Deep Scenario Generation of Financial Markets

Filip Carlsson, Philip Lindgren
Supervisor: Prof. Henrik Hult
Norron Asset Management

2020

Abstract

The goal of this thesis is to explore a new clustering algorithm, VAE-Clustering, and examine if it can be applied to find differences in the distribution of stock returns and augment the distribution of a current portfolio of stocks and see how it performs in different market conditions.

The VAE-clustering method is as mentioned a newly introduced method and not widely tested, especially not on time series. The first step is therefore to see if and how well the clustering works. We first apply the algorithm to a dataset containing monthly time series of the power demand in Italy. The purpose in this part is to focus on how well the method works technically. When the model works well and generates proper results with the Italian Power Demand data, we move forward and apply the model on stock return data. In the latter application we are unable to find meaningful clusters and therefore unable to move forward towards the goal of the thesis.

The results shows that the VAE-clustering method is applicable for time series. The power demand have clear differences from season to season and the model can successfully identify those differences. When it comes to the financial data we hoped that the model would be able to find different market regimes based on time periods. The model is though not able distinguish different time periods from each other. We therefore conclude that the VAE-clustering method is applicable on time series data, but that the structure and setting of the financial data in this thesis makes it to hard to find meaningful clusters.

The major finding is that the VAE-clustering method can be applied to time series. We highly encourage further research to find if the method can be successfully used on financial data in different settings than tested in this thesis.

Sammanfattning

Syftet med den här avhandlingen är att utforska en ny klustringsalgoritm, VAE-Clustering, och undersöka om den kan tillämpas för att hitta skillnader i fördelningen av aktieavkastningar och förändra distributionen av en nuvarande aktieportfölj och se hur den presterar under olika marknadsvillkor.

VAE-klusteringsmetoden är som nämnts en nyinförd metod och inte testad i stort, särskilt inte på tidsserier. Det första steget är därför att se om och hur klusteringen fungerar. Vi tillämpar först algoritmen på ett datasätt som innehåller månatliga tidsserier för strömbehovet i Italien. Syftet med denna del är att fokusera på hur väl metoden fungerar tekniskt. När modellen fungerar bra och ger tillfredställande resultat, går vi vidare och tillämpar modellen på aktieavkastningsdata. I den senare applikationen kan vi inte hitta meningsfulla kluster och kan därför inte gå framåt mot målet som var att simulera olika marknader och se hur en nuvarande portfölj presterar under olika marknadsregimer.

Resultaten visar att VAE-klustermetoden är väl tillämpbar på tidsserier. Behovet av el har tydliga skillnader från säsong till säsong och modellen kan framgångsrikt identifiera dessa skillnader. När det gäller finansiell data hoppades vi att modellen skulle kunna hitta olika marknadsregimer baserade på tidsperioder. Modellen kan dock inte skilja olika tidsperioder från varandra. Vi drar därför slutsatsen att VAE-klustermetoden är tillämplig på tidsseriedata, men att strukturen på den finansiella data som undersöktes i denna avhandling gör det svårt att hitta meningsfulla kluster.

Den viktigaste upptäckten är att VAE-klustermetoden kan tillämpas på tidsserier. Vi uppmuntrar ytterligare forskning för att hitta om metoden framgångsrikt kan användas på finansiell data i andra former än de testade i denna avhandling.

Acknowledgements

First of all we want to thank Henrik Hult and Adam Lindhe for the opportunity to be among the first people to formally write about VAE-Clustering. It's been an fantastic learning experience with new challenges at each turn.

We would also like to thank Henrik Hult for supervising us in this thesis and Adam Lindhe for insightful discussions.

We want to thank Norron Asset Management for letting us do this thesis with them, providing us with data and insight needed to finish our thesis.

Contents

1 Introduction

1.1 Project Description

Norron is a hedgefund and thus has to have a stable profit from investments irregardless of market conditions. Hedgefunds usually uses different models to asset what would happen to their performance in different scenarios, but these models are sometimes based on unrealistic assumptions or fail to explain the complex dynamics that govern in the financial markets.

During the last few years data generation models have seen tremendous progress, and it's now possible to control the generation process, i.e. incrementally changing attributes such as smile on an image of a human face. The goal of this thesis is to use clustering methods to find differences in the distribution of stock returns and augment the distribution of a current portfolio of stocks and see how it performs in the different market conditions. We make make use of recent work on Variational Autoencoders in order to generate various financial scenarios and investigate portfolio performance in each and see how the developed models can be used to hedge against financial market conditions.

The thesis also contains a investigative part where we examine how a new clustering method, *VAE-clustering*, developed by Henrik Hult and Adam Lindhe performs on time series, and foremost financial time series. The *VAE-clustering* method is newly introduced and not widely tested. The first step is therefore to see if and how well the clustering works. Our approach is to first implement the method on a data-set that is likely to contain clusters. This gives us the ability to test how well the method works. The next step is to implement it on financial data and see if we can gain insights and information from the clustering.

1.2 Methodology and data - VAE-clustering

The VAE-clustering is trained on two datasets. One that we know should generate clusters and one dataset with financial data. The first set contains monthly time series of daily power demand in italy, labeled after winter and summer months. The financial data will be daily stock prices for all the stocks in Norron's investment universe. The method in short is that we first build the model and use it on a data-set that we know should generate clusters. When the model generates the expected clusters from this data-set, we can conclude that it works. The hypothesis behind clustering financial regimes is basically that the distributions differ between different regimes.

1.3 Methodology and data - Scenario generation

The data will be daily stock prices for all the stocks in Norron's investment universe. We will study the data thoroughly and compare it to the generative data in order to see that the data generated truly looks like financial data. In this thesis we will create a Variational Autoencoder model in order to generate the data and explore the latent space to see how the financial time series behaves during different regimes (clusters). The Variational Autoencoder generates distributions which we use to calculate risk measures. That is we obtain risk measures for different regimes.

2 Theoretical Background

2.1 Learning Algorithms

A machine learning algorithm is an algorithm that is able to learn from data. What is then learning? Mitchell [1] provides a concise definition in his book *Machine Learning*: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Machine learning gives us the ability to tackle tasks that are too difficult to solve with fixed programs that are written and designed by humans. Machine learning is in many ways interesting because it helps us develop our understanding of the principles that underlie intelligence. When talking about the word "task", the task is not process of learning itself. Learning is rather attaining the ability to perform the task. For example, if we want a robot to be able to walk, then walking is the task. We could program the robot to learn to walk, or we could attempt to directly write a program that specifies how to walk manually.

To be able to evaluate the abilities of a machine learning algorithm, a quantitative measure of its performance has to be designed. The performance measure P is usually specific to the task T that is being carried out by the system.

Machine learning algorithms can be broadly categorized as *unsupervised* or *supervised* by what kind of experience they are allowed to have during the learning process.
Supervised learning algorithms experience a dataset containing features, but each example is also associated with a label or a target. For example, a dataset with information about flora is annotated with the species of flowers. A supervised learning algorithm can study the flora dataset and learn to classify each flower into three different species based on their measurements.
Unsupervised learning algorithms experience a dataset containing many features, then learn useful properties of the structure of this dataset. In the context of deep learning, we usually want to learn the entire probability distribution that generated a dataset, whether explicitly, as in density estimation, or implicitly, for tasks like synthesis or de-noising. Some other unsupervised learning algorithms perform other roles, like clustering, which consists of dividing the dataset into clusters of similar examples[4].

2.2 Artificial Neural Networks

Deep feedforward networks are the essential deep learning models. The goal of a feedforward network is to approximate some function, f^*. Thus for a classifier, $y = f^*(\boldsymbol{x})$ maps an input $\boldsymbol{x}$ to a category y. A feedforward network defines a mapping $\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta})$ and learns the value of the parameters $\boldsymbol{\theta}$ that results in the best approximates the function.
The reason that feedforward neural networks are called *networks* is because they are typically represented by composing together many different functions. The model is associated with a directed acyclic graph describing how the functions are composed together. For example, we

can have three functions f_1, f_2, and f_3 connected in a chain to form $f(\boldsymbol{x}) = f_3(f_2(f_1(\boldsymbol{x})))$. In this case f_i is called the *i:th layer* of the network. The overall length of the chain gives the *depth* of the model. This is where the name *deep learning* comes from. The final layer in the feedforward network is called the *output layer*. When training a neural network, $f(\boldsymbol{x})$ is driven to match $f^*(\boldsymbol{x})$. From the training we receive noisy, approximated examples of $f^*(\boldsymbol{x})$ evaluated at different training points. Each $\boldsymbol{x}$ comes with a label $y \approx f^*(\boldsymbol{x})$. The training examples specify directly what the output layer must do at each point $\boldsymbol{x}$; it must produce a value that is close to y. The other layers' behaviour is not directly specified by the training data, thus the learning algorithm must decide how to use these layers to produce the desired output. The training data does though not say what each individual layer should do. Instead, the learning algorithm decides how these layers are used to best implement an approximation of f^*. Since the training data does not show the desired output for each layer, these are called *hidden layers*[4].

The reason that these networks are called *neural* is because they are loosely inspired by neuroscience. Each hidden layer of the network is typically vector valued. The dimensionality of these hidden layers determines the *width* of the model. Each element of the vector may be interpreted as playing a role analogous to a neuron. Instead of seeing the layer as a representation of a simple vector-to-vector function, we can see the layer as consisting of several units acting in parallel, where each represents a vector-to-scalar function. Each unit therefore resembles a neuron in the sense that it receives input from many other units and computes its own activation value. The idea of using many layers of vector-valued representations is drawn from neuroscience[4]. However, modern neural network research is guided by many mathematical and engineering disciplines, and the goal of neural networks is not to model the brain. The best way to think of feedforward networks is as function approximation machines that are designed to achieve statistical generalization, occasionally drawing some insights from what we know about the brain, rather than as models of brain function[4].

2.3 Autoencoders

An autoencoder is a neural network that is trained to attempt to copy its input to its output. An autoencoder essentially consists of two parts: an encoder and a decoder. The objective of the AE is to encode a set of data x in a representation referred to as code $\mathbf{z}$, and then be able to use this representation to reconstruct the data, hence attaining $\tilde{\mathbf{x}}$. Hence, the autoencoder will map $\mathbf{x} \mapsto \tilde{\mathbf{x}}$ through the encoder $f(\mathbf{x}) \to \mathbf{z}$ and the decoder $g(\mathbf{z}) \to \tilde{\mathbf{x}}$[4].

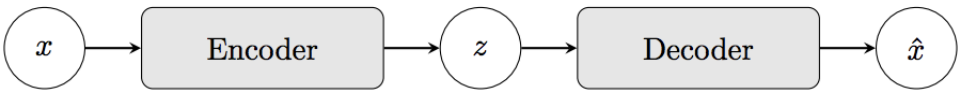


Figure 1: Flow chart of an autoencoder. Grey rectangles represent neural networks. White circles represent input or output data.

Traditionally, autoencoders were used for dimensionality reduction or feature learning. Recently, theoretical connections between autoencoders and latent variable models have brought autoencoders to the forefront of generative modeling[4]. An autoencoder finds a lower-level

representation of higher dimensional data. Therefore, it essentially performs reduction of dimension. It also helps identifying hidden classes in the data, which solves the unsupervised learning problem in some instances. Autoencoders are especially interesting because they tell us things about what kind of jobs neural networks actually does for us. Mathematically a simple autoencoder can be defined as definition 1.

Definition 1. *Autoencoder*
Given an input $x \in \mathbb{R}^d$ we assume that there is a mapping E to $z \in \mathbb{R}^p$ s.t. $E : x \to z$ This mapping is called the encoder and can be defined with an activation function σ, a weight matrix W and a bias term b as

$$z = \sigma(Wx + b)$$

Conversely, we assume that there is a mapping D s.t. $D : z \to x$ which is called the decoder and can be defined in a similar way to the encoder with the corresponding terms $\hat{\sigma}, \hat{W}$ and $\hat{b}$ as

$$\hat{x} = \hat{\sigma}(\hat{W}z + \hat{b})$$

This setup finds some non-linear transformation from x to z and its inverse. In order to train a network to estimate these transformation we need to minimize the reconstruction error of x. This is done by minimizing the loss-function $L(x, \hat{x})$:

$$L(x, \hat{x}) = \|x - \hat{x}\|^2 = \|x - \hat{\sigma}(\hat{W}z + \hat{b})\|^2 = \|x - \hat{\sigma}(\hat{W}(\sigma(Wx + b)) + \hat{b})\|^2 \tag{1}$$

With this in mind, it is not hard to imagine that you can use this to categorize data in an unsupervised way. It seems reasonable that the latent representation z can group some inputs together thereby make a kind of clustering from high dimensional data.

2.4 Variational Autoencoders

When using generative models, one could want to generate a random, new output, that looks similar to the training data, and you can certainly do that with a Variational Autoencoder (VAE). But more often, one would like to alter, or explore variations on data one already has, and not just in a random way either, but in a desired, specific direction. This is where VAEs work better than other methods currently available.
Variational Autoencoders have one fundamentally unique property that separates them from vanilla autoencoders, and it is this property that makes them so useful for generative modeling: their latent spaces are, by design, continuous, allowing easy random sampling and interpolation.

Variational Autoencoders, initially described by Kingma and Welling in their 2014 paper "Auto-encoding Variational Bayes", can be used when we are trying to find a distribution of some underlying process rather than a function. VAEs are the result of combining variational Bayesian methods with the flexibility and scalability provided by neural networks.

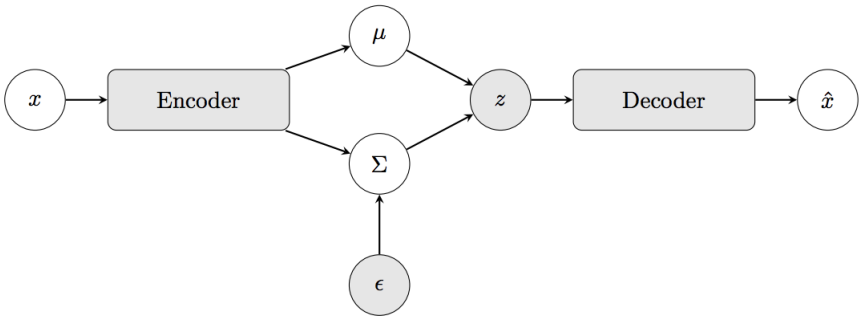


Figure 2: A flow chart describing a VAE. The Encoder is a neural network that mapping the input, x, to the latent space, z. This is done by mapping the mean and variance of these points with the stochastic layer described by ϵ, μ and σ. The Decoder is another neural network that takes us back to the input space and creates a reconstruction of the input in $\hat{x}$.

A VAE is essentially a stochastic generalization of the AE. Here the encoder and the decoder are instead comprised of stochastic mappings. The analogue to the encoding function $f(\boldsymbol{x})$ is an encoding distribution $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$, and the analogue to the decoding function $g(\boldsymbol{z})$ is a decoding distribution $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$, where ϕ denotes the recognition model parameters and θ denotes the generative model parameters[2]. The process can be explained as following:

Consider an i.i.d dataset $\boldsymbol{x} = \{x^{(1)}, x^{(2)}, ..., x^{(n)}\}$ where each datapoint $x^{(i)}$ is generated from $\boldsymbol{z}$ that is drawn from the prior $p_{\theta_t}(\boldsymbol{z})$. The true parameter θ_t is hidden along with the latent variables $\boldsymbol{z}^{(i)}$. The goal is to estimate θ of these parameters by maximizing the probability $p_\theta(\boldsymbol{x})$, which is given by law of total probability

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z}) d\,\boldsymbol{z}\,. \tag{2}$$

That is increase of the model to produce datapoints in the training set, and in doing so increase the probability of generating other similar samples. However, the likelihood is intractable along with the posterior, which is given by Bayes's theorem:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z} \mid \boldsymbol{x}) p_{\boldsymbol{\theta}}(\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z})}. \tag{3}$$

The true intractable posterior is instead approximated with the recognition model $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ and the parameter ϕ is jointly learned with θ.

2.4.1 ELBO

The optimization objective of the variational autoencoder, is the *evidence lower bound*, ELBO. For any choice of recognition model $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$, including the choice of variational parameters we have:

$$
\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}) =& \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x})\right] \\
=& \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right] \\
=& \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right] \\
=& \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]\right]}_{=\mathcal{L}_{\theta,\phi}(\mathbf{x})} + \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}\right]\right]}_{=D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})\right)}.
\end{aligned}
\tag{4}
$$

The second term in the last part is the Kullback-Leibler divergence, D_{KL}, between $q_\phi(\boldsymbol{z}\,|\,\boldsymbol{x})$ and $p_\theta(\boldsymbol{z}\,|\,\boldsymbol{x})$. This term is non-negative

$$
D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\right) \geq 0 \tag{5}
$$

and zero iff $q_\phi(\boldsymbol{z}\,|\,\boldsymbol{x})$ equals the true posterior distribution. The term denoted as $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ is the ELBO.

$$
\mathcal{L}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathbf{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) - \log q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\right] \tag{6}
$$

due to the non-negativity of D_{KL}, the ELBO is a lower bound on the log-likelihood of the data[3]. I.e

$$
\begin{aligned}
\mathcal{L}_{\theta,\phi}(\mathbf{x}) &= \log p_\theta(\mathbf{x}) - D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})\right) \\
&\leq \log p_\theta(\mathbf{x})
\end{aligned}
\tag{7}
$$

In order to estimate the parameters θ and ϕ we have to minimize the ELBO, $\mathcal{L}_{\theta,\phi}$, with respect to these parameters. Therefore, we meet a problem as the first term of ELBO is stochastic.

$$
\phi^*, \theta^* = \underset{\phi,\theta}{\operatorname{argmax}}\, \mathcal{L}_{\theta,\phi}(\boldsymbol{x}) \tag{8}
$$

This means that we can't optimize it right away and need to perform the *reparameterization trick* as in *Welling & Kingma* (2014)[2].

2.4.2 Reparameterization trick

The essential parameterization trick can be described as following. Let $\boldsymbol{z}$ be a continuous random variable, and $\boldsymbol{z} \sim q_\phi(\boldsymbol{z}\,|\,\boldsymbol{x})$ be some conditional distribution. It is then often possible to express the random variable $\boldsymbol{z}$ as a deterministic variable $\boldsymbol{z} = g_\phi(\cdot, \boldsymbol{x})$, where ϵ is an auxiliary variable with independent marginal $p(\epsilon)$, and $g_\phi(\cdot)$ is some vector-valued function parameterized by ϕ [2].

This reparameteterization is useful in our case thus it can be used to rewrite an expectation w.r.t $q_\phi(\boldsymbol{z}\,|\,\boldsymbol{x})$ such that the Monte Carlo estimate of the expectation is differentiable w.r.t ϕ. This can be proved as following:
Given the deterministic mapping $\boldsymbol{z} = g_\phi(\boldsymbol{\epsilon}, \boldsymbol{x})$ we know that

$$
q_\phi(\boldsymbol{z}\,|\,\boldsymbol{x}) \prod_i dz_i = p(\boldsymbol{\epsilon}) \prod_i d\epsilon_i. \tag{9}
$$

Therefore

$$\int q_\phi(\mathbf{z}|\mathbf{x})f(\mathbf{z})d\mathbf{z} = \int p(\boldsymbol{\epsilon})f(\mathbf{z})d\boldsymbol{\epsilon} = \int p(\boldsymbol{\epsilon})f\left(g_\phi(\boldsymbol{\epsilon},\mathbf{x})\right)d\boldsymbol{\epsilon}. \tag{10}$$

It follows that a differentiable estimator can be constructed:

$$\int q_\phi(\mathbf{z}|\mathbf{x})f(\mathbf{z})d\mathbf{z} \simeq \frac{1}{L}\sum_{l=1}^{L} f\left(g_\phi\left(\mathbf{x},\boldsymbol{\epsilon}^{(l)}\right)\right) \tag{11}$$

where

$$\boldsymbol{\epsilon}^{(l)} \sim p(\boldsymbol{\epsilon})$$

as in [2].

2.5 The VAE model

Putting the methods in Section 2.4 this together yields the VAE model.

2.6 Clustering

Cluster analysis is as touched on earlier a form of unsupervised learning algorithms. Cluster analysis has a variety of goals, all related to grouping or segmenting a collection of objects into subsets or *"clusters"*, such that those within each cluster are more closely related to one another than objects assigned to different clusters [15].

Clustering differs from classification and regression, which analyze class-labeled data sets, in that clustering analyzes data objects without consulting class labels. In many cases, class labeled data does not exist naturally. Clustering can be used to generate class labels for a group of data. The objects are clustered or grouped based on the principle of *maximizing the intraclass similarity and minimizing the interclass similarity.* That is, clusters of objects are formed so that objects within a cluster are similar to each other, but are rather dissimilar to objects in other clusters. Each cluster so formed can be viewed as a class of objects, from which rules or observations can be derived[16].

Examples of popular clustering methods are *K-means* and *DBSCAN* described later in this section.

2.6.1 VAE Clustering

In this part we will establish a form of pre-processing of clustering algorithm based on ideas and research by Henrik Hult and Adam Linde. The idea is basically to find minimi in the energy landscape. The energy function of a distribution can be defined as:

$$F(x) = -\log p(x) \tag{12}$$

The lower the energy of an observation, the higher is its probability. Finding the probabilities with the highest probability is the same as minimizing the level of surprise in the observation. The minimi is found by gradient decent, i.e we take small steps in space

$$x_{n+1} = x_n - \nabla F(x) = x_n + \nabla log(p(x)). \tag{13}$$

Therefore we want an analytical expression for the gradient

$$\nabla \log(p(x_i)) = \frac{1}{p(x_i)} \nabla p(x_i) = \frac{1}{p(x_i)} \int \nabla p(x_i|z) p(z) dz. \tag{14}$$

Notice that the integral can be viewed as the expectation value $\mathbb{E}_{p_Z(z)} \left[\nabla p_{X|Z}(x_i|z) \right]$. Taking a draw from the prior will not be informative enough tough, instead we use importance sampling. With the integral rewritten as:

$$\int \nabla p_{X|Z}(x_i|z)\, p_Z(z) dz = \int \nabla p_{X|Z}(x_i|z) \frac{p_Z(z)}{p_{Z|X}(z|x_i)} p_{Z|X}(z|x_i)\, dz. \tag{15}$$

Since this expression is equal to the expectation, $\mathbb{E}_{p_{Z|X}(z|x_i)} \left[\nabla p_{X|Z}(x_i|z) \frac{p_Z(z)}{p_{Z|X}(z|x_i)} \right]$, we can make draws from the likelihood, $p_{Z|X}(z|x_i)$.
Equation (14) together with (15) and Monte Carlo approximation yields

$$\nabla F(x_i) = -\frac{1}{p_X(x_i)} \frac{1}{m} \sum_{j=1}^{m} \nabla p_{X|Z}(x_i|z_j) \frac{p_Z(z_j)}{p_{Z|X}(z_j|x_i)}. \tag{16}$$

Looking at a single pair of observationns x_i, z_j, and using the fact that the gradient is only enforced on $p_{X|Z}$ which we assume follows a Gaussian distribution, (16) can be written as

$$\begin{aligned} -\frac{p_Z(z_j)}{p_X(x_i)\, p_{Z|X}(z_j|x_i)} \nabla p_{X|Z}(x_i|z_j) &= -\frac{p_Z(z_j)}{p_Z(z_j)\, p_{X|Z}(x_i|z_j)} \nabla p_{X|Z}(x_i|z_j) \\ = \frac{\nabla p_{X|Z}(x_i|z_j)}{p_{X|Z}(x_i|z_j)} &= \frac{-\nabla_X (1/\sqrt{2\pi\sigma(z_j)^2}) \exp\left(-\frac{(x_i-\mu(z_j))^2}{2\sigma(z_j)^2}\right)}{(1/\sqrt{2\pi\sigma(z_j)^2}) \exp\left(-\frac{(x_i-\mu(z_j))^2}{2\sigma(z_j)^2}\right)} \\ &= \frac{x_i - \mu(z_j)}{\sigma(z_j)^2}. \end{aligned} \tag{17}$$

Note that the likelihood $p_{X|Z}$ is the decoder part of the variational autoencoder. We can now conclude that taking a step towards a local minima of $F(x)$, can be derived as

$$x_{i+1} = x_i - \frac{1}{m} \sum_{j=1}^{m} \frac{x_i - \mu(z_j)}{\sigma(z_j)^2}. \tag{18}$$

When we have stabilized at a local minima, we view this as the most likely situation, i.e a raw underlying signal of the specific cluster we have found.

To gain a better intuition we describe the process in the two dimensional case in the illustrations below. Before the VAE clustering the points are distributed over the energy landscape. See left in Figure 3. What we want to do is simply to force the points down in the closest valley. See right in Figure 3. In that way we are able to see how many clusters we want in the next clustering step. For example K-means or DBSCAN clustering.

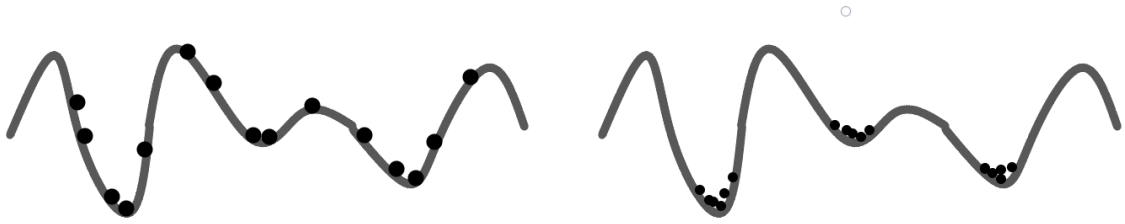


Figure 3: Points distributed in the energy landscape before and after VAE clustering.

If our energy landscape have got a lot of small peaks and valleys, the VAE clustering will probably suggest more clusters than we actually want. A way to fix this problem is to smoothing out the energy landscape by adding noise. We can simply show mathematically that adding noice is the same as the convolution between the energy landscape and the energy landscape of the normal distribution, i.e a mollifier.

Let $F(x) = \log p(x)$ be the energy function. We create a smoother version F_δ by convolution with the Gauss function, $\rho_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-x^2}{2\sigma^2}}$, i.e

$$F_\delta(x) = \int_{\mathbb{R}} F(y)\rho_\sigma(x-y)dy. \tag{19}$$

We see that this is the same as the expected value of $F(x+\xi)$ where $\xi \sim N\left(0, \epsilon^2\right)$, that is

$$F_\delta(x) = E_{\xi\sim N(0,\epsilon^2)}[F(x+\xi)]. \tag{20}$$

The gradient of F_δ can be approximated with a Monte-Carlo estimate

$$\boldsymbol{\nabla} F_\delta(x) = \boldsymbol{\nabla} E_{\xi\sim N(0,\epsilon^2)}[F(x+\xi)] = E_{\xi\sim N(0,\epsilon^2)}[\boldsymbol{\nabla} F(x+\xi)] \approx \boldsymbol{\nabla} F(x+\xi) \tag{21}$$

where $\xi \sim N\left(0, \epsilon^2\right)$. Now we take the gradient step in $\nabla F_\delta(x)$ direction

$$\hat{x} = x_n + \xi \quad \xi \sim N\left(0, \epsilon^2\right)$$

$$x_{n+1} = x_n - \boldsymbol{\nabla} F_\delta\left(x_n\right) \approx x_n - \boldsymbol{\nabla} F(\hat{x}).$$

The algorithm can be summarized as following:

Data: x_0, fitted VAE model, α, ϵ, N;
Result: x_N;
while $i \leq N$ **do**
  $x_z = x_i + \xi$
  $z_i = q_\phi(z_i|x_z)$
  $\mu_i, \sigma_i = p_\theta(x_z|z_i)$
  $\boldsymbol{\nabla} F_i = \frac{x_z - \mu_i}{\sigma_i^2}$
  $x_n = x_i - \boldsymbol{\nabla} F_i * \alpha$
  $\alpha = \alpha/\sqrt{i+1}$
  $\epsilon = \epsilon/\sqrt{i+1}$
end

Algorithm 1: VAE clustering algorithm

In this example we used the fact that if $X \sim N(\mu, \sigma^2)$ then $\boldsymbol{\nabla} F = \frac{X-\mu}{\sigma^2}$.

2.6.2 K-means clustering

In the k-means clustering we partition a dataset up into k-clusters in such a way that a loss function is minimized. This is done by finding the means of these clusters and classify points as belonging to a cluster based on the Euclidean distance to its mean. If a point is categorized as part of the cluster, the mean is updated and the algorithm is repeated[6]. A way to perform k-means clustering is Lloyd's algorithm. It works by alternating between two steps, assignment and update as in algorithm 1.

Data: Your dataset, number of clusters k;
Result: Cluster label for each point;
Randomly initialize a set of k means $\{m_1^1, ..., m_1^k\}$;
while *Assignment* $S_i^t \neq S_i^{t-1}$ **do**
  for *Every point* x_i *in the dataset* **do**
    $S_i^t = \left\{ x_p : \left\| x_p - m_i^t \right\|^2 \leq \left\| x_p - m_j^t \right\|^2 \quad \forall j, 1 \leq j \leq k \right\}$;
    $m_i^t = \left| S_i^t \right|^{-1} \sum_{x_j \in S_i^t} x_j$;
  end
end

Algorithm 2: Lloyd's algorithm (k-means)

Here k is chosen from the VAE-clustering, as described earlier.

2.6.3 DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) is a density-based clustering non-parametric algorithm, i.e given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away)[9]. The algorithms takes two parameters, *minPts* and ϵ, and clusters the data points with regard of those.

To find a cluster, DBSCAN starts with an arbitrary point p and retrieves all points density-reachable from p wrt. ϵ and $MinPts$. If p is a core point, this procedure yields a cluster wrt. ϵ and $MinPts$. If p is a border point, no points are density-reachable from p and DBSCAN visits the next point of the database.

Algorithm

Let N be the number of points in some space, and let ϵ be the radius around some point. In the DBSCAN-algorithm, each point can be classified as:

- If at least $minPts$ points are within distance ϵ of p, p is said to be a *core point*
- If q is within distance ϵ from core point p, q is *directly reachable* from p, Points are only said to be directly reachable from core points.
- q is reachable from p if there is a path $p_1, ..., p_N$ with $p_1 = p$ and $p_N = q$, where each p_{i+1} is directly reachable from p_i. That is, all points on the path must be core points, with the exception of q.
- Two points p and q, are said to be *density-connected* if there is a point o such that both p, q are reachable from o. This is a symmetric relation.
- Points which are not reachable from any other point are denoted *outliers* or *noise points*.

With this in mind we can now define a cluster. Let p be a core point, p then forms a cluster together with all points that are reachable from it. Therefore each cluster must contain at least one core point. Non-core points form the edge of the cluster, thus they are part of the cluster, but cannot be used to reach further points. All clusters will now satisfy two properties;

1. All points in a cluster are mutually density-connected.
2. If a point q is reachable from some other point of a cluster, then q is part of the same cluster.

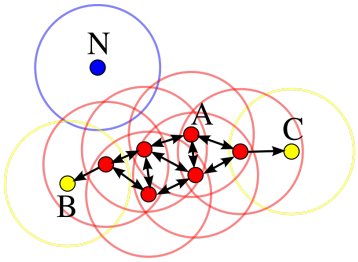


Figure 4: In this figure, minPts = 4. Point A and the other red points are core points, because the area surrounding these points in an ϵ radius contain at least 4 points (including the point itself). Because they are all reachable from one another, they form a single cluster. Points B and C are not core points, but are reachable from A (via other core points) and thus belong to the cluster as well. Point N is a noise point that is neither a core point nor directly-reachable.

Source: wikipedia.org/wiki/DBSCAN

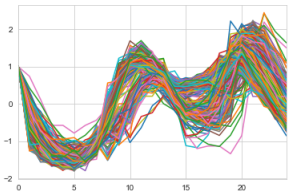
3 Methodology

The purpose of this chapter is to give a walk-through of how we build the model based on the principles in the Theoretical Background. The method in short is that we first build the model and use it on a data-set that we know should generate clusters. When the model generates the expected clusters from this data-set, we can conclude that it works.

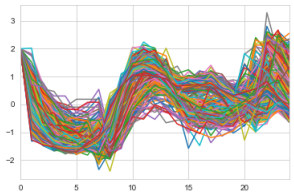
3.1 Data

3.1.1 Italy Power Demand

The UCR Times series Classification Archive is an archive of time series used to test classification algorithms.
The dataset *ItalyPowerDemand* contains 1096 time series of length 24, divided into 2 classes. The time series are labeled depending on if they are from the winter months or summer months, and should therefore be somewhat different from each other, as can be seen in the figures below.



(a) Class 1



(b) Class 2

Figure 5: Italy Power Demand

The data set is split into train and test data with a ratio of 80% in training and 20% in testing. Having done this we standardize both the train and test data sets using the mean and standard deviation of the training data set.

3.1.2 Financial Data

This data set consists of daily log returns for all stocks that are traded in Sweden, Norway, Finland and Denmark during the period 2000-01-01 to 2020-03-31. The data is later split into windows of 120 trading days and the stocks that weren't listed during that time are removed. The resulting data set consists of about 25000 time series, each of length 120. We also create labels to indicate how the SBX stock index has performed during the 120 day trading window. The performance is split up into 5 different labels according to the table below.

Performance	Label
$\geq 20\%$	0
$\leq 20\%$ and $\geq 10\%$	1
$\leq 10\%$ and $\geq -10\%$	2
$\leq -10\%$ and $\geq -20\%$	3
$\leq -20\%$	4

As with the Italy Power data set we split the data into train and test data with a ratio of 80% and 20%, respectively.

3.2 Model Architectures

In both models we have assumed that the data X, given Z is a sample from a normal distribution with some μ and some σ, i.e.

$$p_\theta(x|z) = N(x; \mu, \sigma^2)$$

and thus our reconstruction loss is the negative log likelihood of a normal distribution.

3.2.1 Italy Power Demand Model

The network structure used for the Italy Power data set is the following:

	Layer	Activation Function	Input Shape	Output Shape
Encoder:	Linear	ELU	24	64
	Linear	ELU	64	64
	Linear	ELU	64	16
	Latent	-	16	2

	Layer	Activation Function	Input Shape	Output Shape
Decoder:	Linear	ELU	2	16
	Linear	ELU	16	64
	Linear	ELU	64	64
	Normal	-	64	24

The "latent" layer transform the input to a 2-dimensional normal distribution, i.e. the latent space of the VAE model and the normal layer returns a normal distribution with some mean μ and standard deviation σ.

3.2.2 Financial Data Model

The network structure used for the Financial data set, similar to the model above, is the following:

	Layer	Activation Function	Input Shape	Output Shape
Encoder:	Linear	ELU	120	256
	Linear	ELU	256	128
	Linear	ELU	128	64
	Linear	ELU	64	16
	Latent	-	16	2

	Layer	Activation Function	Input Shape	Output Shape
Decoder:	Linear	ELU	2	16
	Linear	ELU	16	64
	Linear	ELU	64	128
	Linear	ELU	128	256
	Normal	-	256	120

3.3 Clustering

As described in Theoretical Background, the VAE includes a probabilistic decoder in form of a posterior $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$. The gradient of the negative logarithm of this posterior is used in the gradient decent equation (13).
The VAE-clustering can be seen as a pre-processing for other clustering methods such as K-means clustering. Thus the VAE-clustering gives us a hint of how many clusters we expect to find while also making it easier for the K-means algorithm to find the actual clusters.

First we fit our model to the data. Then we follow the VAE-Clustering algorithm to cluster the data. The clustering algorithm will gradually transform each data point towards the center of each likelihood maxima in both the X-space and latent space. Thus it will be easier for us to spot how many clusters there are, and also easier for the K-means algorithm to actually find the different clusters as the points in a specific cluster will be pushed towards each other. The choice of ϵ decides how much we want to smooth the energy landscape. Thus, the larger ϵ the fewer clusters we will have.
After the data points has been shifted towards their local likelihood maximas using the VAE Clustering algorithm we apply K-means to the data, which gives us a label for each cluster.

3.4 Italy Power Demand

The *VAE-clustering* method is as mentioned earlier a newly introduced method and not widely tested, especially not on time series. The first step is therefore to see if and how well the clustering works. We first apply the algorithm to the *ItalyPowerDemand* which contains monthly time series of the power demand in Italy. The time series are labeled after winter months or summer months. It is convenient to start with a data-set with a structure that makes is likely to contain some clusters. The approach here is thus to first implement the method on the Italy Power Demand data. The purpose in this part is thus to focus on how well the method works technically and not to draw any conclusions or insights from the results. Using a data-set from which we know what results we can expect, gives us the ability to evaluate the models performance. We can also test how the result changes when we tune different parameters of the model. This part of the project can be seen as a test and evaluation of the model and its underlying techniques.

3.5 Financial Data

When the model works well and generates proper results with the Italian Power Demand data, we can move forward and apply the model on financial data. The purpose of this part

is to use the model to gain insights and information from the financial data. It is also to demonstrate how the model can be used and its applicability.

4 Results

4.1 Results - Italy Power Demand

The obtain the results using the Italy Power Demand data and model we train the model presented in Section 3.2.1 using the Adam [14] algorithm. The loss on the training and testing data sets is shown in Figure 6 a). We can see that the model does not overfit and the loss on both data sets converges nicely. Figure 6 b) shows that we can reconstruct the data reasonably well.

(a) Train and test loss for Italy Power Demand

(b) Reconstruction

Figure 6: Italy Power Demand loss and reconstruction

Figure 7 and 8 shows the latent space from our VAE trained on the Italy Power Demand data-set before and after VAE-clustering. In both figures the coloring of the data points indicate whether the time series is from a summer or winter month.
We can see that when $\epsilon = 0$ in the VAE-clustering we obtain six clusters, when $\epsilon = 0.65$ we obtain four clusters and when $\epsilon = 1$ we obtain two clusters. In order to further investigate the VAE-clustering algorithm we go ahead with the results using $\epsilon = 0.65$ and apply K-Means on the data set using 4 clusters. Figure 9 shows all the points in the latent space and the colors indicate which cluster they belong to.

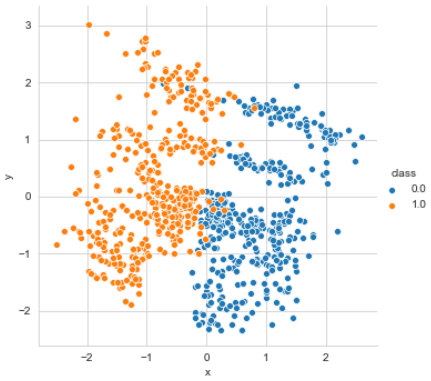
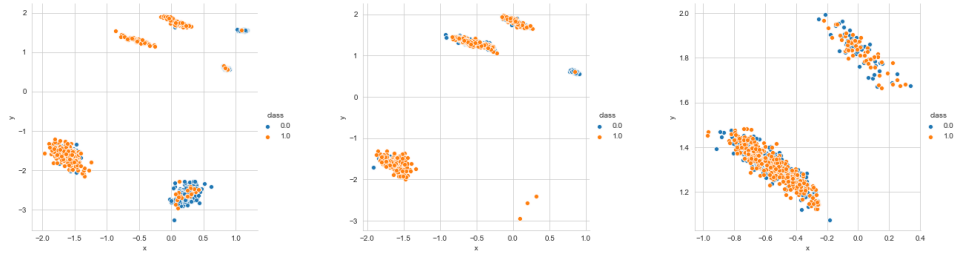


Figure 7: The latent space for Italy Power Demand before clustering.



(a) $\epsilon = 0$ (b) $\epsilon = 0.65$ (c) $\epsilon = 1$

Figure 8: The latent space of Italy Power Demand after clustering for different values of epsilon.

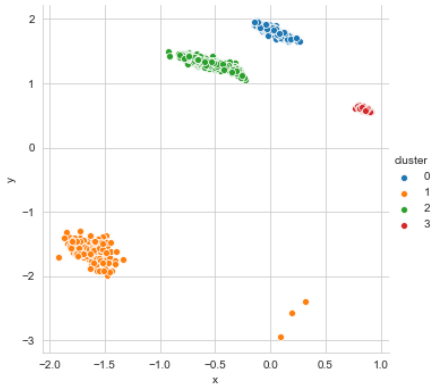
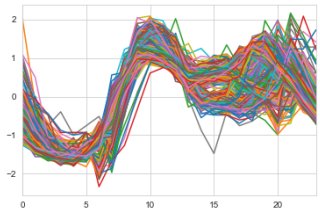


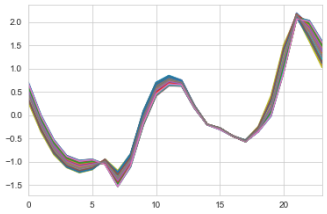
Figure 9: The latent space for Italy Power Demand after VAE-clustering with $\epsilon = 0.65$ and K-Means with $k = 4$.

Figure 10-13 shows the reconstruction of the time series contained in each cluster before and after clustering with $\epsilon = 0.65$. Figure 10-13 a) shows the reconstruction of the time series before VAE-clustering. Figure 10-13 b) shows the reconstruction of the time series after

VAE-clustering, i.e. the time series that K-Means is performed on.

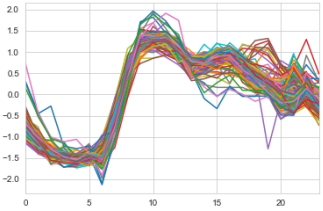


(a) Before clustering

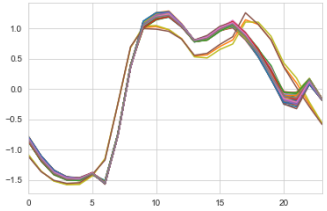


(b) After clustering

Figure 10: Italy Power Demand - Cluster 0

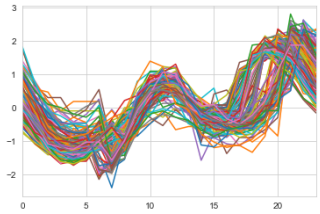


(a) Before clustering

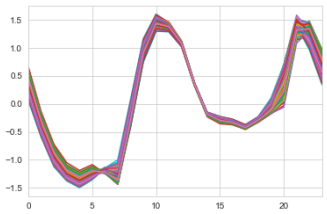


(b) After clustering

Figure 11: Italy Power Demand - Cluster 1



(a) Before clustering



(b) After clustering

Figure 12: Italy Power Demand - Cluster 2

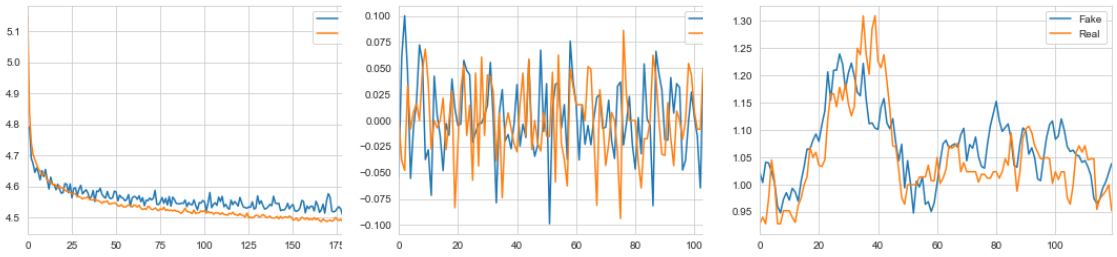
(a) Before clustering

(b) After clustering

Figure 13: Italy Power Demand - Cluster 3

4.2 Results - Financial Data

In Figure 14 a) the train and test loss are shown, using the model presented in Section 3.2.2, trained on the financial returns. As with the Italy Power Demand data set and model we see that the model does not overfit and that the loss converges.
Figure 14 b) displays the reconstruction of a time series and Figure 14 c) is the cumulative return, which makes it easier to see that the model can reproduce the data quite reliably.



(a) Train and test loss for financial returns

(b) Reconstruction of return

(c) Reconstruction of cumulative return

Figure 14: Financial data loss and reconstruction.

Figure 15 shows the latent space of the Financial Data model. The coloring of the points corresponds to the different market regimes defined in Section 3.1.2. We observe that a part of the data points from regime 3 and 4 are somewhat separated from the rest of the points that are centered. Figures 16 a) - c) displays the latent space of the financial data model with different values of epsilon. In all figures the points cluster in one single point at the top of the image, which indicates that the algorithm seems to have found just one cluster.

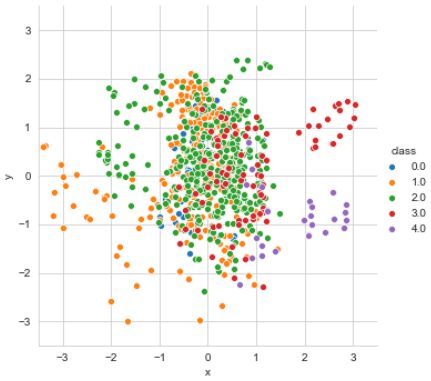
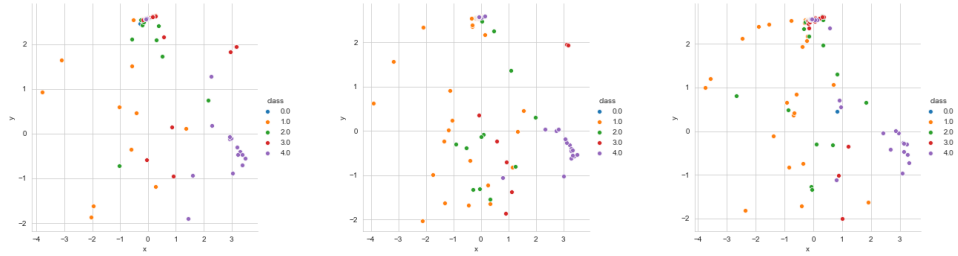


Figure 15: The latent space for the financial returns before clustering.



(a) $\epsilon = 0$ (b) $\epsilon = 0.65$ (c) $\epsilon = 1$

Figure 16: The latent space of financial returns after clustering for different values of epsilon.

5 Discussion

5.1 Discussion - Italy Power Demand

Figure 7 and 8 shows the latent space from our VAE trained on the Italy Power Demand data-set before and after clustering. It is evident that the VAE manages to map the time series into meaningful clusters. When using $\epsilon = 0.65$ we can identify four distinct clusters. We observe that two of the clusters contains points from both classes. Each time series in the data-set is the power demand for each day in a month, and it is classified if it is a winter or summer month. One can therefore conclude that each cluster represents a season. The two clusters containing both classes then represents spring and autumn. Even if the data-set was constructed such that one should find two clusters, one for each class, it is not strange that we found four. This rather shows strength than weakness in our method. Two spring months on each side of the half-year probably have more in common than the last month of the winter half year and a mid winter month.
Figure 10 - 13 shows the reconstruction of the time series contained in each cluster before and after clustering. One can clearly see that time series that are in the same cluster look quite similar before clustering. One can also see that the more dense the cluster, the more similar the time series. There are also some differences in characteristics of the time series for each cluster. In the after clustering figures one can see that the clustering pushes the time

series to look more similar. That is, they are pushed towards the "most likely" time series in the cluster.

Figure 8 demonstrates the smoothing of the energy landscape as described in VAE clustering (Section 2.6.1). The latent space is plotted after clustering with different epsilons, ϵ. A higher value on epsilon corresponds to a more smooth energy landscape. When a higher value on epsilon is used, the model generates fewer clusters, which is expected.

The results from clustering with the Italy Power Demand data indicates that the VAE-clustering method works as desired.

5.2 Discussion - Financial Data

Figure 15 and 16 shows the latent space from our VAE trained on the financial data-set with returns before and after VAE-clustering for different values of epsilon. We observe that a very few points get clustered together and the rest are seen as outliers. That is, we cannot identify meaningful clusters.

In the model used in the result section we used a 2-dimensional latent layer for visualisation purpose. A larger latent layer can hold more information and expanding the latent layer can give the model more "categories" to cluster. We have though tested several sizes on the latent layer and the model still fails to recognise differences between returns.

The results from the Italy Power Demand indicates that the model works and is able to find meaningful clusters. The reason that we can't find meaningful clusters in the financial data can therefore depend on the structure and setting of the data. During this study we have been experimenting with different settings as well, such as using the cumulative return, geometric mean and simple wavelet transforms. Out of the those three the only prepossessing method that found meaningful was using the wavelet transform. We chose not to include that analysis in this thesis due to the fact that it was hard to transform the time series back in order to simulate from the various clusters.
We also experimented, albeit briefly, on other data sets applicable to finance such as fundamental data. We found that the clustering method worked quite well in that setting, as we found clusters of companies that was in the same sector, or similar sectors.
Therefore, we believe that this clustering method has applications in finance, but in a different setting than the one analysed in this thesis.

From this result follows that we cannot further use clustering methods to find differences in the distribution of stock returns and augment the distribution of a current portfolio of stocks and see how it performs in the different market conditions.

6 Conclusion

The VAE-clustering method works well and finds meaningful clusters when the VAE is trained on the Italy Power Demand data. The results shows that this method is applicable for time series. The power demand have clear differences from season to season and the model can successfully identify those differences. When it comes to the financial data we hoped that the model would be able to find different market regimes based on time periods. The model is though not able distinguish different time periods from each other. We therefore conclude that the VAE-clustering method is applicable on time series data, but that the structure and setting of the financial data in this thesis makes it to hard to find meaningful clusters.

6.1 Further studies

6.1.1 Network architectures

In this thesis we have used simple dense layers in order to estimate $p(x|z)$, but there are other network architectures that are known to better learn how to replicate a time series. The closer we are to estimating $p(x|z)$ the more accurate our gradient will be, and thus the VAE Clustering algorithm will results in more accurate clusters. Therefore, using other networks architectures such as Recurrent Neural Networks (RNN) or Convolutional Neural Networks (CNN) might improve the results.
CNN's have recently shown promise in estimating time series data, for example Deepmind's WaveNet have shown excellent results in audio augmentation and replication [10]. Therefore, using proven network architectures such as the WaveNet might improve the results of the clustering algorithm, but has been outside the scope of this thesis.
RNN's, and especially Long Short-Term Memory networks (LSTM) have had incredible results when dealing with time series data [11] and more recently have been used in generative modelling with good results [12]. In [12], the distribution for $p(x, z)$ is modelled as

$$p(x_{\leq T}, z_{\leq T}) = \prod_{t=1}^{T} p(x_t|x_{<t}, z_{\leq t})p(z_t|x_{<t}, z_{<t})$$

and thus making the gradient of $p(x)$ intractable to find analytically, due to the dependence of $x_{<t}$. Therefore, the gradient in this setting would have to be estimated, which might cause results to differ. But, a natural extension of the VAE Clustering algorithm would be to estimate the gradient of $p(x)$ using the reconstruction loss, thus removing the need for an explicit expression of the gradient.

6.1.2 Pre-processing of stock returns

As mentioned in the discussion we failed to find any specific clusters in the financial data set and that one of the causes might be that one should pre-process the returns somehow. [13] uses Wavelets to reduce the noise of the returns, thus making it easier for the network to find different patterns. Similar de-noising methods could be used which might help the VAE Clustering algorithm to find different clusters.

References

[1] Tom M. Mitchell; *Machine Learning* (1997)

[2] Diederik P. Kingma; Max Welling; *Auto-Encoding Variational Bayes* (2014)

[3] Diederik P. Kingma; Max Welling; *An Introduction to Variational Autoencoders* (2019)

[4] Ian Goodfellow; Yoshua Bengio; Aaron Courville; *Deep Learning* (2016)

[5] Rahul Ratnakar Marathe; Sarah M. Ryan; *On the Validity of the Geometric Brownian Motion Assumption* (2014)

[6] S. P. Loyd; *Least squares quantization in PCM* (1982)

[7] Artidoro Pagnoni; Kevin Liu; Shangyan Li; *Conditional Variational Autoencoder for Neural Machine Translation* (2018)

[8] Henrik Hult, et al.; *Risk and portfolio analysis: Principles and methods.* (2012)

[9] Martin Ester; Hans-Peter Kriegel; Jörg Sander; Xiaowei Xu; *A density-based algorithm for discovering clusters in large spatial databases with noise* (1996)

[10] van den Oord, Dieleman et al; *WaveNet: A Generative Model for Raw Audio* (2016)

[11] Schmidhuber et al.; *LSTM: A Search Space Odyssey* (2015)

[12] Chung, Kastner et al.; *A Recurrent Latent Variable Model for Sequential Data* (2015)

[13] Bao W, Yue J, Rao Y; *A deep learning framework for financial time series using stacked autoencoders and long-short term memory* (2017)

[14] Diedrik P. Kingma; Jimmy Lei Ba; *Adam: A Method for Stochastic Optimization* (2015)

[15] Trevor Hastie; Robert Tibshirani; Jerome Friedman; *The Elements of Statistical Learning* (2009)

[16] Jiawei Han; Micheline Kamber; Jian Pe; *Data Mining - Concepts and Techniques* (2011)