Alma Mater Studiorum - Università di Bologna

Department of Economics

Second-cycle/Master's Degree

in

ECONOMICS

Multivariate Autoregressive Denoising Diffusion Model for Value-at-Risk Evaluation

Presented by:

Edoardo Berti

0000923856

Supervisor:

Prof. Sergio Pastorello

Graduation session of December

Academic year 2020/2021

To Silvia and Carla

Multivariate Autoregressive Denoising Diffusion Model for Value-at-Risk Evaluation

Abstract

Neural time series methods are spreading in the financial sector due to their flexibility and are quickly becoming viable alternatives to classical approaches. In this work, we successfully apply an Autoregressive Denoising Diffusion Model, namely Timegrad, to the problem of modeling stock returns and estimating the Value-at-Risk. The utility of this new method is illustrated by VaR Backtesting and empirical analyses on a portfolio of eight equally weighted real stock indices. The study findings show that multivariate autoregressive deep generative models such as Timegrad are able to produce realistic multivariate simulations for portfolios of multiple stocks, giving a reliable tool in the risk management departments that deserves further exploration and analysis.

Contents

1 Introduction **5**
1.1 Motivation and Contributions . . . 5
1.2 Acknowledgements . . . 6

2 Value at Risk **7**
2.1 Introduction to Value at Risk . . . 7
2.2 VaR Forecasting . . . 8
2.2.1 Historical Simulation (HS) . . . 9
2.2.2 Exponential weighted moving average (EWMA) . . . 9
2.2.3 Generalised autoregressive conditional heteroscedasticity (GARCH) . . . 10
2.2.4 Multivariate GARCH (DCC-GARCH) . . . 11
2.2.5 BEKK . . . 12
2.3 Backtesting VaR . . . 13
2.3.1 Binomial Test . . . 13
2.3.2 Traffic Light Test . . . 13
2.3.3 Kupiec's Proportion of Failure (POF) Test . . . 14
2.3.4 Christoffersen's Interval Forecast (IF) Test . . . 14
2.3.5 Haas's Time Between Failures (TBFI) Test . . . 15

3 Background **16**
3.1 Deep Feedforward Networks . . . 16
3.2 Sequence Modeling: Recurrent Neural Networks . . . 17
3.3 Long Short Term Memory . . . 19
3.4 Stochastic Gradient Descent . . . 22
3.4.1 ADAM . . . 23
3.4.2 Stochastic gradient Langevin dynamics . . . 24

4 Denoising Diffusion Probabilistic Models **26**
4.1 Introduction . . . 26
4.2 Architecture . . . 27

5 Timegrad **30**
5.1 Introduction . . . 30
5.2 The Model . . . 30
5.3 Training . . . 31
5.4 Inference . . . 32

6 Timegrad VaR **33**
6.1 Architecture . . . 33
6.2 Dataset . . . 35
6.3 Preprocessing . . . 35
6.4 Hyperparameter Tuning . . . 37
6.5 Results . . . 37
6.5.1 Timegrad Predictions . . . 38
6.5.2 Backtest results . . . 39

7 Conclusion **48**
7.1 Summary . . . 48
7.2 Further developments . . . 49

List of Figures

1 Value at Risk 8
2 Feedforward Neural Network Structure 17
3 Recurrent Neural Network Structure 18
4 The Standard RNN Architecture 20
5 The LSTM Architecture. 20
6 The *forget gate*. 21
7 The *input gate*. 21
8 The updated *Cell state*. 21
9 The *output gate*. 22
10 Diffusion Model Example 26
11 Diffusion Model Representation 27
12 Timegrad Schematic 31
13 Network Architecture 34
14 Index Cumulative Returns 35
15 Heatmap of the correlation matrix 36
16 Windows Slicing Process 36
17 Preprocessed Time Series of Portfolio's Indexes returns 42
18 TimeGrad Prediction Intervals over the rolling windows scenario 43
19 VaR estimates at 95% Confidence level for each index in our portfolio 44
20 VaR estimates at 99% Confidence level for each index in our portfolio 45
21 VaR estimates at 95% Confidence level for FTSE MIB 46
22 VaR estimates at 95% Confidence level for FTSE MIB 47

List of Tables

1 TimeGrad Metrics Evaluation 38
2 GP-Copula Metrics Evaluation 38
3 Backtesting Results for 95% VaR on FTSE MIB 39
4 Summary of the final test results on FTSE MIB 40
5 Backtesting Results for daily 95% VaR on FTSE MIB 40
6 Summary of the final test results on FTSE MIB 41

1 Introduction

1.1 Motivation and Contributions

Today, artificial intelligence (AI) is a growing field with many practical applications and active research topics. Machine Learning (ML) techniques are now actively applied to automate routine labor, understand speech or images, make diagnoses in medicine and practical active research.

Recent advances in artificial neural networks and deep learning allow to use deep learning in time-series forecasting, which provides the ability to process a large amount of data (e.g., a larger portion of the temporal data). Time-series forecasting methods are designed to take cross-temporal relations into account. Classical studies on time-series forecasting focus on linear prediction models such as autoregressive (AR), moving average (MA) and auto-regressive integrated moving average (ARIMA) models where a linear function of past observations is used to predict the future values (Box et al. 2015). However, classical time series forecasting methods such as those in (Hyndman Athanasopoulos, 2018) typically provide univariate point forecasts and are trained individually on each time series in a data set which does not scale with millions of series. Deep learning based time series models (Benidis et al., 2020) are popular alternatives due to their end-to-end training of a global model, ease of incorporating exogenous covariates, and automatic feature extraction abilities. The task of modeling uncertainties is of vital importance for downstream problems that use these forecasts for (business) decision making. More often the individual time series for a problem data set are statistically dependent on each other. Ideally, deep learning models need to incorporate this inductive bias in the form of multivariate (Tsay, 2014) probabilistic methods to provide accurate forecasts. In highly complex systems characterized by non-stationary time series such as those found in financial markets, we want to apply deep generative models to understand the data generating process of single and multiple time series in a better way.
Deep learning applications to the financial world are spreading tremendously in the last few years (Montantes, 2020) due to their reliability in learning high dimensional data distributions. Nevertheless, modelling the full predictive distribution requires low-rank approximations or restrictive assumptions to handle the untractable true data distribution due to the computational and flexibility cost it requires. Energy-based models (EBM) (Hinton, 2002; LeCun et al., 2006), on the other hand, are much less restrictive in terms of functional form. They approximate the non-Gaussian log-probability so that density estimation reduces to a non-linear regression problem. EBMs have been shown to perform well in learning high dimensional data distributions at the cost of being difficult to train (Song Kingma, 2021).

In this work, we make use of an autoregressive EBM to solve the multivariate probabilistic time series forecasting problem in financial time series via a model called *TimeGrad*. This project work has been developed in collaboration with Axyon AI company, a Modena-based leading player in deep learning for time series forecasting and AI asset management. The predicted multivariate

distribution obtained with Timegrad can be leveraged in the risk management framework. Market risk is the risk of losses in positions arising from movements in market prices and one of the main measure to deal with this kind of financial risk is *Value-at-Risk*. Today, VaR is still a crucial element in risk management, as it has always been adopted by the Basel Committee on Banking Supervision (BCBS) regulations. Moreover, an accurate and reliable estimate of the VaR is an opportunity to hedge portfolios by the financial institutions. Some competitive generative models for VaR evaluation can be found in (Zhu, 2020) with GAN or in (Xu, 2016) where a quantile autoregression neural network (QARNN) model is proposed. In this work we try to model VaR estimation in a multivariate case using Timegrad. The interest to look for a multivariate probabilistic model is that stocks in a portfolio usually have some kind of correlation (see Section 6.2), hence more information can be extracted from the data with respect to univariate models.

The main contributions of this report are the following:

- We train an autoregressive Denoising Diffusion Model on (financial) multivariate time series, the model is able to generate a conditional underlying distribution of future time steps and from which we are able to compute VaR forecast. This VaR estimation is tested and it is proved to be superior with respect to baseline methods and a competitive alternative against GARCH models (both univariate and multivariate).
- To the best of our knowledge, this work is the first attempt to adapt Denoising diffusion models to risk management analysis. Further research should focus on optimizing the tuning of hyperparameters and compare it with other Deep Learning models in the field.

1.2 Acknowledgements

This work was supported by the FF4EuroHPC: HPC Innovation for European SMEs, Project Call 1. The FF4EuroHPC project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 951745.

2 Value at Risk

This work will focus on the Value at Risk methodology for hedging against market risk. In this section, we will define the concept of VaR, its traditional estimation methods, the backtesting but also its drawbacks. These classical methods will constitute the baseline against our Timegrad VaR implementation.

2.1 Introduction to Value at Risk

The basic concept was summarized by Linsmeier and Pearson (1996):

> Value at risk is a single, summary, statistical measure of possible portfolio losses. Specifically, value at risk is a measure of losses due to 'normal' market movements. Losses greater than the value at risk are suffered only with a specified small probability. Subject to the simplifying assumptions used in its calculation, value at risk aggregates all of the risks in a portfolio into a single number suitable for use in the boardroom, reporting to regulators, or disclosure in an annual report. Once one crosses the hurdle of using a statistical measure, the concept of value at risk is straightforward to understand. It is simply a way to describe the magnitude of the likely losses on the portfolio. (Linsmeier and Pearson (1996, p. 3))

It has indeed two important characteristics. The first is that it provides a consistent measure of risk across different positions and risk factors. As an example, it enables us to measure the risk associated with a fixed-income position in such a way that it is comparable with the risk of an equity or derivative position. The second characteristics is that it considers correlations between different risk factors. This is essential in Portfolio Theory as it provides a risk measure that accounts for correlations among different types of assets.

A formal definition of Value at Risk is the following

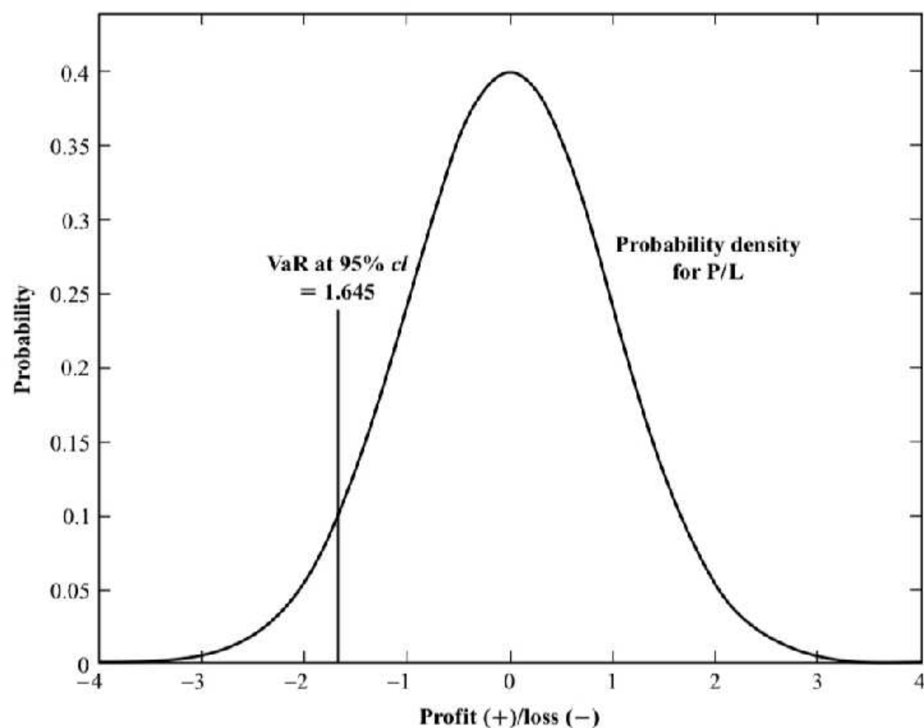
Definition 2.1. Let X be a random variable representing the returns, over a specified period of time (t), of an investment. Let α be the significance level, the α% VaR at the time horizon t is the $1 - \alpha$ quantile of the probability distribution function of X and its absolute value represents the highest possible loss at the significance level α.

$$P\{X \leq VaR_t^\alpha\} = 1 - \alpha$$

The VaR is illustrated in Figure 1, which shows a common probability density function (pdf) of profit/loss over a chosen holding period[1]. To get the VaR, we must choose a confidence level (cl). If this is 95%, then the VaR is given by the negative of the point on the x-axis that cuts off the top 95% of the observations from the bottom 5% of the tail distribution. In $(1 - \alpha)100\%$ of the cases the return is larger (better) than VaR^α. Only in $\alpha \cdot 100\%$ of the cases the return is lower (worse) than VaR^α.

[1]The figure is constructed on the assumption that P/L is normally distributed with mean 0 and standard deviation 1 over a holding period of 1 day.

Figure 1: Value at Risk



Note: Return distributions may have equal quantiles but different expected losses.

VaR has gained considerable importance to the point of becoming the main measure in risk management. Value at risk is also used by bank regulators to determine bank capital requirements against market risk. Under the 1996 Amendment to the Basel II accords, institutions approved the risk management banks' departments to have their capital requirements determined by their own VaR estimates for extreme losses coverage.

Although its conceptual semplicity and adaptability, VaR also has its drawbacks as a risk measure. VaR methodology can suffers from model risk, that is to build a model based on wrong or inappropriate assumptions on the data distribution. Moreover, the measure can discourage portfolio diversification by construction as it fails to take into account the magnitude of losses in excess of VaR.

2.2 VaR Forecasting

This thesis aims to obtain a competitive VaR estimation through the Autoregressive Denoising Model named Timegrad. This can be achieved by comparing the deep generative model to the most widely used classical alternatives such as the non-parametric method of *Historical Simula-*

tion (HS), the *Exponential weighted moving average* (EWMA) improved upon by modeling the conditional volatility with Generalized Autoregressive Conditional Heteroskedasticity (GARCH) (and its multivariate counterpart) models that will be briefly introduced below in the subsequent paragraphs.

2.2.1 Historical Simulation (HS)

Historical Simulation is a popular non-parametric method. It does not assume a particular distribution of the asset returns. Historical simulation forecasts risk by assuming that past profits and losses can be used as the distribution of profits and losses for the next period of returns. In particular, it considers the empirical distribution of the returns, over a specified time interval (from t to k) of the investment whose VaR has to be measured and computes its α quantile. The VaR "today" is computed as the α quantile of the last N returns prior to "today."

$$VaR^{\alpha}_{t,HS} = p_{1-\alpha}(r_{t-1}, r_{t-2}, ..., r_{t-k}) \tag{2.1}$$

The main advantage of this method regards its easy computation, as it can be inferred from the fact that this is a non-parametric method and that assumes that returns are $r_t \sim i.i.d.$, i.e. it considers $r_t|I_{t-1} = r_t$ (Conditional distribution of r_t given the past information is the same of the unconditional distribution of r_t) .
The main draw drawback of this method is that it poorly represents reality as financial returns are not i.i.d. (they are conditionally heteroskedastic), and as long as they are stationary, the empirical quantile will be a consistent estimator of the *unconditional* population quantile (which is constant) and not of the *conditional* population quantile (which instead varies with t). Moreover, if a constant VaR is used, the *exceptions*, i.e. the occurencies of $r_t \leq VaR$, will be concentrated in the most turbolent (riskiest) periods and will *cluster* together (as volatility clusters general clusters of exceptions). Therefore, exceptions will be autocorrelated, and thus, predictable.

2.2.2 Exponential weighted moving average (EWMA)

The exponential weighted moving average (EWMA) method assigns non-equal weights, in particular exponentially decreasing weights. The most recent returns have higher weights because they influence *today's* returns more heavily than returns further in the past. The formula for the EWMA variance over an estimation window of size W_E is

$$\widehat{\sigma}_t^2 = \frac{1}{c} \sum_{i=1}^{W_E} \lambda^{i-1} r_{t-i}^2 \tag{2.2}$$

where c is a normalizing constant:

$$c = \sum_{i=1}^{W_E} \lambda^{i-1} = \frac{1-\lambda^{W_E}}{1-\lambda} \to \frac{1}{1-\lambda} \text{ as } W_E \to \infty \tag{2.3}$$

For convenience, we assume an infinitely large estimation window to approximate the variance:

$$\widehat{\sigma}_t^2 \approx (1-\lambda)\left(r_{t-1}^2 + \sum_{i=2}^{\infty} \lambda^{i-1} r_{t-i}^2\right) = (1-\lambda) r_{t-1}^2 + \lambda \widehat{\sigma}_{t-1}^2 \tag{2.4}$$

where the decaying factor λ is usually set to 0.94 (Nieppola, 2009). We may also wish to make forecasts of future volatility. We begin by leading Equation (2.4) by one period:

$$\widehat{\sigma}_{t+1}^2 \approx (1-\lambda) r_t^2 + \lambda \widehat{\sigma}_t^2 \tag{2.5}$$

By taking expectations as of t, and noting that $E\left(r_t^2\right) = \sigma_t^2$, we get:

$$E\left(\widehat{\sigma}_{t+1}^2\right) \approx \lambda \widehat{\sigma}_t^2 + (1-\lambda)\widehat{\sigma}_t^2 = \widehat{\sigma}_t^2 \tag{2.6}$$

so the one-period ahead forecast of our volatility is approximately equal to our current volatility estimate, $\widehat{\sigma}_t^2$. It is easy to show, by similar reasoning, that our k-period ahead volatility forecast is the same:

$$E\left(\widehat{\sigma}_{t+k}^2\right) = \sigma_t^2, \quad k = 1, 2, 3, \ldots \tag{2.7}$$

The EWMA model therefore implies that our current volatility estimate also gives us our best forecast of volatility any period in the future. However, this result with the volatility forecast is not attractive as it ignores any recent movement in our returns.

2.2.3 Generalised autoregressive conditional heteroscedasticity (GARCH)

The common method used to model the volatility is the generalized autoregressive conditional heteroskedasticity, or GARCH, model. The GARCH model, which Bollerslev (1986) and Taylor (1986) proposed independently of each other, describes an approach to estimate volatility in financial markets. In this model, the conditional variance is a linear function of the q lags of the squared returns, and also p lags of the conditional variance are included:

$$\sigma_{t|t-1}^2 = \alpha_0 + \sum_{j=1}^{q} \alpha_j \epsilon_{t-j}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j|t-j-1}^2 \tag{2.8}$$

where α and β are positive parameters to be estimated. The most popular GARCH model is the GARCH(1,1), given the small number of parameters which makes its application fairly easy. When we use a GARCH process to model returns we can assume a constant mean of zero, this is a reasonable assumption given the short time periods under consideration. However, if necessary we can model the conditional mean with an Autoregressive model (AR). For the estimation of VaR we

use the conditional variance given by GARCH(1,1) model. For the underlined asset's distribution properties we use the normal distribution. For this method, Value at Risk is expressed as:

$$VaR^{\alpha}_{t,G} = \hat{\mu_t} + \hat{\sigma}_{t|t-1} N^{-1}(1-\alpha) \tag{2.9}$$

where μ is the mean stock return, σ is the standard deviation of returns, α is the selected confidence level and N^{-1} is the inverse PDF function of Gaussian distribution. $\hat{\sigma}_{t|t-1}$ is the conditional standard deviation given the information at $t-1$ and the initial value $\hat{\sigma}^2_{1|0}$ is set to the unconditional variance $\sigma^2 = \frac{\alpha_0}{1-\alpha_1-\beta_1}$.

2.2.4 Multivariate GARCH (DCC-GARCH)

Since a portfolio, a collection of financial investments, consists of multiple assets, understanding their volatility and co-volatility are of major interest to compute a more precise VaR estimation of the portfolio.

Multivariate GARCH (MGARCH) can help remedy these issues, one among the main multivariate models is the *Dynamic Conditional Correlation* (DCC) GARCH model by Engle and Shepard (Engle, R., 2002). Let define a DCC-GARCH model.

Let X_t a vector $(n \times 1)$ of stationary process, $X_t \sim DCC - GARCH$ if:

$$\begin{aligned} X_t &= \mu_t + \epsilon_t \\ \epsilon_t &= \Sigma_t^{1/2} \epsilon_t \\ \Sigma_t &= D_t R_t D_t \end{aligned} \tag{2.10}$$

with

$$D_t = diag(\Sigma_{1,t}, ..., \Sigma_{n,t})$$

The matrix Σ_t is a covariance matrix then it is easy to see that R_t is the corresponding correlation matrix and D_t contains the standard deviations of the components on the diagonal, which satisfy the univariate GARCH equations:

$$\Sigma_{i,t} = \alpha_{0i} + \sum_{q=1}^{Q_i} \alpha_{iq} \epsilon^2_{i,t-q} + \sum_{p=1}^{P_i} + \beta_{ip} \Sigma_{i,t-p} \tag{2.11}$$

where:

μ_t: Vector$(n \times 1)$ of conditional expectation of X_t at t,

ϵ_t: Vector $(n \times 1)$ conditionals errors of n actifs at t,with $E(\epsilon_t) = 0$ and $Cov(\epsilon_t) = \Sigma_t$,

Σ_t: is the matrix $(n \times n)$ of conditional variances and covariances of ϵ_t at t,

D_t: diagonal matrix $(n \times n)$ of conditional standard errors of ϵ_t at t,which is always positive,

R_t: Matrix $(n \times n)$ of conditional correlations of ϵ_t at t,

ϵ_t: Vector $(n \times 1)$ of errors *i.i.d.* with $E(\epsilon_t) = 0$ and $E(\epsilon_t \epsilon_t^{'}) = I_n$.

The conditional correlation matrix R_t can be break in:

$$R_t = diag(Q_t)^{-1/2} Q_t diag(Q_t)^{-1/2}$$

For a DCC(p,q), the proxy variable Q is in turn estimated by:

$$Q_t = [1 - \sum_{i=1}^{P} \alpha_{DCC,i} - \sum_{j=1} Q\beta_{DCC,j}]\overline{Q} + \sum_{i=1}^{P} \alpha_{DCC,i}(\epsilon_{t-i}\epsilon_{t-i}^{'}) + \sum_{j=1}^{Q} \beta_{DCC,j} Q_{t-j} \tag{2.12}$$

For a DCC(1,1)

$$Q_t = (1 - \alpha_{DCC} - \beta_{DCC})\overline{Q} + \alpha_{DCC}\epsilon_{t-1}\epsilon_{t-1}^{'} + \beta_{DCC} Q_{t-1}$$

where α_{DCC} and β_{DCC} are non negative scalars and $\overline{Q}$ is the unconditional matrix of the errors ϵ_t.
In our report we will consider the multivariate normal distribution for the residual, $\epsilon \sim N(0, I_d)$.
The DCC-GARCH model can be estimated with a two-step procedure. First we estimate a univariate GARCH model for each set of residuals. Then, the residuals, transformed from their previously estimated standard deviation, are used to estimate the parameters of conditional correlation. Similarly to the GARCH method, we can model the conditional mean with a Vector Autoregressive Moving Average (VARMA) model if necessary.
The VaR of the portfolio is then given by :

$$VaR_{t,DCC} = \hat{\mu_t} + w^{'}\Sigma_t w N^{-1}(1-\alpha) \tag{2.13}$$

if the strict white noise process is a multivariate standard normal.

2.2.5 BEKK

Another standard multivariate GARCH model is the BEKK model, named after its authors, Baba, Engle, Kraft and Kroner. This model takes the following matrix form:

$$\mathbf{\Sigma_t} = \mathbf{A}^{\mathrm{T}}\mathbf{A} + \mathbf{B}^{\mathrm{T}}\mathbf{x}_{t-1}^{\mathrm{T}}\mathbf{x}_{t-1}\mathbf{B} + \mathbf{C}^{\mathrm{T}}\mathbf{\Sigma}_{t-1}\mathbf{C} \tag{2.14}$$

where there are n different returns, Σ_t is the $n(n+1)/2$ matrix of (distinct) conditional variance and covariance terms at t, xt is the 1×n vector of returns, and A, B and C are nn matrices. This model imposes no (questionable) cross-equation restrictions, and ensures that our variance–covariance matrix will be positive definite.

However, the problem with this model is that it has a lot of parameters. For example, with only two different factors (i.e., $n = 2$), the BEKK model involves 11 different parameters, and the number of parameters rapidly rises as n gets larger. This model therefore requires far too many parameters to be used for large-dimensional problems. Of course, we can reduce the number of parameters by imposing restrictions on the parameters, but these only help us so much, and the

restrictions can create problems of their own.

2.3 Backtesting VaR

After estimating the VaR with a certain method, we need to assess the performance of VaR models. This is the goal of Var backtesting, which can be measured in different ways. As a rule of thumb, we test whether the percentage of the number of exceptions significantly differs by the quantile $1-\alpha$ and if the exceptions happen independently from one another. For these reasons we use more than one criterion to backtest the performance of VaR models, because all tests have strengths and weaknesses.

2.3.1 Binomial Test

The most straightforward test is to compare the observed number of exceptions, x, to the expected number of exceptions. From the properties of a binomial distribution, you can build a confidence interval for the expected number of exceptions. Using exact probabilities from the binomial distribution or a normal approximation, the bin function uses a normal approximation. By computing the probability of observing x exceptions, you can compute the probability of wrongly rejecting a good model when x exceptions occur. This is the p-value for the observed number of exceptions x. For a given test confidence level, a straightforward accept-or-reject result in this case is to fail the VaR model whenever x is outside the test confidence interval for the expected number of exceptions. "Outside the confidence interval" can mean too many exceptions, or too few exceptions. Too few exceptions might be a sign that the VaR model is too conservative.

The test statistic is

$$Z_{bin} = \frac{x - Np}{\sqrt{Np(1-p)}} \tag{2.15}$$

where x is the number of failures, N is the number of observations, and p = 1 − VaR level. The binomial test is approximately distributed as a standard normal distribution

2.3.2 Traffic Light Test

A variation on the binomial test proposed by the Basel Committee is the traffic light test or three zones test. For a given number of exceptions x, you can compute the probability of observing up to x exceptions. That is, any number of exceptions from 0 to x, or the cumulative probability up to x. The probability is computed using a binomial distribution. The three zones are defined as follows:

1. The "red" zone starts at the number of exceptions where this probability equals or exceeds 99.99%. It is unlikely that too many exceptions come from a correct VaR model.

2. The "yellow" zone covers the number of exceptions where the probability equals or exceeds 95% but is smaller than 99.99%. Even though there is a high number of violations, the

violation count is not exceedingly high.

3. Everything below the yellow zone is "green." If you have too few failures, they fall in the green zone. Only too many failures lead to model rejections.

2.3.3 Kupiec's Proportion of Failure (POF) Test

The unconditional coverage property formally says that the probability of realizing a loss in excess of the reported VaR, VaR_t^{α}, must be precisely $(1-\alpha)*100\%$:

$$P(I_{t+1}(\alpha)=1)=1-\alpha$$

where

$$I_{t+1}=\begin{cases} 1 \text{ if } x_{t,t+1}\leq VaR_t^{\alpha}; \\ 0 \text{ if } x_{t,t+1}>VaR_t^{\alpha}. \end{cases}$$

In case the exceptions occur more frequently than $(1-\alpha)*100\%$, our VaR measure underestimates the actual level of risk of the investment. Kupiec (1995) introduced a variation on the binomial test called the proportion of failures (POF) test. The POF test works with the binomial distribution approach. In addition, it uses a likelihood ratio to test whether the probability of exceptions is synchronized with the probability p implied by the VaR confidence level. If the data suggests that the probability of exceptions is different than p, the VaR model is rejected. The POF test statistic is

$$LR_{POF}=-2\log\left(\frac{(1-p)^{N-x}p^x}{\left(1-\frac{x}{N}\right)^{N-x}\left(\frac{x}{N}\right)^x}\right) \tag{2.16}$$

where x is the number of failures, N the number of observations and $p=1$–VaR level. This statistic is asymptotically distributed as a χ^2 variable with one degree of freedom. The VaR model fails the test if this likelihood ratio exceeds a critical value, if $LR_{POF}\geq(\chi^2)^{-1}(p)$ where p is the confidence level and $(\chi^2)^{-1}(\cdot)$ denotes the quantile function of the χ^2-distribution with one degree of freedom.

2.3.4 Christoffersen's Interval Forecast (IF) Test

Christoffersen (1998) proposed a test to measure whether the probability of observing an exception on a particular day depends on whether an exception occurred. Unlike the unconditional probability of observing an exception, Christoffersen's test measures the dependency between consecutive days only. The test statistic for independence in Christoffersen's interval forecast (IF) approach is given by

$$LR_{IF}=-2\log\left(\frac{(1-\pi)^{N_{00}+N_{10}}\pi^{N_{01}+N_{11}}}{(1-\pi_0)^{N_{00}}\pi_0^{N_{01}}(1-\pi_1)^{N_{10}}\pi_1^{N_{11}}}\right) \tag{2.17}$$

where

- N_{00} = Number of periods with no failures followed by a period with no failures.

- N_{10} = Number of periods with failures followed by a period with no failures.
- N_{01} = Number of periods with no failures followed by a period with failures.
- N_{11} = Number of periods with failures followed by a period with failures.

and

- $\pi = N_{01}/(N_{00}+N_{01})$, it is the probability of having a failure on period t, given that no failure occurred on period $t-1$.
- $\pi_1 = N_{11}/(N_{10}+N_{11})$, it is the probability of having a failure on period t, given that a failure occurred on period $t-1$.
- $\pi_1 = (N_{01}+N_{11})/(N_{00}+N_{01}+N_{10}+N_{11})$, it is the probability of having a failure on period t.

This statistic is asymptotically distributed as a (χ^2) with one degree of freedom, we reject the model if $LR_{IF} \geq (\chi^2)^{-1}(p)$.

2.3.5 Haas's Time Between Failures (TBFI) Test

Haas (2001) introduced the Time Between Failures test to incorporate the time information between all the exceptions in the sample, the number of periods between exceptions should be independent and geometrically distributed with parameter $1-\alpha$ under the null hypothesis.

$$LR_{TBFI} = -2\sum_{i=1}^{I(\alpha)} \log\left(\frac{p(1-p)^{N_i-1}}{\left(\frac{1}{N_i}\right)\left(1-\frac{1}{N_i}\right)^{N_i-1}}\right) \tag{2.18}$$

In this statistic, $p = 1-\alpha$ and N_i is the number of days between failures $i-1$ and i. This is asymptotically distributed as a χ^2 variable with $I(\alpha)$ degrees of freedom, where $I(\alpha)$ is the number of failures. We reject a model if $LR_{TBFI} \geq (\chi^2_M)^{-1}(p)$ the TBFI tests for both independence and coverage.

3 Background

In this chapter, we will discuss and introduce the required knowledge of generative deep learning and neural networks for the main topics of the thesis. Deep Feedforward Networks, Recurrent Neural Networks (RNNs) architecture and LSTM are introduced in sections 3.1, 3.2 and 3.3 respectively; what follows is an understanding of Stochastic Gradient Descent (SGD) with ADAM and Langevin dynamics for algorithmic training. Ultimately, the Denoising Diffusion Probabilistic Models and the Autoregressive Denoising Model for Multivariate Probabilistic Time series Forecasting will be discussed later in Sections 4 and 5.

3.1 Deep Feedforward Networks

Deep feedforward networks, also called feedforward neural networks, are at the heart of deep learning models. The goal of a feedforward network is to approximate some function f^*. As an example for a classifier, $y = f^*(\boldsymbol{x})$ maps an input $\boldsymbol{x}$ to a category $\boldsymbol{y}$. A feed-forward network defines a mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$ and learns the value of the parameters θ that result in the best function approximation.

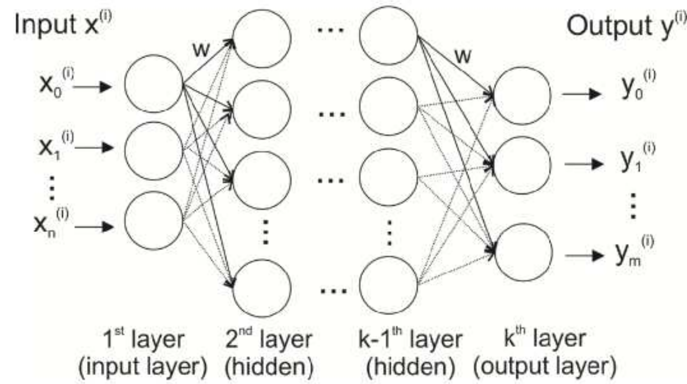
These models are feed-forward because information pass through the function being evaluated from $\boldsymbol{x}$, through the intermediate computations used to define the function f , and it ends to the output $\boldsymbol{y}$. There are no feedback connections in which outputs of the model are fed backwards. When feedforward neural networks are extended to include feedback connections, they are called **recurrent neural networks** and will be discussed in the next sub-session. The *network* is typically represented by a chain structure of functions. For example a chain of the form $f(\boldsymbol{x}) = f^{(3)}\left(f^{(2)}\left(f^{(1)}(\boldsymbol{x})\right)\right)$ is composed by three functions, where $f^{(1)}$ is the first layer, $f^{(2)}$ the second, and so on. The total number of layers is called the length of the chain and represents the depth of the model (that's the origin for deep learning terminology). The ultimate layer is called the output layer. The goal during training is to approximate f^* with the learning algorithm that optimally chooses how to adapt those **hidden** layers for the final output. Given that those layers are typically vectors, the dimensionality of them also determines the width of the model.

One way to start is to consider linear models because of their reliability in logistic and linear regressions. In spite of that, they have their limitations: whenever the input variables are nonlinear, the model cannot understand the interaction between them. To extend linear models to represent nonlinear functions of $\boldsymbol{x}$, we can apply the linear model not to $\boldsymbol{x}$ itself but to a transformed input $\phi(\boldsymbol{x})$, where ϕ is a nonlinear transformation. The strategy in deep learning training is to learn ϕ.
The model is defined by

$$y = f(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{w}) = \phi(\boldsymbol{x}; \boldsymbol{\theta})^\top \boldsymbol{w}$$

Where θ are the parameters that we use to learn ϕ from a broad class of functions, and w the parameters that map from ϕ to the desired output. This principle is what drives deep learning and it applies, with several changes, to the models described in this thesis work. Feed-forward networks have introduced the concept of a hidden layer, and this requires us to choose the activation functions used to compute the hidden layer values and ensure non-linearity. The architecture of the network, as illustrated in Figure 2, refers to the choice of the number of hidden layers, how these should be connected to each other, and how many units should be included in each layer. Learning in deep neural networks requires computing the gradients of such functions via *back-propagation*, that is the calculation of the gradient of the loss function, proceeding backwards through the feed-forward network from the last layer through to the first.

Figure 2: Feedforward Neural Network Structure



Stylized structure of a deep feedforward neural network. Each of the k layers consists of a variable number of fully connected neurons (circles). The network has as many neurons in the input layer as input variables (n), and – for classification – as many output neurons as there are classes in the data (m). A neuron is connected to all neurons in the two adjacent layers via a weighted connection (w).

3.2 Sequence Modeling: Recurrent Neural Networks

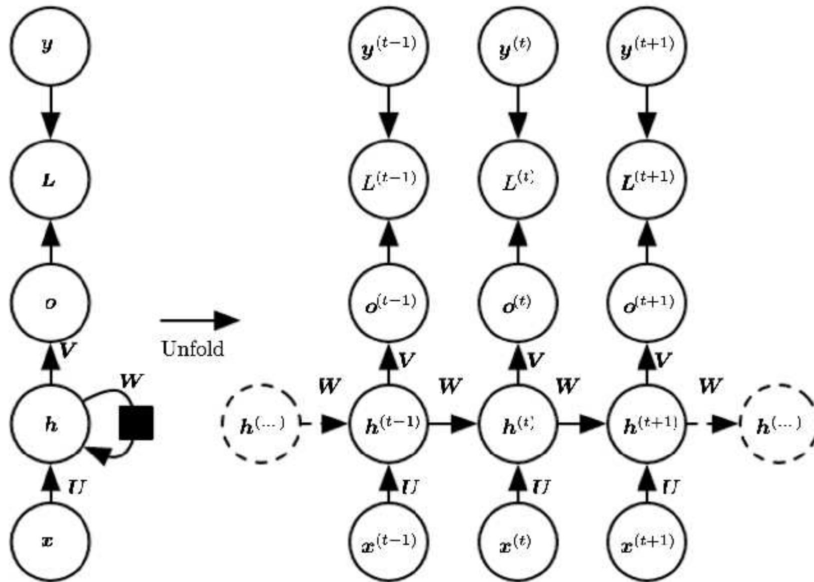
Recurrent neural networks or RNNs (Rumelhart et al., 1986a) are a rich class of neural networks for processing sequential data. A recurrent neural network is a neural network that is specialized for processing a sequence of values $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}$. RNNs can be trained for generating sequences by processing real data sequences one step at a time and predicting the next time steps. The new sequences are generated by iteratively sampling from the trained network's output distribution, then feeding in the sample as input at the next step. This distribution is conditional, since the internal state of the network, and hence its predictive distribution, depends on the previous inputs. To do this, it is necessary to introduce a state that represents the entire previous historical series. Each node of the network will have as input the value of the state in the previous time step and

the value of the input in that time step.

$$\boldsymbol{h}^{(t)} = f\left(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}\right), \tag{3.1}$$

were $\boldsymbol{h}$ is the state that indicate the hidden units of the network. When the recurrent network is trained to perform a task that requires predicting the future from the past, the network typically learns to use $h(t)$ as a kind of lossy summary of the task-relevant aspects of the past sequence of inputs up to t. After processing, it will compute the updated state value and the output value for that time step. The dynamics of the recurrent neural network can be expressed by the computational graph below in Figure 3, both in the compact and unrolled form:

Figure 3: Recurrent Neural Network Structure



The computational graph to compute the training loss of a recurrent network that maps an input sequence of $\boldsymbol{x}$ values to a corresponding sequence of output $\boldsymbol{o}$ values. A loss L measures how far each $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$. The loss L internally computes $\hat{y} = \text{softmax}(\boldsymbol{o})$ and compares this to the target $\boldsymbol{y}$. The RNN has input to hidden connections parametrized by a weight matrix $\boldsymbol{U}$, hidden-to-hidden recurrent connections parametrized by a weight matrix $\boldsymbol{W}$, and hidden-to-output connections parametrized by a weight matrix $\boldsymbol{V}$. (Left)The RNN and its loss drawn with recurrent connections. (Right)The same is seen as an time-unfolded computational graph, where each node is now associated with one particular time instance.

Forward propagation begins with a specification of the initial state $h(0)$. Then, for each time step from $t = 1$ to $t = \tau$, we apply the following update equations:

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh\left(\boldsymbol{a}^{(t)}\right) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}\left(\boldsymbol{o}^{(t)}\right)
\end{aligned}
\tag{3.2}
$$

This is an example of a recurrent network that maps an input sequence to an output sequence of the same length. Similarly to feed-forward networks, RNNs are also trained using the back-propagation algorithm. The gradient computation involves performing a forward propagation pass moving left to right using the "unrolled" version of the network, visible in the right part of the figure 3, followed by a backward propagation pass moving right to left through the graph. The algorithm, called back-propagation through time or BPTT, is very powerful but also expensive to train. Although this method has a consolidated theory behind it, in practice it is unusable as it suffers from the problem known in the literature as *vanishing / exploding gradient problem*[**?**]. The basic problem is that gradients propagated over many stages tend to either *vanish* (most of the time) or *explode* (rarely, but with much damage to the optimization). In particular, the difficulty with long-term dependencies arises from the exponentially smaller weights given to long-term interactions (involving the multiplication of many Jacobians) compared to short-term ones. It can be easily seen that the function composition employed by recurrent neural networks somewhat resembles that of matrix multiplication:

$$\boldsymbol{h}^{(t)} = \boldsymbol{W}^{\top}\boldsymbol{h}^{(t-1)} \tag{3.3}$$

where inputs $\boldsymbol{x}$ and a nonlinear activation function is omitted. Next, it can be recursively simplified to

$$\boldsymbol{h}^{(t)} = \left(\boldsymbol{W}^{t}\right)^{\top}\boldsymbol{h}^{(0)}, \tag{3.4}$$

and if we apply a spectral decomposition to the matrix we get

$$\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\top} \tag{3.5}$$

so that $h^{(t)}$ becomes

$$\boldsymbol{h}^{(t)} = \boldsymbol{Q}^{\top}\boldsymbol{\Lambda}^{t}\boldsymbol{Q}\boldsymbol{h}^{(0)}. \tag{3.6}$$

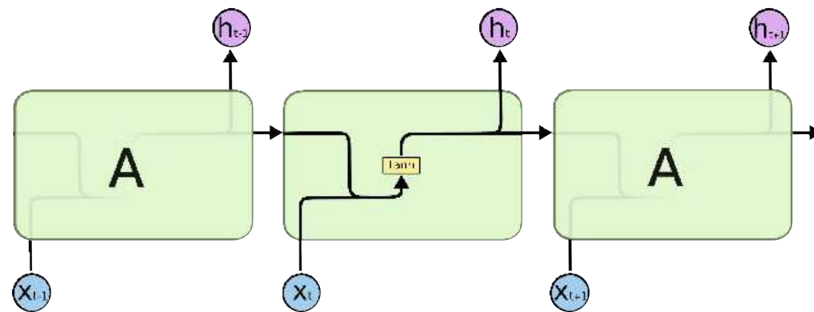
Finally, it can be seen that the eigenvalues that are raised to the power of t cause eigenvalues with magnitude less than one to decay to zero and eigenvalues with magnitude greater than one to explode.

3.3 Long Short Term Memory

As a solution to the *vanishing/exploding gradient* problem, Long Short Term Memory networks have been proposed. Usually just called "LSTMs", they are a special kind of RNN, capable of

learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work. They work extremely well on a large variety of problems and are now widely used. LSTMs are explicitly designed to avoid the long-term dependency problem therefore remembering information for long periods of time is practically their default behavior. In order to better understand the LSTM architecture it is useful to compare it with respect to the standard RNN analyzed above.

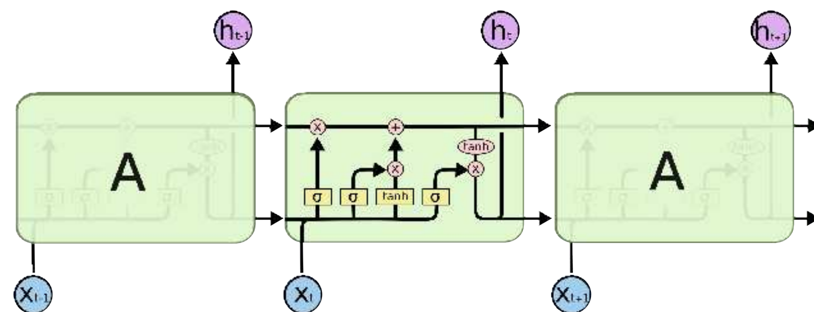
Figure 4: The Standard RNN Architecture



The repeating module in a standard RNN contains a single tanh layer.

LSTMs also have this chain-like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four interactive layers.

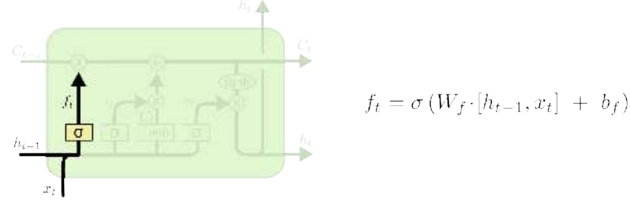
Figure 5: The LSTM Architecture.



The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The LSTM has the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. The LSTM units are conceptually similar to the nodes of the RNN as they in turn process the temporal data time step by step by saving the information about the past in a memory or state cell, but they differ in the use of an *input gate*, an *output gate* and a *forget gate*.

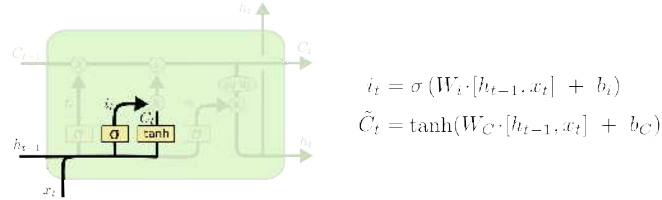
The first step is to decide what information will be preserved into the cell state or not. This decision is made by the *forget gate* through a sigmoid: it looks at h_{t1} and x_t and outputs a number between 0 (discard it completely) and 1 (keep it completely) for each number in the cell state C_{t-1}.

Figure 6: The *forget gate*.



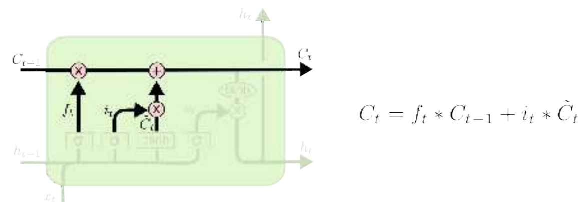
Next we decide what is the new information it is going to be stored in the cell state. First, a sigmoid, namely the *input gate* decides what values get updated. Secondly, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the cell.

Figure 7: The *input gate*.



We now update the previous cell state, C_{t-1}, into the new cell state C_t. It gets done by multiplying the previous state by f_t and then adding $i_t * \tilde{C}_t$.

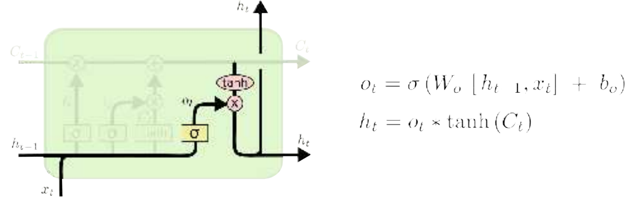
Figure 8: The updated *Cell state*.



Ultimately, we need to decide what to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state

we're going to output. Then, we put the cell state through *tanh* (to push the values to be in the range $[-1, 1]$) and multiply it by the output of the sigmoid gate, so that we only output the parts we have chosen.

Figure 9: The *output gate*.



The gated cells in the LSTM architecture increase the ability of the network to store information regarding time steps for longer periods, but it also partially solves the vanishing/exploding gradient problem(Hochreiter S., 1991). Empirically, LSTM networks and its variants have been shown to handle long-term dependencies more easily than the simple recurrent architectures and are now considered the state-of-the-art in terms of performance for long sequential series (Bengio et al., 1994; Hochreiter and Schmidhuber, 1997; Hochreiter et al., 2001; Graves et al., 2013).

3.4 Stochastic Gradient Descent

Stochastic gradient descent or SGD is probably the most relevant algorithm in nearly all of the deep learning models. It tries to solve a recurring problem in machine learning when dealing with large training sets: they become increasingly computationally expensive. We recall the typical cost function as the negative log-likelihood of the training data

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}, \mathrm{y} \sim \hat{p}_{\text{data}}} L(\boldsymbol{x}, y, \boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right) \tag{3.7}$$

where L is the loss per sample $L(\boldsymbol{x}, y, \boldsymbol{\theta}) = -\log p(y \mid \boldsymbol{x}; \boldsymbol{\theta})$. For these cost functions, gradient descent requires computing

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right) \tag{3.8}$$

The computational cost of this operation is of order $O(m)$, hence, as the data becomes increasingly large, gradient descent becomes impractical as a method.

The idea behind SGD is to consider the gradient as an expectation of a small set of samples. For each step of the algorithm a **minibatch** of samples $\mathbb{B} = \left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m')}\right\}$ is drawn uniformly from the training set. The minibatch size remains fixed as the training size m grows. The estimate of the gradient is now

$$\boldsymbol{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right) \tag{3.9}$$

The SGD algorithm follows the update rule

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{g}, \tag{3.10}$$

where ϵ is the learning rate.

3.4.1 ADAM

Researchers had soon found out that the learning rate configuration had a significant impact on the model performance and reliability, therefore a number of methods have been proposed to adapt the learning rates on model parameters.

One popular method that will be used in this work is named Adam (Kingma and Ba, 2014) which derives from adaptive moments estimation. It is a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. It computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. An exhaustive step-by-step procedure of the algorithm is reported below.

The Adam Algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}, \boldsymbol{r} = \mathbf{0}$
Initialize time step $t = 0$
while stopping criterion not met do
Sample a minibatch of m examples from the training set $\left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\right\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), \boldsymbol{y}^{(i)}\right)$
$t \leftarrow t + 1$
Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1) \boldsymbol{g}$
Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2) \boldsymbol{g} \odot \boldsymbol{g}$
Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t} \boldsymbol{r}$
Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{1-\rho_2^t}{1-}$
Compute update: $\Delta \boldsymbol{\theta} = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$ (operations applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
end while

3.4.2 Stochastic gradient Langevin dynamics

Langevin dynamics is a concept taken directly from physics, where it was developed for statistically modeling molecular systems (Welling & Tech, 2011). This method learns from large scale datasets based on iterative learning from small mini-batches. By adding the right amount of noise to a standard stochastic gradient optimization algorithm the iterates will converge to samples from the true posterior distribution as we anneal the step size.

Let θ denote a parameter vector, with $p(\theta)$ a prior distribution and $p(x|\theta)$ the probability of observation x given our model parameterized by θ. The posterior distribution of a set of N data collection $X = \{x_i\}_{i=1}^N$ is

$$p(\theta \mid X) = p(\theta) \prod_{i=1}^{N} p\left(x_i \mid \theta\right) \tag{3.11}$$

In the optimization literature the prior regularizes the parameters while the likelihood terms constitute the cost function to be optimized, and the task is to find the maximum a posteriori (MAP) parameters θ^*. In stochastic optimization methods ((Robbins Monro, 1951), at each iteration t, a subset of n data items $X_t = \{x_{t1}, \ldots, x_{tn}\}$ is given, and the parameters are updated as follows:

$$\Delta\theta_t = \frac{\epsilon_t}{2}\left(\nabla \log p\left(\theta_t\right) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p\left(x_{ti} \mid \theta_t\right)\right) \tag{3.12}$$

where ϵ_t is a sequence of step sizes. The general idea is that the gradient computed on the subset is used to approximate the true gradient over the whole dataset. Over multiple iterations the whole dataset is used and the noise in the gradient caused by using subsets rather than the whole dataset averages out. The problem regarding MAP estimation is that they do not capture parameter uncertainty and might cause overfitting of the data. Langevin dynamics (Neal, 2010) is thus introduced as a class of MC techniques that take gradient steps as well, but also injects Gaussian noise into the parameter updates in the following way:

$$\begin{aligned} \Delta\theta_t &= \frac{\epsilon}{2}\left(\nabla \log p\left(\theta_t\right) + \sum_{i=1}^{N} \nabla \log p\left(x_i \mid \theta_t\right)\right) + \eta_t \\ \eta_t &\sim N(0, \epsilon) \end{aligned} \tag{3.13}$$

As all MC methods however they require computations over the whole dataset at every iteration, which is proibitively large for massive datasets. Stochastic Gradient Langevin Dynamics combines the two ideas of stochastic potimization and Langevin dynamics which results in an efficient use of large datasets while allowing for parameter uncertainty to be captured in a Bayesian manner.

The process is straightforward: add an amount of Gaussian noise to equation (3.12), balanced with the step size in use and allow step sizes to approach zero:

$$\begin{aligned} \Delta\theta_t &= \frac{\epsilon_t}{2}\left(\nabla \log p\left(\theta_t\right) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p\left(x_{ti} \mid \theta_t\right)\right) + \eta_t \\ \eta_t &\sim N\left(0, \epsilon_t\right) \end{aligned} \tag{3.14}$$

Letting the step sizes decrease towards zero[2] allows to average out the stochasticity in the gradients and, as $t \to \infty$, θ_t will approach samples from the posterior distribution $p(\theta|X)$.

The introduced method has since been applied to model also deep learning algorithms, such as the denoising diffusion model we are going to use in this work, because for early iterations of the algorithm, each parameter update mimics Stochastic Gradient Descent; however, as the algorithm approaches a local minimum or maximum, the gradient shrinks to zero and the chain produces samples surrounding the maximum a posteriori mode allowing for posterior inference.

[2]To ensure convergence to a local minimum, in addition to other technical assumptions, a major requirement is for the step sizes to satisfy the property

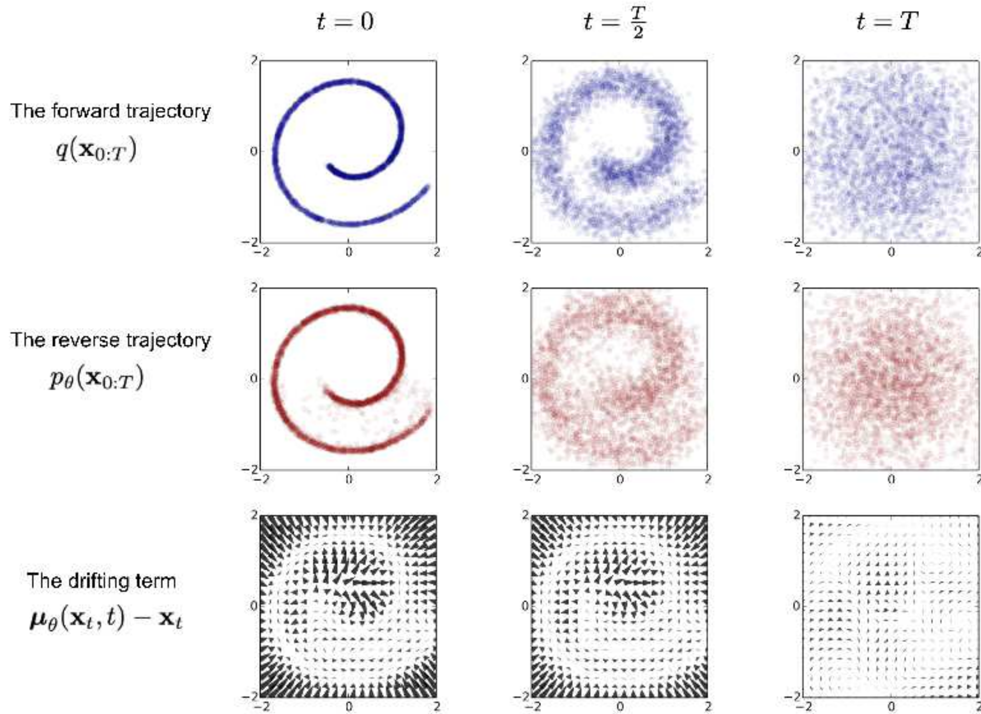
$$\sum_{t=1}^{\infty} \epsilon_t = \infty \quad \sum_{t=1}^{\infty} \epsilon_t^2 < \infty$$

4 Denoising Diffusion Probabilistic Models

4.1 Introduction

The first setup for our empirical analysis is the introduction of a certain type of *Energy-based model* (EBM) of (Ho et al., 2020) using diffusion probabilistic models (Sohl-Dickstein 2015). A diffusion probabilistic model is a parameterized Markov chain trained by using variational inference in order to produce samples matching the data after an arbitrary period of time. Steps of this chain are learned by reversing a diffusion process, which is basically a Markov chain that adds noise to the data at each step forward until the original sampling distribution is destroyed. When the added noise consists of Gaussian noise, the reverse process is simplified as the sampling consists of conditional Gaussians too thus facilitating the network parameterization. A first glimpse of what it means training a diffusion model is given below at Figure 10, in this case for modeling a 2D swiss roll data (Image source: Sohl-Dickstein et al., 2015).

Figure 10: Diffusion Model Example



4.2 Architecture

Diffusion models (Sohl-Dickstein, 2015) are latent variable models of the form $p_\theta\left(\mathbf{x}^0\right) := \int p_\theta\left(\mathbf{x}^{0:N}\right) \mathrm{d}\mathbf{x}^{1:N}$, where $\mathbf{x}^1, \ldots, \mathbf{x}^N$ are latents of dimension $\mathbb{R}^D$, the same dimensionality of the data $\mathbf{x}_0 \sim q\left(\mathbf{x}_0\right)$. The joint distribution $p_\theta\left(\mathbf{x}^{0:N}\right)$ is called the *reverse process* and is defined as a Markov chain with learned Gaussian transitions starting at $p\left(\mathbf{x}^N\right) = \mathcal{N}\left(\mathbf{x}^N; \mathbf{0}, \mathbf{I}\right)$:

$$p_\theta\left(\mathbf{x}^{0:N}\right) := p\left(\mathbf{x}^N\right) \Pi_{n=N}^{1} p_\theta\left(\mathbf{x}^{n-1} \mid \mathbf{x}^n\right), \quad p_\theta\left(\mathbf{x}^{n-1} \mid \mathbf{x}^n\right) := \mathcal{N}\left(\mathbf{x}^{n-1}; \mu_\theta\left(\mathbf{x}^n, n\right), \Sigma_\theta\left(\mathbf{x}^n, n\right) \mathbf{I}\right) \tag{4.1}$$

where θ are the shared parameters; $\mu_\theta : \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}^D$ and $\Sigma_\theta : \mathbb{R}^D \times \mathbb{N} \rightarrow \mathbb{R}^{+}$take two inputs, namely the variable $\mathbf{x}^n \in \mathbb{R}^D$ as well as the noise index $n \in \mathbb{N}$.

The *forward* or *diffusion process* as mentioned above, is a Markov chain that gradually adds Gaussian noise to the data following a given increasing variance schedule $\beta_1, \ldots, \beta_n$ with $\beta_n \in (0,1)$:

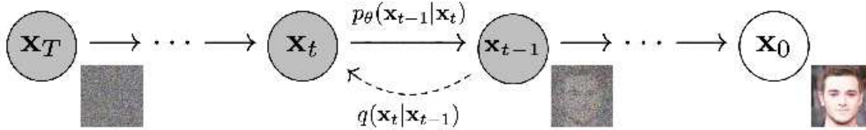
$$q\left(\mathbf{x}^{1:N} \mid \mathbf{x}^0\right) = \Pi_{n=1}^{N} q\left(\mathbf{x}^n \mid \mathbf{x}^{n-1}\right), \quad q\left(\mathbf{x}^n \mid \mathbf{x}^{n-1}\right) := \mathcal{N}\left(\mathbf{x}^n; \sqrt{1-\beta_n} \mathbf{x}^{n-1}, \beta_n \mathbf{I}\right) \tag{4.2}$$

As shown by (Ho et al., 2020) the diffusion (*forward*) process has the property that it can be rewritten in closed form at any arbitrary noise level n:

$$q\left(\mathbf{x}^n \mid \mathbf{x}^0\right) = \mathcal{N}\left(\mathbf{x}^n; \sqrt{\bar{\alpha}_n} \mathbf{x}^0, \left(1-\bar{\alpha}_n\right) \mathbf{I}\right) \tag{4.3}$$

where $\alpha_n := 1 - \beta_n$ and $\bar{\alpha}_n := \Pi_{i=1}^{n} \alpha_i$, its cumulative product.

Figure 11: Diffusion Model Representation



The model developed in the work of Sohl-Dickstein

When it comes to training the model, the parameters θ are learned by minimizing the negative log-likelihood via a variational bound using Jensen's inequality:

$$\min_\theta \mathbb{E}_{q(\mathbf{x}^0)}\left[-\log p_\theta\left(\mathbf{x}^0\right)\right] \leq \min_\theta \mathbb{E}_{q(\mathbf{x}^{0:N})}\left[-\log p_\theta\left(\mathbf{x}^{0:N}\right) + \log q\left(\mathbf{x}^{1:N} \mid \mathbf{x}^0\right)\right], \tag{4.4}$$

where the upper bound on the RHS can be rewritten as

$$L := \min_\theta \mathbb{E}_{q(\mathbf{x}^{0:N})}\left[-\log p\left(\mathbf{x}^N\right) - \sum_{n=1}^{N} \log \frac{p_\theta\left(\mathbf{x}^{n-1} \mid \mathbf{x}^n\right)}{q\left(\mathbf{x}^n \mid \mathbf{x}^{n-1}\right)}\right] \tag{4.5}$$

The *Loss* function is trained by SGD. In order to have a tractable objective function however, we should first rewrite it in terms of distances between Gaussian distributions using the Kullback-Leibler (KL)-divergence[3]:

$$\begin{aligned}
L &= \mathbb{E}_q\left[-\log p\left(\mathbf{x}^N\right)-\sum_{n=1}^{N}\log\frac{p_\theta\left(\mathbf{x}^{n-1}\mid\mathbf{x}^n\right)}{q\left(\mathbf{x}^n\mid\mathbf{x}^{n-1}\right)}\right] \\
&= \mathbb{E}_q\left[-\log p\left(\mathbf{x}^N\right)-\sum_{n>1}\log\frac{p_\theta\left(\mathbf{x}^{n-1}\mid\mathbf{x}^n\right)}{q\left(\mathbf{x}^n\mid\mathbf{x}^{n-1}\right)}-\log\frac{p_\theta\left(\mathbf{x}^0\mid\mathbf{x}^1\right)}{q\left(\mathbf{x}^1\mid\mathbf{x}^0\right)}\right] \\
&= \mathbb{E}_q\left[-\log p\left(\mathbf{x}^N\right)-\sum_{n>1}\log\frac{p_\theta\left(\mathbf{x}^{n-1}\mid\mathbf{x}^n\right)}{q\left(\mathbf{x}^{n-1}\mid\mathbf{x}^n,\mathbf{x}^0\right)}\cdot\frac{q\left(\mathbf{x}^{n-1}\mid\mathbf{x}^0\right)}{q\left(\mathbf{x}^n\mid\mathbf{x}^0\right)}-\log\frac{p_\theta\left(\mathbf{x}^0\mid\mathbf{x}^1\right)}{q\left(\mathbf{x}^1\mid\mathbf{x}^0\right)}\right] \\
&= \mathbb{E}_q\left[-\log\frac{p\left(\mathbf{x}^N\right)}{q\left(\mathbf{x}^N\mid\mathbf{x}^0\right)}-\sum_{n>1}\log\frac{p_\theta\left(\mathbf{x}^{n-1}\mid\mathbf{x}^n\right)}{q\left(\mathbf{x}^{n-1}\mid\mathbf{x}^n,\mathbf{x}^0\right)}-\log p_\theta\left(\mathbf{x}^0\mid\mathbf{x}^1\right)\right] \\
&= \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}^N\mid\mathbf{x}^0\right)\|p\left(\mathbf{x}^N\right)\right)}_{L^N}+\sum_{n>1}\underbrace{D_{\mathrm{KL}}\left(q\left(\mathbf{x}^{n-1}\mid\mathbf{x}^n,\mathbf{x}^0\right)\|p_\theta\left(\mathbf{x}^{n-1}\mid\mathbf{x}^n\right)\right)}_{L^{n-1}}\underbrace{-\log p_\theta\left(\mathbf{x}^0\mid\mathbf{x}^1\right)}_{L^0}]
\end{aligned} \tag{4.6}$$

In (Ho et al.,2020) it is shown how this KL-divergence formulation makes it possible to directly compare the two processes, as they become tractable when conditioned on x^0:

$$q\left(\mathbf{x}^{n-1}\mid\mathbf{x}^n,\mathbf{x}^0\right)=\mathcal{N}\left(\mathbf{x}^{n-1};\tilde{\mu}_n\left(\mathbf{x}^n,\mathbf{x}^0\right),\tilde{\beta}_n\mathbf{I}\right) \tag{4.7}$$

where

$$\tilde{\mu}_n\left(\mathbf{x}^n,\mathbf{x}^0\right):=\frac{\sqrt{\bar{\alpha}_{n-1}}\beta_n}{1-\bar{\alpha}_n}\mathbf{x}^0+\frac{\sqrt{\alpha_n}\left(1-\bar{\alpha}_{n-1}\right)}{1-\bar{\alpha}_n}\mathbf{x}^n \quad \text{and} \quad \tilde{\beta}_n:=\frac{1-\bar{\alpha}_{n-1}}{1-\bar{\alpha}_n}\beta_n \tag{4.8}$$

We now take a look at each component of the objective function in order to have an algorithm both for training and sampling the generative process. First of all, by keeping the forward process variances β_n constants, the posterior q has no learnable parameters and we can ignore L^N during training since it remains constant as well. For our work we are interested in the middle term, which can now be rewritten as

$$L^{n-1}:=D_{\mathrm{KL}}\left(q\left(\mathbf{x}^{n-1}\mid\mathbf{x}^n,\mathbf{x}^0\right)\|p_\theta\left(\mathbf{x}^{n-1}\mid\mathbf{x}^n\right)\right)=\mathbb{E}_q\left[\frac{1}{2\sigma_\theta}\left\|\tilde{\mu}_n\left(\mathbf{x}^n,\mathbf{x}^0\right)-\mu_\theta\left(\mathbf{x}^n,n\right)\right\|^2\right]+C \tag{4.9}$$

where C is a constant which does not depend on θ. We have also set $\mathbf{\Sigma}_\theta\left(\mathbf{x}_t,t\right)=\sigma_t^2\mathbf{I}$ to untrained time dependent constants. We can take advantage of the closed form property of the forward process in equation (4.3) by reparameterizing it as $\mathbf{x}^n\left(\mathbf{x}^0,\epsilon\right)=\sqrt{\bar{\alpha}_n}\mathbf{x}^0+\sqrt{1-\bar{\alpha}_n}\epsilon$ for $\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})$ and the formula in (4.8) to obtain:

[3] Extended derivation from Sohl-Dickstein.

$$L_{n-1} - C = \mathbb{E}_{\mathbf{x}_0,\epsilon}\left[\frac{1}{2\sigma_n^2}\left\|\tilde{\boldsymbol{\mu}}_n\left(\mathbf{x}^n\left(\mathbf{x}^0,\boldsymbol{\epsilon}\right), \frac{1}{\sqrt{\bar{\alpha}_n}}\left(\mathbf{x}^n\left(\mathbf{x}^0,\boldsymbol{\epsilon}\right) - \sqrt{1-\bar{\alpha}^n}\boldsymbol{\epsilon}\right)\right) - \boldsymbol{\mu}_\theta\left(\mathbf{x}^n\left(\mathbf{x}^0,\boldsymbol{\epsilon}\right), n\right)\right\|^2\right]$$
$$= \mathbb{E}_{\mathbf{x}^0,\epsilon}\left[\frac{1}{2\sigma_n^2}\left\|\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}^n\left(\mathbf{x}^0,\boldsymbol{\epsilon}\right) - \frac{\beta^n}{\sqrt{1-\bar{\alpha}_n}}\boldsymbol{\epsilon}\right) - \boldsymbol{\mu}_\theta\left(\mathbf{x}^n\left(\mathbf{x}^0,\boldsymbol{\epsilon}\right), n\right)\right\|^2\right] \tag{4.10}$$

Equation (4.10) shows that μ_θ must predict $\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}^n - \frac{\beta^n}{\sqrt{1-\bar{\alpha}_n}}\boldsymbol{\epsilon}\right)$ given x^n. Since x^n is an input to the network, we can choose:

$$\mu_\theta\left(\mathbf{x}^n, n\right) := \frac{1}{\sqrt{\alpha_n}}\left(\mathbf{x}^n - \frac{\beta_n}{\sqrt{1-\bar{\alpha}_n}}\epsilon_\theta\left(\mathbf{x}^n, n\right)\right) \tag{4.11}$$

so that the objective function in equation (4.19) simplifies to

$$\mathbb{E}_{\mathbf{x}^0,\epsilon}\left[\frac{\beta_n^2}{2\sigma_\theta\alpha_n\left(1-\bar{\alpha}_n\right)}\left\|\epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_n}\mathbf{x}^0 + \sqrt{1-\bar{\alpha}_n}\epsilon, n\right)\right\|^2\right] \tag{4.12}$$

The applied reparameterization shows in **Algorithm 1** how training is achieved via the MSE loss between the predicted Gaussian noise ϵ_θ and the true noise $\epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I})$. Ultimately to sample the reverse process $\mathbf{x}^{n-1} \sim p_\theta\left(\mathbf{x}^{n-1} \mid \mathbf{x}^n\right)$ it is sufficient to compute

$$\mathbf{x}^{n-1} = \frac{1}{\sqrt{\alpha_n}}\left(\mathbf{x}^n - \frac{\beta_n}{\sqrt{1-\bar{\alpha}_n}}\epsilon_\theta\left(\mathbf{x}^n, n\right)\right) + \sqrt{\sigma_\theta}\mathbf{z} \tag{4.13}$$

where $\mathbf{z} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ for $n = N,\ldots,2$ and $\mathbf{z} = \mathbf{0}$ when $n = 1$. The complete sampling procedure, starting from x^N until x^0 as reported in **Algorithm 2**, resembles Langevin dynamics where we sample from the most perturbed sample and progressively reduce the magnitude of the noise until we reach the smallest one at x^0.

Algorithm 1 Training

1: **repeat**
2 : $\mathbf{x}^0 \sim q\left(\mathbf{x}^0\right)$
3 : $n \sim \text{Uniform}\left(\{1,\ldots,N\}\right)$
4 : $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$
5 : Take gradient descent step on
$\nabla_\theta\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\left(\sqrt{\bar{\alpha}_n}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_n}\boldsymbol{\epsilon}, n\right)\right\|^2$
6 : **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}^N \sim \mathcal{N}(\mathbf{0},\mathbf{I})$
2: for $n = N,\ldots,1$ do
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ if $n > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}^{n-1} = \frac{1}{\sqrt{\alpha_n}}\left(\mathbf{x}^n - \frac{1-\alpha_n}{\sqrt{1-\bar{\alpha}_n}}\boldsymbol{\epsilon}_\theta\left(\mathbf{x}^n, n\right)\right) + \sigma_n\mathbf{z}$
5: end for
6: return $\mathbf{x}_0$

5 Timegrad

5.1 Introduction

We are now ready to introduce *Timegrad* method to generate new examples of sampled data from the distribution of the original dataset. In deep learning a straightforward time series model for multivariate real-valued data could take the full multivariate vector x_t and covariates as inputs to the RNN and use a factorizing output distribution. For this purpose, a full joint distribution at each time step has to be modeled, which happens to be a multivariate Gaussian most of the times for practical purposes. However, this has not proved to be a viable solution as the full covariance matrix of the multivariate Gaussian distribution exponentially increases both the number of parameters and the computational cost in training the loss function. Other works such that referred as Vec-LSTM propose to approximate the Gaussian distribution with with diagonal or low-rank covariance matrices and has proven to be effective in popular datasets (Salinas et al., 2019a).

The method proposed i *Timegrad*(Rasul et al., 2021), learns the gradient of the data distribution by optimising the variational lower bound and then generates samples through the denoising diffusion model introduced in Section 4. Using a model architecture defined by LSTM, residual, and dilated convolutional layers, the authors achieved SotA results across six popular time series datasets and that motivated our interest in applying this method to financial time series for VaR Estimation and backtesting.

5.2 The Model

We start by defining multivariate time series as $x_{i,t}^0 \in \mathbb{R}$ for $i \in \{1, \ldots, D\}$ where t is the time index. We want to predict the multivariate distribution for some chosen prediction time steps ahead so we will consider contiguous sequences sampled from the whole time series training set which will be splitted into a context window sized interval $[1, t_0)$ and our chosen prediction length sized interval $[t_0, T]$.

Time grad models the conditional distribution of the future time steps of a multivariate time series given its past and covariates as:

$$q_{\mathcal{X}}\left(\mathbf{x}_{t_0:T}^0 \mid \mathbf{x}_{1:t_0-1}^0, \mathbf{c}_{1:T}\right) = \Pi_{t=t_0}^{T} q_{\mathcal{X}}\left(\mathbf{x}_t^0 \mid \mathbf{x}_{1:t-1}^0, \mathbf{c}_{1:T}\right) \tag{5.1}$$

where $\mathcal{X} = \mathbb{R}^D$ is the input space and $\mathbf{c}_t$ the covariates that are known for all the time points and each factor resembles those learned by the conditional denoising diffusion model introduced above. The method employ the RNN architecture (Graves, 2013) with the LSTM algorithm to encode the time series sequences up until $t-1$, given the covariates of the next time step $\mathbf{c}_t$, by the updated hidden state $\mathbf{h}_{t-1}$

$$\mathbf{h}_{t-1} = \mathrm{RNN}_\theta\left(\text{ concat }\left(\mathbf{x}_{t-1}^0, \mathbf{c}_t\right), \mathbf{h}_{t-2}\right) \tag{5.2}$$

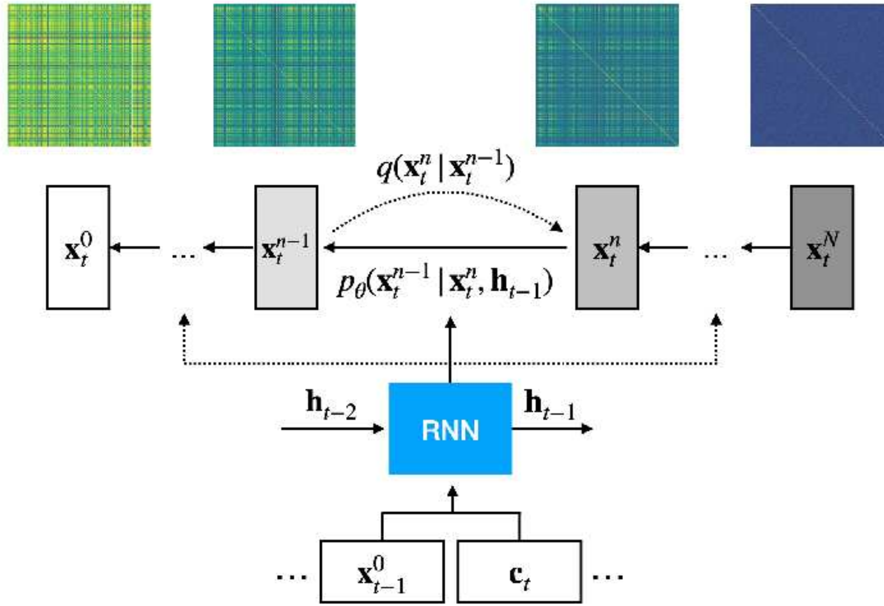
where RNN_θ is a multi-layer LSTM parameterized by shared weights θ and $\mathbf{h}_0 = \mathbf{0}$.

The conditional distribution in equation (5.1) is approximated by the *TimeGrad* model as

$$\Pi_{t=t_0}^{T} p_\theta \left(\mathbf{x}_t^0 \mid \mathbf{h}_{t-1}\right) \tag{5.3}$$

where θ includes the weights of both the RNN and the denoising diffusion model. The model is autoregressive as it takes the observation of the previous time step to learn or to sample the distribution of the next period as shown in Figure 12.

Figure 12: Timegrad Schematic



An RNN conditioned diffusion probabilistic model at some time $t-1$ showing the fixed forward process that adds Gaussian noise to the signal and the learned reverse processes. The cross-correlation of the time series is also depicted during the stages of the process, from *pseudo*-white noise (right) to the original distribution (left).

5.3 Training

Training algorithm is performed by random sampling the context and adjoining prediction length sized windows from the training data and learning the parameters θ that minimize the loss function of *Timegrad* in (5.3):

$$\sum_{t=t_0}^{T} -\log p_\theta \left(\mathbf{x}_t^0 \mid \mathbf{h}_{t-1}\right) \tag{5.4}$$

We can now recall the objective function (4.12) derived in Section 4 to express the objective for training the *Timegrad* model in a similar way except that now the network is also conditioned on the hidden state, that is

$$\mathbb{E}_{\mathbf{x}_t^0,\epsilon,n}\left[\left\|\epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_n}\mathbf{x}_t^0 + \sqrt{1-\bar{\alpha}_n}\epsilon, \mathbf{h}_{t-1}, n\right)\right\|^2\right] \tag{5.5}$$

which discards the weighting for simplicity. In a similar way from Section 4 Algorithm 1 is the training procedure for each step in the chosen prediction window using this objective.

Algorithm 1 Training for each time series step $t \in [t_0, T]$

Input: data $\mathbf{x}_t^0 \sim q_{\mathcal{X}}\left(\mathbf{x}_t^0\right)$ and state $\mathbf{h}_{t-1}$
repeat
Initialize $n \sim$ Uniform $(1, \ldots, N)$ and $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
Take gradient step on
$\nabla_\theta \left\|\epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_n}\mathbf{x}_t^0 + \sqrt{1-\bar{\alpha}_n}\epsilon, \mathbf{h}_{t-1}, n\right)\right\|^2$
until converged

5.4 Inference

The next step is to make inference with our trained model. As usual for Deep Learning models, we want to compare the model performance with the related time series test set. We run the RNN in (5.2) over the training set until the hidden state $\mathbf{h}_T$ is obtained. Recalling the sampling procedure in Section 4 we obtain a sample $\mathbf{x}_{T+1}^0$ of the next period, which can subsequently be inserted autoregressively into the RNN (and the covariates $\mathbf{c}_{T+2}$) to obtain the next hidden state $\mathbf{h}_{T+1}$ and repeat until the forecast on our prediction length is completed. In order to get the compulsory quantiles for our VaR Estimation and Backtesting it is necessary to repeat this process many times ($\simeq 100$).

Algorithm 2 Sampling $\mathbf{x}_t^0$ via annealed Langevin dynamics

Input: noise $\mathbf{x}_t^N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and state $\mathbf{h}_{t-1}$
for $n = N$ to 1 do
if $n > 1$ then
$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
else
$\mathbf{z} = \mathbf{0}$
end if
$\mathbf{x}_t^{n-1} = \frac{1}{\sqrt{\alpha_n}}\left(\mathbf{x}_t^n - \frac{\beta_n}{\sqrt{1-\bar{\alpha}_n}}\epsilon_\theta\left(\mathbf{x}_t^n, \mathbf{h}_{t-1}, n\right)\right) + \sqrt{\sigma_\theta}\mathbf{z}$
end for
Return: $\mathbf{x}_t^0$

6 Timegrad VaR

The performance of VaR Estimation of the classical parametric and non-parametric methods reviewed in section 2.2 depends on how much those assumptions fit the real data. We have already mentioned the flaws of non-parametric methods such as *HS* in providing a good representation of the exceptions of returns in 2.2.1. On the other side time-series econometrics models' performance strictly rely on the model construction as well as their assumptions. The deep learning approach instead generates new samples from the data in order to learn the parameters of the underlying conditional distribution. Our attempt is to adapt the Timegrad Model to Value-at-Risk Estimation with a chosen portfolio of assets. We will first adapt our data to the train the model and then we will proceed to VaR Estimation with it.

6.1 Architecture

We train *Timegrad* with SGD using Adam algorithm (Section 3.4.1) with a learning rate of 1×10^{-3} on the training split of the dataset with $N = 100$ diffusion steps using a linear variance schedule starting from $\beta_1 = 1 \times 10^{-4}$ to β_N. We adopt for batches the same size of the original paper (Rasul et al., 2021) by taking random windows (with possible overlapping), with the context size set to the number of prediction steps, from the total time steps of our dataset. As a preliminary evaluation metric for the model we perform $CRPS_{SUM}$ using a rolling windows prediction starting from the last context window history before the start of the prediction and compare it to the test set by sampling S = 100 generated trajectories.

The RNN consists of 2 layers of an LSTM or GRU (Chung et al., 2014) with the hidden state $\mathbf{h}_t \in \mathbb{R}^{40}$ and we encode the noise index $n \in 1, ..., N$ using the Transformer's (Vaswani et al., 2017) Fourier positional embeddings, with $N_max = 500$, into $\mathbf{h}_t \in \mathbb{R}^{32}$ vectors.
We heavily rely on *Timegrad* original paper in order to construct the network architecture of ϵ_θ, which we recall is found in (5.5) to predict the noisy signal, consisting of conditional 1-dim dilated *ConvNets* with residual connections adapted from the *WaveNet* (van den Oord et al., 2016a) and *DiffWave* (Kong et al., 2020) models. Figure 13 shows the schematics of a single residual block $b = 0, ..., 7$ together with the final output from the sum of all the 8 skip-connections. All, but the last, convolutional network layers have an output channel size of 8 and we use a bidirectional dilated convolution in each block b by setting its dilation to $2^{b\%2}$.

We use a validation set from the training data of the same size as the test set to tune the number of epochs for early stopping. The source code of the model has been developed with PyTorchTS[4] (Rasul, 2021), a PyTorch Probabilistic Time Series forecasting framework which provides state of

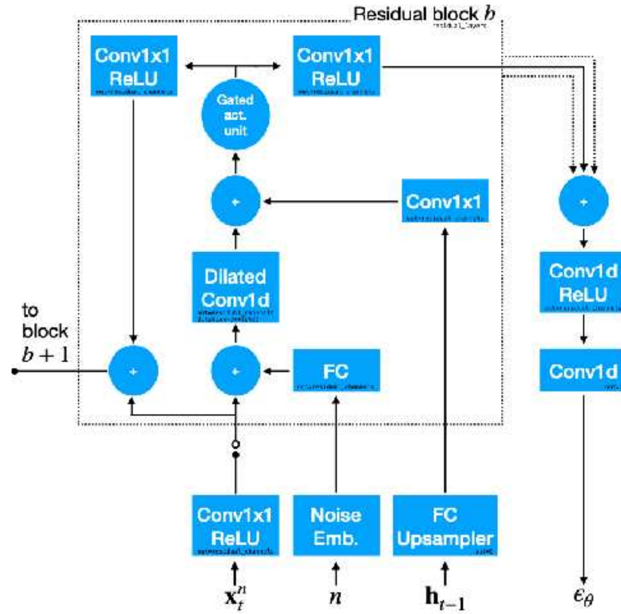
[4]https://github.com/zalandoresearch/pytorch-ts

the art PyTorch time series models by utilizing GluonTS[5] as its back-end API and for loading, transforming and back-testing time series data sets. The package is utilized in order to perform probabilistic forecasts, while the VaR is calculated leveraging Timegrad's output.

Axyon AI has access to CINECA HPC environment and all experiments have been performed on MARCONI100[6], an accelerated cluster based on IBM Power9 processors and NVIDIA Volta GPUs (980 nodes) with

1. 2x16 cores IBM POWER9 AC922 at 3.1 GHz
2. 4 x NVIDIA Volta V100 GPUs, Nvlink 2.0, 16GB
3. 256 GB RAM

Figure 13: Network Architecture



The network architecture of ϵ_θ consisting of residual-layers = 8 conditional residual blocks with the Gated Activation Unit $\sigma(\cdot) \odot \tanh(\cdot)$ from (van den Oord et al., 2016b); whose skip-connection outputs are summed up to compute the final output. Conv1x1 and Conv1d are 1D convolutional layers with filter size of 1 and 3, respectively, circular padding so that the spatial size remains D, and all but the last convolutional layer has output channels residual channels = 8. FC are linear layers used to up/down-sample the input to the appropriate size for broadcasting.

[5] https://github.com/awslabs/gluon-ts

[6] At the time of writing MARCONI100 is the 18th largest cluster globally. (Source:https://www.top500.org/system/179845/)

6.2 Dataset

The dataset we used for our experiments consists of daily close prices, ranging from 2000-01-01 to 2019-07-29[7], with the data being obtained from the Python package *yfinance*. In order to take advantage of the multivariate setting we selected a majority of highly correlated indices. We computed the cumulative returns of each stock indices by dividing the current closing price value of the stock with the initial closing price value of the stock. We then plotted the returns with the help of the Matplotlib package on Python and we got the chart below as the result. The returns can also be calculated in a daily timeframe but the reason for choosing cumulative returns is that it will be easier to notice correlations between stocks when plotted in a graph. For example, from the chart, we could notice that a strong correlation exists between all eight[8] of the stocks since it all shows similar fluctuations or movements in its price. On contrary, it is impossible to observe such movements in a daily returns plot since the lines will be overlapping one and another. The equally weighted portfolio consists of the US SP500 Index (SP500), The Dow Jones Industrial Average (DJIA), the NASDAQ Composite, the UK Financial Times Stock Exchange 100 Index (FTSE100), the French CAC40, the italian FTSE MIB, the japanese NIKKEI 225 and the brazilian Bovespa.

Figure 14: Index Cumulative Returns

We are also representing a 'heatmap' function in Figure 15 provided by the Seaborn package to make a heatmap plot out of the correlation matrix of the portfolio. It is easy to see that values inside the plot are nothing but the correlation scores.

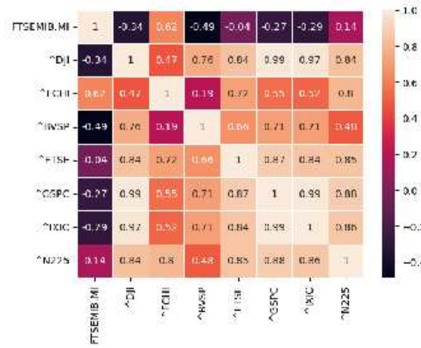
6.3 Preprocessing

Before feeding the model with our data, we need a preprocessing phase to transform our time series into a manageable dataset for VaR Backtesting. All of the transformations are invertible allowing a post-process phase where the results will be compared to their original time series data.

[7]We do not include the data of the on-going COVID-19 pandemic crisis in order to avoid high volatily determined by an exogenous shock in the market.

[8]Bovespa Index is not included since its scale was significantly different from the others but it shows a very similar path anyway.

Figure 15: Heatmap of the correlation matrix



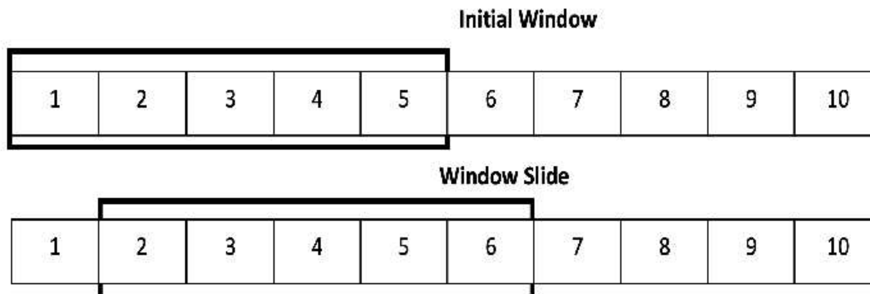
1. **Log return r_t**

$$r_t = \log\left(\frac{s_t}{s_{t-1}}\right) \forall t \in 1, \dots, T \tag{6.1}$$

2. **Scaling**
   In order to simplify the problem for the model we normalize scales of the training set. We divide each time series entity by their context window mean (or 1 if it's zero) before feeding it into the model. During inference, the samples are then multiplied by the same mean values to match the original scale. This results in significantly improved empirical performance of the model as seen in (Salinas et. al, 2019b).

3. **Rolling windows**
   We have also partioned the data with a rolling windows evaluation scenario. A sliding window is placed on the preprocessed time series and all points within the window form a sub-sequence. The sliding window is then moved on and on until the end of the dataset. Such method,illustrated in Figure 16, is necessary for VaR Backtesting and it is referred as Window Slicing (Le Guennec et. al., 2016).

Figure 16: Windows Slicing Process



6.4 Hyperparameter Tuning

In the context of neural networks, deep learning algorithms come with many hyperparameters that control many aspects of the algorithm's behavior. The performance of those algorithms strongly relies on well those parameters to fit the data. Selecting hyperparameters, namely *hyperparameter tuning* can be done with two different appproaches: choosing them manually or automatically. We will stick with the first option as Automatic Hyperparameter Optimization Algorithms require their own hyperparameters, such as the range of values that should be explored for each of the learning algorithm's hyperparameters. Moreover, given that we want to follow the original architecture of *Timegrad*, we have chosen the default values as a starting point for our setting. The relevant parameter that has to be changed from the original work was that of the context window length. We tried three different window lengths: 5, 10 and 30 periods. The performance in the backtest showed that with a too short context window length (5) the model is too sensitive to small changes and with a longer window (30) it does not react fast enough. With 10 periods we have had the best results.

Model hyperparameters:

- *Batch size* = 64. Number of samples present in a single training step.
- *Epochs*= 20.
- *Number of batches per epoch*= 100.
- *Optimizer learning rate*= 10^{-4}.
- *Context window length* = 5. Number of days given as input to LSTM encoding block.
- *Frequency*= $1B$. *Pandas* format to indicate the time series to consist of business days only (Monday to Friday).
- *Cell type*= $LSTM$. The algorithm to encode the time series sequence.
- *Diffusion steps*= 100
- *Dataset Split* = $[0.7, 0.3]$. Proportions into which the dataset is divided. In our case it implies that 70% of the data is reserved for the training set, 30% for the test set.

6.5 Results

We will now show the results running the code and training the model for our equally weighted portfolio of eight indices. First of all we will start with the predictions and predictions intervals as reported in the original paper of (Rasul et al., 2021). We will then proceed to the core of this thesis work: computing the predicted VaR for our rolling windows scenario and compare it with our benchmark baseline models.

6.5.1 Timegrad Predictions

To highlight the predictions of *Timegrad* we show in Figure 18 the predicted median, 95% and 99% distribution intervals of all the 8 dimensions of the Portfolio's Indexes returns. We set the forecasting horizon to 30 periods to replicate the original paper prediction length of *Exchange* dataset, which has a similar data structure. For evaluation, GluonTS provides a *Multivariate Evaluator* object that iterates over true targets and forecasts in a streaming fashion, and calculates metrics, such as normalized root-mean-squared error (NRMSE), weighted quantile loss and the Continuous Ranked Probability Score (CRPS)[9] (Matheson & Winkler, 1976) on each time series dimension, as well as on the sum of all time series dimensions (the latter denoted by $Metrics_{SUM}$). The results are summarized in Table 1.

Table 1: TimeGrad Metrics Evaluation

Dimension	CRPS	ND	NRMSE
Single (FTSE MIB)	0.7770502908	1.0183983714	1.3164482648
Global ($\text{Metrics}_{\text{sum}}$)	0.7787133470	1.0512513727	1.2362482353

We are also comparing TimeGrad Evaluation against another deep learning based methods with the same metrics and forecasting horizon. The model, namely GP-Copula (Salinas et al.,2019a) is another multivariate deep learning method which unrolls an LSTM on each individual time series and then the joint emission distribution is given by a low-rank plus diagonal covariance Gaussian copula. The results are summarized below in Table 2.

Table 2: GP-Copula Metrics Evaluation

Dimension	CRPS	ND	NRMSE
Single (FTSE MIB)	0.7675156733	1.0001101909	1.3762850638
Global ($\text{Metrics}_{\text{sum}}$)	0.7207674164	0.9437672189	1.1652926851

The tables exhibit how despite in the original paper TimeGrad models sets the state-of-the-art on all but the smallest of the benchmark datasets, it fails to replicate the performance when it comes to a portfolio of stock indices. We will return on that in the subsequent paragraphs.

[9]CRPS measures the compatibility of a cumulative distribution function (CDF) F with an obsservation x as

$$\text{CRPS}(F, x) = \int_{\mathbb{R}} (F(z) - \mathbb{I}\{x \leq z\})^2 \, \mathrm{d}z,$$

where $\mathbb{I}\{x \leq z\}$ is the indicator function which is one if $x \leq z$ and zero otherwise. CRPS is a proper scoring function, hence CRPS attains its minimum when the predictive distribution F and the data distribution are equal.

6.5.2 Backtest results

In this section, we provide the results of the model Value-at-Risk Backtesting. Our first results are provided for each index at 95% VaR in Figure 19.

We have successfully integrated Value-at-risk evaluation by choosing the prediction length that fitted the optimal number of exceptions. The model is characterized by a high level of stochasticity as the same hyperparameters have generated different levels of Value-at-Risk estimation. We are representing the average scenario for those simulations where we trained Timegrad with returns at 10 days (i.e. two weekly returns) as it was the most reliable in terms of exceptions. Unfortunately the daily VaR Backtesting resulted in a poor performance in terms of exceptions albeit it is clearly able to capture the high volatility of the stock returns as it is shown in Figure **??**. The dark grey line represents the real return time series while the red line is the 95% VaR estimate with Timegrad. The model seems sensitive to the volatility of the time series, when the variability increases the underlying distribution learned by the RNN component generates a higher volatility and so the VaR estimates will be lowered. In the rolling windows, at the 95% significance level, we take a look at exceptions for each time series obtained as discussed in section 2.3. We can also estimate the VaR at 99% confidence level, which is a crucial element in risk management, following the Basel Committee regulations. We can see the 99%VaR results of the model in Figure 20.

First, we proceed to compare our model forecast for daily 95%VaR with the baseline parametric model the non-parametric model Historical Simulation and the Exponential weighted moving average (EWMA). We decide to compare the 10-days 95% VaR, instead of 99% confidence level because with a lower level the hypothesis of our backtest are more reliable, as more exceptions make more significant test statistics. We can easily focus our attention to just one index as the empirical results provide consistent evidence for a global analysis. The backtesting results for FTSE MIB are reported below in Table 5[10] and 6.

Table 3: Backtesting Results for 95% VaR on FTSE MIB

	$Timegrad$	$HS250$	$EWMA$
Number of exceptions	92	45	95
$Test$		$p - value5\%$	
BIN	0.091051	0.00015207	0.041399
TL	0.95688	$3.0478e - 05$	0.97954
POF	0.10027	$4.1825e - 05$	0.048499
CCI	0.20262	$1.555e - 06$	0.12577
TBFI	0.0036578	$7.7204e - 09$	0.040914

[10]The three zones for Traffic Light Test are defined based on the cumulative probability of observing up to x failures, hence, what is reported is this cumulative probability instead of the p-value.

Table 4: Summary of the final test results on FTSE MIB

	Timegrad	*HS250*	*EWMA*
Actual number of exceptions	92	45	95
Expected number of exceptions		78	
Test		Results	
BIN	fail to reject	reject	reject
TL	yellow	green	yellow
POF	fail to reject	reject	reject
CCI	fail to reject	reject	fail to reject
TBFI	reject	reject	reject

The multivariate Timegrad model seems to handle the risk quite well given that the actual number of exceptions found by this model is slightly lower than the number that we get from EWMA (92 vs 95). As expected, the historical simulation model performs the worse out of all the models, this is due to their slowness to react to the change in volatility. The Exponential Weighted Moving Average Method (EWMA) proves to have a better understanding of the volatility of the returns, as Figure 21 points out, showing an improvement against the HS method. Table 6 clearly shows the superior validity of Timegrad, albeit it fails to pass the Time Between Failures Test which means that we have some clustering of the exceptions during our backtest.

Now we are going to compare Timegrad against the univariate GARCH(1,1) model and the competitive deep learning model GP-Copula at 1-day VaR Backtesting.

Table 5: Backtesting Results for daily 95% VaR on FTSE MIB

	Timegrad	$GARCH(1,1)$	*GPCopula*
Number of exceptions	150	97	75
Test		$p-value 5\%$	
BIN	0	0.0174	0.84214
TL	1	0.99085	0.45094
POF	$2.1714e-14$	0.02211	0.84159
CCI	0.081622	0.44059	0.48651
TBFI	$3.826e-06$	0.019197	0.15301

Table 6: Summary of the final test results on FTSE MIB

	$Timegrad$	$GARCH(1,1)$	$GPCopula$
Actual number of exceptions	150	97	75
Expected number of exceptions		78	
$Test$		Results	
BIN	reject	reject	fail to reject
TL	red	yellow	green
POF	reject	reject	fail to reject
CCI	fail to reject	fail to reject	fail to reject
TBFI	reject	reject	fail to reject

With respect to the univariate GARCH(1,1) model, it is shown in Figure 22 that despite being two comparable processes, GP Copula outperforms both Timegrad and the classic time series econometrics method in the relevant tests for the reliability of the model in VaR Backtesting. This should not be much a surprise since the univariate GARCH(1,1) does not take into account the dependencies of the indexes and it relies only on the single process. The difference between the two deep learning model however is remarkable. Despite having a narrow margin with respect to Timegrad in terms of CRPS metrics, it is undoubtedly the best overall model to perform this kind of task.

Figure 17: Preprocessed Time Series of Portfolio's Indexes returns

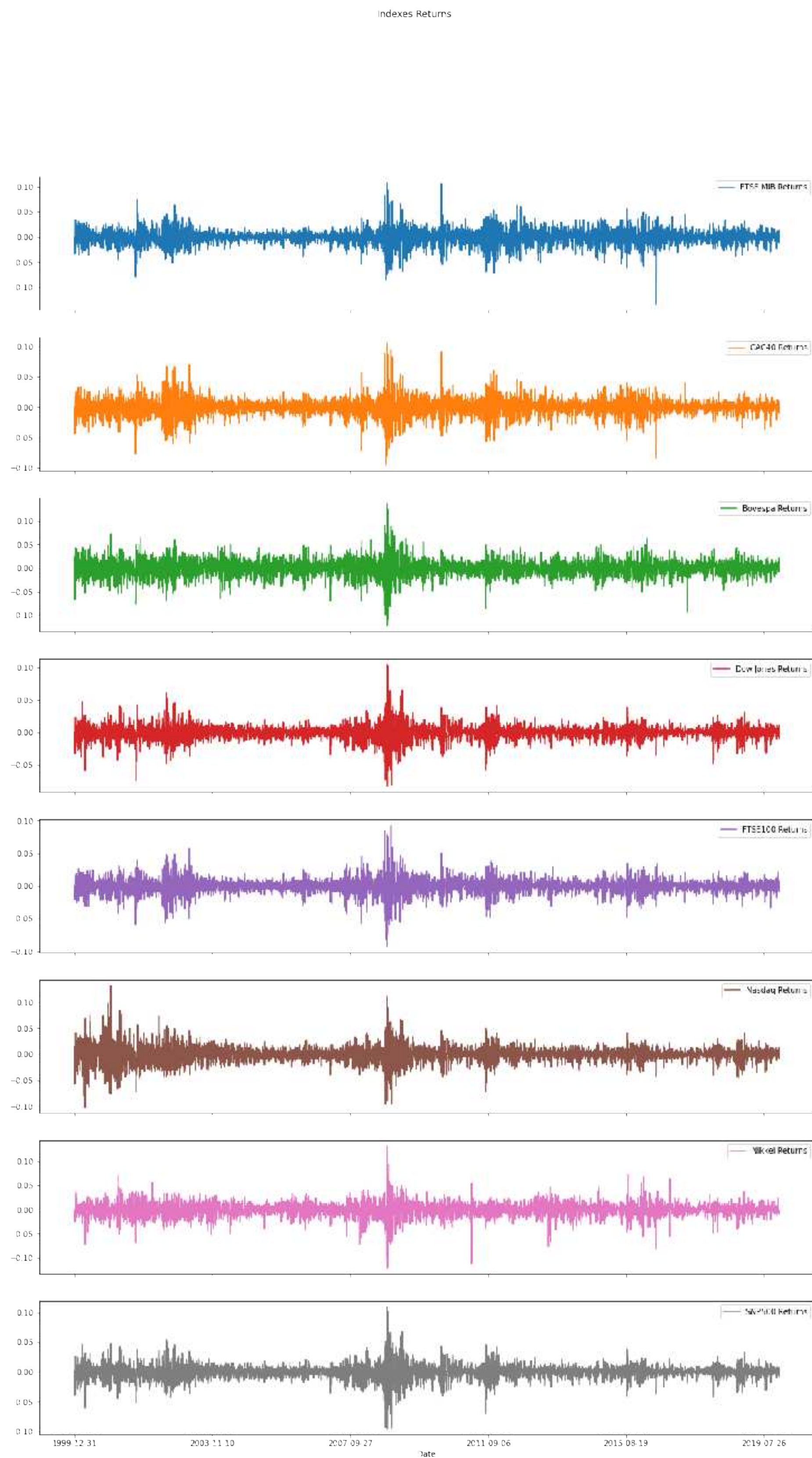


Figure 18: TimeGrad Prediction Intervals over the rolling windows scenario

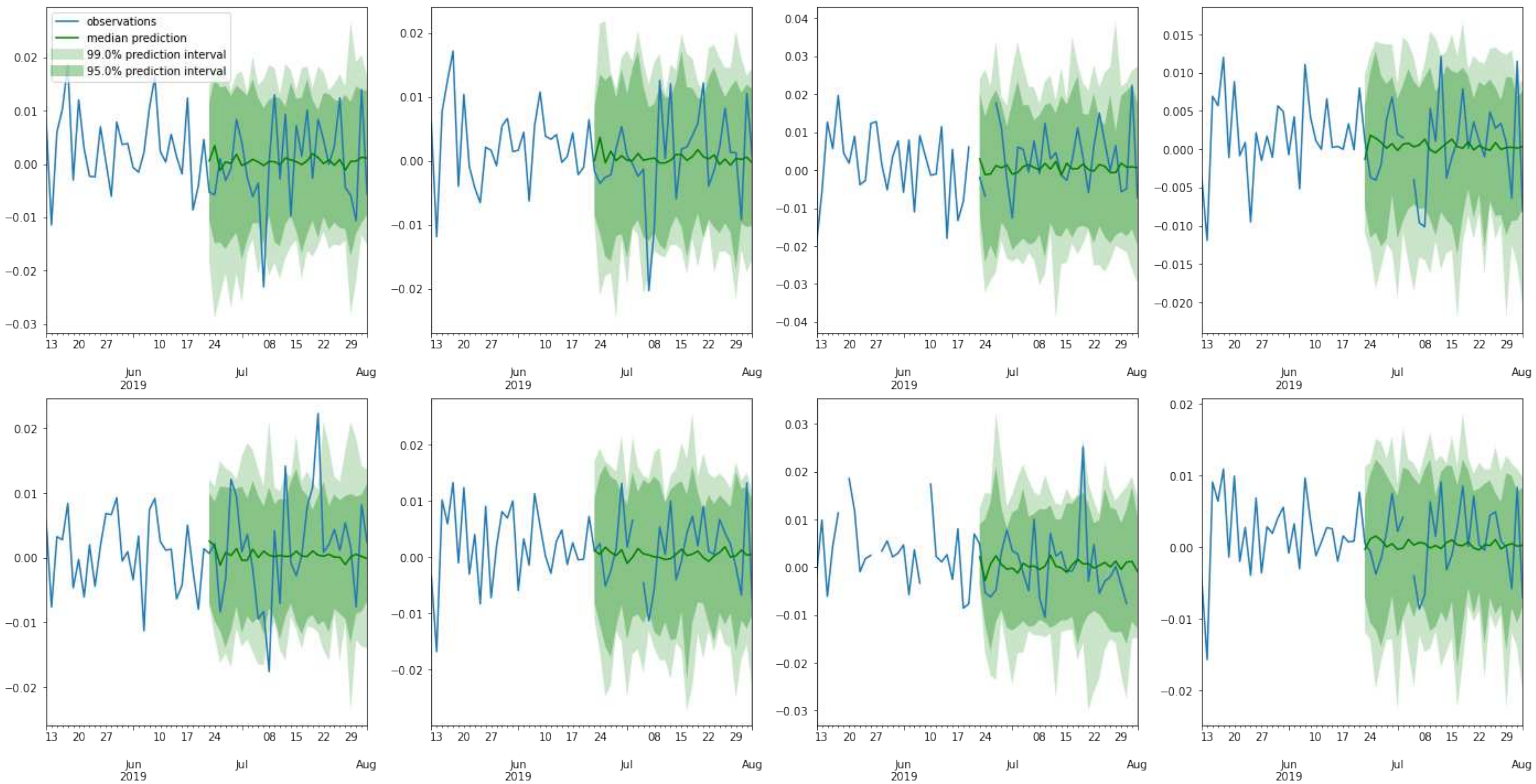


Figure 19: VaR estimates at 95% Confidence level for each index in our portfolio

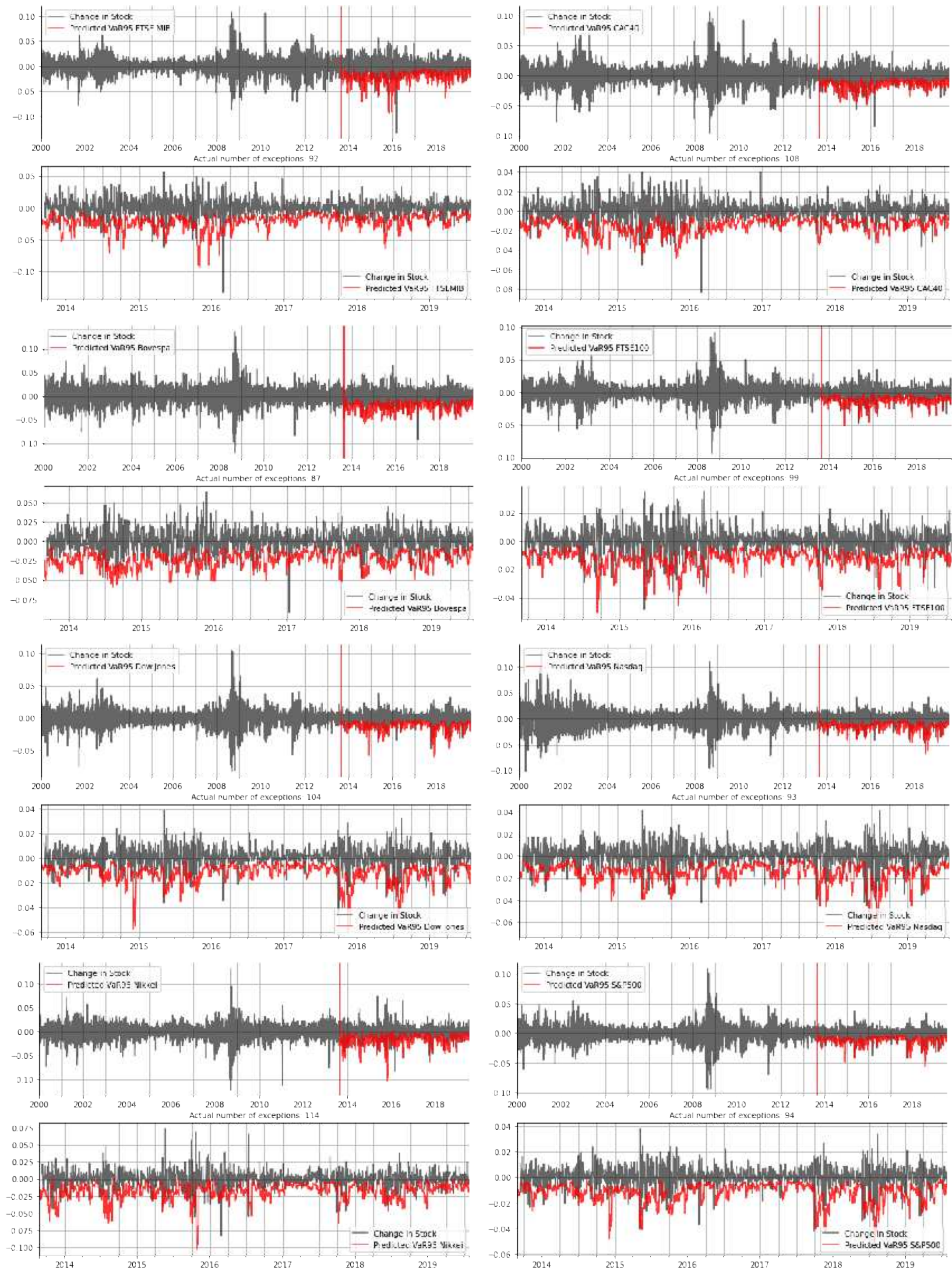


Figure 20: VaR estimates at 99% Confidence level for each index in our portfolio

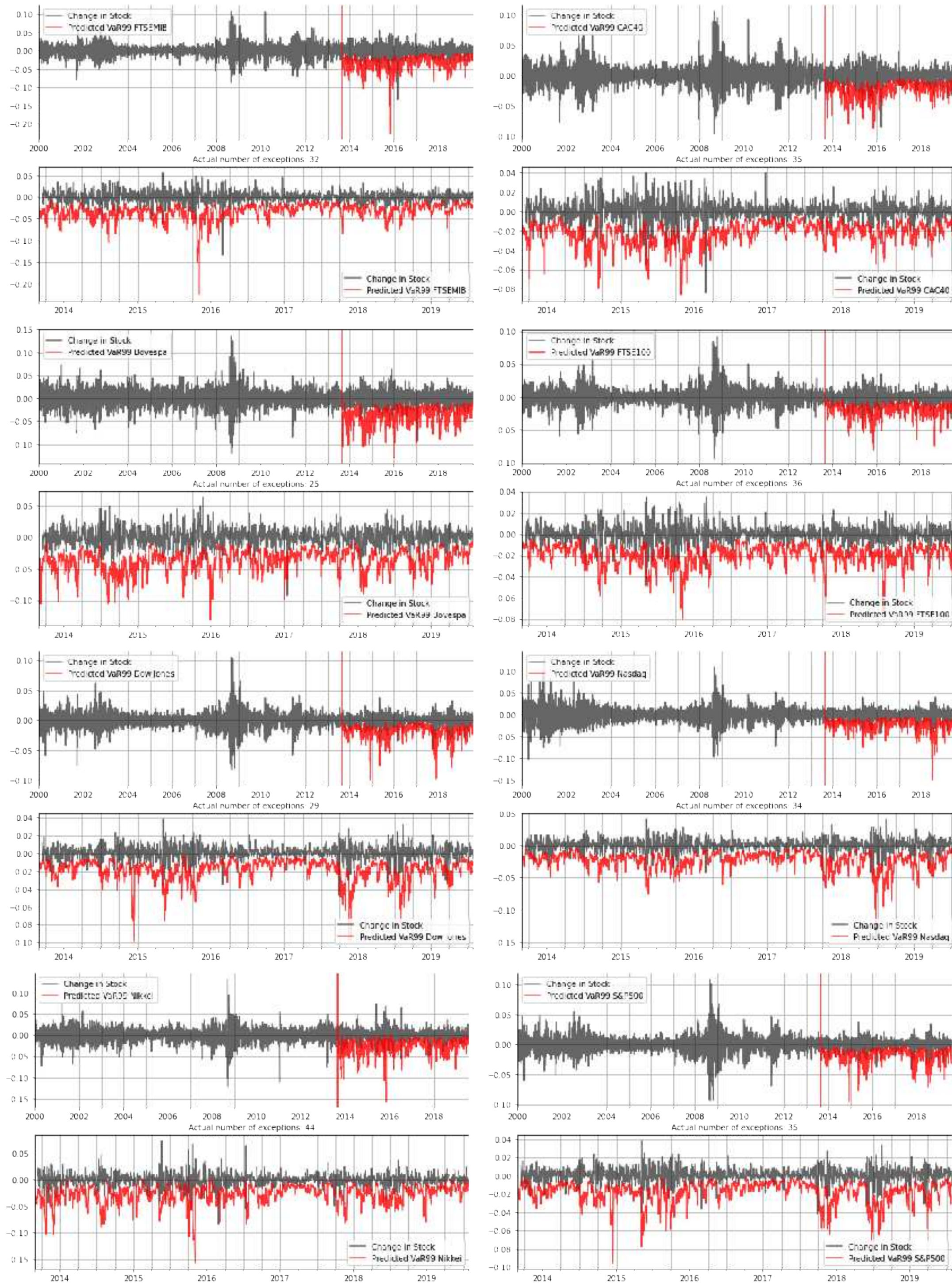


Figure 21: VaR estimates at 95% Confidence level for FTSE MIB

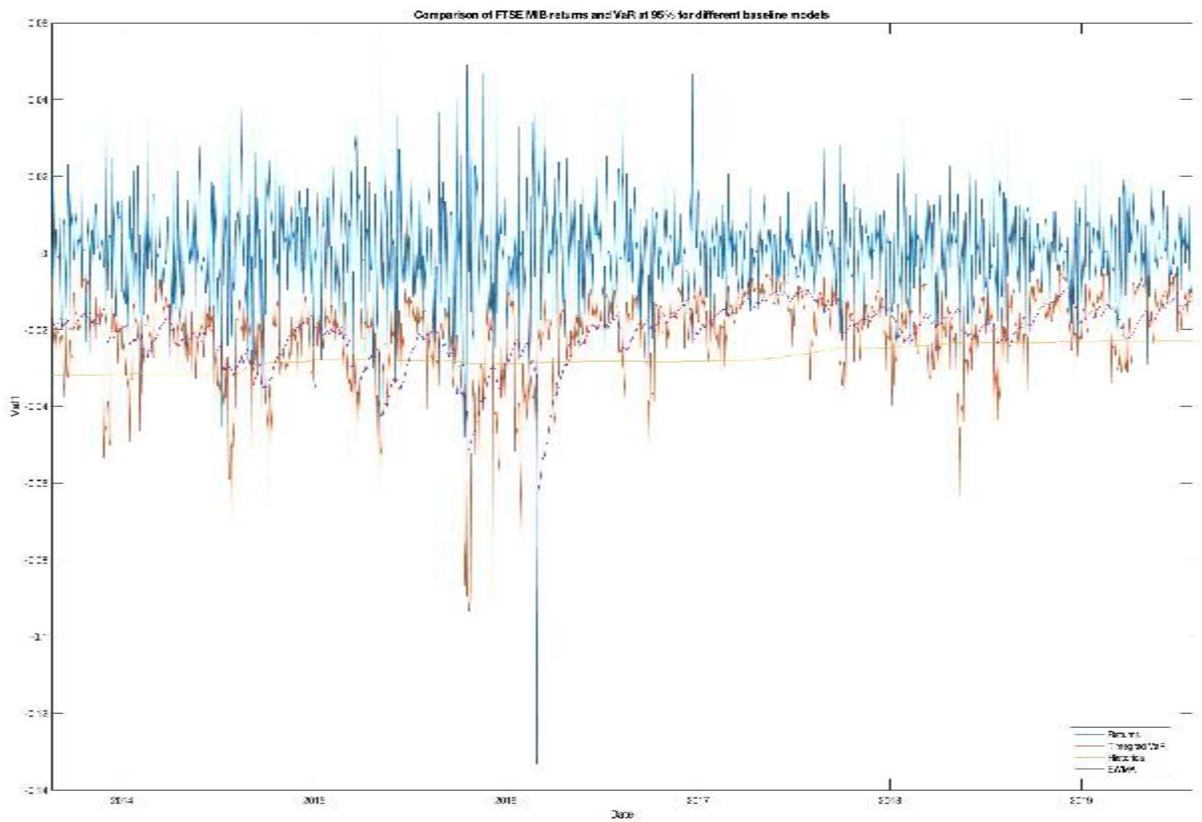
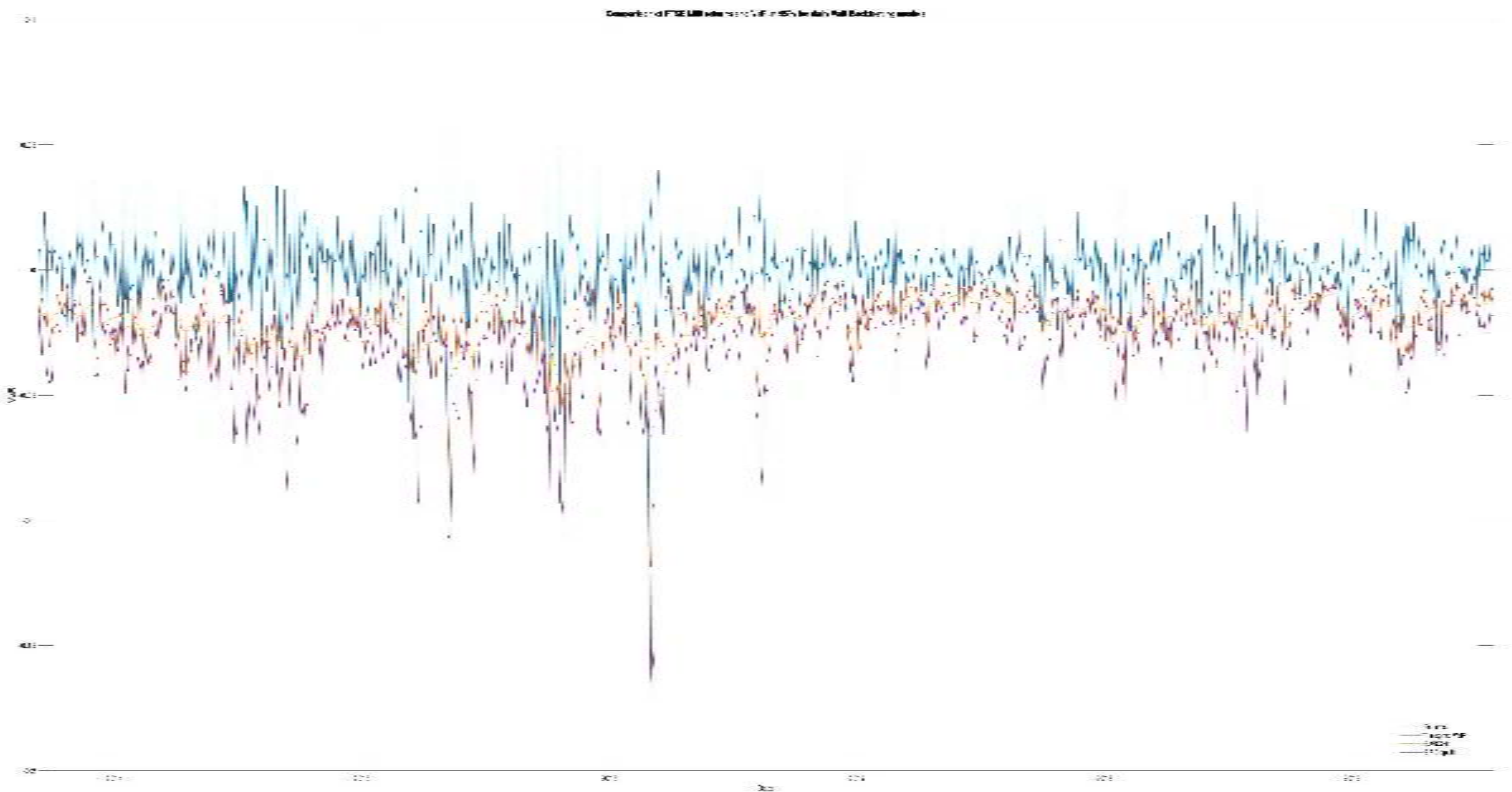


Figure 22: VaR estimates at 95% Confidence level for FTSE MIB



7 Conclusion

In this last section, we draw our conclusions on the work and the goals that have been achieved. In the first subsection, we briefly summarize what were our motivations and intentions. What follows is a set of ideas for further development of this work.

7.1 Summary

The research project main goal was to develop a new approach of generative models for risk management, going beyond previous works at Axyon AI with GANs. This thesis shows that a multivariate Denoising Diffusion Model with an autoregressive component not only is applicable to financial time series data, but it also proves to be a competitive alternative to the classic models. Being a new and constantly evolving environment, the project's first aim was to allocate a consistent amount of time to the literature review. Then we decided to proceed with a model, Timegrad, that exhibited state-of-the-art forecasting predictions against time series econometric models and other Deep learning approaches (Rasul et al., 2021).

We started this report providing the basics on the Value at Risk theory as the most common measure for market risk and its conventional estimation with parametric and non-parametric methods. We then presented the neural network background necessary to build Diffusion probabilistic models integrated with a RNN. In particular, our model is conditioned on past returns (autoregressive component) and thus it can learn and generate the conditional distribution of the time series.

We started developing Timegrad as a global model as its libraries in PyTorchTS and its code were built under a multivariate environment (an attempt to the univariate case was made but it led to poor results). We took advantage of the multivariate dimension by using highly correlated assets to macroeconomic factors such as the most valuable stock indices. Except prediction length and number of epochs, the hyper-parameters of GPVAR(GP Copula), Timegrad were the default hyperparameters, described in their documentation[11]. We used 20 epochs for Timegrad and 50 epochs for GPVAR respectively. As pointed out in the previous section, the major problem encountered concerns the presence of exception clusters in the backtest. It is clear then that the model is still far from being perfect by incorrectly generating the underlying distribution of the data. Nevertheless, it proved to be a consistent competitor to classical methods (at least for the 10-days estimator). Considering also the superb performance of alternative methods such as GP Copula we may infer that further research on this field can improve the performances of Denoising models for multivariate probabilistic time series forecasting and that sooner or later deep generative models may overcome and substitute traditional ones for risk management.

[11] https://github.com/awslabs/gluon-ts

7.2 Further developments

Our work shows that multivariate autoregressive denoising models are successfully able of dealing with the probability distribution of financial time series, albeit with limitations, and we think that the proposed model could have many possible opportunities for further research.

1. We may improve the forecast by adding manual features to the RNN architecture given that this model takes covariates automatically through lag orders.

2. As previous works with Axyon AI pointed out (Davoli, 2021), generative models such as GAN and Timegrad are challenging to train and time-consuming. Models such as Low-Rank Gaussian Copula Processes (Salinas, 2019) provide more efficient results with little cost in terms of performance. There is a trade-off that needs further development regarding the criterion for model selection but the models from Salinas seems to perform better in this field.

3. As a first attempt, the experiment is built with a small number of assets in order to replicate the characteristics of one of the datasets in the original paper. Once verified that an implementation of these kinds of models is indeed possible we can think to expand the dimension of the portfolio to a more complex version. Not only we could evaluate the VaR but also the sign of the weight on each instrument (long/short position) should be considered. This would allow the comparison with other classical multivariate methods such as DCC-GARCH and BEKK presented in Section 2.2.4 and 2.2.5

Those are the main proposals that have emerged during several meetings held within the Axyon ML team, which I personally thank for their punctual and steady support.

References

[1] Bengio, Y., Simard, P., Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks, 5(2), 157-166.

[2] Benidis, K., Rangapuram, S. S., Flunkert, V., Wang, B., Maddix, D., Turkmen, C., ... Januschowski, T. (2020). Neural forecasting: Introduction and literature overview. arXiv preprint arXiv:2004.10240.

[3] Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. Journal of econometrics, 31(3), 307-327.

[4] Box, George EP, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung.(2015) Time Series Analysis: Forecasting and Control. John Wiley Sons.

[5] Chung, J., Gulcehre, C., Cho, K., Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

[6] Davoli, Giovanni (2021). VaR Estimation with conditional GANs and GCNs.

[7] Engle R. (2002). Dynamic Conditional Correlation: A Simple Class of Multivariate GARCH Models. Journal of Business and Economic Statistics.

[8] Goodfellow, I., Bengio, Y. and Courville, A. (2016). Deep Learning. MIT Press.

[9] Gouttes, A., Rasul, K., Koren, M., Stephan, J., Naghibi, T. (2021). Probabilistic Time Series Forecasting with Implicit Quantile Networks. arXiv preprint arXiv:2107.03743.

[10] Graves, A. (2013). Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850.

[11] Haas, M. (2001). New Methods in Backtesting, Financial Engineering. Bonn: Research Center Caesar.

[12] Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. Neural computation, 14(8), 1771-1800.

[13] Ho, J., Jain, A., Abbeel, P. (2020). Denoising diffusion probabilistic models. arXiv preprint arXiv:2006.11239.

[14] Hochreiter, S., Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[15] Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies

[16] Hochreiter S. (1991). Untersuchungen Zu Dynamischen Neuronalen Netzen. In: Diploma, Technische Universitat Munchen 91.1

[17] Kingma, D. P., Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[18] Kong, Z., Ping, W., Huang, J., Zhao, K., Catanzaro, B. (2020). Diffwave: A versatile diffusion model for audio synthesis. arXiv preprint arXiv:2009.09761.

[19] Kupiec, P. (1995). Techniques for verifying the accuracy of risk measurement models. The J. of Derivatives, 3(2).

[20] Le Guennec A., Malinowski S., and Tavenard R. (2016). Data Augmentation for Time Series. Classification using Convolutional Neural Networks.

[21] LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., Huang, F. (2006). A tutorial on energy-based learning. Predicting structured data, 1(0).

[22] Linsmeier, T. J., Pearson, N. D. (1996). Risk measurement: An introduction to value at risk (No. 1629-2016-134959).

[23] Matheson, J. E., Winkler, R. L. (1976). Scoring rules for continuous probability distributions. Management science, 22(10), 1087-1096.

[24] McAndrews, K. (2015). Evaluating the Accuracy of Value at Risk Approaches.

[25] Bollerslev T. (1986). Generalized autoregressive conditional heteroskedasticity. Journal of Econometrics.

[26] Teräsvirta, T. (2009). An introduction to univariate GARCH Models. Handbook of Financial Time Series, 17–42.

[27] McKinsey et al. (2012). Managing market risk: Today and tomorrow. Report of 2012

[28] Montantes. (2020). What You Need to Know About Deep Reinforcement Learning.

[29] Oord, A. V. D., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499.

[30] Rasul, K., Seward, C., Schuster, I., Vollgraf, R. (2021). Autoregressive Denoising Diffusion Models for Multivariate Probabilistic Time Series Forecasting. arXiv preprint arXiv:2101.12072

[31] Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533-536.

[32] Salinas, D., Bohlke-Schneider, M., Callot, L., Medico, R., Gasthaus, J. (2019). High-dimensional multivariate forecasting with low-rank gaussian copula processes. arXiv preprint arXiv:1910.03002.

[33] Salinas, D., Flunkert, V., Gasthaus, J., and Januschowski, T. (2019b). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting. ISSN 0169-2070

[34] Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., Ganguli, S. (2015, June). Deep unsupervised learning using nonequilibrium thermodynamics. In International Conference on Machine Learning (pp. 2256-2265). PMLR.

[35] Song, Y., Kingma, D. P. (2021). How to train your energy-based models. arXiv preprint arXiv:2101.03288.

[36] Talagala, T.S., Hyndman, R.J., Athanasopoulos, G. (2018). Meta-learning how to forecast time series. Monash Econometrics and Business Statistics Working Papers, 6, 18.

[37] Taylor, S., (1986). Modelling Financial Time Series. Wiley, New York.

[38] Teräsvirta, T. (2009). An introduction to univariate GARCH Models. Handbook of Financial Time Series, 17–42

[39] Tsay, R. S. (2014). An introduction to analysis of financial data with R. John Wiley Sons.

[40] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

[41] Welling, K. N. (2011). Modeling the water consumption of Singapore using system dynamics (Doctoral dissertation, Massachusetts Institute of Technology).

[42] Xu, Q., Liu, X., Jiang, C., Yu, K. (2016). Quantile autoregression neural network model with applications to evaluating value at risk. Applied Soft Computing, 49, 1-12.

[43] Zhao, S., Liu, Z., Lin, J., Zhu, J. Y., Han, S. (2020). Differentiable augmentation for data-efficient gan training. arXiv preprint arXiv:2006.10738.