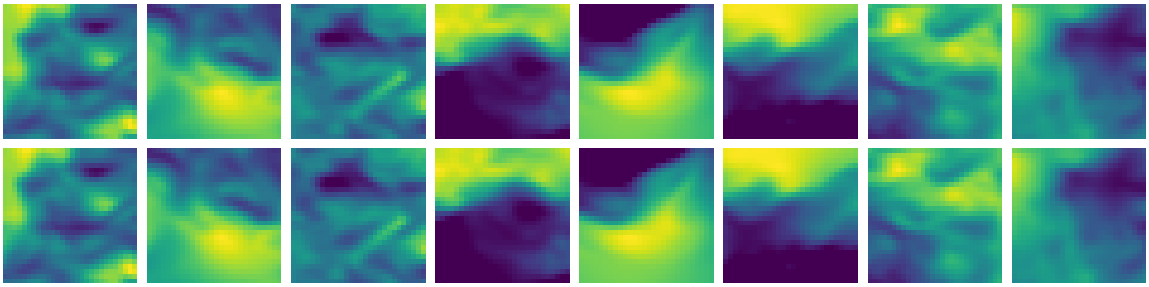


Figure 3.3: Grid of two dimensional turbulence slices after pre-processing.

The proposed model is the Convolutional Variational Autoencoder (CVAE) introduced in chapter 2. We have taken as a baseline architecture the one proposed in [30]. The encoder and decoder of the CVAE have 4 symmetrical convolutional hidden layers. Each layer of the encoder has twice the number of convolutional filters as its predecesor, this is in order to learn more complex flow features. Encoder/decoder are composed of 32, 64, 128 and 256 filters. The dense layer at the end of the encoder is used commonly to combine all the feature maps from the last hidden layer. After that, there is a variational layer that similar to VRAEs, computes the parameters of the posterior distribution and from here we compute latent vectors using the re-parametrization trick, in this case it is 8×8 latent images. The decoder takes the latent vectors and performs the same number of operations using transposed convolutions in an upsampling manner in order to recover (reconstruct) into the original dimensions. The initial hyper-parameters of the network were taken from the work by *Higgins et al. (2017)* [12] and *Xing et al. (2018)* [30]. After this, a fine-tunning of the hyperparameters is done according to our train and validation losses.

4

Experiments and Results

In this section we present the results obtained in unsupervised representation learning and clustering of time series for anomaly detection in wind turbines using a Variational Recurrent Autoencoder. We cover the case of two classes: normal and abnormal case as well as multiple classes. This section also contains the reconstruction and generation results of the two dimensional HVAC duct turbulence data using a Convolutional Variational Autoencoder. All our neural network models are implemented using the Pytorch framework [24] and a Tesla V100 GPU card with 32 GB of RAM.

4.1 Unsupervised time series clustering and anomaly scoring: two classes

Raw effect of PCA on time series

As an initial test, we projected the original given input data (N samples, 200 timesteps, 6 features) directly in two dimensions using PCA. The figure below corresponds to the PCA projection of the data (no furhter training). As noticed, pure raw PCA applied to the data is not good enought to cluster normal and abnormal time series in distinctive regions. This is mainly because of the fact that such technique is not always capable of capturing the non-linearities that exist in multi-variate time series, for example.

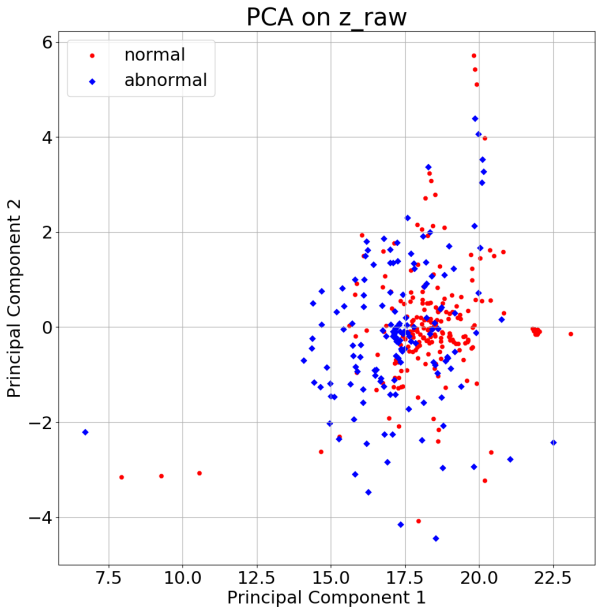
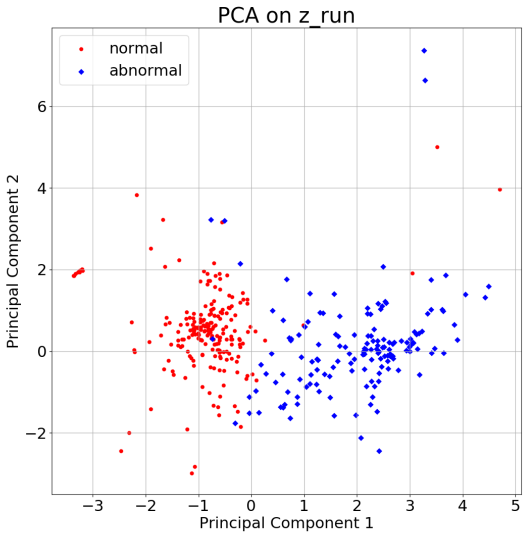


Figure 4.1: PCA applied to raw time series data without neural network processing.

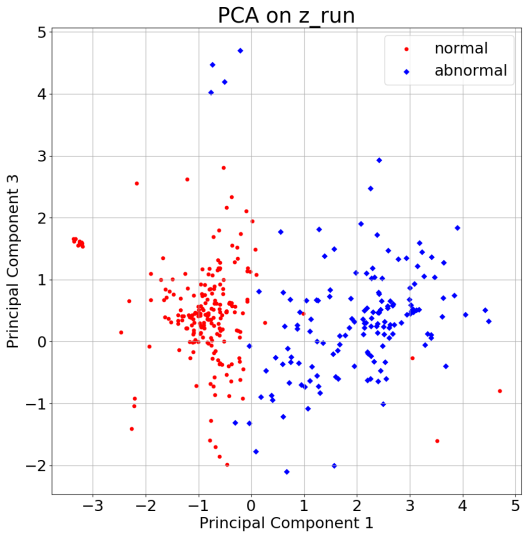
VRAE unsupervised representation learning with two classes

Next, we trained a VRAE using a single hidden LSTM layer with 90 units (a single hidden layer architecture gave better results than a multi-layered one). The bottleneck layer (variational layer) project these 90 dimensional representations into latent vectors of 20 dimensions. We use the Adam optimizer [18], gradient clipping (to avoid gradient explosion) and dropout in the hidden layer with rate of 0.2. The learning rate is $l = 0.0005$ and momentum 0.9. The data is split in 70% training and 30% validations sets with time series split into chunks of 200 time steps and the 6 features described in the previous chapter. In this case, we are only comparing normal case versus abnormal cases corresponding to the zone 1 of the turbine blade. We load batches of 64 into memory and perform training for 2000 epochs. The reconstruction term in the loss objective is the MSEloss given that we assume that the parametrized posterior probability $p(x|z)$ is a normal distribution. The training time is approximately 30 minutes.

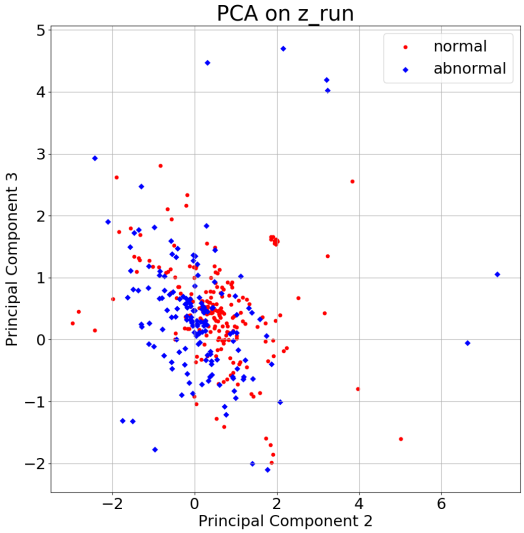
The figure below shows the results obtained with this model. The points correspond to a projection of the 20-dimensional learned latent representations into 2-dimensional vectors using PCA. We note the data is properly grouped according to normal and abnormal cases in different regions of the plane. Figures (a) and (b) compare the first principal component versus the second and the third one, respectively. Figure (c) compares the second principal component versus the third one. Note how in (a) and (b) the separation between classes is visually evident. In contrast, in (c) there is a considerable overlap between classes, this can have to do with the fact that there is more correlation between the second and the third components of this projection.



(a) First component versus second component.



(b) First component versus third component.



(c) Second component versus third component.

Figure 4.2: PCA projection of the learned 20-dimensional representations using a VRAE. The figures compare the relationship between the first three principal components of the projection.

Analysis of other dimension reduction methods and latent vectors as lines

Besides PCA, we have implemented two other methods for non-linear dimensionality reduction; t-SNE

and spectral embedding (SE) [31][22]. In general, t-SNE is a method capable of projecting data into well defined clusters, note in the left figure below, that this is indeed the case with the time series latent representations. t-SNE is capable of almost grouping all normal and abnormal cases perfectly. However, in terms of interpretation of mapping and stochasticity of the final solution, we can not yet rely on t-SNE other than pure visual evaluation since it is not a deterministic approach as PCA, refer to this forum for a detailed comparsion between these two methods. The right figure below corresponds to the spectral embedding projection of the data, this algorithm is in close proximity to kernel PCA [22] and proved to cluster the time series latent representations almost perfectly, this method is yet to be better explored since it is easier to interpret and shows promising results.

Note that all the three methods for dimensionality reduction compared so far (PCA, t-SNE and SE) take as input the latent representation outputed by the VRAE. The VRAE processes the 6 sensor features and fuses them into one.

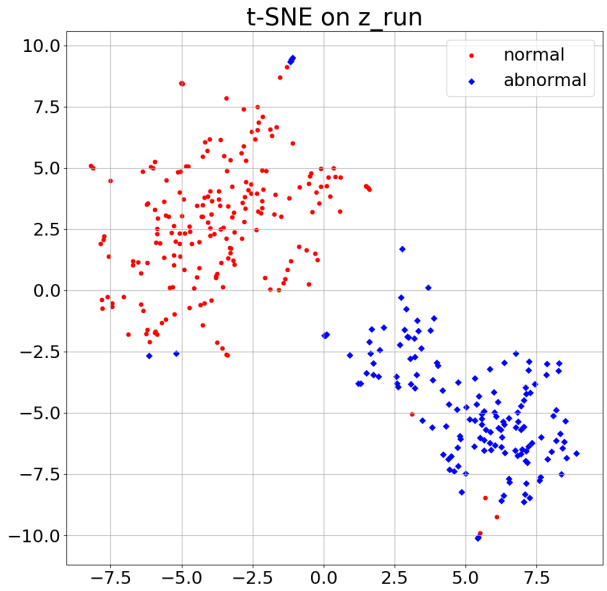


Figure 4.3: 2D t-SNE projections of the latent vectors.

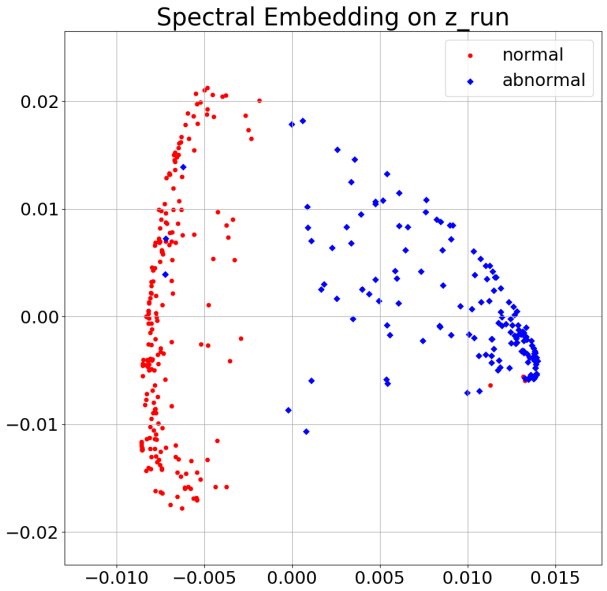


Figure 4.4: 2D SE projections of the latent vectors.

In this experiment we also plotted the 20-dimensional latent vectors generated by the VRAE as lines. We took 15 samples of normal and abnormal latent vectors and compared them in the plot below. The x-axis corresponds to each entry of the 20-dimensional vectors and the y-axis are the normalized amplitudes provided directly by the model. The red lines correspond to normal latent vectors and the blue lines to abnormal ones. Remarkably, we can see that even in their 20-dimensional representation, red lines tend to describe different maxima and minima compared to blue lines. Note for instance, that the dimensions 1. 5. 7, 13 and 19 are the ones where normal and abnormal classes have the less correlation.

Knowing what dimensions provide the highest contrast between our classes can in principle allow us to further reduce the dimensions of the latent vectors from 20 to 5 or 6.

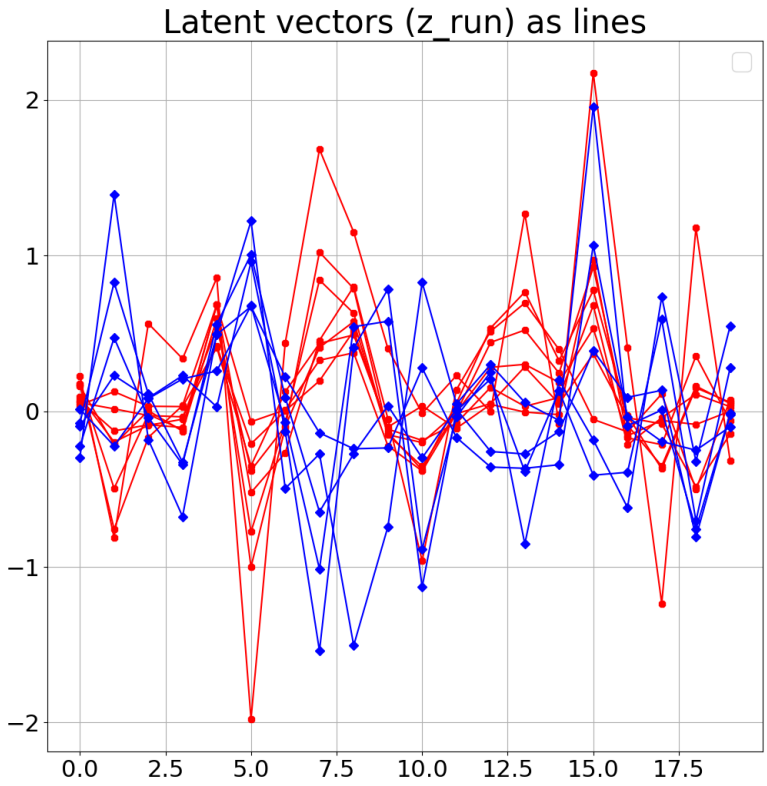


Figure 4.5: Latent vectors plotted as lines. Red lines correspond to normal classes and blue lines correspond to abnormal classes .

Effect of features used for time series representation learning

Another characterization we did in this experiment is the exploration of the effect of the number of input features when training a VRAE. The left figure below for example, corresponds to the PCA projections of the latent vectors when the model is trained only with 3 features, namely the x-components corresponding to the flapwise acceleration of span station 1: Spn1ALxb1, Spn1ALxb2, Spn1ALxb3. Similarly, the right figure below corresponds to training using only 3 features corresponding to the y-components of the local edgewise acceleration of span station 1: Spn1ALyb1, Spn1ALyb3, Spn1ALyb3. As we can see in both cases, projecting the data with PCA is not helping the model to learn representations efficiently in order to cluster time series in a significative way. This finding is relevant since the number of features fed into a model is a crucial parameter. From this observation, we proceeded to train every other model using at

least six input features.

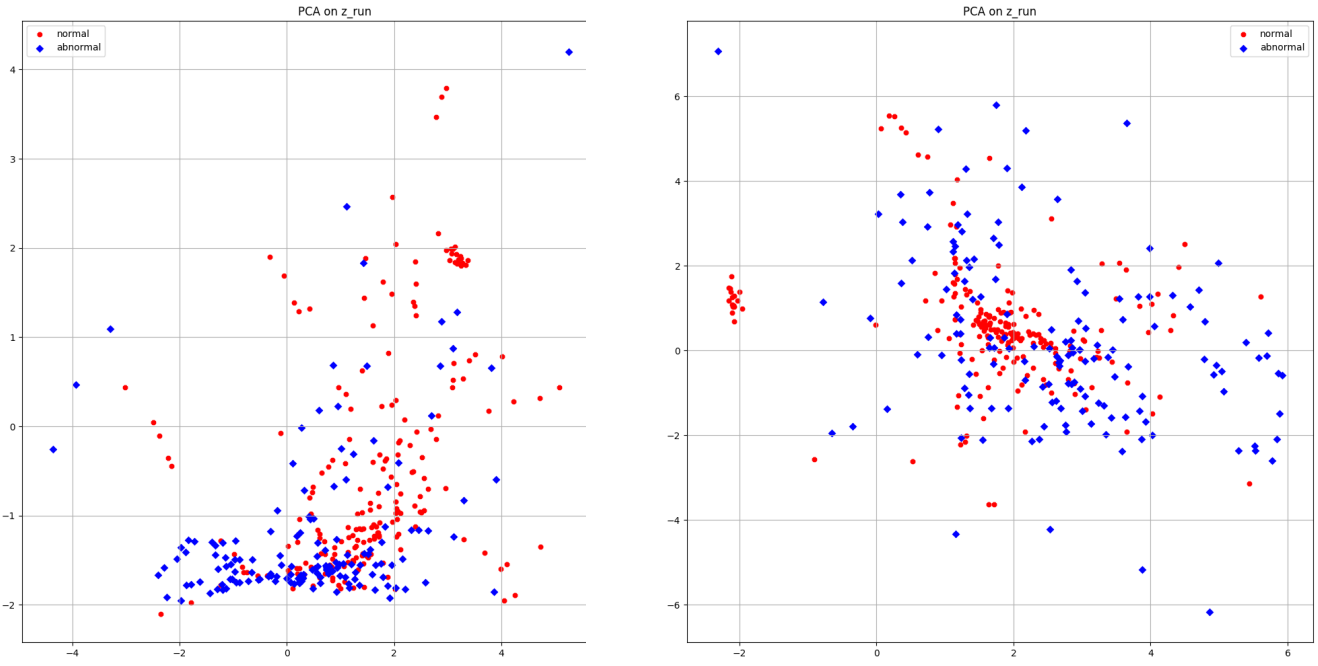


Figure 4.6: 2D PCA projection of latent vectors when training VRAE with only 3 features. Left figure: x-componentes of blaeds accelerations. Right figure: y-components of blades accelerations.

Unsupervised clustering methods and anomaly scoring

So far we described several experiments that characterized technical aspects of representation learning using a VRAE regarding its capacity to cluster time series in lower dimensions. The next stage in the anomaly detection pipeline, as discussed in the previous chapter, consists in applying clustering algorithms on the latent representations.

In the figures below, we show a comparison of the three clustering methods mentioned in the previous chapter; KMeans++, hierarchical clustering and DBSCAN. In the left upper figure, we show the projected latent vector, we colored these projections based on their groundtruth class correspondence (labels). The right upper figure and the two at the bottom correspond to the clustering/label predictions of each algorithm algorithms. Note for instance, that KMeans++ and hierarchical clustering color the points in the upper region of the plane with the same label (as belonging to cluster 1), this is not correct given that we know the groundtruth labels. This speaks about the limitations of the models in their ability to assign class labels. However, in general, we see that these two methods perform an efficient clustering classification on top of the latent vectors when compared to the groundtruth. The other method is

DBSCAN, note for example, that this method is not assigning class labels to all the points (that is why the y-axis limits differ), this has to do with the choice of hyperparameters (the clustering results presented here correspond to the best results obtained when running a hyperparameter search).

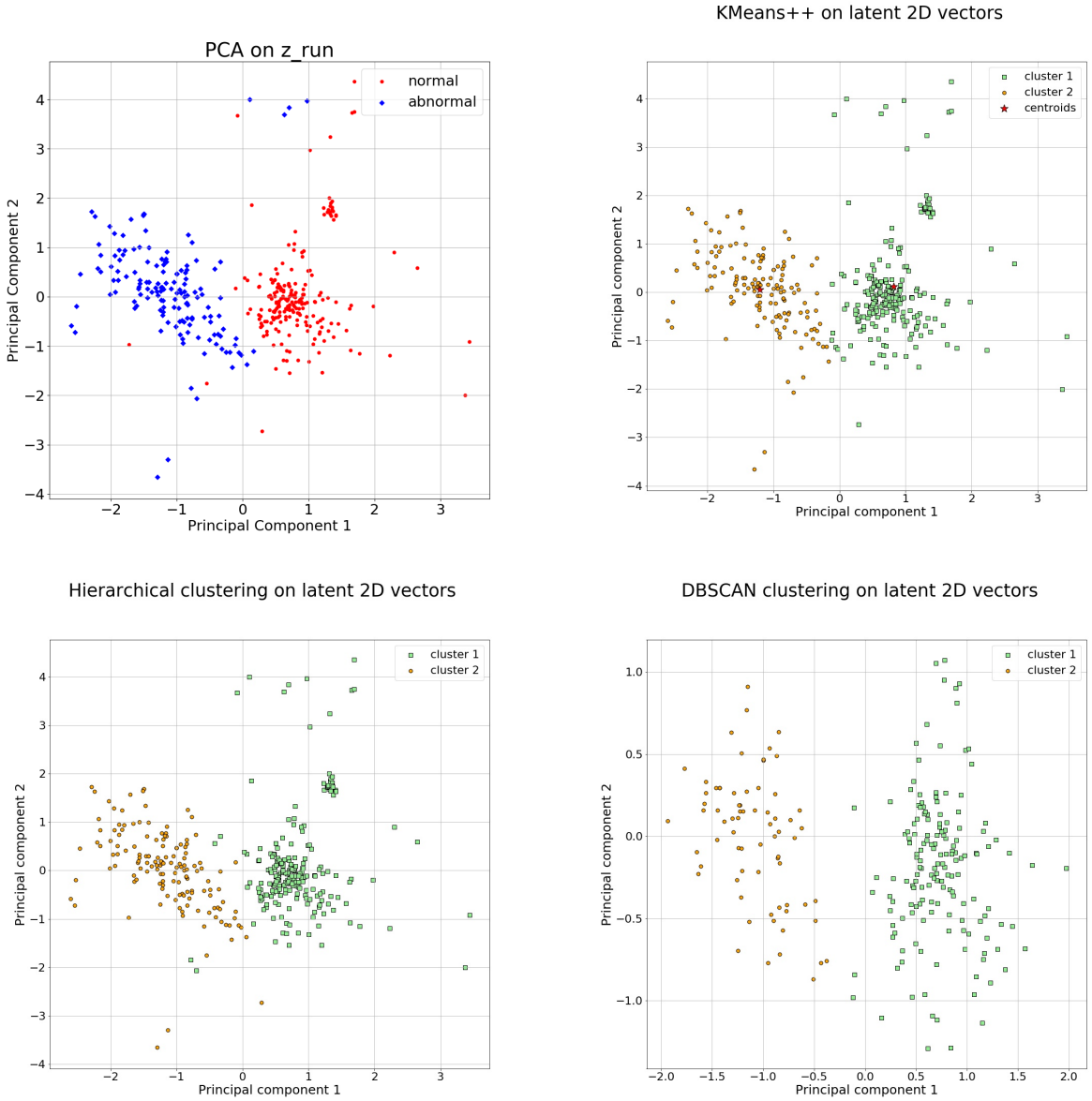


Figure 4.7: Class prediction of three different clustering algorithms operating on top of the projected latent vectors learned by the VRAE.

In order to provide a more quantitative evaluation of the classification output from these models, we

have computed classification accuracy, area under the curve (AUC), precision, among other metrics. The table below shows the results obtained:

Table 4.1: Anomaly scoring classification results using 3 unsupervised clustering algorithms.

Anomaly scoring results			
Metric	KMeans++	Hierarchical clustering	DBSCAN
Accuracy	0.9667	0.9639	0.6315
AUC	0.9619	0.9605	0.5507
Precision	0.9674	0.9641	0.9890
Recall	0.9667	0.9639	0.6315
F1-score	0.9666	0.9638	0.7616

The results from the table above indicate that DBSCAN had a low classification accuracy. This can have to do mainly with the hyperparameters chosen and should not be discarded as a clustering method. The other two methods on the contrary, achieved classification accuracies of up to 96%, which is 2% below the state of the art using a benchmark dataset [25]. These classification results indicate that our anomaly detection proposed framework is being able to classify between normal and abnormal time series up to 96% of the time.

4.2 Unsupervised time series clustering and anomaly scoring: multiple classes

The next problem we tackled is the extension of anomaly detection given multiple classes. Here, we have taken into account all three zones to compose our training dataset (weights in zone 1, zone 2 and zone 3). We trained a VRAE again using a single hidden LSTM layer, in this case, we modified this layer to 128 units. The bottleneck layer (variational layer) project these 128 dimensional representations into latent vectors of 5 dimensions. We have also changed the latent dimensionality based on fine-tunning of hyperparameters. Adam optimizer is used, gradient clipping (to avoid gradient explosion) and dropout in the hidden layer with rate of 0.2. The learning rate is $l = 0.0005$ and momentum 0.9. The data is split in 70% training and 30% validations sets with time series split into chunks of 200 time steps and the 6 features described in the previous chapter. We load batches of 64 into memory and perform training for 2000 epochs. The reconstruction term in the loss objective is the MSEloss. The training time is approximately 1.5 hours.

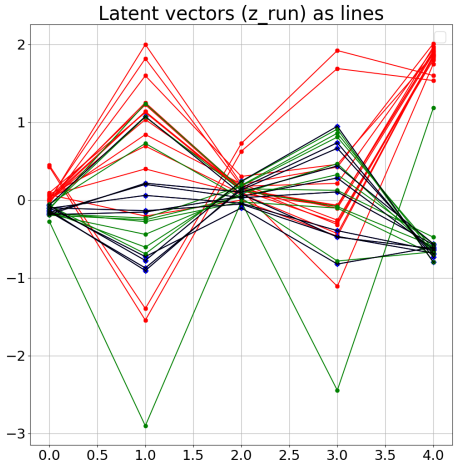
VRAE unsupervised representation learning with multiple classes

The figures below correspond to the PCA projections of the learned representations in the case of multiple classes. Similar to the case of the classes, we compare principal components and plot latent vectors as lines. In this case, we obtained that the model is capable of clustering normal versus abnormal cases, however, it is not capable of clustering appropiately abnormal classes depending on the zone. As we can see in both figures 4.8(a) and 4.8(b), regardless of the principal components we compare, the

projections always show that abnormal cases never spread with respect to each other. This leads us to think that there is no real variability for time series corresponding to each zone and therefore the VRAE is clustering all together of them together. Recall that the labels are colored simply based on groundtruth values, the model never uses labels when training.

(a) Multiclass first component vs second component.

(b) Multiclass first component vs third component.



(c) Multiclass projections: Sampled latent vectors plotted as lines.

Figure 4.8: PCA projections of learned representations in the case of multiple classes. (a) Projection in 2D using first and second components. (b) Projection in 2D using first and third component. (c) Plots of normal and abnormal 5-dimensional latent vectors as a line.

The observed behavior in figures 4.8(a) and 4.8(b) is actually consistent with figure 4.8(c), note that there is high variability between normal and abnormal cases, but there is no variability between abrnomal cases themselves, they rather tend to describe the same behavior. Given these 5-dimensional latent vectors,

we also note that the highest contrasts between latent vectors are found in the first and third components of these vectors.

For the sake of comparison, we also used t-SNE and spectral embeddings as alternative dimensionality reduction methods. We obtained a similar behavior as in PCA; both methods are capable of grouping normal versus abnormal cases, but none of them is capable of clustering the different zones adequately. See figure below for a visualization.

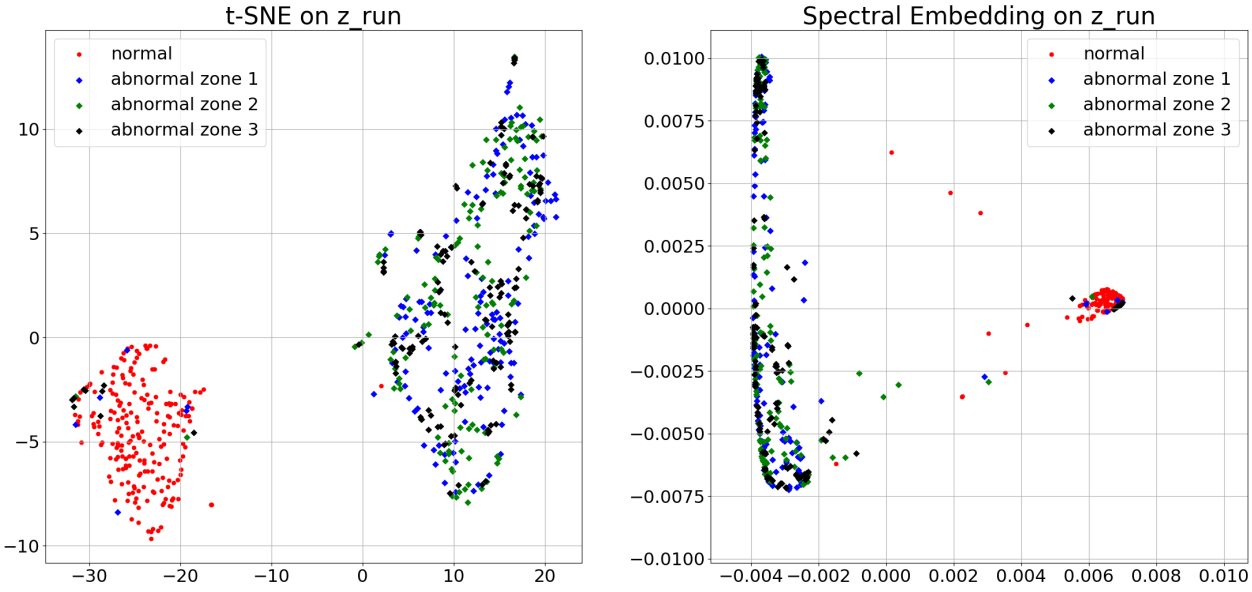


Figure 4.9: Left figure: 2D t-SNE projections of the latent vectors in multiclass case. Right figure: 2D SE projections of the latent vectors in multiclass case.

The results obtained so far led us to try other approaches in order to see if could cluster abnormal zones better. We implemented a VRAE using bi-directional LSTMs and also feeding samples of 500 time steps (not 200). However, for the bi-directional LSTM we obtained similar results as the ones shown above (uni-directional LSTM), in the case of longer time steps, we obtained even worse results, this can have to do with the fact that longer sequences implies less available training samples. Due to these reasons we did not run an anomaly scoring on top of these projections, we have to re-evaluate what we consider constitutes actual significant zone classes, increase the size of our dataset and test a multi-layered VRAE architecture.

VRAE multi-variate time series reconstructions

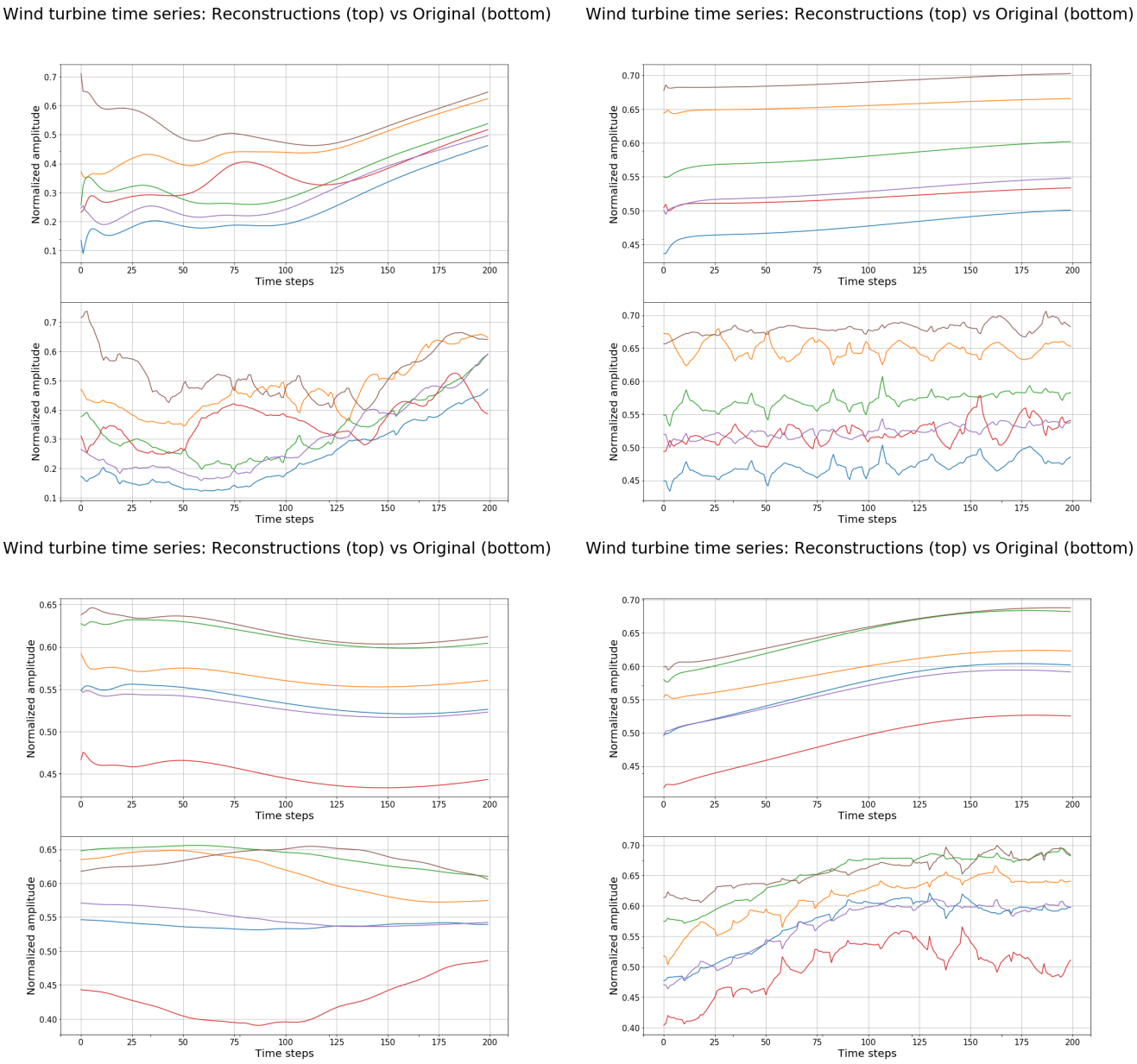


Figure 4.10: VRAE time series reconstruction results for a time series of 200 timesteps and 6 features. Upper left: reconstruction at 0-th epoch. Upper right: reconstruction at 200-th epoch. Lower left: reconstruction at 400-th epoch. Lower right: reconstruction at 600-th epoch.

During this experiment we also tested the reconstruction capabilities of the VRAE. The figures above correspond to a reconstructed sample at every 200-th iteration (epoch). In each figure, bottom row corresponds to the original signal and upper row corresponds to the reconstructed signal by the model. Each line represents one of the 6 features that are the input to the model. Note that in every case, the

model is being able to reconstruct the tendency and the amplitude of each individual feature signal. It is not, however, being able to reconstruct each signal with the original level of detail. This reconstruction evaluation can also give us better hints in order to optimize further the architecture and parameters of our model. For instance, a pre-processing of the data consisting of Fourier transforms might help recover the level of detail that is being missed in the figures above.

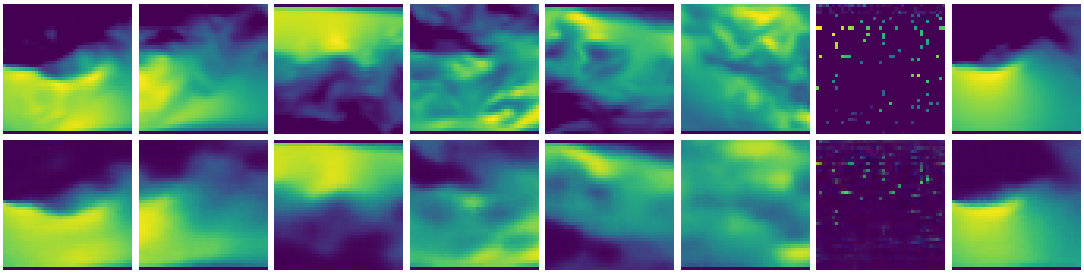
The reconstruction of tendencies is a good result given that there is no further pre-processing being done, however, it has been recently proven that VAEs can learn further finer reconstruction details by adding a latent constraint network that forces the model to learn new latent variables that are similar to training samples [33]. Further improvements in reconstructions will be in this direction.

4.3 Turbulence reconstruction and generation: two dimensions

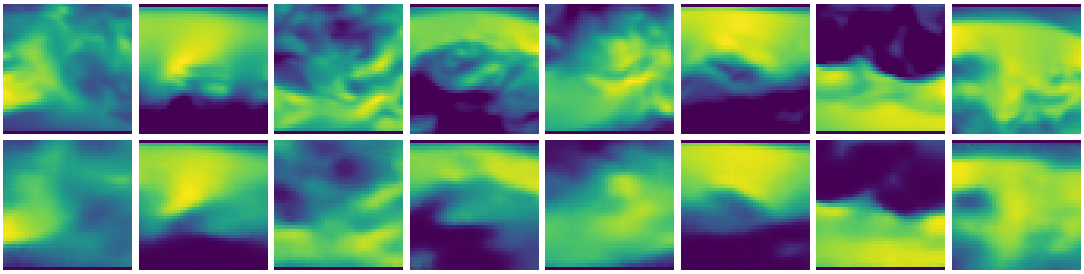
We have trained a CVAE on a dataset consisting of 500 turbulence slices. We built a custom pytorch dataset that takes as input numpy arrays containing the flow data and converts them into iterables in pytorch. This custom dataset can perform any image augmentation technique. We restricted these transformations to a simple normalization of the data to a particular range. Using this normalized data, we assigned a heat map to the slices in order to represent the intensity of the physical variable in 2D. As mentioned in the previous chapter, both encoder and decoder have 4 convolutions each followed by ReLU and batch normalization after each convolution. The dimensionality of the latent space in this case is 64 (8×8 pixel matrices). We used Adam optimization and binary cross entropy as the reconstruction term in the loss function. The data was split in 70% training and 30% validation. the learning rate $l_r = 1e^{-3}$ and we trained for 600 epochs with batch size of 128. This implementation was also using Tesla V100 GPU cards with 32 GB RAM. Training took 2 hours approximately.

The figure below shows the reconstruction results obtained with the CVAE trained with the HVAC duct turbulent flow. We show snapshots during training every 200-th epoch until the last 600-th epoch. As we can see (zoom in the image), the model is efficiently being able to reconstruct the slices up to a certain pixel resolution. There are particular regions where reconstruction fails, for instance, the second slice (from left to right) in Figure 4.11(c), shows that the model is not capable of reconstructing up to the highest resolution possible, it is at most being able to reconstruct the interface-like structure of the slice.

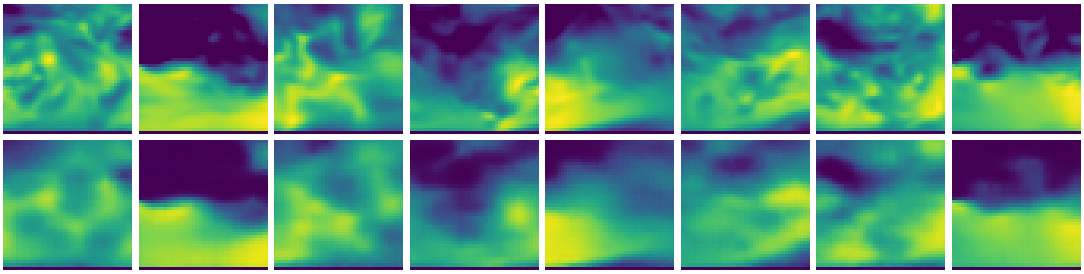
These results can be further improved by tunning the architecture and obtain pixel-level reconstruction results. The reconstructions obtained look promising and the next thing is to use a metric to evaluate the reconstruction accuracy as well as a comparison with other methods such as bicubic interpolation. We have not added the analysis on 3D cubes due to time constrains, that is left for future work.



(a) CVAE reconstruction results at 200-th epoch.



(b) CVAE reconstruction results at 400-th epoch.



(c) CVAE reconstruction results at 600-th epoch.

Figure 4.11: CVAE turbulent flow reconstruction results. Upper row: original turbulence data. Lower row: reconstructed data.

We have added a plot of the latent representation of the CVAE. The figure below corresponds to the latent matrices encoded by the model, they are 8×8 matrices and encode the information of the input slices in a compressed form. We believe this can help us achieve turbulence data clustering in the future.

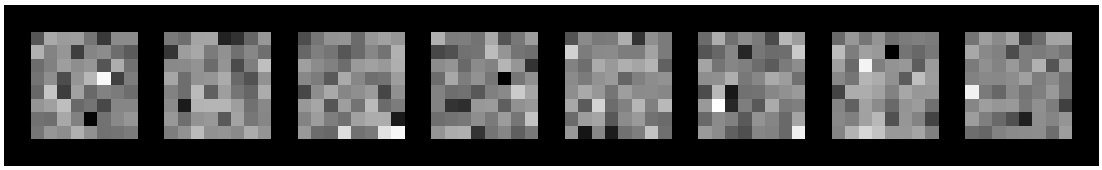


Figure 4.12: Latent representations of turbulence slices encoded by the CVAE.

In this experiment we also analyzed the potential of our model to generate 2D turbulence slices. The

figure below corresponds to a grid of a total of 64 different slices that were synthetically generated by the model. We took samples from the latent distribution and decoded using the transposed convolutions. Future work on this regard will consist in exploring the potential of the model to generate synthetic 3D turbulence cubes, this is based on the idea that turbulence is mainly a 3D phenomena. From here, we can test if the synthetic cubes are physically coherent.

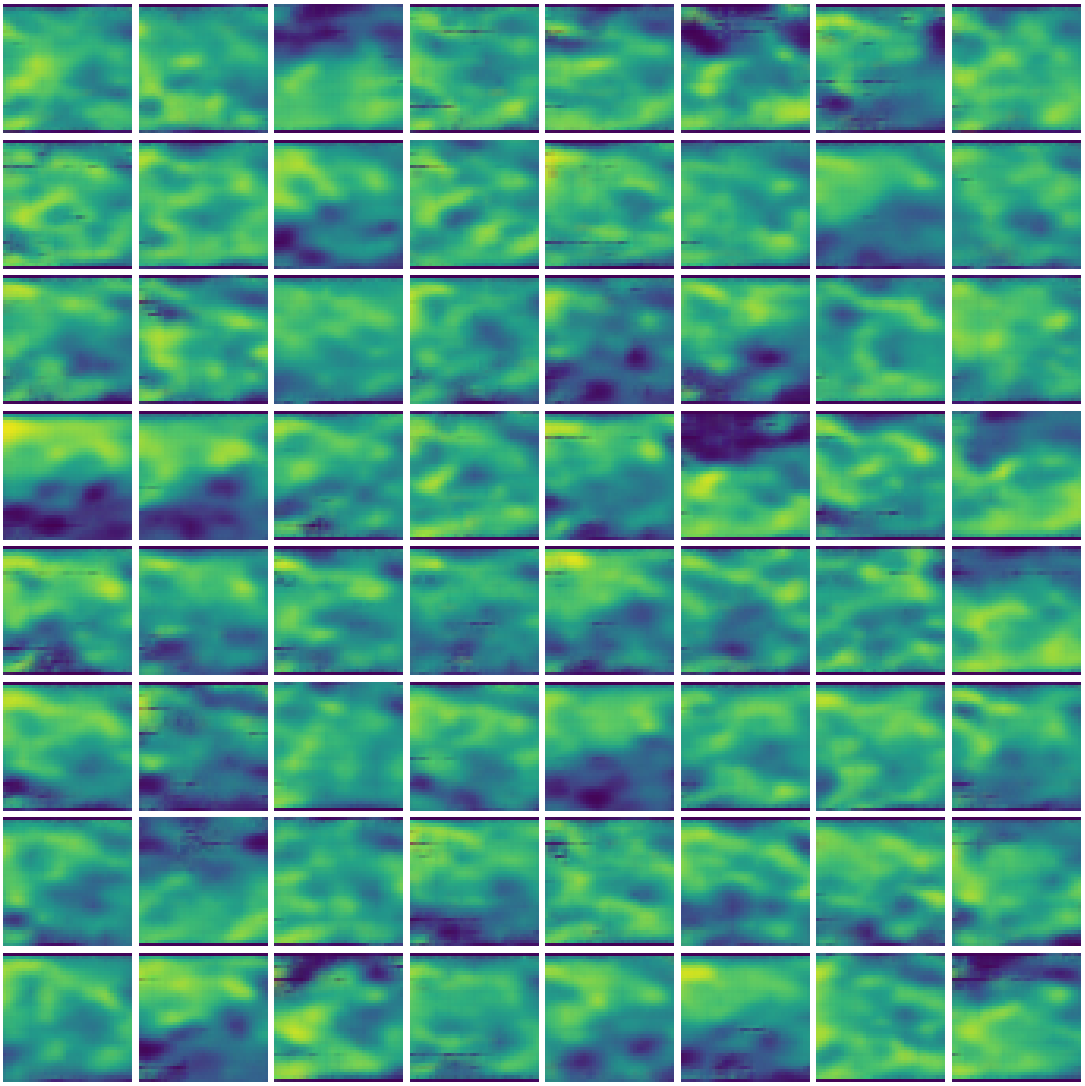


Figure 4.13: Synthetic turbulence data generated from the CVAE learned latent distribution.

5

Conclusions

This chapter summarizes the work that has been done with Variational Autoencoders so far. We provide a recapitulation of the main contributions of this project. We reflect on the lessons learned and finally describe the future research avenues emerged from this work.

5.1 Contributions

For the wind turbine anomaly detection problem, we have built a dataset composed of wind turbine simulations, we have developed tools to pre-process this dataset in order to feed the data into a neural network. Moreover, we have trained a VRAE on this data in order to learn low-dimensional latent representations of multi-variate time series and optimized it in order to cluster such low-dimensional representations. We have also implemented clustering algorithms that work on top of these latent representations and predict the class correspondence for each one of them. This whole pipeline has been built for two classes and multiple-classes. For the case of two classes, we have achieved an accuracy of 96% for correctly predicted time series.

For the turbulence data, we developed a pipeline to process two dimensional turbulence slides from a HVAC duct in order to feed them into a CVAE. We trained and optimized the parameters of the CVAE in order reconstruct and generate synthetic data. We were able to achieve a good resolution quality up to a certain pixel level as well as generating synthetic samples.

5.2 Lessons learned

We have learned the importance of building datasets that are expressive enough in order to train neural networks with them. Defining what a normal and abnormal time series must be done with careful criteria, otherwise the model might not provide the expeceted outputs. We have also worked out the technicalities when pre-processing wind turbine and turbulence data. Each one of these data domains require different pre-processing steps that are crucial in order to train neural networks. Moreover, we have gained deeper insights into the training of Variational Autoencoders. Great part of this project was spent building and fine-tunning the hyperparameters of the models in order to improve our results.

5.3 Future work

There are many aspects to be considered for future work. In the case of wind turbines, we plan to improve the clustering of time series for the multiclasses problem. For this, we want to explore attention mechanisms in Recurrent Networks and a more complex multi-layered architecture. At the same time, we plan to do a more extensive analysis on the dimensionality of the latent vectors by studying the correlations between normal and abnormal classes. We also plan to compare kernel PCA with a Radial Basis Function in order to compare it with the dimensionality reduction methods used (PCA, t-SNE and SE).

We also want to explore the benefits of taking a Fourier transformation approach in the pre-processing of the data in order to improve the reconstruction results of the Variational Autoencoder. So far it has been able to reproduce tendencies but not able to reconstruct all the fine details of time series. For anomaly scoring methods, we plan to implement a Wasserstein metric in order to compare it to the clustering algorithms used.

For turbulence data, we have started to work with 3D turbulence cubes. This is a line of research in its own since this requires a CVAE that uses 3D convolutions, for example. We also will carry out more concrete studies on the quality of the reconstruction of the flow by comparing it with other standard methods like bicubic interploation. Added to this, an addition of a custom loss function based on the physics of the turbulent flow at hand is within our interest. Moreover, we plan to design a scheme to study the characteristics of the synthetic turbulence data generated.

A

Design Details and Parameters

A.0.1 Wind turbine dataset simulation parameters

The table below corresponds to the physical parameters used in the simulation of the wind turbine dataset.

Wind turbine simulation parameters	
Parameter	Description
Wind1VelX	Wind velocity in x component
Wind1VelY	Wind velocity in y component
BldPitch1	Blade 1 pitch angle (position)
Azimuth	Rotor azimuth angle (position)
RotSpeed	Rotor azimuth angular speed
Spn3MLxb1	Blade 1 local edgewise moment at span station 3
Spn3MLyb1	Blade 1 local flapwise moment at span station 3
Spn1ALxb1	Blade 1 local flapwise acceleration (absolute) of span station 1
Spn1ALyb1	Blade 1 local edgewise acceleration (absolute) of span station 1
Spn1ALxb2	Blade 2 local flapwise acceleration (absolute) of span station 1
Spn1ALyb2	Blade 2 local edgewise acceleration (absolute) of span station 1
Spn1ALxb3	Blade 3 local flapwise acceleration (absolute) of span station 1
Spn1ALyb3	Blade 3 local edgewise acceleration (absolute) of span station 1
Spn2ALxb1	Blade 1 local flapwise acceleration (absolute) of span station 2
Spn2ALyb1	Blade 1 local edgewise acceleration (absolute) of span station 2
Spn2ALxb2	Blade 2 local flapwise acceleration (absolute) of span station 2
Spn2ALyb2	Blade 2 local edgewise acceleration (absolute) of span station 2
Spn2ALxb3	Blade 3 local flapwise acceleration (absolute) of span station 2
Spn2ALyb3	Blade 3 local edgewise acceleration (absolute) of span station 2
Spn3ALxb1	Blade 1 local flapwise acceleration (absolute) of span station 3
Spn3ALyb1	Blade 1 local edgewise acceleration (absolute) of span station 3
Spn3ALxb2	Blade 2 local flapwise acceleration (absolute) of span station 3
Spn3ALyb2	Blade 2 local edgewise acceleration (absolute) of span station 3
Spn3ALxb3	Blade 3 local flapwise acceleration (absolute) of span station 3
Spn3ALyb3	Blade 3 local edgewise acceleration (absolute) of span station 3
GenPwr	Electrical generator power

References

[1] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 07, pages 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics. ISBN 9780898716245.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Python Awesome Blog. Variational recurrent autoencoder for timeseries clustering in pytorch. `https://pythonawesome.com/variational-recurrent-autoencoder-for-timeseries-clustering-in-pytorch/`. Accessed: 2020-08-04.

[4] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD 96, pages 226–231. AAAI Press, 1996.

[5] Otto Fabius and Joost R. van Amersfoort. Variational Recurrent Auto-Encoders. *arXiv e-prints*, art. arXiv:1412.6581, 2014.

[6] Kai Fukami, Koji Fukagata, and Kunihiko Taira. Super-resolution reconstruction of turbulent flows with machine learning. *Journal of Fluid Mechanics*, 870:106–120, 2019. doi: 10.1017/jfm.2019.238.

[7] Sudhakar Gantasala, Jean-Claude Luneno, and Jan-Olov AidanpÃdÃd. Identification of ice mass accumulated on wind turbine blades using its natural frequencies. *Wind Engineering*, 42(1):66–84, 2018. doi: 10.1177/0309524X17723207. URL `https://doi.org/10.1177/0309524X17723207`.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

[9] K. Grogan. Variational autoencoder for turbulence generation. Discussion paper, Department of Mechanical Engineering, Stanford University, 2017.

[10] Berthold Hahn, Thomas Welte, Stefan Faulstich, Pramod Bangalore, Cyril Boussion, Keith Harrison, Emilio Miguelanez-Martin, Frank OâĂŹConnor, Lasse Pettersson, Conaill Soraghan, Clym Stock-Williams, John Dalsgaard SÃÿrensen, Gerard van Bussel, and JÃÿrn Vatn. Recommended practices for wind farm data collection and reliability assessment for o&m optimization. *Energy Procedia*, 137:358–365, 2017. ISSN 1876-6102. doi: https://doi.org/10.1016/j.egypro.2017.10.360. URL `http://www.sciencedirect.com/science/article/pii/S1876610217353353`.

[11] K. He, G. Gkioxari, P. DollÃąr, and R. Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[12] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *International Conference on Learning Representations (ICLR)*, 2017.

[13] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. ISSN 0036-8075. doi: 10.1126/science.1127647. URL `https://science.sciencemag.org/content/313/5786/504`.

[14] Sepp Hochreiter and JÃijrgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL `https://doi.org/10.1162/neco.1997.9.8.1735`.

[15] Joel Jaskari and Jyri J. Kivinen. A Novel Variational Autoencoder with Applications to Generative Modelling, Classification, and Ordinal Regression. *arXiv e-prints*, art. arXiv:1812.07352, Dec 2018.

[16] Jordan Jeremy. Variational autoencoders. `https://www.jeremyjordan.me/variational-autoencoders/`. Accessed: 2020-08-02.

[17] Sergios Karagiannakos. How to generate images using autoencoders. `https://theaisummer.com/Autoencoder/`. Accessed: 2020-01-12.

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv e-prints*, art. arXiv:1312.6114, Dec 2013.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi: 10.1038/nature14539. URL `https://doi.org/10.1038/nature14539`.

[22] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pages 849–856. MIT Press, 2001.

[23] Lan Huong Nguyen and Susan Holmes. Ten quick tips for effective dimensionality reduction. *PLOS Computational Biology*, 15(6):1–19, Jun 2019. doi: 10.1371/journal.pcbi.1006907. URL `https://doi.org/10.1371/journal.pcbi.1006907`.

[24] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[25] J. Pereira and M. Silveira. Learning representations from healthcare time series data for unsupervised anomaly detection. In *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 1–7, Feb 2019. doi: 10.1109/BIGCOMP.2019.8679157.

[26] JoÃčo Pereira and Margarida Silveira. Unsupervised representation learning and anomaly detection in ecg sequences. *International Journal of Data Mining and Bioinformatics*, 22:389–407, Aug 2019. doi: 10.1504/IJDMB.2019.101395.

[27] Alan P Reynolds, Graeme Richards, Beatriz de la Iglesia, and Victor J Rayward-Smith. Clustering rules: a comparison of partitioning and hierarchical clustering algorithms. *Journal of Mathematical Modelling and Algorithms*, 5(4):475–504, 2006.

[28] David Ruelle. The turbulent fluid as a dynamical system. In *New Perspectives in Turbulence*, pages 123–138. Springer New York, 1991. ISBN 978-1-4612-3156-1.

[29] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL `http://www.jmlr.org/papers/v9/vandermaaten08a.html`.

[30] Xing Victor, Lapeyre Corentin, and Joly Laurent. Exploration of the ability of deep learning to learn the characteristics of turbulent flows. Discussion paper, Centre Europeen de Recherche et de Formation Avancee en Calcul Scientifique (CERFACS), France, 2018.

[31] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[32] Sebastian J. Wetzel. Unsupervised learning of phase transitions: From principal component analysis to variational autoencoders. *Physical Review E*, 96:022140, Aug 2017. doi: 10.1103/PhysRevE.96.022140. URL `https://link.aps.org/doi/10.1103/PhysRevE.96.022140`.

[33] Chunkai Zhang, Shaocong Li, Hongye Zhang, and Yingyang Chen. Velc: A new variational autoencoder based model for time series anomaly detection, 2019.

[34] Hongshan Zhao, Huihai Liu, Wenjing Hu, and Xihui Yan. Anomaly detection and fault analysis of wind turbine components based on deep learning network. *Renewable Energy*, 127:825 – 834, 2018. ISSN 0960-1481. doi: https://doi.org/10.1016/j.renene.2018.05.024. URL `http://www.sciencedirect.com/science/article/pii/S0960148118305457`.